

# Криптография с секретным ключом

Фрэнк Рубин



MANNING



Фрэнк Рубин

# Криптография с секретным ключом

# *Secret Key Cryptography*

CIPHERS, FROM SIMPLE TO UNBREAKABLE

**FRANK RUBIN**

**Foreword by RANDALL K. NICHOLS**



**MANNING**  
**Shelter Island**

# *Криптография с секретным ключом*

ШИФРЫ – ОТ ПРОСТЫХ ДО НЕВСКРЫВАЕМЫХ

**ФРЭНК РУБИН**

С предисловием Рэндалла К. Николса



Москва, 2023



УДК 004.382  
ББК 32.973-018  
P82

**Рубин Ф.**

P82 Криптография с секретным ключом / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2022. – 386 с.: ил.

**ISBN 978-5-97060-748-0**

В книге объясняется, как создавать шифры с секретным ключом – от простых, для которых хватает карандаша и бумаги, до очень сложных, применяемых в современной компьютерной криптографии. Вы научитесь конструировать 30 невскрываемых шифров, измерять стойкость шифров и гарантированно обеспечивать их безопасность, противостоять гипотетическим ультракомпьютерам будущего. А для развлечения предлагается вскрыть несколько несложных мини-шифров.

Издание предназначено для профессиональных инженеров, специалистов по информатике и криптографов-любителей.

УДК 004.382  
ББК 32.973-018

Copyright © DMK Press 2022. Authorized translation of the English edition © 2022 Manning Publications. This translation is published and sold by permission of Manning Publications, the owner of all rights to publish and sell the same.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

# Оглавление

---

1 ■ Введение .....	24
2 ■ Что такое криптография?.....	27
3 ■ Предварительные сведения .....	41
4 ■ Инструментарий криптографа .....	48
5 ■ Подстановочные шифры.....	61
6 ■ Контрмеры.....	94
7 ■ Перестановка .....	109
8 ■ Цилиндрический шифр Джефферсона.....	128
9 ■ Фракционирование .....	135
10 ■ Фракционирование переменной длины.....	163
11 ■ Блочные шифры.....	188
12 ■ Принципы безопасного шифрования.....	214
13 ■ Поточковые шифры.....	246
14 ■ Одноразовый блокнот .....	276
15 ■ Матричные методы .....	292
16 ■ Трехпроходный протокол .....	326
17 ■ Коды .....	342
18 ■ Квантовые компьютеры.....	

# Содержание

---

	Оглавление .....	5
	Вступительное слово.....	13
	Предисловие .....	16
	Благодарности .....	18
	Об этой книге.....	19
	Об авторе .....	22
	Об иллюстрации на обложке.....	23
<b>1</b>	<b>Введение .....</b>	<b>24</b>
<b>2</b>	<b>Что такое криптография? .....</b>	<b>27</b>
	2.1 Невскрываемые шифры .....	28
	2.2 Виды криптографии .....	30
	2.3 Симметричная и асимметричная криптография .....	32
	2.4 Блочные и потоковые шифры.....	33
	2.5 Механические и цифровые шифры.....	33
	2.6 Зачем выбирать шифр с секретным ключом? .....	37
	2.7 Зачем создавать собственный шифр?.....	38
<b>3</b>	<b>Предварительные сведения .....</b>	<b>41</b>
	3.1 Биты и байты.....	41
	3.2 Функции и операторы.....	42
	3.3 Булевы операторы .....	43
	3.4 Системы счисления .....	44
	3.5 Простые числа.....	46
	3.6 Модульная арифметика.....	46
<b>4</b>	<b>Инструментарий криптографа .....</b>	<b>48</b>
	4.1 Система оценивания .....	49

4.2	Подстановка .....	50
4.2.1	Коды Хаффмана .....	51
4.3	Перестановка .....	52
4.4	Фракционирование .....	53
4.5	Генераторы случайных чисел .....	54
4.5.1	Цепной генератор цифр .....	56
4.6	Полезные комбинации, бесполезные комбинации .....	58
4.6.1	Шифр Базери типа 4 .....	59

<b>5</b>	<b>Подстановочные шифры .....</b>	<b>61</b>
5.1	Простая подстановка .....	62
5.2	Перемешивание алфавита .....	67
5.3	Номенклаторы .....	70
5.4	Многоалфавитная подстановка .....	70
5.5	Шифр Беласо .....	71
5.6	Метод Касиски .....	72
5.7	Индекс совпадения .....	76
5.8	И снова об индексе совпадения .....	77
5.9	Вскрытие многоалфавитного шифра .....	78
5.9.1	Вскрытие шифра Беласо .....	78
5.9.2	Вскрытие шифра Виженера .....	81
5.9.3	Вскрытие общего многоалфавитного шифра .....	83
5.10	Автоключ .....	85
5.11	Бегущий ключ .....	86
*5.12	Моделирование роторных машин .....	88
5.12.1	Однороторная машина .....	90
5.12.2	Трехроторная машина .....	91
5.12.3	Восьмироторная машина .....	92

<b>6</b>	<b>Контрмеры .....</b>	<b>94</b>
6.1	Двойное шифрование .....	95
6.2	Null-символы .....	96
6.3	Прерванный ключ .....	96
6.4	Омофоническая подстановка .....	99
6.4.1	Шифр 5858 .....	100
6.5	Подстановка биграмм и триграмм .....	100
*6.6	Соккрытие сообщений в изображениях .....	101
6.7	Добавление null-битов .....	103
6.8	Объединение нескольких сообщений .....	105
6.9	Внедрение сообщения в файл .....	107

<b>7</b>	<b>Перестановка .....</b>	<b>109</b>
7.1	Маршрутная перестановка .....	109
7.2	Столбцовая перестановка .....	111
7.2.1	Cysquare .....	115
7.2.2	Перестановка слов .....	116

7.3	Двойная столбцовая перестановка .....	117
7.4	Столбцовая перестановка с циклическим сдвигом .....	118
7.5	Перестановка со случайными числами .....	120
7.6	Селекторная перестановка .....	121
7.7	Перестановка с ключом .....	122
7.8	Деление перестановки пополам .....	125
7.9	Множественные анаграммы .....	126

<b>8</b>	<b>Цилиндрический шифр Джефферсона .....</b>	<b>128</b>
8.1	Вскрытие при наличии известных слов .....	131
8.2	Вскрытие при наличии только шифртекста .....	132

<b>9</b>	<b>Фракционирование .....</b>	<b>135</b>
9.1	Квадрат Полибия .....	136
9.2	Шифр Плейфера .....	137
9.2.1	Вскрытие шифра Плейфера .....	139
9.2.2	Укрепление шифра Плейфера .....	140
9.3	Шифр Two Square .....	142
9.4	Шифр Three Square .....	143
9.5	Шифр Four Square .....	146
9.6	Шифр Bifid .....	148
9.6.1	Bifid с сопряженной матрицей .....	150
9.7	Диагональный Bifid .....	151
9.8	Квадраты 6×6 .....	152
9.9	Шифр Trifid .....	152
9.10	Шифр Three Cube .....	154
9.11	Прямоугольные сетки .....	156
9.12	Шестнадцатеричное фракционирование .....	157
9.13	Битовое фракционирование .....	158
9.13.1	Шифр Cyclic 8×N .....	159
9.14	Другие виды фракционирования .....	160
9.15	Повышение стойкости блоков .....	161

<b>10</b>	<b>Фракционирование переменной длины .....</b>	<b>163</b>
10.1	Шифр Morse3 .....	164
10.2	Моном-биномные шифры .....	165
10.3	Периодические длины .....	167
10.4	Подстановка Хаффмана .....	168
10.5	Тэг-системы Поста .....	171
10.5.1	Таги одинаковой длины .....	172
10.5.2	Таги разной длины .....	174
10.5.3	Несколько алфавитов .....	176
10.5.4	Короткие и длинные перемещения .....	177
10.6	Фракционирование в системах счисления по другим основаниям .....	177
10.7	Сжатие текста .....	178

10.7.1	Метод Лемпеля–Зива .....	178
10.7.2	Арифметическое кодирование .....	181
10.7.3	Адаптивное арифметическое кодирование.....	184

<b>11</b>	<b>Блочные шифры.....</b>	<b>188</b>
11.1	Подстановочно-перестановочная сеть .....	189
11.2	Стандарт шифрования данных (DES) .....	191
11.2.1	Double DES.....	192
11.2.2	Triple DES.....	193
*11.2.3	Быстрая перестановка битов .....	194
11.2.4	Неполные блоки.....	195
11.3	Умножение матриц.....	196
11.4	Умножение матриц.....	197
11.5	Улучшенный стандарт шифрования (AES) .....	198
11.6	Фиксированная подстановка и подстановка с ключом .....	200
11.7	Инволютивные шифры.....	201
11.7.1	Инволютивная подстановка.....	202
11.7.2	Инволютивная многоалфавитная подстановка .....	202
11.7.3	Инволютивная перестановка .....	202
*11.7.4	Инволютивный блочный шифр .....	203
11.7.5	Пример – шифр Poly Triple Flip.....	204
11.8	Подстановки переменной длины.....	204
11.9	Пульсирующие шифры .....	205
11.10	Сцепление блоков .....	208
11.10.1	Многоалфавитное сцепление .....	209
11.10.2	Зашифрованное сцепление.....	210
11.10.3	Сцепление с запаздыванием .....	210
11.10.4	Внутренние отводы .....	210
11.10.5	Сцепление ключей.....	211
11.10.6	Сводка режимов сцепления.....	211
11.10.7	Сцепление с неполными блоками .....	211
11.10.8	Сцепление блоков переменной длины .....	211
11.11	Укрепление блочного шифра .....	212

<b>12</b>	<b>Принципы безопасного шифрования.....</b>	<b>214</b>
12.1	Большие блоки .....	214
12.2	Длинные ключи .....	215
12.2.1	Избыточные ключи .....	216
12.3	Конфузия .....	217
12.3.1	Коэффициент корреляции .....	219
12.3.2	Линейность по основанию 26 .....	223
12.3.3	Линейность по основанию 256 .....	226
12.3.4	Включение закладки .....	227
12.3.5	Конденсированная линейность .....	231
12.3.6	Гибридная нелинейность .....	232
12.3.7	Конструирование S-блока .....	232
12.3.8	S-блок с ключом .....	236

12.4	Диффузия .....	236
12.5	Насыщение .....	240
	Резюме .....	245

<b>13</b>	<b>Потоковые шифры</b> .....	246
13.1	Комбинирующие функции .....	247
13.2	Случайные числа .....	248
13.3	Мультипликативный конгруэнтный генератор .....	249
13.4	Линейный конгруэнтный генератор .....	253
13.5	Цепной XOR-генератор .....	254
13.6	Цепной аддитивный генератор .....	256
13.7	Сдвиговой XOR-генератор .....	256
13.8	FRand .....	257
13.9	Вихрь Мерсенна .....	259
13.10	Регистры сдвига с линейной обратной связью .....	259
13.11	Оценивание периода .....	261
13.12	Укрепление генератора .....	263
13.13	Комбинирование генераторов .....	264
13.14	Истинно случайные числа .....	268
	13.14.1 Линейное суммирование с запаздыванием .....	268
	13.14.2 Наложение изображений .....	269
13.15	Обновление случайных байтов .....	270
13.16	Синхронизированные гаммы .....	272
13.17	Функции хеширования .....	273

<b>14</b>	<b>Одноразовый блокнот</b> .....	276
14.1	Шифр Вернама .....	278
14.2	Запас ключей .....	280
	14.2.1 Возвращение ключей в оборот .....	281
	14.2.2 Комбинированный ключ .....	281
	14.2.3 Ключ выбора .....	281
14.3	Индикаторы .....	282
14.4	Алгоритм распределения ключей Диффи–Хеллмана .....	283
	*14.4.1 Построение больших простых чисел, старый подход .....	285
	14.4.2 Построение больших простых чисел, новый подход .....	286

<b>15</b>	<b>Матричные методы</b> .....	292
15.1	Обращение матрицы .....	293
15.2	Матрица перестановки .....	296
15.3	Шифр Хилла .....	296
15.4	Шифр Хилла, компьютерные версии .....	299
15.5	Умножение больших целых чисел .....	303
	15.5.1 Умножение и деление сравнений .....	304
*15.6	Решение линейных сравнений .....	305
	15.6.1 Приведение сравнения .....	305
	15.6.2 Правило половины .....	306

15.6.3	Лесенка .....	308
15.6.4	Цепные дроби .....	309
15.7	Шифры на основе больших целых чисел.....	310
15.8	Умножение на малое число .....	311
15.9	Умножение по модулю $P$ .....	313
15.10	Изменение основания .....	315
*15.11	Кольца .....	317
15.12	Матрицы над кольцом .....	318
15.13	Построение кольца .....	319
15.13.1	Гауссовы целые числа.....	321
15.13.2	Кватернионы .....	322
15.14	Нахождение обратимых матриц .....	323
<b>16</b>	<b>Трехпроходный протокол .....</b>	<b>326</b>
16.1	Метод Шамира .....	328
16.2	Метод Мэсси–Омуры .....	329
16.3	Дискретный логарифм.....	329
16.3.1	Логарифмы .....	330
16.3.2	Степени простых чисел.....	330
16.3.3	Коллизия .....	331
16.3.4	Факторизация .....	331
16.3.5	Оценки .....	333
16.4	Матричный трехпроходный протокол.....	333
16.4.1	Коммутативное семейство матриц .....	334
16.4.2	Мультипликативный порядок .....	334
16.4.3	Максимальный порядок .....	335
16.4.4	Атаки Эмили .....	336
16.4.5	Некоммутативное кольцо.....	337
16.4.6	Решение билинейных уравнений .....	337
16.4.7	Слабые элементы .....	339
16.4.8	Как сделать побыстрее .....	339
16.5	Двусторонний трехпроходный протокол.....	340
<b>17</b>	<b>Коды.....</b>	<b>342</b>
17.1	Джокер .....	343
<b>18</b>	<b>Квантовые компьютеры .....</b>	<b>346</b>
18.1	Суперпозиция .....	347
18.2	Квантовая запутанность.....	348
18.3	Исправление ошибок .....	349
18.4	Измерение .....	350
18.5	Квантовый трехэтапный протокол .....	351
18.6	Квантовое распределение ключей.....	352
18.7	Алгоритм Гровера .....	352
18.8	Уравнения .....	353
18.8.1	Перестановки .....	353



18.8.2	Подстановки .....	354
18.8.3	Карты Карно .....	354
18.8.4	Промежуточные переменные .....	355
18.8.5	Известный открытый текст .....	355
18.9	Минимизация .....	356
18.9.1	Восхождение на вершину .....	356
18.9.2	Тысяча вершин .....	357
18.9.3	Имитация отжига .....	358
18.10	Квантовая имитация отжига .....	360
18.11	Квантовая факторизация .....	360
18.12	Ультракомпьютеры .....	360
18.12.1	Подстановка .....	361
18.12.2	Случайные числа .....	362
18.12.3	Ультраподстановочный шифр US-A .....	363
18.12.4	Ультрапоточковый шифр US-B .....	364
	Развлечения .....	366
	Задачи .....	369
	Эпилог .....	371
	Предметный указатель .....	374

# Вступительное слово

---

От тайных дешифровальных колец до правительственных директив, задачи сокрытия и обнаружения информации в составе другой информации давно будоражили человеческий ум. Криптология – завораживающий предмет, с которым на практике сталкивался едва ли не всякий школьник. И вместе с тем имеются веские причины, по которым эта дисциплина на протяжении веков была окутана глубочайшей тайной, поскольку государства использовали ее для защиты своего самого секретного оружия. В военных и дипломатических делах к криптографии всегда относились в высшей степени серьезно. Не будет преувеличением сказать, что успехи и провалы криптографии влияли на исход войн и ход истории, и точно так же они определяют нашу современную историю.

Возьмем сражение при Энтитеме в ходе Гражданской войны в США, произошедшее в сентябре 1862 года близ Шарпсбурга, штат Мэриленд, в котором Федеральная армия под командованием Джорджа Макклеллана противостояла армии Конфедерации под командованием Роберта Ли. За несколько дней до него два солдата федералов нашли недалеко от лагеря листок бумаги, оказавшийся копией приказа Ли, в котором были подробно изложены планы вторжения в Мэриленд. Приказ не был зашифрован. Располагая этой информацией, Макклеллан точно знал местоположение рассеянных отрядов и смог уничтожить армию Ли, не дав им соединиться.

Успехи и провалы криптографии оказывали влияние и на более близкие к нам события. Сокрушительное поражение русской армии при Танненберге в августе 1914 года стало прямым следствием перехвата сообщений немцами. Удивительно, но сообщения русских передавались открытым текстом, потому что у полевых командиров не было ни шифров, ни ключей. Поэтому русские не могли безопасно координировать действия соседних подразделений.

50 лет холодной войны, последовавшей за Второй мировой войной, тоже стали результатом провала криптографии, на этот раз со стороны японцев в битве за Мидуэй в 1942 году. Американские криптоаналитики взломали японские коды и могли читать многие донесения Объединенного флота. Подобные истории – вотчина классической криптографии. Книга «Криптография с секретным ключом» как раз на этом поле и играет.

Никто не сможет лучше доктора Фрэнка Рубина провести интересующегося читателя по всем закоулкам классической криптологии на любительском уровне, от математических истоков до социальных последствий. Доктор Рубин получил образование в области математики и информатики. Тридцать лет он проработал в компании IBM, в отделе автоматизации проектирования, и свыше 50 лет занимался криптографией. Доктор Рубин был редактором журнала «Cryptologia» и других изданий. Он автор десятков статей по математике и компьютерным алгоритмам, а также тысяч математических головоломок.

«Криптография с секретным ключом» – не просто новая версия классической книги Helen F. Gaines «Elementary Cryptanalysis». Здесь предмет рассматривается с древних времен до эры квантовых компьютеров. И, что немаловажно, описывается уникальный метод измерения стойкости шифра<sup>1,2</sup>.

Книга выходит в стратегически важный момент. Это своевременный и существенный вклад в понимание критической технологии. Не важно, испытывает ли читатель бескорыстный интерес к криптологии как таковой или занимается практической защитой информации, материал, изложенный на этих страницах, благодаря глубине и широте охвата станет желанным источником полезной информации, а сама книга – ценным пополнением библиотеки.

– Рэндалл К. Николс, DTM

*Рэндалл К. Николс – бывший президент  
Американской ассоциации криптограмм,  
отвечавший, в частности, за обзоры книг;  
директор программы сертификации беспилотных летательных систем  
на предмет кибербезопасности в Канзасском университете в Салине;  
заслуженный профессор отделения послевузовского образования  
в области кибербезопасности и компьютерно-технической экспертизы  
в колледже Ютики.*

---

<sup>1</sup> И в книге R. K. Nichols «ICSA Guide to Cryptography», и в классическом труде Брюса Шнейера «Прикладная криптография» приведены методы оценки стойкости шифров и случайности. Первая посвящена в основном классической криптографии, вторая – в большей степени современным шифрам (Nichols, 1999; Schneier, 1995).

<sup>2</sup> В «Криптографии с секретным ключом» лучше отобран и лучше изложен материал, чем в двух моих первых книгах по классической криптографии: «Classical Cryptography Course», т. I и II (LANAKI, 1998; 1999).

## ЛИТЕРАТУРА

- Gaines, H. F. (1956). *Cryptanalysis: A Study of Ciphers and their Solution*. NYC: Dover.
- LANAKI. (1998). *Classical Cryptography Course Vol. I*. Laguna Hills, CA: Aegean Park Press.
- LANAKI. (1999). *Classical Cryptography Course Vol. II*. Laguna Hills, CA: Aegean Park Press.
- Nichols, R. K. (1999). *ICSA Guide to Cryptography*. New York City: McGraw Hill.
- Rubin, F. (2022). *Secret Key Cryptography*. Shelter Island, New York: Manning Books.
- Schneier, B. (1995). *Applied Cryptography: Protocols, Algorithms and Source Code in C*. New York: John Wiley & Sons.

# Предисловие

---

К идее написать эту книгу меня привели разные дорожки. Начну с моего школьного друга Чарли Роуза. Чарли работал в школьном книжном магазине. В один прекрасный день, заказывая книги для магазина, он обратил внимание на книгу Хелен Ф. Гейнс «Криптоанализ». Чарли захотел приобрести ее, да еще и с отраслевой скидкой. Но вот незадача – магазин должен был заказать как минимум три экземпляра.

Чарли нужно было найти еще двух желающих купить книгу. Он пообещал, что мы все вместе прочтем ее, а затем будем придумывать криптограммы, которые другие должны будут решать. Я книгу купил, прочел и начал составлять криптограммы, а Чарли утратил интерес.

На обратной стороне обложки «Криптоанализа» был напечатан давно устаревший адрес Американской ассоциации криптограмм ([www.cryptogram.org](http://www.cryptogram.org)), но я все-таки нашел ее и вступил в ее члены. И начал решать разные типы криптограмм, которые публиковались в бюллетене для любителей «The Cryptogram». А спустя несколько лет стал заместителем редактора. И вот уже более 40 лет остаюсь членом Ассоциации.

В 1977 году был основан более профессиональный журнал по криптографии, «Cryptologia». Его можно найти в интернете по адресу <https://www.tandfonline.com/toc/ucry20/current>. Сначала я читал статьи, потом начал писать и в конце концов стал редактором. Как-то так получилось, что ко мне стекались все статьи разных фриков, и приходилось продираться сквозь хитросплетения нелогичной логики – вдруг где-то в глубине притаилась хорошая идея. И один раз такое случилось. Я превратил эту идею в статью для «The Cryptogram». Автор был так благодарен, что посадил в мою честь дерево в Израиле.

Этот опыт научил меня отделять статьи, которые просто плохо написаны или переоценивают стойкость шифра, от трудов совсем уж чокнутых авторов. И вот что я понял: любитель, придумавший слабый шифр, может его описать и разложить по шагам. Мечтатель не сможет излить смутные, но грандиозные плоды своего воображения на бумагу. Он будет изводить целые стопки бумаги, расписывая чудесные свойства своего шифра, но не в силах выписать его шаги. Он не способен превратить свои бессвязные мысли в конкретный алгоритм.

Начиная с 2005 года я стал посещать курсы в колледже Марист по программе непрерывного образования. Вскоре я читал лекции по sudoku, SumSum и другим головоломкам (я написал три книги о sudoku), своим путешествиям по Танзании и Монголии, конструкции Эмпайр Стейт Билдинг, жизни Алана Тьюринга и другим темам. Я стал членом комиссии по учебным планам.

В 2018 году я вызвался прочесть двухсеместровый курс по криптографии. Подготовив почти 450 слайдов, я понял, что материала достаточно для книги. И на мое счастье, обнаружилось, что годом раньше я уже начал писать как раз такую книгу. Вот эту.

# Благодарности

---

На днях я случайно услышал, как моя жена Мириам в разговоре по телефону сказала: «У нас тут секс втроем, я, Фрэнк и книга». Спасибо, Мириам, что ты 18 месяцев терпеливо сносила, как я писал эту книгу, потом год искал издателя, шесть месяцев охотился за литературным агентом, год наблюдал за отсутствием у агента каких-либо результатов и, наконец, месяц искал пристанище для этой книги в издательстве Manning. И еще 18 месяцев рецензирования, правки, редактирования, верстки, правки, составления указателя и всякой всячины.

Я благодарен всем сотрудникам издательства Manning Publications, которые помогали мне с этой книгой, особенно Майклу Стивенсу, который ухватился за шанс и предложил мне контракт, а потом помогал на всех стадиях процесса; Марине Майклз за обширную редакторскую правку; Ребекке Райнхарт, которая сделала мой путь не таким тернистым; Джэн Хоул и Сьюзан Хониуэлл за работу над иллюстрациями; Тиффани Тэйлор за многочисленные ценные замечания по поводу грамматики и пунктуации; Полу Уэллсу и Кэри Хейлз за работу над производством книги; Сэму Вуду за рекламный текст; Дэннису Далиннику за верстку и, конечно, Марджану Бейсу, издателя.

Отдельное спасибо профессору Рэндаллу К. Николсу, написавшему предисловие к книге и рецензию в «The Cryptogram», несмотря на крайне сжатые сроки. Спасибо также профессору Томасу Перера из Музея Энигмы за предоставленные изображения Фиалки.

Я благодарю рецензентов, которые прочитали рукопись и сделали множество полезных критических замечаний и предложений: Кристофера Карделла, Алекса Лукаса, Габора Хаджба, Михала Рутка, Джейсон Тэйлор, Роя Принса, Мэттью Харвелла, Риккардо Маротти и Пола Лава. Ваши предложения помогли сделать книгу лучше.

Наконец, не могу не отметить нечаянную роль Ли Харви Освальда: совершив чудовищное убийство президента Кеннеди, он помешал мне прийти на собеседование по безопасности в штаб-квартиру ФБР, что не дало мне возможности поступить на работу в АНБ, где написание подобной книги считалось бы преступным деянием.

# Об этой книге

---

## *Для кого предназначена эта книга*

Книга рассчитана на широкую аудиторию: массового читателя, криптографов-любителей, почитателей истории, студентов компьютерных специальностей, инженеров-электротехников, математиков и профессиональных криптографов. Это усложнило мне работу, потому что невозможно сделать все части книги одинаково интересными для всех. Для кого-то в некоторых частях окажется слишком много математики. А кому-то какие-то части покажутся чересчур элементарными. В этом разделе я попробую подсказать читателям, что, на мой взгляд, им стоит прочитать.

- **Массовые читатели** могут читать всё подряд до конца главы 8. Если математика покажется слишком сложной или изложение – перенасыщенным техническими подробностями, просто пропускайте соответствующие страницы. Начиная с главы 9 материал становится более трудным. Дальше можно читать выборочно, только то, что кажется интересным. Быть может, имеет смысл прочитать главу 12, чтобы получить общее представление, не вдаваясь в детали.
- **Криптографы-любители**, вероятно, захотят прочитать книгу целиком, а затем более внимательно изучить разделы 4.2–5.11, 6.1–6.5, 6.7, большую часть главы 7, а также разделы 9.1–9.9 и главы «Развлечения» и «Задачи».
- **Почитатели истории** могут прочитать книгу целиком, пропуская всю математику, но обращая внимание на то, когда и кем были изобретены различные методы.
- **Студентам компьютерных специальностей** рекомендую уделить особое внимание разделам 5.6–5.11, главе 8 и главам 11–16.
- **Инженеров-электротехников** могут заинтересовать практические методы. Им стоит прочитать главы 2 и 4, где излагаются



основы, а затем разделы 7.2–7.8, главу 9 и главы 11–16, обращая особое внимание на главу 12.

- **Математикам** будут особенно интересны раздел 4.5, разделы 5.6–5.12, 10.4–10.7, 11.7–11.10, 12.3–12.6, главы 13–16, в особенности раздел 16.4.6, и глава 18.
- Для **профессиональных криптографов** интерес могут представлять разделы 7.8, 8.2, 10.5, 10.7, 11.4, 12.3–12.6, 13.8, 13.15, 14.2, 14.4, 15.4–15.14, 16.4, 16.5 и 18.12.

## О шифрах

Я включил несколько развлекательных головоломок и более серьезных задач для читателей, которые хотят попробовать свои силы во вскрытии шифров. Для решения головоломок достаточно стандартных методов, описанных в книге.

При решении задач применяются методы, которые я придумал сам. Они достаточно просты, так что любитель сможет догадаться о методе и решить задачу. Я старался не вредничать и дать возможность интересующимся читателям найти решение. Не бойтесь – там нет ничего заумного или чрезмерно сложного. Никаких несуществующих слов или искаженных частот букв. И достаточно материала для решения.

Некоторые разделы начинаются символом \* и заканчиваются символами \*\*. Это факультативные разделы, которые могут содержать компьютерные алгоритмы или углубленную математику. Кто-то захочет их пропустить.

## Форум на сайте liveBook

Приобретение этой книги открывает бесплатный доступ к платформе liveBook онлайн-чтения, созданной издательством Manning. Средства обсуждения на liveBook позволяют присоединять комментарии как к книге в целом, так и к отдельным разделам или абзацам. Совсем несложно добавить примечания для себя, задать или ответить на технический вопрос и получить помощь от автора и других пользователей. Для доступа к форуму перейдите по адресу <https://livebook.manning.com/book/secret-key-cryptography/discussion>. Узнать о форумах Manning и правилах поведения на них можно по адресу <https://livebook.manning.com/discussion>.

Издательство Manning обязуется предоставлять площадку для содержательного диалога между читателями, а также между читателями и автором. Но это обязательство не подразумевает какого-то конкретного объема присутствия со стороны автора, участие которого в работе форума остается добровольным (и не оплачивается). Мы рекомендуем задавать автору трудные вопросы, чтобы его интерес не угасал! Форум и архивы прошлых обсуждений остаются доступны на сайте издательства до тех пор, пока книга продолжает допечатываться.

## Другие онлайн-ресурсы

Криптографические продукты, созданные автором, можно найти на его сайте по адресу [www.mastersoftware.biz](http://www.mastersoftware.biz).

## Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте [www.dmkpress.com](http://www.dmkpress.com), зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com); при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу [http://dmkpress.com/authors/publish\\_book/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com).

## Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу [dmkpress@gmail.com](mailto:dmkpress@gmail.com). Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

## Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты [dmkpress@gmail.com](mailto:dmkpress@gmail.com).

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

## Об авторе

---



**Фрэнк Рубин** – обладатель степени бакалавра и магистра математики и доктора информатики. Он 28 лет проработал в подразделении автоматизации проектирования компании IBM, где разрабатывал специализированное программное обеспечение, которым инженеры IBM пользовались при проектировании компьютеров и электрических схем. Он владелец компании Master Software Corp., разрабатывающей криптографические продукты. Фрэнк – автор четырех патентов США по криптографическим методам. Он автор примерно 50 работ, опубликованных в реферируемых журналах по криптографии, а также нескольких внутренних документов IBM (руководств пользователя и проектных спецификаций). В области криптографии он известен прежде всего тем, что вскрыл цилиндрический шифратор Джефферсона. В информатике хорошо известен его метод арифметического кодирования, ставший одним из стандартных методов сжатия текстов, а также его алгоритм нахождения гамильтоновых путей. В чистой математике известна его идея применить распознаватель с конечным числом состояний к теории меры. Фрэнк опубликовал три книги по sudoku, а также две «самиздатовские» книги по головоломкам SumSum. Он автор более 3500 задач, опубликованных в журналах «The Cryptogram», «Technology Review» и «Journal of Recreational Mathematics» (JRM), и единственный, удостоившийся специального выпуска JRM, посвященного исключительно его задачам.

# Об иллюстрации на обложке

---

На обложке книги изображен «Le Garçon de Bureau», или «Конторский служащий». Рисунок взят из книги, изданной под редакцией Луи Кюрмера в 1841 году. Все иллюстрации мастерски нарисованы и раскрашены вручную. В те времена легко было по одежде определить, где человек живет, чем занимается и каков его статус. Издательство Manning откликается на новации и инициативы в компьютерной отрасли обложками своих книг, на которых представлено широкое разнообразие местных укладов быта в прошлых веках. Мы возвращаем его в том виде, в каком оно запечатлено на рисунках из таких собраний, как это.

# 1

## Введение

---

Я занимаюсь криптографией больше 50 лет. За это время я очень многому научился. И в этой книге я хочу передать свои знания следующему поколению криптографов. Многие изложенные здесь сведения – новые открытия, которых вы не найдете ни в какой другой литературе.

Я знаю, что на тему криптографии уже написано много книг. И если я хочу, чтобы мою книгу читали, то должен предложить какие-то мысли, которых нет в других книгах, идеи, о которых другие авторы не знают или считают их невозможными. Книга должна стать **СЕНСАЦИЕЙ**. Поехали! Вот что я сделаю:

- расскажу простым нетехническим языком, как построить невскрываемый шифр;
- предложу свыше 140 шифров, готовых к применению. 30 из них невскрываемые;
- снабжу вас инструментарием и методами, позволяющими комбинировать и дополнительно укреплять шифры;
- опишу вычисление, которое позволит точно измерить стойкость шифра и гарантировать, что он невскрываемый;
- покажу, как построить и включить в проект коды, сжимающие данные;
- раскрою практический метод получения невскрываемого шифра с помощью одноразового блокнота;
- расскажу, как генерировать сразу много истинно случайных чисел;

- покажу, как находить очень большие и безопасные простые числа;
- научу добавлять необнаруживаемую закладку в шифр;
- раскрою потенциально фатальный дефект в квантовой криптографии;
- объясню, как бороться с гипотетическими ультракомпьютерами, которые, возможно, будут разработаны через несколько десятилетий (а возможно, уже существуют, только это не афишируется).

Книга написана разговорным языком, как будто мы ведем дружескую беседу. Говоря «мы» или «нас», я имею в виду, что вы, читатель, и я, автор, совместными усилиями стараемся решить какую-то задачу или защитить какой-то секрет.

Эта книга – не научный труд. Я упоминаю о происхождении методов и идей, когда примерно знаю источники и даты, но многие знания я приобрел неформально. Вы почти не найдете ссылок, сносок и комментариев эрудита. Я хотел написать практически полезную книгу. Следуйте изложенным рекомендациям – и получите безопасный шифр. Сто пудов.

Иногда я включаю любопытные исторические факты – отчасти чтобы снять напряжение, а отчасти чтобы воссоздать историческую правду. Я знаю, что изучать криптографию – тяжкий труд. И надеюсь, что речь от первого лица, анекдот-другой и толика юмора помогут немного облегчить его.

В книге много нового материала. Приведены методы построения и взлома шифров, которые раньше нигде не публиковались. Есть даже несколько моих собственных математических открытий. Их вы найдете только в этой книге. Есть куча практических советов, как сделать то или другое, несколько компьютерных методов, рассказано, как можно ускорить вычисления или обойтись меньшей памятью.

Упор в книге сделан на особо безопасную криптографию. У вас имеется информация, которую необходимо сохранить в секрете от противника, располагающего суперкомпьютерами или даже квантовыми компьютерами. Из этой книги вы узнаете, как это сделать. Я представлю широкий набор инструментов, новых и давно известных, которые можно комбинировать бесчисленными способами, получая в итоге шифры сколь угодно высокой стойкости. Студенты, изучающие криптографию, и программисты, применяющие ее в работе, найдут здесь широчайший спектр практических методов, которые можно использовать для разработки новых криптографических продуктов и сервисов.

При всем при том я хочу, чтобы изложенный материал был доступен как профессионалам, так и любителям. Есть немало методов, которые можно реализовать, имея лишь листок бумаги и карандаш. Один такой метод описан в конце раздела 9.6.1. Эти методы пригодны для работы в полевых условиях, когда нет ни электричества, ни электронных устройств. Есть даже несколько шифров, доступных детям.

*Любой человек может создать невскрываемый шифр.*

И вы можете создать невскрываемый шифр. Нужно только знать, как это сделать. Если вы сумеете прочесть и понять эту книгу целиком или хотя бы наполовину, то сможете создать невскрываемый шифр. Эта книга научит любого желающего, как построить шифр, который устоит против атаки, всерьез организованной профессиональным криптографом, располагающим суперкомпьютером. Никакая другая книга не может этим похвастаться. На самом деле для разработки собственного безопасного шифра не нужно ничего, кроме карандаша и бумаги. Я собрал большую коллекцию методов и идей начиная с XV века и покажу вам, какие комбинации увеличивают стойкость шифра, а какие являются пустой тратой времени. Я вооружу вас проверенными временем приемами, дополнив их совсем новыми техниками, которые позволят возвести неприступную крепость.

Честное предупреждение: по образованию я математик, а по профессии специалист по информатике, так что без стеснения пользуюсь математической нотацией и математическими понятиями. Эта книга адресована не только инженерам и математикам, но и более широкой аудитории. Я буду объяснять всю необходимую математику, так что обращаться к другим источникам не придется. Если вы понимаете, что такое нижние индексы и показатели степени, если можете читать выражения, содержащие скобки, то никаких других математических знаний и не понадобится. Все сверх того – простые числа, модульная арифметика, операции над матрицами и математические кольца – я объясню здесь же.

Если вы не понимаете какую-то математическую идею, то есть три пути: (1) поверить мне на слово, (2) пропустить раздел целиком или (3) не использовать соответствующий криптографический метод. Есть достаточно других. И некоторые точно вас устроят.

Или просто впрягайтесь и читайте разделы, посвященные математике. Вы удивитесь тому, как много нового узнаете. Не расстраивайтесь, если не понимаете какую-то тему. Возможно, следующая окажется проще. Даже профессиональные математики понимают не всё.

# 2

## Что такое криптография?

---

### **Краткое содержание главы:**

- основные криптографические термины;
- что такое невскрываемый шифр;
- какие есть виды криптографии.

Криптографию часто называют «искусством тайнописи». Но этим она не исчерпывается. Криптография включает всё: от невидимых чернил до передачи сообщений с применением квантового запутывания фотонов. В частности, криптография включает придумывание и вскрытие кодов и шифров.

Разные авторы придают криптографическим терминам разный смысл, поэтому с самого начала договоримся о некоторых основных терминах.

*Открытым*, или *незашифрованным*, *текстом* называется сообщение или документ, который мы хотим сохранить в секрете. В традиционной криптографии сообщение было бы записано текстом на некотором языке, известном как отправителю, так и получателю. В век компьютеров это может быть файл любого типа, например: PDF (текст), JPG (изображение), MP3 (аудио) или AVI (мультимедиа).

*Шифром* называется метод или *алгоритм*, который искажает сообщение до неузнаваемости, например изменяя порядок символов или заменяя одни символы другими. В общем случае шифры при-



меняются к отдельным символам или группам символов текста безотносительно к их смысловому содержанию.

*Ключом* называется секретная информация, известная только отправителю и правомочному получателю (или получателям). Ключ определяет, какое преобразование применяется к каждому сообщению. Например, если шифр (метод) заключается в изменении порядка букв в сообщении, то ключ может указывать, какой порядок использовать в сообщениях текущего дня. Ключ может быть буквой, словом или фразой, числом либо последовательностью букв, слов и чисел. Стойкость шифра сильно зависит от длины используемых в нем ключей.

*Ключевым словом* или *ключевой фразой* называется слово или фраза, используемые в роли ключа.

*Шифрованием* называется процесс преобразования открытого текста в нечитаемую абракадабру полномочным отправителем, знающим ключ.

*Шифртекстом* называется нечитаемое сообщение или документ, предназначенные для передачи или хранения.

*Дешифрированием*, или *расшифровыванием*, называется процесс, который используется полномочным получателем, знающим метод и ключ, для преобразования нечитаемого шифртекста в исходный открытый текст.

*Кодом* также называется преобразование сообщения, делающее его нечитаемым. В отличие от шифра, код обычно применяется к словам или фразам сообщения. Типичный код заменяет слова или фразы группами цифр или букв. (Тут имеет место путаница – слово *код* также означает стандартизованное представление букв, например код Морзе. Надеюсь, что смысл будет понятен из контекста.)

*Криптологией* называется формальное изучение криптографии, т. е. математические идеи и методы, применяемые для построения и вскрытия шифров. Ученые занимаются криптологией, а взломщики шифров применяют криптоанализ.

*Криптоанализом* называется изучение кодов и шифров с конкретной целью – найти в них слабые места и способы вскрытия или, наоборот, способы повысить стойкость.

*Вскрытием кода* называется процесс прочтения зашифрованных сообщений третьей стороной (врагом или противником), не знающей ключа, а возможно, даже и метода. Это можно проделывать, применяя математические методы или просто терпеливо перехватывая и сопоставляя большое число сообщений, но на практике чаще все сводится к трем В: bribery (подкуп), blackmail (шантаж) и break-in (взлом системы).

## 2.1 Невскрываемые шифры

Итак, о терминологии мы договорились, а теперь позвольте мне перейти к главному вопросу. Что я понимаю под словом «не-

вскрываемый»? Во-первых, что шифр невозможно вскрыть криптографическими средствами. В их число не входят взлом, подкуп, принуждение, предательство, шантаж, медовые ловушки и другие подобные способы. Все это нам неинтересно. Во-вторых, я имею в виду, что шифр нельзя вскрыть на практике. Противник располагает конечными ресурсами и конечным временем для вскрытия шифра. Выбирая шифр, вы должны хотя бы примерно представлять, сколько человеческих ресурсов и вычислительных мощностей потенциальный противник может потратить на его вскрытие. Делайте предположения с запасом, учитывайте возможность усовершенствования компьютеров, добавьте еще немного для пушей безопасности и определитесь с числовой величиной. Тогда при выборе шифра вам будет на что ориентироваться. Добейтесь выполнения этого ориентира – и ваш шифр будет практически невскрываемым.

Помните, что время жизни многих сообщений ограничено. Если сообщение гласит «АТАКУЕМ НА РАССВЕТЕ», а враг прочтет его в полдень, то будет уже поздно. Атака уже произошла. Шифр, который нельзя вскрыть за 12 часов, можно считать практически невскрываемым, если у противника этих 12 часов нет.

Еще раз поясню, чтобы не было никаких сомнений: говоря, что шифр вскрыт, я имею в виду, что противник может читать сообщения, отправленные с применением этого шифра. Даже если противник может прочесть всего 1 % или 0.01 % сообщений, шифр все равно считается вскрытым. Но где-то проходит рубеж. Если противник может прочитать сообщение, только если предварительно перехватил много сообщений такой же длины, зашифрованных тем же ключом, или если 63 из 64 бит ключа нулевые, то шифр вскрытым не считается. У противника нет априорного способа узнать, какие сообщения каким ключом зашифрованы или какие ключи состоят почти из одних нулей. Может случиться, что вы никогда и не отправите двух сообщений одинаковой длины, зашифрованных одним ключом или ключом, в котором 63 из 64 бит нулевые.

Если в шифре используется 256-разрядный ключ и вражеский криптоаналитик нашел математический или вычислительный способ сократить его длину до 200 или даже до 150 бит, то шифр может быть ослабленным, но все равно не считается вскрытым, коль скоро вы выбрали уровень безопасности 128 бит. Использование 256-битового ключа для обеспечения 128-битового уровня безопасности дает огромный запас прочности.

Когда правительство США решило, что старый стандарт шифрования данных DES уже не является безопасным, оно объявило международный конкурс на разработку нового шифра. Предложения поступили со всего мира. Их количество исчислялось десятками. Сотни криптографов оценивали предложенные шифры с точки зрения скорости и безопасности. С 1997 по апрель 2000 года состоялось три раунда выявления победителя. Именно так и следует поступать, когда

речь идет о шифре, который станет международным государственным стандартом для банков, промышленности и армии. Если вы подумываете принять участие в следующем конкурсе, то эта книга поможет подготовиться.

Но большинство читателей вряд ли будут пытаться. У их шифров будет более ограниченная сфера применения. Они могут довериться собственному суждению или придуманному ими процессу верификации при оценке своих шифров. Принципы, изложенные в главе 12, помогут принять верное и обоснованное решение.

## 2.2 Виды криптографии

Существует много видов криптографии. Перечислим несколько видов, которые использовались в прошлом:

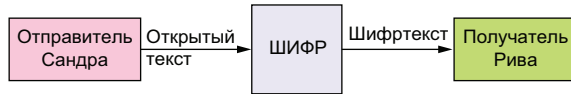
- *скрытое сообщение*, например гонец мог проглотить сообщение, спрятать его в каблуке или в седле либо просто запомнить. В древности часто заставляли гонца заучивать сообщение на не понятном ему языке;
- *тайный метод*, например шифр Цезаря, в котором каждая буква алфавита заменялась буквой, отстоявшей от нее на три позиции. То есть А заменялась на D, В – на Е, С – на F и так далее;
- *замаскированное сообщение*, похожее на что-то другое, например на деталь одежды гонца;
- *невидимое сообщение*, например микроточки или невидимые чернила, которые проявляются при нагревании или обработке кислотой;
- *ложный путь*, например когда истинным сообщением является подпись или форма и цвет бумаги, а все остальное служит только для отвлечения внимания или дезинформирования.

Все эти методы носят общее название *стеганография*, впервые этот подход был описан в книге «Steganographia», изданной в 1499 году бенедиктинским аббатом Иоганном Тритемием, урожденным Иоганном Гейденбергом. Сама книга Тритемия является примером стеганографии, потому что замаскирована под книгу о магии.

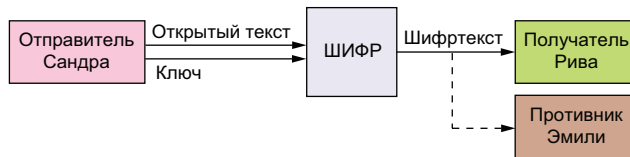
У некоторых стеганографических методов есть современные аналоги. Например, сообщение можно скрыть в JPEG-файле изображения, используя только младшие биты каждого пикселя. Другой пример – использование генератора случайных чисел для выбора некоторых битов в каждом байте файла. Выбранные биты содержат сообщение, а остальные могут быть ничего не значащим мусором.

Прежде чем приступать к описанию современных шифров, я хотел бы ввести полезное соглашение. Сообщение передается от отправителя к получателю, а цель шифрования – не дать противнику прочитать сообщение. Для краткости я буду называть отправите-

ля Сандра, полномочного получателя Рива, а противника Эмили<sup>1</sup>. Это более естественно, чем традиционные Алиса, Боб и Кэрл, не правда ли?



Обычно Сандра шифрует сообщение на своей стороне, прежде чем отправить его Риве. Сообщение можно отправлять любым способом, включая письмо, телефон, интернет, коротковолновую радиосвязь, сигнальный прожектор, кратковременный импульс, телеграф, волоконно-оптический кабель, семафор, квантовое запутывание или даже дымовые сигналы, правда, только в пределах прямой видимости. Чтобы картина была полной, отмечу, что шифр может требовать не только открытого текста, но и ключа, а противник может подслушивать. Вот более полная схема.



Современные шифры обычно попадают в одну из трех категорий: с секретным ключом, с открытым ключом и с персональным ключом. Ниже приведены их основные характеристики.

**Шифр с секретным ключом.** Сандра владеет секретным ключом, которым пользуется для зашифровывания сообщений. Рива имеет соответствующий секретный ключ, которым пользуется для дешифрирования сообщений. Это может быть тот же самый или *обратный* ключ. Обычно контроль над ключом находится в руках Сандры. При изменении ключа Сандра должна отправить новый ключ или обратный к нему Риве. Это стандартная парадигма классической криптографии.

**Шифр с открытым ключом.** Рива владеет открытым ключом шифрования, который сообщает всем желающим. Когда Сандра хочет отправить Риве сообщение, она шифрует его открытым ключом Ривы. Рива также владеет закрытым ключом, известным только ей. С его помощью она может дешифрировать полученные сообщения. Эта схема будет работать, только если никто не может вычислить секретный закрытый ключ по открытой информации. Из методов с открытым ключом наиболее распространен алгоритм RSA, придуманный Рональдом Ривестом, Ади Шамиром и Леном Адлеманом в 1975 году.

<sup>1</sup> Английские имена Sandra, Riva и Emily созвучны словам sender (отправитель), receiver (получатель) и enemy (противник). – *Прим. перев.*

*Шифр с персональным ключом.* И у Сандры, и у Ривы есть персональный ключ, который они никому не сообщают. Поскольку ключи не передаются и не разделяются, криптографию с персональным ключом иногда называют *бесключевой*. Работает это следующим образом. (Проход 1) Сандра зашифровывает сообщение своим персональным ключом и отправляет его Риве. (Проход 2) Рива зашифровывает полученное сообщение своим персональным ключом и отправляет дважды зашифрованное сообщение Сандре. (Проход 3) Сандра дешифрирует сообщение своим персональным ключом и отправляет его назад Риве. Теперь сообщение зашифровано только ключом Ривы, который она и использует, чтобы прочитать его.

Хитрость в том, что операции шифрования Сандры и Ривы должны *коммутировать*. То есть результат не зависит от того, кто шифрует сообщение первым: Сандра или Рива. Символически это записывается в виде  $SRM = RSM$ , где  $M$  – сообщение, а  $S$  и  $R$  – операции шифрования, выполняемые Сандрой и Ривой. Достоинство криптографии с персональным ключом в том, что любой человек может безопасно общаться с любым другим, не готовя ключи предварительно и не передавая их, поэтому можно не опасаться, что ключ будет перехвачен.

Криптографию с персональным ключом называют еще *трехпроходным протоколом*. Протокол – это последовательность шагов, используемая для достижения определенной цели, например передачи сообщений. Иными словами, протокол – это алгоритм. Идея трехпроходного протокола была предложена Ади Шамиром в 1975 году, а конкретный метод, представленный в этой книге, – мое изобретение.

## 2.3 Симметричная и асимметричная криптография

Во многих книгах пишут, что есть два типа криптографии: с *симметричным* и *асимметричным* шифром. Идея в том, что в криптографии с секретным ключом Сандра и Рива пользуются одним и тем же ключом для шифрования или дешифрирования сообщения, тогда как в криптографии с открытым ключом Сандра использует один ключ, а Рива – обратный к нему. В этой дихотомии не нашлось места криптографии с персональным ключом, которая не является ни симметричной, ни асимметричной, а равно различным классическим методам, упомянутым в начале раздела 2.2. Более того, классификация симметричная–асимметричная не всегда точна. В разделе 15.1 я опишу шифр Хилла – метод с секретным ключом, в котором шифрование сводится к умножению сообщения на ключ, а дешифрирование – к умножению на обратный ключ – так же, как в криптографии с открытым ключом.

Классификация шифра как симметричного или асимметричного не особенно полезна. Она не отражает существенного различия между криптографией с секретным и открытым ключом: в криптографии с секретным ключом все ключи хранятся в секрете, а в криптографии с открытым ключом каждая сторона хранит в секрете один ключ, а второй делает общедоступным.

Криптография с открытым ключом и с персональным ключом появились примерно в 1975 году. Криптография с открытым ключом так распалила воображение, что с тех пор методам с секретным и персональным ключом уделялось мало внимания. Криптография с открытым ключом подробно описана во многих книгах. Но эта книга посвящена в основном криптографии с секретным ключом – фундаменту, на котором покоится криптография.

## 2.4 Блочные и потоковые шифры

Другой способ классификации – разделение шифров на блочные и потоковые. Блочные шифры применяются к блокам символов сообщения, скажем к блокам по 5 символов. Обычно размер всех блоков одинаков и для каждого используется один и тот же ключ.

Потоковые шифры применяются к одному символу сообщения за раз. У каждого символа имеется собственный ключ, называемый *ключом символа*, который обычно получается из более длинного *ключа сообщения*. В старых потоковых шифрах ключ сообщения повторялся. Например, если длина ключа сообщения равна 10 символам, то первый его символ обычно использовался для шифрования символов сообщения с номерами 1, 11, 21, 31, ..., второй символ – для шифрования символов сообщения с номерами 2, 12, 22, 32, ... и так далее. Шифр с регулярно повторяющимся ключом называется *периодическим*. В более современных потоковых шифрах длина ключа сообщения обычно совпадает с длиной самого сообщения и называется *гаммой*. Такой *непериодический* подход к шифрованию называется одноразовым блокнотом. В главе 13 мы обсудим, как генерируются гаммы.

Разделение шифров на блочные и потоковые не является взаимно исключающим. Существуют гибридные шифры, в которых сообщение разбивается на блоки, но разные блоки шифруются разными ключами, т. е. шифр применяется к потоку блоков, а не к потоку символов.

## 2.5 Механические и цифровые шифры

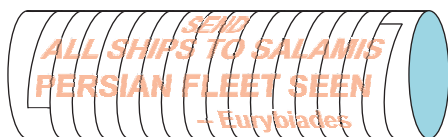
Шифры можно классифицировать также по средствам вычисления. На заре истории шифрование производилось вручную. И не с по-

мощью карандаша бумаги, а с помощью стилоса и пергамента или стилоса и глиняной таблички.

Первым механическим средством шифрования была скитала, или считала, которая использовалась в Древней Греции и Спарте предположительно еще в 700 году до н. э. Она представляла собой цилиндр, обвитый по спирали узкой полоской кожи или пергамента таким образом, что края соседних витков точно совпадали – без пробелов и перекрытий. Когда полоска разворачивалась, были видны только разрозненные части букв, в которых враг не мог распознать сообщение. Иногда наносились дополнительные завитки или раскраска, чтобы предмет выглядел как украшение.

Отправитель сохранял инструмент для чтения и записи будущих сообщений. Гонец мог носить полоску кожи как пояс, подвязывать ей волосы или использовать как седельную подпругу. Получателю для прочтения сообщения нужна была палка такого же диаметра. Разумеется, гонцу не сообщали о назначении этой ленты или ремня. Возможно, она даже вшивалась в одежду без ведома владельца.

Ниже показано изображение скиталы из книги Джованни Баттиста Порты «De Occultis Literarum Notis», изданной в 1593 году. Обратите внимание, что греческие буквы расположены на нескольких витках кожаной ленты.



Греки хранили секрет скиталы в течение 700 лет или около того. Римлянам, однако, не так повезло. В конце концов их враги из Северной Европы разгадали, для чего эти палки предназначены и как ими пользоваться. Поэтому римляне изобрели специальный измерительный инструмент, представляющий собой полый медный или бронзовый додекаэдр – правильный многогранник с 12 пятиугольными гранями – с круглым отверстием на каждой грани. Эти отверстия позволяли изготавливать деревянные цилиндры нужного диаметра. Когда губернатор (сатрап), посол или шпион отправлялся куда-то через вражескую территорию, безопаснее было везти с собой этот инструмент, а не саму скиталу, которую могли захватить. Двенадцать отверстий разного диаметра позволяли безопасно обмениваться сообщениями с другими губернаторами, послами или шпионами, например: маленькое для Лондиниума (нынешний Лондон), среднее для Лугдунума (нынешний Лион), а большое для Таррако (нынешняя Таррагона в Каталонии).

Насколько нам известно, о назначении этих додекаэдров так и не догадались ни жители Северной Европы, ни, кстати говоря, современные археологи. Археологи высказывали многочисленные экс-



травагантные гипотезы о цели этих предметов – детские игрушки, седельные украшения, кузнечные приспособления, канделябры, артиллерийские дальномеры и, когда ничего другого не осталось, предметы религиозного культа. Показанный ниже бронзовый додекаэдр был найден недалеко от Тонгерена, древнейшего города Бельгии, и экспонируется в музее галло-римской цивилизации.

Интересное замечание по ходу: в Википедии и на других сайтах говорится, что скитала использовалась для порождения перестановочного шифра, поскольку каждая буква записывалась в пределах одного витка полоски. Это неправда. Такую полоску легко было бы опознать как зашифрованное сообщение. Уж не важно, смог бы враг прочесть сообщение или нет, но гонцу точно помешали бы его доставить. Скрупулезный анализ вопроса о целых и разнесенных буквах можно найти в статье по адресу [cryptiana.web.fc2.com/code/scytale.htm](http://cryptiana.web.fc2.com/code/scytale.htm). В 1841 году Эдгар Алан По, который, кстати, был еще и талантливым криптографом, написал статью «Несколько слов о тайнописи», в которой привел хорошее описание скиталы и свой метод дешифрирования таких сообщений.

И усугубляя эту ошибку, в статье «Transposition cipher» в англоязычной Википедии написано, что скитала использовалась для порождения «заборного шифра» (rail fence cipher), называемого также зигзаговым. В этом шифре текст пишется на «штакетинах» воображаемого забора – сначала вниз по диагонали, по достижении нижнего края вверх, затем снова вниз и т. д. Но при записи сообщения вдоль или вокруг цилиндра направление никогда не изменяется. Поэтому если бы скитала и использовалась для порождения перестановочного шифра, то это была бы столбцовая перестановка, а никак не заборный шифр. (Я исправил эти ошибки в Википедии, но мои исправления удалили. И я решил, что роль википедийной полиции не по мне.)

В 1960-х годах появился такой вариант скиталы: взять отсортированную колоду перфокарт, написать карандашом сообщение на внешнем крае колоды, а затем тщательно перетасовать ее. Если затем подать колоду сортировочной машине, то порядок карт восстановится и сообщение можно будет прочитать. Идея широко обсуждалась в среде программистов, но я не знаю, была ли она воплощена на практике. Еще один современный эквивалент – написать сообщение на обратной стороне собранного пазла, а затем рассыпать пазл. Получатель должен будет собрать пазл, перевернуть его и прочитать сообщение.

Еще один механический шифр – цилиндр Джефферсона, изобретенный Томасом Джефферсоном между 1790 и 1793 годом. Он состоит из 36 деревянных дисков одинакового размера, нанизанных на железную ось, так что получается деревянный цилиндр. По внешнему краю каждого диска написано 26 букв латинского алфавита в произвольном порядке. Диски независимо вращаются, так что можно составить любое сообщение. Варианты шифратора Джеффер-



сона с дисками или бумажными полосками использовались вплоть до 1960-х годов.

С XV по XIX век было разработано много типов дисковых шифраторов. В самом распространенном использовалось несколько тонких концентрических дисков, вращающихся вокруг центральной оси. Вдоль ободка каждого диска написан алфавит или какой-то набор чисел и символов в некотором порядке. Диски фиксируются в определенном положении, а процесс шифрования выглядит так: найти букву открытого текста на одном диске, а затем использовать соответствующую букву или символ на одном из остальных дисков в качестве буквы шифртекста. В более поздних дисковых шифраторах внутренний диск сдвигался после шифрования каждой буквы – вручную или с помощью какого-то заводного механизма.

Следующее изображение дискового шифратора Леона Баттиста Альберти приведено в книге Августо Буонафальче «De compendis cifri», изданной в 1467 году. (Распространяется фондом WikiMedia Commons.)



Начиная с 1915 года последовала длинная череда электромеханических роторных шифров. Самый знаменитый из них – шифровальная машина Энигма, созданная в 1920-х годах немецким инженером Артуром Шербиусом. Количество типов устройств, выпущенных на рынок с пришествием компьютеров, исчисляется десятками. Все они порождали потоковые шифры. Идея заключалась в том, чтобы определять заменяющую букву с помощью прохождения электрического тока через ряд вращающихся роторов. После зашифрования буквы некоторые роторы проворачивались под управлением различных кулачков, шестеренок, лапок и собачек, которые изменяли порядок подстановки мириадами возможных способов. Поэтому если однажды из слова **INFANTRY** получилось **PMRNQGFW**, то такое событие могло не повториться на протяжении миллиардов оборотов.

Начиная с 1960-х годов криптография все активнее становится цифровой и переводится на компьютеры. В 1975 году компания IBM разработала стандарт шифрования данных DES (Data Encryption Standard), который был сертифицирован Национальным бюро

стандартов в 1977 году. Это положило начало целой серии блочных шифров, например Serpent и TwoFish, и в 2001 году увенчалось принятием улучшенного стандарта шифрования AES (Advanced Encryption Standard) Национальным институтом стандартов и технологий (NIST). Этот класс шифров рассматривается в главе 11.

Итак, развитие происходило по пути ручной → механический → электромеханический → цифровой.

## 2.6 Зачем выбирать шифр с секретным ключом?

В нашу эру криптографии с открытым ключом возникает естественный вопрос: зачем вообще нужна криптография с секретным ключом? Причин несколько.

Криптография с секретным ключом намного быстрее. Даже самые стойкие и сложные методы с секретным ключом работают в сотни, а то и тысячи раз быстрее лучших методов с открытым ключом. На самом деле основное применение криптографии с открытым ключом – шифровать ключи для криптографии с секретным ключом. Ключи передаются с применением методов с открытым ключом, но сами сообщения – с помощью методов с секретным ключом.

Для криптографии с открытым ключом (РКС) необходима инфраструктура открытых ключей. Должны существовать серверы открытых ключей, которые раздают открытые ключи потенциальным корреспондентам. Криптография с открытым ключом является мишенью самых разных атак с противником в середине или с подлогом, когда противник представляется отправителем, получателем или сервером распределения ключей. Поэтому для РКС необходима тщательная аутентификация и верификация. Лицо, запрашивающее открытый ключ, должно доказать свою принадлежность той же сети, что и получатель. Для сообщения, содержащего открытый ключ, необходимо проверить, что оно поступило от сервера. Получателя необходимо аутентифицировать как при первом размещении открытого ключа на сервере, так и при каждом последующем изменении. Если в сеть добавляется новая сторона, то авторизующее ее лицо должно быть аутентифицировано. Когда на сервер добавляется новая сторона, необходимо аутентифицировать все вовлеченные в процесс субъекты. Получатель должен проверять каждое полученное сообщение на предмет изменения или подмены третьей стороной. Все это приводит к обилию циркулирующих сообщений.

Для работы криптографии с секретным ключом все эти административные хлопоты не нужны. Два человека могут обмениваться сообщениями, зашифрованными секретным ключом, ни привлекая никого постороннего и не пользуясь системой-посредником. Когда несколько человек обмениваются такими сообщениями, нужно

проверить лишь, что каждая сторона располагает текущим ключом. Неавторизованное лицо не получит ключей и не сможет читать сообщения.

Обмен сообщениями – не единственное применение криптографии. Не менее важная роль – обеспечение секретности данных, хранящихся в компьютере, или на внешнем устройстве типа флешки, или в облачном хранилище – часто в течение длительного времени. Криптография с открытым ключом для этой цели не годится. Только методы с секретным ключом пригодны для хранения файлов данных в секрете.

Когда нужно разослать сообщение сразу нескольким получателям, сделать это с помощью методов с секретным ключом просто. Нужно лишь, чтобы у каждой стороны был ключ. Можно было бы использовать специальный широковещательный ключ, отличный от персональных ключей участников. Или каждой стороне можно было бы отправить ключ сообщения, воспользовавшись отдельным ключом для передачи ключей. Для применения методов с открытым ключом в этом случае нужно было бы получить открытые ключи всех получателей, выполнив всю необходимую авторизацию и верификацию. Заранее организовать это невозможно, потому что участники вправе изменять свои открытые ключи в любой момент.

Самый распространенный метод с открытым ключом – RSA. Его стойкость опирается на то, что в настоящее время очень трудно разложить на множители большие числа (см. раздел 3.4). Сейчас не существует практически осуществимого способа разложить на множители 200-значное десятичное число, не имеющее малых простых множителей. Но когда станут доступны квантовые компьютеры, все это изменится. Профессор MIT Питер Шор разработал квантовый алгоритм, который без труда разложит на множители число такого размера. Когда это случится, все зашифрованные RSA сообщения, хранящиеся в компьютерах, можно будет прочитать.

Но пока что неизвестен способ применить квантовый компьютер для вскрытия шифров с открытым ключом. Если вы озабочены появлением квантовых компьютеров, то криптография с открытым ключом – единственный выбор.

## 2.7 Зачем создавать собственный шифр?

Если вы любитель шифров, то зачем создавать собственные шифры, понятно. Хобби у вас такое. Любители моделей поездов строят и пускают по рельсам модели поездов. Любители авиамоделей строят и запускают в небо модели самолетов. А любители шифров строят и взламывают шифры.

Если вы студент, изучающий криптографию, то построение собственного шифра – хорошее упражнение. Это лучший способ на-

учиться создавать и оценивать шифры. Стандартный на текущий момент шифр AES (раздел 11.5) не будет таковым вечно, и кому-то придется проектировать ему замену. Если вы хотите принять участие в этой работе, то эта книга станет отличной отправной точкой.

Если вы профессиональный криптограф, отвечающий за защиту данных и коммуникаций, то построение собственных шифров может быть продиктовано здоровым скептицизмом по поводу того, так ли безопасны одобренные государством шифры. С вашего позволения, я расскажу одну историю, которая может оправдать такие сомнения.

В 1975 году IBM предложила шифр, ныне известный под названием DES (Data Encryption Standard). Ему предстояло стать мировым стандартом шифрования с секретным ключом. Поначалу, на этапе проектирования, ключ в DES имел длину 64 бита. Агентство национальной безопасности (АНБ) потребовало уменьшить длину ключа до 56 бит, а оставшиеся 8 использовать как контрольную сумму.

В этом не было никакого смысла. Если контрольная сумма так уж необходима, то следовало увеличить длину ключа с 64 до 72 бит. Многие считали, что истинная причина такого требования заключалась в том, что АНБ знало, как вскрывать шифр с 56-битовым ключом, а вот читать сообщения, зашифрованные 64-битовым ключом, не умело. Оказалось, что так оно и было.

Логично предположить, что АНБ никогда не одобрит стандарт шифрования, который не сможет взломать. А раз так, то можно заключить, что АНБ умеет взламывать все варианты AES, улучшенного стандарта шифрования. А если АНБ умеет взламывать AES, то вполне вероятно, что русские и китайцы тоже могут это делать.

В мире есть всего горстка экспертов, способных строить шифры-кандидаты, из которых потом выбираются мировые стандарты. Хорошо известно, что эти эксперты регулярно присутствуют на брифингах в штаб-квартире АНБ в Форт-Миде, штат Мэриленд. На этих встречах сотрудники АНБ рекомендуют методы, с помощью которых можно укрепить или ослабить шифры. Вполне возможно, что рекомендованные методы содержат, в частности, закладки, которые позволят АНБ и только АНБ легко вскрывать шифры. Допускаю также, что АНБ могло предлагать должности, контракты и гранты на исследования, чтобы побудить экспертов принять эти уязвимые методы.

Конечно, все это лишь предположения, но криптографы обычно очень осторожны. Если можно представить себе теоретическую слабость или уязвимость, то не важно, сумеет противник эксплуатировать ее на практике или нет, лучше будет защититься от нее.

Наконец, побудительным мотивом может быть повышенная скорость, простота реализации или более дешевое оборудование. Быть может, вы хотите построить собственный шифр, чтобы достичь этих целей, не жертвуя безопасностью. Описанные мной методы помогут в этом.

Вместе с тем не забывайте о многочисленных подводных камнях. Мало просто создать шифр и посчитать, что он «достаточно стой-

кий». Во многих и многих шифрах обнаруживались неожиданные слабости. Даже самый стойкий шифр может пасть жертвой ошибок оператора, допустим: начинать каждое сообщение стандартным заголовком, часто использовать ключи повторно или шифровать одно и то же сообщение разными ключами и отправлять. Например, многие немецкие сообщения во время Второй мировой войны были вскрыты, потому что начинались словами «Хайль Гитлер».

В этой книге имеется вся информация, необходимая для построения невскрываемого шифра, но помните, что прочтение всего одной книги по криптографии не сделает из вас эксперта. Обязательно укрепляйте свой шифр, применяя принципы, описанные в главе 12.

# Предварительные сведения

---

## **Краткое содержание главы:**

- биты и байты;
- функции и булевы операторы;
- простые числа и модульная арифметика.

Прежде чем переходить к существу предмета, познакомимся с некоторыми понятиями. Я буду двигаться довольно быстро, потому что сегодня этот материал преподают в школах даже не самого высокого уровня. Дополнительные сведения будут приведены позже, по мере необходимости.

## **3.1 Биты и байты**

Данные хранятся в компьютерах в форме *битов*, или *двоичных цифр*. Бит – это просто число, которое может принимать значения 0 или 1. Бит может храниться в компьютере по-разному. Это может быть замкнутое или разомкнутое реле. Или магнит, у которого северный полюс ориентирован вверх или вниз. Свет может быть поляризован

по часовой стрелке или против нее. Амплитуда электрического импульса может быть большой или малой.

Из двоичных цифр можно составлять двоичные числа. Ниже приведены 3-битовые двоичные числа и их десятичные эквиваленты. Такие 3-битовые числа называются восьмеричными, т. е. это числа в системе счисления по основанию 8.

000 = 0	100 = 4
001 = 1	101 = 5
010 = 2	110 = 6
011 = 3	111 = 7

Биты используются также для представления *логических значений*, или *значений истинности* в компьютерной логике. 0 представляет значение *ложь*, а 1 – значение *истина*.

Символ (или знак), например буква или цифра, может быть представлен 8-битовым двоичным числом – *байтом*. Термин *байт* был предложен в 1954 году Вернером Бухгольцем из компании IBM. Поскольку у каждого бита два возможных значения, восьмью битами можно представить  $2^8 = 256$  различных символов. Этого достаточно для 26 строчных букв, 26 заглавных букв, 0 десятичных цифр, 33 знаков препинания, например = и \$, а также ряда управляющих символов, таких как табуляция и перевод строки.

Существуют схемы, позволяющие представить дополнительные символы, например кириллическую букву Ж, арабскую *س* и даже китайский иероглиф 是; для этого используется до 4 байт на одну логограмму. Нас они интересовать не будут. Шифры могут работать с любыми строками символов, что бы они ни означали. Совершенно несущественно, что шифруемый байт – третий из 4 байт, представляющих китайский иероглиф.

Для наших целей важны три ипостаси байта: (1) это строка из 8 логических значений истина–ложь; (2) это 8-битовое двоичное число, т. е. целое число от 0 до 255 включительно; (3) это представление какого-то символа, например буквы, цифры, знака препинания, или часть логограммы.

## 3.2 Функции и операторы

Математические функции теперь изучают в начальной школе, поэтому я уверен, что разжевывать это понятие не нужно, но полезно поговорить о нотации и терминологии. Функция принимает одно или несколько значений и порождает новое значение. Принимаемые значения называются *входами*, или *аргументами*, *функции*, а возвращаемое – *выходом*, или *результатом*. Мы говорим, что для порождения результата функция *применяется* к аргументам.

Функцию можно обозначать символом, например  $+$  или буквой. Если используется символ, то он называется *оператором*, так что  $+$  и  $\times$  – операторы. Если функция имеет один аргумент, то символ можно поставить как перед аргументом, например  $-5$  или  $\sqrt{9}$ , так и после аргумента, например  $5!$  (число 5 факториал, равное  $1 \times 2 \times 3 \times 4 \times 5 = 120$ ). Если аргументов два, то символ ставится между ними, например  $3+4$  или  $6 \times 7$ . Если используется буква, то аргументы заключаются в скобки, например  $f(x)$ . Здесь функция обозначена буквой  $f$ , а аргумент – буквой  $x$ . Если аргументов несколько, то они разделяются запятыми, например  $f(a,b,c)$ . В некоторых книгах по языкам программирования проводится различие между аргументами и параметрами, но нам это не важно.

### 3.3 Булевы операторы

Функции, а также операторы сложения, вычитания, умножения и т. д. применяются к числам. Аналогично существуют функции, применяемые к битам, представляющим значения истинности. Эти функции называются *логическими*, или *булевыми*, *операторами* в честь английского математика Джорджа Буля.

Если  $A$  и  $B$  – значения истинности, то логические функции *not*, *and*, *or* и *xor* определяются следующим образом:

**not**  $A$  истинно, если  $A$  ложно, и ложно, если  $A$  истинно;  
**A and B** истинно, если и  $A$ , и  $B$  истинны, и ложно в противном случае;  
**A or B** истинно, если или  $A$ , или  $B$ , или оба одновременно истинны, и ложно в противном случае;  
**A xor B** истинно, если ровно одно из  $A$  и  $B$  истинно, и ложно в противном случае.

Иными словами, **A xor B** истинно, если  $A$  истинно и  $B$  ложно или если  $B$  истинно и  $A$  ложно. Оператор **xor** называется *ИСКЛЮЧАЮЩИМ ИЛИ*. Обычно он обозначается символом  $\oplus$ . Операторы **and** и **or** часто обозначаются символами  $\wedge$  и  $\vee$  соответственно. Запомнить, что есть что, просто: символ  $\wedge$  для **and** похож на заглавную букву  $A$  без черточки.

Ниже приведены значения всех четырех булевых функций в табличной форме.

and	or	xor	not
00 0	00 0	00 0	0 1
01 0	01 1	01 1	1 0
10 0	10 1	10 1	
11 1	11 1	11 0	

Эти четыре оператора можно распространить на строки битов – нужно только применять их к парам соответствующих битов. Если  $A$



равно 0011, строке четырех битов, представляющих значения ложь, ложь, истина, истина, а В равно 0101, т. е. представляет логические значения ложь, истина, ложь, истина, то применение булевых операторов дает:

	<b>and</b>	<b>or</b>	<b>xor</b>	<b>not</b>	Оператор
<b>A</b>	<b>0011</b>	<b>0011</b>	<b>0011</b>	<b>0011</b>	Первый операнд
<b>B</b>	<u><b>0101</b></u>	<u><b>0101</b></u>	<u><b>0101</b></u>	_____	Второй операнд
	<b>0001</b>	<b>0111</b>	<b>0110</b>	<b>1100</b>	Результат

Оператор ИСКЛЮЧАЮЩЕЕ ИЛИ используется в криптографии сплошь и рядом. Например, в простой реализации одноразового блокнота (см. главу 14) он применяется к байтам сообщения и байтам гаммы:

	<b>H</b>	<b>E</b>	<b>L</b>	<b>P</b>	Открытый текст
<b>Сообщение</b>	<b>01001000</b>	<b>01000101</b>	<b>01001100</b>	<b>01010000</b>	Открытый текст в кодировке UTF-8
<b>Ключ</b>	<u><b>10101100</b></u>	<u><b>10001011</b></u>	<u><b>11000010</b></u>	<u><b>00111001</b></u>	
	<b>11100100</b>	<b>11001110</b>	<b>10001110</b>	<b>01101001</b>	Шифртекст

## 3.4 Системы счисления

В обычной арифметике числа записываются в десятичной нотации. Эту нотацию изобрели индусы и арабы между V и VII веком. Поэтому десятичные цифры называются также *арабскими*. Распространению данной системы в Европе способствовал Леонардо Пизанский, в то время больше известный как Фибоначчи.

### Историческое отступление

Во времена Леонардо, приблизительно 1175–1250 гг., бешеной популярностью пользовались сдвижные головоломки (некоторые полагают, что это то же самое, что игра в пятнашки, предположительно изобретенная Нойесом Чапманом в 1874 году). Проводились публичные состязания с денежными призами. В этом деле Леонардо достиг небывалых высот. Он неизменно побеждал. Соперники в шутку прозвали его «Фибоначчи», т. е. «везунчик», и Леонардо принял это прозвище. Имя Фибоначчи гремело по всей Италии. В 1202 году он написал свою «Книгу абака» и хотел, чтобы люди знали, что ее автор – знаменитый Фибоначчи. Считая недостойным хвастливо заявлять об этом прямо, он написал на титульной странице **Filius Bonacci**, что могло означать как «Счастливый сын», так и «Сын Боначчи».

Более поздние авторы не поняли его намерения и отвергли саму мысль о том, что великого Леонардо Пизанского можно называть везунчиком. Они предположили, что Леонардо происходит из семьи Боначчи. По той

же причине – напомнить читателям, что он и есть знаменитый Фибоначчи, – Леонардо в частных письмах иногда лукаво называл себя Леонардо Боначчи (Счастличик Леонардо).

Со временем имя и репутация Фибоначчи как гения головоломок были забыты, и лишь в 1836 году библиофил и знаменитый книжный вор Гильельмо Либри сложил два и два и додумался, что **Filius + Bonacci = Fibonacci**. Термины «число Фибоначчи» и «последовательность Фибоначчи» ввел в оборот французский математик Эдуард Люка в 1870 году.

Но вернемся к работе. Для объяснения десятичных чисел мы воспользуемся *экспоненциальной нотацией*. Слово «экспоненциальный» означает, что число умножается само на себя заданное число раз. Например,  $5^3$  означает, что 5 умножается на себя 3 раза, т. е. равно  $5 \times 5 \times 5 = 125$ . В выражении  $B^E$ , которое читается «В в степени Е», В называется *основанием*, а Е – *показателем степени*. Для любого числа N выражение  $N^1$  равно самому N. По соглашению,  $N^0$  равно 1 для любого N, кроме 0. Выражение  $0^0$  не определено, потому что различные способы его вычисления приводят к разным результатам.

Запись десятичного числа, или, иначе говоря, числа в системе счисления по основанию 10, например 3456, означает  $3 \times 1000 + 4 \times 100 + 5 \times 10 + 6 \times 1$ . В экспоненциальной нотации это то же самое, что  $3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$ . Читаем справа налево: младшая цифра, в данном случае 6, умножается на 1, следующая цифра, 5, умножается на 10, следующая на  $10^2$ , затем на  $10^3$  и т. д. Если бы цифр было 50, то старшая, самая левая, цифра была бы умножена на  $10^{49}$ .

Точно так же обстоит дело и в системах счисления по другому основанию. Например, в двоичной системе основание равно 2. Двоичное число 11001 вычисляется как  $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ , или  $16 + 8 + 0 + 0 + 1 = 25$ . В информатике часто используется основание 16, такая система счисления называется *шестнадцатеричной*. В такой системе цифрами являются 0123456789ABCDEF или 0123456789abcdef. Я предпочитаю заглавные буквы ABCDEF, потому что при этом все шестнадцатеричные цифры имеют одинаковую высоту, а так читать проще. Шестнадцатеричное число 9AB вычисляется как  $9 \times 16^2 + 10 \times 16^1 + 11 \times 16^0 = 9 \times 256 + 10 \times 16 + 11$ , т. е. 2475 в десятичной нотации.

Системы счисления по разным основаниям используются в криптографии, например, для преобразования текста в число. 26 букв латинского алфавита естественно ассоциируются с числами в системе по основанию 26:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Слово WORK можно было бы записать в виде  $22 \times 26^3 + 14 \times 26^2 + 17 \times 26 + 10 = 396\,588$ . С этим значением можно обращаться как с любым числом, в частности складывать, вычитать и умножать.

Большие числа можно записывать в экспоненциальной нотации, называемой также *научной*, например  $1.23 \times 10^7 = 12\,300\,000$ . Это то же самое, что взять число 1.23 и перенести десятичную точку на 7 позиций вправо.

## 3.5 Простые числа

Целые числа, большие 1, бывают *простыми* и *составными*. Если число является произведением двух меньших чисел, то оно называется составным, иначе простым. Перечислим первые составные числа:  $4 = 2 \times 2$ ,  $6 = 2 \times 3$ ,  $8 = 2 \times 4$  и  $9 = 3 \times 3$ . И первые простые числа: 2, 3, 5, 7, 11. Число 1 не является ни простым, ни составным.

Важное свойство простых чисел заключается в том, что любое целое число можно записать в виде произведения простых, причем единственным образом с точностью до порядка сомножителей. Например, поскольку  $30 = 2 \times 3 \times 5$ , никакое простое число, кроме 2, 3 и 5, не может быть делителем 30. Здесь 2, 3 и 5 называются *простыми множителями* 30. Множество простых множителей любого целого числа определено единственным образом. Его нахождение называется *разложением на простые множители*, или *факторизацией*.

Два целых числа  $A$  и  $B$ , не имеющих общих простых множителей, называются *взаимно простыми*. Например, 20 и 27 – взаимно простые числа. Если  $N$  – целое число, то  $N$  и 1 всегда взаимно просты, тогда как  $N$  и 0 взаимно просты, только когда  $N = 1$ . Числа  $N$  и  $N + 1$  всегда взаимно просты.

Когда целое положительное число  $A$ , называемое *делимым*, делится на целое положительное число  $B$ , называемое *делителем*, то в результате получается *частное* и *остаток*. Обозначим частное  $Q$ , а остаток  $R$ . Тогда  $Q$  определяется как наибольшее целое число такое, что  $QB$  не превосходит  $A$ , а остаток показывает, что осталось после деления, т. е.  $R = A - QB$ . Отметим, что  $0 \leq R < B$ . Например, пусть  $A = 40$  и  $B = 11$ . Наибольшее целое кратное 11, не превосходящее 40, равно 33, поэтому частное равно 3, т. е.  $3 \times 11 = 33$ . Остаток равен 7, потому что  $40 - 33 = 7$ .

## 3.6 Модульная арифметика

Изучение остатков от деления называется *модульной арифметикой*. Модульную арифметику разработал Карл Фридрих Гаусс в Гёттингенском университете в 1801 году. В модульной арифметике частное игнорируется, делитель называется *модулем*, а остаток – *вычетом*. В примере выше модуль равен 11, а вычет 7. Два числа  $X$  и  $Y$  с одинаковым вычетом называются *сравнимыми по модулю  $N$* . По-другому говорят, что  $X$  и  $Y$  принадлежат одному классу вычетов по модулю  $N$ .

Это записывается в виде  $X \equiv Y \pmod{N}$ . Например,  $40 \equiv 7 \pmod{11}$ , так что 40 и 7 принадлежат одному классу вычетов по модулю 11.  $X$  и  $Y$  сравнимы по модулю  $N$  тогда и только тогда, когда  $X - Y$  кратно  $N$ , т. е.  $X = Y + aN$  для некоторого целого  $a$ .

Для классов вычетов действуют те же арифметические правила, что для обычных целых чисел, например:

$$\begin{aligned} a+b &\equiv b+a \pmod{N} \text{ и } ab \equiv ba \pmod{N}, \\ a+0 &\equiv a \pmod{N}, a-a \equiv 0 \pmod{N} \text{ и } a \times 1 \equiv a \pmod{N}, \\ (a+b)+c &\equiv a+(b+c) \pmod{N} \text{ и } a(bc) \equiv (ab)c \pmod{N}, \\ a(b+c) &\equiv ab+ac \pmod{N} \text{ и } (a+b)c \equiv ac+bc \pmod{N}. \end{aligned}$$

Число  $-a$  называется *аддитивным обращением*  $a$ . Нотацию  $a - b$  можно рассматривать как сокращенную запись  $a + (-b)$ .

Ситуация с *мультипликативным обращением* сложнее. Для сравнения  $ax \equiv b \pmod{N}$  следует рассмотреть 3 случая: (1)  $a$  и  $N$  взаимно простые, (2)  $a$  и  $N$  имеют общий множитель  $d$ , который не делит  $b$ , (3)  $a$ ,  $b$  и  $N$  имеют общий множитель  $d$ .

- 1 Пусть  $a$  и  $N$  взаимно просты. Тогда существует единственный вычет  $a'$ , являющийся мультипликативным обращением  $a$  по модулю  $N$ , т. е.  $aa' \equiv 1 \pmod{N}$  и  $a'a \equiv 1 \pmod{N}$ . Если  $a'$  существует, то сравнение  $ax \equiv b \pmod{N}$  легко решается, и решение имеет вид  $x \equiv a'b \pmod{N}$ . В разделе 15.3.2 я опишу эффективные способы вычисления  $a'$  при больших  $N$ .
- 2 Если  $a$  и  $N$  имеют общий множитель  $d > 1$ , то мультипликативного обращения  $a$  по модулю  $N$  не существует. Не может существовать такого  $a'$ , что  $aa' \equiv 1 \pmod{N}$ . Если  $b$  не делится на  $d$ , то сравнение  $ax \equiv b \pmod{N}$  не имеет решений. Например, у сравнения  $4x \equiv 5 \pmod{12}$  нет решений.
- 3 Пусть  $d$  – наибольший общий делитель (НОД, англ. *gcd*)  $a$  и  $N$ , который мы будем обозначать  $\gcd(a, N)$ . Это значит, что  $d$  – наибольшее целое число, на которое делятся  $a$  и  $N$ . Если  $a$ ,  $b$  и  $N$  делятся на  $d$ , то сравнение можно сократить, разделив  $a$ ,  $b$  и  $N$  на  $d$ :  $(a/d)x \equiv (b/d) \pmod{N/d}$ .

Рассмотрим пример. Возьмем сравнение  $8x \equiv 4 \pmod{12}$ . Поделив все его элементы на 4, получаем сокращенное сравнение  $2x \equiv 1 \pmod{3}$ . Его решением является  $x \equiv 2 \pmod{3}$ , т. е.  $x$  может быть любым целым числом вида  $3n + 2$ . Вернемся к исходному сравнению;  $x$  – вычет по модулю 12, поэтому должен находиться в диапазоне от 0 до 11 включительно. Числа вида  $3n + 2$ , попадающие в этот диапазон, – 2, 5, 8 и 11. Это означает, что  $x$  может принимать любое из значений 2, 5, 8, 11. Поэтому сравнение  $8x \equiv 4 \pmod{12}$  имеет 4 решения.

Далее в этой книге **mod** используется как арифметический оператор. Выражение  $x \bmod y$ , где  $x$  – целое, а  $y$  – положительное целое число, означает остаток от деления  $x$  на  $y$ . Таким образом,  $27 \bmod 3 = 0$ ,  $27 \bmod 4 = 3$ , а  $27 \bmod 5 = 2$ .

# Инструментарий криптографа

---

## *Краткое содержание главы:*

- система оценивания шифров;
- подстановочные шифры;
- перестановочные шифры;
- фракционирование – разбиение букв на меньшие единицы;
- генераторы псевдослучайных чисел.

Шифры с секретным ключом строятся из нескольких базовых элементов. Их можно рассматривать как профессиональные инструменты. Чтобы построить стойкий шифр, нужно владеть всеми этими инструментами. Это не значит, что в каждом шифре будет использоваться каждый элемент. Такой подход только привел бы к усложнению без какого-либо повышения безопасности. Шифр стал бы работать медленнее, но никакого выигрыша мы не получили бы. В этой главе рассматриваются подстановки, перестановки, фракционирование и случайные числа. С другими инструментами мы познакомимся позже, например со сжатием текста – в главе 10, а со сцеплением блоков – в главе 11.

Прежде чем переходить к обсуждению элементов, поговорим о стойкости. Стойкость шифра измеряется в битах. Каждый бит представляет двоичный выбор. Если бы существовал такой шифр, что любой шифртекст представляет один из двух возможных открытых текстов, то стойкость этого шифра была бы равна 1 биту. Например:

0 = Мы проиграли.

1 = Мы выиграли.

Лимитирующим фактором при определении стойкости шифра является длина ключа. Если используются ключи длиной 64 бита, то стойкость шифра не может быть больше 64 бит, но для слабого шифра может оказаться меньше.



## 4.1 Система оценивания

Чтобы помочь вам составить общее представление о стойкости описанных в книге шифров, я буду оценивать их по десятибалльной шкале. Это мое личное мнение, основанное на собственном опыте и анализе того, сколько усилий нужно приложить, чтобы вскрыть шифр с применением лучших известных мне методов. Кроме того, я учитывал результаты сравнения шифров друг с другом и со старыми шифрами, которые удалось или, наоборот, не удалось вскрыть. Я буду знакомить вас с анализом, перед тем как приводить каждую оценку.

- Оценка 1 означает, что шифр может вскрыть даже новичок, не имеющий никакой подготовки, вооружившись карандашом и бумагой и приложив скромные усилия.
- Оценка 2 означает, что шифр может быть вскрыт опытным любителем, который не пользуется ничем, кроме карандаша и бумаги.
- Оценка 3 означает, что подготовленный криптограф-любитель может вскрыть шифр, применяя только ручные методы.
- Оценки 4 и 5 означают, что необходим компьютер, подготовленный криптограф или то и другое.
- Оценки от 6 до 9 показывают, какая вычислительная мощность была бы необходима противнику-специалисту.
- Оценка 10 означает, что шифр устоит против национального криптографического агентства, располагающего множеством профессиональных криптографов и самыми большими из современных суперкомпьютеров.

Иногда я выхожу за пределы этой шкалы. Оценка 0 означает, что в шифре можно разобраться даже без карандаша и бумаги. Примерами может служить молодежный жаргон<sup>1</sup> или текст **УШИП АВОЛС**

<sup>1</sup> «Поросычья латынь», аналог нашего олбанского. – Прим. перев.

**ДЕРЕПАН МОДАЗ.** Оценка 11 означает, что шифр нельзя будет вскрыть даже с применением гипотетического будущего ультракомпьютера, гораздо более мощного, чем квантовые компьютеры и суперкомпьютеры, которые мы можем вообразить сегодня.

Глядя на проставленные мной баллы, вы сможете уяснить, как оценивать другие шифры, которые в книге не упоминаются, или ваши собственные. Имейте в виду, что баллы – это всего лишь оценка, а не гарантия стойкости. Гарантии дает анализ, описанный в главе 12.

## 4.2 Подстановка

Первым инструментом в арсенале криптографа является подстановка. Одна единица текста заменяется другой. Единицами открытого текста могут быть буквы, пары букв или более длинные блоки. Единицами шифртекста могут быть буквы, блоки букв, блоки цифр или комбинации букв и цифр. Если все единицы – одиночные буквы, то шифр называется *простым подстановочным*, или *одноалфавитным*. В компьютерной криптографии единицами могут быть биты, байты или блоки битов либо байтов произвольной длины. В этом разделе приводится лишь краткий обзор. Более полное обсуждение см. в главах 5 и 6.

Один из самых старых подстановочных шифров – шифр Цезаря, который использовал и предположительно изобрел Юлий Цезарь. Каждая буква алфавита заменяется буквой, отстоящей от нее на 3 позиции. В современных условиях число позиций может быть любым, но фиксированным, и позиции могут отсчитываться как вперед, так и назад. Шифру Цезаря присвоена оценка 1.

Единицы открытого текста необязательно должны быть одинаковой длины. Допустим, что шифр принимает буквы алфавита и подставляет вместо них пары цифр. В алфавите 26 букв, а возможных пар цифр 100. Следовательно, 74 лишние пары криптограф может использовать для других целей. Так, на протяжении сотен лет, помимо замены отдельных букв, практиковалась замена часто встречающихся пар букв, таких как TH, ER, ON, AS и NT, и, возможно, коротких слов типа THE и AND. При таком подходе единицы открытого текста могут состоять из 1, 2 и 3 букв, в результате чего распределение частот пар цифр становится более равномерным. Различия в частотах встречаемости букв можно использовать для вскрытия шифра, поэтому чем равномернее их распределение, тем выше стойкость шифра.

Другой подход – использовать лишние пары для дополнительных подстановок часто встречающихся букв. Это называется *омофонической* подстановкой. Например, можно определить 10 подстановок для буквы E, 8 – для буквы T и т. д. Различные подстановки для одной буквы называются *омофонами*. Тут есть прямая аналогия с омофонами F и PH, которые в английском языке представляют один звук. Наличие нескольких вариантов подстановки делает распределение



частот пар цифр более равномерным. Естественно, оба подхода, пары букв и омофоническую подстановку, можно сочетать, чтобы получить еще более равномерное распределение пар цифр. Иначе говоря, эти методы препятствуют использованию анализа частот для вскрытия шифра.

### 4.2.1 Коды Хаффмана

При использовании компьютеров блоками шифртекста могут быть строки битов. Хороший пример дает кодирование Хаффмана, разработанное Дэвидом А. Хаффманом в 1952 году в бытность студентом MIT. Я не буду останавливаться на методе оптимизации множества кодов, а опишу лишь общую идею как пример двоичного кода переменной длины. В схеме Хаффмана часто встречающимся буквам сопоставляются короткие коды, а редко встречающимся – длинные. Для принятия решения используется таблица частот букв. В результате для кодирования сообщения требуется меньше битов. Это называется *сжатием текста*. В разделе 10.7 описываются более эффективные методы сжатия.

В английском языке самыми частыми буквами являются Е и Т, на каждую приходится примерно по 1/8 текста. Поскольку  $8 = 2^3$ , для представления Е и Т отведем по 3 бита. Эти 3-битовые значения можно выбрать произвольно, скажем Е = 100 и Т = 111. Буду называть этот метод «перемешанным методом Хаффмана». Следующими по частоте являются буквы А, О, I, N, S, R, H. На каждую из них приходится примерно по 1/16 текста, так что для их представления будем использовать по 4 бита. Можно взять любые 4-битовые коды, кроме начинающихся с уже используемых комбинаций 100 и 111. Следующая группа букв – D, L, U, C, M, F, Y, на каждую из них приходится примерно 1/32 текста, поэтому нужны 5-битовые коды. И так далее.

Ниже приведены перемешанные коды Хаффмана, созданные мной на основе частот букв в англоязычном тексте, содержащем 150 000 букв. Для других языков результаты будут иными. Коды Хаффмана обладают *префиксным свойством*, т. е. никакой код не является префиксом более длинного кода. Например, если **abcd** – код, то **abcde** не может быть кодом ни при каком выборе двоичных цифр а, b, c, d, e. Префиксное свойство впервые описал математик Эмиль Леон Пост в 1920 году.

E 100	D 00000	P 010010
T 111	L 01000	B 010011
A 0001	U 10110	V 110101
O 0010	C 10111	K 1101000
I 0011	M 11000	X 11010011
N 0101	F 11001	Q 110100101
S 0110	Y 11011	J 1101001000
R 0111	W 000010	Z 1101001001
H 1010	G 000011	



При таких кодовых группах слово STYLE кодируется как **0110 111 11011 01000 100**. Перегруппировав биты по четыре, получим **0110 1111 1011 0100 0100**, или в шестнадцатеричном виде **6FB44**.

Для Эмили почти невозможно идентифицировать кодовые группы отдельных букв в шифртексте, но она может поискать длинные повторяющиеся строки битов. Они представляют часто встречающиеся пары букв, называемые *биграммами*, тройки букв – *триграммы* – или слова. Например, любая 10-битовая строка должна в среднем встретиться один раз в любой последовательности  $2^{10} = 1024$  бит. Если 10-битовая строка встречается 20 или более раз в строках длиной 1024 бита, то почти наверняка она представляет слово THE, самое частое в английском языке. Идентифицировав слово THE в тексте, можно затем поискать его продолжения, например THERE или THESE; их легко найти благодаря повторению буквы E. Перемешанный код Хаффмана получает оценку 3.

## 4.3 Перестановка

Второй из важнейших криптографических инструментов – перестановка, т. е. изменение порядка символов в сообщении. Простейший метод называется *маршрутной перестановкой*. Буквы сообщения записываются в прямоугольник в одном порядке, а считываются в другом порядке. В этом разделе мы лишь кратко познакомимся с этим методом, а более полное описание отложим до главы 7.

Например, сообщение THERE IS NO LOVE AMONG THIEVES, состоящее из 25 букв, записывается в квадрат  $5 \times 5$  по строкам слева направо, а считывается по столбцам сверху вниз. Самый левый столбец содержит строку **TI00I**.

T H E R E	Открытый текст:	T H E R E I S N O L O V E A M O N G T H I E V E S
I S N O L		
O V E A M	Шифртекст:	T I 0 0 I H S V N E E N E G V R O A T E E L M H S
O N G T H		
I E V E S		

Среди маршрутов записи букв в сетку и чтения их из сетки чаще всего встречаются: по строкам (слева направо или справа налево), по столбцам (сверху вниз или снизу вверх), по строкам с чередованием направления, по столбцам с чередованием направления, по диагонали, начиная с любого угла, по диагонали с чередованием направления, по спирали (по часовой стрелке или против часовой стрелки). Маршрутные перестановочные шифры получают оценку 1.

## 4.4 Фракционирование

Фракционированием называется разбиение символов на меньшие единицы. Мы уже видели один способ представления символа двоичным числом. С каждым битом этого числа можно производить операции как с отдельной единицей, применяя к ним подстановки и перестановки. В этом разделе дается краткое введение во фракционирование. Более подробное обсуждение см. в главах 9 и 10.

Классический способ представить букву двумя цифрами – квадрат Полибия, придуманный во втором веке до нашей эры греческим историком Полибием. Ниже показан квадрат  $5 \times 5$ , в котором используется смешанный алфавит с ключевым словом SAMPLE. Обратите внимание, что буквы I и J занимают одну клетку, чтобы 26-буквенный алфавит уместился в квадрат с 25 клетками.

	1	2	3	4	5
1	U	V	W	X	Y
2	Z	S	A	M	P
3	L	E	B	C	D
4	F	G	H	IJ	K
5	N	O	Q	R	T

Квадрат Полибия, перемешанный  
ключевым словом SAMPLE

Поскольку буква A находится на пересечении строки 2 со столбцом 3, она представлена парой цифр 23. Аналогично B соответствует 33, C – 34 и т. д. вплоть до Z, представленной парой 21. Обе буквы I и J представлены парой 44. К этим цифрам можно затем применять подстановки, перестановки и перегруппировки. Пары цифр можно преобразовать обратно в буквы, используя эту сетку или другой квадрат Полибия, построенный в ином порядке.

В современных реалиях каждый символ нужно было бы заменить его шестнадцатеричным представлением в кодировке ASCII или UTF-8. Так, A = 41, B = 42, C = 43, ..., Z = 5A. К этим шестнадцатеричным цифрам можно точно так же применять подстановки, перестановки, перегруппировки и в конечном итоге преобразовать снова в байты.

Любопытный пример фракционированного кода Морзе придумал М. Э. Охавер (М. Е. Ohaver) в 1910 году. Охавер всегда именовал себя М.Е., потому что не любил свое первое имя, Мерль.

### Историческая справка

В сноске на странице 241 книги Craig Bauer «Secret History: The Story of Cryptology» утверждается, что имя М. Е. Ohaver было одним из псевдонимов плодотворного автора бульварных романов Кенделла Фостера Кроссена. Это неправда. Кроссен иногда подписывался псевдонимом М. Е. Cha-ber – от слова *mechaber*, מכבר, которое на иврите означает «автор».

Во фракционированном коде Морзе буквы собираются в группы фиксированного размера, скажем 7, и заменяются их эквивалентами в коде Морзе, а в качестве разделителя используется знак /. Затем порядок длин кодовых групп меняется на противоположный и группы нового размера преобразуются назад в буквы.

Е	Х	А	М	Р	Л	Е	Открытый текст
./----	/./--/----	/./--/----	/./--/----	/./--/----	/./--/----	/./--/----	Эквивалентные коды Морзе
1	4	2	2	4	4	1	Длины кодовых групп
1	4	4	4	2	4	1	Длины в обратном порядке
./----	/./--/----	/./--/----	/./--/----	/./--/----	/./--/----	/./--/----	Перегруппированные коды Морзе
Е	Х	Ј	А	Н	Л	Е	Эквивалентные буквы

Код Морзе изобрел Альфред Вайль в 1840 году и назвал его по имени своего работодателя Самуэля Ф. Б. Морзе.

Слабости этого шифра очевидны. Поскольку в нем используется стандартная азбука Морзе, единственным ключом является длина группы букв, которую можно подобрать за несколько попыток. Буквы открытого текста часто заменяются ими же. Всего существует 30 разных кодовых групп Морзе, но букв только 26, так что нужны еще четыре символа. Охавер взял немецкие буквы ä, ё, ö и ü. Фракционированный код Морзе получает оценку 1.

Частично упомянутые проблемы можно решить, внося два изменения: (1) использовать только группы Морзе с длинами 1, 3 и 4. Таких групп 26, ровно столько, сколько букв в латинском алфавите; (2) перемешать алфавит или, что эквивалентно, перемешать кодовые группы. Я называю эту улучшенную версию *ФР-акционированным кодом Морзе*. Например, если воспользоваться для перемешивания алфавита ключевым словом MIXEDALPHBT и сохранить стандартный порядок групп Морзе, то получится следующее:

М .	Р ---	У ....	Н ----
І -	Н ---	Z ....	О ----
Х ...	В ---	С ....	Q ----
Е ...	Т ....	F ....	Р ----
Д ...	U ....	Г ....	S ----
А ---	V ....	Ј ....	
Л ---	W ....	К ....	

Но даже с этими улучшениями ФР-акционированный код Морзе получает только оценку 2.

## 4.5 Генераторы случайных чисел

Генератором случайных чисел может быть что угодно, порождающее последовательность чисел в заданном диапазоне. Числами мо-

гут быть одиночные биты, 8-битовые байты, десятичные цифры или числа из любого другого диапазона. Например, для криптографических целей полезны числа в диапазоне от 0 до 25, соответствующие 26 буквам алфавита. В этом разделе приводится только введение в тему. Полное обсуждение см. в главе 13.

Важно понимать, что нет такой вещи, как «случайное число». Нельзя сказать, что 51 – случайное число, а 52 – не случайное, или наоборот. Но можно сказать, что последовательность 51, 52, 53, 54, ... не случайна, т. к. она вполне предсказуема. Случайность – свойство последовательности или генератора, но не отдельных членов последовательности. Правильнее говорить «случайная последовательность чисел», а не «последовательность случайных чисел».

Генератором может быть физический процесс, например космические лучи, отсчеты счетчика Гейгера, точное время нажатий клавиш на клавиатуре, флаг, развевающийся на ветру, брызги волн или люди, спешащие на электричку. Физические источники в большинстве своем недостаточно быстры для криптографических целей, однако последовательность чисел можно сохранить в компьютерном файле для последующего использования.

Генератором может быть также математическая функция или компьютерная программа, которая при каждом вызове порождает число. Случайные числа, порождаемые математическими алгоритмами, называются *псевдослучайными*, чтобы отличить их от *истинно случайных* чисел. Они считаются слабее истинно случайных чисел, потому что противник, знающий часть случайной последовательности, теоретически может вычислить предыдущие и последующие числа и таким образом прочесть сообщение. Истинно случайные числа никогда не порождаются математической функцией. В разделе 13.8 я представлю методы генерирования криптографически безопасных последовательностей псевдослучайных чисел, спроектированные так, чтобы не дать противнику продолжить участок последовательности.

Одним из главных различий между последовательностями псевдослучайных и истинно случайных чисел является то, что псевдослучайные последовательности рано или поздно повторяются, тогда как истинно случайные не повторяются никогда. Количество членов последовательности, после которого начинается повторение, называется *периодом*. Например, период последовательности 3,1,9,2,4, 3,1,9,2,4, 3,1,9,2,4, 3, ... равен 5, повторяющаяся часть подчеркнута. В общем случае чем период длиннее, то более стойким является шифр.

Из того, что последовательность чисел случайна, еще не следует, что все числа равновероятны. Например, предположим, что мы записываем цвета автомобилей, едущих по мосту с оживленным движением. Цвета случайны, но некоторые встречаются чаще других. Белые, черные, серебристые и красные машины встречаются гораздо чаще оранжевых, цвета фуксии или зеленовато-желтых. Анало-

гично, при игре в кости (при условии что кости геометрически правильные) результат каждого броска случаен, но выпадение суммы 7 в шесть раз вероятнее, чем суммы 12.

В разделах 13.14.1 и 13.14.2 мы обсудим, как, воспользовавшись случайностью таких последовательностей, получить последовательности, в которых все числа встречаются практически с одинаковой вероятностью. Поэтому в дальнейшем я буду предполагать, что любой генератор случайных чисел порождает равновероятные числа. Если генератор хороший, то пары, тройки и т. д. сгенерированных чисел тоже будут распределены равномерно.

### 4.5.1 Цепной генератор цифр

В заключение этого раздела я хотел бы показать простой генератор псевдослучайных чисел, который легко реализовать с помощью карандаша и бумаги. Компьютер вообще не нужен. Назовем его цепным генератором цифр. Сначала выпишем какое-нибудь 7-значное десятичное число. Эти 7 цифр будем называть *начальным значением*, или *вектором инициализации*. Можно считать, что это ключ или часть ключа шифра, в котором используется данный генератор. Для получения первой псевдослучайной цифры сложим первую и последнюю цифры. Допишем новую цифру в конец последовательности, а первую цифру закрасим. Так, если вначале была последовательность 3920516, то мы дописываем цифру  $3+6 = 9$ .

$$\begin{array}{ccccccc} 3 & 9 & 2 & 0 & 5 & 1 & 6 \\ & & & & & & \downarrow \\ & & & & & & \oplus \\ & & & & & & \downarrow \\ \text{3} & 9 & 2 & 0 & 5 & 1 & 6 & 9 \end{array}$$

Если сумма цифр превышает 9, то оставляем только разряд единиц. То есть сложение производится по модулю 10. Для получения второй псевдослучайной цифры процесс повторяется. В данном случае  $9+9 = 18$ . Отбрасываем разряд десятков и получаем цифру 8.

$$\begin{array}{ccccccc} \text{3} & 9 & 2 & 0 & 5 & 1 & 6 & 9 \\ & & & & & & \downarrow \\ & & & & & & \oplus \\ & & & & & & \downarrow \\ \text{3} & 9 & 2 & 0 & 5 & 1 & 6 & 9 & 8 \end{array}$$

Этот процесс можно повторять для получения любого желаемого количества псевдослучайных десятичных цифр.

**3920516980056219 9940232...**

В результате имеем псевдослучайную последовательность **9800562199940232...**

Заметим, что если все цифры начального числа четные, то и все сгенерированные цифры будут четными. Аналогично, если все начальные цифры делятся на 5, т. е. равны 0 или 5, то все сгенерированные цифры будут делиться на 5. В этом случае период не может быть больше 128, потому что начальное значение было 7-значным, а различных комбинаций из семи цифр 0 и 5 существует всего  $2^7 = 128$ . Поскольку такие начальные значения не могут порождать последовательности с длинным периодом, они называются *неподходящими*. Для цепного генератора цифр *подходящее* начальное значение должно содержать по меньшей мере одну нечетную цифру и одну цифру, не делящуюся на 5. Например, начальное значение 2222225 подходящее, а 2222222 и 5555555 неподходящие. При подходящем начальном 7-значном значении период всегда равен 2 480 437.

Это пример типичного кустарного генератора псевдослучайных чисел. Существует  $10^7$  возможных 7-значных начальных значений. Если начать с произвольного начального значения, то генератор будет порождать какую-то последовательность, пока снова не встретится данное значение, после чего последовательность будет циклически повторяться. Поэтому множество всех 7-значных чисел распадается на несколько дискретных циклов, каждый со своим периодом. Если выбрать подходящее начальное значение, то цикл всегда будет иметь максимально возможный период 2 480 437. Существует 4 разных цикла такой длины плюс несколько гораздо более коротких циклов, порождаемых неподходящими начальными значениями.

Для начальных значений другой длины поведение аналогично. Даже когда максимальный цикл очень короткий, все равно вероятность получить максимальный цикл часто бывает высокой, потому что максимальных циклов может быть много. В следующей таблице приведены вероятности получить цикл заданной длины при выборе подходящего начального значения.

Число цифр	Период	Вероятность
4	1560	100 %
5	168	86,7 %
6	196 812	99,974 %
7	2 480 437	100 %
8	15 624	98,817 %
9	28 515 260	79,999 %
	не менее 2 851 526	99,9988 %
10	1 736 327 236	86,9 %
	не менее 248 046 748	99,31 %
	не менее 13 671 868	100 %

Из таблицы видно, что начальные значения длиной 5 и 8 небезопасны, поскольку доля очень коротких циклов высока. Наилучший результат дают начальные значения длиной 7 и 10, потому что для них длинный период гарантирован.

Этот пример генератора случайных чисел приведен только для демонстрации с целью показать, чего можно достичь с помощью простых ручных методов. Для задач, требующих высокого уровня безопасности, такой генератор не годится.

## 4.6 Полезные комбинации, бесполезные комбинации

Описанные в этой главе четыре метода можно комбинировать многочисленными способами, которые я и буду исследовать далее в этой книге. Но важно с самого начала понять, что не каждая комбинация приносит пользу. Некоторые лишь увеличивают объем работы, не повышая стойкость шифра.

Рассмотрим идею, которая привлекает некоторых начинающих: выполнить простую подстановку для сообщения, затем вторую простую подстановку для результата, затем еще одну и так далее на протяжении, скажем, 5, 10 или даже 100 раундов. Это бесполезная трата времени. Две простые подстановки эквивалентны одной, только с другим перемешанным алфавитом, поэтому выполнение большого числа простых подстановок не повышает стойкость. Приведу иллюстрацию. Возьмем две подстановки с ключами FIRST и SE-COND. Их выполнение друг за другом эквивалентно выполнению третьей подстановки.

ABCDEFGHIJKLMNQRSTUWXYZ XYZFIRSTABCDEFGHIJKLMNPQVW	Первая подстановка
ABCDEFGHIJKLMNQRSTUWXYZ UVWXYZSECONDAFBGHIJKLMPQRT	Вторая подстановка
ABCDEFGHIJKLMNQRSTUWXYZ QRTZCIJKUVWXYZSEONDAFBGHLMP	Эквивалентная

Проверим на примере. Зашифровав слово **EXAMPLE** с помощью первой подстановки, получим **IUXEJDI**. Зашифровав **IUXEJDI** с помощью второй подстановки, получим **CLQY0XC**. Можете сами проверить, что в результате шифрования **EXAMPLE** с помощью эквивалентной подстановки тоже получается **CLQY0XC**.

Выполнение двух операций шифрования друг за другом называется *композицией*. Рассмотренный выше пример показывает, что композиция двух простых подстановок сама является простой подстановкой. Если результатом первого шифрования является какой-то код, то последующее применение к нему шифра называется *перешифрованием*, или *многократным шифрованием*. Самой распространенной формой перешифрования является сложение без переноса, или сложение по модулю 10, которое работает следующим образом:

<b>12155</b>	<b>12155</b>	<b>12155</b>	<b>12155</b>	Ключ перешифрования
<b>61587</b>	<b>02954</b>	<b>70069</b>	<b>53028</b>	Кодовые группы открытого текста
<b>73632</b>	<b>14009</b>	<b>82114</b>	<b>65173</b>	Перешифрованные кодовые группы

### 4.6.1 Шифр Базери типа 4

Рассмотрим противоположный случай – шифр, в котором выполнение шага перестановки после шага подстановки порождает гораздо более стойкий шифр.

Этот шифр был предложен блестящим, запальчивым и бранчливым французским криптографом Этьеном Базери в 1898 году. Я не знаю, какое название сам Базери дал своему шифру. Я назвал его шифром Базери типа 4, потому что это последний из четырех шифров, предложенных вниманию шифровального отделения на протяжении 1890-х годов. Шифрование легко выполняется вручную.

Шифр Базери типа 4 состоит из простой подстановки, за которой следует простая перестановка. Такую операцию я называю *кусочным обращением*. Перестановка меняет порядок букв в коротких участках текста на противоположный согласно ключу, представляющему собой последовательность малых целых чисел. В примере ниже ключевое слово BAZERIS используется для перемешивания подстановочного алфавита, а ключ 4, 2, 3 – для перестановки.

<b>ABCDEFGHIJKLMN OPQRSTUVWXYZ</b>	Алфавит открытого текста
<b>HGFDCBAZERISYXWVUTQPONMLKJ</b>	Алфавит шифртекста
<b>THEQUICKBROWNFOXJUMPSOVERTHELAZYDOG</b>	Открытое сообщение
<b>PZCUOEFIGTWMXBWLROYVQWNCPTZCSHJKDWA</b>	После подстановки
<b>4 2 3 4 2 3 4 2 3 4 2 3</b>	Ключ перестановки
<b>PZCU OE FIG TWMX BW LRO YVQW NC TPZ CSHJ KD WA</b>	
<b>UCZP EO GIF XMWT WB ORL WQVY CN ZPT JHSC DK AW</b>	Конечный шифртекст
<b>UCZPE OGIFX MWTWB ORLWQ VYCNZ PTJHS CDKAW</b>	Шифртекст, сгруппированный по 5

Перестановку такого типа можно использовать для укрепления шифров разных типов, поэтому она заслуживает отдельного названия. Ее можно укрепить, оставив несколько участков текста в исходном порядке. Для определения таких участков можно, например, задавать отрицательный ключ. В следующем примере используется числовой ключ 3, 4, -3, 2. Отметим, что он эквивалентен ключу 3, 4, 1, 1, 2.

<b>3 4 -3 2 3 4 -3 2 3 4 -3 2</b>	Ключ перестановки
<b>THE QUIC KBR OW NFO XJUM PSO VE RTH ELAZ YDO GS</b>	
<b>ENT CIUQ KBR WO OFN MUJX PSO EV HTR ZALE YDO SG</b>	Конечный шифртекст



Криптографы из шифровального отделения не смогли вскрыть ни одно из сообщений, предложенных Базери. Несмотря на все усилия, сообщения оставались нерасшифрованными 40 лет, пока знаменитый архитектор и криптограф-любитель Розарио Кандела не вскрыл шифр и не написал книгу о том, как он это сделал («The Military Cipher of Commandant Bazeries», Cardanus Press: New York, 1938).

Однако Кандела не сумел дешифровать сообщения непосредственно. Вместо этого он нашел и воспользовался слабостью в использованном Базери способе генерирования подстановочного алфавита по ключу. Если бы Базери применил более стойкий метод перемешивания алфавита, то Кандела не смог бы дешифровать сообщения. Таким образом, шифр Базери типа 4 с хорошо перемешанным алфавитом получает оценку 5. Неплохо для комбинации двух методов, получивших оценку 1.

#### Историческая справка

Кандела окончил Колумбийскую школу архитектуры, поэтому планировал напечатать книгу в издательстве Колумбийского университета. Но Вильям Ф. Фридман, тогдашний глава американских криптологов, прознал об этом и тайно воспрепятствовал публикации. Это еще одно свидетельство в пользу стойкости шифра Базери типа 4.

# Подстановочные шифры

---

## *Краткое содержание главы:*

- простые и многоалфавитные подстановочные шифры;
- вскрытие многоалфавитных шифров с помощью метода Касиски и индекса совпадения;
- шифры с автоключом и с бегущим ключом и методы их вскрытия;
- моделирование роторных шифровальных машин.

Теперь мы готовы к углубленному исследованию основных инструментов, описанных в предыдущей главе. Прежде чем переходить к описанию разнообразных шифров, я хочу явно сформулировать цели, которых эти шифры пытаются достичь. Голландский лингвист и человек энциклопедических познаний, Огюст Керкгоффс, впервые назвал эти цели в двух статьях, опубликованных в журнале «Journal des Sciences Militaires» в 1883 году.

- 1 Шифр должен быть невскрываемым если не в теории, то на практике.
- 2 Это должно оставаться верным, даже если система попала в руки врага.
- 3 Ключ должен быть легко запоминаемым (без бумажных записей) и легко изменяемым.

- 4 Зашифрованные сообщения должны быть пригодны для передачи по телеграфу.
- 5 Оборудование и документы должны быть легко переносимы, для работы с системой должно быть достаточно одного человека.
- 6 Шифр должен быть прост в использовании, не требуя сложных правил или вычислений.

Правило 4 можно изменить: «Зашифрованные сообщения должны быть пригодны для передачи в цифровом виде», а в остальном эти принципы сохраняют свою актуальность.

Из второго принципа следует, что стойкость шифра должна определяться только ключом. Керкгоффс также считал, что лишь криптографы обладают необходимой квалификацией, чтобы судить о стойкости шифра. Слишком часто решение о том, какой шифр использовать, принимается чиновниками, не имеющими опыта в криптографии, что иногда приводит к катастрофическим последствиям.

## 5.1 Простая подстановка

Простая подстановка, называемая еще *одноалфавитной*, – хорошо знакомый тип шифра, который часто встречается в отделах гололомоков в газетах и журналах. Каждая буква алфавита заменяется другой буквой, причем заменяющая буква всегда одна и та же. Так, если буква М была заменена буквой Т в одном месте, то любая буква М в сообщении будет заменена буквой Т, и каждая Т в шифртексте будет представлять М.

Поскольку большинство знакомо с методами вскрытия простых подстановочных шифров, я лишь кратко упомяну их: частота букв, частота начальных букв, частота конечных букв, частота двойных букв, частота пар букв, короткие слова, общие префиксы и суффиксы, распределение гласных и согласных, слова с характерной структурой, использование знаков препинания.

Для простых криптограмм, публикуемых в газетах, достаточно лишь взглянуть на короткие слова. Если встретились АВ и СВА, то, скорее всего, они соответствуют словам TO и NOT, тогда как АВ и ВАС чаще всего соответствуют OF и FOR. Строка АВСА – это, вероятно, THAT, ABCDC – THERE, а ABCDB – WHICH. В интернете можно купить списки слов с характерной структурой, и есть сайты, которые ищут слова по заданному вами образцу.

Для более сложных криптограмм, например текстов вроде **FOXU PIXU MANX AXED TOXIC LUXURY ONYX SPHINX**, необходим более методичный подход.

Я продемонстрирую процедуру на примере показанной ниже криптограммы. Известно, что язык английский. В тексте 73 буквы и 11 слов со средней длиной слова 6.64 буквы, а не 5.0, как в обыч-

ном англоязычном тексте. Отсутствие коротких слов длиной меньше 5 букв и слов с характерной структурой наводит на мысль, что криптограмма намеренно сделана трудной.

**RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS  
KSYIWZ, ZLORUS LOBZQC SQXMV**

Первый шаг – идентифицировать гласные и согласные. Начнем с подсчета числа вхождений каждой буквы алфавита, чтобы получить их *частоты*. В нашей криптограмме нет букв А, три буквы В, три буквы С и т. д. Полный перечень частот представлен ниже:

A-0	B-3	C-3	D-2	E-4	F-0	G-3	H-0	I-3
J-0	K-4	L-8	M-2	N-0	O-5	P-0	Q-4	R-4
S-7	T-0	U-3	V-5	W-3	X-3	Y-4	Z-3	

Заметим, что есть всего две высокочастотные буквы, **L** и **S**, их частоты равны соответственно 8 и 7. В обычном англоязычном тексте примерно 40 % букв – гласные. Эта доля стабильна и, как правило, поддерживается даже тогда, когда частоты букв подвергались сознательным манипуляциям. В криптограмме, содержащей 73 буквы, должно быть 29 гласных. Отсюда следует, что **L** и **S** – скорее всего, гласные, если только частотность букв не была значительно искажена.

Далее составим схему контактов, для чего выпишем буквы алфавита сверху вниз по центру страницы. Каждой букве, встречающейся в криптограмме, отведем строку. В каждой строке буквы, непосредственно предшествующие центральной, записываются слева от нее, а непосредственно следующие за ней – справа. Например, первая строка схемы, **EO B DWZ**, означает, что букве **B** в криптограмме по одному разу предшествуют **E** и **O**, а после нее встречаются **D**, **W** и **Z**, тоже по разу. Ниже показана схема контактов целиком (в 3 столбца, чтобы сэкономить место).

<b>EO B DWZ</b>	<b>UYKVMQZ L EEEEE000</b>	<b>GDKIM V LSMQ</b>
<b>LMQ C R</b>	<b>VX M CV</b>	<b>BOI W LUZ</b>
<b>XB D OV</b>	<b>DQLLL O RWRB</b>	<b>YGQ X DGM</b>
<b>LLLL E YYBK</b>	<b>VGZS Q OLCX</b>	<b>EES Y SLXI</b>
<b>IX G VXQ</b>	<b>COO R USU</b>	<b>WB Z LQ</b>
<b>KY I VGW</b>	<b>YRVUKU S YQ</b>	
<b>E K LVIS</b>	<b>RWR U LSS</b>	

Схема контактов нужна для идентификации гласных и согласных. В общем случае гласные контактируют с более широким множеством букв слева и справа, тогда как у согласных обычно ограниченное количество различных контактов. Построенная схема контактов позволяет идентифицировать 4 вероятные гласные, **L**, **Q**, **S** и **V**, и 4 вероятные согласные, **E**, **O**, **R** и **U**. Обозначим о гласные буквы, а × согласные и посмотрим, правдоподобно ли получившееся распределение.

```

RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS
xxox-o -o-xo -ox---x -oox-oo -oox-o-- -ooxx -----ox-xo

KSYIWZ, ZLORUS LOBZQC SQXMV
-o----- -oxxxx ox---- o-----

```

Мы видим 3 длинных участка без единой гласной: в третьем слове, **EYXDO**, в седьмом слове, **KIGXG**, и в восьмом слове, **YIWZ**. Это позволяет предположить, что **I** и (или) **X** может быть гласной. У обеих много разных контактов, но крайне маловероятно, что **X** представляет гласную, потому что тогда слово 11 начиналось бы тремя гласными и заканчивалось гласной. В английском языке такое бывает редко. Единственный известный мне пример – OUIJA. (**SQXMV** не может совпадать с AIOLI из-за повторяющейся I.) Поэтому **I** – скорее всего, гласная, а **X** – согласная.

Перейдем ко второму этапу разделения гласных и согласных. В шифртексте имеется пять пар предположительных гласных, **VL**, **VS**, **VQ**, **QL** и **SQ**. Пары гласных – нечастое явление в английском языке, поэтому вполне вероятно, что либо **V**, либо **Q** действительно согласная. Скорее всего, это не **Q**, потому что тогда слово 10 заканчивалось бы пятью согласными. Я знаю только одно такое 6-буквенное слово – ANGSTS.

Допустим, что **I** – гласная, **V** и **X** – согласные, и посмотрим, что при этом получается.

```

RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS
xxox-o -o-xo -ox-x-x -ox--xo --ox-x-- oxox -o-x-ox-xo

KSYIWZ, ZLORUS LOBZQC SQXMV
-o----- -oxxxx ox---- ox-x-

```

Похоже на правду. Теперь в слове 3 можно идентифицировать **D** как гласную, а из слова 11 следует, что **M** – вероятно, согласная. Итак, мы нашли все шесть гласных, поэтому все остальные буквы должны быть согласными. Ниже приведен окончательный результат разбиения на гласные и согласные.

```

RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS
xxxxxo xxxo xxxxxx xxoxoxo xxxxxxo oxox xxxxxxoxo

KSYIWZ, ZLORUS LOBZQC SQXMV
xxoxxx, xxxxo xxxox oox-

```

```

xx B  ox  xxxxxx L xxxxxxxx  xoxo V  ooo
ooo C  x          xx M  xx          xox W  ox
xx D  xx          ooooo O  xxxx          xox X  ox
ooo E  xxx      xxxo Q  xxx          xox Y  oox
ox G  xo          xxx R  ox          xx Z  oo
xx I  xxx      xxxxxx S  x
x K  oxoo          xxx U  ooo

```

Второй шаг – идентификация отдельных букв. В таблице контактов букв ниже показаны контактные характеристики букв в английском языке. У других языков характеристики иные. Например, для букв M, V и Z чаще всего с обеих сторон находятся гласные, а букве N обычно предшествует гласная, а за ней следует согласная. Я построил эту таблицу, воспользовавшись корпусом англоязычных текстов из проекта Гутенберга.

Таблица контактов букв

До \ После	Гласные	Те и другие	Согласные
Гласные	MVZ	RX	N
Те и другие	BJQW	CDFGLPST	
Согласные	H	Y	AEIOU

В этой криптограмме есть хороший кандидат на роль буквы N в открытом тесте – буква U в шифртексте. Однако U дважды встречается в биграмме **US** в конце слов, так что это маловероятно. Есть два хороших кандидата на роль N – буквы **E** и **O** в шифртексте. Та и другая представляют согласные, которым всегда предшествуют гласные и за которыми следуют согласные. Однако **O** более вероятна, потому что у нее выше частота и потому что ей предшествуют две известные согласные в слове **ZLORUS**. В английском языке сочетания трех согласных часто начинаются с буквы N, например **NST** или **NTH**. Скорее всего, **O** представляет N. Тогда получаем:

RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS  
 xx°xx° xx°xx° x°xxx°N xx°xx°x° xx°xxx°x° o°x°N° x°xxx°o°Nxx°

KSYIWZ, ZLORUS LOBZQC SQXMV  
 xx°xxx°, x°Nxx° o°Nxx°x° o°x°x°

Теперь можно попытаться найти слова, соответствующие этим образцам. Я нашел 67, отвечающих образцу **x°Nxx°** (**ZLORUS**). Из них 32 оканчиваются на E, а 27 – на Y. Я нашел 37 слов, отвечающих образцу **o°x°x°** (**SQXMV**) и не содержащих N. Из них 15 начинаются с Y, но только 1 начинается с E. Поэтому весьма вероятно, что букве **S** в шифртексте соответствует Y в открытом тексте. Это дает:

RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS  
 xx°xx°Y° xx°xx°Y° x°xxx°N° xx°xx°x°Y° xx°xxx°x° o°x°N° x°xxx°o°Nxx°Y°

KSYIWZ, ZLORUS LOBZQC SQXMV  
 x°Y°xxx°, x°Nxx°Y° o°Nxx°x° Y°x°x°

Слова шифртекста **IVQOR** и **ZLORUS** содержат биграмму **OR**. Попробуем идентифицировать ее. Мы знаем, что слово **IVQOR**, имеющее вид **o°x°N°**, не содержит буквы Y; это оставляет всего 24 вероятных слова.

Из них 12 оканчиваются на G, а 8 на S, поэтому шифртекст **OR** – скорее всего, либо NG, либо NS. Количество вероятных слов открытого текста для слова **ZLORUS**, имеющего вид  $\times \circ N \times \times Y$ , теперь сократилось до 26. Среди них четвертая буква в 8 случаях равна G, в 6 случаях T и всего в одном случае S. Поэтому самая вероятная расшифровка шифртекста **OR** – открытый текст NG. Теперь мы имеем:

**RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS**  
**G** $\times \circ \times \times Y$   $\times \circ \times GY$   $\times \circ \times \times \times \circ N$   $\times \times \circ \times \times \circ \times Y$   $\times \times \circ \times \times \times \circ \times$   $\circ \times \circ NG$   $\times \circ \times \times \times \circ N \times \times Y$

**KSUIWZ, ZLORUS LOBZQC SQXMV**  
 $\times Y \circ \times \times \times$ ,  $\times \circ NG \times Y$   $\circ N \times \times \circ \times$   $Y \circ \times \circ \times$

Для первого слова, **RULEYS**, есть 8 возможностей. Среди них только 6 вариантов расшифровки биграммы **UL**. Для каждого из них рассмотрим, что получается для слова **ZLORUS**.

<u>UL</u>	<u>RULEYS</u>	<u>ZLORUS</u>
HO	GHOSTY	нет
LU	GLUMPY	JUNGLY
NA	GNARLY	нет
RI	GRIMLY	нет
	GRISLY	нет
RO	GROSZY	нет
	GROWLY	нет
RU	GRUMPY	HUNGRY

Вариант **ZLORUS**=JUNGLY можно исключить, потому что тогда слово **KSUIWZ** имело бы вид  $\times Y \circ \times \times J$ . Такого слова в английском языке нет. Следовательно, **RULEYS** расшифровывается как GRUMPY, а **ZLORUS** – как HUNGRY. Вставив новые буквы, получаем:

**RULEYS YLCRS KLEYXDO GVLEBDVS BWLEKVMC IVQOR KIGXGQLOWUS**  
**GRUMPY PU** $\times GY$   $\times UMP \times \circ N$   $\times \times UM \times \circ \times Y$   $\times \times UM \times \times \circ \times$   $\circ \times \circ NG$   $\times \circ \times \times \times \circ UN \times RY$

**KSUIWZ, ZLORUS LOBZQC SQXMV**  
 $\times Y P \circ \times H$ , **HUNGRY UN** $\times H \circ \times$   $Y \circ \times \circ \times$

Остальные буквы можно заполнить, просто подбирая слова зрительно. Очевидно, что второе слово – PUDGY, а третье – BUMPKIN. Тогда восьмое слово – BYPATH и т. д. Целиком криптограмма расшифровывается как GRUMPY PUDGY BUMPKIN CLUMSILY STUMBLLED ALONG BACKCOUNTRY BYPATH, HUNGRY UNSHOD YOKEL.

Простой подстановочный шифр получает оценку 1. Но если частоты букв и контактов намеренно искажены, как в рассмотренном примере, то оценка может быть повышена до 2 и даже до 3. Хорошо для головоломок, но бесполезно для передачи данных в общем случае.

## 5.2 Перемешивание алфавита

Для простых подстановок нужен перемешанный алфавит. Есть несколько традиционных методов перемешивания, для которых хватает карандаша и бумаги. Один из них – воспользоваться ключевым словом. В простейшем случае мы просто записываем ключевое слово, начиная с какой-то позиции, а затем дописываем остаток алфавита после него, возвращаясь к началу по достижении конца. Оставшиеся буквы можно записывать в прямом или в обратном порядке. Вот три примера:

ABCDEFGHIJKLMN <strong>OP</strong> QRSTUVWXYZ	Открытый текст
UVWXYZ <u>SAMPLE</u> BCDFGHIJKNOQRT	Шифртекст
ABCDEFGHIJKLMN <strong>OP</strong> QRSTUVWXYZ	Открытый текст
HGFDCBS <u>SAMPLE</u> EZYXWVUTRQONKJI	Шифртекст
ABCDEFGHIJKLMN <strong>OP</strong> QRSTUVWXYZ	Открытый текст
XYZBCDS <u>SAMPLE</u> FGHIJKNOQRTUVW	Шифртекст

Впервые этот метод был использован семьей Ардженти, члены которой служили секретарями-шифровальщиками у нескольких пап и епископов где-то в 1600 году. Полученные таким образом алфавиты были недостаточно хорошо перемешаны. В улучшенном методе используются два ключевых слова, например:

KLMNOPQUVWXYZ <strong>FIRST</strong> ABCDEFGHIJ	Открытый текст
HJFBASECONDZYXWVUTRQPM <strong>LKJI</strong>	Шифртекст

### Столбцовое перемешивание

Разместить алфавит в прямоугольном блоке. Записать ключевое слово в верхней строке, а остаток алфавита пусть занимает столько строк, сколько необходимо. Чем длиннее слово, тем лучше перемешивание. Затем прочитать буквы в блоке по столбцам. В примере ниже ключевое слово SAMPLE записано в верхней строке. Первый столбец, прочитанный сверху вниз, дает **SBIRY**, второй столбец – **ACJ TZ** и т. д. Если хотите, можете чередовать чтение сверху вниз и снизу вверх или использовать другие маршруты.

SAMPLE	Чтение сверху вниз:	SBIRY ACJ TZ MDKU PFNV LGOW ENQX
BCDFGH		
IJKNOQ	С чередованием:	SBIRY ZTJCA MDKU VNFP LGOW XQHE
RTUVWX		
YZ	По диагонали:	Y RZ IT BJU SCKV ADNW MFOX PGQ LH E



## SkipMix

В школе я придумал еще один метод, пригодный для криптографии с карандашом и бумагой, который назвал *SkipMix*. В нем в качестве ключа для перемешивания алфавита используется строка небольших чисел, называемых *пропусками* (skips), например 3, 1, 4. Начать со стандартного алфавита. Пропустить 3 буквы и взять следующую, в данном случае D.

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Удалить эту букву, затем пропустить одну букву и взять следующую, F.

ABCDEFGHIJKLMNOPQRSTUVWXYZ D

Удалить эту букву, потом пропустить 4 буквы и взять следующую, K.

ABCEFGHIJKLMNOPQRSTUVWXYZ D F

Затем повторить цикл 3, 1, 4. Пропустить 3 буквы и взять следующую, O.

ABCEGHIJKLMNOPQRSTUVWXYZ D F K

Продолжать, пока не будут выбраны все 26 букв. В результате получается такой перемешанный алфавит:

DFKOQVZBINRXEHNSYCPAJWTGML

Метод SkipMix можно применять с ключевым словом. Возьмем ключевое слово SAMPLE. Заменяем каждую букву ее позицией в алфавите, в данном случае 19, 1, 13, 16, 12, 5. При желании можно заменить двузначные числа отдельными цифрами, в данном случае 1, 9, 1, 1, 3, 1, 6, 1, 2, 5. Используем эту строку пропусков в качестве ключа перемешивания. Отметим, что пропуск 0 допустим.

SkipMix хорошо подходит для компьютера. В этом случае алфавит состоит из 256 различных 8-битовых кодов символов. Пропусками могут быть любые целые числа от 0 до 255. Числовой ключ по-прежнему можно вывести из ключевого слова. Ключевое слово хорошо тем, что его проще запомнить и правильно ввести, чем последовательность целых чисел. Снова возьмем ключевое слово SAMPLE. Его буквам соответствуют следующие коды ASCII: 83, 65, 77, 80, 76, 69. Эти значения принадлежат узкому диапазону от 65 до 90, из-за чего перемешивание получается не таким тщательным. Чтобы расширить диапазон кодов, можно умножить их на какую-нибудь константу по модулю 256. Множитель может быть любым нечетным

числом от 7 до 39 включительно. Например, в результате умножения ASCII-кодов букв ключевого слова SAMPLE на 17 по модулю 256 получается последовательность 131, 81, 29, 80, 12, 149. Она принадлежит диапазону шириной  $149 - 12 = 137$ , гораздо более широкому, чем исходный диапазон шириной  $83 - 65 = 18$ .

Для пропусков по-прежнему допустимо только 26 значений. Получить большее множество значений можно, например, перемножив соседние числа по модулю 256. Тогда последовательность пропусков будет состоять из чисел  $83 \times 65$ ,  $65 \times 77$ ,  $77 \times 80$ ,  $80 \times 76$ ,  $76 \times 69$ ,  $69 \times 83$ , все по модулю 256. Так,  $83 \times 65 = 5395 \equiv 19 \pmod{256}$ . Числовой ключ становится равен 19, 141, 16, 192, 124, 95, т. е. покрывает диапазон шириной  $192 - 16 = 176$ .

Другой способ расширить множество значений пропусков – умножить первую букву ключевого слова на 7, вторую на 9, третью на 11 и т. д., причем все умножения производить по модулю 256.

Поскольку преобразование ключевого слова в числовой ключ будет выполнять компьютер, а не человек, вычисление может быть сколь угодно сложным. Я рекомендую брать квадратичную, а не линейную функцию, чтобы затруднить определение ключевого слова противнику, сумевшему получить открытый текст некоторых сообщений. Например, если  $K_i$  – числовые значения символов ключевого слова, то для вычисления элементов числового ключа  $N_i$  можно взять такую функцию:

$$N_i = (K_i K_{i+1} + K_{i+2} K_{i+4}) \pmod{256},$$

где нижние индексы возвращаются к единице по достижении длины ключевого слова. Например, если ключевое слово содержит 10 символов, то  $K_{11}$  будет равно  $K_1$ ,  $K_{12} - K_2$  и т. д.

\* Функция  $K_i K_{i+1} + K_{i+2} K_{i+3}$  недостаточно стойкая. Если длина ключевого слова равна  $L$ , то в ней всего  $L$  различных квадратичных членов. Эмили могла бы рассматривать их как  $L$  переменных и, решив систему  $L$  линейных уравнений, найти значения  $L$  произведений  $K_1 K_2$ ,  $K_2 K_3$ ,  $K_3 K_4$ , ...  $K_L K_1$ . После этого нетрудно найти значения всех  $K_i$ . \*\*

В этой книге алфавиты перемешаны с помощью простых ключевых слов, чтобы сразу было понятно, как они построены, например:

SAMPLEBCDFGHIJKNOQRTUVWXYZ

Это совершенно небезопасно. *Не делайте так на практике.* Я поступаю так, только чтобы помочь читателям. Поскольку вы вряд ли захотите помогать противнику, используйте хорошо перемешанные алфавиты, применяя столбцовое перемешивание, SkipMix или иную стойкую функцию перемешивания. Другие методы см. в разделе 12.3.8.

## 5.3 Номенклаторы

С XV по XVIII век королем подстановочных шифров был Номенклатор, использовавшийся равно царствующими особами, папами, дипломатами и шпионами. Каждый Номенклатор включал сотни, а то и тысячи элементов: одиночных букв, чисел, биграмм, слогов, слов и имен, предлагая для каждого до 25 подстановок. Номенклаторы сродни скорее кодам, чем шифрам, из-за чего в этой книге не рассматриваются.

## 5.4 Многоалфавитная подстановка

Методы вскрытия простых подстановочных шифров сводятся к анализу частот и контактов букв. Если вы хотите спроектировать шифр, устойчивый к таким атакам, то для начала стоит сделать бесполезной информацию о частотах и контактах.

Предположим, что для шифрования каждой буквы используется не один, а два алфавита. Скажем, первый алфавит служит для шифрования букв в нечетных позициях, а второй – в четных.

Теперь частоты букв шифртекста происходят наполовину из первого алфавита, наполовину из второго, т. е. являются средними двух наборов частот. Буква шифртекста будет иметь высокую частоту, только если ее частота высока в обоих алфавитах. Например, буква шифртекста **К** могла бы представлять букву **Е** в первом алфавите и букву **А** во втором, поэтому ее частота равна полусумме частот **Е** и **А** в открытом тексте.

Наоборот, буква шифртекста будет иметь низкую частоту, только если представляет низкочастотные буквы в обоих алфавитах, скажем **К** в первом и **В** во втором. Таким образом, в шифртексте будет меньше букв с высокой и низкой частотами, чем в открытом тексте. Если построить столбчатую *гистограмму* распределения частот букв, то пики в ней будут ниже, а впадины – не такими глубокими. Следовательно, использование двух алфавитов сглаживает распределение частот.

То же самое относится к частоте контактов. Любая часто встречающаяся биграмма, например **ТН**, будет в половине случаев начинаться в четной позиции сообщения, а в половине – в нечетной. В первом случае **Т** шифруется первым алфавитом, а **Н** – вторым, во втором – наоборот. Поэтому распределение частот контактов тоже сглаживается.

Чем больше используется алфавитов, тем более пологим становится распределение частот. На практике со времен гражданской войны в США и до Первой мировой войны применялось порядка 20 алфавитов. Существует статистический критерий, который измеряет пологость распределения частот и на ее основе дает оценку

числа алфавитов, но он очень неточен. Более точные методы описаны в разделах 5.6 и 5.7.

Пара слов об историческом развитии многоалфавитных шифров. Их ранние формы были разработаны Леоном Баттиста Альберти в 1467 году и Иоганном Тритемием в 1499 году (но опубликованы только в 1606 году). Современный вид они начали принимать после выхода книги Джована Баттиста Беласо «La cifra del Sig» в 1553 году.

## 5.5 Шифр Беласо

В шифре Беласо, придуманном Джованом Баттиста Беласо в 1553 году, используется 26 различных алфавитов, каждый из которых получается сдвигом стандартного алфавита на некоторое число позиций. Все 26 алфавитов можно расположить в виде таблицы, по одному сдвинутому алфавиту в каждой строке:

```

ABCDEFGHIJKLMNOPQRSTUVXYZ
BCDEFGHIJKLMNOPQRSTUVXYZA
CDEFGHIJKLMNOPQRSTUVXYZAB
...
ZABCDEFGHIJKLMNOPQRSTUVXY

```

Первая буква в каждой строке определяет алфавит, так что в верхней строке находится алфавит А, в следующей алфавит В и т. д. Беласо первым использовал ключ для выбора алфавита, с помощью которого шифруется буква сообщения. (Отметим, что семья Ардженти использовала ключ для перемешивания алфавита.) Беласо располагал сообщение по горизонтали. Над буквами открытого текста записывался ключ – столько раз, сколько необходимо. Для шифрования буквы он находил ее в верхней строке таблицы алфавитов, с помощью соответствующей ей буквы ключа определял строку таблицы и заменял букву открытого текста буквой шифртекста, расположенной прямо под ней в выбранной строке. В примере ниже показано, как буква открытого текста S шифровалась бы с помощью буквы ключа С:

```

ABCDEFGHIJKLMNOPQRSTUVXYZ
BCDEFGHIJKLMNOPQRSTUVXYZA
CDEFGHIJKLMNOPQRSTUVXYZAB

```

Открытая буква S (верхняя строка) шифруется буквой С (третья строка), что дает букву шифртекста U

Мы находим букву S в верхней строке таблицы. Ее ключ равен С, поэтому она шифруется с помощью третьей строки таблицы. В третьей строке, прямо под S, находится буква U. Поэтому S заменяется на U.

При шифровании слова SAMPLE ключом CAB с использованием приведенной выше таблицы алфавитов буква S заменяется буквой U.

<b>CAB CAB</b>	Ключ
<b>sa<u>m</u> ple</b>	Открытый текст
<b>UA<u>N</u> RLF</b>	Шифртекст

Следующая буква сообщения – А, ее ключ тоже А, поэтому А шифруется верхней строкой и, значит, буквой **А**. Букве М в слове SAMPLE соответствует ключ В. Она шифруется второй строкой и становится **Н**. И так далее. В итоге получается шифртекст **UANRLF**.

Вместо использования таблицы шифрование можно было бы производить с помощью линейки Сен-Сира, названной так в честь французской военной академии Сен-Сир. На рисунке ниже линейка установлена в позиции М.



Вы можете смастерить собственную линейку из дерева, картона или пластика. Верхняя шкала двойной ширины неподвижна, а нижний движок одинарной ширины может перемещаться вдоль нее. Удерживать его в нужном положении можно с помощью резинок. Любой алфавит или оба сразу могут быть перемешаны.

Шифр Беласо *симметричен*, потому что шифрование буквы Х буквой К дает в точности такой же результат, как шифрование буквы К буквой Х. Шифры, основанные на сложении ключа с открытым текстом или применении к ним операции ИСКЛЮЧАЮЩЕЕ ИЛИ, обычно являются симметричными в этом смысле.

По причинам, выходящим за пределы моего понимания, шифр Беласо теперь известен как *шифр Виженера*, а шифр, изобретенный Блезом де Виженером и описанный в разделе 5.8.2, теперь называется шифром *с автоключом*. Чтобы воздать должное автору, я продолжу называть изобретенный Беласо шифр с использованием стандартного алфавита шифром Беласо. А шифр с перемешанным алфавитом буду называть шифром Виженера. Наконец, шифр с автоключом, изобретенный Виженером, я буду называть *автоключом Виженера*.

Приписывание шифра Беласо Виженеру – пример закона Стиглера об эпонимии, утверждающего, что никакое важное научное открытие не было названо в честь первооткрывателя. Из примеров в области криптографии упомяну еще шифр Плейфера, придуманный Чарльзом Уитстоном, и код Морзе, придуманный Альфредом Вайлем. Да и сам закон Стиглера был подмечен Робертом К. Мертонем, который назвал его *эффектом Матфея* в честь апостола Матфея.

## 5.6 Метод Касиски

Больше 300 лет шифр Беласо считался невскрываемым. Французы называли его «Le Chiffre Indéchiffable», т. е. недешифрируемый

шифр. Переломным моментом стал 1863 год, когда майор Фридрих В. Касиски, прусский офицер от инфантерии, издал книгу, в которой подробно описал, как определить период многоалфавитного шифра. Теперь он называется *методом Касиски*, или *тестом Касиски*. Есть свидетельства в пользу того, что Чарльз Бэббидж, возможно, использовал этот метод еще в 1846 году, но не опубликовал его. Оле Имануил Франксен из Датского технического университета, который много писал о Бэббидже и его разностной машине, написал книгу «Mr. Babbage's Secret», где и высказал это предположение.

Идея в том, чтобы поискать повторяющиеся последовательности букв в шифртексте. Какие-то могли появиться случайно, особенно биграммы, но большинство будет связано с тем, что одни и те же буквы открытого текста зашифрованы одной и той же частью ключа. Чем длиннее повторяющаяся последовательность, тем меньше вероятность, что это произошло случайно. Если одной и той же частью ключа зашифрованы две повторяющиеся последовательности букв, то расстояние между ними должно быть кратно длине ключа. Расстояние измеряется от первого символа одного вхождения до первого символа другого вхождения. Рассмотрим следующий фрагмент шифра с ключевым словом EXAMPLE.

EXAMPLE	EXAMPLE	EXAMPLE	EXAMPLE	EXAMPLE	Ключ
therain	inspain	staysma	inlyint	heplain	Открытый текст
XEEDPTR	MKSBPTR	WQAKHXE	MKLKXYX	LBPXPTR	Шифртекст
-----+	--1----	+----2-	---+---	-3-----+	Позиция

В шифртексте триграмма PTR встречается 3 раза, в позициях 5, 12 и 33. Все три вхождения – результат шифрования открытого текста AIN символами ключа PLE. То есть это результат шифрования одной и той же триграммы открытого текста одной и той же частью ключа.

Триграмма открытого текста AIN, начинающаяся в позиции 21, порождает другую триграмму шифртекста, **EMK**, потому что зашифрована другой частью ключа, а именно EEX. С другой стороны, триграмма открытого текста THE, встречающаяся в позициях 1 и 29, и триграмма INS, встречающаяся в позициях 8 и 13, не порождают повторяющихся триграмм шифртекста, потому что зашифрованы разными частями ключа.

В этом фрагменте расстояния между повторяющимися триграммами равны  $12 - 5 = 7$ ,  $33 - 5 = 28$  и  $33 - 12 = 21$ . Все эти расстояния, 7, 21 и 28, кратны 7, т. е. длине ключевого слова EXAMPLE. Касиски показал, как такие повторения можно использовать для нахождения периода шифрования.

Рассмотрим в качестве примера криптограмму

ZVZPV TOGGE KHXS N LRYRP ZHZIO RZHA ZCOAF PNOHF  
VEYHC ILCVS MGRYR SYXYR YSIEK RGBYX YRRCR IIVYH  
CIYBA GZSWE KDMIJ RTHVX ZIKG

Это зашифрованный шифром Беласо текст на английском языке. Поиск повторяющихся последовательностей букв дает: **EK** в позициях 10, 64 и 90, **RZR** в позициях 17 и 53 и т. д. Полный перечень приведен ниже.

1	EK	10	64	90
2	RZR	17	53	
3	RY	17	60	
4	YR	18	54	59 71
5	ZHZ	21	27	
6	HZ	22	28	
7	ZI	23	101	
8	YHCI	43	79	
9	HCI	44	80	
10	CI	45	81	
11	YXYR	57	69	
12	XYR	58	70	

Сразу бросаются в глаза две повторяющиеся тетраграммы, **YHCI** и **YXYR**. Такое почти никогда не бывает случайно. Расстояние между вхождениями **YHCI** равно  $79 - 43 = 36$ , а расстояние между вхождениями **YXYR** равно  $69 - 57 = 12$ . Расстояния 12 и 36 означают, что длина ключа может быть равна 4, 6 или 12. Число вариантов можно уменьшить, проанализировав другие повторяющиеся последовательности.

Расстояние между вхождениями **RZR** равно 36, а между вхождениями **ZHZ** – 6. Другие повторяющиеся триграммы, **HCI** и **XYR**, – просто части повторяющихся тетраграмм **YHCI** и **YXYR**, поэтому они не дают дополнительной информации. Самый вероятный период этой криптограммы равен 6.

Это было просто. Посмотрим, что бывает в более сложных случаях. В других книгах рекомендуется такой метод: взять все расстояния между повторяющимися последовательностями и выписать всех их множители. Самый часто встречающийся множитель, мол, и будет периодом. Например, если расстояние равно 36, то множителями будут 1, 2, 3, 4, 6, 9, 12, 18 и 36. Такое рассуждение может направить на ложный путь.

*Во-первых*, вы можете ошибочно заключить, что период в два раза больше истинного значения. Почему? Да потому, что примерно половина расстояний будет четной, просто в силу случайности. Половина интересных нам расстояний, тех, что объясняются повторяющимися последовательностями в открытом тексте, будут четными кратными периода. Для некоторых сообщений количество таких четных кратных будет больше, чем количество нечетных кратных. Точно так же для половины случайно возникших повторяющихся последовательностей в шифртексте расстояния будут четными. Следовательно, просто по случайности будет много четных расстояний. Аналогично,  $1/3$  расстояний будет кратна 3 – тоже по чистой случайности.



При подсчете количества множителей в найденных расстояниях нужно уменьшить количество множителей 2 на  $1/2$ , количество множителей 3 – на  $1/3$  и так далее. Тогда сравнение будет более точным. Например, если множитель 3 встречается 6 раз, то уменьшение на  $1/3$  даст 4, а 2 из 6 вхождений возникли чисто случайно.

*Во-вторых*, когда повторяющаяся последовательность встречается несколько раз, расстояния между парами повторений могут приводить к неверным выводам. Если имеется  $N$  повторений, то число пар равно  $N(N-1)/2$ . В нашем примере шифртекста есть 4 вхождения **YR**, т. е.  $4 \times 3/2 = 6$  пар. Эти шесть расстояний между парами равны  $54-18=36$ ,  $59-18=41$ ,  $71-18=53$ ,  $59-54=5$ ,  $71-54=17$  и  $71-59=12$ . И какие из них являются кратными периода, да и есть ли такие вообще? Предположим, что шифртекст **XYZ** встречается 5 раз и что 3 из этих повторений возникли из одного и того же открытого текста. В результате получаем 10 расстояний, из которых только 3 связаны с повторениями в открытом тексте, а остальные 7 случайные.

Мы не хотим отбрасывать важные повторения только потому, что не можем отличить их от случайных. И вот что мы можем сделать. Предположим, что имеется потенциальный период, например 6. Возьмите номера позиций, в которых встречается повторяющаяся последовательность, по модулю 6. (Вы еще не забыли модульную арифметику? Если забыли, еще раз прочитайте раздел 3.5.)

Попробуем этот *модульный метод*. Снова возьмем все 5 вхождений биграммы **YR**, вычислим их позиции по модулям 5, 6 и 7 и посмотрим, что получается.

<b>18 54 59 71</b>	Позиции <b>YR</b> в шифртексте
<b>3 4 4 1</b>	Позиции по модулю 5
<b>0 0 5 5</b>	Позиции по модулю 6
<b>4 5 3 1</b>	Позиции по модулю 7

Все 4 вычета по модулю 7 различны. Если период равен 7, то все повторения **YR** случайны. По модулю 5 есть всего два одинаковых вычета. Если период равен 5, то только 2 из 4 вхождений объясняются повторяющимся открытым текстом. Но вот с модулем 6 мы наткнулись на золотую россыпь. Если период равен 6, то все 4 вхождения **YR** происходят из 2 разных повторяющихся биграмм в открытом тексте: одна в позициях 18 и 54 с расстоянием 36, другая в позициях 59 и 71 с расстоянием 12.

Как это случилось? Взгляните еще раз на список повторяющихся последовательностей. Видно, что биграмма **YR** встречается в повторяющейся триграмме **R Y R** и в повторяющейся тетраграмме **Y X Y R**. Каждая привносит одно повторение.

Теперь рассмотрим второй метод определения периода многоалфавитного шифра. На случай, когда повторяющиеся последовательности не дают убедительных свидетельств, хорошо бы иметь запасной план.



## 5.7 Индекс совпадения

Индекс совпадения предложил американский криптоаналитик Уильям Ф. Фридман в 1922 году. Идея очень проста, но имеет весьма важные последствия. Допустим, что два сообщения зашифрованы многоалфавитным шифром, но с разными ключами и, возможно, с разными периодами. Если сравнивать оба шифртекста побуквенно, то шанс, что две соответственные буквы совпадут, равен  $1/26$ , т. е. приблизительно 0.0385. Если длина обоих сообщений равна 52, то мы ожидаем, что в  $52/26 = 2$  соответственных парах буквы совпадут. Я зашифровал 52-буквенный открытый текст ON THE FIRST DAY OF SPRING A YOUNG MANS FANCY TURNED TO BASEBALL шифром Беласо с ключами MARS и VENUS. (Совпавшие буквы выделены; то, что в обоих случаях они равны F, – чистая случайность.)

ANKZQ FZJET USKOM KBRZF SAPGG NXEMN JXMNT QFUIF QDKGN AJWNA CD  
JRGBW AMEML YELIX NTECF BELIM IKZUF NJNHU TXNLF ZHGIT VWRVS GP

Теперь представьте два сообщения, зашифрованных одним и тем же ключом. Каждая пара соответственных букв шифруется одним и тем же символом ключа, поэтому если буквы открытого текста совпадают, то совпадут и буквы шифртекста. Частота буквы А равна приблизительно 0.08, поэтому вероятность, что обе буквы открытого текста равны А, составляет  $0.08^2 = 0.0064$ , вероятность, что обе буквы равны В, составляет приблизительно  $0.015^2 = 0.000225$ , и т. д. для всего алфавита. Сумма по всем 26 буквам заключена между 0.0645 и 0.0675 в зависимости от того, какой таблицей частот мы пользуемся. В любом случае это приблизительно 1/15. Вероятность, что две соответственные буквы шифртекста совпадают при условии использования одного ключа, приближенно равна 1/15, что на 73 % выше вероятности 1/26, получающейся, когда ключи различны.

Этот факт можно использовать для определения длины ключа многоалфавитного шифра. Пронумеруем символы шифртекста  $C_1, C_2, C_3, \dots$  и обозначим длину ключа  $L$ . Мы можем сравнить символы шифртекста с теми же символами, сдвинутыми на  $S$  позиций. То есть мы сравниваем  $C_1$  с  $C_{1+S}$ ,  $C_2$  с  $C_{2+S}$ ,  $C_3$  с  $C_{3+S}$  и т. д.

Если сдвиг  $S$  кратен  $L$ , то  $C_i$  зашифрована тем же алфавитом, что  $C_{i+S}$  для любой позиции  $i$ , поэтому вероятность, что два соответственных символа шифртекста совпадают, равна 1/15. Если сдвиг не кратен  $L$ , то соответственные символы не будут зашифрованы одним и тем же алфавитом и вероятность их совпадения равна всего 1/26. Количество совпавших символов должно быть наибольшим при  $S = L$ ,  $S = 2L$  и т. д. Чтобы все стало ясно, нужно попробовать несколько сдвигов. Сдвиги, при которых число совпадений максимально, обычно являются кратными периоду.

На первый взгляд, проверка большого числа сдвигов – работа для компьютера, но в действительности это нетрудно сделать и вруч-

ную. Напишем криптограмму на двух длинных полосках бумаги. Затем будем двигать одну полосу вдоль другой и подсчитывать количество совпавших символов для каждой величины сдвига. Буквы нужно располагать с равными промежутками, чтобы они правильно совмещались. Для этого можно воспользоваться миллиметровкой или писать буквы по линейке.

ZVZPVT	OGGEKH	XSNL	RYRP	ZHZI	ORZH	HAZC	OA	FPNO	H	FVEYH	CILCVS...
ZVZPVT	OGGEKH	XSNL	RYRP	ZHZI	ORZH	HAZC	OA	FPNO	H	FVEYH	CILCVS...

У индекса совпадения есть и другое применение, оказавшееся просто неоценимым для криптоаналитиков. Он может обнаружить, что два сообщения зашифрованы одним и тем же ключом. Допустим, Эмили пользуется машинным шифром, который порождает многоалфавитный шифр с очень длинным периодом, скажем 100 000. Для сравнения машина Энигма, которая использовалась в немецкой армии во время Второй мировой войны, имела период  $26 \times 25 \times 26 = 16\,900$ . Пусть имеется несколько тысяч перехваченных сообщений. Каждое сообщение зашифровано каким-то сегментом этого длинного ключа. Сдвигая одно сообщение относительно остальных и используя как индекс совпадения, так и анализ повторяющихся последовательностей шифртекста, мы можем найти участки различных сообщений, зашифрованные одной и той же частью ключа.

Отыскав достаточно таких перекрывающихся сегментов ключа, мы сможем приступить к сращиванию сегментов с целью получить более длинные сегменты. Имея достаточно сообщений, зашифрованных одним и тем же сегментом ключа, можно начинать вскрывать их, применяя обычные методы: частоту букв, частоту контактов, определение общих слов и т. д.

## 5.8 И снова об индексе совпадения

Существует еще один метод оценки периода многоалфавитного шифра, также называемый индексом совпадения и также открытый Уильямом Ф. Фридманом. Он заключается в вычислении вероятности совпадения двух букв при использовании двух, трех и т. д. алфавитов. Вероятности вычисляются заранее и хранятся в виде таблицы. Идея в том, чтобы вычислить ту же самую статистику для сообщения и сравнить ее с таблицей. Предполагается, что наилучшее соответствие и будет периодом шифра. На практике метод действительно дает близкий результат, но иногда ошибается на 1, 2 или даже 3. Если период больше 10, то этот метод бесполезен. Он не намного лучше случайного угадывания, поэтому не будем тратить времени на объяснение деталей.

Шифры Беласо и Виженера широко использовались на протяжении 1880-х годов. Но по мере распространения информации о методе Касиски их применение постепенно сходило на нет, а когда был опубликован индекс совпадения, их вообще перестали употреблять. Тем не менее и сегодня они остаются одними из самых популярных среди шифровальщиков-любителей. Не раз, сообщая какому-то человеку, что я пишу книгу по криптографии, в ответ я слышал, что собеседник знает невскрываемый шифр. И неизменно это оказывался шифр Беласо, который они называли шифром Виженера. Тогда мне приходилось доказывать, что собеседник неправ, вскрывая предложенный им шифр. Но их описание шифра было настолько перепутано, что я вынужден был создать веб-страницу *mastersoftware.biz/vigenere.htm*, на которой объяснил, как правильно использовать эти шифры.

## 5.9 Вскрытие многоалфавитного шифра

Определив период с помощью метода Касиски или индекса совпадения, можно переходить к следующему шагу – определению отдельных алфавитов. Рассмотрим сначала более простой случай – шифр Беласо.

### 5.9.1 Вскрытие шифра Беласо

В шифре Беласо все подстановочные алфавиты представляют собой простые сдвиги стандартного алфавита на сколько-то позиций. Чтобы вскрыть шифр, достаточно определить величину сдвига. Первый шаг – разделить символы, зашифрованные каждой буквой ключа. Снова обратимся к примеру из раздела 5.5. Поскольку мы определили, что период равен 6, запишем шифртекст группами по 6.

ZVZPVT OGGEKH XSNLRY RPZHZI ORZHZA ZCOAFP NOHFVE YHCILC VSMGRY  
RSYXYR YSIEKR GBYXYR RCRIIV YHCIYB AGZSWE KDMIJR THVXZI KG

Первая буква в каждой группе зашифрована первой буквой ключа, вторая – второй и т. д. Если бы мы записали шифртекст по вертикали в 6 столбцов:

123456  
ZVZPVT  
OGGEKH  
XSNLRY  
...

то первый столбец содержал бы буквы, зашифрованные первой буквой ключа, второй – буквы, зашифрованные второй буквой ключа, и т. д.





самое большое произведение. Если пробежаться по алфавиту и сложить все 26 произведений, то сумма будет наибольшей, когда все высокие пики совмещаются, и наименьшей, когда самые высокие пики находятся против самых глубоких впадин.

В этом и состоит идея. Пробуем все 26 возможных сдвигов. Совмещаем частоты букв шифртекста со сдвинутыми стандартными частотами для английского языка и складываем все 26 произведений. Наибольшая сумма укажет наиболее вероятный сдвиг. А это даст нам самую вероятную букву ключа. Вторая по величине сумма даст следующий по величине вероятности сдвиг и т. д. Я называю эту технику методом *высоких пиков*.

Шифр Беласо получает оценку 2.

### 5.9.2 Вскрытие шифра Виженера

Примерно через 30 лет после Беласо Блез де Виженер улучшил шифр Беласо в двух отношениях. Во-первых, он добавил линейки за пределами таблицы алфавитов. Это дало возможность порождать перемешанный алфавит, не прибегая к перемешиванию таблицы. В примере ниже используется ключевая фраза в горизонтальных линейках и ключевое слово YOUTH в вертикальных. Второе усовершенствование, автоключ, описывается в разделе 5.10.

FIRSTLOVEAB	CDGHJKMNPQUWXYZ	Верхняя линейка
Y	ABCDEFGHIJKLMNOPQRSTUVWXYZ Y	
O	BCDEFGHIJKLMNOPQRSTUVWXYZA O	
U	CDEFGHIJKLMNOPQRSTUVWXYZAB U	
T	DEFGHIJKLMNOPQRSTUVWXYZABC T	
H	EFGHIJKLMNOPQRSTUVWXYZABCD H	
A	FGHIJKLMNOPQRSTUVWXYZABCDE A	
B	GHIJKLMNOPQRSTUVWXYZABCDEF B	
	. . .	
Z	ZABCDEFGHIJKLMNOPQRSTUVWXYZ Z	
FIRSTLOVEAB	CDGHJKMNPQUWXYZ	Нижняя линейка

Чтобы зашифровать букву В с помощью буквы ключа U, находим букву ключа U в *ключевой линейке* слева или справа от строки и букву открытого текста В в *буквенной линейке* сверху или снизу от столбца. Буква шифртекста находится на пересечении строки U и столбца В, это М. При дешифрировании используем букву ключа для нахождения строки, находим букву шифртекста в этой строке и берем букву открытого текста из буквенной линейки сверху или снизу.

Для шифрования вручную я рекомендую рисовать горизонтальные и вертикальные разделители через каждые 4 или 5 строк или столбцов. Или использовать пластмассовый угольник, чтобы точно находить точки пересечения.

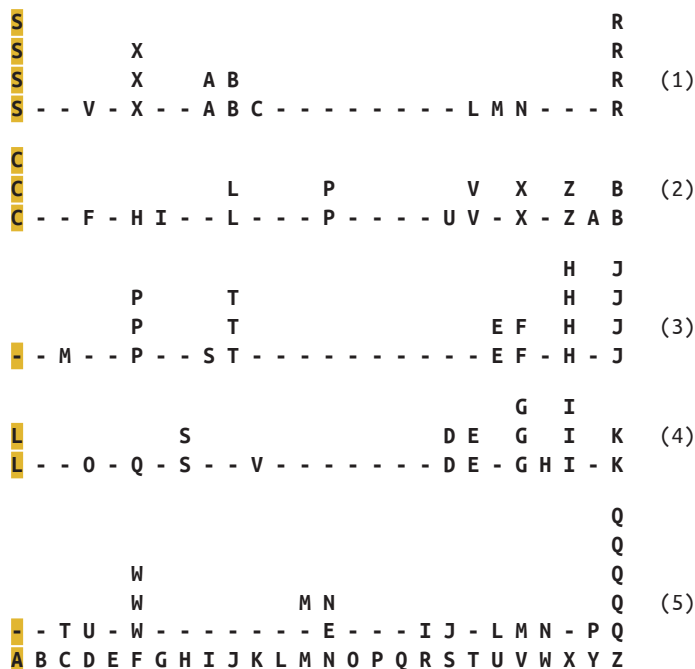
Ниже приведен пример сообщения, зашифрованного с применением этой формы шифра Виженера. В данном случае период равен 5.

SLMDQ BXSLM XINSQ NHJEQ SVJGW LBJSJ BFEII CBHVN RUTGW RPHEH  
VXPIL RPPIW SAJHQ SVTKU ACFQQ ACFDT MCTOM XZEKE XZPLP RLHG

У этого шифра есть серьезная слабость. Поскольку каждая строка в таблице – стандартный алфавит, сдвинутый на какое-то число позиций, то каждый алфавит шифра будет таким же, как все остальные алфавиты, только сдвинутым на сколько-то позиций. Бесполезно сравнивать алфавиты шифров со стандартным алфавитом, потому что они перемешаны, однако можно определить сдвиги, сравнивая алфавиты шифров друг с другом – на глаз или используя метод высоких пиков.

На рисунке ниже показаны гистограммы для пяти алфавитов шифров, сдвинутые так, чтобы пики и ущелья совместились. Все буквы шифртекста в первом столбце (выделенном) представляют одну и ту же букву открытого текста. Это означает, что буква S в первом алфавите, буква C во втором алфавите и буква L в четвертом алфавите представляют одну и ту же букву. Заменяем их все буквой A. Во втором столбце нет букв. В третьем столбце M из третьего алфавита и T из пятого алфавита представляют одну и ту же букву шифртекста. Заменяем их все буквой C. Все буквы шифртекста в 26-м столбце будут заменены буквой Z.

Гистограммы для сдвинутых алфавитов шифров



Это превращает шифртекст в простую подстановку, которую можно вскрыть методами из раздела 5.1. Шифр Виженера получает оценку 2.

### 5.9.3 Вскрытие общего многоалфавитного шифра

Общий многоалфавитный шифр также можно реализовать с помощью таблицы алфавитов. Строки таблицы могут быть перемешаны по любой схеме и независимо друг от друга. Отметим, что число строк не обязано совпадать с числом столбцов. Для компьютерных шифров удобна таблица с 512 строками и 256 столбцами, так чтобы любой символ шифртекста встречался в каждом столбце дважды. Это усложнит задачу определения ключа противнику, который получил шифртекст и соответствующий ему открытый текст. Ниже приведен частичный пример таблицы алфавитов со 100 строками:

```
00 IBVMRUCNJYSAWEPZODXGQKTHLF
01 LOEIBQXJTMFRWAPUCZNVGKYHSD
02 GTAOKYSFUJPERHXLBVQDIZMWCN
03 DPNETHVBZJSGMWAXIQOFYRCUKL
. . .
99 BRXIZPYLVJCNQHTKESUAMGWDOF
```

Для этой таблицы нужен числовой ключ, в котором строка, используемая для шифрования, определяется двумя десятичными цифрами. 20-значный ключ породил бы многоалфавитный шифр с периодом 10.

Вскрытие общего многоалфавитного шифра очень похоже на вскрытие одноалфавитного шифра. Мы начинаем с подсчета частот и составления таблицы контактов для каждого столбца. В этом случае контактами для столбца *C* будут столбцы *C*–1 и *C*+1, причем одним из контактов последнего столбца является первый. В одном столбце будет меньше вхождений каждой буквы, поэтому выводы придется делать на основе меньшего объема данных. А значит, большую ценность приобретают вдохновенные озарения, но это приходит с опытом.

Начнем с такого многоалфавитного шифртекста:

```
OH0YO RKKDF JKYSU ZONSO OKGSC LHKDK FKHUW ZGGSN ZYYZK JPHZO
RKKDP KCHUK LHYYF BGBSC FKKFK CZIUX VOZRU TZWSN UZYSU ZONSO
OPHCO RPNDZ ZPIHK OGDHN UWOSN ZYYZK XOQDX BNMUO R
```

Немного приглядевшись к нему, мы обнаруживаем две длинные повторяющиеся последовательности: **YSUZONSOO** в позициях 13 и 93 и **SNZYYZK** в позициях 39 и 124. В обоих случаях расстояние между вхождениями кратно 5, откуда следует, что период равен 5. Вероятно, эти длинные повторения соответствуют расхожим словам или фразам либо словам, тесно связанным с темой сообщения.

Ниже показаны схемы контактов для каждой из 5 букв ключа. Чтобы упростить объяснения, я буду помечать каждую букву шифртекста цифрой, равной номеру ее алфавита. Так, **C1** означает букву



шифртекста **С** в алфавите 1 (т. е. зашифрованную первой буквой ключа), **НЗ** – букву шифртекста **Н** в алфавите 3 (зашифрованную третьей буквой ключа) и т. д.

Напомним, что буквы с большим числом разных контактов с обеих сторон обычно оказываются гласными, а буквы с меньшим числом разных контактов – согласными.

(1)	(2)	(3)
FX B GN	K C H	G B S
K C Z	ZBO G GBD	G D H
KC F KK	OLL H OKY	KG G SS
FK J KP	RJOFRF K KYGHKK	KPCP H WZUC
P K C	B N M	ZP I UH
CK L HH	ZVZX O NZNQ	KHKK K DDDF
OOK O HKPG	JORZ P HHNI	N M U
OOOO R KKP	U W O	OOP N SSD
U T Z	ZZ Y YY	HW O YS
NN U ZW	CTU Z IWY	O Q D
X V O		Z W S
K X O		KYHZY Y SZYSZ
UUNUZ N Z OGYOPY		O Z R
(4)	(5)	
H C O	SS C LF	
KKKNQ D FKPZX	DY F JB	
K F K	DZUFHZ K FJLCOX	
ID H KN	SSHS N ZUUZ	
Z R U	YSZSCU O RORORR	
YNGGBWYNO S UOCNCNUON	D P K	
HIM U KXO	SWRS U ZTZ	
H W U	UD X VB	
OY Y OF	D Z Z	
YHY Z KOK		

Зная эти контакты, мы можем предварительно идентифицировать **G2**, **K2**, **O2**, **P2**, **НЗ**, **K5** как гласные, а **R1**, **Z1**, **K3**, **НЗ**, **D4**, **S4**, **O5**, **U5** как согласные. Исходя из высокой частоты, можно предположить, что **S4** представляет букву **T**.

Дальше все происходит так, как для простой подстановки. Мы обновляем схемы контактов с учетом того, какие буквы идентифицированы как гласные и согласные, а также помечаем в шифртексте гласные и согласные. Эта информация используется для уточнения и исправления произведенной ранее классификации букв по типам и для определения отдельных букв.

Я не стану повторять все шаги из раздела 5.1. Логика такая же, только шаги помельче и их побольше, да и возвращаться назад приходится чаще. Общий многоалфавитный шифр получает оценку 3.

## 5.10 Автоключ

В разделе 5.9.2 я упоминал, что Вижнер внес два усовершенствования в шифр Беласо. Первое – размещение линеек по краям таблицы алфавитов для порождения перемешанного алфавита. Второе – *автоключ*.

Идея автоключа заключается в том, чтобы использовать открытый текст сообщения в качестве ключа для шифрования остальной части сообщения. Раннее воплощение этой идеи придумал итальянский физик, математик и астролог Джироламо Кардано. В системе Кардано каждая буква зашифровывалась с использованием ее самой в качестве ключа. Это работает только для алфавитов с нечетным числом букв. В английском алфавите, где букв 26, и А, и N преобразовывались бы в А, поэтому получателю пришлось бы гадать, что имелось в виду. Но даже для алфавита нечетной длины автоключ Кардано порождает простую подстановку, не более того.

Вижнер усовершенствовал метод Кардано, добавив запаздывание. Он использовал однобуквенный ключ для шифрования первой буквы, первую букву открытого текста для шифрования второй буквы, вторую букву открытого текста для шифрования третьей буквы и т. д. В наши дни ключевое слово используется для шифрования первой группы букв, затем эта группа букв открытого текста используется для шифрования второй группы и т. д. В примере ниже ключ SAMPLE используется совместно с таблицей алфавитов Беласо, т. е. с перемешанными алфавитами.

<u>SAMPLE</u> THEDEL EGATIO NMUSTP RESENT AUNITE	Ключ
THEDEL EGATIO NMUSTP RESENT AUNITE DFRONT	Открытый текст
LHQSPX XNEWMZ RSULBD EQMWGI RYFMGX DZEWGX	Шифртекст

В случае перемешанных алфавитов вскрытие не вызывает трудностей. Для определения длины ключа можно использовать индекс совпадения, описанный в разделе 5.7. Зачастую индекс оказывается намного выше, если шифртекст сдвинут на величину, кратную длине ключа, например:

LHQSPX XNEWMZ RSULBD EQMWGI RYFMGX DZEWGX
LHQSPX XNEWMZ RSULBD EQMWGI RYFMGX DZEWGX

Предположим, мы определили, что ключевое слово состоит из 6 букв. Пробуем каждую букву алфавита в роли первой буквы ключа. Начинаем с А. Поскольку первая буква шифртекста L, первая буква открытого текста также должна быть L. Она также будет ключом для 7-й буквы сообщения. Поскольку седьмая буква шифртекста X, седьмой буквой открытого текста должна быть М.

Продолжаем в том же духе. Каждая гипотеза насчет первой буквы ключа даст соответствующие ей буквы открытого текста в позициях 1, 7, 13, 19, 25 и 31, т. е. каждую шестую букву открытого текста. Всего имеется 26 наборов букв, по одной для каждой буквы ключа. У некоторых из этих шести букв будут характерные для английского языка частоты букв, у других – маловероятные. Повторим процедуру для второй буквы ключа. Каждая гипотеза даст буквы открытого текста в позициях 2, 8, 14, 20, 26 и 32.

Теперь возьмем 5 самых вероятных вариантов для букв 1, 7, 13, ... и объединим их в пары с 5 самыми вероятными вариантами для букв 2, 8, 14, .... Это даст 25 наборов биграмм. Одни из них будут очень вероятными, другие – неправдоподобными. Возьмем 10 самых правдоподобных биграмм и объединим их с 5 самыми вероятными вариантами для третьей буквы ключа. Этот даст 50 наборов триграмм. Выберем из них 10 самых правдоподобных и объединим в пары с 5 наилучшими вариантами для четвертой буквы ключа. К этому моменту начнут проявляться некоторые слова открытого текста, и как правильно выбрать буквы ключа, станет очевидным.

Если вы проделываете все это на компьютере, опустите этап построения биграмм. Просто переберите все  $26^3$  комбинаций первых трех букв ключа и сразу переходите к триграммам. Затем повторите это для трех букв ключа, начиная со второй. Наиболее вероятные варианты для первых трех букв и следующих трех букв должны перекрываться. То же самое верно для третьего и четвертого наборов букв ключа. В итоге выбор быстро сузится, и вы найдете ключевое слово. Автоключ Виженера со стандартными алфавитами получает оценку 3.

## 5.11 Бегущий ключ

Бегущий ключ похож на автоключ, но вместо короткого ключевого слова или фразы используется *ключевой текст*, возможно, такой же длины, как само сообщение. Метод бегущего ключа так и не получил широкого распространения на практике, потому что требуется, чтобы у обеих сторон были в точности одинаковые ключевые тексты. Если одна сторона запомнила или записала ключ в виде MINE EYES HAVE SEEN THE GLORY OF THE COMING OF THE LORD, а другая в виде MY EYES HAVE SEEN THE GLORY OF THE COMING OF THE LORD, то общение станет невозможным. Решить эту проблему можно, например, с помощью двух экземпляров печатной книги, по одному у каждой стороны, но тогда они всегда должны иметь эту книгу при себе. Для взаимодействия с помощью компьютеров это не проблема, потому что там можно хранить тысячи книг.

И снова, если используется таблица Беласо со стандартным английским алфавитом, то шифр с бегущим ключом вскрыть нетрудно,

хотя и утомительно. Один из способов, работающий как для автоключа, так и для бегущего ключа, – угадать слово, которое может встречаться в тексте. Слово может быть частью как открытого, так и ключевого текста – с этим криптограф должен будет разобраться позже. Вероятным словом, или *зацепкой* (*crib*), может быть часто встречающееся английское слово, например THE или AND, или слово, относящееся к предположительной теме сообщения.

Например, если сообщение касается коммерческих переговоров, то вероятными словами могут быть TARIFF, SHIPPING, REPRESENTATIVE, BARGAINING или еще что-то в этом роде.

Идея в том, чтобы попробовать вероятное слово во всех возможных позициях сообщения. Этот процесс называется *волочением слова* (*word dragging*). Зная слово открытого текста и соответствующее ему слово шифртекста, мы можем получить фрагмент ключа. Если позиция слова найдена правильно, то этот фрагмент будет выглядеть как обычное английское слово. Чем длиннее вероятное слово, тем больше уверенности в его правильности. Определив слово, можно попытаться угадать буквы, затем предшествующие или последующие слова, чтобы расширить брешь.

Есть и другая методика, рассчитанная на компьютер. В ней используется математическое понятие условной вероятности. Это вероятность наступления события А, при условии что событие В имело место. Простая вероятность события А обозначается  $P(A)$ , а условная вероятность события А при условии события В –  $P(A|B)$ . Если АВ обозначает событие «А и В», то условная вероятность А при условии В равна  $P(A|B) = P(AB)/P(B)$ . Следовательно,  $P(AB) = P(A|B)P(B)$ .

Пример поможет разобраться. При бросании двух стандартных костей вероятность выбросить 12 равна  $1/36$ . Но если после броска первой кости выпало 6, то вероятность выпадения 12 увеличивается до  $1/6$ . Пусть А означает «выпало 12», в В «на первой кости выпало 6». Тогда  $P(A) = 1/36$ ,  $P(B) = 1/6$ . АВ означает, что выпало 12 и после первого броска выпало 6.  $P(AB)$  также равно  $1/36$ , потому что если выпало 12, значит, при первом броске наверняка выпало 6. Применяя нотацию записи условной вероятности, имеем  $P(A|B) = P(AB)/P(B) = (1/36)/(1/6) = 1/6$ . Таким образом, условная вероятность выбросить 12, при условии что на первой кости выпало 6, равна  $1/6$ .

Воспользуемся условной вероятностью для вскрытия шифра с бегущим ключом. Для этого нам понадобятся таблицы вероятностей одиночных букв, биграмм и триграмм. Их можно вычислить, подсчитав буквы, биграммы и триграммы в большом корпусе текстов. Много таких корпусов имеется на сайте проекта Гутенберг по адресу [www.gutenberg.org](http://www.gutenberg.org). При скачивании выберите вариант plaintext (простой текст). Кое-какие готовые результаты можно найти в интернете.

Нужно будет назначить вероятности всем возможным биграммам и триграммам, а не только встречающимся в корпусе. Для биграмм все понятно. Если биграмма АВ ни разу не встретилась, то можно

положить  $P(AB) = P(A)P(B)$ , однако я предлагаю задавать меньшее значение, просто потому, что  $AB$  не встречается. Я беру  $P(AB) = P(A)P(B)/3$ . Имея полный набор вероятностей биграмм, можно распространить его на триграммы, положив  $P(ABC)$  равной максимуму из  $P(A)P(BC)$  и  $P(AB)P(C)$ . И снова я предлагаю брать меньшее значение, потому что триграмма  $ABC$  ни разу не встретилась. Например, можно положить  $P(ABC)$  равной максимуму из  $P(A)P(BC)/3$  и  $P(AB)P(C)/3$ . Наличие искусственных вероятностей означает, что сумма вероятностей всех биграмм и всех триграмм будет больше 1. С точки зрения математики, это нонсенс, но на практике несущественно.

Теперь, располагая необходимыми инструментами, мы можем заняться шифром с бегущим ключом. Выберем начальную позицию в сообщении, скажем  $s$ , и попробуем все возможные триграммы в позициях  $s, s+1, s+2$ . Посмотрим, какие им соответствуют триграммы в открытом тексте. Перемножим вероятности триграммы ключа и триграммы текста, чтобы получить вероятность такой конфигурации. Сохраним 10 000 наиболее вероятных конфигураций и отбросим остальные. Для каждой выбранной триграммы попробуем все возможные буквы ключа в позиции  $s+3$  и посмотрим на соответствующую букву открытого текста. Предположим, что триграмма равна  $JKL$ , следующая буква ключа  $M$ , а соответствующая тетраграмма открытого текста  $ABCD$ . Мы можем оценить вероятность тетраграммы ключа  $JKLM$ , воспользовавшись условной вероятностью  $P(KLM|KL)$ , т. е. вероятностью того, что за биграммой  $KL$  следует  $M$ . Она вычисляется по вероятностям триграмм как  $P(KLM)/P(KL)$ , т. е. как результат деления вероятности триграммы  $KLM$  на вероятность биграммы  $KL$ . Таким образом, вероятность тетраграммы оценивается как  $P(JKL)P(KLM|KL)$ . Это делается как для тетраграммы ключа, так и для тетраграммы открытого текста,  $ABCD$ .

Оценим вероятность этой конфигурации, перемножив вероятности тетраграммы ключа и тетраграммы открытого текста. Снова сохраним 10 000 наиболее вероятных конфигураций и отбросим остальные. Продолжаем до тех пор, пока решение не станет очевидным. Все это можно проделать на компьютере без участия человека.

Шифр Виженера с бегущим ключом со стандартными алфавитами получает оценку 4.

## \*5.12 Моделирование роторных машин

Многоалфавитные шифры были непрямым атрибутом электро-механических роторных машин, которые использовались начиная с 1920-х годов. Период этих машин может быть порядка миллиарда или триллиона. Периода может и вообще не быть, если движение роторов зависит от символов открытого текста или шифртекста.

Начиная с 1915 годов и до окончания Второй мировой войны было произведено по меньшей мере 70 разных типов таких машин. Есть несколько сайтов, на которых приведены их фотографии и описания.

У любой машины есть один или несколько роторов, обычно от 3 до 6, но их число может доходить и до 10. Каждый ротор выполняет простую подстановку. После зашифровывания каждой буквы некоторые роторы проворачиваются, так что для следующей буквы используется другая подстановка. Благодаря различным системам кулачков, шестеренок, лапок и собачек движение роторов непредсказуемо. Я хочу сказать – непредсказуемо для противника.

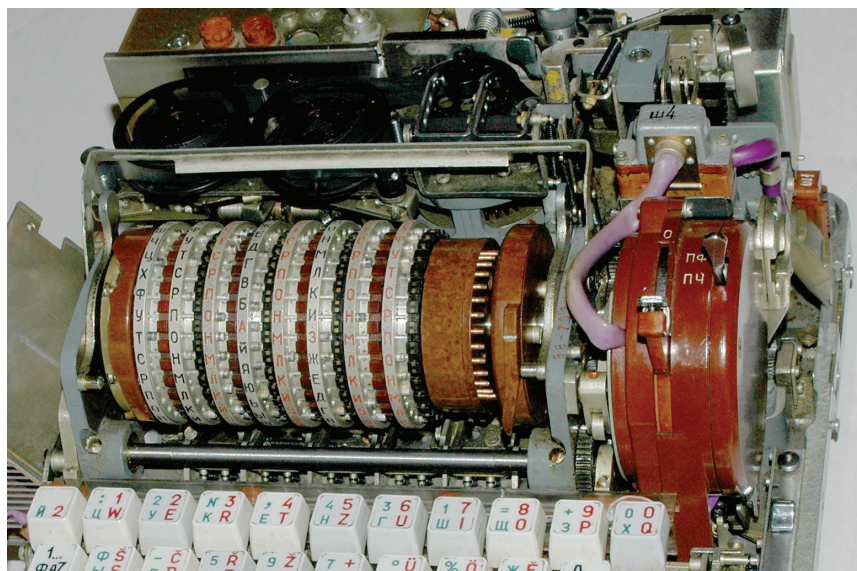


Описывать роторную машину будет проще, если заменить буквы алфавита числами. Для механических роторных машин, в которых каждый ротор может находиться в одной из 26 позиций, соответствующих 26 буквам алфавита, заменим А на 0, В на 1, С на 2 и так далее вплоть до Z, заменяемой на 25. Иными словами, будем использовать порядковую нумерацию, начиная с 0. При компьютерном моделировании будем использовать 8-битовые байты и заменять символы их числовыми кодами в какой-нибудь стандартной кодировке, например UTF-8. При такой системе А соответствует 65, В – 66, С – 67, ..., Z – 90. Другие символы – строчные буквы, цифры и знаки препинания – также заменяются их кодами в UTF-8.

Коль скоро мы перешли к работе с числами, мы можем производить над ними арифметические операции, например складывать и вычислять вычеты по модулю 26 или 256. О модульной арифметике см. раздел 3.6.

Производились шифровальные машины, имевшие до 16 роторов. На рисунке ниже показана 10-роторная советская машина Фиалка, которая использовалась странами Варшавского договора с 1956 по 1990-е годы. Фотография публикуется с разрешения Пола Хадсона на условиях лицензии CC BY 2.0.





### 5.12.1 Однороторная машина

Начнем с простого механического ротора. Ротор выполняет простую подстановку, поэтому его можно смоделировать таблицей подстановки  $S$ , представляющей из себя перемешанный алфавит, как в строке таблицы алфавитов. Элементы списка пронумерованы от 0 до 25, по числу букв в алфавите.  $N$ -й элемент таблицы подстановки, обозначаемый  $S(N)$ , подставляется вместо  $N$ -й буквы алфавита. То есть  $S(0)$  подставляется вместо А,  $S(1)$  – вместо В и т. д.

При повороте меняется позиция ротора. Позицию можно представить числом  $P$ , изменяющимся от 0 до 25. Провернувшись на 26 позиций, ротор вернется в исходную позицию 0. Когда ротор находится в позиции  $P$ , вместо  $N$ -й буквы алфавита подставляется  $S(N+P)$ . То есть если ротор находится в позиции 5, то вместо А подставляется  $S(5)$ , вместо В –  $S(6)$  и т. д. Естественно,  $N+P$  изменяется циклически, т. е.  $S(26)$  совпадает с  $S(0)$ ,  $S(27)$  – с  $S(1)$  и т. д. Иными словами,  $N+P$  – сокращенная запись для  $(N+P) \bmod 26$ .

В механической роторной машине роторы поворачиваются на разное число позиций после шифрования каждой буквы. Это нерегулярное движение можно смоделировать последовательностью величин шагов, скажем  $(a, b, c, d, e)$ . В первом цикле ротор продвигается (шагает) на  $a$  позиций. Во втором цикле он продвигается на  $b$  позиций и т. д. В шестом цикле последовательность повторяется. То есть если в начальный момент ротор занимал позицию  $P$ , то после одного цикла он займет позицию  $P+a$ , после двух циклов – позицию  $P+a+b$ , после 5 циклов – позицию  $P+a+b+c+d+e$ . После шести циклов ротор будет находиться в позиции  $P+2a+b+c+d+e$ . В механических устройствах каждый ротор обычно поворачивается на небольшое

число позиций, часто на 0 или 1 позицию в цикле в зависимости от того, поднят определенный кулачок или опущен. При компьютерном моделировании таких ограничений нет. Величины шагов могут принимать значения от 0 до 25 при моделировании механического ротора или от 0 до 255 при использовании 8-битовых байтов для представления символов.

Поскольку мы выбрали ключ с пятью шагами, эта однороторная машина будет повторяться после  $5 \times 26 = 130$  циклов. Если сумма  $a+b+c+d+e$  четная, то машина будет повторяться после 65 циклов, а если  $a+b+c+d+e$  кратно 13, то после всего лишь 10 циклов. Очевидно, что один ротор не обеспечивает достаточной безопасности. Однороторный машинный шифр получает оценку 3.

### 5.12.2 Трехроторная машина

Рассмотрим моделирование более практичной роторной машины. В ней используется три ротора и 8-битовая кодировка UTF-8. Для трех роторов нужны три таблицы подстановки,  $S_1$ ,  $S_2$  и  $S_3$ . Когда роторы занимают позиции  $P_1$ ,  $P_2$  и  $P_3$ ,  $N$ -я буква алфавита зашифровывается как  $S_3(S_2(S_1(N+P_1)+P_2)+P_3)$ .

Для каждой из трех таблиц подстановки определен свой список шагов: для  $S_1$  – шаги  $(a_1, a_2, a_3, \dots, a_i)$ , для  $S_2$  – шаги  $(b_1, b_2, b_3, \dots, b_j)$  и для  $S_3$  – шаги  $(c_1, c_2, c_3, \dots, c_k)$ . Если сумма шагов каждого ротора нечетна, а  $i, j, k$  – взаимно простые числа, то период такой машины равен  $256ijk$ . Например, если  $i = 10$ ,  $j = 11$  и  $k = 13$ , то  $ijk = 1430$ , т. е. период равен  $1430 \times 256 = 366\,080$ . Результат такой же, как для многоалфавитного шифра с таблицей алфавитов, содержащей 366 080 строк, при том что каждая строка используется в цикле только один раз.

Допустим, что все три таблицы подстановки и последовательность шагов известны, например стандартизованы и приняты большим сообществом. Может возникнуть мысль, что Эмили нужно проверить всего  $256^3 = 1.67 \times 10^7$  начальных конфигураций роторов, чтобы вскрыть любое сообщение. На современном персональном компьютере это заняло бы несколько секунд. Однако такой ход рассуждений ошибочен.

Рассмотрим два разных состояния машины. В обоих состояниях роторы занимают одни и те же позиции, но эти позиции находятся в разных точках последовательности шагов. Начав шифрование из этих двух состояний, мы получим разные последовательности алфавитов шифра, т. е. одно и то же сообщение будет зашифровано по-разному. Для вскрытия шифра методом полного перебора нужно было бы проверить все возможные конфигурации роторов и все возможные места в последовательности шагов, т. е. всего рассмотреть  $256^3 \times 1430$ , или  $2.40 \times 10^{10}$  случаев. Это все еще практически осуществимо, но займет не несколько секунд, а несколько часов.

При условии что роторы и последовательности шагов известны, этот 3-роторный шифр получает оценку 4.



Если роторы и последовательности шагов неизвестны, то Эмили придется прибегнуть к общим методам вскрытия многоалфавитных шифров, т. е. перехватить достаточное количество сообщений и сопоставить их с целью найти участки, зашифрованные с одними и теми же параметрами. Чтобы отделить истинные совпадения от случайных, нужно посчитать индекс совпадения (раздел 5.7) на длинных перекрытиях. Я рекомендую не менее 200 символов. Пытаться сопоставить имеет смысл только сообщения, содержащие более 200 символов. Для сообщения длины  $L \geq 200$  число позиций, где возможно совпадение, равно  $L - 199$ . Когда общее число  $M$  допускающих сопоставление позиций во всех перехваченных сообщениях превысит  $\sqrt{2.40 \times 10^{10}} = 1.55 \times 10^5$ , можно приступить к поиску соответственных участков текста.

На первый взгляд, это немного, но объем работы по нахождению перекрытий имеет порядок  $M^2$ . Кроме того, одного перекрытия совершенно недостаточно. Нужно много перекрытий, чтобы можно было начать выделение высокочастотных букв и отделение гласных от согласных. Для такой работы нужен большой компьютер и группа талантливых криптоаналитиков. Трехроторная машина с неизвестными роторами и последовательностями шагов получает оценку 6.

### 5.12.3 Восьмироторная машина

Три ротора – неплохое начало. Но чтобы добиться по-настоящему высокой стойкости смоделированной роторной машины, увеличим число роторов с 3 до 8. Пусть роторы могут поворачиваться на 11, 13, 17, 19, 23, 25, 27 и 31 шаг, а сумма шагов каждого ротора нечетна. Период такой машины равен приблизительно  $5.69 \times 10^{12}$ .

Если это аппаратное устройство, то внутренняя электрическая схема роторов и последовательности шагов могут быть встроены. Даже если это так, все равно практически невозможно провести сопоставление сообщений, как мы делали это для 3-роторной машины. Действительно, теперь имеется  $256^8 = 1.84 \times 10^{19}$  возможных начальных позиций восьми роторов. Поскольку период равен  $5.69 \times 10^{12}$ , полное число состояний машины оказывается равным  $(1.84 \times 10^{19}) \times (5.69 \times 10^{12}) = 1.05 \times 10^{32}$ . Восьмироторная машина с неизменяемыми роторами и последовательностями шагов получает оценку 9.

Пойдем дальше. Допустим, что имеется не 8, а 16 роторов. Для каждого сообщения выбирается 8 роторов из 16 возможных в некотором порядке. Всего таких перестановок  $5.19 \times 10^8$ . Для каждой перестановки имеется  $1.84 \times 10^{19}$  возможных начальных позиций роторов и  $5.69 \times 10^{12}$  позиций в последовательностях шагов. Всего получается  $5.43 \times 10^{40}$  состояний.

Даже если Эмили каким-то образом узнает таблицы подстановки и последовательности шагов для всех 16 роторов, она не сможет вскрыть сообщение, зашифрованное такой машиной, даже восполь-

зовавшись самым большим и быстрым суперкомпьютером. (На момент написания книги самым быстрым в мире суперкомпьютером был Summit с быстродействием 200 петафлоп.) Такой 8-роторный шифр получает оценку 10.

Если таблицы подстановки и последовательности шагов хранятся в секрете или часто изменяются, то такой роторный шифр с 8 взаимозаменяемыми роторами будет неприступен для самых больших суперкомпьютеров еще 10, 20 или, быть может, даже 30 лет.

Поскольку мы моделируем роторную машину программно, роторы можно изменять, как нам будет угодно. Вместо фиксированного набора 16 роторов роторы можно было бы изменять для каждого сообщения, используя ключ для перемешивания всех восьми роторных алфавитов. Это значительно повысило бы безопасность ценой отдельного этапа инициализации для каждого сообщения. Промежуточный уровень безопасности дает использование семи стандартных роторов из шестнадцати и одного ротора, алфавит которого генерируется независимо для каждого сообщения. Это уменьшает время инициализации на 87 %.

Хотя этот шифр уже заслужил оценку 10, у вас все равно может возникнуть желание укрепить его. Возможно, вы не доверяете моим оценкам или думаете, что противник располагает невообразимой вычислительной мощностью. Один из возможных путей – использовать выход некоторых роторов для модификации работы. Я предлагаю взять выход четвертого ротора в середине процесса шифрования и использовать этот символ для продвижения первого ротора. Можно использовать символ непосредственно, а можно произвести для него простую подстановку, чтобы получить количество позиций, на которое должен провернуться ротор. Для всех символов сообщения, кроме первого, первый ротор провернется дважды: один раз – на количество позиций, заданное в его собственной последовательности шагов, а другой – в результате обратной связи с четвертым ротором.

Такой двойной шаг не оказывает влияния на шифрование текущего символа. Модифицированная конфигурация применяется к шифрованию следующего символа. Возможно, реализовать эту идею в аппаратной роторной машине трудно, но в компьютерной модели это делается легко, т. к. роторы моделируются по одному.

Кстати говоря, может показаться, что шифр получился бы более стойким, если использовать выход восьмого ротора, но это не так. На выходе восьмого ротора мы имеем символ шифртекста, который станет известен подслушивающему. Выходы двух средних роторов, т. е. четвертого и пятого, наименее уязвимы для подслушивания, а потому являются самыми безопасными.

Обратная связь с четвертым ротором делает смоделированную 8-роторную машину апериодической. Сколько бы сообщений ни было перехвачено, Эмили не сможет найти два сообщения с одной и той же последовательностью конфигураций роторов. \*\*

# Контрмеры

## **Краткое содержание главы:**

- двойное шифрование;
- null-символы и null-биты;
- омофоны;
- сокрытие сообщений внутри изображений или файлов.

В разделе 5.9 отмечалось, что многоалфавитный шифр можно вскрыть в результате двухшагового процесса. На первом шаге с помощью метода Касиски или индекса совпадения определяется длина ключа. При этом шифртекст разбивается на несколько участков, каждый из которых зашифрован всего одной буквой ключа. На втором шаге эти участки дешифрируются стандартными методами, применяемыми для простых подстановочных шифров. Я имею в виду анализ частот и контактов.

А теперь взглянем на это с другой стороны. Что может сделать криптолог, чтобы помешать такому вскрытию многоалфавитного шифра? Мы рассмотрим несколько контрмер.

## 6.1 Двойное шифрование

Если сообщение зашифровано одним многоалфавитным шифром с периодом  $P$ , а получившийся на выходе промежуточный текст зашифрован вторым многоалфавитным шифром с периодом  $Q$ , то результат эквивалентен многоалфавитному шифру, период которого равен наименьшему общему кратному  $P$  и  $Q$ , которое обозначается  $lcm(P, Q)$ . Иначе говоря, период равен наименьшему целому числу, делящемуся на  $P$  и на  $Q$ . Например, если  $P = 10$  и  $Q = 11$ , то двойной шифр будет иметь период 110, но если  $P = 10$  и  $Q = 12$ , то двойной шифр будет иметь период 60, потому что 60 делится на 10 и на 12.

Каждый алфавит, участвующий в двойном шифровании, является композицией двух алфавитов, взятых из первого и второго шифров, как описано в разделе 11.7.4. Если это сдвинутые стандартные алфавиты, то результат также будет сдвинутым стандартным алфавитом. Если это перемешанные алфавиты, то результирующий алфавит, вероятно, будет перемешан более тщательно.

Хотя двойное шифрование по-прежнему многоалфавитное, оно может оказаться более стойким, чем одинарное, потому что период больше, а число букв, шифруемых каждым символом ключа, меньше. Такой тип двойного шифрования получает оценку 3.

Если каждый из двух многоалфавитных шифров имеет автоключ или бегущий ключ, то двойной шифр является шифром с бегущим ключом. Однако ключ не будет англоязычным текстом, поэтому техника волочения из раздела 5.11 неприменима. Зато для работы с двумя бегущими ключами можно приспособить вероятностную технику, описанную там же.

Если при шифровании используются неперемешанные алфавиты, т. е. таблица Беласо, то порядок применения шифров не играет роли. Если сначала зашифровать сообщение  $M$  бегущим ключом  $R$ , а затем перешифровать его бегущим ключом  $S$ , то результат будет таким же, как если бы мы зашифровали ключ  $R$  ключом  $S$ , а затем применили получившийся составной ключ  $C$  к шифрованию сообщения  $M$ .

Ключи, полученные путем шифрования одного бегущего ключа другим бегущим ключом, не являются случайными. У них имеется собственное характерное распределение частот букв и контактов. Существуют типичные последовательности, например слово  $TNE$ , зашифрованное ключом  $TNE$ , или слово  $AND$ , зашифрованное ключом  $TNE$ . Все это можно заранее вычислить и сохранить в таблице. Если проволочь длинную фразу, например  $UNITED STATES OF AMERICA$  или  $NEGOTIATING STRATEGY$ , по тексту, то можно будет искать участки бегущего ключа, отвечающие этому распределению. Поэтому шифрование двойным бегущим ключом можно вскрыть с помощью компьютера.

Шифрование композицией двух автоключей и (или) бегущих ключей с неперемешанными алфавитами получает оценку 4. Если же алфавиты хорошо перемешаны, то оценка повышается до 6.

## 6.2 Null-символы

Null-символы – старый и заслуженный способ расстроить планы взломщика кодов. Их еще в XV веке использовала семья Ардженти. Так называются бессмысленные символы, которые вставляются в сообщение, специально чтобы запутать вражеских криптоаналитиков. Чаще всего они используются в кодах. В случае многоалфавитных шифров они могут искажать счетчики частот и препятствовать анализу методами Касиски и индекса совпадения.

Null-символы можно использовать несколькими способами. Самый простой – добавить null-символ в алфавит. Обычно его представляют звездочкой \*. Затем этот символ можно вставлять в разные места открытого текста. Но не слишком часто, иначе противник его заметит и раскроет уловку. Достаточно от 3 до 6 процентов null-символов. Полезно вставлять их в высокочастотные слова, чтобы помешать атаке Касиски. Но делать это нужно случайным образом. Если заменить *каждое* вхождение TNE на T\*NE, то вы только поможете Эмили, предоставив ей 4-значные повторения. Лучше в половине случаев использовать TNE, в четверти T\*NE и еще в четверти TH\*E. Бессмысленно использовать \*TNE или TNE\*, потому что триграмма TNE при этом остается.

Таким образом, в таблице алфавитов будет 27 столбцов, а в шифр-тексте будут встречаться звездочки. Вы, наверное, подумали, что так мы выдадим факт использования null-символов, но существует трехчастный шифр Trifid, описанный в разделе 9.9, в котором используется 27-символьный алфавит. Эмили могла бы подумать, что ваш многоалфавитный шифр трехчастный (не путать с трехногими растениями-монстрами из фантастического романа Джона Уиндэма «День триффидов», вышедшего в 1951 году).

Но такое использование null-символов почти не укрепляет шифр. Частоты букв меняются не сильно, а эффективность методов Касиски и индекса совпадения практически не снижается. Метод получает оценку 3.

Другой способ использования null – вставлять в открытый текст специальные null-последовательности букв. Они должны быть легко распознаваемы. Я рекомендую строить null-последовательности из небольшого числа среднечастотных букв, например C, D и P. Для представления букв C, D и P можно было бы взять биграммы CC, DD и PP, а остальные шесть биграмм, составленные из этих букв, считать null-последовательностями. Этот метод также получает оценку 3.

## 6.3 Прерванный ключ

Более эффективный способ – вставлять null-символы в шифртекст, так чтобы они прерывали повторяющийся период. Простая реали-

зация этой идеи заключается в том, чтобы сначала зашифровать сообщение многоалфавитным шифром, как обычно, затем при каждом возникновении в шифртексте *триггерного* события, например вхождении заранее выбранной буквы или биграммы, вставлять после него null-символ. Null-символом может быть любая буква или даже биграмма. О том, что это именно null-символ, свидетельствует предшествующий ему триггер.

Эту идею можно развить. Можно было бы вставлять null-символ:

- через 4 символа после каждой буквы W шифртекста, тогда шифртекст **NPGWSOVKLEWPIDF** превратился бы в **NPGWSOVTKLEW-PIDCF**;
- после каждой второй буквы H шифртекста;
- после первой буквы A, следующей за каждой буквой Q шифртекста;
- в первой позиции после первой буквы V, во второй позиции после следующей буквы B, в третьей позиции после следующей буквы L, далее цикл V, B, L, V, B, L, ... повторяется;
- после каждой удвоенной буквы шифртекста, или после трех последовательных гласных, или после четырех букв, следующих друг за другом в алфавитном порядке по возрастанию или по убыванию.

Возможны также комбинации всего вышеперечисленного. Число вариантов ограничено лишь вашим воображением. Только не надо усложнять до такой степени, что Сандра и Рива не смогут быстро и точно зашифровывать и расшифровывать текст. Если Сандра должна вставлять null-символ после каждой второй K и после каждой третьей M, но по ошибке вставит его после четвертой M, то Рива не сможет расшифровать сообщение.

Этот метод вставки null-символов получает оценку 4 при использовании стандартных алфавитов и 5, когда алфавиты хорошо перемешаны. Как всегда, предполагается, что перемешанные алфавиты хранятся в секрете.

Есть еще несколько способов прервать периодическое повторение ключа. Один из них – применять ключ с начала, если в открытом тексте возникло какое-то триггерное событие. Это безопаснее, чем предыдущий метод, когда триггер ищется в шифртексте, потому что Эмили видит шифртекст, а открытый текст не видит. С другой стороны, это затрудняет работу полномочного получателя. Если триггеры присутствуют в шифртексте, то Риве достаточно просто посмотреть его и удалить null-символы. Если же триггеры находятся в открытом тексте, то Рива должна дешифрировать его по одному символу и искать триггеры.

Сами триггеры в открытом тексте могут походить на рассмотренные выше триггеры в шифртексте. После обнаружения триггера можно было бы предпринять следующие действия:

- пропустить заданное число символов ключа;
- повторить заданное число символов ключа;
- начать применение ключа с первого символа;
- поменять направление движения по ключу.

Ниже приведены примеры всех четырех типов прерывания ключа SAMPLE, когда триггером служит буква A:

Пропустить 2	Повторить 1	Начать сначала	Сменить направление	
<u>SAMPLE</u>	<u>SAMPLE</u>	<u>SAMPLE</u>	<u>SAMPLE</u>	Ключ
MA..RY	MA....	MA....	MA....	Открытый текст
HA..DA	.RYHA.	RYHA..	YR....	
..LITT	....DA	DA....	....AH	
LELA..	.....L	LITTLE	....DA	
MBITSF	ITTLEL	LA....	ELTTIL	

Эта форма прерывания ключа получает оценку 5 при использовании стандартных алфавитов и 8, когда алфавиты хорошо перемешаны. Ключ не следует перезапускать с начала слишком часто, иначе первый символ ключа будет задействован чрезмерно, а последний окажется в небрежении.

Более стойкая форма прерывания ключа – использовать два отдельных ключа разной длины. Наибольшая стойкость достигается, когда длины ключей – взаимно простые числа. После триггера мы переключаемся с одного ключа на другой. Этот метод получает оценку 6 при использовании хорошо перемешанных алфавитов. Ниже приведен пример для ключей FIRST и SECOND и триггера A:

<u>FIRST</u>	<u>SECOND</u>	Два ключевых слова
MA...	RYHA..	Открытый текст, показывающий,
		какая буква ключа используется
..DA.	....LI	Шифртекст не показан
.....	TTLELA	
....M	.....	
BITSF	.....	
LEECE	.....	
WA....	SWHITE	

В этом шифре запоминается, какая буква ключа использовалась последней. При переключении на другой алфавит шифрование продолжается со следующей буквы ключа. Например, открытый текст МА зашифрован буквами ключа FI, поэтому следующей буквой является R. После зашифровывания RYHA буквами SECO второго ключа возобновляется шифрование первым ключом, а именно буквами RS.

Таким образом, все буквы каждого ключа используются примерно одинаковое число раз.



## 6.4 Омофоническая подстановка

Идея омофонической подстановки, с которой мы познакомились в разделе 4.2, – использовать несколько подстановок вместо каждой буквы открытого текста, чтобы сгладить частоты букв. Чаще всего алфавит шифртекста расширяется путем включения дополнительных подстановок. Поскольку в классических многоалфавитных шифрах используется 26-буквенный алфавит, по крайней мере для английского языка, омофонические подстановки в них обычно не применяются.

При реализации на компьютере с 8-битовыми байтами омофоническая подстановка не представляет никаких сложностей. Каждый байт может принимать 256 различных значений. 26 заглавных букв, 26 строчных букв, 10 цифр и, возможно, 32 знака препинания занимают всего 94 символа. Ну, пусть будет 98, если мы хотим включить еще и управляющие символы: табуляцию, забой, перевод строки и возврат каретки. Остается 158 символов, которые можно использовать для null-символов, биграмм, триграмм и прерывания ключа.

Рассмотрим реализацию омофонической подстановки с помощью карандаша и бумаги на примере обычной таблицы перемешанных алфавитов  $26 \times 26$ . Если в качестве триггера зарезервировать одну букву, то эта буква будет встречаться так часто, что ее легко заметить. С двумя триггерными буквами дело обстоит так же. Я рекомендую брать 3 триггерные буквы, каждую с частотой меньше 4 %. Назовем этот шифр *Trig3*. Подходят буквы BCDFGJKLMPQUVWXYZ. Выберем для определенности B, C и D. Существует  $3 \times 26 = 78$  биграмм, начинающихся с этих букв. Использовать биграммы, содержащие высокочастотные буквы AEINORST, не следует, потому что это противоречит нашему стремлению сгладить частоты букв. Остается 54 биграммы, которые могут использоваться в роли null-символов, офомонов и для прерывания ключа. Вот один из возможных наборов:

BB O	BM T	CB +2	CM O	DB N	DM B
BC RE	BP A	CC -	CP S	DC -	DP IN
BD -	BQ +3	CD C	CQ E	DD T	DQ R
BF N	BU E	CF I	CU +2	DF +1	DU -
BG R	BV ON	CG D	CV I	DG +2	DV S
BH E	BW E	CH A	CW ER	DH AN	DW -
BJ +1	BX R	CJ -	CX +3	DJ -	DX I
BK O	BY +1	CK +3	CY -	DK S	DY T
BL -	BZ T	CL E	CZ A	DL TH	DZ N

Здесь - представляет null-символ, т. е. открытый текст BD, BL, CC и т. д. – null-символы. Коды +1, +2 и +3 – прерыватели-ключи, означающие, что нужно пропустить 1, 2 или 3 буквы ключа соответственно. Множество офомонов состоит из шести биграмм: AN, ER, IN, ON, RE и TH, а также одиночных букв.



Важно следить за балансом. Если увлечься использованием этих подстановок, то у букв В, С и D окажется слишком большая частота, и их легко будет опознать как триггеры. Если, наоборот, прибегать к ним редко, то не будет полезного эффекта. 10 % – это нормально, причем биграммы с буквами В, С и D должны использоваться примерно с равной частотой, около 3 % каждая. Помните, что использовать буквы В, С и D в роли самих себя теперь нельзя, нужно подставлять вместо них DM, CD и CG.

При правильном применении и с хорошо перемешанными алфавитами шифр Trig3 получает оценку 5.

### 6.4.1 Шифр 5858

Прежде чем переходить к подстановке биграмм, я хотел бы представить еще один шифр, который назову *шифр 5858*. В этом компьютерном шифре используются 5-битовые символы. Пять бит дают алфавит из 32 символов, достаточный для 26 букв, 3 null-символов и 3 омофонов. (1) Открытый текст записывается в виде последовательности 5-битовых символов с использованием перемешанного алфавита. (2) Вставляются null-символы и омофоны, каждый в пропорции примерно 3 %, т. е. всего они будут занимать 18 % открытого текста. Лучше расставлять их случайно, без всякой системы. (3) При необходимости открытый текст дополняется до длины, кратной 8, null-символом и случайными битами числом не более 4. (4) Дополненное сообщение рассматривается как строка 8-битовых байтов, и для нее производится хорошо перемешанная подстановка. Например, если сообщение содержало восемьдесят 5-битовых символов, то эти 400 бит можно рассматривать как пятьдесят 8-битовых байтов. (5) Сообщение снова трактуется как строка 5-битовых символов. Три из них назначаются прерывателями ключа +1, +2 и +3, как в шифре Trig3. (6) Сообщение зашифровывается общим многоалфавитным шифром с хорошо перемешанной таблицей 5-битовых алфавитов размера  $32 \times 32$ . (7) Затем строка 5-битовых символов разбивается на 8-битовые байты и производится вторая 8-битовая подстановка. Таким образом, в шифре 5858 выполняется четыре подстановки: начальная 5-битовая подстановка, 8-битовая подстановка, общая 5-битовая многоалфавитная подстановка и окончательная 8-битовая подстановка. Этот шифр получает оценку 7.

## 6.5 Подстановка биграмм и триграмм

Еще один способ помешать Эмили использовать частоты букв и контактов для вскрытия шифра – выполнять подстановки биграмм и даже триграмм. Проще всего сделать это с помощью таблицы. Для биграмм используются таблицы  $26 \times 26$ , каждый элемент которой является биграммой:

	A	B	C	D	E	F	G	...
A	BL	TC	UB	NK	RA	KS	BW	...
B	CA	CS	FN	GX	OD	MH	YL	...
C	PS	DE	YO	UJ	BK	GC	NZ	...
...								

Вместо AA подставляется **BL**, вместо AB – TC и т. д. Для подстановки триграмм следует использовать буклет из 26 таблиц, по одной для каждой первой буквы триграммы.

Подстановки такого типа можно использовать самостоятельно или в сочетании с каким-нибудь другим методом, например многоалфавитной подстановкой. В первом случае подстановка биграмм получает оценку 3, а подстановка триграмм – оценку 4. Подстановке биграмм с последующей многоалфавитной подстановкой с секретными хорошо перемешанными алфавитами я ставлю оценку 5, а подстановке триграмм с многоалфавитной подстановкой с хорошо перемешанными алфавитами – оценку 6.

## \*6.6 Соккрытие сообщений в изображениях

В 1999 году была высказана интересная идея – скрывать сообщения внутри компьютерных файлов данных разных типов. Это современный вариант стеганографии (раздел 2.2). Рассмотрим один такой метод – соккрытие сообщений в растровом изображении, т. е. файле формата BMP. Растровые изображения хранятся попиксельно. В самом распространенном растровом формате каждый пиксель представлен тремя байтами, описывающими доли синего, зеленого и красного цветов в этой точке изображения. (Этот не зависящий от устройства порядок определен в стандарте растровых изображений, принадлежащем Microsoft. Если трудно запомнить порядок, заметьте, что названия цветов Blue (синий), Green (зеленый) и Red (красный) следуют в алфавитном порядке.) Например, 0,0,0 означает, что цвета нет, т. е. это чисто-черный пиксель; 255,255,255 означает максимальную насыщенность каждого цвета, т. е. белый цвет, а 255,0,0 – чисто-синий.

Значения пикселей обычно записываются в шестнадцатеричном виде, так что чисто-синий цвет представляется числом FF0000. В некоторых языках программирования это число записывается в виде \$FF0000, X'FF0000' или даже 0xFF0000, поскольку 255 в шестнадцатеричной системе равно FF. Иногда порядок цветовых компонент записывается наоборот. Так, в HTML чисто-синий цвет записывается в виде #0000FF.

Изображение может содержать сотни или тысячи строк, а каждая строка содержит сотни или тысячи пикселей. Не редкость растровые изображения размером 3000 строк по 4000 пикселей. В таком изображении будет 12 000 000 пикселей, и оно будет занимать

36 000 000 байт в памяти плюс 54 байта заголовка. Именно поэтому изображения с высоким разрешением так быстро заполняют память компьютера.

Идея сокрытия заключается в том, чтобы использовать младший байт каждой цветовой компоненты для хранения одного бита сообщения. Это может остаться незамеченным, потому что разница между FF0000 и FE0000 или даже FE0101 зрительно едва уловима. А в большом изображении изменение одного пикселя практически невозможно обнаружить. Кроме того, в половине пикселей значение младших битов вовсе не изменяется. Если сообщение скрыто в изображении, то важно, чтобы содержащий его файл передавался точно. Изображение нельзя увеличивать, уменьшать, кадрировать, поворачивать, наклонять, сжимать или преобразовывать в другой формат.

Сообщение может быть зашифровано любым методом. Но если Эмили заподозрит, что сообщение скрыто таким способом, то никакой дополнительной секретности мы не добьемся. Заплатив за передачу 8 бит на каждый бит сообщения, мы не получим никакого выигрыша. Нужно лишь отбирать младшие биты каждого пикселя, так что оценка этого метода такая же, как у самого метода шифрования сообщения.

Чтобы эта схема дала улучшение безопасности, нужно использовать не все биты, а выбирать только некоторые биты из каждого пикселя в циклическом порядке. Для этого воспользуемся строкой восьмеричных цифр (см. таблицу в разделе 3.1), например 1, 3, 7, 4, 6, как ключом для выборки битов сообщения. Можно назвать его *ключом выборки*. Мы имеем 5 восьмеричных цифр и, стало быть, 15 бит выборки. Начинаем с первого пикселя изображения и с первой цифры ключа выборки. Если первый бит этой цифры равен 1, то помещаем один бит сообщения в младший бит синей компоненты пикселя, иначе случайным образом полагаем младший бит равным 0 или 1. Если второй бит ключа равен 1, проделываем то же самое для зеленой компоненты, а если третий бит равен 1, то для красной. Повторяем те же действия для второго пикселя и второй цифры ключа выборки. И так далее.

Может возникнуть мысль, что если бит ключа равен 0, то лучше оставить соответствующий бит изображения без изменения. Тогда искажений было бы внесено меньше и Эмили было бы труднее определить, что оно содержит скрытое сообщение. Так-то оно так, но если Эмили заподозрит, что используется такой метод, то ей будет проще узнать ключ выборки.

Рассмотрим эту ситуацию подробнее. Предположим, что Эмили перехватила сообщение, содержащее растровое изображение. Также предположим, что Эмили произвела поиск в интернете и нашла исходное изображение. Ничто не мешает ей сравнить оба изображения попиксельно и составить карту отличий. Если в каком-то пикселе младшие биты совпадают, то Эмили ставит в карте X, а если раз-

личаются, то |. Затем Эмили пробует разные длины ключа выборки. Пусть выбрана правильная длина L; если расположить участки отметок такой длины друг под другом, то каждый столбец, для которого бит ключа выборки равен 0, будет содержать только X, тогда как столбцы, соответствующие биту 1, будут состоять наполовину из X и наполовину из |. Например, если ключ выборки равен 1, 3, 7, 4, 6, то можно увидеть такую картину:

<u>001</u>	<u>011</u>	<u>111</u>	<u>100</u>	<u>110</u>	Ключ выборки
xxx	x x	xx	xxx	x x	Карта различий
xxx	xx	xx	xxx	x x	
xx	x	xx	xxx	xx	
xxx	xxx	x	xx	xxx	
xx	xx	x x	xx	x x	

Для каждого столбца, содержащего |, соответствующий бит ключа выборки должен быть равен 1. Все остальные биты ключа выборки, вероятно, равны 0. Чем больше в таблице различий, тем эта вероятность выше.

Поэтому всякий раз, как бит выборки оказывается равным 0, младший бит цветовой компоненты следует задавать случайным образом. Использование циклического ключа выборки в этом методе сокрытия сообщений увеличивает на 2 оценку базового шифра, если она находится в диапазоне от 1 до 4, и на единицу – если в диапазоне от 5 до 8.

Ключ выборки также можно было бы сгенерировать с помощью цепного генератора псевдослучайных цифр из раздела 4.5, взяв подходящее значение из 7, 9 или 10 цифр. Используем сгенерированные цифры от 0 до 7 в качестве цифр ключа выборки. Если сгенерирована цифра 8 или 9, отбрасываем ее и переходим к следующей. Здесь не важно, удовлетворяют ли псевдослучайные числа статистическому критерию случайности. Важно лишь, чтобы длина последовательности сгенерированных цифр была больше длины сообщения, измеренной в битах, тогда Эмили не сможет сопоставить участки шифртекста с одинаковым ключом выборки.

При использовании цепного генератора цифр ключа выборки этот метод сокрытия сообщения прибавляет 3 к оценке базового шифра, если она находится в диапазоне от 1 до 4, 2 – если в диапазоне от 5 до 7, и 1 – если оценка равна 8. \*\*

## 6.7 Добавление null-битов

Идею перемешать биты сообщения с null-битами также можно реализовать, не встраивая сообщение в изображение или иной файл, и сделать это можно вручную. Сначала зашифруем сообщение простым подстановочным шифром или еще каким-то методом. Предста-



дать префиксным свойством, иначе Рива не сможет расшифровать сообщение.

Есть два способа избежать добавления лишних символов для второй подстановки – с заменой битов буквами. (1) Можно объединять биты в группы по 4 и использовать только 16 букв алфавита или 16 шестнадцатеричных цифр и (2) применять код переменной длины с шестью 4-битовыми кодовыми группами и двадцатью 5-битовыми кодовыми группами.

Как и раньше, эти кодовые группы должны обладать префиксным свойством. Приведу пример:

E 0101	S 00010	C 01000	P 01111	J 01110	4-битовая/5-битовая подстановка
T 1011	H 11011	M 11010	B 11110	Z 00001	
A 1110	R 10000	F 10010	V 10001		
O 0011	D 00011	Y 00101	K 11001		
I 0110	L 11000	W 11111	X 00000		
N 1010	U 01001	G 00100	Q 10011		

Обратите внимание, что я использовал 4-битовые подстановки для 6 букв с наибольшей частотой. В результате эти 6 букв будут иметь в шифртексте примерно вдвое большую частоту, чем остальные 20. Это могло бы подтолкнуть неосмотрительного противника к мысли о том, что используется шифр совершенно другого типа.

Техника добавления null-битов применима ко многим типам шифров. Поскольку null-бит неотличим от любого другого, их добавление, с точки зрения повышения стойкости, лучше, чем добавление null-символов. Оно может прибавить к оценке шифра целых 3 балла.

Рассмотрим конкретный пример шифра, который назовем *Null5*. Как и раньше, имеются три шага: преобразование из букв в биты, добавление null-битов и обратное преобразование из битов в буквы. Буквы преобразуются в биты с использованием омофонической подстановки букв в группы из 5 бит. Для каждой из букв E, T, A, O, I, N предлагается две подстановки. Null-биты вставляются с помощью ключа выборки, как в разделе 6.6. Биты преобразуются в буквы с помощью 4- и 5-битовых подстановок, как в таблице выше.

Шифр Null5 получает оценку 6.

## 6.8 Объединение нескольких сообщений

Двоичную форму ключа можно также использовать как ключ для объединения двух сообщений. Это означает, из двух сообщений образуется одно, в котором биты исходных сообщений чередуются. Ключ объединения, записанный в системе счисления по основанию 3 или 4, можно использовать для объединения трех или четырех сообщений соответственно. Ключ в системе по основанию 4 можно использовать для объединения трех сообщений со вставкой null-битов.

У объединения нескольких сообщений есть два преимущества: длина увеличивается не так сильно, как при использовании null-битов, и можно использовать более простой и быстрый метод шифрования. Если чередовать 4 сообщения с помощью длинного ключа объединения и использовать разные простые подстановки для каждого сообщения, то только это позволяет присвоить шифру оценку 5. А если еще применить дополнительную простую подстановку к объединенному сообщению, то оценка повышается до 8.

Ключи для объединения сообщений бывают двух видов: счетчика битов или выборки. В случае счетчика битов сообщения перебираются циклически. Каждая цифра ключа определяет, сколько битов брать из очередного сообщения. В случае выборки сообщения можно перебирать в любом порядке, но из каждого сообщения берется только один бит. Цифра ключа определяет, из какого сообщения брать следующий бит. Примеры ниже показывают, как сообщения **010101111010001101011** и **11101010011011100110** объединяются методом счетчика битов с ключом 123123 и методом выборки с ключом 12122112.

12 3 12 3 12 3 12 3 12 3 12 3 12 3	Ключ счетчика битов
0 101 01 1 110 10 0 011 01 0 1	Сообщение 1
11 1 010 10 0 110 11 1 001 10	Сообщение 2
0111011010101101100101100110111010010101	Шифртекст
<b>12122112</b> 12122112 <b>12122112</b> 12122112 <b>12122112</b>	Ключ выборки
0 1 01 0 1 11 1 0 10 0 0 11 0 1 01	Сообщение 1
1 11 0 1 01 0 0 11 0 1 11 0 0 11 0	Сообщение 2
0111101001101110100111000101111000111010	Шифртекст

Объединение нескольких сообщений может оказаться муторным, когда длины сообщений различны. Необходим маркер конца для всех сообщений, кроме самого длинного. Другой способ справиться с неравными длинами – сбалансированное объединение. Сначала выпишем все сообщения подряд, разделив их каким-то зарезервированным символом или последовательностью символов. Затем просто разобьем эту длинную строку на равные части. Например, если сообщения содержат 50, 60, 70 и 80 символов, то длина объединенного сообщения равна 260 символов плюс 3 разделителя – итого 263 символа. Это сообщение можно разбить на три участка длиной 66, 66, 66 и 65 символов. Если используются 8-битовые байты, то можно разбить  $263 \times 8 = 2104$  бита на 4 участка по 526 бит. Необязательно разбивать битовую строку по границам байтов, но ключи нужно выбирать так, чтобы брать равное число битов из каждого участка.

Кстати говоря, количество сообщений и количество участков независимы. Сообщений может быть одно или много, а участков – два и более. Например, одно сообщение можно разбить на три участка или три сообщения разбить на два участка.



Балансировка – это только половина решения. Еще надо подумать о том, что делать в конце процесса объединения. Рано или поздно наступит такой момент, когда все биты из одной строки уже включены в объединенное сообщение, тогда как в других строках еще остались биты. Если ключ объединения требует, чтобы был включен бит из уже исчерпанной строки, просто пропустите ее и переходите к выбору следующего бита.

Соберем все сказанное выше в одном шифре, который назовем *Merge8*. Он применяется к одному или более сообщениям, которые предварительно были конкатенированы, т. е. записаны подряд, одно за другим. В варианте *base-26* все 26 букв преобразуются в двоичную форму с применением 4-битового/5-битового кода, как при 4-битовой/5-битовой подстановке из раздела 6.7. Разделителем сообщений может быть любая последовательность букв, например XXX или END. В версии *base-256* хорошо перемешанная простая подстановка применяется к ASCII-кодам. Получившаяся битовая строка разбивается на 8 участков равной длины. Для объединения 8 участков используется ключ, состоящий из 32 восьмеричных цифр. Каждая из 8 восьмеричных цифр встречается в ключе 4 раза, так что всего существует  $32!/(4!)^8 = 2.39 \times 10^{24}$  возможных ключей объединения. К получившейся строке применяется вторая простая подстановка. И в версии *base-26*, и в версии *base-256* используется 8-битовая подстановка. Шифр *Merge8* получает оценку 6.

## 6.9 Внедрение сообщения в файл

Когда сообщение скрыто в файле изображения, приходится использовать как минимум 7 бит изображения на каждый бит сообщения. Это крайне неэффективно. Можно скрыть в файле гораздо больше битов, если не пытаться сделать его похожим на что-то другое. Это шифртекст, так пусть он и выглядит как шифртекст. Существует масса способов скрыть открытое сообщение внутри файла. Назову лишь несколько. Как и в разделе 6.6, для каждого байта файла некоторое число битов открытого текста скрыто среди 8 бит.

- Каждый раз берется фиксированное или переменное число битов открытого текста.
- Биты помещаются в фиксированные или переменные позиции внутри каждого байта.
- Биты помещаются по порядку или переставляются.
- Биты открытого текста вставляются в неизменном виде или шифруются несложным шифром, например простым подстановочным.
- Биты шифртекста оставлены «как есть» или зашифрованы несложным шифром, например простым подстановочным.

Количество, позиции и порядок битов могут задаваться периодическим ключом или какой-то случайной последовательностью.



Оценка шифров из этого класса зависит от параметров и может варьироваться от 1 до 10.

Приведу пример такого шифра. (1) Применить к сообщению хорошо перемешанную ключом простую подстановку. (2) С помощью генератора псевдослучайных чисел с большим внутренним состоянием выбрать от 2 до 6 позиций в каждом байте. Поместить следующие биты сообщения в количестве от 2 до 6 по порядку в эти позиции. В остальные биты записать случайные значения. (3) Применить вторую хорошо перемешанную ключом простую подстановку.

Этот метод, называемый *EmbedBits*, очень прост и работает исключительно быстро. Недостаток в том, что шифртекст получается примерно в два раза длиннее открытого текста. Шифр *EmbedBits* получает оценку 8. Чтобы повысить ее до 10, замените подстановку одиночных символов подстановкой биграмм, например как в шифре *Two Square*.

# 7

## Перестановка

---

### *Краткое содержание главы:*

- маршрутная и столбцовая перестановка;
- перестановка со случайными числами;
- перестановка с ключом;
- множественные анаграммы.

В главах 5 и 6 мы рассматривали подстановочные шифры. Вторая большая категория методов шифрования с секретным ключом – перестановочные шифры. Под перестановкой понимается изменение порядка элементов сообщения. Элементами могут быть слова, слоги, буквы и отдельные цифры или биты, представляющие буквы. В этой главе мы будем иметь дело преимущественно с перестановкой букв, но помните, что те же самые методы применимы и к другим элементам, например к перестановке слов, описанной в разделе 7.2.2. Мы рассмотрим много разных типов перестановочных шифров. По большей части их можно реализовать, имея только бумагу и карандаш.

### 7.1 *Маршрутная перестановка*

Маршрутная перестановка – простейший и самый старый из перестановочных шифров. Никакого ключа в нем нет. Секретность обеспечивается выбором маршрутов, или путей.

Маршрутная перестановка – действенный способ заинтересовать криптографией ребенка. Это замечательное развлечение в классе, в лагере отдыха и на других тусовках. Главное – научить детей писать буквы ровно по столбцам, иначе прочесть сообщение правильно не получится. Выручит миллиметровка с широким шагом.

Основная идея заключается в том, что сообщение записывается в прямоугольник с использованием одного маршрута, а читается с использованием другого. Например, для сообщения длиной 30 символов лучше всего взять прямоугольник  $5 \times 6$ . Если в сообщении 29 символов, просто добавьте null-символ. Дополняйте сообщение null-символами, так чтобы оно поместилось в прямоугольник подходящего размера. Полезно расчертить прямоугольник, прежде чем приступить к его заполнению. Длинное сообщение разбейте на блоки удобного размера. Например, сообщение длиной 1000 символов можно разбить на 20 блоков  $5 \times 10$ .

Мы уже видели пример маршрутной перестановки в разделе 4.3. Сообщение записывалось в сетку размером  $5 \times 5$  по горизонтали слева направо, а считывалось по вертикали сверху вниз. Горизонталь и вертикаль – два типа маршрутов. Но не единственные, ниже приведен более полный перечень.

- По горизонтали слева направо, справа налево или с чередованием направлений.
- По вертикали сверху вниз, снизу вверх или с чередованием направлений.
- По диагонали из верхнего левого угла в правый нижний, из левого нижнего в правый верхний или с чередованием направлений.
- По спирали из любого угла внутрь, из центра наружу, по часовой стрелке или против часовой стрелки.

Можно начать с любого угла прямоугольника, использовать любой маршрут для записи сообщения и любой другой маршрут для чтения. Ниже приведен пример затейливого маршрута:

1	2	3	4	5	26	27	28	29
36	6	7	8	9	10	30	31	32
37	38	11	12	13	14	15	33	34
39	40	41	16	17	18	19	20	35
42	43	44	45	21	22	23	24	25

Сообщение записывается в сетку в порядке, указанном числами, а считывается по столбцам, т. е. в порядке 1, 36, 37, 39, 42, 2, 6, 38, ....

В случае маршрутной перестановки Эмили должна лишь угадать маршрут, по которому считывалось сообщение. После того как Эмили впишет сообщение в прямоугольник, его можно будет прочесть визуально. Отметим, что Эмили не важно, записывали вы сообщение в прямоугольник  $5 \times 6$  по горизонтали или в прямоугольник  $6 \times 5$  по вертикали. Не важно и то, записывали вы его сверху вниз или снизу

вверх. Эмили этого знать не может, и ей это безразлично. Маршрутная перестановка получает оценку 1.

## 7.2 Столбцовая перестановка

Столбцовая перестановка – рабочая лошадка перестановочных шифров. Она использовалась военными, дипломатами и шпионами начиная с XVII века. Впервые этот метод описан в книге Джона Фальконера «Cryptomenysis Patefacta» (Секреты тайной переписки). После Славной революции 1688 года Джон Фальконер последовал за свергнутым королем Яковом II в изгнание во Францию, где и умер после опубликования в 1692 году второго издания книги под новым названием «Правила объяснения и расшифровки всех видов тайнописи».

В столбцовой перестановке используется ключ, в роли которого может выступать строка последовательных чисел в измененном порядке либо ключевое слово или фраза, преобразуемые в строку последовательных чисел, соответствующих порядку букв слова в алфавите. Рассмотрим ключевое слово SAMPLE. Из букв этого слова А первая в алфавите, поэтому она получает номер 1. Следующей за ней в алфавите идет буква Е, она получает номер 2. Далее по порядку следуют буквы L, М, Р и S. Поэтому SAMPLE преобразуется в 6, 1, 4, 5, 3, 2. Если одна и та же буква встречается более одного раза, то вхождения нумеруются слева направо. Например, ANACONDA преобразуется в строку 1, 6, 2, 4, 8, 7, 5, 3.

SAMPLE	ANACONDA	M I S S I S S I P P I
614532	16248753	5 1 8 9 2 10 11 3 6 7 4

Запишем слово в сетку по горизонтали слева направо. Число столбцов равно размеру ключа. Для ключа SAMPLE будет шесть столбцов, а для ключа ANACONDA – восемь. Над сеткой выпишем числовой ключ, как показано ниже:

ANACONDA	Ключевое слово
<u>16248753</u>	Числовой ключ
ENEMYTAN	Открытый текст
KSSECTOR	
FORTYTWO	
HEADINGW	
EST	
EKFHE ESRAT NROW METD AOWG NSOES TTTN YCUI	Шифртекст
EKFHE ESRAT NROWM ETDAO WGN50 ESTTT NYCUI	Группы шифртекста по 5

Прочитаем сообщение сверху вниз согласно числовому ключу. Первым читается столбец с номером 1, EKFHE, за ним столбец с но-

мером 2, ESRAT, столбец с номером 3, NROW, и так далее до столбца с номером 8, YCYI.

Полномочному получателю Риве придется немного посчитать, чтобы прочесть это сообщение. Ключ состоит из 8 букв, сообщение – из 35 букв. 35, поделенное на 8, равно 4 с остатком 3. Это значит, что массив будет содержать 4 полные строки из 8 букв и неполную строку из 3 букв. Рива должна начертить прямоугольник, соответствующий такому массиву, прежде чем заполнять столбцы, ведь ей нужно поместить в каждый правильное число символов.

Задача противника, Эмили, немного труднее. Порядок действий таков: записать буквы из каждого столбца вертикально на полоске бумаги, а затем прикладывать эти полоски друг к другу, пытаясь определить порядок столбцов. Она ищет пары полосок, в которых соответственные буквы образуют часто встречающиеся биграммы. Найдя хорошее соответствие, она пытается приложить третью полоску до или после первых двух. Когда 3 или 4 полоски будут сложены правильно, начнут проявляться короткие слова, и дальше работа пойдет быстрее.

Эмили не знает длину ключевого слова, поэтому придется ее угадывать. Она могла бы начать с длины 5 и постепенно ее увеличивать. Предположим, что она дошла до правильной длины 8. Как и Рива, она делит 35 на 8. Она знает, что есть 5 коротких столбцов по 4 буквы и 3 длинных столбца по 5 букв. Проблема в том, где начать и закончить каждую полоску, так чтобы она содержала хотя бы один полный столбец.

Первая полоска начинается с первого символа шифртекста и должна содержать 5 букв в случае, если первый прочитанный из массива столбец был длинным. Вторая полоска начинается с пятой буквы шифртекста, если первый столбец был коротким, и заканчивается на десятой букве, если оба столбца, первый и второй, были длинными. Точно так же для третьей и четвертой полосок. Затем Эмили проделяет аналогичные действия для оставшихся четырех полосок, только начинает с последней буквы шифртекста и движется от нее к центру.

Потом Эмили сопоставляет полоски, сдвигая их друг относительно друга, чтобы найти правильное взаимное расположение. Все это делается вручную, поэтому Эмили должна знать частоты наиболее употребительных биграмм и триграмм наизусть. На компьютере было бы быстрее и проще.

Чтобы воспрепятствовать такой процедуре сопоставления, отправитель Сандра считывает одни столбцы сверху вниз, а другие снизу вверх. Это значит, что Эмили понадобится второй набор полосок, читаемых в обратную сторону. И количество сопоставляемых полосок удвоится.

Столбцовая перестановка с чтением всех столбцов сверху вниз получает оценку 2, если массив прямоугольный, и 3 в противном случае. Когда столбцы читаются в чередующихся направлениях, оцен-

ка повышается до 3, если сетка прямоугольная, сетки или столбцы длинные, и до 4 в противном случае.

Столбцовая перестановка – проверенный временем метод увеличения стойкости любого подстановочного шифра. В сочетании с хорошо перемешанной подстановкой она получает оценку 5. В сочетании с хорошо перемешанным общим многоалфавитным шифром ее оценка 7. Стойкость такой комбинации максимальна, если длины двух ключей – взаимно простые числа.

Самый распространенный способ укрепления столбцовой перестановки – ввести строки переменной длины. Тогда Эмили будет труднее понять, где должны начинаться и заканчиваться полоски. Ниже продемонстрированы четыре идеи в этом направлении<sup>1</sup>. Из них идея (4) дает самый стойкий шифр, потому что полоски разрываю- ваются в непредсказуемых точках посередине, а не на концах. Возможны и более изощренные расстановки пробелов, например два и более пробелов в некоторых столбцах.

Столбцовая перестановка с такими вариациями получает оценку 4, при условии что Эмили не знает расстановки. Для варианта (4) оценка повышается до 5, если ключ длинный и число пробелов в столбцах переменное. Французы применяли подобную систему в конце Первой мировой войны. Есть мнение, что немцы смогли прочитать по крайней мере часть этих сообщений, главным образом потому, что французы многократно использовали один и тот же ключ.

(1)	(2)	(3)	(4)
TRANSPOSIT	TRANSPOSIT	TRANSPOSIT	TRANSPOSIT
961375482X	961375482X	961375482X	961375482X
INTHISCIPH	INTHISTIM	HEREISA	THISONEU
ERSHORTRO	EEACHROW	TILTEDV	SE SAPRES
WSALTERNAT	ISLONGERT	ERSIONO	ETPA TTER
EWITHLONG	HANTHEROWA	RSTAIRC	NOFBLANK
ROWS	BOVE	ASE	S

(1) TSAIW POAGH HLTSC TROSR ELNRS WOIOI HIRNN IEWER HT

(2) IALNV TWSCO TEMOE RIRGE HESAO THNHW ROTEI HBA

(3) RIEOR ELRRA DOAES EITSE TITSS AVNIH C

(4) TPF EEEN HAB OPT SAA ETO ISL NRT SENS USRK

Еще две вариации на тему лесенки (3): вариант (5) – начинать со столбца 1 при достижении правого края, и вариант (6) менять направление при достижении правого края, в результате чего полу-

<sup>1</sup> Для понимания текста полезно знать, как переводятся открытые тексты в примерах (1)–(6): (1) в этом шифре короткие строки чередуются с длинными; (2) на этот раз каждая строка длиннее расположенной над ней; (3) здесь мы имеем текст, сдвинутый лесенкой; (4) а здесь используется предустановленный паттерн пробелов; (5) этот паттерн начинается заново с первого столбца; (6) этот паттерн образует зигзаг или последовательность шевронов. – Прим. перев.

чается зизгагообразный узор. Преимущество этих вариаций в том, что в каждой строке, кроме, быть может, последней, число символов одинаково, поэтому Риве очень просто вычислить число строк. Ниже приведены примеры этих двух вариаций.

(5)	(6)	Шифртекст
EXAMPLE	EXAMPLE	(5) ITNAN MSLTT IRFRA OENST SGFTT
2715643	2715643	UPETA RHCMH ASIRO
THISP	THISP	(6) ITNAG HOETZ MQRRG REEST FZOEN
■ATTER■	■ATTER■	NPEOI RVSCH ASACU
■NSTAR■	■NFORM■	
TSAGA■	■SAZIG■	
■INFRO■	ZAGOR■	
■MTHEF■	■CHEVR■	
IRSTC■	■ONSEQ■	
■OLUMN■	■UENCE■	

Можно также создать две и более отдельных лесенок разной ширины или организовать лесенки в направлении диагонали и анти-диагонали: \ и /.

Если при дешифрировании сообщения, отправленного с применением любого из этих вариантов столбцовой перестановки, возникают трудности с определением количества строк или длины последней строки, то можно воспользоваться таким приемом. Подсчитаем число букв в сообщении и заполним массив слева направо таким количеством точек, следуя той же схеме, которой Сандра пользовалась при вписывании букв. Затем вставим буквы поверх точек.

Например, в случае варианта (2) предположим, что мы договорились всегда начинать с 7 букв в первой строке. Шифртекст (2) содержит 38 букв, поэтому помещаем 7 точек в первую строку, 8 точек во вторую и т. д., пока не разместим все 38 точек. Затем начинаем вписывать буквы в нужные столбцы, заменяя точки:

<u>961375482X</u>	<u>961375482X</u>	<u>961375482X</u>	<u>961375482X</u>
.....	..I....	..IS..M	..HIS..IM
.....	..A.....	..AC..O.	..EAC..RO.
.....	..L.....T	..LO..E.T	..SLO..GE.T
.....	..N.....W.	..NT..R.W.	..ANT..ER.W.
....	..V.	..VE	..OVE

Еще один способ использовать черные клетки – заполнить их null-символами. Null-символы следует выбирать так, чтобы они образовывали редко встречающиеся пары букв с обеих сторон, это затруднит Эмили сопоставление столбцов. Лучше использовать расхожие, а не редкие буквы, в которые легко распознать null-символы. Вот пример:

961375482X	961375482X	Шифртекст
•WHEN•ANEE	FWHENAANEE	HVGAC NMEYV ITTEI HEADO AIDIT HUAEN
L•BITE•SYO	LWBITEISYO	ANNAW WRHUA ONTAP HTRNS PNSAY FLUTO
UR•HAND•WI	URCHANDPWI	AAEOI YGS
THA•PAIN•Y	THAEPAINTY	
OUCA•NTST•	OUCAHNTSTG	
•ANDT•HATS	•AANDTNHATS	
A•MORA•Y	AOMORAUUY	

Столбцовая перестановка с null-символами получает оценку 3. При фиксированном паттерне закрашенных пробелов оценка повышается до 4.

## 7.2.1 Cysquare

Историческая справка. Во время Второй мировой войны британцы использовали вариант этой идеи, предложенный бригадиром Джоном Х. Тилтманом в 1941 году и названный им *Cysquare*. Шифр *Cysquare* представлял собой столбцовый перестановочный шифр с большим числом пробелов. Британцы применяли блокноты с разграфленными квадратами  $26 \times 26$ , в которых примерно 60 % случайно расположенных клеток было закрашено. Расположение закрашенных клеток на разных страницах различалось. Сообщение записывалось в белые клетки по строкам, а считывалось по столбцам в некотором порядке. Поскольку сетка квадратная, ориентация могла быть любой.

Ключом служил номер страницы блокнота, ориентация, а также начальная и конечная позиции в сетке. Шифровальщик должен был провести на странице линии, обозначающие область сообщения. Благодаря выбору разных областей одну страницу можно было использовать для шифрования нескольких сообщений.

Недостаток – необходимость раздавать много блокнотов. Чтобы уменьшить их количество, каждая страница использовалась в течение полного дня для шифрования до 50 сообщений. Стало быть, писать приходилось со слабым нажимом и много раз стирать написанное. Страницы становились нечитаемыми, и в конце концов шифровальщики отказались от блокнотов. В 1944 году шифр *Cysquare* отменили.

Захватив несколько таких блокнотов вместе с инструкциями, немцы стали сами пользоваться этой системой, с 1944 года и до конца войны. Они называли ее *Rasterschlüssel*, т. е. *сеточный ключ*. Но немцы не уделяли должного внимания выбору черных и белых клеток – в их блокнотах было слишком много соседних белых клеток, поэтому британцы смогли идентифицировать биграммы и триграммы методом сопоставления полосок. Эти сообщения стали ценным источником разведывательной информации. Шифр *Cysquare* получает оценку 7, а шифр *Rasterschlüssel* – оценку 4.



Отметим, что в эпоху компьютеров черные и белые клетки можно передавать в виде комбинаций битов, а сама сетка может быть любого размера и изменяться после зашифровывания каждого сообщения. Я рекомендую выбирать от 65 до 75 % черных клеток. В этом случае Cysquare получил бы оценку 8.

Есть упрощенная версия Cysquare, которая не требует напечатанных сеток и позволяет использовать числовой ключ для задания черных клеток. Ниже показаны два варианта перестановочного шифра *Blackout* – с чередованием направления и лесенкой. В обоих случаях черные клетки задаются ключом 3174255. Этот ключ может использоваться повторно или формироваться генератором псевдослучайных чисел. Для задания порядка чтения столбцов следует использовать отдельный ключ.

С чередованием	Лесенкой
3 ■■■ JACKAND	3 ■■■ JACKAND
1 JILLWENTU ■	1 JIL ■ LWENTU
7 ■■■ PTH	7 ■ PTH ■■■
4 EHILLT ■	4 E ■ HILLT
2 ■ OFETCHAP	2 OFETC ■ HAP
5 AILOF ■	5 ■ AILOF ■
5 ■ ■ WATER	5 WA ■ ■ TER

Еще один способ укрепления столбцовой перестановки – разбить текст на блоки нерегулярного размера. Например, при длине сообщения 150 его можно было бы разбить на блоки, содержащие 37, 71 и 42 буквы. Этот метод получает оценку 4. При использовании разных ключей для каждого блока оценка повышается до 5.

Сочетание столбцовой перестановки с любым подстановочным шифром резко повышает стойкость. Даже за сочетание с простой подстановкой оценка повышается до 5, поскольку сопоставление полосок сильно затруднено. И не важно, какая операция применяется первой. Сочетание общего многоалфавитного шифра со столбцовой перестановкой при 12 и более столбцах повышает оценку до 7, даже если период многоалфавитного шифра равен всего лишь 3. Причина в том, что возможность сопоставить полоски практически исключена.

### 7.2.2 Перестановка слов

*Перестановка слов* – важная историческая столбцовая перестановка не букв, а слов. Это основной метод, применявшийся Федеральной армией во время Гражданской войны в США. Идею придумал Энсон Стейджер, телеграфист, который впоследствии основал телеграфную компанию Вестерн Юнион. Линии связи федералов страдали от большого числа ошибок передачи. Часто командиры отказывались от телеграфа и просто посылали пешего или конного курьера. Стейд-

жер понял, что если передавать не отдельные буквы, а целые слова, то частота ошибок снизится и повторно передавать сообщения придется реже.

Шифровальщики федералов записывали сообщения слово за словом слева направо в прямоугольный массив, а затем считывали по различным маршрутам, например по столбцам с чередованием направления или выбирая столбцы попеременно из левой и правой половин массива. Текст щедро уснащался null-словами. Пример приведен ниже. Обратите внимание, что строка заполнена ничего не значащими null-словами. Столбцы считываются в порядке 1, 3, 5, 2, 4.

BRING	CANNON	BRIGADE	SIX	FIVE	Открытый текст
SOUTH	WEST	TO	ATTACK	ENEMY	
SHOULD	RECENT	ABOUT	HORSE	NORTH	
LEFT	FLANK	MOUNT	CHARGE	FROM	
EAST	TO	DRIVE	ENEMY	TOWARD	
OUR	GUNS				

BRING SOUTH SHOULD LEFT EAST OUR BRIGADE TO ABOUT MOUNT DRIVE  
 FIVE ENEMY NORTH FROM TOWARD CANNON WEST RECENT FLANK TO GUNS  
 SIX ATTACK HORSE CHARGE ENEMY

## 7.3 Двойная столбцовая перестановка

Как понятно из названия, двойная столбцовая перестановка подразумевает выполнение двух столбцовых перестановок подряд, желательно с разными ключами. В результате исключается сама возможность сопоставления полосок. Удивительно, но в 1934 году Соломон Кулльбак нашел общее решение, которое было напечатано Службой разведки сигналов. Эта брошюра, занимающая 31 страницу, была рассекречена в 1980 году и опубликована издательством Aegean Park Press. Это издательство в течение многих лет было бесценным источником книг по криптографии. Оно прекратило существование в 2001 году со смертью своего основателя Уэйна Дж. Баркера, после чего книги стали недоступны. Рад сообщить, что теперь они имеются на сайте [www.openlibrary.org](http://www.openlibrary.org) (я заходил туда в июле 2019 года).

Не буду повторять здесь анализ Кулльбака, скажу только, что он основан на определении мест шифртекста, в которых встречается каждая буква открытого текста. Вместо этого я обсужу три способа противостоять решению Кулльбака. (Кстати, Кулльбак учился в той же школе, что мой отец, но шестью годами раньше.)

Простой способ – изменить форму сетки, вычеркнув несколько клеток. Эти клетки могут образовывать прямоугольник или какую-то другую фигуру, располагаться в одном из углов или даже в середине сетки. Вот несколько примеров:



Черные участки на двух шагах перестановки могут иметь разные размеры, формы или местоположения. Эти параметры можно включить в состав ключа, так что для каждого сообщения будут использоваться свои черные участки. Двойная столбцовая перестановка получает оценку 4. А с черными клетками – оценку 5.

Противоположный метод, называемый *NullBlock*, также эффективен. Можно вставить блок null-символов в промежуточный шифртекст, в окончателный шифртекст или туда и сюда. Вставлять такой блок в открытый текст бессмысленно. Размер и положение блока можно задать с помощью числового ключа. Ключи для каждого сообщения должны быть различны.

Сочетание любого перестановочного шифра с любым подстановочным повышает стойкость обоих. Двойная столбцовая перестановка в сочетании с простой подстановкой получает оценку 6. А в сочетании с общим многоалфавитным шифром – оценку 8.

## 7.4 Столбцовая перестановка с циклическим сдвигом

Еще одна вариация на ту же тему – *столбцовая перестановка с циклическим сдвигом*. Есть две разновидности: со сдвигом по горизонтали и по вертикали. Для циклической перестановки с циклическим сдвигом по горизонтали нужно два ключа: один для задания циклического сдвига строк, другой для определения порядка столбцов. Сначала записываем сообщение в прямоугольный блок слева направо по строкам. Затем записываем ключи циклического сдвига по вертикали слева от строк. Если строк больше, чем длина ключа, то повторяем ключи столько раз, сколько необходимо. Если в качестве ключа циклического сдвига выбирается слово или фраза, то преобразуем его в число обычным способом, исходя из порядка букв в алфавите.

Имея числовой ключ, циклически сдвигаем каждую строку влево на указанное в ключе число позиций. Затем считываем буквы по столбцам в порядке, заданном столбцовым ключом. Ниже приведен пример с ключом циклического сдвига CYCLES и столбцовым ключом PAULREVERE.

Ключ циклического сдвига	PAULREVERE <u>619572x384</u>	Столбцовый ключ
C 1 ONEIFBYLAN	NEIFBYLANO	Шифртекст
Y 6 DANDTWOIFB	OIFBDANDTW	EIAPR LYAIT AADNH EOWSO HFBNS SNOEP
C 2 YSEAANDION	EAANDIONYS	OEBDD IHNTY ESIFA OEBLN OTL
L 4 THEOPPOSIT	PPOSITTHEO	
E 3 ESHORESHAL	ORESHALES	
S 5 LBE	ELB	

Метод сопоставления бумажных полосок, который мы использовали для вскрытия столбцовой перестановки, работает и для столбцовой перестановки с циклическим сдвигом. Он лишь немногим труднее, потому что последняя буква в каждой строке соседствует с первой буквой, потенциально образуя низкочастотную бигramму. Но это задержит Эмили ненадолго. Столбцовая перестановка с циклическим сдвигом по горизонтали получает оценку 3.

Циклический сдвиг по вертикали аналогичен. Вместо циклического сдвига строк влево мы сдвигаем столбцы вверх. Ниже приведен пример с ключевым словом CYCLE для циклического сдвига столбцов и с ключевым словом PAULREVERE, задающим порядок чтения столбцов.

CYCLECYCLE	PAULREVERE	Ключевая фраза
<u>1524315243</u>	<u>619572x384</u>	Числовой ключ
ONEIFBYLAN	DBEOPWYIAT	Открытый текст
DANDTWOIFB	YNEIRNOSAL	
YSEAANDION	TAHDFPDHFN	
THEOPPOSIT	ESEATEOLOB	
ESHOESHAL	LHEOABSIIN	
LBE	OSN	
BNASH SWNPE BISHL ITLNB NOIDA ODYTE		Шифртекст
LORPF TAAAF OIEEH EENYO DOS		

Этот шифр по-прежнему можно вскрыть сопоставлением бумажных полосок, как и в случае регулярной столбцовой перестановки, но Эмили понадобится две полоски для каждого столбца: для верхней и для нижней секций. Когда столбец циклически сдвигается, некоторые символы, находившиеся сверху, перемещаются вниз и становятся новой нижней секцией. Остальные буквы перемещаются вверх и становятся новой верхней секцией. В примере выше левый столбец **ODYTEL** сдвигается вверх на одну позицию, поэтому **DYTEL** становится новой верхней секцией, а **O** – новой нижней секцией. Эти секции должны быть записаны на отдельных полосках, потому что Эмили не знает, откуда взялись буквы: из длинного столбца или из короткого. Поэтому процесс сопоставления значительно осложняется. Столбцовая перестановка с циклическим сдвигом по вертикали получает оценку 4.

Можно подвергнуть блок одновременно циклическому сдвигу по горизонтали и по вертикали. По стойкости это сравнимо с двойной столбцовой перестановкой. Напомним, однако, что чем сложнее шифр, тем больше времени и труда занимает зашифровывание и расшифровывание. Двойная столбцовая перестановка с циклическим сдвигом получает оценку 5.

## 7.5 Перестановка со случайными числами

Рассмотрим теперь перестановку совершенно другого типа. В ней не будет никаких массивов и сеток. А просто буквы сообщения нумеруются случайным образом.

Можно использовать любой генератор случайных чисел. Несколько таких представлено в главе 13. До сих пор я описывал только цепной генератор цифр в разделе 4.5.1, поэтому им и воспользуемся для иллюстрации. Сгенерируем по одной случайной цифре для каждой буквы сообщения:

```
4319332748659278301459896508643752196
INCREASEBIDTOTWELVEPOINTSEVENMILLION
```

Сначала возьмем все буквы с номером 1, слева направо. Это C, V и I.

```
4319332748659278301459896508643752196
IN-REASEBIDTOTWEL-EPOINTSEVENMILL-ON CVI
```

Затем возьмем все буквы с номером 2. Это A, O и L.

```
4319332748659278301459896508643752196
IN-RE-SEBIDT-TWEL-EPOINTSEVENMILL--ON CVIAOL
```

Далее возьмем все буквы с номером 3. Это N, E, E и M.

```
4319332748659278301459896508643752196
I--R--SEBIDT-TW-L-EPOINTSEVEN-IL--ON CVIAOLNEEM
```

Продолжаем, пока еще остаются буквы.

Чтобы дешифровать сообщение, Рива сначала генерирует случайные цифры. Среди них имеются три единицы, поэтому она записывает первые три буквы шифртекста, CVI, под этими тремя единицами:

```
4319332748659278301459896508643752196
--C-----V-----I--
```

Далее Рива записывает следующие три буквы, **AOL**, под двойками:

431932748659278301459896508643752196  
--C--A-----0----V-----LI--

И так далее.

Перестановка со случайными числами получает оценку 4. Ее можно взломать, перебрав все возможные начальные значения генератора случайных чисел.

Шифр можно укрепить, взяв более длинное начальное значение или выбирая буквы открытого текста не в естественном порядке 1, 2, 3, ..., а в каком-то другом. Это эквивалентно применению простой подстановки к выходу генератора случайных чисел. Например, если бы мы захотели начать с букв с номером 4, то следовало бы заменить все четверки на единицы. А если бы далее мы захотели взять все буквы с номером 7, то нужно было бы заменить все семерки на двойки и т. д. Все остальное делается, как и раньше. Количество возможных ключей при этом увеличивается в  $10! = 3\,628\,800$  раз.

При таком улучшении перестановка с цепным генератором цифр получает оценку 5. Компьютерная версия метода, в которой генератор случайных цифр порождает байты, получает оценку 7 из-за очень большого числа возможных перестановок 256 различных байт.

## 7.6 Селекторная перестановка

Раз уж мы заговорили о случайных числах, рассмотрим еще один перестановочный шифр на основе случайных чисел – *селекторную перестановку*. Идея в том, чтобы разбить сообщение на приблизительно равные части, а затем объединить их в случайной последовательности.

Пусть открытый текст содержит 100 символов, и мы хотим разбить его на три части. Предположим, что имеется генератор случайных чисел, который порождает цифры 0, 1 и 2 с равной вероятностью, и что мы выбираем начальное значение, играющее роль ключа перестановочного шифра. Необходимо знать размер каждой части сообщения. Тут все просто. Нужно лишь сгенерировать первые 100 случайных цифр и подсчитать, сколько среди них экземпляров каждой цифры. Допустим, сгенерировано 36 нулей, 25 единиц и 39 двоек. Разобьем сообщение на три части: P0 длиной 36, P1 длиной 25 и P2 длиной 39.

С шифрованием проблем не возникает. Всякий раз, как генератор порождает 0, берем следующую букву из P0. Когда генератор порождает 1, берем букву из P1, а когда он порождает 2 – букву из P2. Дешифрование еще проще, поскольку Риве не нужно знать размеры частей. Получив от генератора 0, она помещает следующую букву в P0, получив 1 – в P1, а получив 2 – в P2. Затем она конкатенирует

все три части или просто читает сообщение, не обращая внимания на разрывы строк.

Если частей всего две, то Эмили реконструирует сообщение тривиально. Оценка 1. Если частей три, то задача немного усложняется – оценка 2. При 20 и более частях шифр получает оценку 5.

## 7.7 Перестановка с ключом

Иногда предпочтительнее переставлять блоки сообщения. Лучший шифр такого рода – перестановка с ключом. Запишем числовой ключ над символами сообщения, а затем переместим каждый символ в позицию, совпадающую с соответственной цифрой ключа. В примере ниже размер блока равен 8, а ключ равен 41278563. Первой букве, **R**, соответствует цифра ключа 4, поэтому перемещаем **R** в четвертую позицию блока. Второй букве, **U**, соответствует цифра 1, поэтому перемещаем **U** в первую позицию блока. И так далее.

<b>41278563</b>	<b>41278563</b>	<b>41278563</b>	<b>41278563</b>	<b>41278563</b>	<b>4127</b>
<b>RUSSIA</b>	<b>NT RADE</b>	<b>DELE GATI</b>	<b>ONEX PE</b>	<b>CTEDTO</b>	<b>ARRIV</b>
<b>EL</b>	<b>EL</b>	<b>EL</b>	<b>EL</b>	<b>EL</b>	<b>EL</b>
<b>USTRANSI</b>	<b>ADTRELED</b>	<b>ATXRNEIO</b>	<b>ECOPDTTE</b>	<b>RROAELIV</b>	<b>DONN</b>

Перестановку с ключом можно применять к открытому тексту, к шифртексту или к тому и другому. Сама по себе перестановка с ключом – слабый шифр. Ее оценка – от 1 до 3 в зависимости от размера блока.

\* Приглядимся к перестановкам пристальнее. В примере ниже я использовал шестнадцатеричные цифры A, B, C для обозначения чисел 10, 11, 12. В шифре эти числа будут представлять биты, буквы или иные переставляемые единицы.

<b>123456789ABC</b>	Исходный порядок
<b>4A1729C5B683</b>	После перестановки

Верхняя строка стандартная. Она представляет исходный порядок, до перестановки. Вторая строка – результат перестановки. Далее в этом разделе вторая строка будет использоваться для описания перестановок.

В этой перестановке число из позиции 1 перемещается в позицию 4, число из позиции 4 – в позицию 7, из позиции 7 – в позицию 12, из позиции 12 – в позицию 3, а из позиции 3 – в позицию 1, завершая цикл **1→4→7→C→3→1**. Этот цикл можно записать в виде (1, 4, 7, 12, 3).

Первое число, не принадлежащее этому циклу, – 2. Начав с позиции 2, мы найдем цикл **2→A→6→9→B→8→5→2**, который можно записать в виде (2, 10, 6, 9, 11, 8, 5). Тогда всю перестановку можно записать в виде (1, 4, 7, 12, 3) (2, 10, 6, 9, 11, 8, 5).

Эти два цикла имеют период 5 и 7 соответственно, так что период всей перестановки равен 35. Это значит, что если повторно применять ее к блоку из 12 букв, то получится 35 различных перестановок букв, а 36-я совпадет с исходным блоком.

Допустим, что мы хотели создать стойкий блочный перестановочный шифр, в котором для каждого блока определена своя перестановка. Применение показанной выше перестановки разное число раз к каждому блоку не годится, потому что через каждые 35 циклов мы возвращаемся в исходное место. Эту проблему можно решить, если использовать две разные перестановки и чередовать их.

Пусть есть две перестановки, А и В. Если А и В выбраны правильно, то можно сгенерировать огромное число перестановок: А, В, АА, АВ, ВА, ВВ, ААА, ААВ, АВА, ..., – и все они будут различны.

Вот здесь важно понимать циклическую структуру перестановок. Предположим, что выбраны перестановки (1, 4, 7, 12, 3) (2, 10, 6, 9, 11, 8, 5) и (1, 4, 3, 12, 7) (2, 10, 9, 6, 5, 11, 8). Они разбивают блок из 12 единиц на две одинаковые части, а именно: [1, 3, 4, 7, 12] и [2, 5, 6, 8, 9, 10, 11]. Если чередовать две перестановки, то часть [1, 3, 4, 7, 12] будет переставляться независимо от части [2, 5, 6, 8, 9, 10, 11]. То есть между двумя множествами чисел нет никакого взаимодействия. Чтобы получить длинный период, каждый цикл второй перестановки должен как можно сильнее перекрываться с каждым циклом первой перестановки. Вот подходящий набор перестановок:

(1, 4, 7, 12, 3) (2, 10, 6, 9, 11, 8, 5)  
(1, 10, 8) (4, 6, 5, 12) (2, 11, 9, 7, 3)

Этот перестановочный шифр получает оценку 3. Чтобы его вскрыть, нужно перебрать все 12! возможных перестановок первого блока. Это всего  $4.79 \times 10^8$ . Для каждой перестановки, порождающей разумный текст первого блока, Эмили может перебрать все 12! перестановок второго блока. На практике попыток будет гораздо меньше, чем  $(12!)^2 = 2.29 \times 10^{17}$ , потому что знания первых 3 или 4 символов блока может оказаться достаточно для исключения маловероятных комбинаций. На самом деле вскрыть этот шифр реально даже вручную. Трудность медленно возрастает с увеличением размера блоков. \*\*

Существует несколько способов повысить безопасность перестановки с ключом. Один из них – перекрытие блоков. Например, если размер блока равен 16, то можно начинать блоки не в позициях сообщения 1, 17, 33, ..., а в позициях 1, 9, 17, 25, 33, .... Тогда каждый блок перекрывает 8 единиц предыдущего и 8 единиц последующего блоков. Последние восемь единиц сообщения можно объединить с восьмью первыми единицами, образовав заворачивающийся блок. Этот шифр получает оценку 4.

Величина перекрытия может быть и переменной. Если текущий блок начинается в позиции Р и имеет длину L, то следующий блок



может начинаться в любой позиции от  $P+1$  до  $P+L$ . Этот шифр получает оценку 5. Если используются две разные перестановки и позиции перекрытия выбираются случайно, то оценка повышается до 7.

\* Второй способ повысить стойкость блочного перестановочного шифра – выполнить композицию перестановок. Если  $T$  и  $U$  – перестановки, то их композицией, обозначаемой  $TU$ , называется перестановка, получающаяся, если сначала выполнить  $U$ , а затем  $T$ . Результат будет таким же, как если применить  $T$  для перестановки  $U$ , а затем получившуюся перестановку применить к тексту. Рассмотрим пример. Пусть  $T$  – перестановка **419628573**, а  $U$  – перестановка **385917462**. Поскольку  $U$  – блок из 10 символов, то можно использовать  $T$  для перестановки  $U$  точно так же, как мы использовали бы  $T$  для перестановки 10-буквенного слова. Записываем  $T$  в первой строке, чтобы использовать как ключ перестановки, а  $U$  во второй строке – как переставляемый текст. Первая цифра результата – это цифра, расположенная под цифрой 1 ключа, т. е. **8** (см. закрашенные серым цифры). Вторая цифра результата – это цифра, расположенная под цифрой 2 ключа, т. е. **1**, и т. д. При использовании  $T$  для перестановки  $U$  получается **812349675**.

<b>419628573</b>	$T$
<b>385917462</b>	$U$
<b>812349675</b>	$TU$

С помощью композиции мы можем генерировать последовательность перестановок,  $U$ ,  $TU$ ,  $TTU$ ,  $TTTU$ , .... Период этой последовательности такой же, как у  $T$ . Для блока размера 12 самый длинный период получается при длинах циклов 3, 4 и 5, т. е. он равен  $3 \times 4 \times 5 = 60$ . Если количество блоков в сообщении больше 60, то желателен более длинный период. Это можно сделать, применяя к перестановке либо  $U$ , либо  $T$  в каком-то регулярном или случайном порядке, например  $U$ ,  $TU$ ,  $TTU$ ,  $UTTU$ ,  $UUTTU$ , .... Так можно сгенерировать очень большой набор различных перестановок, при условии что сами  $T$  и  $U$  имеют длинные периоды, а циклы  $T$  перекрываются с циклами  $U$ , как было описано выше.

Чтобы проверить, достаточно ли перекрытие циклов, можно воспользоваться *тестом прироста*. Начнем с любого цикла  $T$  или  $U$ . Он образует множество, содержащее всего один этот цикл. Добавим к множеству любой другой цикл  $T$  или  $U$ , имеющий общие элементы с первым. Затем добавим в новое множество еще какой-то цикл  $T$  или  $U$ , имеющий общие элементы с ранее выбранными циклами. Продолжаем в том же духе до тех пор, пока добавление циклов с общими элементами возможно. Если теперь множество циклов содержит все циклы  $T$  и  $U$ , то перекрытие хорошее. Если вы решите использовать больше двух перестановок, скажем  $T$ ,  $U$  и  $V$ , то перекрываться должна каждая из пар  $T$  и  $U$ ,  $T$  и  $V$ ,  $U$  и  $V$ .

Проиллюстрирую на примере перестановок, встречавшихся ранее в этом разделе,  $T = (1, 4, 7, 12, 3) (2, 10, 6, 9, 11, 8, 5)$  и  $U = (1, 10, 8) (4, 6, 5, 12) (2, 11, 9, 7, 3)$ . Начнем с цикла  $(1, 4, 7, 12, 3)$ .

$(1, 4, 7, 12, 3)$

Он имеет общий элемент 1 с циклом  $U (1, 10, 8)$ , поэтому добавим этот цикл в множество.

$(1, 4, 7, 12, 3) (1, 10, 8)$

Оно имеет общий элемент 4 с циклом  $U (4, 6, 5, 12)$ , поэтому добавляем в него и этот цикл.

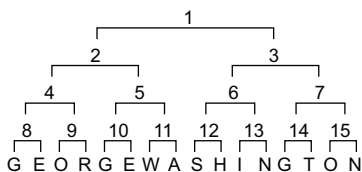
$(1, 4, 7, 12, 3) (1, 10, 8) (4, 6, 5, 12)$

И так далее. Поскольку множество включает все циклы  $T$  и  $U$ , эта пара перестановок имеет хорошее перекрытие и генерирует очень большое семейство перестановок в результате композиций. \*\*

## 7.8 Деление перестановки пополам

*Деление перестановки пополам* – предназначенный для применения на компьютере метод, в котором двоичный ключ используется для обмена местами единиц – битов, байтов или даже шестнадцатеричных цифр. IBM опубликовала его во внутреннем бюллетене Invention Disclosure Bulletin, и он рассматривался как кандидат на включение в стандарт шифрования данных DES. Метод применяется к блоку, размер которого равен степени 2, обычно 32 или 64 единицы. Для блока из  $n$  единиц длина ключа равна  $n - 1$  бит.

Рассмотрим метод на примере блока из 16 символов. Открытым текстом будет GEORGE WASHINGTON. В перестановке используется 15-битовый ключ. Первый бит ключа определяет, нужно ли менять местами половины блока по 8 символов: 0 – не менять, 1 – менять. Следующие 2 бита ключа определяют, нужно ли менять местами 4-символьные половины этих половин. Если бит 2 ключа равен 1, то первая четверть меняется местами со второй. Если бит 3 ключа равен 1, то третья четверть меняется местами с четвертой. Следующие 4 бита ключа определяют, нужно ли менять местами половины этих четвертей. Например, если бит 4 равен 1, то первая восьмушка блока меняется местами со второй. Последние 8 бит управляют обменом шестнадцатых частей блока. Например, если последний бит ключа равен 1, то последние 2 единицы, 15-я и 16-я, т. е. буквы O и N, меняются местами.



Для дешифрирования перестановки шаги производятся в обратном порядке. То есть сначала меняются местами отдельные единицы, затем пары, четверки и т. д.

## 7.9 Множественные анаграммы

Общий метод вскрытия перестановочных шифров, который работает для многих типов перестановок и даже когда тип неизвестен, называется *множественными анаграммами*. Для его использования необходимо перехватить несколько сообщений одинаковой длины. Если для перестановки этих сообщений применялся один и тот же ключ, то первые буквы всех сообщений окажутся в одной и той же позиции всех шифртекстов, равно как вторые и все последующие буквы.

Этим фактом можно воспользоваться. На первой бумажной полоске выпишите все первые буквы шифртекстов, на второй полоске – все вторые буквы и т. д. Количество полосок должно совпадать с длиной шифртекстов. Чем больше имеется сообщений, тем длиннее будут полоски и тем выше шансы на успех. В общем случае считается, что необходимо как минимум 3 сообщения.

Рассмотрим пример. Пусть имеется три зашифрованных сообщения:

	1	2	3	4	5	6	7	8	9	10	11	12
(1) TTWACNAATKAD	T	T	W	A	C	N	A	A	T	K	A	D
(2) NEMSMORMEODA	N	E	M	S	M	O	R	M	E	O	D	A
(3) ETCMMEANEETO	E	T	C	M	M	E	A	N	E	E	T	O

Длина сообщения равна 12. В правой части показаны все 12 полосок.

Сообщение (1) содержит букву К, которой часто предшествуют буквы С и N. Сообщение (1) содержит ту и другую. Сообщение (2) содержит букву D. Ей с большой вероятностью может предшествовать буква N. В сообщении (2) есть одна буква N. Посмотрим, существуют ли какие-то правдоподобные варианты. Имеем:

5 10	6 10	1 11
С К	Н К	Т А
М О	О О	Н D
М Е	Е Е	Е Т

В сообщении (2) самая вероятная буква, предшествующая **ND**, – **A** или **E**. Это дает три возможности:

2 1 11	9 1 11	12 1 11
T T A	T T A	D T A
E N D	E N D	A N D
T E T	E E T	O E T

Сопоставляя каждую из них с 9 остальными полосками, обнаруживаем, что лучшее соответствие дает полоска 4.

4 9 1 11
A T T A
S E N D
M E E T

Это хорошо согласуется со столбцами 5 и 10, которые мы уже объединили ранее.

4 9 1 11 5 10
A T T A C K
S E N D M O
M E E T M E

Теперь легко завершить все три сообщения: (1) ATTACK AT DAWN, (2) SEND MORE AMMO и (3) MEET ME AT ONCE.

# Цилиндрический шифр Джефферсона

---

## **Краткое содержание главы:**

- цилиндрический шифр Томаса Джефферсона;
- вскрытие цилиндрического шифра, когда есть известное слово;
- вскрытие цилиндрического шифра, когда известных слов нет.

Томас Джефферсон изобрел *цилиндрический шифр* между 1790 и 1793 годом, когда исполнял должность государственного секретаря при Джордже Вашингтоне. Устройство состоит из железного стержня, или оси диаметром от  $1/8$  до  $1/4$  дюйма и длиной от 6 до 8 дюймов, на которую нанизано 36 деревянных дисков диаметром 2 дюйма и толщиной  $1/6$  дюйма. Эти диски образуют деревянный цилиндр. Плоские поверхности дисков соприкасаются, а внешние круглые края видны. На один конец оси насажен глухой стопор, напоминающий шляпку гвоздя, а на другом нарезана резьба, на которую навинчивается гайка, не дающая дискам рассыпаться.

На плоских поверхностях дисков проставлены номера от 1 до 36. Внешний край поделен на 26 равных секторов, в которые вписаны или выгравированы 26 букв алфавита в случайном порядке, своем для каждого диска. Порядок дисков на оси – это ключ шифра, в наши дни называемого *мультиплексным шифром*.

Ниже приведено изображение шифровального цилиндра Джефферсона с 26 дисками. Оригинал хранится в Национальном музее криптологии в Форт-Миде, штат Мэриленд.



Чтобы зашифровать сообщение с помощью этого устройства, диски нужно нанизать на ось в порядке, определяемом ключом. Гайка закручивается не до конца, чтобы диски можно было поворачивать. На первом диске находим первую букву сообщения, а второй диск поворачиваем так, чтобы вторая буква сообщения находилась рядом с первой. Затем третий диск поворачивается так, чтобы третья буква оказалась рядом со второй, и так далее, пока первые 36 букв сообщения не выстроятся в ряд. После этого гайка закрепляется и фиксирует диски.

Вращая цилиндр, мы будем видеть еще 25 строк букв, содержащих бессмысленную абракадабру. Сандра может выбрать любую из них в качестве шифртекста. Рива выполнит описанную выше процедуру, собрав в одном ряду букв шифртекст. После этого будет ясно, какая из остальных 25 строк содержит сообщение.

По-видимому, Джефферсон так никогда и не воспользовался этим шифром. Идея находилась под спудом, пока в начале 1890-х годов ее не открыл заново Этьен Базери. Французы приняли ее на вооружение в 1901 году. В варианте Базери было два усовершенствования. Он снабдил устройство подставкой, чтобы его можно было поставить на стол и использовать обе руки, а также добавил направляющую, чтобы пользователю было проще выравнивать буквы и выбирать строку для считывания шифртекста. Вариант этого шифра с 25 алюминиевыми дисками изобрел полковник Паркер Хитт в 1914 году, в 1922 году он был принят армией США под кодовым названием М-94, а в 1926 – военно-морским флотом США под названием CSP-488. Длина устройства Хитта составляла всего 4,25 дюйма (10,8 см), так что его можно было носить в кармане, а на плоских поверхностях имелись выемки и выступы, чтобы диски не проскальзывали после фиксации.

Ниже приведена фотография устройства CSP-488 из Национального музея криптологии.



Плоскую версию этого шифра Хитт изобрел в 1916 году, а армия приняла ее на вооружение в 1935 году под кодовым названием М-138. Устройство представляло собой плоскую алюминиевую дощечку, в которой было прорезано 25 каналов для размещения бумажных полосок, которые могли скользить в обе стороны, имитируя вращение дисков. На каждой полоске было напечатано две копии перемешанного алфавита. Это устройство было безопаснее, потому что бумажные полоски легко заменить и даже написать от руки в полевых условиях. Вскоре оно было заменено устройством М138А, или CSP-845 по флотской классификации, в котором было 30 прорезей для полосок. В комплект поставки устройства входило 100 полосок, пронумерованных двузначными числами, так что для любого сообщения использовалось 30 из 100 полосок. Это дает  $100!/70! = 7.79 \times 10^{57}$  возможных ключей.

Устройство М-138А имело петлю в центре, так что его можно было складывать. Каждая половина оснащалась отдельной направляющей, которая служила для выравнивания полосок и считывания 15 букв шифртекста. Эти усовершенствования значительно повысили стойкость шифра.

Армия отказалась от шифров на полосках в 1942 или 1943 году, но на флоте они по-прежнему использовались как резервный вариант на случай, если отключение электричества сделает невозможным использование электронных или электромеханических устройств.

Мультиплексный шифр практически невозможно вскрыть, если у Эмили нет копии устройства и она не знает алфавитов. Если же устройство у нее есть, то вскрыть шифр относительно просто, коль скоро известны какие-то вероятные слова. При наличии устройства и знании некоторых слов цилиндрический шифр Джефферсона получает оценку 4 или 5. Если никакие слова неизвестны, то оценка повышается до 6 или 7. Чем больше шифртекстов есть у Эмили, тем ниже оценка. Наоборот, при наличии большого числа дополнительных дисков оценка растет. Например, если устройство вмещает 30 дисков, выбираемых из запаса в 100 дисков, то оценка может возрасти до 8. Для очень коротких сообщений – длиной меньше удвоенного числа дисков – вскрыть шифр вряд ли возможно, если только Эмили не располагает несколькими перехваченными сообщениями, зашифрованными одним и тем же ключом. Это может случиться, если отправитель меняет порядок дисков только один раз в день.



## 8.1 Вскрытие при наличии известных слов

Прочитать сообщение, зашифрованное цилиндрическим шифром Джефферсона, можно, если имеется достаточно текста и известна хотя бы часть сообщения. Часто хватает знания всего одного слова. Предположим, нам известно, что Сандра пользуется устройством М-94 с 25 дисками, и удалось перехватить сообщение

CLPOXFDQBOMTUCESZITNCVGWX  
ESIWVILLSCQYRNPFJCNSRWXGK  
GAFOEMZTGHJWQZTYMSAXTBILF  
UICSBHNNPMBZQRCDH

Предположим также, что мы знаем, что открытый текст начинается со слова URGENT. Оно было преобразовано в шифртекст **CLPOXF**. Поскольку URGENT находится в одном ряду, в **CLPOXF** в другом, расстояния между парами соответственных букв должны быть одинаковы. Условимся считать номер ряда, содержащего URGENT, равным 1 и предположим, что ряд, содержащий **CLPOXF**, имеет номер 8. Первая буква открытого текста, U, и первая буква шифртекста, C, берутся из ряда 1 и ряда 8 первого диска. Поэтому на первом диске расстояние от U до C должно быть равно 7. На втором диске расстояние от R до L должно быть равно 7. На третьем диске должно быть равно 7 расстояние от G до P. И так далее для пар E и O, N и X, T и F.

Самый простой способ поиска – пробовать каждое возможное расстояние от 1 до 25 по очереди. Начинаем с расстояния 1. Найдём все диски, для которых расстояние от U до C равно 1. Иными словами, C должно быть следующей буквой после U. Если таких дисков нет, значит, расстояние точно не равно 1. Затем найдём все диски, для которых расстояние от R до L равно 1. И снова, если таковых не оказалось, значит, расстояние не может быть равно 1.

Предположим, мы нашли 12 наборов дисков, в которых все пары букв находятся на расстоянии 1 друг от друга. Нужно проверить эти 12 наборов и посмотреть, есть ли среди них правильные. Пусть в первый набор входят диски 18-4-21-9-13-11. Начинаем проверять со второго блока шифртекста, содержащего буквы от 26 до 50. Этот блок начинается с **ESIWVI**. Установим на диске 18 букву **E**, на диске 4 – букву **S**, на диске 21 – букву **I** и т. д. Теперь рассмотрим остальные 25 рядов. Если все они содержат чепуху вроде **HNSAEI** или **TFFGUM**, то можно точно сказать, что последовательность дисков 18-4-21-9-13-11 неправильная. С другой стороны, если мы увидим какой-то разумный текст, например строку NCONDI, которая могла бы быть частью слова UNCONDITIONAL, то, возможно, последовательность 18-4-21-9-13-11 правильная. Проверяем далее, взяв третий блок шифртекста, начинающийся с GAFOEM. Если и в третьем, и в четвертом блоке удастся выделить разумные фрагменты текста, то последовательность 18-4-21-9-13-11, вероятно, правильная... но надо продолжать про-



верку, потому что, возможно, удастся найти лучшую последовательность.

Если мы не видим никаких сколько-нибудь обнадеживающих фрагментов текста, то пробуем оставшиеся 11 последовательностей дисков. Если так ничего и не нашлось, пробуем расстояние 2, затем 3 и т. д. до 25. Вероятно, придется проверить несколько сотен комбинаций порядка дисков и расстояния. Это утомительно, но все же возможно без помощи компьютера. Если ничего не получилось, возвращаемся и ищем последовательности дисков, для которых две проверки из трех дают правдоподобный текст.

Выбрав самую вероятную последовательность для первых 6 дисков и соответствующие расстояния, пытаемся продолжить ее на седьмой диск. Для каждого выбора диска мы уже знаем расстояние между открытым текстом и шифртекстом, поэтому процесс продолжения пойдет сравнительно быстро.

## 8.2 Вскрытие при наличии только шифртекста

Мультиплексный шифр можно вскрыть и тогда, когда известных слов нет. Это называется вскрытием при наличии только шифртекста. Я первым нашел такое решение («Computer Methods for Decrypting Multiplex Ciphers». *Cryptologia* 2 (Apr. 1978), pp. 152–160). В оригинальной статье 1978 года я использовал частоты биграмм и переходил к триграммам. Сегодня компьютеры гораздо быстрее и имеют гораздо больше памяти, поэтому шаг биграмм можно пропустить. Метод предполагает, что имеется таблица с вероятностями всех возможных триграмм в английском языке. Вы можете построить такую таблицу самостоятельно или скачать ее из интернета. Опишу основную идею метода.

Предположим, как и раньше, что Эмили использует устройство М-94 с 25 шифровальными алфавитами и что мы перехватили сообщение, содержащее как минимум 3 блока, или 75 букв. Нам известно только, что сообщение написано по-английски. Пусть перехваченное сообщение имеет вид:

CLPOXFDQBOMTUCESZITNCVGWX  
ESIWILLSCQYRNPFJCNRSRWXGK  
GAFOEMZTGNJWQZTYMSAXTBILF

Сначала проверим все возможные варианты для первых трех дисков. Их  $25 \times 24 \times 23 = 13\,800$ . Для каждого варианта выставим на дисках первые три буквы каждого блока шифртекста, а именно CLP, ESI и GAF. Для каждой из этих триграмм посмотрим на остальные 25 рядов. В них встречаются возможные триграммы открытого текста, соот-

ветствующие триграммам шифртекста. Поскольку для каждого из трех рядов имеется 25 вариантов, общее число возможностей равно  $13800 \times 25^3 = 215\,625\,000$ . Их легко перебрать на настольном компьютере и даже на ноутбуке.

Вероятность каждой комбинации трех дисков и трех рядов равна произведению вероятностей трех триграмм открытого текста. Эквивалентно, логарифм этой вероятности равен сумме логарифмов вероятностей трех триграмм. Идея заключается в том, чтобы оставить только самые вероятные комбинации и отбросить остальные. Например, можно было бы оставить только 1 % самых вероятных комбинаций или фиксированное число лучших комбинаций, скажем 1 000 000. Допустим, что мы решили сохранять 2 000 000 лучших комбинаций.

Сделать это можно, например, так: генерируем все 215 625 000 комбинаций, сортируем их по вероятности и отбрасываем последние 99 %. Это потребовало бы очень много памяти. Есть способы лучше. Сначала выделим память для таблицы, которая на 10–25 % больше нужного нам числа комбинаций, скажем с 2 500 000 элементами. Начнем генерировать комбинации и помещать их в таблицу. Когда таблица заполнится, ее нужно будет сократить примерно на 20 %.

Это можно сделать, отсортировав таблицу и удалив нижние 20 %. То есть мы сортируем по вероятности, а затем уменьшаем число элементов таблиц до 2 000 000. Отсортировать 2 500 000 элементов быстрее, чем все 215 625 000, но есть способы еще быстрее – и намного. Выберем из таблицы 10 случайных элементов. (Если вы не знаете, как выбирать случайно, то разбейте таблицу на 11 равных частей и возьмите начальные элементы первых десяти частей.) Отсортируйте эти 10 элементов по вероятности в порядке возрастания. Назовем отсортированные элементы a, b, c, d, e, f, g, h, i, j. Обозначим P вероятность элемента b. Удалим из таблицы все элементы с вероятностью меньше P.

Продолжаем генерировать комбинации, но больше не добавляем в таблицу элементы с вероятностью, меньшей или равной P. После каждого заполнения таблицы повторяем процесс выборки, сортировки и пересчета пороговой вероятности P.

В конечном итоге у нас останется примерно 2 000 000 комбинаций трех дисков и трех рядов. Следующий шаг – продолжить на 4 диска. Пробуем все 22 возможных варианта выбора 4-го диска. Это дает примерно 44 000 000 комбинаций. Теперь рассмотрим триграммы, образованные дисками 2, 3 и 4. Для каждой комбинации умножим вероятность триграммы на дисках 1, 2, 3 на вероятность триграммы на дисках 2, 3, 4 и получим приближенную вероятность тетраграммы на всех 4 дисках. Перемножим вероятности тетраграмм открытого текста, соответствующих всем трем тетраграммам шифртекста **CLPO**, **ESIW** и **GAFO**.

Это даст нам вероятности 44 000 000 комбинаций четырех дисков с тремя рядами. И снова мы можем оставить 1 % лучших комбина-

ций, что даст 440 000 наборов тетраграмм. Используем тот же метод, что и для триграмм.

Продолжая поступать таким же образом, получаем пентаграммы, гексаграммы, гептаграммы и т. д. При добавлении каждого следующего диска можно сохранять меньше комбинаций, чем в прошлый раз. Когда количество комбинаций уменьшится до 100, можно выбрать правильную визуально и закончить вскрытие вручную.

Но есть проблема, из-за которой эта процедура может не привести к успеху: даже в нормальном тексте может встретиться триграмма, для которой табличная вероятность равна 0. А если Эмили использует null-символы, то такое может происходить довольно часто. В результате правильный открытый текст будет отвергнут.

Возможное решение – ограничить вероятности триграмм, так чтобы вероятность 0 никогда не возникала. Обозначим  $P(x)$  вероятность строки  $x$ . Если вероятность триграммы  $XYZ$  равна нулю, то можно использовать большее из чисел  $P(X)P(YZ)$  и  $P(XY)P(Z)$ . Я предлагаю делить его, скажем, на 3, потому что  $XYZ$  ни разу не встречалась при подсчете триграмм. Если вероятность  $XYZ$  по-прежнему равна 0, то используйте вероятности отдельных букв, например сделайте  $P(XYZ)$  равной  $P(X)P(Y)P(Z)/10$ .

Другое решение – не перемножать вероятности, а использовать для их комбинирования какую-нибудь другую функцию. Например, можно складывать квадраты вероятностей. Так мы будем вознаграждать часто встречающиеся триграммы, игнорируя редкие.

Если все описанное выше не принесит результата, попробуйте начать процедуру с другого места шифртекста. Например, начните с пятого диска. \*\*

# Фракционирование

---

## *Краткое содержание главы:*

- квадрат Полибия;
- разбиение буквы на меньшие части, например биты или шестнадцатеричные цифры;
- перемешивание и рекомбинация этих частей.

Два главных инструмента криптографии – подстановка и перестановка – были рассмотрены в главах 5–8. Третий фундаментальный элемент – фракционирование. Под этим понимается разбиение обычных структурных элементов языка – букв, слогов и слов – на меньшие единицы и оперирование этими единицами. Меньшими единицами обычно являются биты, десятичные цифры, шестнадцатеричные цифры или цифры в других системах счисления. В этой главе рассматривается фракционирование с использованием цифр в системах счисления по основаниям 2, 3, 5, 6 и 16, а также некоторые другие формы фракционирования.

## 9.1 Квадрат Полибия

Наверное, древнейшим методом представления букв в виде меньших единиц является *квадрат Полибия*, который рассматривался в разделе 4.4. Каждая буква представляется двумя цифрами в пятиричной системе счисления, что дает 25 различных двузначных комбинаций. (У греков не было представления нуля, поэтому цифры начинались с 1.)

Ниже показан квадрат Полибия из раздела 4.4. Каждая буква представляется своими *координатами* в квадрате, т. е. номерами строки и столбца. Например, буква Р находится на пересечении строки 2 и столбца 5, поэтому представляется числом 25. Когда требуется избежать возможной путаницы, мы будем записывать это в виде 2, 5.

	1	2	3	4	5
1	U	V	W	X	Y
2	Z	S	A	M	P
3	L	E	B	C	D
4	F	G	H	IJ	K
5	N	O	Q	R	T

Перемешанный квадрат Полибия  
с ключевым словом **SAMPLE**

Квадрат Полибия сам по себе способен породить ряд шифров. Например, он может порождать простую подстановку путем замены каждой буквы сообщения буквой, стоящей справа от нее в квадрате (U переходит в V) или снизу (U переходит в Z) либо снизу и справа (U переходит в S) или слева (U переходит в P) и т. д. Эту идею можно распространить на многоалфавитный шифр и менять направления, скажем вправо, влево, вниз, вправо, влево, вниз и т. д. Можно также смещаться на 2 буквы или сдвигаться, как ходит шахматный конь.

Квадрат Полибия можно еще использовать для порождения шифра *Полибиева рябь* (Polybius Ripple). Сначала заменим каждую букву сообщения ее координатами, выписав их в одну строку. Начиная со второго числа в этом списке будем прибавлять предыдущее число к текущему. Если сумма больше 5, вычитаем 5, чтобы числа всегда оставались в диапазоне от 1 до 5. Затем преобразуем числа обратно в буквы, снова воспользовавшись квадратом Полибия.

S E N D H E L P  
2232513543323125  
2424453325353411  
M M J B P D C U

Открытый текст  
Координаты  
После применения операции «рябь»  
Шифртекст

Полибиева рябь получает оценку 3. Шифр можно укрепить, взяв другой квадрат Полибия для обратного преобразования координат в буквы.

В разделах 9.2–9.7 рассматривается несколько ручных шифров, разработанных в XIX веке на основе квадрата Полибия. Еще несколько ручных методов я опишу в разделах 9.8–9.11. И в оставшейся части главы обсужу некоторые компьютерные методы.

## 9.2 Шифр Плейфера

Шифр *Playfair* изобрел Чарльз Уитстон в 1854 году. Уитстон хорошо известен инженерам-электротехникам благодаря мосту Уитстона для измерения электрического сопротивления. Уитстон и Уильям Кук изобрели игольчатый телеграфный аппарат на несколько лет раньше, чем Сэмюэль Морзе изобрел телеграфный аппарат с ключом. Кук коммерциализировал игольчатый телеграф в Англии за несколько лет до того, как Морзе основал телеграфную компанию в США.

Шифр Уитстона назван *Playfair*, потому что именно Бэрн Лайон Плейфер, друг Уитстона, который, кстати, был на него очень похож (оба ярко-рыжие и ростом 183 см), выступил его ярким сторонником и убедил Министерство иностранных дел Великобритании использовать этот шифр для дипломатической переписки.

### Историческое отступление

Поскольку этот шифр не назван в честь Уитстона, его имя осталось свободным для другого шифра, который Уитстон изобрел приблизительно в 1860 году и представил на Парижской всемирной выставке 1867 года. Криптограф Уитстона, похожий на большие карманные часы, состоял из двух концентрических колец из жесткого картона и двух подвижных стрелок, соединенных простым часовым механизмом. Внутреннее кольцо можно было стирать и заменять для каждого сообщения. На этом кольце был нанесен 26-буквенный перемешанный алфавит, а на внешнем кольце – стандартный 26-буквенный алфавит и пробел, т. е. всего было 27 позиций. Пользователь вращал длинную стрелку, так чтобы она указывала на букву открытого текста на внешнем кольце, при этом связанная с ней короткая стрелка указывала на букву шифртекста на внутреннем кольце. Когда длинная стрелка совершала полный оборот на 27 позиций, короткая также пробегала 27 позиций, т. е. совершала полный оборот плюс 1 позиция. Поэтому после каждого оборота короткая стрелка начинала движение с новой точки. Эквивалентное устройство, состоявшее из подвижных колец без стрелок, полковник Деций Уодсуорт, начальник артиллерийско-технической службы, изготовил еще в 1817 году, опираясь на чертежи Томаса Джефферсона 1790 года, но с этой идеей навсегда связано имя Уитстона.

Фотографию предоставил Ральф Симпсон. Выгравированная надпись гласит «The Cryptograph. C. Wheatstone Inv'» («Криптограф. Изобретатель Ч. Уитстон»).



Шифр Плейфера основан на квадрате Полибия и зашифровывает сразу две буквы, т. е. биграммы. Для подготовки квадрата следует перемешать алфавит любым из способов, описанных в разделе 5.2. Одна низкочастотная буква алфавита, например J, Q или Z, опущена, чтобы алфавит уместился в квадрат  $5 \times 5$ . (Во французском языке буквы J, Q и Z встречаются часто, поэтому опустить следует букву W. В немецком языке можно опустить любую из букв Q, X или Y.) Если опущенная буква встретится в сообщении, она заменяется какой-то другой буквой. В нашем примере J заменяется на I.

Следующий шаг – разбить сообщение на биграммы, например: ME ET ME TO MO RR OW. Если биграммой оказывается удвоенная буква, то ее следует разбить на две части, обычно для этого в середину вставляется X. (Это веская причина оставить X в квадрате.) Кроме того, если сообщение содержит нечетное число букв, то X добавляется в конец. В итоге сообщение принимает вид ME ET ME TO MO RX RO WX. Теперь его можно зашифровать.

В шифре Плейфера действуют 3 правила: (1) если обе буквы находятся в одной строке, то каждая заменяется буквой справа от нее; (2) если обе буквы находятся в одном столбце, то каждая заменяется буквой снизу от нее; (3) во всех остальных случаях каждая буква заменяется буквой, находящейся в одной с ней строке, но в том столбце, в котором находится вторая буква биграммы. Понятно, что все описанные операции «заворачиваются», т. е. если нужная буква оказывается за пределами квадрата, то берется буква с противоположной его стороны. Таким образом, в квадрате из раздела 9.1 буквой справа от Y будет U, а буквой снизу от Q будет W.

Эти правила можно переформулировать в терминах координат. Пусть шифруется биграмма  $r_1c_1 r_2c_2$ , т. е. первая буква находится на

пересечении строки  $r1$  и столбца  $c1$ , а вторая на пересечении строки  $r2$  и столбца  $c2$ . Тогда три правила принимают вид:

- 1 если  $r1 = r2$ , то выполнить подстановку  $r1, c1+1, r2, c2+1$ ;
- 2 если  $c1 = c2$ , то выполнить подстановку  $r1+1, c1, r2+1, c2$ ;
- 3 в противном случае выполнить подстановку  $r1, c2, r2, c1$ .

Посмотрим, как применяются эти правила к шифрованию сообщения ME ET ME TO MO RX RO WX. Первая биграмма ME. Буквы M и E находятся в разных строках и в разных столбцах, следовательно, применимо правило 3. M находится на пересечении строки 2 и столбца 4, а E на пересечении строки 3 и столбца 2. Поэтому M заменяется буквой на пересечении своей строки, т. е. 2, и столбца буквы E, т. е. 2. В этом месте находится буква S, так что M заменяется на S. Аналогично E заменяется буквой на пересечении строки 3 и столбца 4, т. е. C.

Точно так же биграмма ET заменяется на DO, а вторая биграмма ME – на SC. Буквы T и O находятся в одной строке, поэтому применимо правило 1. Они заменяются буквами справа от них, т. е. T заменяется на N, а O – на Q. Биграмма TO заменяется на NQ.

К MO применимо правило 3, т. е. она заменяется на SR. Буквы R и X находятся в одном столбце, значит, применимо правило 2. RX заменяется на XM. К обеим биграммам RO и WX применимо правило 1, они заменяются соответственно на TQ и XY. Стало быть, все сообщение принимает вид SC DO SC NQ SR XM TQ XY, или после перегруппировки SCDOS CNQSR XMTQX Y.

Диаграммы ниже помогают наглядно представить порядок шифрования биграмм LY, TO и RX.

	1	2	3	4	5
1	U	V	W	X	Y
2	Z	S	A	M	P
3	L	E	B	C	D
4	F	G	H	I	K
5	N	O	Q	R	T

LY → DU

	1	2	3	4	5
1	U	V	W	X	Y
2	Z	S	A	M	P
3	L	E	B	C	D
4	F	G	H	I	K
5	N	O	Q	R	T

TO → NQ

	1	2	3	4	5
1	U	V	W	X	Y
2	Z	S	A	M	P
3	L	E	B	C	D
4	F	G	H	I	K
5	N	O	Q	R	T

RX → XM

Шифр Плейфера применялся военными и дипломатами по крайней мере до 1960 года. Далее мы кратко рассмотрим методику вскрытия этого шифра.

## 9.2.1 Вскрытие шифра Плейфера

Заметим, что каждую букву можно зашифровать только одной из пяти возможных замен, а именно четырьмя буквами в той же строке и одной, расположенной прямо под ней. Для каждой буквы в квадрате имеется еще 24, кроме нее. Из них только 4 буквы в одном столбце с ней могут привести к замене данной буквы той, что находится под



ней. Поэтому вероятность того, что буква будет заменена нижестоящей, равна  $4/24 = 1/6$ . А вероятность, что она будет заменена другой буквой из той же строки, равна  $5/6$ .

Поскольку всего в квадрате 5 строк и в английском языке существует 9 букв с частотой больше 5 %, должно найтись несколько строк, содержащих хотя бы по две высокочастотные буквы. Остальные буквы в этих строках будут встречаться в шифртексте чаще других. Если шифртекста достаточно, то высоки шансы, что от 3 до 5 самых частых букв шифртекста встречаются в одной и той же строке квадрата.

Если удалить все биграммы, содержащие эти буквы, то от 3 до 5 самых частых букв в оставшихся биграммах с большой вероятностью находятся в одной и той же строке квадрата. Знания высокочастотных букв в двух из пяти строк достаточно, чтобы начать реконструкцию квадрата. Следующим шагом будет попытка выделить некоторые вероятные слова.

Шифр Плейфера получает оценку 3. Есть несколько способов повысить его стойкость. Рассмотрим некоторые из них.

## 9.2.2 Укрепление шифра Плейфера

Ниже приведено несколько более стойких вариантов шифра Плейфера.

### NULLFAIR или NOFAIR

В шифртекст можно добавить null-символы с регулярными интервалами, например:

2	3	2	3	2	3		Ключ null-символов равен 23
B	R	C	N	T	F	G	Шифртекст Плейфера
B	R	E	C	N	T	P	Шифртекст с null-символами
F	G	R	I	U	S		
I	U	S		M	H		
M	H		R	A	O		
R	A	O		L			

Шифр Nullfair получает оценку 5.

### PLAYFAIR+1

Это совсем простенькое улучшение добавляет повторяющийся двоичный ключ к шифру Плейфера. Если в ключе встретился бит 1, то используется следующая буква алфавита. Playfair+1 будет более стойким, если длина двоичного ключа нечетна.

0	1	0	1	1	0	1	1	0	1	1	0	Двоичный ключ равен 01011
B	R	C	N	T	F	G	I	U	S	M	H	Шифртекст Плейфера
B	S	C	O	U	F	H	I	V	T	M	I	Шифртекст+1
R	B	P	L									

Playfair+1 получает оценку 5. Этот шифр можно использовать и с троичными цифрами. Цифры в аддитивном ключе малы, по-

этому сложение можно произвести в уме, не пользуясь специальной таблицей.

## Двойной ПЛЕЙФЕР

Шифр Плейфера можно укрепить, применив его дважды. Вторым раундом должен раздвинуть биграммы, созданные на первом. (1) Зашифровать сообщение шифром Плейфера. (2) Либо переместить первую букву в конец, а последнюю в начало, либо добавить null-символы с обоих концов. (3) Выполнить еще один раунд шифра Плейфера. Стойкость будет максимальной, если во втором раунде использовать алфавит, перемешанный по-другому. Двойной Плейфер получает оценку 6.

## ПЛЕЙФЕРОВА РЯБЬ

Это вариант двойного Плейфера с одним проходом по сообщению и одним квадратом Полибия. Обозначим открытый текст  $P_1P_2P_3P_4\dots$ . Начнем с левого конца и зашифруем бигramму открытого текста  $P_1P_2$  шифром Плейфера, в результате получится биграмма  $C_1C_2$ . Затем зашифруем  $C_2P_3$  и получим  $D_2C_3$ . Отметим, что буква  $D_2$  зашифрована дважды. Далее шифруется  $C_3P_4$  – получается  $D_3C_4$  и т. д. На каждом шаге мы продвигается вправо на один символ. Плейферова рябь получает оценку 6.

Поскольку первая буква  $C_1$  и последняя буква  $C_n$  шифртекста зашифрованы только один раз, можно зашифровать их как биграмму и тем самым завершить цикл.

$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_6$	Открытый текст
/	/	/	/	/	/	
$C_1$	$C_2$	$C_3$	$C_4$	$C_5$	$C_6$	Промежуточный текст
$C_1$	$D_2$	$D_3$	$D_4$	$D_5$	$D_6$	Шифртекст

## POLYPLAYFAIR

Использовать два разных квадрата Полибия и переключаться между ними, применяя повторяющийся ключ. Например, ключ 11212 означает, что в каждой последовательности из пяти биграмм первая, вторая и четвертая биграммы должны шифроваться квадратом 1, а третья и пятая – квадратом 2. Эту идею можно обобщить на три и более квадратов, но время инициализации увеличится. С двумя квадратами и ключом не длиннее 10 цифр шифр PolyPlayfair получает оценку 5. Если ключ генерируется алгоритмом сцепленных цифр и первый квадрат используется, когда цифра ключа от 0 до 4, а второй, когда цифра ключа от 5 до 9, то оценка повышается до 6. (Примечание: при использовании четности в последовательности сцепленных цифр период будет гораздо короче, что ослабляет шифр.)

## ПЕРЕСТАНОВКА

Шифртекст, получившийся в результате шифрования открытого текста шифром Плейфера, можно подвергнуть перестановке. Это может быть сложная столбцовая перестановка, рассмотренная в разделе 7.2, или простое кусочное обращение с помощью шифра Базери типа 4 из раздела 4.6.1. Шифр Плейфера со столбцовой перестановкой получает оценку 7, а с кусочным обращением – оценку 5.

## 9.3 Шифр Two Square

Шифр *Two Square* (два квадрата), который иногда называют *двойным Плейфером*, – усовершенствованная версия шифра Плейфера. Его изобрел французский криптограф-любитель Феликс-Мари Дела-стель и описал в своей книге «*Traité Élémentaire de Cryptographie*», изданной в 1902 году. Как следует из названия, в шифре используется два квадрата Полибия вместо одного и соответственно два перемешанных алфавита. Два квадрата можно расположить бок о бок или один под другим. Ниже показано расположение по горизонтали. Два квадрата перемешаны с помощью ключевых слов FIRST и SE-COND, а буква Q из сеток 5×5 исключена.

F	I	R	S	T	M	P	R	T	U
A	B	C	D	E	V	W	X	Y	Z
G	H	J	K	L	S	E	C	O	N
M	N	O	P	U	D	A	B	F	G
V	W	X	Y	Z	H	I	J	K	L

Как и в шифре Плейфера, сообщение шифруется по две буквы за раз, т. е. шифруются биграммы. Чтобы зашифровать бигramму SO, мы находим букву S в левом квадрате и букву O в правом квадрате. Вместо S подставляется буква из правого квадрата, находящаяся в той же строке, что S, и в том же столбце, что O, т. е. T. Вместо O подставляется буква из левого квадрата, находящаяся в той же строке, что O, и в том же столбце, что S, т. е. K. Следовательно, биграмма SO преобразуется в TK.

В отличие от шифра Плейфера, не нужно разбивать удвоенные буквы. Две буквы могут оказаться в разных строках в двух квадратах. Например, биграмма SS преобразуется в MK. В большинстве случаев удвоенной букве в шифртексте не будет соответствовать удвоенная буква в открытом тексте.

Ниже процесс подстановки показан наглядно.

F	I	R	S	----->	T	U	S преобразуется в T			
A	B	C	I	E	V	W	X	I	Z	O преобразуется в K
G	H	J	K	----->	O	N	SO преобразуется в TK			
M	N	O	P	U	D	A	B	F	G	
V	W	X	Y	Z	H	I	J	K	L	

У шифра Two Square имеется существенная слабость: когда обе буквы биграммы оказываются в одной строке квадрата, подстановка сводится просто к их обращению. Например, ST преобразуется в TS. Эта слабость, называемая *прозрачностью*, иногда позволяет раскрыть целое слово. Например, SU ND AY преобразуется в US DN YA.

Чтобы помешать этому, я предлагаю *правило той же строки*: если две буквы находятся в одной строке, то они заменяются буквами, расположенными прямо под ними, а если эта строка была последней – то буквами, находящимися в первой строке. Например, ST преобразуется в DY, а VI в FP.

С правилом той же строки шифр Two Square получает оценку 4. Будем называть этот вариант *Two Square B*.

Немцы восприняли название двойной Плейфер буквально. Они зашифровывали каждую биграмму шифром Two Square, а затем еще раз шифром Two Square с теми же двумя квадратами. Результатом по существу является подстановка биграмм общего вида (раздел 6.5).

Методы, использованные для укрепления шифра Плейфера, можно применить и для укрепления шифра Two Square. Результирующие шифры *TwoSquare+1* и *Two Square с рябью* получают такие же оценки. Приведу дополнительный вариант.

## PLAYFAIR TWO SQUARE

В шифре Two Square используется два квадрата Полибия. Каждый из них можно было бы использовать также для шифра Плейфера. Это наводит на мысль о гибридном методе. Как и раньше, будем использовать числовой ключ, управляющий шифрованием последовательных биграмм. Цифра 1 означает, что биграмма шифруется шифром Плейфера в левом квадрате, цифра 2 – что шифром Плейфера в правом квадрате, а цифра 3 – что шифром Two Square или Two Square B. Оптимально, когда ключ содержит каждую цифру хотя бы по одному разу. Поскольку шифр Two Square более стойкий, чем шифр Плейфера, 3 должна встречаться чаще, чем 1 и 2. Примерно 50 % было бы в самый раз. Шифр *Playfair TwoSquare* получает оценку 6.

## 9.4 Шифр Three Square

Шифр *Three Square* (три квадрата) – моя собственная идея. Других особых достоинств он не имеет. Я включил его только потому, что во время работы над этой книгой где-то прочел, что шифр Two Square, мол, нельзя обобщить на число квадратов больше двух. А я люблю принимать вызовы.

Как следует из названия, в методе Three Square используется три квадрата Полибия. Эти квадраты следует хорошо перемешать с применением независимых ключей. Three Square шифрует 3 буквы за раз, т. е. триграммы. Поэтому он более стойкий, чем Two Square.

F I R S T	U V W X Y	L M N O P
A B C D E	Z S E C O	S U V W X
G H J K L	N D A B F	Y Z T H I
M N O R U	G H I J K	R D A B C
V W X Y Z	L M P R T	E F G J K

Основная идея заключается в том, чтобы заменить каждую букву буквой, взятой из квадрата справа. Заменяющая буква находится на пересечении той же строки и столбца, содержащего следующую букву триграммы.

Предположим, что требуется зашифровать триграмму THE. Мы шифруем букву T в первом квадрате, букву H во втором и букву E в третьем, как показано на рисунке ниже.

F I R S <b>T</b>	U V W X Y	L M N O P
A B C D E	Z S E C O	S U V W X
G H J K L	N D A B F	Y Z T H I
M N O R U	<b>G H</b> I J K	R D A B C
V W X Y Z	L M P R T	<b>E</b> F G J K

Замена T находится во втором квадрате на пересечении той же строки и столбца, содержащего H; это буква V. Замена H находится в третьем квадрате на пересечении той же строки и столбца, содержащего E; это буква R. Замена E находится в первом квадрате на пересечении той же строки и столбца, содержащего T; это буква Z. Таким образом, THE преобразуется в VRZ.

На картинке это выглядит так:

F I R S <b>T</b> ----	V W X Y	L M N O P	Шифрование
A B C D E	Z S E C O	S U V W X	T преобразуется в V
G H J K L	N D A B F	Y Z T H I	H преобразуется в R
M N O R U	G <b>H</b> -----	R D A B C	E преобразуется в Z
-----> <b>Z</b>	L M P R T	<b>E</b> ----->	THE преобразуется в VRZ

Дешифрирование производится в обратном направлении. Поскольку первая буква триграммы шифртекста VRZ взята из второго квадрата, дешифрирование с него и начинается:

F I R S T	<b>T</b> -----	V W X Y	L M N O P	Дешифрирование
A B C D E	Z S E C O	S U V W X	V преобразуется в T	
G H J K L	N D A B F	Y Z T H I	R преобразуется в H	
M N O R U	G H	-----< <b>R</b>	D A B C	Z преобразуется в E
-----< <b>Z</b>	L M P R T	<b>E</b> -----<	VRZ преобразуется в THE	

У шифра Three Square проблема с буквами, попадающими в одну строку, серьезнее, чем у Two Square. В триграмме вида XYZ может оказаться, что какие-то из пар X и Y, Y и Z, Z и X находятся в одной

и той же строке. Необходимо два дополнительных правила, предотвращающих прозрачность – когда буква представляет саму себя.

**Правило 1.** Если две соседние буквы триграммы попадают в одну строку, то первая из них шифруется как буква справа от второй; при необходимости самый правый столбец «заворачивает» на самый левый. Например, в триграмме SUB буква S находится в первой строке первого квадрата, а U – в первой строке второго квадрата. Поэтому S заменяется на V вместо U. Аналогично в триграмме LET буква T находится в третьей строке третьего квадрата, а L в третьей строке первого квадрата. Поэтому T заменяется на G, а не на L.

Следующая диаграмма иллюстрирует правило 1. Не будь его, буква S в триграмме SUB была бы заменена буквой U. А так она заменяется буквой справа от U, находящейся в середине квадрата, конкретно V. Без правила 1 буква T в триграмме LET была бы заменена буквой L. Вместо этого она заменяется буквой справа от L в левом квадрате. При этом необходимо завернуть с пятого столбца на первый, где находится буква G.

F I R S	----->U V	W X Y	L M N O P	В триграмме SUB
A B C D E	Z S E C O	S U V W X	S U V W X	S преобразуется в V, а не в U
G H J K L	N D A B F	Y Z T----->	Y Z T----->	
M N O R U	G H I J K	R D A B C	R D A B C	В триграмме LET
V W X Y Z	L M P R T	E F G J K	E F G J K	T преобразуется в G, а не в L

**Правило 2.** Если все три буквы триграммы попадают в одну строку, то каждая заменяется буквой, находящейся прямо под ней; при необходимости нижняя строка заворачивает на верхнюю. Таким образом, FUN была бы заменена на AZV, а WRE – на IXL.

С этими правилами шифр Three Square получает оценку 5.

## ШИФР PLAYFAIR THREESQUARE

В шифре Three Square используются три квадрата Полибия. Любой из них можно было бы использовать для шифра Плейфера. Это наводит на мысль о гибридном методе. Мы можем использовать числовой ключ, например 1, 4, 1, 3, 4, 2, 4, управляющий тем, как шифруются соседние биграммы или триграммы. 1 означает, что нужно шифровать следующие 2 буквы как биграмму методом Плейфера, примененным к первому квадрату; 2 – шифровать следующие 2 буквы как биграмму методом Плейфера, примененным ко второму квадрату; 3 – шифровать следующие 2 буквы как биграмму методом Плейфера, примененным к третьему квадрату; 4 – шифровать следующие 3 буквы как триграмму методом Three Square. Оптимально, если числовой ключ содержит каждую цифру хотя бы по одному разу. Поскольку метод Three Square значительно более стойкий, чем шифр Плейфера, цифра 4 должна встречаться чаще, чем 1, 2 и 3. Примерно 50 % будет в самый раз. То есть 4 должно встречаться столько раз, сколько 1, 2 и 3 вместе взятые. Эквивалентно, можно генерировать

случайные числа от 1 до 6 и использовать Three Square, когда генератор выдал 4, 5 или 6.

Поскольку в шифре Playfair ThreeSquare используются как биграммы, так и триграммы, примерно половина биграмм и две трети триграмм не будут попадать на точные границы. Это значит, что стойкость шифра повышается сильнее, чем в случае Playfair TwoSquare. Playfair ThreeSquare получает оценку 7.

Можно объединить шифры Плейфера, Two Square и Three Square в еще более сложный шифр, который, без сомнения, окажется более стойким, но Playfair ThreeSquare и так уже на пределе возможностей шифровальщика-человека. А при усложнении пострадают скорость и точность.

Существует противоположный подход, который я называю *раздвижным Three Square*. Сгруппируем открытый текст в строки, содержащие четыре блока по три символа. Зашифруем каждый блок шифром Three Square. Возьмем последнюю букву блока 1 и первую букву блока 2 и зашифруем эту биграмму шифром Плейфера на первом квадрате Полибия. Возьмем последнюю букву блока 2 и первую букву блока 3 и зашифруем эту биграмму шифром Плейфера на втором квадрате Полибия. Возьмем последнюю букву блока 3 и первую букву блока 4 и зашифруем эту биграмму шифром Плейфера на третьем квадрате Полибия. Это повышает стойкость шифра Three Square, не сильно увеличивая сложность и время. Необходимо всюду использовать правило одной и той же строки.

## 9.5 Шифр Four Square

Шифр *Four Square* изобрел Феликс-Мари Деластьель примерно в 1890 году и описал его в своей книге «*Traité Élémentaire de Cryptographie*», опубликованной через три месяца после его смерти, в 1902 году. Деластьель изобрел шифр Two Square, упростив Four Square и сделав его немного менее безопасным. Однако если применяется правило одной и той же строки, описанное в разделе 9.3, то оба шифра можно считать одинаковыми по стойкости.

Как следует из названия, в шифре Four Square используется четыре квадрата Полибия. Два из них содержат стандартный алфавит, а два других – алфавиты, перемешанные независимыми ключами. Сообщение шифруется по две буквы за раз, т. е. биграммами.

Ниже показан пример конфигурации.

a b c d e	M O N K E
f g h i j	Y A B C D
k l m n o	F G H I J
p r s t u	L P R S T
v w x y z	U V W X Z

Ключевое слово: **MONKEY**

U V W X Y	a b c d e	Ключевое слово: <b>CHIMPANZEE</b>
C H I M P	f g h i j	
A N Z E B	k l m n o	
D F G J K	p r s t u	
L O R S T	v w x y z	

При шифровании применяется знакомая прямоугольная схема. Находим две буквы открытого текста в стандартных алфавитах и заменяем их буквами из противоположных квадратов, как на рисунке ниже.

a b c d e	M O N K E	Открытый текст <b>TH</b> заменяется шифртекстом <b>RM</b>
f g h i j	Y A B C D	
k l m n o	F G H I J	
p r s t	R S T	
v w x y z	U V W X Z	

U V W X Y	a b c d e
C H I M	h i j
A N Z E B	k l m n o
D F G J K	p r s t u
L O R S T	v w x y z

Поскольку обе буквы открытого текста никогда не могут находиться в той же строке или том же столбце сетки 10×10, не нужны никакие специальные правила, в частности для разделения удвоенных букв. Null-символ может понадобиться, только чтобы дополнить последнюю бигramму. Шифр Four Square получает оценку 5.

## Циклический метод

Чтобы немного увеличить стойкость, можно дополнить кусочное обращение из раздела 4.6.1 простой перестановкой, в которой используется повторяющийся числовой ключ, например 1, 3, 1, 4, 2, 6. Разобьем шифртекст на блоки по 7 символов или любой другой нечетной длины. Над каждым блоком запишем последовательные цифры ключа. Циклически сдвинем каждый блок влево на число позиций, равное соответствующей ему цифре ключа. Например, если над блоком написана цифра 4, то 4 крайние левые цифры следует переместить в правый конец блока:

1	3	1	4	2	6	1	3
BSMTPSZ	LDNTPRB	EXYFHW	IXRCNIO	OKLPRSC	UBEACZV	NEULHDF	PLECNGU
SMTPSZB	TPRBLDN	XYFHWME	NIOIXRC	LPRSCOK	VUBEACZ	EULHDFN	CNGUPL

Шифр Four Square, дополненный циклическим методом, получает оценку 6.



## Метод деления пополам

Еще один подход к укреплению шифра Four Square – заранее применить к сообщению перестановку. Пусть имеется сообщение AMBASSADOR WILKINS ASSASSINATED KABUL TODAY. Оно состоит из 39 букв. Поделив 39 на 2 и округлив, получим 20. Запишем сообщение в две строки по 20 букв в каждой и считаем биграммы по вертикали. Зашифруем эти биграммы шифром Four Square.

```
AMBASSADORWILKINSASS
ASSINATEDKABULTODAYX
AA MS BS AI SN SA AT DE OD RK WA IB LU KL IT NO SD AA SY SX
```

При таком подходе не наблюдается нормальная частота биграмм или контактов, характерная для английского языка. Шифр Four Square, дополненный делением пополам, получает оценку 7.

## 9.6 Шифр Bifid

Рассмотрим еще один исторический ручной шифр на основе квадрата Полибия  $5 \times 5$ . Это шифр *Bifid* (двухчастный), также изобретенный Феликсом-Мари Деластановом в 1890-х годах. Он состоит из трех шагов: (1) преобразовать буквы в соответствующие координаты Полибия, (2) изменить порядок координат и (3) преобразовать координаты обратно в буквы. В оригинальной версии Деластанов выписывал все сообщение, располагая под каждой буквой ее координаты (выше строку, ниже столбец), затем объединял пары координат, читая их по горизонтали, сначала из верхней строки, потом из нижней.

В современной версии сообщение разбивается на блоки фиксированной длины, которая должна быть нечетной, например 5, 7 или 9. Если бы длина блока была четной, то Эмили смогла бы разбить блоки на биграммы.

Первый шаг – преобразовать буквы в координаты относительно квадрата Полибия. Пусть длина блока равна 5. Пять букв открытого текста обозначим  $X_1, X_2, X_3, X_4, X_5$ , а их координаты (строку и столбец) –  $R_1C_1, R_2C_2, R_3C_3, R_4C_4, R_5C_5$ . Каждый символ  $R$  и  $C$  может принимать значения от 1 до 5. Эти пары координат записываются под каждой буквой, как показано на рисунке ниже.

$X_1$	$X_2$	$X_3$	$X_4$	$X_5$
$R_1$	$R_2$	$R_3$	$R_4$	$R_5$
$C_1$	$C_2$	$C_3$	$C_4$	$C_5$

Затем координаты читаются по строкам в порядке  $R_1R_2, R_3R_4, R_5C_1, C_2C_3, C_4C_5$ . В примере ниже слово MAJOR зашифровано квадратом Полибия, перемешанным ключевым словом SAMPLE.

	1 2 3 4 5	<u>M A J O R</u>	Открытый текст
1	S A M P L	1 1 3 4 4	По вертикали записаны координаты
2	E B C D F	3 2 4 2 3	13, 12, 34, 42, 43
3	G H I J K		
4	N O R T U	11 34 43 24 23	Координаты считываются по горизонтали
5	V W X Y Z	S J R D C	Шифртекст

Отметим, что третий набор координат букв шифртекста, **R5C1**, содержит пару (строка, столбец). Это означает, что третья буква шифртекста происходит из той же строки (**R5**) квадрата Полибия, что и пятая буква открытого текста, и из того же столбца (**C1**), что и первая буква открытого текста. Присутствие строки и столбца в составе пары координат называется *натуральным*.

Поскольку в каждой строке и в каждом столбце квадрата 5 букв, шанс, что третья буква шифртекста совпадет с пятой буквой открытого текста, составляет 1 к 5, и такой же шанс 1 к 5 у совпадения третьей буквы шифртекста с первой буквой открытого текста. То есть 20 % за то, что **R5C1** совпадает с **R5C5**, и столько же за то, что **R5C1** совпадает с **R1C1**. В рассматриваемом примере так и случилось. Пятая буква открытого текста – R, и третья буква шифртекста – тоже R.

Теперь рассмотрим первую букву шифртекста, **R1R2**. Это пара (строка, строка), а не (строка, столбец). Здесь лишь первая координата, **R1**, занимает правильное место в паре (строка, столбец). Вторая координата, **R2**, – номер строки в позиции столбца. Присутствие в составе пары только номеров строк или только номеров столбцов называется *полунатуральным*. Это означает, что первая буква шифртекста происходит из той же строки квадрата Полибия, что и первая буква открытого текста. Поэтому есть 20-процентный шанс, что первая буква шифртекста совпадает с первой буквой открытого текста. То же самое относится ко второй, четвертой и пятой буквам шифртекста. Каждая из них происходит либо из той же строки, либо из того же столбца, что и одна из букв открытого текста. Следовательно, 20 % за то, что она совпадает с этой буквой открытого текста. Такое наблюдается в нашем примере – вторая буква шифртекста, **J**, совпадает с третьей буквой открытого текста.

Это серьезная слабость шифра Bifid, благодаря которой Эмили может легко угадать и разместить вероятные слова. С другой стороны, если буквы открытого и шифртекста различны, то мы точно знаем, что они находятся в одной и той же строке или одном и том же столбце. В нашем примере первая буква открытого текста **R1C1** равна M, а первая буква шифртекста **R1R2** равна S. Это значит, что M и S должны находиться в одной строке квадрата Полибия. Когда Эмили точно или предположительно устанавливает какое-то слово, образуется несколько таких эквивалентностей. А это, в свою очередь, дает возможность разместить дополнительные слова. Набрав достаточно таких пар букв, Эмили сможет реконструировать квадрат.

Из-за этих слабостей шифр Bifid получает оценку 3.

### 9.6.1 Bifid с сопряженной матрицей

Эти проблемы можно устранить, если использовать другой квадрат Полибия для обратного преобразования координат в буквы. В примере ниже квадрат 2 дает шифртекст **VBJEF**.

	<u>Квадрат 1</u>	Открытый	<u>Квадрат 2</u>	Шифртекст
	1 2 3 4 5	М А Ж О Р	1 2 3 4 5	11 34 43 24 23
1	S A M P L	1 1 3 4 4	1 V W X Y Z	V B J E F
2	E B C D F	3 2 4 2 3	2 D I F E R	
3	G H I J K		3 N T A B C	
4	N O R T U		4 G H J K L	
5	V W X Y Z		5 M O P S U	

Шифр Bifid с двумя квадратами Полибия высокопарно называется *Bifid с сопряженной матрицей*. В данном контексте матрица – это просто прямоугольный массив букв или символов. Bifid с сопряженной матрицей получает оценку 5.

Есть несколько способов укрепить шифр Bifid. Один из них – варьировать длину блока в соответствии с повторяющимся ключом, например 5, 11, 7. Это значит, что длина блоков изменяется периодически: 5, 11, 7, 5, 11, 7, 5, .... Если вам так больше нравится, можете воспользоваться цепным генератором цифр и преобразовывать выданные им цифры в нечетные длины блоков. Например, так:

Цифра	0	1	2	3	4	5	6	7	8	9
Длина	5	7	9	11	13	5	7	9	11	13

Таким образом, если генератор выдал цифры 3, 6, 2, 7, ..., то длины блоков будут равны 11, 7, 9, 9, ....

При использовании короткого повторяющегося ключа и сопряженных матриц шифр получает оценку 6. Если ключ длинный или длины блоков берутся от генератора случайных чисел, то оценка повышается до 7.

Похожая идея заключается в том, чтобы в каждом блоке считать координаты, начиная с разных точек. Последовательность начальных точек можно задать с помощью числового ключа. Если длина блока равна L, то каждый элемент ключа должен быть числом от 1 до 2L. Число от 1 до L означает, что начальная позиция находится в верхней строке координат, а от L+1 до 2L – что в нижней, например:

1	2	3	4	5	6	7	Начальные позиции
8	9	10	11	12	13	14	для считывания координат

Координаты должны читаться парами слева направо. Ниже показан порядок чтения координат по ключу 4, 9. Начальные позиции 4 и 9 выделены.

			4						9						Числовой ключ
12	13	14	1	2	3	4		7	8	9	10	11	12	13	Блок 1 позиция 4
5	6	7	8	9	10	11		14	1	2	3	4	5	6	Блок 2 позиция 9

Еще один способ укрепить шифр Bifid – использовать более стойкую перестановку для перемешивания координат. В стандартном Bifid с блоком длины  $L$  берется  $2L$  координат и записывается в блок  $2 \times L$ . Координаты записываются по вертикали, а считываются по горизонтали. Нетрудно узнать в этом очень простую маршрутную перестановку, описанную в разделе 7.1. В главе 7 рассматривались и более стойкие перестановки, прежде всего столбцовая. Примером шифра такого рода может служить шифр *ADFGVX*, предложенный офицером разведки лейтенантом Фрицем Небелем. Он использовался немцами во время Первой мировой войны. В этом шифре координаты, представленные буквами A, D, F, G, V, X, перемешиваются столбцовой перестановкой, а затем передаются в виде строки букв. Этот шифр получает оценку 5.

Использование более длинных блоков, скажем по 20 символов, дало бы 40 координат (данный метод допускает как четную, так и нечетную длину блока). Этого достаточно для эффективного применения столбцовой перестановки с целью перемешивания координат. Или можно было бы вернуться к оригинальной идее Деластеля и брать координаты для всего сообщения единым блоком. В любом случае применение сопряженных матриц к этому шифру повышает оценку до 8, а использование двойной столбцовой перестановки – до 10. В предположении, что имеется четыре независимых ключа и хорошо перемешанные алфавиты, шифр становится не вскрываемым с помощью карандаша и бумаги. Назовем его *двойным столбцовым Bifid*.

## 9.7 Диагональный Vifid

Одна из вариаций на тему шифра Bifid – записывать координаты Полибия вертикально под каждой буквой, как обычно, но считать их по диагонали, направляясь из левого верхнего в правый нижний угол (с юго-запада на северо-восток). Такой шифр называется *лево-диагональным* или *антидиагональным*. (В геральдике такая полоса черного цвета обозначает незаконнорожденность.) По достижении последней буквы производится переход к первому столбцу (выделенная цифра 1). Преимущество – в отсутствии натуральных и полунатуральных пар, помогающих Эмили указывать слова. Пример приведен ниже.

	1 2 3 4 5	M A J O R	Открытый текст
1	S A M P L	1 1 3 4 4 <b>1</b>	Координаты записываются по вертикали
2	E B C D F	/ / / / /	13, 12, 34, 42, 43
3	G H I J K	3 2 4 2 3	
4	N O R T U		
5	V W X Y Z	31 23 44 24 31	Координаты считываются по диагонали
		G C T D C	Шифртекст

Диагональный Bifid получает оценку 4, а в сочетании с сопряженными матрицами – оценку 5. При использовании сопряженных матриц и блоков переменной длины оценка повышается до 6. В отличие от классического Bifid, диагональный допускает блоки любой длины, четной и нечетной.

## 9.8 Квадраты 6×6

Если сообщения содержат много чисел, то может быть выгоднее использовать квадрат Полибия 6×6, а не 5×5. Такой квадрат позволяет представить полный алфавит плюс цифры от 0 до 9. Нет нужды исключать из алфавита букву J или Q. При ручном шифровании нужно быть внимательным и не путать буквы O, I, Z, S, G с цифрами 0, 1, 2, 5, 6. С этой целью иногда вводят специальные соглашения, например подчеркивают все цифры. Мне это кажется неудобным и чреватым ошибками. Я обычно просто выделяю признаки, отличающие похожие символы, например пишу букву I с очень широкими засечками.

Все описанные выше методы, Плейфера, Two Square, Three Square, Four Square и Bifid, как и их вариации, можно использовать с квадратами 6×6.

## 9.9 Шифр Trifid

Если вам нравятся квадраты, то как насчет кубов? Феликс-Мари Делагель изобрел в 1890-х годах еще один метод фракционирования – шифр *Trifid* (трехчастный). В нем каждая буква алфавита представляется не двумя пятеричными цифрами, а тремя троичными. В результате получается  $3 \times 3 \times 3 = 3^3$  комбинаций трех цифр. Этого достаточно для всех 26 букв алфавита плюс один дополнительный символ. Делагель добавил к алфавиту знак +.

Дополнительный символ + можно было бы использовать как знак препинания и как сигнал, означающий, что следующую букву открытого текста следует интерпретировать как цифру. Соответствие могло бы быть таким: +A = 1, +B = 2, ..., +J = 0. Другие буквы алфавита тоже можно было использовать как специальные символы, например +K – точка, +L – запятая и т. д.

Как сочетания двух цифр можно представить в виде квадрата  $5 \times 5$ , заполненного буквами, так сочетания трех цифр можно представить в виде куба  $3 \times 3 \times 3$ . Три цифры в каждой тройке интерпретируются как координаты, показывающие, в каком месте куба находится буква. Часто эти координаты называются слой, строка и столбец.

Ниже приведены все 27 троичных комбинаций и эквивалентные им буквы; для перемешивания использовалось ключевое слово EXAMPLE с чередующимися столбцами. Например, буква N представлена тройкой 102, поэтому находится в слое 1, строке 0 и столбце 2 куба  $3 \times 3 \times 3$ .

E 000	Q 100	+ 200	Пример таблицы подстановки в шифре Trifid
X 001	O 101	R 201	
A 002	N 102	S 202	
M 010	K 110	T 210	
P 011	J 111	U 211	
L 012	I 112	V 212	
B 020	H 120	W 220	
C 021	G 121	Y 221	
D 022	F 122	Z 222	

Шифр Trifid работает по тому же принципу, что и Bifid. Открытый текст разбивается на блоки фиксированного размера, который не должен быть кратен 3. Три цифровые координаты записываются вертикально под каждой буквой сообщения, а затем считываются по горизонтали группами по 3. После этого они преобразуются обратно в буквы по той же таблице. В примере ниже открытый текст SEND HELP зашифрован с размером блока 4.

S E N D	H E L P	201 000 022 022	Шифртекст REDDQ BKC
2 0 1 0	1 0 0 0	R E D D	
0 0 0 2	2 0 1 1	100 020 110 021	
2 0 2 2	0 0 2 1	Q B K C	

Те же методы анализа, что использовались для Bifid, можно применить и к Trifid, и оценки у них такие же. Можно взять две разные таблицы подстановки для преобразования букв в цифры и обратного преобразования цифр в буквы. Можно варьировать размеры блоков. Можно в каждом блоке начинать чтение цифр с другого места. Можно использовать стойкую перестановку для перемешивания троичных цифр.

Возникает естественный вопрос: существует ли для шифра Trifid аналог диагонального Bifid? Преимущество диагонального Bifid перед оригинальным заключается в том, что в диагональном варианте не возникают полунатуральные пары, ослабляющие оригинальный шифр. В аналогичной диагональной версии Trifid средняя цифра каждой группы будет треть-натуральной, так что это преимущество утрачивается. Однако проблема натуральности исчезает, если ис-

пользовать два разных перемешанных алфавита, один для записи цифр, а другой для считывания. Диагональный Trifid с двумя алфавитами получает оценку 5.

## 9.10 Шифр *Three Cube*

Я писал предыдущий абзац о шифре Trifid и внезапно понял, что конфигурация в виде куба  $3 \times 3 \times 3$  прямым ведет к трехмерному аналогу шифра Two Square, описанного в разделе 9.3. Легко наглядно представить шифр Two Square в двух измерениях, но гораздо труднее визуализировать трехмерный куб, поэтому я опишу новый шифр исключительно в терминах координат. Назову его *Three Cube* (три куба).

При использовании Two Square шифруются сразу две буквы по двум таблицам подстановки, следовательно, Three Cube будет шифровать сразу три буквы по трем таблицам подстановки. Ниже показан набор из трех таблиц, хорошо перемешанных ключевыми словами COLUMBIA, STANFORD и HOPKINS. Обозначим эти три таблицы S, T и U. Здесь S означает Substitution (подстановка), T – Table (таблица), а U – просто следующая буква алфавита.

Каждая таблица подстановки сопоставляет 26 буквам алфавита и символу + одно из 27 троичных трехзначных чисел.

Таблица подстановки S	Таблица подстановки T	Таблица подстановки U
A 000 Y 100 H 200	A 000 U 100 W 200	H 000 D 100 L 200
N 001 I 101 T 201	E 001 N 101 S 201	A 001 Q 101 W 201
X 002 K 102 O 202	P 002 G 102 B 202	J 002 Y 102 P 202
B 010 W 110 E 210	+ 010 Q 110 L 210	V 010 N 110 C 210
J 011 L 111 Q 211	D 011 O 111 Y 211	I 011 F 111 M 211
V 012 F 112 Z 212	K 012 I 112 T 212	E 012 T 112 X 212
C 020 R 120 U 220	X 020 V 120 C 220	R 020 + 120 S 220
D 021 + 121 G 221	F 021 R 121 M 221	Z 021 O 121 G 221
P 022 M 122 S 222	H 022 J 122 Z 222	K 022 B 122 U 222

Как и в случае Trifid, мы начинаем с того, что выписываем под каждой буквой соответствующую ей тройку цифр. Все три цифры для первой буквы берутся из таблицы подстановки S, три цифры для второй буквы – из таблицы T, а три цифры для третьей буквы – из таблицы U. Ниже эта схема продемонстрирована на примере триграммы FLY.

	F L Y	Записывается тройками
S1 T1 U1	1 2 1	S1S2S3, T1T2T3 и U1U2U3
S2 T2 U2	1 1 0	Открытый текст FLY
S3 T3 U3	2 0 2	

Затем цифры считываются слева направо по три и горизонтальные тройки преобразуются обратно в буквы. Казалось бы, естественно использовать таблицу S для преобразования верхней строки, таблицу T для преобразования средней и таблицу U для нижней. Но тогда с вероятностью 1 к 9 верхняя строка совпала бы с левым столбцом, поэтому первая буква открытого текста была бы заменена ей самой. То есть с вероятностью 1 к 9 тройки S1S2S3 и S1T1U1 совпали бы. То же самое справедливо для средней и верхней строк. Назовем такую ситуацию – совпадение одной буквы – *частично натуральной*.

По этой причине таблица подстановки S используется для второй строки, таблица T – для третьей, а таблица U – для первой. При этом сама возможность натуральности исключается. Схема выглядит следующим образом:

	<b>F L Y</b>	Считываются тройки
<b>U U U</b>	<b>1 2 1</b>	<b>121 110 202</b>
<b>S S S</b>	<b>1 1 0</b>	<b>0 W B</b>
<b>T T T</b>	<b>2 0 2</b>	Шифртекст <b>OWB</b>

Поскольку при шифровании вручную трудно держать все в памяти, я рекомендую записывать выбранную таблицу подстановки над каждой тройкой цифр. Это напоминает запись буквы ключа над каждой буквой открытого текста при работе с шифром Беласо (раздел 5.5). Ниже приведен пример применения шифра Three Cube к сообщению FLY TO ROME.

<b>STU STU STU</b>	<b>U S T U S T U S T</b>
<b>FLY TOR OME</b>	<b>121 110 202 210 012 110 220 021 212</b>
<b>121 210 220</b>	<b>0 W B C V Q S D T</b>
<b>110 012 021</b>	
<b>202 110 212</b>	Шифртекст <b>OWBCV QSDT</b>

Шифр Three Cube получает оценку 7.

Есть простой способ укрепить шифр Three Cube. Вместо того чтобы преобразовывать тройки цифр обратно в буквы, последовательно ротируя таблицы подстановки, как мы только что проделали, можно использовать ключ для задания порядка чтения из таблиц. Ключ будет состоять из букв S, T, U в каком-то произвольном порядке, например SUTUTTUUSTS. Длина ключа не должна быть кратна 3. Я называю этот вариант *Three Cube Plus*. Ниже он продемонстрирован на примере шифрования сообщения FLY TO ROME.

<b>STU STU STU</b>	<b>S U T U T T U U S T S</b>
<b>FLY TOR OME</b>	<b>121 110 202 210 012 110 220 021 212</b>
<b>121 210 220</b>	<b>+ N B C K Q S Z Z</b>
<b>110 012 021</b>	
<b>202 110 212</b>	Шифртекст <b>+NBCK QSZZ</b>



При использовании шифра Three Cube Plus приблизительно трети букв будут соответствовать частично натуральные тройки. То есть одна из трех записанных цифр будет совпадать с одной из считанных. Однако Эмили не знает, для каких букв имеет место этот эффект, и не сможет его эксплуатировать.

Шифр Three Cube Plus получает оценку 9.

А нельзя ли, спросите вы, повысить оценку до 10, не делая шифр слишком сложным для ручного использования? Хороший вопрос. Для начала увеличим количество таблиц подстановки с 3 до 6. Назовем их S, T, U, V, W и X. Вместо того чтобы ротировать таблицы, в которые записываются тройки, строго регулярно, например STU, STU, STU, ..., мы заведем еще один ключ, содержащий эти 6 букв в перемешанном порядке. Ключ записи пусть будет TWXUSTTVWV, а ключ чтения VWTXXSUSVTU. В идеале длины этих ключей должны быть взаимно простыми числами и ни одна длина не должна делиться на 3. В данном случае выбраны длины 10 и 11. Назовем этот шифр *Three Cube Super*. Ниже приведен пример его применения к сообщению FLY TO NEW YORK.

<u>TWX</u>	<u>UST</u>	<u>TWV</u>	<u>VTW</u>		<u>V</u>	<u>W</u>	<u>T</u>	<u>X</u>	<u>X</u>	<u>S</u>	<u>U</u>	<u>S</u>	<u>V</u>	<u>T</u>	<u>U</u>	<u>V</u>
FLY	TON	EWY	ORK		021	202	121	121	100	221	001	021	111	210	020	012
021	121	001	210		S	P	R	C	V	G	A	D	+	L	R	M
202	100	021	020													
121	221	111	012		Шифртекст <b>SPRCV GAD+L RM</b>											

Шифр Three Cube Super получает оценку 10. Это еще один не-вскрывааемый ручной шифр.

## 9.11 Прямоугольные сетки

До сих пор мы обсуждали только квадратные и кубические массивы букв. Но криптография не требует, чтобы размеры буквенной сетки были одинаковыми по всем направлениям. Просто так получилось, что в английском алфавите 26 букв, а 26 очень близко к  $5 \times 5$ . Если бы алфавит был русским, то, наверное, мы выбрали бы прямоугольник  $4 \times 8$  или  $5 \times 7$ .

Если бы мы хотели представить все 26 букв алфавита, то предпочтительнее был бы прямоугольник  $3 \times 9$  или  $4 \times 7$ . Этого хватило бы для всех букв и еще одного или двух дополнительных символов. Выше мы уже обсуждали использование таких дополнительных символов, например, для переключения между буквами и цифрами. Большинство шифров на основе квадратов Полибия работают с прямоугольниками  $3 \times 9$  или  $4 \times 7$  ровно так же, как с квадратами  $5 \times 5$ , в предположении, что все прямоугольники ориентированы одинаково. Так обстоит дело с шифром Плейфера, Two Square, Three Square, Four Square и диагональным Bifid.

На самом деле эти пять шифров могут быть даже более стойкими при использовании прямоугольников, потому что для каждой буквы алфавита возможно больше подстановок. Недостатком шифров Плейфера и Two Square в такой конфигурации является более высокая вероятность попадания двух букв в одну строку и, как следствие, замены их буквами, находящимися под ними или справа от них.

Вот пример шифра Плейфера с прямоугольником 3×9:

	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>			
	A	J	S	B	K	T	C	L	U	ME	ET	AT
1										MA	IN	CA
2	X	D	M	V	E	N	W	F	O	MP	US	
3	R	+	G	P	Y	H	Q	Z	I	VN	NK	JC
										XS	HO	JL
										VG	AB	

Открытый текст

Шифртекст

## 9.12 Шестнадцатеричное фракционирование

До сих пор в этой главе речь шла исключительно о ручных методах. А это значит небольшие массивы и алфавит, содержащий только заглавные буквы. При работе на компьютере обычно хочется иметь полный алфавит, с заглавными и строчными буквами, цифры, знаки препинания, специальные символы, диакритические знаки и, быть может, несколько алфавитов. Короче говоря, все возможности компьютера по работе с текстом. Сделать это проще всего, представив каждый символ 8-битовым байтом в одной из стандартных кодировок, например UTF-8 или UTF-16.

Естественный способ фракционировать 8-битовый байт – разбить его на две 4-битовые шестнадцатеричные цифры. Все методы фракционирования, основанные на квадратах Полибия, – шифры Плейфера, Two Square, Three Square, Four Square и Bifid – работают также и для квадратов 16×16. Если такой квадрат хорошо перемешан большим ключом, то эти методы оказываются более стойкими, чем для квадратов 5×5, просто потому, что комбинаций из 256 символов гораздо больше, чем из 5 символов, а именно  $8.58 \times 10^{506}$  против  $1.55 \times 10^{25}$ .

Простой метод использования шестнадцатеричного фракционирования состоит в следующем: (1) преобразовать символы сообщения в шестнадцатеричные цифры, применяя хорошо перемешанную таблицу подстановки, (2) перемешать эти цифры, применив какой-нибудь перестановочный шифр, и затем (3) преобразовать пары шестнадцатеричных цифр обратно в байты, применив вторую хорошо перемешанную таблицу подстановки.

Простейшая перестановка – переместить первую шестнадцатеричную цифру в конец, т. е. 12 34 56 78 преобразуется в 23 45 67 81. Такой шифр можно было бы назвать *Cycle Hex*. По существу, это диагональный Bifid (раздел 9.7), но в системе счисления по основанию 16, а не 5. Cycle hex получает оценку 5. Можно было бы также использовать перестановку кусочного обращения, описанную в разделе

ле 4.6.1, для перемешивания букв. Такой шифр естественно назвать *Piecewise Hex*. Его оценка – тоже 5. Более стойкий метод – перемешать шестнадцатеричные цифры с помощью столбцового перестановочного шифра. Назовем его *Columnar Hex* и поставим оценку 7. В случае двойной столбцовой перестановки оценка увеличивается до 10.

Эти методы годятся для шифрования любого компьютерного файла. Но если файлы чисто текстовые, то их можно усовершенствовать. В чисто текстовом файле обычно задействовано менее 100 из 256 возможных значений байтов. Оставшиеся коды можно использовать для null-символов, биграмм, триграмм и других целей, описанных в разделе 6.4. При правильной реализации это повышает оценку Cycle hex до 6, Piecewise Hex до 6 и Columnar Hex до 8.

## 9.13 Битовое фракционирование

Мы можем также применить фракционирование на уровне отдельных битов, составляющих символы сообщения. Блок N символов представляется 8N битами. Их можно организовать в виде прямоугольника несколькими способами, например:  $2 \times 4N$ ,  $4 \times 2N$ ,  $8 \times N$  и  $N \times 8$ . Так, блок из 5 букв представляется 40 битами, которые можно расположить в 2 строки по 20 бит, в 4 строки по 10 бит, в 8 строк по 5 бит или в 5 строк по 8 бит. При работе вручную это неудобно, но компьютер легко справляется.

В примере ниже показано, как пять символов можно записать по горизонтали в блок  $5 \times 8$ , а затем прочесть по вертикали. Здесь используется стандартная кодировка UTF-8. Например, буква А имеет код 01000001. В качестве открытого текста используется слово DELTA.

Запись	Чтение	Шифртекст	
D 01000100	01000100	00000111	(BELL)
E 01000101	01000101	11000000	À
L 01001100	01001100	00100010	"
T 01010100	01010100	01111000	x
A 01000001	01000001	00001001	(HTAB)

Биты считываются по столбцам сверху вниз. Поскольку каждый столбец содержит всего 5 бит, каждый байт шифртекста занимает два или более столбцов. Восемь бит первого байта шифртекста находятся в столбцах 1 и 2 и выделены светлым оттенком серого. Первый столбец содержит биты 00000, а первые 3 бита второго столбца равны 111, т. е. первый байт шифртекста – 00000111, или 07 в шестнадцатеричной форме. Это управляющий символ BELL (звонок), доставшийся в наследство от эпохи телетайпов, когда он вызывал звонок, сопровождающий возврат каретки. В настоящее время у него нет графического представления. Я буду использовать для его обозначения символ 📞.

Второй байт шифртекста состоит из битов, закрашенных более темным оттенком и расположенных в столбцах 2, 3 и 4. Последние 2 бита во втором столбце 11, третий столбец содержит биты 00000, а первый бит четвертого столбца – 0. Собирая их вместе, находим, что второй байт шифртекста – 11000000. Это код символа **Å** – заглавная буква **A** с диакритическим символом гравис.

Байты 3 и 4 – " и x, двойная кавычка и строчная буква x. Пятый байт расположен в столбцах 7 и 8 и состоит из частей 000 и 01001. Байт 00001001 – код символа **HTAB**, горизонтального табулятора, не имеющего графического начертания. Я буду обозначать его символом ►. Итак, шифртекст имеет вид **Å"x►**.

Несмотря на загадочность результата, метод слабый, потому что для преобразования открытого текста в биты и обратно используется стандартный алфавит. Он получает оценку 1. Если бы для этих шагов использовались два независимых хорошо перемешанных ключом алфавита, то шифр оказался бы просто двоичной версией Bifid с сопряженной матрицей (раздел 9.6.1). Этот метод можно было бы назвать *Hex Rectangle* (шестнадцатеричный прямоугольник). Его оценка такая же, как у Bifid с сопряженной матрицей, т. е. 5.

Естественно организовать из восьми 8-битовых байт квадрат битов 8×8. Запишем 8 бит каждого символа по вертикали в квадрат, используя один перемешанный алфавит, а прочитаем по горизонтали, используя другой перемешанный алфавит. Это просто вариант шифра Hex Rectangle с квадратом 8×8, его оценка 6.

### 9.13.1 Шифр Cyclic 8×N

Стойкость этого шифра несложно повысить. Для любого блока N символов запишем их 8-битовые представления в прямоугольник 8×N по вертикали. Сдвинем каждую строку циклически влево на какое-то число битовых позиций от 0 до N–1. Например, если циклически сдвинуть **abcdefgh** на 2 позиции (правда, байтовых) влево, то получится **cdefghab**. Затем прочитаем 8-битовый столбец по вертикали. В примере ниже используется битовый квадрат 8×8. Каждая строка циклически сдвигается влево на число позиций, указанное слева от нее.

	AMBUSHED	После сдвига	Открытый текст <b>AMBUSHED</b>
2	10101110	10111010	
3	11100100	00100111	
0	01011110	01011110	
5	00001011	01100001	
1	11000010	10000101	
7	00100001	10010000	
2	11011110	01111011	
4	00011101	11010001	
8		8%bŸd†J;	Шифртекст 8%bŸd†J;



Здесь открытый текст RETREAT преобразуется в шифртекст @w«0K\_]. Назовем этот шифр *BitCycle Substitution* (подстановка с циклическим сдвигом битов). Он получает оценку 5. Как и шифр Cyclic 8×N из раздела 9.13.1, его можно применить два, три и больше раз, а размеры блоков варьировать.

Эту основную идею можно эффективно улучшить двумя способами.

Во-первых, байты можно разбивать на части по-разному, например 1, 3, 2, 2 или 2, 4, 2. Скажем, можно было бы сначала зашифровать блок, используя разбиение 3, 2, 3, затем перешифровать с разбиением 1, 3, 2, 2 и, наконец, с разбиением 2, 4, 2. Для этого нужно 7 ключей и 7 шагов. (1) С помощью первой подстановки построить представление сообщения с группами по 3, 2 и 3 бита. (2) Сдвинуть все три строки, применив первый ключ сдвига. (3) С помощью второй подстановки построить представление байтов с группами по 1, 3, 2 и 2 бита. (4) Сдвинуть все четыре строки, применив второй ключ сдвига. (5) С помощью третьей подстановки построить представление байтов с группами по 2, 4 и 2 бита. (6) Сдвинуть все три строки, применив третий ключ сдвига. (7) С помощью четвертой подстановки породить окончательные байты шифртекста.

Во-вторых, сами блоки сообщения можно разбивать по-разному. Пусть, например, мы использовали длинные блоки открытого текста, скажем, по 32 символа. На шаге 2 описанной выше процедуры можно было бы разбить 32 байта на группы по 6, 14 и 12 байт. На шаге 4 можно было бы разбить 32 байта на группы по 11, 8 и 13 байт. На шаге 6 их можно было бы разбить на группы по 8, 17 и 7 байт. Каждая группа сдвигается независимо. Разбиение могло бы быть различным для каждого сообщения.

Можно взять на вооружение и более общий подход. На шаге 2 разбиваем все сообщение на блоки размера X. На шаге 4 разбиваем сообщение на блоки размера Y. На шаге 6 разбиваем сообщение на блоки размера Z. X, Y и Z могут принимать любые значения от 6 до полной длины сообщения.

Я не стану оценивать все варианты подстановочного шифра BitCycle. Скажу лишь, что оценка может варьироваться от 5 до 10. В главе 12 я опишу, как проверить, действительно ли блочный шифр заслуживает оценки 10.

## 9.15 Повышение стойкости блоков

В нескольких шифрах, описанных в этой главе, используются блоки открытого текста. С ними можно проделать несколько трюков, чтобы немного затруднить задачу Эмили. Приведу краткий список идей:

- варьировать длины блоков, периодически или псевдослучайно;
- изменять направление нескольких первых букв блоков на противоположное, периодически или псевдослучайно;

- изменять направление нескольких последних букв блоков на противоположное, периодически или псевдослучайно;
- циклически сдвигать каждый блок влево или вправо, периодически или псевдослучайно;
- менять местами последние N букв блока с первыми N буквами следующего блока.

Но предупреждаю: если вы зашифровываете или расшифровываете вручную, не увлекайтесь этими методами. Шифр, настолько сложный, что невозможно точно зашифровывать или расшифровывать сообщение, становится бесполезным.

# 10

## Фракционирование переменной длины

---

### **Краткое содержание главы:**

- шифры, основанные на коде Морзе;
- перемешанные буквы и биграммы;
- двоичные кодовые слова переменной длины;
- шифры, основанные на сжатии текста.

В этой главе рассматривается большая группа шифров, в которых группы элементов открытого и (или) шифртекста имеют переменную длину. К ним относятся моном-биномный шифр (раздел 10.2), подстановка Хаффмана (раздел 10.4) и таг-системы Поста.

В разделе 4.4 я иллюстрировал идею фракционирования на примере двух вариантов фракционированного кода Морзе, предложенного М. Э. Охавером. Это пример фракционирования переменной длины, потому что используются группы, состоящие из 1, 3 и 4 символов кода Морзе. Более широкое обсуждение фракционирования переменной длины я начну с другой формы фракционированного кода Морзе, которая напоминает трехчастный шифр Trifid, описанный в разделе 9.9. Назову его Morse3.





## 10.2 Моном-биномные шифры

Моном-биномным называется класс шифров, в которых каждая буква заменяется одной или двумя цифрами. Самым знаменитым представителем этого семейства является шифр ВИК, который использовался советскими разведчиками в период примерно с 1920 по 1960 год. Название происходит от клички Виктор, которую ФБР присвоили агенту КГБ Рейно Хейханену. Шифр ВИК не был вскрыт, пока в 1957 году Хейханен не сбежал в США, где и раскрыл все его детали.

Шифр ВИК состоит из двух частей, моном-биномной подстановки и сложения по модулю 10 со случайной последовательностью чисел. Начнем с подстановки. Каждая буква алфавита заменяется одной или двумя десятичными цифрами. Чтобы полномочный получатель Рива могла прочесть сообщение, выбираются какие-то две цифры, которые будут первыми во всех двузначных парах. Предположим, что Сандра, отправитель, выбрала 2 и 5. Все двузначные подстановки будут начинаться с 2 или 5, а все остальные цифры будут однозначными подстановками. Всякий раз, видя, что очередная цифра сообщения равна 2 или 5, получатель понимает, что это начало двузначной последовательности, в противном же случае это однозначная подстановка. Подстановки можно представить таблицей с тремя строками, которая носит странное название – *расширительная шахматная доска* (straddling checkerboard). Название неудачное, потому что размер таблицы не  $8 \times 8$ , она даже не квадратная и не расчерчена на белые и черные поля. Да и не для шахмат и шашек она предназначена. А во всем остальном отличное название. Приведу пример:

	0	1	2	3	4	5	6	7	8	9
-	N	E	■	A	S	■	I	T	R	O
2	F	K	C	J	U	V	*	B	P	Q
5	Z	G	X	W	Y	L	H	#	D	M

Все восемь однозначных подстановок занимают первую строку, а двадцать двузначных, начинающихся с 2 или 5, – вторую и третью. Цифры 2 и 5 нельзя использовать в качестве однозначных подстановок, поэтому в соответствующих им позициях стоят черные прямоугольники. Например, вместо S подставляется 4, вместо U – 24, а вместо Y – 54.

Поскольку имеется 28 позиций и всего 26 букв в английском алфавите, остается два лишних символа, которые я обозначил \* и #. Обычно \* используется как универсальный знак препинания, заменяющий . ? , “ и все остальные знаки, облегчающие чтение сообщения. Символ # служит для переключения между буквами и цифрами. Сообщение 600 TANKS ARRIVE 1800 TODAY следовало бы записать как #600#TANKSARRIVE#1800#TODAY, а в зашифрованном виде – 57600 57730 21438 86251 57180 05779 58354.

Очевидная слабость такой подстановки заключается в том, что больше трети подстановок (в действительности 10 из 28, т. е. 35.7 %) начинается с 2 и столько же с 5, поэтому две выбранные цифры встречаются гораздо чаще остальных восьми. Они будут выделяться как слоны на конкурсе балльных танцев. Чтобы сгладить эту проблему, 8 наиболее частых букв, а именно ETAONIRS, размещаются в верхней строке. Запомнить их поможет мнемоническое слово SERATION, т. е. SERRATION (зубчатость) с одним R. Или можете использовать мою любимую анаграмму, RAT NOISE. Если использовать для самых частых букв однозначные подстановки, то длина шифртекста уменьшится.

Сам по себе шифр расширительной шахматной доски получает оценку 3.

Однако в шифре ВИК имеется еще и второй шаг. (В самом сложном варианте он также переставляет цифры.) Результат моном-биномной подстановки рассматривается как промежуточный шифртекст. К каждой его цифре прибавляется цифра ключа по модулю 10, т. е. без переноса. У этого шага есть две вариации. Можно просто прибавлять повторяющийся числовой ключ, например 2793. Работает это следующим образом:

<b>27932 79327 93279 32793 27932 79327 93279</b>	Повторяющийся ключ <b>2793</b>
<b>57600 57730 21438 86251 57180 05779 58354</b>	Промежуточный шифртекст
<b>74532 26057 14607 18944 74012 74096 41523</b>	Окончательный, ключ + промежуточный

Эта форма шифра ВИК получает оценку 5.

\* В более стойком варианте шифра ВИК используется неповторяющийся числовой ключ, порожденный генератором случайных чисел. Для этой цели русские использовали *генератор Фибоначчи с запаздыванием*. Вы, возможно, знакомы с последовательностью Фибоначчи – последовательностью целых чисел, в которой каждый следующий член равен сумме двух предыдущих. Начальные члены равны  $x_0 = 0$  и  $x_1 = 1$ , а следующие вычисляются по формуле

$$x_n = x_{n-1} + x_{n-2}, n = 2, 3, 4, \dots$$

То есть  $n$ -й член равен сумме  $(n-1)$ -го и  $(n-2)$ -го. Для шифра ВИК существенна только младшая цифра. Это можно записать в виде

$$x_n = (x_{n-1} + x_{n-2}) \bmod 10.$$

Генератор Фибоначчи с запаздыванием обобщает эту формулу в трех направлениях. Во-первых, можно складывать не только два последних числа, т. е.

$$x_n = (x_{n-j} + x_{n-k}) \bmod 10.$$

Заметим, что цепной генератор цифр, описанный в разделе 4.5.1, имеет как раз такую форму с  $j = 1$  и  $k = 7$ .

Во-вторых, числа можно генерировать по разным модулям. Чаще всего модуль равен какой-то степени простого числа,  $p^e$ :

$$x_n = (x_{n-j} + x_{n-k}) \bmod p^e.$$

В-третьих, к двум членам можно применять не только операцию сложения. Иногда используют вычитание, умножение и ИСКЛЮЧАЮЩЕЕ ИЛИ. Это можно записать в виде

$$x_n = (x_{n-j} \bullet x_{n-k}) \bmod p^e,$$

где  $\bullet$  может представлять  $+$   $-$   $\times$   $\oplus$  или еще какой-то бинарный оператор. (Деление тоже допустимо. Это то же самое, что умножение на элемент, обратный второму операнду. См. раздел 3.6.) На практике сложение используется чаще всего, потому что аддитивные генераторы порождают последовательности с самыми длинными периодами.

При использовании такой формы генератора псевдослучайных чисел моном-биномный шифр получает оценку 7. \*\*

## 10.3 Периодические длины

Простой способ реализовать шифрование с блоками переменной длины – использовать несколько таблиц подстановки, по одной для каждой желаемой длины блока. Если бы это были блоки букв, то таблицы подстановки очень быстро достигли бы огромного размера. Поэтому будем использовать биты. Пусть сообщение представлено строкой битов. Чтобы зашифровать сообщение, мы разбиваем его на короткие блоки битов и подставляем блок того же размера, пользуясь таблицей подстановки для этой длины. Длины блоков могут изменяться периодически в соответствии с повторяющимся числовым ключом либо порождаться генератором случайных чисел.

Продемонстрирую на небольшом примере. Имеются три таблицы подстановки для блоков из 2, 3 и 4 бит. В реальном шифре я использовал бы блоки по 3, 4, 5 и 6 бит, но можно увеличить длину до 16 и даже больше, если хватает памяти. Для этой простой демонстрации я взял стандартный алфавит, дополнив его до 32 символов знаками в верхнем ряду клавиатуры слева направо.

A 00000	I 01000	Q 10000	Y 11000
B 00001	J 01001	R 10001	Z 11001
C 00010	K 01010	S 10010	@ 11010
D 00011	L 01011	T 10011	# 11011
E 00100	M 01100	U 10100	\$ 11100

F 00101	N 01101	V 10101	% 11101
G 00110	O 01110	W 10110	& 11110
H 00111	P 01111	X 10111	* 11111

Таблицы подстановки имеют вид:

2 бита	3 бита	4 бита
00 11	000 101	0000 1110 1000 1111
01 00	001 010	0001 0100 1001 1001
10 10	010 111	0010 1101 1010 0010
11 01	011 000	0011 0101 1011 1010
	100 011	0100 0000 1100 0001
	101 100	0101 0110 1101 1011
	110 001	0110 0111 1110 0011
	111 110	0111 1000 1111 1100

Вот пример шифрования с повторяющимся ключом 3, 2, 2, 4, 2:

M	O	S	C	O	W	Открытый текст
01100	01110	10010	00010	01110	10110	Открытый текст в двоичном виде
3	2	2	4	2	3	Ключ
011	00	01	1101	00	100	Двоичный текст, разбитый на группы ключом
000	11	00	1011	11	011	После подстановки
00011	00101	11101	11110	10000	00000	Блоки по 5 бит
D	F	%	&	Q	A	Шифртекст DF%&QA

Этот вариант, который я назову *BitBlock SA*, где SA означает Standard Alphabet (стандартный алфавит), имеет умеренную стойкость. Один ключ используется для перемешивания всех таблиц подстановки, а другой определяет последовательность размеров блоков. Когда таблиц подстановки немного, все размеры блоков малы и последовательность их размеров тоже мала, шифр BitBlock SA получает оценку 3. В противном случае оценка повышается до 4.

Чтобы укрепить шифр, можно, например, использовать хорошо перемешанные ключом алфавиты для преобразования букв в биты и обратного преобразования битов в буквы. Назовем такую версию с перемешанным алфавитом *BitBlock MA*. Она получает оценку 7.

## 10.4 Подстановка Хаффмана

В разделе 4.2 было описано, как использовать коды Хаффмана для сжатия текста. *Подстановка Хаффмана* – способ применения кодов Хаффмана к шифрованию. В этом шифре используется два набора кодов. Коды из второго набора подставляются вместо кодов из первого. Наборы могут совпадать по составу, но отличаться порядком.

Сообщение представлено строкой битов, например, в одной из стандартных компьютерных кодировок, таких как UTF-8. Эта строка разбивается на строку кодов из первого набора кодов Хаффмана, затем эти коды заменяются кодами из второго набора. Подстановка Хаффмана не сжимает сообщение, хотя его длина в битах может измениться, если длины кодов из первого набора отличаются от длин замещающих их кодов из второго набора.

Напомним, что набор кодов Хаффмана должен обладать префиксным свойством. То есть ни один код не может начинаться другим кодом из того же набора. Так, в одном наборе не может быть одновременно кодов **1101** и **11011**, потому что если декодируемая строка начинается битами **11011**, то невозможно сказать, состоит ли первый код из четырех или пяти бит. При использовании префиксных кодов не нужен разделитель между соседними кодами, как в случае кодовых групп Морзе.

Поговорим о том, как построить набор кодов Хаффмана, обладающий префиксным свойством. Начнем с того, что запишем одиночные биты в произвольном порядке: 0, 1 или 1, 0. Например:

1  
0

Для каждого элемента в этом списке либо принимаем его в качестве окончательного кода, либо продолжаем, чтобы получить два более длинных кода, для чего дописываем 0 в конец одной копии и 1 в конец другой – в любом порядке. Например, мы могли бы считать код 1 окончательным, а код 0 продолжить до 00 и 01; в результате получился бы такой набор:

1  
00  
01

Этот процесс можно повторять сколько угодно раз. Например, можно было бы считать код 01 окончательным, а код 00 продолжить еще на один шаг, получив 000 и 001:

1  
001  
000  
01

Так можно продолжать, пока не наберется достаточное количество кодов или не будет исчерпан заданный диапазон длин кодов. Но для нашего примера четырех кодов вполне хватит. Примем коды 000 и 001 за окончательные, так что получится полный набор из четырех кодов.

\* Можно оценить среднюю длину строки битов, зашифрованной с помощью этих кодов. С вероятностью 1/2 строка начинается с 1,

поэтому с вероятностью  $1/2$  длина кода будет равна 1. С вероятностью  $1/8$  строка начинается с 000 и с такой же вероятностью с 001. В любом случае длина кода будет равна 3. С вероятностью  $1/4$  строка начинается с 01 и длина кода будет равна 2. Поскольку это полный набор кодов, других возможностей нет. Суммируя, получаем  $1/2 + 3/8 + 3/8 + 2/4 = 14/8 = 1.75$  бита.

После подстановки первого кода вероятности последующих будут такими же, поэтому ожидаемая длина всех кодов равна 1.75 бита. Это меньше, чем средняя длина кодов, равная 2.25 бита. \*\*

Ниже приведен пример подстановки Хаффмана. Имеется два набора кодов. Коды из левого столбца заменяются кодами из правого. Оба набора обладают префиксным свойством. Открытый текст LIBERTY представлен стандартным 5-битовым кодом: A= 00000, B = 000001, C = 00010, ..., Z = 11001.

												<u>Набор1</u>	<u>Набор2</u>	
												000	011	
												001	0100	
												01	00	
												100	0101	
												101	11	
												11	10	
L	I	B	E	R	T	Y						Открытый текст		
01011	01000	00001	00100	10001	10011	11000						Битовая строка		
01	01	101	000	000	01	001	001	000	11	001	11	100	01	Коды Хаффмана
00	00	11	011	011	00	0100	0100	011	10	0100	10	0101	00	Подстановки
00001	10110	11000	10001	00011	10010	01001	0100							Группы по 5
B	W	Y	R	D	S	J	I							Шифртекст

Первая строка 5-битовых групп – слово LIBERTY, закодированное стандартным способом: A= 00000, B = 00001 и т. д. Вторая строка битов совпадает с первой, но разбита на коды Хаффмана из набора 1. Подчеркнутая цифра 1 – дополнение, необходимое для того, чтобы последний код Хаффмана был корректным. Третья строка битов – результат замены кодов из набора 1 соответствующими кодами из набора 2, т. е. результат шага подстановки. Четвертая строка битов совпадает с третьей, но разбита на группы по 5 бит. Отметим, что строка 4 на четыре бита длиннее строки 1. И последняя строка содержит шифртекст в том же стандартном 5-битовом представлении алфавита. Последней буквой шифртекста могла бы быть I или J, поскольку в последней группе только 4 бита.

\* Если вы проделываете эти операции на компьютере, то нет нужды сравнивать начало битовой строки с каждым кодом Хаффмана по очереди. Предположим, что самый длинный код состоит из 6 бит. Можно составить таблицу всех 64 возможных 6-битовых комбинаций. В каждом элементе таблицы хранится длина кода в битах и его

замена. При выполнении подстановки мы берем первые 6 бит строки и используем их как индекс таблицы. Предположим, к примеру, что первый код Хаффмана равен 00000, а вместо него подставляется 0110. Возможные значения первых 6 бит строки, начинающейся этим кодом, – 000000 и 000001. Поэтому в обоих элементах таблицы 000000 и 000001 должна храниться длина кода 5 и замена 0110. Чтобы выполнить подстановку, следует удалить первые 5 бит строки и добавить 0110 в конец результирующей строки. \*\*

## 10.5 Таг-системы Поста

Математик Эмиль Леон Пост из Института Куранта при Нью-Йоркском университете изобрел *таг-системы* в 1920 году. Основная идея очень проста. Начинаем со строки битов. Затем берем несколько битов из начала строки, заменяем их другими битами и помещаем в конец строки. Продолжаем этот процесс. В результате может произойти одно из трех: либо строка уменьшится так, что следующая операция станет невозможной, либо мы войдем в бесконечный цикл, либо строка будет расти неограниченно.

### Историческое отступление

Пост создавал свои таги, не имея в виду применение в криптографии. Пост доказал, что на вопрос о том, уменьшается строка, растет или повторяется, нельзя дать ответ, оставаясь в рамках стандартной математики. А затем он использовал этот факт для доказательства знаменитых теорем Курта Гёделя о неполноте. Мне кажется, что это доказательство проще и элегантнее, чем доказательство Алана Тьюринга, в котором используются символы, записываемые на бесконечную ленту, хотя между битовой строкой Поста и лентой Тьюринга имеется разительное сходство.

Эта *подстановка Поста* похожа на подстановку Хаффмана, только замененный символ перемещается в конец битовой строки. Преимущество такой системы в том, что процедуру можно продолжать и после того, как вся строка заменена. То есть мы можем совершать по строке несколько проходов. Это стирает границы между кодами Хаффмана.

Части, которые берутся из начала строки, называются *тагами*. Множество тагов следует выбирать так, чтобы на каждом шаге можно было взять не более одного тага. То есть процесс замены *детерминирован*. Для этого необходимо, чтобы множество тагов обладало префиксным свойством. Это позволит использовать его для шифрования сообщения, представленного битовой строкой. Префиксное свойство обсуждается в разделе 4.2.1 в контексте кодов Хаффмана. В двух словах, никакой таг не может начинаться другим тагом. На-



пример, не может быть одновременно тагов 1101 и 11011, потому что если строка начинается с 11011, то невозможно сказать, надо ли брать первые 2 или первые 5 бит. Если префиксное свойство имеет место, то между тагами не нужно вставлять разделителей, как между кодовыми группами Морзе. По форме коды Хаффмана и таги Поста одинаковы, но используются по-разному. Начать с того, что коды Хаффмана применяются для сжатия битовой строки, а таги Поста – нет.

Метод построения кодов Хаффмана описан в разделе 10.4.

При шифровании с помощью тагов Поста каждый таг заменяется другим тагом, и новый таг перемещается в конец строки. Поскольку Рива будет дешифровать сообщение справа налево, заменяющие таги должны обладать *суффиксным свойством*, противоположным префиксному: никакой суффиксный таг не должен заканчиваться другим суффиксным тагом. Например, если 1011 – суффиксный таг, то ни 01011, ни 11011 таковым быть не могут.

Построить набор суффиксных тагов можно так же, как набор префиксных, только продолжать таг следует влево, а не вправо. Если вам это неудобно и непривычно, то можете просто построить второй набор префиксных тагов, а затем поменять в них порядок битов на противоположный – получатся суффиксные таги. Суффиксных тагов должно быть не меньше, чем префиксных, но может быть и больше. Дополнительные таги можно использовать как омофоны. Например, префиксный таг 0111 можно было бы заменить одним из суффиксных тагов 110 или 10101 на выбор.

Если ожидаемая длина суффиксов меньше ожидаемой длины префиксов, то строка, вероятно, будет сжата. То есть с большой вероятностью существуют начальные строки, которые станут короче. Наоборот, если суффиксы длиннее префиксов, то с большой вероятностью некоторые начальные строки вырастут. Обычно это так, если используются омофоны. Чем больше разность между ожидаемыми длинами, тем сильнее будут расти или укорачиваться начальные строки. Однако вероятность – это не гарантия. Можно построить наборы префиксных и суффиксных тагов, демонстрирующие противоположное поведение.

Чтобы воспользоваться тагами Поста для шифрования, мы сначала должны представить сообщение в виде строки битов, а затем несколько раз выполнить подстановку тагов. При шифровании ручную биты затем преобразуются обратно в символы. При шифровании на компьютере последний шаг необязателен, достаточно просто передать результирующую битовую строку.

### 10.5.1 Таги одинаковой длины

У описанного в предыдущем разделе шифра есть недостаток – Рива не знает, как разбить полученное сообщение на блоки. Возможно, понадобится отдельное поле длины для каждого блока. Или же мож-

но рассматривать все сообщение как один блок. Для длинных сообщений это неудобно. Одно из решений – заменять каждый префиксный тэг суффиксным той же длины. Тогда в процессе шифрования длина блоков останется неизменной, и проблемы с разграничением блоков не возникнет. Чаще всего выбирают блоки длиной 32 или 64 бита.

Я рекомендую выполнять фиксированное число подстановок для каждого блока. Определить подходящее число можно по наименьшей и ожидаемой длине тага. Предположим, что длина блоков составляет 32 бита, длина самого короткого тага – 3 бита, а математическое ожидание длины тагов равно 4.3 бита. Если брать наименьшую длину, то, выполнив не менее  $32/3 = 10.67$  подстановки, мы гарантируем, что каждый бит в блоке заменен по меньшей мере один раз. Округлим до 11. Если брать ожидаемую длину, то в среднем для замены каждого бита нужно  $32/4.3 = 7.44$  подстановки.

Хороший запас безопасности получается, если в среднем каждый бит заменен дважды. Удвоим 7.44 и после округления получим 15 шагов подстановки. Это больше 11, так что хотя бы один раз каждый бит будет заменен гарантированно, а в среднем каждый бит будет заменен дважды. Примерно в половине случаев некоторые биты будут заменены трижды. Но самое главное – Эмили не будет знать, сколько раз заменен каждый конкретный бит.

Вы, наверное, обратили внимание, что я говорю «каждый *бит* будет заменен», а не «каждый *таг* будет заменен». Разберемся в этом нюансе. При первом проходе по блоку каждый тэг заменяется новым тагом той же длины. Поэтому в первом раунде действительно заменяются таги. Но второй раунд подстановки может начинаться не на границе тага. То есть следующий тэг может перекрывать два или более тагов, оставшихся после первого раунда.

Мини-пример ниже иллюстрирует этот момент для 12-битового блока. Первый бит блока выделен, а префиксные таги подчеркнуты.

<u>Set1</u>	<u>Set2</u>	<u>101101110100</u>	Оригинальный 12-битовый блок, тэг 10
00	01	<u>1101110100</u> 111	Заменить 10 на 11, следующий тэг 110
010	110	<u>1110100</u> 111010	Заменить 110 на 010, следующий тэг 111
011	000	<u>0100</u> 11010100	Заменить 111 на 100, следующий тэг 010
10	11	<u>011010100</u> 110	Заменить 010 на 110, следующий тэг 011
110	010	Начинается второй раунд	
111	100		

После четырех подстановок первый бит оказался во второй позиции, прямо в середине следующего префиксного тага 011.

При ручном применении я рекомендую кодировать буквы алфавита пяти- или шестибитовыми группами, использовать от 20 до 30 пар тагов длиной от 3 до 6 бит, 32-битовые блоки и 16 шагов подстановки, т. е. проходить по блоку примерно два раза. Для обратного преобразования результирующих битов в строки используйте 4-би-

товые группы для представления букв от А до Р в каком-то перемешанном порядке. Такой шифр получил бы оценку 6.

Для компьютерного применения я рекомендую стандартную 8-битовую кодировку, например UTF-8, для представления букв, цифр и специальных символов в сообщении. Используйте от 40 до 80 пар тагов длиной от 4 до 8 бит, 64-битовые блоки и 32 шага подстановки. 32 шагов достаточно для трех проходов по блоку. Прежде чем выполнять подстановки тагов Поста, хорошо перемешайте символы сообщения ключом, а для подстановки результирующих байтов после завершения подстановок тагов используйте второй независимый ключ. Такой шифр, называемый *Post64*, получил бы оценку 10. В нем используется 4 разных ключа для перемешивания начальной подстановки, конечной подстановки, тагов Поста и заменяющих их тагов.

Еще один способ применения подстановки тагов Поста – использовать короткие перекрывающиеся блоки. Начнем с первых 4 байт сообщения и выполним две подстановки Поста. В предположении, что длина тагов от 4 до 8 бит, этого достаточно, чтобы все биты первого байта были гарантированно заменены. Затем сдвинемся на один байт вправо. Следующий 4-байтовый блок сообщения занимает байты 2, 3, 4 и 5. Снова выполним две подстановки Поста для этого блока. Продолжаем, пока не дойдем до последнего 4-байтового блока сообщения. Последние три блока заворачиваются в начало сообщения. Этот метод, *PostOv*, получает оценку 6.

### 10.5.2 Таги разной длины

Когда для разных тагов применяются подстановки разной длины, возникает целый ряд сложностей: длина блока может измениться и блоки необязательно заканчиваются на границах байтов. Например, 32-битовый блок может стать 35-битовым. Это значит, что Риве необходимы средства для разделения блоков. Проще всего передавать длину каждого блока.

Может показаться, что достаточно просто применять к блоку подстановки тагов Поста, пока его длина снова не станет кратной 8. Увы, для этого могут потребоваться тысячи и даже миллионы шагов подстановки. Возможно даже, что этого никогда не случится.

Самое простое решение – зашифровать все сообщение как один блок. Сама длина сообщения скажет Риве, сколько в блоке байтов. Сандре нужно только добавить 3-битовое поле, чтобы Рива знала, сколько битов занято в последнем байте (от 1 до 8). Это поле можно поместить в начало сообщения или в 3 последних бита последнего байта. Для поля длины может понадобиться дополнительный байт.

Ниже приведен пример шифрования с применением тагов Поста переменной длины. Префиксные таги и заменяющие их суффиксные таги подчеркнуты одинаково.

Префикс	Суффикс		Биты открытого текста
00	101	10110100001000111001	
010	100	010000100011100111	
011	1010	000100011100111100	
1000	000	0100011100111100101	
1001	001	00111001111001011100	
1010	110	11100111100101100101	
1011	11	1001111001011001010010	
11	0010	111001011001010010001	

\* Может показаться, что нужно сдвигать все сообщение всякий раз, как тэг удаляется из начала. Но эти сдвиги можно устранить, если хранить указатели на первый и последний биты сообщения. Указателем в данном случае является целое число, описывающее положение бита в сообщении. Младшие 3 бита указателя определяют позицию внутри байта, а старшие биты – позицию самого байта. Выделите область памяти в 4 раза больше длины сообщения. Поместите сообщение в начало этой области, а остальные байты обнулите.

Чтобы удалить тэг из начала строки, нужно просто прибавить к указателю на начало длину префиксного тага. Чтобы дописать тэг в конец строки, нужно поместить его в нужные биты машинной командой сдвига, затем применить команду OR к нему и последним двум байтам строки, после чего увеличить указатель на конец. Эта процедура продолжается, пока не будет достигнут конец выделенной области. Следовательно, количество шагов подстановки Поста зависит от самого сообщения.

И еще нужен один сдвиг в самом конце, чтобы выровнять битовую строку с границей байта. Однако и этого длинного сдвига можно избежать, сообщив корреспонденту позиции начального и конечного битов в первом и последнем байтах сообщения соответственно. Для этого необходимо всего 6 бит, которые можно упаковать в один байт и поместить в начало сообщения. Я рекомендую шифровать этот байт простой подстановкой, что не выдать Эмили начальную и конечную позиции. Кроме того, обязательно заполняйте неиспользованные части первого и последнего байтов сообщения случайными битами.

Остается один вопрос: поскольку Рива не знает длины оригинального сообщения и, следовательно, размера области шифрования, как ей узнать, когда прекратить дешифрирование? Рива не в курсе, сколько было выполнено шагов подстановки, и не может просто выделить память, в 4 раза большую длины полученного сообщения, потому что его длина необязательно совпадает с длиной отправленного сообщения.

Решается проблема следующим образом. Рива знает три вещи: открытое сообщение начиналось на границе байта, сообщение заканчивалось на границе байта, а область шифрования была в 4 раза больше длины исходного сообщения. В самом начале Рива может

поместить полученное сообщение в конец области, в 5 раз превышающей по длине шифртекст. Этого должно быть более чем достаточно. Затем Рива двигается от конца к началу, пока не будут выполнены все три условия, в частности пока расстояние от начала частично дешифрованного сообщения до конца области дешифрования не станет в точности равно длине сообщения, умноженной на 4. Это может произойти только один раз. \*\*

Я рекомендую использовать от 50 до 80 пар тагов длиной от 4 до 8 бит. Ожидаемая длина исходных тагов должна быть близка к ожидаемой длине заменяющих. Примерно  $1/3$  заменяющих тагов должна быть короче исходных,  $1/3$  – такой же длины и  $1/3$  – длиннее. Необязательно, чтобы длина каждого заменяющего тага отличалась от длины исходного. Символы сообщения должны быть представлены 8-битовыми байтами, взятыми из хорошо перемешанного алфавита. Если ожидаемая длина тагов равна  $T$  бит, а длина сообщения  $L$  бит, то необходимо выполнить по меньшей мере  $3L/T$  шагов подстановки. То есть сделать 3 или более проходов по сообщению. Конечную битовую строку, включая индикатор длины, следует преобразовать обратно в символы, используя простую подстановку со вторым, независимым от первого ключом. При выполнении всех рекомендаций этот шифр, назовем его *PostDL*, получает оценку 10.

Дочитав до раздела 12.6, вы узнаете, что шифр *PostDL* не отвечает всем критериям невскрываемости шифра. Оценку 10 он получил, потому что Эмили не знает, где в шифртексте окажется любой наперед заданный бит открытого текста. Позиции будут меняться от блока к блоку. Поэтому Эмили не может установить соответствие между битами открытого и шифртекста, а значит, не может выписать уравнения, связывающие биты шифртекста с битами открытого текста и ключа.

### 10.5.3 Несколько алфавитов

Укрепить шифр на основе тагов Поста или кодов Хаффмана можно несколькими способами. Мы уже рассмотрели выполнение нескольких раундов подстановки. Другой прием – использовать несколько алфавитов. Каждый алфавит будет состоять из набора тагов, обладающего префиксным свойством, и соответствующего набора заменяющих тагов, обладающего суффиксным свойством. Можно просто использовать несколько алфавитов по очереди или выбирать их в соответствии с ключевым словом. Если делать это вручную, то, скорее всего, вы ограничитесь двумя, максимум тремя, алфавитами, поэтому я рекомендую использовать числовой ключ, например 01101011.

Эти шифры, которые можно назвать *PolyPost* и *PolyHuff*, получают оценку от 4 до 8 в зависимости от числа раундов, числа алфавитов и длины ключа.

### 10.5.4 Короткие и длинные перемещения

До сих пор мы предполагали, что все В бит тага Поста перемещаются в конец блока. Однако можно перемещать как меньше, так и больше В бит. Например, можно было бы перемещать В–1 бит, а один оставшийся включать в следующий заменяемый таг. В таком случае таги перекрываются. Преимущество в том, что при этом мы скрываем границы тагов. А недостаток – один раунд будет содержать больше шагов подстановки, что замедляет шифрование.

С другой стороны, можно было бы перемещать в конец блока В+1 бит. При этом один бит остается неизменным, и он всегда является последним битом блока. Это не особенно серьезная проблема, если раундов несколько, так что бит, не изменившийся в одном раунде, скорее всего, будет заменен в другом. Но все же остается шанс, что некоторые биты не будут изменены в процессе шифрования. С этой слабостью можно смириться, если Эмили не может определить, какие биты не изменились. Биты анонимны. Ничто в бите не говорит: «Этот бит занимал позицию 2 в байте 5 открытого текста».

Наконец, число перемещенных битов можно сделать независимым от длины тага. Можно завести таблицу, которая говорит, сколько битов переместить. Оно может быть меньше, больше или равно длине тега. Таких таблиц может быть несколько.

Когда число перемещаемых битов отличается от длины тага, суффиксное свойство теряет смысл для набора заменяющих тагов. Им должен обладать набор фактически перемещаемых битовых строк. Например, если таг 0110 заменяется битами 1101, но перемещается 5 бит, то набор суффиксных строк должен включать и 11010, и 11011.

## 10.6 Фракционирование в системах счисления по другим основаниям

До сих пор в этой главе мы обсуждали моном-биномные шифры в пятеричной системе и шифры Хаффмана и Поста в двоичной системе. Подстановки переменной длины возможны и при других основаниях. Для ручного шифрования подстановки Хаффмана и Поста проще делать по основанию 3 или 4, а не в двоичной системе. Но ничто не препятствует и выбору таких эксцентричных оснований, как 11 или 13. Это дает дополнительные подстановки, которые можно использовать в качестве омофонов или для кодирования биграмм.

При работе в системе по основанию 13 для подстановки можно использовать 13 из 16 шестнадцатеричных цифр, а остальные три цифры зарезервировать в качестве null-символов. Если сделать все

правильно, так чтобы у всех 16 цифр были приблизительно одинаковые частоты и распределение, то Эмили не сможет отличить настоящие цифры от null-символов.

## 10.7 Сжатие текста

В разделе 4.2.1 обсуждалось использование кодов Хаффмана для сжатия текста. На основе сжатия можно предложить несколько стойких схем шифрования. В этом разделе я представляю некоторые из наиболее передовых схем сжатия текста и схем шифрования на базе кодов Хаффмана. Оставшаяся часть главы 10 факультативна. Если в какой-то момент вы устанете от математики, можно сразу перейти к следующей главе.



### 10.7.1 Метод Лемпеля–Зива

Схему сжатия текста *Лемпеля–Зива* разработали израильские ученые Абрахам Лемпель и Якоб Зив в 1977 году и назвали *LZ77*. Усовершенствованная версия, опубликованная в 1978 году, называется *LZ78*. Идея та же, что при кодировании Хаффмана, – буквы и их комбинации представляются двоичными кодами, т. е. группами битов. Однако подход Лемпеля–Зива к реализации этой идеи противоположен. В схеме Хаффмана используются более короткие коды для экономии памяти. В схеме Лемпеля–Зива длины кодов примерно одинаковы, но для экономии памяти некоторые коды представляют более длинные комбинации букв.

Схемы Хаффмана и Лемпеля–Зива противоположны и еще в одном смысле. В схеме Хаффмана длины кодов определяются на основе заранее заданной таблицы частот букв. В схеме же Лемпеля–Зива самые частые комбинации букв определяются динамически в процессе кодирования текста. Такой подход называется *адаптивным кодированием*. Кодирование Хаффмана хорошо подходит только для текстов на одном языке. В другом языке частоты букв отличаются. Даже при переходе от текста, набранного только заглавными буквами, к тексту, набранному буквами обоих регистров, требуется другой набор кодов Хаффмана. Напротив, метод Лемпеля–Зива применим к компьютерным файлам любого типа, к тексту на любом языке или на смеси языков, к компьютерному коду, изображениям, телеметрии, музыке, видео и т. д.



Существует несколько вариантов метода Лемпеля–Зива. Представленный ниже называется алгоритмом *Лемпеля–Зива–Уэлча*, или *LZW*; он был разработан Терри Уэлчем из компании Sperry Research в 1984 году. У LZW имеются версии с фиксированной и с переменной шириной, я опишу только последнюю, потому что ее проще адаптировать для применения в криптографии.

Во всех вариантах метода Лемпеля–Зива используется список букв и их комбинаций, называемый *словарем*. Словарь строится динамически, по мере того как алгоритм проходит по файлу. В версиях LZ77 и LZ78 словарь изначально пуст. Кодом любой комбинации букв является ее местоположение в словаре.

Вначале LZW назначает код каждому одиночному символу, встречающемуся в файле. У всех кодов LZW число битов одинаково. Например, если файл содержит сообщение на английском языке, записанное только заглавными буквами без знаков препинания и деления на слова, то потребуется 26 кодов, поэтому можно было бы ограничиться 5-битовыми кодами. Но чаще начинают с 256 кодов, по одному для каждого из возможных значений 8-битового байта.

По мере прохождения по файлу алгоритм ищет комбинации букв, пока еще отсутствующие в словаре. Найденные комбинации добавляются в словарь. Предположим, к примеру, что алгоритм нашел в файле строку ТНЕ, которая уже есть в словаре. И пусть следующая буква М, а строки ТНЕМ в словаре нет. Тогда он формирует код комбинации ТНЕ+М и добавляет строку ТНЕМ в словарь. Кодом ТНЕМ будет номер следующей доступной позиции в словаре, скажем 248.

Поскольку ТНЕ уже была в словаре, алгоритм не рассматривает комбинации, начинающиеся с НЕ или Е. Поиск следующей отсутствующей в словаре комбинации начинается с буквы М. Если это комбинация MOR, то MOR помещается в позицию словаря 249 и получает код 249. В следующий раз, когда алгоритм встретит в файле комбинацию ТНЕМ, она уже будет иметь код 248, а следующее вхождение MOR – код 249.

После того как все 256 записей словаря, отведенных для 8-битовых кодов, заполнены, следующий код должен состоять из 9 бит. С этого момента алгоритм переключается с 8-битовых кодов на 9-битовые. Строка ТНЕМ по-прежнему будет иметь код 248, но это будет 9-битовый код 011111000, а не 8-битовый 11111000. Когда окажутся заполнены все 512 записей словаря, отведенных под 9-битовые коды, код ТНЕМ станет 10-битовым, 0011111000, но по-прежнему равным 248. Обратите внимание на порядок операций. Код текущей комбинации букв сначала выводится в старом размере, затем в словарь добавляется новая комбинация и размер кода увеличивается. И Сандра, и Рива должны использовать один и тот же порядок, иначе сообщения будут распакованы неправильно. Увеличение размера кодов обычно прекращается на 12 битах. Последующее увеличение с 12 до 13, как правило, не улучшает степень сжатия, а иногда даже ухудшает.



Рассмотрим пример. Закодируем этим методом слово ТЕТЕ-А-ТЕТЕ. Предположим, что в начальный момент словарь содержал одиночные буквы А, Е и Т с 2-битовыми кодами. Посмотрим, как он строится дальше. На каждом шаге битовые строки слева показывают закодированное слово, в буквы справа – оставшуюся часть слова.

		<u>Код</u>	<u>Словарь</u>	<u>Остаток</u>	
		00	А		
		01	Е		
		10	Т	<u>ТЕТЕАТЕТЕ</u>	Исходный текст
		10	11	<u>ЕТЕАТЕТЕ</u>	Добавить <b>ТЕ</b> в словарь
		10 01	100	<u>ТЕАТЕТЕ</u>	Переключиться на 3-битовые коды
		10 01 011	101	<u>ТЕА</u>	Добавить <b>ТЕА</b> в словарь
		10 01 011 000	110	<u>АТЕ</u>	Добавить <b>АТ</b> в словарь
		10 01 011 000 011	111	<u>ТЕ</u>	Добавить <b>ТЕТ</b> в словарь
10 01 011 000 011 011		---	---		Конец, больше нечего добавлять

В процессе распаковки словарь должен строиться точно так же. Отметим, что одной лишь битовой строки **10 01 011 000 011 011** недостаточно для распаковки сообщения. Рива должна еще знать, что коды **00**, **01** и **10** представляют символы А, Е и Т.

Ну что ж, с алгоритмом сжатия Лемпеля–Зива мы познакомились. Но это книга о криптографии. Как сжатие Лемпеля–Зива можно применить в криптографии?

При построении словаря алгоритм Лемпеля–Зива назначает коды последовательно. 43-я буква или комбинация букв получит код 42 (не 43, потому что нумерация записей начинается с нуля). Чтобы воспользоваться этой схемой для шифрования, добавим в словарь еще один столбец. Первый столбец содержит комбинации букв, а второй – соответствующие коды. В качестве кода комбинации мы будем использовать не позицию в словаре, а число во втором столбце.

Допустим, что в начальный момент словарь содержит 256 однобайтовых символов. Первый столбец содержит сами символы. Во второй столбец поместим числа от 0 до 255 в каком-то перемешанном порядке. Для перемешивания можно использовать методы, описанные в разделе 5.2. Сандра и Рива должны использовать одинаковый порядок, который можно задать ключевым словом или с помощью начального значения для генератора случайных чисел. Когда понадобится первый 9-битовый код, в следующие 256 записей словаря будут записаны коды от 256 до 511 тоже в перемешанном порядке. Аналогично при переходе от 9-битовых кодов к 10-битовым будут выделены следующие 512 кодов. Выделение кодов пакетом, а не по одному эффективнее.

Альтернатива пакетному выделению кодов – выделять с помощью ключевого слова или генератора случайных чисел только первые 256 кодов, а затем вычислять каждый новый код, прибавляя 256 к коду, находящемуся в записи с номером на 256 меньше. То есть  $X(N) = X(N - 256) + 256$ .

Этот шифр, который я назову *Lempel-Ziv Substitution* (подстановка Лемпеля–Зива), получает оценку 3. Оценка такая низкая, потому что первые несколько символов сообщения по существу зашифрованы простой подстановкой. Каждый код будет представлять один символ, пока не встретится первая повторяющаяся биграмма. Возможно, это произойдет, когда закодировано уже 30, 40 или больше символов. И даже после этого большинство 9-битовых кодов будут представлять одиночные буквы. Эти коды легко отличить, потому что только они начинаются с 0. У Эмили будет масса возможностей использовать частоты букв и контактов для прочтения сообщения.

Чтобы укрепить подстановку Лемпеля–Зива, мы можем добавить второй шаг подстановки. Эта подстановка не должна выполняться по границам байтов. Я рекомендую использовать 7-битовые группы, которые не будут совпадать с кодовыми группами, пока длина кода не достигнет 14 байт. Но такого никогда не будет, потому что длина кодов обычно ограничивается 12 битами. Подстановка Лемпеля–Зива, за которой следует 7-битовая подстановка, получает оценку 6. Обе подстановки можно выполнить на одном проходе слева направо.

## 10.7.2 Арифметическое кодирование

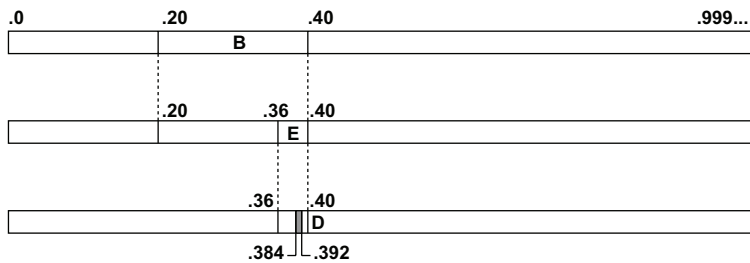
*Арифметическое кодирование* – это метод сжатия текста, который я придумал в 1970-х годах («Arithmetic Stream Coding Using Fixed Precision Registers», *IEEE Trans. on Info. Theory* vol. 25 (Nov. 1979), pp. 672–675). Он основан на удачной идее Питера Элиаса из MIT.

Идея Элиаса состояла в том, чтобы кодировать каждый символ дробью. Представьте все возможные дроби от 0.0 до 0.999.... Здесь многоточие означает бесконечное количество девяток. Теперь разобьем этот диапазон на части в соответствии с первым символом строки. Для простоты будем предполагать, что алфавит содержит 25 символов, как в квадрате Полибия. Каждой букве отведем  $1/25$  полного диапазона. Строки, начинающиеся с А, получают первую  $1/25$  часть, или 4 %, а именно диапазон от 0.0 до 0.04. Строки, начинающиеся с В, получают следующую  $1/25$  часть – диапазон от 0.04 до 0.08. Строки, начинающиеся с Z, получают последний диапазон от 0.96 до 0.999.... (Я привел этот пример в десятичной нотации, чтобы было проще читать. В компьютерной программе использовались бы двоичные дроби.)

Для второго символа снова разобьем диапазон. Строки, начинающиеся с АА, попадут в диапазон от 0.0 до 0.0016, начинающиеся с АВ – в диапазон от 0.0016 до 0.0032, начинающиеся с ВА – в диапазон от 0.0400 до 0.0416. И так далее. Строки, начинающиеся с ZZ, попадут в диапазон от 0.9984 до 0.9999....

Для наглядности воспользуемся миниатюрным 5-буквенным алфавитом, так что букве А сопоставляется диапазон от 0.0 до 0.2, букве В – диапазон от 0.2 до 0.4, букве С – диапазон от 0.4 до 0.6, букве

D – диапазон от 0.6 до 0.8 и букве E – диапазон от 0.8 до 0.999.... Пользуясь этим алфавитом, закодируем слово BED.



Слово BED можно было бы закодировать любой дробью  $f$  такой, что  $0.384 \leq f < 0.392$ . Чем больше будет символов в строке, тем уже интервал.

Идея-то интересная. Но при таком кодировании строк дробями мы не получаем никакого сжатия. На помощь приходит еще одна идея. Вместо того чтобы назначать каждой букве алфавита одинаковую долю диапазона, сделаем эту долю пропорциональной частоте буквы. А получит 8.12 %, B – 1.49 %, ... Z – 0.07 %. Букве A будет сопоставлен диапазон от 0.0 до 0.0812, букве B – от 0.0812 до 0.0961, а букве Z – от 0.9993 до 0.9999....

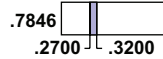
Теоретически это могло бы дать оптимальное сжатие, основанное на частоте отдельных букв. Но, к сожалению, на практике возникает проблема. Метод порождает дроби, которые потенциально могут состоять из тысяч и даже миллионов цифр. Как такие дроби представить в компьютере? Как производить с ними арифметические действия?

Итак, метод, который казался таким замечательным в теории, похоже, неосуществим на практике. Кажется, что нужны дроби неограниченной точности. Время, необходимое для сложения и умножения длинных дробей, все равно – десятичных или двоичных, возрастает вместе с их длиной, так что даже если бы мы придумали хороший способ представления таких дробей, метод все равно оказался бы слишком медленным для практических целей.

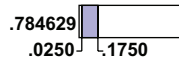
Найденное мной решение заключалось в использовании скользящего окна, в котором производятся все арифметические действия. Это позволяет работать с обыкновенными 32-битовыми целыми. Арифметика с плавающей точкой не нужна. Чтобы целые числа не выходили за пределы диапазона, представимого 32 битами, частоты букв аппроксимировались 15-битовыми целыми, т. е. дробями вида  $N/2^{15}$  или  $N/32768$ . Например, частота буквы A равна 8.12 %. Это число можно приближенно записать в виде  $2660/32768$ , или  $665/8192$ . Как выяснилось, такая аппроксимация не приводит к заметному уменьшению степени сжатия.

Покажем на примере (десятичном), как кодируется буква и как работает скользящее окно. Предположим, что первые несколько сим-

волов уже закодированы и диапазон теперь составляет от 0.784627 до 0.784632. Первые 4 цифры начала и конца диапазона одинаковы – 0.7846. Эти 4 цифры будут выведены, а окно сдвинуто на 4 цифры вправо, так что в нем показывается диапазон от 0.2700 до 0.3200.



Ширина этого диапазона равна 0.0500. Предположим, что следующий символ сообщения имеет частоту 0.0300 и ему соответствует диапазон от 0.4050 до 0.4350. Этот символ кодируется путем выбора такой доли текущего диапазона от 0.2700 до 0.3200. Ширина этой доли равна  $0.0500 \times 0.0300 = 0.0015$ . Она простирается от  $0.2700 + 0.0500 \times 0.4050$  до  $0.2700 + 0.0500 \times 0.4350$ , т. е. от 0.29025 до 0.29175. Заметим, что ширина этого диапазона равна 0.0015, как и ожидалось.



Поскольку начало и конец диапазона начинаются цифрами 0.29, их можно вывести. В этот момент выведены цифры 784629. Окно можно сдвинуть на две цифры вправо, так что текущий диапазон станет равным 0.0250–0.1750.

Арифметическое кодирование идеально подходит для шифрования, потому что больше нет никаких дискретных кодов для каждой буквы или комбинации букв. Отсутствуют границы, по которым поток битов может быть разбит на отдельные коды. Вместо этого код каждой буквы влияет на представление всех последующих букв.

Поняв, как работает метод арифметического кодирования, сделаем следующий шаг – посмотрим, как применить его к шифрованию. Мы не хотим изменять процентную долю диапазона, выделенную каждому символу, потому что тогда утратили бы преимущества сжатия. Но зато мы можем изменить порядок символов, так что диапазон каждого символа попадет в непредсказуемую часть полного диапазона – для Эмили непредсказуемую. Например, если используются только буквы A, B, C, D, E, то диапазоны могут быть такими:

Стандартный порядок			Перемешанный порядок		
A	8.12%	0.0000 - 0.0812	D	4.32%	0.0000 - 0.0432
B	1.49%	0.0812 - 0.0961	A	8.12%	0.0432 - 0.1244
C	2.71%	0.0961 - 0.1232	B	1.49%	0.1244 - 0.1393
D	4.32%	0.1232 - 0.1664	E	12.02%	0.1393 - 0.2595
E	12.02%	0.1664 - 0.2866	C	2.71%	0.2595 - 0.2866

Эти интервалы можно использовать для кодирования букв сообщения. Назовем этот метод *арифметическим шифрованием*. Посколь-

ку Эмили не знает ни начальных, ни конечных точек диапазонов, нет никакой отправной точки для атаки. Да, Эмили знает, что первый диапазон начинается в точке 0.0, а последний заканчивается в точке 0.999..., но не знает, какие символы представляют эти диапазоны.

У метода арифметического кодирования есть одна трудность, которая до сих пор не обсуждалась. При использовании обычного алфавита Рива не знает, где заканчивается сообщение. Тот же код, который обозначает ROTUND, может обозначать ROTUNDA, ROTUNDAA, ROTUNDAAA и так далее до бесконечности, в предположении, что диапазон для A начинается в точке 0. При традиционном арифметическом кодировании эту проблему можно решить, используя различные способы кодирования длины сообщения и добавив код длины в сжатый текст, либо же включив в алфавит специальный символ конца сообщения. Мы не обсуждали этот вопрос раньше, потому что для арифметического шифрования это не нужно.

При арифметическом шифровании нужно лишь сопоставить редкому символу или любому символу, который редко встречается в конце сообщения, первый диапазон, т. е. диапазон, начинающийся с 0.0000. Тогда Рива, увидев строку ROTUNDVVV... или ROTUND###..., сразу поймет, что сообщение закончилось.

В описанном выше виде арифметическое шифрование получает оценку 5 в случае применения к 26-буквенному алфавиту и оценку 6, если алфавит 256-буквенный. Можно использовать все обычные приемы: null-символы, омофоны и биграммы. Использование null-символов уменьшает или вообще сводит на нет сжатие, поэтому не рекомендуется. Омофоны, по сути дела, разбивают диапазон буквы на два или более отдельных диапазонов. В результате диапазоны букв выравниваются по длине, что эквивалентно сглаживанию частот букв. Это может повысить безопасность, не снижая степень сжатия. Использование биграмм или даже триграмм иногда увеличивает степени сжатия, одновременно повышая безопасность. С омофонами и биграммами арифметическое шифрование получает оценку 8.

Поскольку арифметическое шифрование так стойко само по себе, потребуется очень мало усилий, чтобы довести оценку до 10. Я рекомендую многоалфавитный шифр с периодом 4, т. е. подстановку с четырьмя независимыми хорошими перемешанными алфавитами, которые используются по очереди. Арифметическое шифрование, за которым следует стандартное многоалфавитное шифрование с периодом 4 или больше, получает оценку 10. Оно не оставляет противнику никаких зацепок – ни частот букв или контактов, ни какого-либо способа ухватиться за вероятное слово.

### **10.7.3 Адаптивное арифметическое кодирование**

Метод Лемпеля–Зива неплохо сжимает файлы любого типа в силу своей адаптивности. Кодирование Хаффмана и арифметическое

кодирование сжимают лучше, но только такие файлы, в которых частоты символов соответствуют заранее заданной таблице частот. Существует несколько способов сделать кодирование Хаффмана и арифметическое кодирование адаптивными, и все они повышают стойкость соответствующих методов шифрования. Во всех этих методах по ходу кодирования подсчитываются символы в файле.

Чем ближе счетчики символов к их частотам, тем выше степень сжатия. Естественно возникает мысль, почему бы просто не подсчитать все символы в файле и не вычислить реальные частоты. Беда в том, что Рива-то не может подсчитать символы в файле. Но Рива обязана использовать те же частоты, что и Сандра, иначе не сможет дешифровать файл. Решение этой дилеммы состоит в том, что Сандра подсчитывает символы в процессе шифрования, а Рива – в процессе дешифрования, поэтому на любом этапе они будут оперировать одинаковыми счетчиками.

В начальный момент все счетчики символов равны 1. Если вы знаете частоты символов заранее, пусть даже это грубые оценки, то можно увеличить счетчики для самых частых символов. Например, если используется 256-символьный набор и ожидается, что сообщения будут содержать 1 % заглавных букв Е и примерно 10 % строчных букв е, то можно было бы увеличить счетчик для Е на 2, а счетчик для е на 25, т. е. примерно до 10 % от 256. Начальный диапазон для каждого символа пропорционален его начальному счетчику. Например, сумма счетчиков всех 256 символов равна 500, а начальный счетчик для строчной е равен 25, тогда е получит диапазон  $25/500 = 0.05$ .

Есть два основных режима корректировки кодов: символьный и пакетный. Символьный режим пригоден только для арифметического кодирования. В этом режиме при обработке каждого символа в файле корректируется его диапазон и оба соседних. (Только один соседний диапазон, если символу сопоставлен первый или последний диапазон. Для стандартного 26-буквенного алфавита это означает буквы А и Z.)

Рассмотрим пример. Пусть встретилась буква Т, и соседние диапазоны принадлежат буквам S и U. (При арифметическом шифровании, скорее всего, будет не так. В перемешанном алфавите буквы S, Т, U вряд ли встречаются последовательно именно в таком порядке.) Предположим, что счетчики для S, Т, U равны 15, 20 и 5, что в сумме дает 40. Предположим также, что диапазоны для S, Т и U равны 0.062, 0.074 и 0.024, что в сумме дает 0.160. Этот объединенный диапазон перераспределяется в отношении 15:20:5. То есть  $S = 0.160 \times 15/40 = 0.060$ ,  $T = 0.160 \times 20/40 = 0.080$ , а  $U = 0.160 \times 5/40 = 0.020$ . Со временем длины диапазонов символов сойдутся к правильным значениям.

Символьный режим довольно хорошо работает с 26-буквенным алфавитом. Но из рук вон плохо с 256-символьным алфавитом. Большая часть 256 символов не соседствует с высокочастотными символами, поэтому их частоты остаются статичными. В особенности это



справедливо для представления в стандартном коде ASCII, где все буквы занимают один непрерывный участок.

Пакетный режим работает и для арифметического кодирования, и для кодирования Хаффмана. В пакетном режиме все множество диапазонов корректируется в определенных точках в процессе кодирования. Например, диапазоны можно было бы корректировать после кодирования 64, 128, 256 символов и т. д. В каждой из этих точек весь диапазон перераспределяется в соответствии с текущими счетчиками символов. Процесс сходится быстрее, чем в символьном режиме, но между перераспределениями мы будем работать со старыми, неоткорректированными частотами.

В пакетном режиме можно подсчитывать частоты биграмм и даже триграмм. Биграммам и триграммам, встречающимся более одного раза, можно было бы назначать собственные коды Хаффмана или диапазоны арифметических кодов. С таким усовершенствованием арифметическое кодирование почти всегда дает большую степень сжатия, чем метод Лемпеля–Зива.

С подсчетом частот биграмм и триграмм связана одна проблема – объем потребной памяти. Для 256-символьного алфавита существует 65 536 различных биграмм и 16 777 216 триграмм. Если памяти много, то это, возможно, и не проблема. Но если памяти недостаточно, то можно, например, подсчитывать только биграммы и триграммы, содержащие самые частые буквы. Так, если ограничиться только 20 самыми частыми символами, то придется подсчитывать всего 200 биграмм и 8000 триграмм. Чтобы определить, какие символы самые частые, подсчет частот биграмм и триграмм можно отложить до того момента, как будет закодировано некоторое фиксированное количество одиночных символов, скажем 256 или 1024.

Например, мы можем подсчитывать только одиночные символы в первом пакете и таким образом выяснить, какие символы встречаются чаще всего. Во втором пакете подсчитываются биграммы, содержащие эти высокочастотные символы. В третьем пакете подсчитываются триграммы, содержащие только сочетание высокочастотной биграммы и высокочастотного символа. После того как высокочастотные биграммы и триграммы определены, им назначаются собственные коды Хаффмана или арифметические диапазоны. Иными словами, они рассматриваются как одиночные символы.

Для арифметического кодирования символьный и пакетный режимы не являются взаимоисключающими. Диапазоны отдельных символов можно балансировать, когда они встречаются, а для наборов символов, расширенных биграммами и триграммами, – производить балансировку в конце каждого пакета.

При арифметическом шифровании или шифровании Хаффмана в конце каждого пакета алфавит следует заново перемешивать, перед заменой кодов или перебалансировкой диапазонов. Особенно это важно, если биграммы или триграммы были добавлены либо удалены. Это значит, что Эмили сможет набрать мало материала для

атаки, перед тем как коды изменятся. При шифровании желательно использовать пакеты нерегулярной длины, скажем 217 символов, потом 503 символа и т. д., чтобы Эмили не знала, когда происходит изменение кодов.

Еще одно улучшение адаптивного кодирования – деление всех счетчиков на 2 после перебалансировки диапазонов. Это дает кодам возможность адаптироваться к ситуации, когда частоты символов изменяются. Старые частоты будут слабее влиять на диапазоны, а новые – сильнее. Например, предположим, что текст представляет собой сборник рассказов разных авторов. У каждого автора свой словарь, свой предмет повествования, да даже и языки могут быть разные.

Разумеется, Сандра и Рива должны договориться обо всем этом заранее, только тогда Рива сможет правильно расшифровать и распаковать сообщение.



# 11

## Блочные шифры

---

### **Краткое содержание главы:**

- стандарты шифрования DES и AES;
- шифры на основе умножения матриц;
- инволютивные шифры, в которых процессы шифрования и дешифрирования идентичны;
- пульсирующие шифры;
- сцепление блоков.

Мы уже рассматривали несколько шифров, применяемых к тексту, разбитому на блоки символов. Некоторые работают с блоками, содержащими всего 2 или 3 символа, например шифр Плейфера, Two Square, Three Square и Four Square. Другие применяются к более длинным блокам, но изменяют всего 2 или 3 символа за раз, например Bifid, Trifid и FR-Actionated Morse. Это шифры локальные, воздействующие всего на часть каждого блока. Изменение одного символа открытого текста, как правило, приводит к изменению не более 2 или 3 символов шифртекста.

В этой главе мы будем иметь дело с гораздо более стойкими типами блочных шифров, в которых изменение всего одного бита открытого текста или ключа приводит к изменению примерно половины битов и почти всех байтов шифртекста. Это означает, что шифр в высшей степени нелинеен (см. раздел 12.3). Такие шифры предна-

значены только для компьютеров, зачастую оснащенных специализированным оборудованием для повышения быстродействия.

Большая часть последующих глав посвящена компьютерным шифрам и методам. Если вас эта тема не интересует, просто пропустите соответствующие разделы.

## 11.1 Подстановочно-перестановочная сеть

Многие блочные шифры имеют вид *подстановочно-перестановочной сети* (SP-сети). Впервые эта идея была выдвинута Хорстом Фейстелем из компании IBM в 1971 году. Шифрование состоит из нескольких раундов, каждый из которых может включать один или несколько шагов подстановки и (или) перестановки. Обычно имеется один главный ключ, управляющий всей работой алгоритма.

На шаге подстановки чаще всего используются (1) простая подстановка, (2) применение операции ИСКЛЮЧАЮЩЕЕ ИЛИ к части блока и части ключа, (3) многоалфавитная подстановка под контролем ключа. Ключ может состоять из битов, взятых из главного ключа и (или) из той части блока, к которой подстановка не применяется. Например, байты блока с нечетными номерами можно использовать как ключ для шифрования байтов с четными номерами или наоборот. В чуть более сложной подстановке берутся некоторые биты из ключа, к ним и к такому же количеству битов блока применяется ИСКЛЮЧАЮЩЕЕ ИЛИ, а результат используется в качестве многоалфавитного ключа для подстановки другой части блока.

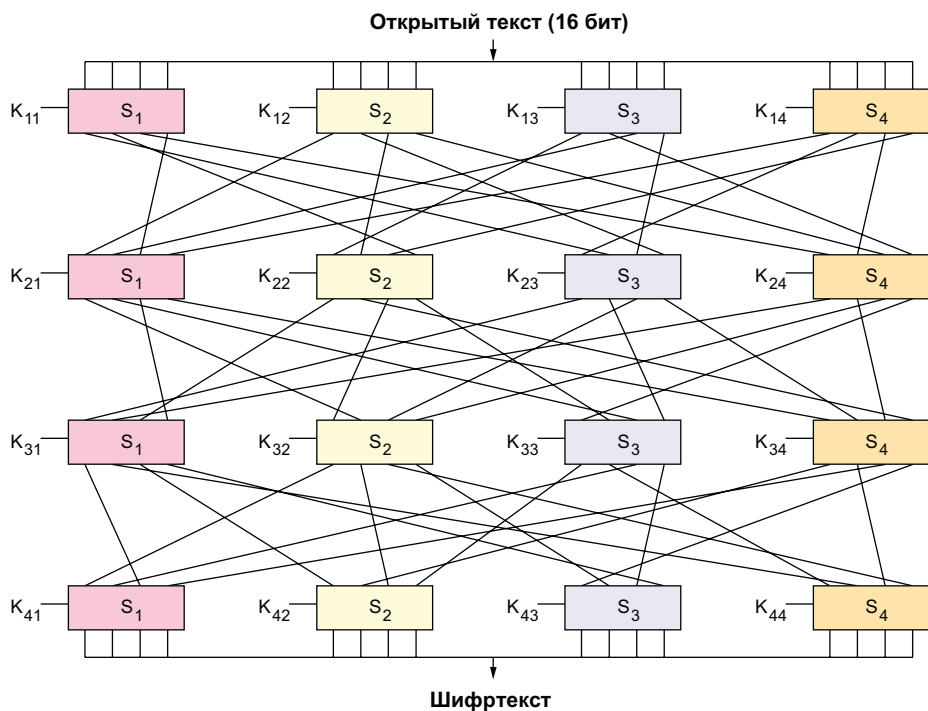
Алфавиты подстановки обычно выбираются заранее и никогда не изменяются. Они называются *S-блоками*. Это могут быть как простые, так и многоалфавитные подстановки, т. е. S-блок – компьютерный эквивалент таблицы алфавитов. Обычно для выбора строки из таблицы используется от 4 до 8 бит ключа, на вход подается от 4 до 8 бит блока и столько же битов оказывается на выходе. Нередко при построении алфавитов подстановки применяется сложная математика. В частности, алфавиты должны быть нелинейными, что подробно описано в разделе 12.3.

Перестановки для каждого раунда обычно также предопределены и не изменяются. Они могут применяться к различным единицам: одиночным битам, 4-битовым группам или 8-битовым байтам. В большинстве блочных шифров нет специальных ключей для перестановок, они зашиты в программу или шифровальную микросхему.

Самым ранним из современных блочных шифров является *Licifer*, спроектированный Хорстом Фейстелем. Название несколько раз изменялось, пока наконец Фейстель не остановился на Люцифере, поскольку хотел, чтобы имя подчеркивало дьявольскую природу изобретения. Позже Фейстель несколько раз изменял структуру своего шифра; оригинальный шифр имел 48-битовый ключ и применялся к 128-битовым блокам, а конечная версия работала уже

со 128-битовым ключом, применявшимся к 128-битовым блокам. Дополнительные сведения о шифре Lucifer можно почерпнуть по адресу <https://derekbruff.org/blogs/fywscrypto/tag/lucifer> (я обращался к нему в мае 2022 года).

Ниже показана схема миниатюрной подстановочно-перестановочной сети. Шифр принимает 16-битовый открытый текст и порождает 16-битовый шифртекст. Он состоит из 4 раундов подстановки и 3 раундов перестановки. Подстановки и перестановки фиксированы и встроены в оборудование. Имеется 4 различные подстановки:  $S_1$ ,  $S_2$ ,  $S_3$  и  $S_4$ . Каждая подстановка принимает 4 входных бита и биты ключа (обычно 4, 6 или 8), от  $K_{11}$  до  $K_{44}$ , так что шифр потенциально мог бы иметь 64-, 96- или 128-битовые ключи, при условии что все ключи независимы. Перестановки на каждом раунде различны.



Этот шифр на основе мини-сети получает оценку 3, потому что он эквивалентен подстановке биграмм. Но его можно масштабировать с 16 до 64 бит с 6 раундами подстановки, и тогда он получит оценку 8. Если же увеличить размер блока до 128 бит, а число раундов – до 8, то оценка повысится до 10.

Дизайн окончательной версии Lucifer проложил прямой путь к стандарту шифрования данных (Data Encryption Standard – DES), который был формально принят Национальным бюро стандартов (NBS) в 1977 году. Поэтому к нему и перейдем.

## 11.2 Стандарт шифрования данных (DES)

DES был разработан в IBM в 1976 году путем понижения стойкости текущей на тот момент версии Lucifer. В оригинальной версии использовались 128-битовые ключи, а сообщение разбивалось на 128-битовые блоки. Размер блока был уменьшен до 64 бит, это можно было оправдать снижением стоимости оборудования. IBM хотела использовать 64-битовый ключ, но АНБ настояло на еще большем уменьшении размера ключа, до 56 бит, под смехотворным предлогом, что дополнительные 8 бит можно использовать в качестве контрольной суммы. Общеизвестно, что истинной причиной было то, что АНБ умело вскрывать 56-битовый DES, но не 64-битовый.

Первоначально IBM планировала использовать в DES шесть раундов. Но когда АНБ сообщило, что умеет вскрывать версию с 6 раундами, IBM увеличила их число сразу до 16, т. е. столько же, сколько было в окончательной версии шифра Фейстеля Lucifer.

Новой особенностью DES, отсутствующей в Lucifer, стала перестановка битов перед первым и после последнего шагов подстановки. Это были столбцовые перестановки квадрата  $8 \times 8$ , при которых изменялся порядок как строк, так и столбцов. Для начальной перестановки 64 бита блока записывались в квадрат слева направо. Столбцы считывались в обратном порядке: 8, 7, 6, 5, 4, 3, 2, 1. Строки считывались в порядке 2, 4, 6, 8, 1, 3, 5, 7. Конечная перестановка является обращением начальной.

У этих перестановок нет криптографической ценности. Они ничуть не увеличивают стойкость DES и были добавлены только потому, что АНБ потребовало, чтобы шифрование было быстрым при аппаратной реализации, но медленным при программной. Делалось это для того, чтобы Эмили потратила больше времени на вскрытие шифра полным перебором ключей. IBM полагала, что перестановка битов сделает программную реализацию очень медленной. По-видимому, она считала, что биты будут выделяться по одному путем маскирования, а затем сдвигаться в нужную позицию.

Все оказалось неверно. Во-первых, противник, стремящийся вскрыть DES, мог бы просто купить криптографические микросхемы через посредников. Во-вторых, в некоторые вполне легитимные приложения необходимо было встраивать DES, поэтому программная реализация должна была работать быстро. В-третьих, перестановки битов можно реализовать быстро, не выделяя отдельных битов. Как это делается, я покажу в разделе 11.2.3.

Между начальной и конечной перестановками DES выполняет 16 раундов подстановки. 64-битовый блок разбивается на две 32-битовые половины. В каждом раунде правая половина используется для шифрования левой. Правая половина сначала расширяется с 32 до 48 бит следующим образом. 32 бита интерпретируются как восемь 4-битовых групп. Каждая группа расширяется с 4 до 6 бит путем

добавления одного бита в начало и одного в конец; эти биты берутся из соседних групп. Например, третья группа состояла бы из битов 9–12. Эта 4-битовая группа расширяется путем добавления бита 8 слева и бита 13 справа, т. е. состоит из битов 8, 9, 10, 11, 12, 13. Восемь таких 6-битовых групп образуют 48-битовый блок.

Затем к этому 48-битовому блоку и 48 битам, взятым из 56-битового ключа, применяется операция ИСКЛЮЧАЮЩЕЕ ИЛИ. Какие именно 48 бит использовать в каждом раунде, определяется алгоритмом *развертки ключа*; по существу, это сдвиг полного 56-битового ключа на несколько позиций после каждого раунда. После этого восемь результирующих 6-битовых групп подаются на вход восьми фиксированных S-блоков, т. е. подстановок. Каждый S-блок выдает 4-битовый результат, и восемь таких результатов объединяются в 32-битовый блок.

### Историческая справка

IBM не собиралась включать в DES развертку ключа. Оригинальная идея заключалась в том, чтобы циклически сдвигать 64-битовый ключ на 4 позиции после каждого из 16 раундов. В итоге ключ возвращался к исходному значению и был готов к шифрованию следующего блока. IBM была вынуждена ввести развертку, после того как АНБ потребовала уменьшить размер ключа до 56 бит, поскольку в таком случае сдвиги на 4 бита уже не достигали цели. Разумеется, IBM назвала развертку ключа «функциональной особенностью».

На S-блоки можно взглянуть и по-другому, изобразив их в виде таблицы 4×16. Как и в таблице алфавитов в шифрах Беласо и Виженера, каждая строка является таблицей подстановки для 4-битовых групп. Два дополнительных бита, добавленных к 4-битовым группам, используются для выбора одной из четырех строк.

Все S-блоки были тщательно спроектированы таким образом, чтобы корреляция между 6 входными и 4 выходными битами была как можно меньше. АНБ придумало сверхсекретный способ проектирования S-блоков с минимально возможной корреляцией. В силу особой важности DES АНБ решило поделиться этим секретом с инженерами из IBM. Однако, внимательно изучив проект IBM, АНБ пришлось к выводу, что этот метод был известен IBM и использован в проекте.

После каждого раунда, кроме последнего, левая и правая половины 64-битового блока меняются местами.

### 11.2.1 Double DES

С самого начала было понятно, что 56-битового ключа недостаточно для обеспечения криптографической стойкости. Всего через четыре месяца после принятия DES компания Electronic Frontier Foun-

dation построила специализированный компьютер стоимостью 250 000 долларов, названный Deep Crack, который вскрывал сообщение, зашифрованное DES, всего за 56 часов.

Для устранения этой очевидной слабости было предложено шифровать сообщение дважды шифром DES с различными ключами. Эту идею отвергли, потому что она оставляла теоретическую возможность организовать атаку со встречей посередине, которая вскрыла бы DES. Это означает, что мы идем вперед от открытого текста и назад от шифртекста и встречаемся посередине. Для этого нужен блок шифртекста с известным открытым текстом. Мы шифруем открытый текст всеми  $2^{56}$  возможными ключами и дешифрируем шифртекст также всеми  $2^{56}$  возможными ключами. Если при сравнении результатов обнаружено совпадение, то мы нашли возможную пару ключей.

Эта атака была чисто теоретической. Для полного сравнения потребовалось бы хранить 2 набора по  $2^{56}$  решений, т. е.  $2^{60}$  байт. В 1970-х годах не было компьютеров с такой памятью. Кроме того, имеется  $2^{48}$  ожидаемых совпадений, и все их нужно было бы проверить. Это чрезвычайно трудная задача. Но IBM и АНБ рассчитывали, что DES будет использоваться 20–30 лет, а за это время такая атака могла бы стать практически реализуемой. Double DES можно использовать, но в качестве стандарта он не был принят.

### 11.2.2 Triple DES

*Triple DES*, или *3DES*, – еще одна попытка компенсировать небольшой размер ключа в DES. Идея в том, чтобы взять 64-битовый блок, зашифровать его одним ключом, затем расшифровать другим и снова зашифровать третьим. Очевидно, что работать это будет в 3 раза дольше обычного DES. Шифр не получил широкого распространения из-за своей медлительности.

Существует гораздо более быстрый способ повысить безопасность DES. Просто нужно применить ИСКЛЮЧАЮЩЕЕ ИЛИ к 64-битовому блоку и 64-битовому ключу перед выполнением DES и то же самое проделать с другим 64-битовым ключом после выполнения DES. Всего, стало быть, имеется три независимых ключа: два 64-битовых для ИСКЛЮЧАЮЩЕГО ИЛИ и 56-битовый для самого DES, суммарной длиной 184 бита. Этот метод лишь ненамного медленнее одного DES-шифрования.

Даже если бы ключ второго ИСКЛЮЧАЮЩЕГО ИЛИ можно было определить путем анализа волновых форм, все равно этот шифр гораздо более стойкий, чем простой DES. Устранить опасность раскрытия ИСКЛЮЧАЮЩЕГО ИЛИ в результате анализа волновых форм можно, выполнив простые подстановки с ключом до и после шага DES.

### \*11.2.3 Быстрая перестановка битов

DES начинается и заканчивается перестановкой битов. Простодушный способ выполнения этой операции – распаковывать биты по одному, сдвигать их в нужную позицию, а затем ставить на место с помощью операции ИЛИ. Но есть гораздо более быстрый способ, придуманный независимо мной и Дэвидом Стивенсоном из отделения IBM Research в Йорктауне, штат Нью-Йорк, в 1975 году. Продемонстрирую его на примере перестановки 32-битового блока. Предположим, что открытый текст в битовой форме имеет вид

abcdefgh ijklmnop qrstuvwx yzαβδεζ

где латинские и греческие буквы представляют один бит, т. е. могут принимать значение 0 или 1. Посмотрим, как выполнить перестановку, чтобы бит а перешел в третью позицию, b – в шестую, c – в девятую и т. д.

Для этого нам понадобятся четыре специальные таблицы, каждая с 256 элементами. Элементом является 32-битовый блок, или одно машинное слово. Первая таблица показывает переставленные позиции 8 бит первого байта 32-битового блока:

..a..b.. c..d..e. .f..g..h .....

Вторая таблица показывает позиции 8 бит второго байта 32-битового блока:

k..l..m. .n..o..p .....i..j..

Третья таблица показывает позиции 8 бит третьего байта 32-битового блока:

.v..w..x .....q..r.. s..t..u.

Четвертая таблица показывает позиции 8 бит четвертого байта 32-битового блока:

.....y..z.. α..β..γ. .δ..ε..ζ

Точки нужны для того, чтобы лучше видеть, куда помещены все 32 бита. Они представляют нули в машинном слове. Теперь для выполнения перестановки нужно найти все четыре байта в этих специальных таблицах и объединить четыре найденных 32-битовых блока операцией ИЛИ:

..a..b.. c..d..e. .f..g..h .....  
k..l..m. .n..o..p ..... i..j..  
.v..w..x ..... .q..r.. s..t..u.  
..... .y..z.. α..β..γ. .δ..ε..ζ  
kva1wbmx cnydozep afqβgrγh sδitejuζ

Не нужно ни сдвигов, ни маскирования. Перестановка всех 32 бит сводится к четырем поискам в таблице и трем операциям ИЛИ. Одно из применений этой техники – транспонирование блока битов  $8 \times 8$ . Это можно сделать с помощью восьми таблиц с 256 элементами или всего одной таблицы, сдвигающей биты в нужные позиции внутри каждого байта:

Без сдвига	a.....	b.....	c.....	d.....	e.....	f.....	g.....	h.....
Сдвиг на 2	..a.....	..b.....	..c.....	..d.....	..e.....	..f.....	..g.....	..h.....

\*\*

### 11.2.4 Неполные блоки

Проблема, возникающая в DES и других блочных шифрах, – что делать с неполными блоками. В DES все блоки должны содержать ровно 8 символов. Допустим, что в сообщении 803 символа, т. е. 100 полных блоков по 8 символов плюс еще 3. Как быть с этими последними тремя символами?

Традиционное решение – дополнить последний блок null-символами. На бумаге это обычно делали, дописывая XXXXX или NULLS в качестве последних 5 символов. К сожалению, это дает Эмили 5 известных букв открытого текста. Для ручных шифров есть решения лучше, например использовать маркер типа XX или JQ, а остальные дополняющие символы выбирать случайно, скажем XXESV, или просто дополнить блок произвольной комбинацией низкочастотных символов, например ZPGWV. Дешифрирование зависит от способности Ривы понять, где кончается реальное сообщение и начинаются дополняющие символы.

В компьютерной реализации дополнение должно решить две проблемы. Во-первых, Рива должна понимать, где кончается сообщение, или, эквивалентно, сколько добавлено байтов дополнения. Во-вторых, Сандра хочет, чтобы объем известного Эмили открытого текста был как можно меньше. Некоторые предложенные схемы не удовлетворяют обоим критериям. Например, в одной из схем предлагалось дополнять сообщения следующим образом:

```

01
02 02
03 03 03
. . .

```

Это может дать Эмили до 31 байта известного открытого текста при размере блока 32. В файле общего вида последний блок мог бы быть полным и заканчиваться байтом 01 или даже 02 02. Эти байты по ошибке можно было бы принять за дополнение.

Лучше было бы поместить поле длины где-то в открытом тексте. Это необязательно 4-байтовое полное число байтов в файле, до-



статочно указать число дополняющих байтов в последнем блоке. В случае DES таковых может быть от 0 до 7, т. е. хватило бы всего трех бит. Поле длины может находиться в любом месте файла. Чаще всего используется первый байт, последний байт или первый байт последнего блока. Сами заполняющие байты можно выбирать произвольно.

Чтобы не выдать Эмили ни одного байта открытого текста, длину можно закодировать в младших или старших битах индикатора длины, а остальные, неиспользуемые, биты выбрать случайным образом. Тогда индикатор длины сможет принимать любое значение от 0 до 255.

Кстати говоря, никто не заставляет включать дополняющие байты именно в конец файла. Хотите – размещайте их в начале последнего блока, хотите – в середине тринадцатого – полная свобода. Коль скоро Сандра и Рива договорятся, они могут делать что угодно, чтобы сбить Эмили со следа. Интересная мысль – разбросать дополняющие байты по всему файлу. Например, если требуется 4 дополняющих байта, то можно было бы поместить их в конец второго, четвертого, шестого и восьмого блоков файла. Главное – чтобы Рива могла понять, сколько байтов было добавлено.

Альтернативой дополнению является *метод перекрытия*. Снова предположим, что размер блока  $B = 8$ , что сообщение содержит 803 символа и что первые 800 зашифрованных символов образуют 100 блоков по 8 символов. После этого мы формируем 101-й блок, в котором зашифрованы символы с 796 по 803. Теперь длина сообщения не изменилась, но Рива должна дешифровать блок 101 раньше, чем блок 100.

## 11.3 Умножение матриц

Далее мы рассмотрим блочный шифр *Advanced Encryption Standard* (AES – улучшенный стандарт шифрования). Но в AES используется еще не встречавшаяся в этой книге математическая операция – умножение матриц. Во введении я обещал, что буду вводить математический аппарат по мере необходимости и, оставаясь верным обещанию, расскажу об умножении матриц именно здесь. Эта концепция понадобится нам в нескольких последующих главах. Если вы уже знакомы с умножением матриц, можете пропустить этот раздел.

*Матрица* – это просто прямоугольный массив элементов, называемых скалярами. Последовательность скаляров образует *вектор*, поэтому строки и столбцы матрицы – векторы. Они так и называются: *векторы-строки* и *векторы-столбцы*. Матрица, содержащая  $m$  строк и  $n$  столбцов, называется матрицей  $m \times n$ . Если  $m = n$ , то матрица называется *квадратной*. Ниже приведен пример матрицы  $M$  с 3 строками и 5 столбцами, т. е. матрицы  $3 \times 5$ . Она содержит 15 скалярных элементов, обозначенных буквами от  $a$  до  $o$ :

$$\begin{pmatrix} a & b & c & d & e \\ f & g & h & i & j \\ k & l & m & n & o \end{pmatrix}.$$

В матрице три вектора-строки:  $[a,b,c,d,e]$ ,  $[f,g,h,i,j]$  и  $[k,l,m,n,o]$  – и пять векторов-столбцов:  $[a,f,k]$ ,  $[b,g,l]$ ,  $[c,h,m]$ ,  $[d,i,n]$  и  $[e,j,o]$ . Строки матрицы нумеруются сверху вниз, а столбцы – слева направо. Элемент матрицы  $M$ , расположенный на пересечении строки  $i$  и столбца  $j$ , обозначается  $M_{ij}$ , т. е.  $M_{11}$  – это  $a$ ,  $M_{15}$  –  $e$ , в  $M_{31}$  –  $k$ .

Скаляры могут быть целыми числами, целыми числами по модулю  $N$ , рациональными, вещественными, комплексными числами и значениями других типов, описанных ниже. Операция умножения матриц не зависит от типа числа.

Произведение двух матриц  $X$  и  $Y$ , обозначаемое  $XY$ , образуется путем умножения строк  $X$  на столбцы  $Y$ . Рассмотрим этот вопрос подробнее. Строками и столбцами матрицы являются векторы. Для двух векторов одинаковой длины определено *внутреннее*, или *скалярное*, *произведение*, иногда оно обозначается точкой  $\cdot$ . Оно вычисляется как сумма произведений соответственных элементов векторов.

Пусть первый вектор –  $[a,b,c,d]$ , а второй –  $[e,f,g,h]$ . Поскольку длины одинаковы, векторы можно перемножить. Скалярное произведение равно

$$[a,b,c,d] \cdot [e,f,g,h] = ae + bf + cg + dh.$$

Пусть  $X$  и  $Y$  – матрицы  $4 \times 4$ , а  $P$  – их произведение, т. е.  $P = XY$ . Пусть  $[a,b,c,d]$  –  $i$ -я строка  $X$ , а  $[e,f,g,h]$  –  $j$ -й столбец  $Y$ . Их произведение обозначим  $P_{ij}$ . Иными словами, элементом на пересечении  $i$ -й строки и  $j$ -го столбца произведения матриц является произведение  $i$ -й строки  $X$  и  $j$ -го столбца  $Y$ . В форме с индексами это можно записать так:

$$P_{ij} = [X_{i1}, X_{i2}, X_{i3}, X_{i4}] \cdot [Y_{1j}, Y_{2j}, Y_{3j}, Y_{4j}] = X_{i1}Y_{1j} + X_{i2}Y_{2j} + X_{i3}Y_{3j} + X_{i4}Y_{4j}.$$

Аналогично выглядит выражение для матриц других размеров. Две матрицы размера  $a \times b$  и  $c \times d$  можно перемножить, только если  $b = c$ .

## 11.4 Умножение матриц

Нет, повторяющееся название раздела – не ошибка. В математике много объектов, помимо чисел, которые можно складывать и умножать. Вот лишь несколько примеров: векторы, матрицы, полиномы, кватернионы и вообще элементы любого кольца. Можно рассматривать даже вектор матриц, матрицы полиномов и т. д. О кольцах мы

еще поговорим в разделах 15.6–15.8. Умножение матриц основано на типах их элементов и правил их сложения и умножения. Сама же процедура всегда одинакова. Для получения элемента на пересечении  $i$ -й строки и  $j$ -го столбца матрицы  $XY$  нужно вычислить скалярное произведение  $i$ -й строки  $X$  и  $j$ -го столбца  $Y$ .

Умножение матриц не коммутативно, т. е. произведение квадратных матриц  $X$  и  $A$  зависит от порядка сомножителей.  $AX \neq XA$ . Эти операции называются умножением  $X$  на  $A$  слева и справа.

В случае AES нас интересует сложение и умножение многочленов (они же полиномы). Все мы в школе на уроках алгебры учили, как это делается. Те, кто в дальнейшем работал в сфере науки и техники, наверное, и до сих пор помнят. Полиномы можно также делить. При делении может образовываться остаток, поэтому для полиномов, как и для целых чисел, определено понятие модуля (если вы забыли, что это такое, вернитесь к разделу 3.6).

Умножение скаляров, применяемое в AES, – это умножение не целых чисел, а двух полиномов по модулю третьего полинома. И дальше, пожалуй, не стоит углубляться, потому что книга рассчитана на широкую аудиторию.

## 11.5 Улучшенный стандарт шифрования (AES)

AES – более современный блочный шифр, сменивший DES в 2001 году. Первоначально он назывался *Rijndael* по именам авторов, бельгийских криптографов Винсента Рэймана (Vincent Rijmen) и Йоана Даймена (Joan Daemen). Поначалу AES мог использоваться в пяти конфигурациях: 128- или 256-битовые блоки в сочетании с 128-, 192- или 256-битовыми ключами. Однако Национальный институт стандартов и технологий (NIST) выбрал в качестве стандарта 128-битовый блок. Количество раундов зависит от размера ключа: 10 раундов для 128-битовых ключей, 12 для 192-битовых и 14 для 256-битовых.

В каждом раунде используется раундовый ключ, состоящий из 128 бит, выбранных из полного ключа по алгоритму *развертки*. Перед первым раундом выполняется предварительная операция *AddRoundKey*, заключающаяся в применении ИСКЛЮЧАЮЩЕГО ИЛИ к блоку и раундовому ключу. Каждый из последующих 9, 11 или 13 раундов состоит из четырех операций: *SubBytes*, *ShiftRows*, *MixColumns* и *AddRoundKey*. В последнем раунде шаг *MixColumns* отсутствует.

128-битовый блок рассматривается как матрица байтов  $4 \times 4$ , организованная *по столбцам*. Это означает, что байты записываются в матрицу по столбцам сверху вниз, а не по строкам:

$$\begin{pmatrix} b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \\ b_4 & b_8 & b_{12} & b_{16} \end{pmatrix}.$$

Первый шаг каждого раунда называется SubBytes. Это фиксированная простая подстановка, применяемая к каждому байту по отдельности. Подстановка специально спроектирована в высшей степени нелинейной. Свойство линейности подробно обсуждается в разделе 12.3.1.

Следующий шаг – ShiftRows. Это перестановка, при которой строки матрицы циклически сдвигаются на 0, 1, 2 и 3 позиции влево, как показано ниже:

$$\begin{pmatrix} b_1 & b_5 & b_9 & b_{13} \\ b_6 & b_{10} & b_{14} & b_2 \\ b_{11} & b_{15} & b_3 & b_7 \\ b_{16} & b_4 & b_8 & b_{12} \end{pmatrix}.$$

Третий шаг каждого раунда, MixColumns, – умножение матриц. Но это не обычное умножение целочисленных матриц, описанное в разделе 11.3. Элементы матриц интерпретируются как коэффициенты полинома. Операции сложения и умножения скаляров заменяются операциями над полиномами по модулю еще одного полинома. Все это было тщательно спроектировано таким образом, чтобы операции быстро выполнялись оборудованием. В последнем раунде шаг MixColumns опущен.

Последний шаг каждого раунда – AddRoundKey. Это просто операция поразрядного ИСКЛЮЧАЮЩЕГО ИЛИ блока с частью ключа, определяемой алгоритмом развертки.

Лично мне это ИСКЛЮЧАЮЩЕЕ ИЛИ в конце кажется очень подозрительным. Несколько инженеров-электротехников говорили мне, что волновые формы, генерируемые ИСКЛЮЧАЮЩИМ ИЛИ с 00 и 11, отличаются от генерируемых ИСКЛЮЧАЮЩИМ ИЛИ с 01 и 10, так что пассивный противник может понять, чему равны оба бита. Потенциально это может выдать противнику 128 бит ключа. Занимаясь криптографией высшего класса безопасности, я стараюсь всюду, где возможно, избегать ИСКЛЮЧАЮЩЕГО ИЛИ.

Если я все-таки вынужден использовать ИСКЛЮЧАЮЩЕЕ ИЛИ в конце шифрования, например при реализации стандартизованного алгоритма, то обязательно инвертирую каждый бит шифртекста четное число раз. Я храню две случайные битовые строки R1 и R2 того же размера, что и блок, и результат применения к ним ИСКЛЮЧАЮЩЕГО ИЛИ,  $R3 = R1 \oplus R2$ . Затем я применяю ИСКЛЮЧАЮЩЕЕ ИЛИ к шифртексту и R1, потом к R2 и, наконец, к R3. Это возвращает битовую строку к исходному значению, и, хочется надеяться, предательские волновые формы нейтрализуются.

Альтернативно для инвертирования всех битов блока можно использовать подстановку, а не ИСКЛЮЧАЮЩЕЕ ИЛИ. Это нужно сделать дважды, так что вместо трех ИСКЛЮЧАЮЩИХ ИЛИ мы имеем два шага подстановки. При использовании AES я настоятельно рекомендую добавить этот дополнительный финальный шаг.

## 11.6 Фиксированная подстановка и подстановка с ключом

Ранее в этой книге во всех подстановках использовались алфавиты, перемешанные с помощью ключевых слов или числовых ключей. В шифрах, рассматриваемых в этой главе, DES и AES, используются фиксированные подстановки, которые можно зашить в S-блоки. Что лучше? Чья стойкость выше?

Фиксированную подстановку с помощью хитроумной математики можно спроектировать так, что она устоит против разнообразных атак. Например, если некоторые выходные биты сильно коррелированы с какими-то входными битами, то Эмили сможет организовать статистическую атаку на шифр типа той, что я применил против цилиндрического шифра Джефферсона в разделе 8.2.

К сожалению, фиксированная подстановка для Эмили является неподвижной мишенью. Она может изучать ее месяцами или годами и, возможно, найдет-таки слабость, которую проектировщик просмотрел. Тщательно подобранная подстановка часто обладает математической регулярностью. Она описывается специфической математической функцией. Это само по себе может являться слабостью, потому что дает Эмили короткий путь к моделированию вашего шифра.

Я предпочитаю подстановки, определяемые ключом, который можно менять для каждого сообщения. Каждый отдельный экземпляр подстановки с ключом, быть может, и слабее фиксированной подстановки, но Эмили не сможет воспользоваться этой слабостью, потому что не располагает таблицей подстановки для ее изучения. Если Эмили удастся получить открытый текст, быть может, шпионскими методами, она, возможно, сумеет восстановить подстановку и изучить ее слабости, но к тому времени будет слишком поздно. Знание слабостей ценно только тем, что позволяет дешифровать сообщение и получить открытый текст. Но если Эмили и так его знает, то ключ не представляет никакой ценности. Разведанная слабость не поможет дешифровать следующее сообщение, потому что оно зашифровано другим ключом, у которого слабость если и есть, то другая.

Любой другой экземпляр подстановки с алфавитом, перемешанным тем же методом, но с другим ключом, может быть лишен той же слабости. Возможно, у него есть слабость того же типа, например корреляция между некоторыми битами блока и (или) битами ключа и выходными битами, но в каждом экземпляре эти биты будут различны.

Аргумент в пользу фиксированных S-блоков – возможность синхронной работы шифровального оборудования, когда одно сообщение следует за другим без перерывов. При использовании перемешанных алфавитов возможны паузы на время, пока алфавит

перемешивается заново. Паузу можно устранить или, по крайней мере, уменьшить, если перемешивать алфавит параллельно, т. е. перемешивать алфавит для следующего сообщения, пока текущее зашифровывается или расшифровывается. Или же можно возложить обязанность перемешивать алфавиты на пользователя, который должен включать перемешанный алфавит в состав длинного ключа.

Если требуется синхронная работа, а параллельное перемешивание алфавитов по какой-то причине невозможно, то можно прибегнуть к запасному варианту: применить ИСКЛЮЧАЮЩЕЕ ИЛИ к блоку и ключу такого же размера до и после шага DES или AES. Я называю этот метод *XDESX* или *XAESX*. Операция ИСКЛЮЧАЮЩЕЕ ИЛИ выполняется очень быстро и заметно повышает безопасность. Полный размер ключа составляет 184 бита, на 16 бит больше, чем в 3DES. Я рекомендую дважды инвертировать конечный выход, чтобы замаскировать волновые формы.

## 11.7 Инволютивные шифры

*Инволютивный шифр* – просто научнообразный способ сказать, что «шифр является обратным самому себе». Иными словами, шифрование и дешифрирование производятся в точности одинаково. Если дважды зашифровать сообщение инволютивным шифром (с одним и тем же ключом), то получится исходный открытый текст. Инволютивные шифры еще называют *самообратными*. Мы уже встречали примеры инволютивных шифров. Применение ИСКЛЮЧАЮЩЕГО ИЛИ к открытому тексту и двоичному ключу – инволютивная операция (раздел 3.3). Перестановка кусочного обращения в шифре Базери типа 4 (раздел 4.6.1) тоже инволютивна. И транспонирование квадратной матрицы, т. е. запись символов в квадратную сетку слева направо и последующее считывание сверху вниз, – инволюция. Вот пример транспонирования матрицы 3×3:

До	После	
1 2 3	1 4 7	Транспонирование матрицы 3×3 Строки становятся столбцами, а столбцы – строками
4 5 6	2 5 8	
7 8 9	3 6 9	

Быстрый метод транспонирования матрицы описан в разделе 11.2.3.

Если вы собираетесь реализовывать свой шифр аппаратно, то использование инволютивных шифров может снизить стоимость оборудования и сложность эксплуатации. Наличие у шифровальной машины отдельных режимов шифрования и дешифрирования в таком случае необязательно.

Посмотрим, как строятся некоторые типы инволютивных шифров.

### 11.7.1 Инволютивная подстановка

Инволютивная подстановка характеризуется тем, что если буква  $X$  переходит в  $Y$ , то  $Y$  должна переходить в  $X$ . То есть буквы образуют пары. Для построения инволютивной подстановки сначала выпишем в ряд все буквы или символы. Выберем любую букву и ее пару. Вычеркнем их из списка. Затем выберем следующую букву и пару к ней. Вычеркнем их. Продолжаем, пока большинство букв не будут объединены в пары. Все оставшиеся буквы обратны сами себе. Последовательность выбора букв может управляться числовым ключом, как в методе SkipMix (раздел 5.2).

Любую инволютивную подстановку можно представить двумя рядами. Буквы верхнего ряда подставляются вместо стоящих под ними букв нижнего ряда, а буквы нижнего ряда – вместо стоящих над ними букв верхнего ряда. В примере ниже используются ключевые слова WORDGAME и TULIP. Буква R переходит в L, а L в R.

WORDGAMEBCFHJ
TULIPKNQSVXYZ

Иными словами, ключом этой инволютивной подстановки является приведенный выше массив с двумя строками.

Необязательно ставить в пару к каждой букве другую букву. Несколько букв подстановка может оставлять на месте. Они называются *инвариантами*, или *неподвижными точками*.

Инволютивную подстановку биграмм можно построить таким же способом.

### 11.7.2 Инволютивная многоалфавитная подстановка

Для построения инволютивного многоалфавитного шифра достаточно сделать каждую строку таблицы алфавитов инволютивной подстановкой.

### 11.7.3 Инволютивная перестановка

Инволютивные перестановки проще всего строить для сообщений, разбитых на блоки фиксированного размера. Обозначим размер такого блока  $B$ . Перестановка является инволютивной, если для каждой буквы, переходящей из позиции  $X$  в позицию  $Y$ , буква в позиции  $Y$  переходит в позицию  $X$ . Иными словами, перестановка образована попарными обменами букв.

Чтобы построить инволютивную перестановку, сначала выпишем в ряд числа от 1 до  $B$ . Выберем любые два числа. Это первая пара обмениваемых позиций. Удалим эти два числа из списка и выберем следующую пару. Ее тоже удалим. Продолжаем, пока в списке останется не более одного числа. Если хотите оставить в перестановке



несколько неподвижных точек, прекратите образование пар раньше. Другой способ создания неподвижных точек – выбирать из списка сразу два числа случайным образом. Если окажется, что они совпали, то эта точка становится неподвижной.

Один из способов представления общего перестановочного шифра – выписать в ряд все позиции в блоке, а под ними написать их новые позиции. Например:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
13	7	17	8	20	15	2	4	11	18	9	16	1	19	6	12	3	10	14	5

Это наилучший формат для компьютера. Если перестановку выполняет человек, то может быть удобнее наполовину сократить запись:

1	2	3	4	5	6	9	10	12	14
13	7	17	8	20	15	11	18	16	19

Это та же самая перестановка, но занимающая в два раза меньше места. Любую форму можно использовать в качестве ключа перестановки. В обоих случаях буква в позиции 1 перемещается в позицию 13, а буква в позиции 13 – в позицию 1. Вторая буква перемещается на седьмое место, а ее прежнее место занимает седьмая. И так далее.

### \*11.7.4 Инволютивный блочный шифр

Поняв, как строить инволютивные подстановки и перестановки, мы можем собрать все элементы вместе и построить инволютивный блочный шифр.

Будет удобно ввести обозначения. Пусть  $M$  – любое сообщение, открытое или зашифрованное. Будем обозначать  $CM$  применение шифра  $C$  к сообщению  $M$ . Если  $D$  – другой шифр, то применение  $D$  к тексту  $CM$  будем обозначать  $DCM$ . Нотация выглядит немного странно, потому что  $DCM$  означает, что сначала применяется  $C$ , а потом  $D$ , но она удобна. Можете считать, что  $DCM$  – сокращенная запись  $D(C(M))$ .

Тогда  $DC$  – шифр, получающийся применением к сообщению сначала  $C$ , а потом  $D$ . Этот новый шифр называется *композицией*  $D$  и  $C$ . Операция композиции строит новый шифр из двух других. (Некоторые авторы называют ее результат *произведением* шифров  $C$  и  $D$  и обозначают  $C \circ D$ .)

Например, шифр Базери типа 4 (раздел 4.6.1) объединяет подстановку с перестановкой. Композиция обладает математическим свойством, важным для построения инволютивных шифров: она ассоциативна. Это означает, что если  $A$ ,  $B$  и  $C$  – шифры, то  $(AB)C = A(BC)$ . Благодаря этому свойству композицию нескольких шифров можно записывать без скобок, например  $ABC$  или даже  $ABCDEFGH$ .



Скобки можно было бы расставить любым способом, результат от этого не изменится. Например, ABCDEFGH можно было бы записать в виде  $A((BC)(DE))F(GH)$ .

Обозначим  $I$  тождественный шифр, преобразующий любой открытый текст в себя. То есть  $IM = M$  для любого сообщения  $M$ . Пусть  $C$  – произвольный шифр. Обозначим его обращение  $C'$ . ( $C$  должен быть обратим, иначе сообщения нельзя было бы прочитать.) Тогда  $CC' = C'C = I$ . Шифр  $C$  называется инволютивным, если  $C = C'$ .

Предположим, что  $T$  – инволютивный шифр, и пусть  $C$  – любой шифр. Тогда шифр  $CTC'$  инволютивный. Действительно:

$$(CTC')(CTC') = CTC'CTC' = CT(C'C)TC' = CTTC' = C(TT)C' = CC' = I.$$

Аналогично, если  $A$  и  $B$  – произвольные шифры, то  $BCTC'B'$  и  $ABCTC'B'A'$  – инволютивные шифры. И так далее.

### 11.7.5 Пример – шифр Poly Triple Flip

Рассмотрим пример инволютивного блочного шифра, который я назову *Poly Triple Flip*. Он применяется к 64-битовым блокам и имеет вид  $ABCTC'B'A'$ , где  $A$  и  $C$  – многоалфавитные шифры общего вида,  $B$  – столбцовая перестановка 64 бит, а  $T$  – операция транспонирования 64-битовой квадратной матрицы.

Шифры  $A$  и  $C$  многоалфавитные с периодом 8. Это значит, что для шифрования каждой строки применяется свой алфавит. Таблица алфавитов для каждого шифра будет состоять из восьми строк, используемых по очереди. Ключей для выбора строк из таблицы не предусмотрено. Вместо них есть 8 ключей для перемешивания восьми алфавитов. Вместе  $A$  и  $C$  требуют 16 различных ключей, каждый из которых может быть последовательностью чисел, рассчитанных на применение алгоритма SkipMix (раздел 5.2). Я рекомендую включать в эти ключи от 3 до 8 чисел в диапазоне от 0 до 255.

Шифр  $B$  – столбцовая перестановка, при которой 64-битовый блок рассматривается как сетка  $4 \times 16$ , так что число возможных перестановок столбцов равно  $16!$ . 64 бита записываются в сетку по строкам слева направо, а читаются по столбцам сверху вниз. Порядок столбцов определяется ключевым словом или ключевой фразой либо эквивалентной строкой из 16 чисел.

Шифр Poly Triple Flip получает оценку 10. \*\*

## 11.8 Подстановки переменной длины

В состав блочных шифров могут входить подстановки фиксированной или переменной длины. *VLA* и *VLB* – примеры блочных шифров с подстановкой переменной длины. Оба они применяются к 128-би-

товым блокам, рассматриваемым как 4 строки по 32 бита. Идея в том, чтобы использовать подстановку переменной длины в строках, а затем перемешать блок с помощью 4-битовых подстановок столбцов. Ключами для каждого шифра служат ключи перемешивания набора тагов и 4-битовая подстановка.

В шифрах VLA и VLB используется подстановка тагов Поста, которая была описана в разделе 10.5.1. Таким образом, 4-битовый таг заменяется 4-битовой подстановкой, 5-битовый таг – 5-битовой подстановкой и т. д. Поэтому длина каждой строки блока остается равной 32 битам.

После каждой подстановки новый таг перемещается в конец строки и строка сдвигается влево, чтобы остаться на границе 4 байт. Средняя длина тагов должна быть не менее 6 бит.

VLA – более простой вариант шифра. В каждом раунде сначала выполняется 4-битовая подстановка левых (старших) битов в строке. Затем в каждой строке выполняется одна подстановка тагов Поста, со сдвигом. Эти действия повторяются на протяжении 32 раундов. Весь процесс шифрования состоит из 129 подстановок переменной длины и 32 четырехбитовых подстановок фиксированной длины. Этот шифр получает оценку 8.

При средней длине тага 6 бит я рекомендую шифр VLB с 4 раундами. Каждый раунд должен включать 6 шагов подстановки в первой строке, 7 шагов подстановки во второй строке, 8 шагов подстановки в третьей строке и 9 в четвертой.

Вертикальные подстановки в столбцах должны выполняться после раундов 1, 2 и 3. Для быстроты необязательно выполнять подстановки в каждом столбце в каждом раунде. Разумный выбор – производить подстановки в каждом третьем столбце, например в столбцах 1, 4, 7, ..., 31 после раунда 1, в столбцах 2, 5, 8, ..., 32 после раунда 2 и столбцах 3, 6, 9, ..., 30 после раунда 3.

VLB получает оценку 10 и, пожалуй, является самым быстрым шифром с такой оценкой. Он требует 120 подстановок переменной длины со сдвигами и 32 вертикальных 4-битовых подстановок, поэтому немного быстрее, чем VLA.

## 11.9 Пульсирующие шифры

*Пульсирующие шифры* (англ. *ripple cipher*), называемые также *циклическими* (англ. *wraparound* или *end-around*), – это блочные шифры, основанные на совершенно другом принципе. Основная идея заключается в том, что каждый 8-битовый символ блока используется как ключ для шифрования следующего за ним символа. Тот, в свою очередь, используется для шифрования своего правого соседа и т. д. По достижении конца блока происходит возврат в начало, т. е. последний символ блока является ключом для шифрования первого.

Пульсирующие шифры больше подходят для программной реализации, потому что практически не поддаются распараллеливанию.

Типов пульсирующих шифров много. Длина блока может варьироваться от двух и выше и изменяться периодически или случайным образом. Я рекомендую минимальную длину блока 5 символов, но, возможно, вы предпочтете начать с 8. Для выбора длин блоков можно использовать цепной генератор цифр. Если генератор выдает цифру  $D$ , то делаем длину следующего блока или  $D + 8$ , или даже  $20 - D$ .

Блоки могут перекрываться. Например, можно было бы использовать блоки фиксированной длины 8, начинающиеся в позициях 1, 6, 11, 16, ..., т. е. через каждые 5 символов. Если для последнего блока не хватает символов, то он заворачивает в начало. При длине сообщения 20 последний блок состоял бы из символов в позициях 16, 17, 18, 19, 20, 1, 2, 3.

Пульсирующие шифры чисто подстановочные, в них вообще нет перестановок. В простейшей форме пульсирующий шифр сводится к применению ИСКЛЮЧАЮЩЕГО ИЛИ к текущему и следующему за ним символу, т. е.  $x_n = x_{n-1} \oplus x_n$ ,  $x_{n+1} = x_n \oplus x_{n+1}$  и т. д., так что волна пульсации пробегает по всему блоку.

Существует много способов использовать предыдущий символ для шифрования следующего. Перечислим лишь некоторые. Ниже  $A$ ,  $B$ ,  $C$  – простые подстановочные шифры,  $P$  – многоалфавитный шифр общего вида.  $A(x)$ ,  $B(x)$  и  $C(x)$  обозначают результат шифрования символа  $x$  шифрами  $A$ ,  $B$  и  $C$  соответственно, а  $P(k, x)$  – результат шифрования  $x$  шифром  $P$  с ключом выбора строки из таблицы алфавитов  $k$ .

<b>xor</b>	ИСКЛЮЧАЮЩЕЕ ИЛИ	$x_n = x_{n-1} \oplus x_n$
<b>sxor</b>	Подстановка, затем ИСКЛЮЧАЮЩЕЕ ИЛИ	Имеется три варианта: $x_n = A(x_{n-1}) \oplus x_n$ , $x_n = x_{n-1} \oplus B(x_n)$ и $x_n = A(x_{n-1}) \oplus B(x_n)$
<b>xors</b>	ИСКЛЮЧАЮЩЕЕ ИЛИ, затем подстановка	$x_n = A(x_{n-1} \oplus x_n)$
<b>add</b>	Сложение	$x_n = x_{n-1} + x_n$ . Как всегда, сложение производится по модулю 256
<b>madd</b>	Умножить и сложить; также называется <i>линейной заменой</i>	$x_n = rx_{n-1} + x_n$ , или $x_{n-1} + qx_n$ , или $rx_{n-1} + qx_n$ , где $r$ – любое целое число, а $q$ – нечетное число. (Если размер алфавита отличен от 256, то $q$ должно быть взаимно простым с этим размером)
<b>sadd</b>	Подстановка, затем сложение	$x_n = A(x_{n-1}) + x_n$ , или $x_{n-1} + B(x_n)$ , или $A(x_{n-1}) + B(x_n)$
<b>adds</b>	Сложение, затем подстановка	$x_n = A(x_{n-1} + x_n)$
<b>poly</b>	Многоалфавитная подстановка общего вида	$x_n = P(x_{n-1}, x_n)$

Поскольку операции **xor** и **sxor** могут раскрывать информацию о своих операндах, я рекомендую использовать вместо них **xors**, тогда простая подстановка, выполняемая после ИСКЛЮЧАЮЩЕГО ИЛИ, маскирует волновые формы.

Отметим, что **madd** – частный случай **sadd**, поскольку  $rx_{n-1}$  – частный случай  $A(x_{n-1})$ . Преимущество **madd** в том, что не требуется этап предварительной инициализации для перемешивания алфавита подстановки. Аналогично отметим, что операция  $P(A(x_{n-1}), B(x_n))$  просто переставляет строки и столбцы таблицы алфавитов, поэтому она эквивалентна  $P(x_{n-1}, x_n)$ , только с другой таблицей.

Самый стойкий метод пульсации **poly**, когда предыдущий символ  $x_{n-1}$  используется как ключ для выбора строки таблицы алфавитов, применяемой для шифрования  $x_n$ . Я называю его *пульсацией с ключом* (Key Ripple). Для него требуется таблица алфавитов  $256 \times 256$  байт. Если это слишком много, то диапазон  $x_{n-1}$  можно уменьшить, применив к  $x_{n-1}$  *сужающую подстановку*, прежде чем использовать его в качестве ключа. Например,  $x$  можно было бы уменьшить до  $x \bmod 16$  или до  $(13x+5) \bmod 32$ . Подходящие уменьшенные диапазоны: 0–15, 0–31 и 0–63. Если  $R$  – сужающая подстановка, а  $P$  – многоалфавитная подстановка, то  $x_n$  заменяется на  $Q(R(x_{n-1}), x_n)$ , где  $Q$  – многоалфавитный шифр с уменьшенной таблицей алфавитов, которая состоит из 16, 32 или 64 строк таблицы  $P$ .

Если вы не можете использовать многоалфавитный шифр, быть может, потому что даже уменьшенная таблица занимает слишком много места или потому что время инициализации слишком велико, то следующее упрощение – использовать три простые подстановки. Замените  $x_n$  на  $A(B(x_{n-1}) + C(x_n))$  или  $A(B(x_{n-1}) \oplus C(x_n))$ . Это пример *компромисса между временем и памятью*. Три простые подстановки могут занять немного больше времени, чем одна многоалфавитная, зато потребление памяти сокращается с 65 536 до 768 байт, т. е. на 98.8 %.

Пульсирующие шифры не ограничиваются использованием только предыдущего символа для шифрования текущего. При желании можно отступить назад на несколько символов, например заменить  $x_n$  на  $A(x_{n-i} \oplus x_n)$ , где  $i$  – любое число, меньшее размера блока. Можно также использовать несколько предыдущих символов, например  $x_{n-2} + x_{n-1} + x_n$  или, более общо,  $x_{n-j} + x_{n-k} + x_n$ . Прибегнув к многоалфавитным подстановкам общего вида, мы можем заменить  $x_n$  на  $P(x_{n-4} \oplus x_{n-2}, x_n)$  или  $P(x_{n-5}, P(x_{n-1}, x_n))$ . Число комбинаций не ограничено.

Я уже говорил, что размер блока может быть любым. Подстановки могут начинаться и заканчиваться в любом месте блока, при условии что каждый символ подставляется хотя бы один раз. При желании можно выполнить несколько проходов по блоку, переходя от последнего символа к первому, если это необходимо. Возможно даже перекрытие более двух блоков или размещение одного блока целиком внутри другого – одного или нескольких. Размеры блоков, начальные положения внутри блока, число подставляемых символов и перекрытие с предыдущим и (или) последующим блоком могут быть фиксированы, изменяться периодически или порождаться генератором случайных чисел.

Можно пойти еще дальше. Сообщение можно шифровать, применяя несколько раундов пульсирующего шифра. В каждом раунде сообщение можно разбивать на блоки разных размеров, так что совпадение границ блоков будет происходить нечасто или вообще никогда, а шифрование – начинать и заканчивать в разных местах блоков. Это дает мозаичный или даже калейдоскопический эффект.

Вариаций пульсирующих шифров так много, что все и не перечислить. Их оценки изменяются от 4 до 10. Приведу несколько примеров. Простая операция **xor** с блоками фиксированного размера, с двумя раундами подстановок, начинающихся первым и заканчивающихся последним байтом, с использованием только предыдущего байта в качестве ключа подстановки получает оценку 4. Пульсирующий шифр **sadd** с блоками переменного размера, по меньшей мере тремя раундами подстановок, начинающихся и заканчивающихся в переменной позиции блока, с использованием предыдущего байта в качестве ключа подстановки получает оценку 7. Пульсирующий шифр **poly** с блоками переменного размера, по меньшей мере тремя раундами подстановок, начинающихся и заканчивающихся в переменной позиции блока, с использованием предыдущего и еще одного байта, меняющегося от блока к блоку, в качестве ключа подстановки получает оценку 10. Мозаичные методы более стойки, чем одноуровневые.

## 11.10 Сцепление блоков

Сцепление блоков – ценный инструмент укрепления любого блочного шифра. Под сцеплением понимается, что каждый блок принимает участие в шифровании следующего. По сути дела, сцепление – это пульсирующий шифр, применяемый к блокам, а не отдельным символам. Группа байтов, переносимых из блока N в блок N+1, называется *вектором сцепления*. Поскольку у первого блока сообщения нет предшественника, в большинстве схем сцепления используется *вектор инициализации* (IV) для шифрования первого блока, играющий роль вектора сцепления с воображаемым предшествующим блоком. Вектор инициализации можно вывести из ключа шифрования или рассматривать как дополнительный ключ.

**Отступление.** Сцепление блоков, используемых в криптовалюте Bitcoin и других, – специализированная форма сцепления блоков, применяемого в криптографии. Именно отсюда авторы позаимствовали идею.



Самая распространенная форма сцепления – посимвольное комбинирование вектора сцепления со следующим блоком. А самый распространенный способ комбинирования символов – ИСКЛЮЧАЮЩЕЕ ИЛИ. Однако можно использовать все способы комбинирования, описанные в разделе 11.8. Обычно применяется один из четырех режимов.

Режим	Описание
<b>РР</b>	Открытый текст блока N комбинируется с открытым текстом блока N+1 до шифрования блока N+1
<b>РС</b>	Открытый текст блока N комбинируется с шифртекстом блока N+1 после шифрования блока N+1
<b>СР</b>	Шифртекст блока N комбинируется с открытым текстом блока N+1 до шифрования блока N+1
<b>СС</b>	Шифртекст блока N комбинируется с шифртекстом блока N+1 после шифрования блока N+1

Для большей стойкости операция сцепления должна быть кумулятивной. Сначала вектор сцепления из блока N–1 комбинируется с блоком N. Результат становится новым вектором сцепления, который комбинируется с блоком N+1. Режим сцепления **РР** самый стойкий, вслед за ним идет режим **РС**. Режимы **СР** и **СС** значительно слабее, потому что Эмили может видеть вектор сцепления. Я рекомендую использовать режимы **СР** и **СС** только с комбинирующими функциями **xors**, **adds** и **poly**.

Хотя режимы **СС** и **СР** слабее, у них все же есть преимущества. В режимах **СС** и **СР** необязательно заводить отдельный вектор инициализации. Сандра может использовать в этом качестве последний блок открытого текста. А Рива просто начнет с дешифрирования последнего блока. На самом деле Рива может дешифрировать любой блок, не дешифрируя предыдущий. Это бывает полезно, когда в шифре используются индикаторы (раздел 14.3). Риве нужно будет дешифрировать индикаторный блок первым.

Рассмотрим некоторые более стойкие режимы сцепления блоков.

### 11.10.1 Многоалфавитное сцепление

ИСКЛЮЧАЮЩЕЕ ИЛИ – слабый способ комбинирования блоков N и N+1. Лучше брать операцию **xors**, т. е. сначала применить ИСКЛЮЧАЮЩЕЕ ИЛИ, а затем к результирующим символам применить простую подстановку. Еще лучше **poly**, многоалфавитный шифр общего вида. Каждый символ вектора сцепления используется в качестве ключа, выбирающего строку таблицы алфавитов для шифрования соответствующего символа в блоке N+1. Можно использовать любой из четырех режимов сцепления, самый стойкий **РР**.

### 11.10.2 Зашифрованное сцепление

В стандартных режимах сцепления используется либо открытый текст, либо шифртекст блока  $N$  в качестве вектора сцепления, без каких-либо модификаций. Шифр будет гораздо более стойким, если применить к вектору сцепления какое-то шифрование. Оно может быть рудиментарным, например простая подстановка или кусочное обращение (раздел 4.6). Такие простые методы могут оказаться эффективными, если для каждого блока используется своя подстановка или перестановка. Для этой цели хорошо подходит пульсация с ключом (раздел 11.8). У шифрования вектора сцепления должен быть независимый ключ. Если вектор сцепления зашифрован лучше, то режимы **СС** и **СР** уже не являются слабыми.

### 11.10.3 Сцепление с запаздыванием

Сцепление необязательно должно быть ограничено только предыдущим блоком. Можно использовать и более ранние блоки. Блок  $N$  можно комбинировать с блоком  $N-i$  или с несколькими предыдущими блоками, скажем  $N-i$  и  $N-j$ . Если  $i > j$ , то потребуется вектор инициализации, состоящий из  $i$  блоков.

Сам вектор сцепления может занимать несколько предыдущих блоков. Например, он может состоять из последней половины блока  $N-2$  и первой половины блока  $N-1$ .

### 11.10.4 Внутренние отводы

Слабость использования открытого или шифртекста в качестве вектора сцепления заключается в том, что они известны или могут стать известны Эмили. Одно из решений проблемы – шифровать вектор сцепления, как описано в разделе 11.10.2. Другое решение – взять вектор сцепления с какого-то промежуточного раунда шифрования блока. Это называется *отводом* (tap). Например, если блочный шифр включает 10 раундов, то можно было бы использовать выход пятого раунда в качестве вектора сцепления и комбинировать его с открытым текстом следующего блока до начала его шифрования. Этот режим называется **IP**.

Можно сделать еще один шаг – использовать несколько отводов и комбинировать их со следующим блоком в нескольких местах: с открытым текстом, с шифртекстом или между раундами шифрования. Каждый отвод порождает отдельный вектор сцепления, так что для  $N$  отводов должно быть  $N$  векторов инициализации. Все или некоторые из этих векторов сцепления можно зашифровать. Можно использовать один общий ключ шифрования или независимые ключи для каждого вектора. Для шифрования векторов сцепления хорошо подходит пульсирующий шифр (раздел 11.8).



### 11.10.5 Сцепление ключей

Обычно сцепление применяется к тексту каждого блока. Но можно также сцеплять ключи. Пусть имеется блочный шифр, в котором все блоки шифруются одним и тем же ключом  $K$ . Такой шифр можно значительно укрепить, применяя для каждого блока разные ключи. И один из способов сделать это – сцепление. Первый блок мы шифруем ключом  $K$  (для сцепления ключей вектор инициализации необязателен). Затем второй блок шифруется ключом  $K \bullet P_1$ ,  $K \bullet C_1$  или  $K \bullet I_1$ , где знак  $\bullet$  представляет одну из комбинирующих функций, например **xors** или **adds**. Аналогично третий блок шифруется ключом  $K \bullet P_2$ ,  $K \bullet C_2$  или  $K \bullet I_2$  и т. д. Это дает три новых режима сцепления: **РК**, **СК** и **ИК**. Можно одновременно использовать режимы сцепления ключей и блоков, например **РК** и **ИР**. Это исключительно эффективная комбинация.

### 11.10.6 Сводка режимов сцепления

Всего существует 12 режимов сцепления. Вектор сцепления можно брать из трех источников: открытый текст, внутренний шаг или шифртекст текущего блока. Вектор сцепления можно комбинировать с любым из четырех целевых объектов: ключом, открытым текстом, внутренним шагом или шифртекстом следующего блока.

Кроме того, вектор сцепления можно каждый раз брать заново или комбинировать с вектором сцепления от предыдущего блока. Его можно использовать в исходном виде или зашифровать до комбинирования с целевым объектом. Вектор сцепления может применяться к последовательным блокам или с запаздыванием. Вариантов множество.

### 11.10.7 Сцепление с неполными блоками

Если последний блок сообщения неполный и для их обработки используется метод перекрытия (раздел 11.2.4), то не ясно, как сцеплять с перекрытым блоком. Решение – отступить на два блока назад. Если в сообщении  $N$  блоков, то векторы сцепления из блока  $N-2$  используются для обоих блоков  $N-1$  и  $N$ .

### 11.10.8 Сцепление блоков переменной длины

Осталось рассмотреть последний вопрос, перед тем как оставить тему сцепления блоков, а именно блоки переменной длины. Я рекомендую делать длину вектора сцепления фиксированной. Если длина блока сообщения  $L$  меньше длины вектора сцепления, то комбинируйте первые  $L$  байт вектора с блоком сообщения. Замените эти  $L$  байт вектора сцепления, а остальные не трогайте. Например, если вектор сцепления равен **1234567890**, а блок – **SAMPLE**, то комбинируем



**123456** с **SAMPLE**. Если в результате получается  $ZQm''w+$ , то новый вектор сцепления будет равен  $ZQm''w+7890$ .

<b>1234567890</b>	Вектор сцепления от предыдущего блока
<b>SAMPLE</b>	Блок открытого текста
$ZQm''w+$	Блок шифртекста
$ZQm''w+7890$	Новый вектор сцепления

Если вектор сцепления короче, чем блок сообщения, то продолжим его, скопировав столько раз, сколько нужно. Например, если вектор сцепления равен **123456**, а блок – **CONVENTION**, то будем объединять **1234561234** с **CONVENTION**. Если в результате получается  $qA\&Vm!7^{\wedge}oS$ , то новым блоком становится  $qA\&Vm!7^{\wedge}oS$ , а вектором сцепления –  $qA\&Vm!$ .

<b>123456</b>	Вектор сцепления от предыдущего блока
<b>CONVENTION</b>	Блок открытого текста
$qA\&Vm!7^{\wedge}oS$	Блок шифртекста
$qA\&Vm!$	Новый вектор сцепления

В обоих случаях ни длина блока, ни длина вектора после сцепления не изменяются.

## 11.11 Укрепление блочного шифра

Если изначально имеется стойкий блочный шифр, то для его дальнейшего укрепления нужно приложить совсем немного усилий. Достаточно слегка зашифровать открытый текст перед применением блочного шифра и шифртекст после применения. Я называю этот прием техникой *сэндвича*, а дополнительные шаги – предшифрованием и постшифрованием. Ехидные читатели могут называть его *сэндвичем Рубина*. Говоря «слегка», я имею в виду простой шифр с одним одношаговым раундом, например простую подстановку или перестановку с ключом (раздел 7.6). Например, можно было бы использовать простую подстановку до блочного шифра и перестановку с ключом после или наоборот. Более стойкий и быстрый способ – рассматривать первые 8 байт блока как два 32-битовых целых и умножить каждое из них на нечетное число в диапазоне от 3 до  $2^{32}-1$  по модулю  $2^{32}$ .

Поскольку блочный шифр уже стойкий, основная цель дополнительных шагов – увеличить общую длину ключа, чтобы помешать атакам полным перебором и со встречей посередине. Лучшее всего эта идея работает, когда ключи пред- и постшифрования длинные и не зависят от ключа блочного шифра. Например, если пред- или постшифрование – простая подстановка, то длинный ключ можно было сгенерировать алгоритмом SkipMix.

Практический пример – в шифре DES использовался короткий 56-битовый ключ. Если бы мы добавили шаги пред- и постшифрования в виде простой подстановки, каждый с 64-битовым ключом перемешивания, то общую длину ключа удалось бы довести до 184 бит. Это более стойкий шифр, чем 3DES, и работает в 3 раза быстрее.

Однако при проектировании DES фаза инициализации не закладывалась. Предшифрование легко выполнить и без инициализации – достаточно применить ИСКЛЮЧАЮЩЕЕ ИЛИ к ключу предшифрования и открытому тексту. Тогда общий размер ключа увеличится с 56 до 120 бит. Уже в таком виде шифр более стойкий, чем 2DES, и более устойчив к атаке со встречей посередине. Шаг постшифрования немного сложнее. Мы хотим избежать ИСКЛЮЧАЮЩЕГО ИЛИ на последнем шаге по причинам, которые обсуждались выше, и одновременно не хотим никакой инициализации. Задачу можно решить с помощью фиксированного многоалфавитного шифра. То есть таблица алфавитов выбирается заранее и встраивается в устройство или программу.

Возможный вариант – использовать таблицу алфавитов  $16 \times 16$ , состоящую из 4-битовых групп. 64-битовый блок рассматривается как шестнадцать 4-битовых групп. Каждая 4-битовая группа шифруется с помощью четырех бит 64-битового ключа постшифрования. Таким образом, полный размер ключа снова равен 184 битам. Это лучше, чем 3DES, и примерно в 3 раза быстрее.

Работает это потому, что DES – сам по себе достаточно стойкий шифр, и единственная практически реализуемая атака – полный перебор. Увеличение ключа на 128 бит делает такую атаку неосуществимой.

# 12

## Принципы безопасного шифрования

---

### **Краткое содержание главы:**

- пять принципов безопасного шифрования;
- большие блоки и длинные ключи;
- конфузия, или нелинейность;
- диффузия, или насыщение.

Соберем воедино все, что мы узнали в главе 11. В разделах 12.1–12.5 мы сформулируем пять принципов, обеспечивающих безопасность блочного шифра. Главный признак безопасного шифра таков: изменение одного любого бита ключа или открытого текста должно приводить к изменению примерно половины битов шифртекста и желательно без какой-либо прослеживаемой закономерности. Изменение любого другого бита тоже должно приводить к изменению примерно половины битов шифртекста, но совершенно других. Назовем это свойством *пятьдесят на пятьдесят*. В этой главе описывается, как достичь поставленной цели.

### **12.1 Большие блоки**

Мы видели, что биграммный шифр можно вскрыть как простой подстановочный шифр, составив таблицу частот биграмм и контактов.

Это возможно также для триграмм и тетраграмм, хотя понадобятся очень большие объемы шифртекста. Для блочного шифрования вручную размер блока не должен быть меньше 5 символов. Для компьютерных шифров минимальный размер блока – 8 байт. Большие блоки нужны, в частности, для того, чтобы шифр нельзя было вскрыть как код, когда Эмили находит повторяющиеся блоки шифртекста и делает выводы об их смысле, исходя из частоты и позиций в сообщении. В предельном случае, когда блок состоит всего из одного символа, шифр является простой подстановкой вне зависимости от длины ключа и числа шагов шифрования.

В английском языке существует много расхожих 8-символьных последовательностей, которые с большой вероятностью повторяются в длинном сообщении. Ниже приведено двенадцать примеров, в которых многоточие обозначает пробел.

...AND...THE	THAT...ARE	WHICH...IS
FOR...THE...	THERE...IS	WHO...WERE
FROM...THE	THEY...ARE	...WILL...BE
IT...WILL...	...OF...THE...	WITH...THE

В настоящее время стандартная длина блока – 16 байт. В английском языке нет высокочастотных фраз такой длины. В конкретных контекстах встречаются длинные фразы, например: UNITED STATES GOVERNMENT (правительство Соединенных Штатов), EXECUTIVE COMMITTEE (исполнительный комитет), INTERNATIONAL WATERS (международные воды) и т. д. Но для порождения повторяющихся блоков шифртекста эти фрагменты открытого текста должны быть выровнены на границы блоков. Например, 16-байтовые блоки открытого текста UNITED...STATES...GO и NITED...STATES...GOV не порождают похожих блоков шифртекста, если используется стойкий блочный шифр.

Проблемы повторяющихся блоков шифртекста не возникает, если используется сцепление блоков (раздел 11.9). В этом случае годятся размеры блоков от 8 байт и выше.

## 12.2 Длинные ключи

Мы знаем, что безопасный шифр должен иметь длинный ключ для предотвращения атаки полным перебором. В настоящее время стандартными считаются 128-битовые ключи. Если нужно сохранить секретность сообщений в течение 20 и более лет, то я рекомендую использовать ключ не короче 160 бит. Это приблизительно 48 десятичных цифр, 40 шестнадцатеричных или 34 буквы в одном регистре.

Если вы вводите ключ вручную, то я рекомендую определить структуру ключа. Разбейте ключ на блоки равного размера и единого

формата. Ниже приведены примеры двух способов форматирования ключа. В первом случае все символы в каждом блоке одного типа: строчные буквы, заглавные буквы или цифры. Во втором случае все блоки имеют одинаковый формат: 2 заглавные буквы и 3 цифры.

18682 dcmpr KV0WZ 96583 русмх 70584 GDNLS gsbif ZNEJR  
BF242 KG679 UX591 WB485 DT649 MH537 PS506 CK841 HI458

Первый ключ эквивалентен примерно 191 биту, а второй 74 битами. При вводе таких длинных ключей важно видеть, что печатается, чтобы можно было найти и исправить ошибки. По завершении ввода приложение должно отображать контрольную сумму, позволяющую удостовериться в правильности ключа.

Преимущество такой регулярности в том, например, что вы не примете букву O за цифру 0 или букву I за цифру 1. Я не рекомендую смешивать символы разных типов, например \$v94H;t}=Nd^8, потому что это ведет к ошибкам. Если файл данных был зашифрован ключом \$v94H;t}=Nd^8, а дешифровать его вы пытаетесь ключом \$V94H;t}=Nd^8, то ничего не получится, и вы даже не узнаете, в чем ошибка и как ее исправить. Восстановить данные из файла не удастся. Запись ключей в виде регулярных блоков поможет избежать такой беды. Другая форма ключа, позволяющая предотвратить ошибки ввода, – искусственные слова. Придумайте какую-нибудь легкопроизносимую комбинацию букв, например:

obel ipsag lokitar malabak zendug foritut glapmar

Старайтесь избегать закономерностей, например использования комбинаций с одинаковыми гласными типа **palek mafner vadel glabet** (во всех словах наблюдается один и тот же паттерн гласных A-E).

Такие буквенно-цифровые ключи легко программно преобразовать в двоичную форму. Для этой цели очень удобен пульсирующий шифр **madd** (раздел 11.8).

Альтернатива ручному вводу ключевого слова для каждого сообщения или файла данных – *менеджер ключевых слов*, который генерирует ключевые слова и ассоциирует их с сообщениями или файлами. Такой менеджер можно установить на сайте, доступном Сандре и Риве. Но эту тему мы здесь рассматривать не будем. Отметим лишь, что менеджер ключевых слов – не то же самое, что менеджер паролей, потому что Сандра и Рива должны использовать для каждого файла одно и то же ключевое слово, хотя работают на разных компьютерах.

## 12.2.1 Избыточные ключи

В некоторых случаях Эмили способна вывести уравнения, связывающие шифртекст с открытым текстом и ключом. Если Эмили знает или может угадать часть открытого текста, то из этих уравнений,

возможно, удастся определить ключ. Например, она может знать, что некоторые сообщения начинаются словом ATTENTION, написанным заглавными буквами. Этого может оказаться достаточно для определения 64-битового ключа, если используются 8-байтовые блоки.

Один из способов противостоять такой атаке – увеличить длину ключа. Например, если размер блока – 64 бита, а ключ на 32 бита длиннее, т. е. состоит из 96 бит, то можно ожидать, что в среднем существует  $2^{32}$  возможных ключей, преобразующих известный открытый текст в шифртекст. Эмили должна будет проанализировать все  $2^{32}$  решений и найти среди них правильное. Это может оказаться трудной задачей, потому многие из четырех с лишним миллиардов потенциальных ключей могут давать правдоподобные тексты.

Увеличение длины ключа значительно затрудняет задачу Эмили, но не делает ее неразрешимой. Если она располагает вдвое большим объемом известного открытого текста, то для нахождения ключа имеются уравнения, построенные по двум шифртекстам. Впрочем, так много известного открытого текста удастся получить редко, а время, необходимое для решения  $N$  и  $2N$  уравнений, отличается не в два раза, а гораздо больше. В зависимости от типа уравнений Эмили, возможно, сможет решить систему из 64 уравнений, но не из 128.

Если у Эмили нет допускающих решение уравнений, то избыточные ключи все равно существенно затрудняют и удорожают атаку полным перебором. В любом случае они усложняют работу Эмили.

## 12.3 Конфузия

В 1945 году Клод Шеннон, основоположник теории информации, сформулировал два свойства, которыми должен обладать стойкий шифр. Он назвал их *конфузия* и *диффузия*. Под конфузией Шеннон понимал отсутствие сильной корреляции между открытым и шифртекстом. Точно так же не должно быть сильной корреляции между ключом и шифртекстом. Говоря о диффузии, Шеннон имел в виду, что любая часть шифртекста должна зависеть от каждой части открытого текста и каждой части ключа.

Я добавлю еще и третье свойство – *насыщение*. Идея в том, чтобы измерить, насколько сильно каждый бит или байт шифртекста зависит от каждого бита или байта открытого текста и ключа. Чем больше насыщение, тем выше стойкость шифра. В этом и двух последующих разделах мы обсудим все три свойства подробнее.

В блочных шифрах используется два типа подстановок: фиксированные и с ключом. Подстановки с ключом могут изменяться для каждого сообщения и даже для каждого блока. В разделе 11.6 мы об-

суждали плюсы и минусы обоих методов. Если вы решите использовать подстановку с ключом или если математика в этом разделе покажется слишком трудной, пропустите его и переходите к разделу 12.4. Построить собственный перемешанный алфавит или таблицу алфавитов можно с помощью алгоритма SkipMix, описанного в разделах 5.2 и 12.3.7, а последовательность пропусков – выбирать с помощью генератора псевдослучайных чисел.

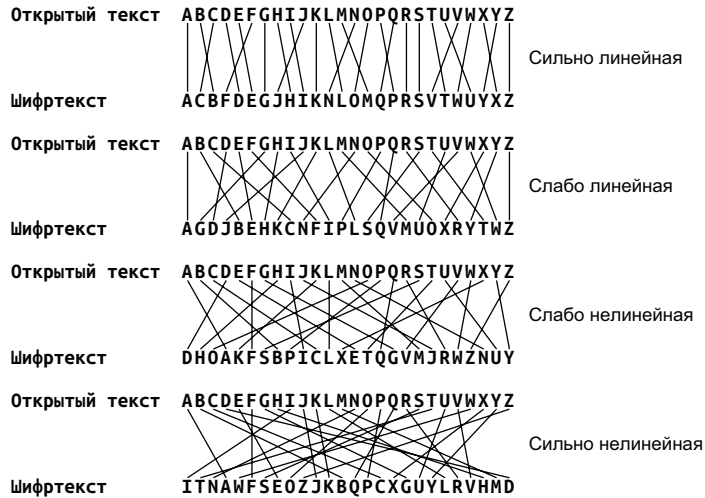
Конфузия в смысле Шеннона – это, по существу, вопрос о линейности и нелинейности. Если в блочном шифре используется фиксированный алфавит или таблица алфавитов, то линейность приобретает первостепенное значение. Термин *линейность* происходит из аналитической геометрии. Уравнение прямой линии имеет вид  $ax+by = c$ , где  $a$ ,  $b$ ,  $c$  – постоянные, а переменные  $x$  и  $y$  – декартовы координаты точки на прямой. Если прямая не параллельна оси  $y$ , то уравнение можно записать в виде  $y = ax+b$ . Обе формы,  $ax+by = c$  и  $y = ax+b$ , – примеры линейных уравнений, или линейных связей.

Шифр Цезаря (раздел 4.2) – пример линейного шифра. В нем шифртекст получается сложением ключа с открытым текстом,  $c = p+k$ . Здесь  $c$  – буква шифртекста,  $p$  – буква открытого текста, а  $k$  – ключ. Ключ определяет, на сколько букв сдвинут алфавит. Сам Юлий Цезарь пользовался сдвигом на 3 позиции, т. е. каждая буква заменялась отстоящей от нее на три позиции вправо,  $c = p+3$ , а по достижении конца алфавита производился переход в начало.

Кстати говоря, метод Цезаря совсем не такой слабый, как кажется, потому что Цезарь писал сообщения на греческом языке, пользуясь греческим алфавитом. Во времена Цезаря образованная элита римского общества, в частности Цезарь и его военачальники, знала греческий, как в XIX веке высший класс английского общества владел латынью, а русские аристократы свободно говорили по-французски.

Блочный шифр, включающий шаги подстановки и перестановки, в целом нелинеен, если нелинейны отдельные подстановки. На самом деле если блочный шифр включает несколько раундов подстановок, то он в целом нелинеен, если всего один из начальных раундов нелинеен, при условии что этот раунд затрагивает все элементы блока. Коль скоро линейность утрачена, ее нельзя восстановить в последующих раундах. Стойкость оказалась бы значительно выше, если бы каждый раунд был нелинеен, но даже один нелинейный раунд лучше, чем отсутствие таковых, особенно если он близок к началу шифра.

У линейности и нелинейности есть степени. Подстановка может быть сильно линейной, слабо линейной, слабо нелинейной и сильно нелинейной. Пример подстановки каждого типа прояснит ситуацию. Я соединил линиями позиции каждой буквы в исходном и перемешанном алфавитах. Видно, что чем сильнее нелинейность, тем лучше перемешан алфавит.



Далее я буду называть входы S-блока открытым текстом и ключом, а выход – шифртекстом. Но имеются в виду только открытый и шифртекст одного S-блока, а не всего многораундового блочного шифра. В некоторых блочных шифрах у S-блоков нет ключей, они выполняют простую подстановку. В таком случае можно считать, что S-блок имеет постоянный ключ 0 или что длина ключа S-блока равна нулю.

Предполагается, что Эмили может проверить S-блок(и) на предмет линейности, потому что шифр опубликован или ей удалось заполучить шифровальное устройство. Если все, чем располагает Эмили, – вход первого и выход последнего раунда, то проверка на линейность может оказаться неосуществимой.

### 12.3.1 Коэффициент корреляции

Существует надежный статистический метод проверки корреляции между двумя числовыми величинами. Например, можно было бы проверить наличие корреляции между дневной температурой по Цельсию и продолжительностью светового дня в часах. И температура, и часы – числовые величины. Можно было бы провести несколько измерений температуры в одно и то же время суток и записать продолжительность светового дня на эту дату. В итоге мы получили бы два списка чисел: температуру и соответствующую ей продолжительность светового дня. Для измерения корреляции между этими списками следует применить статистику.

В нашем случае величинами являются буквы открытого и шифртекста. «Измерения» – это их позиции в алфавите. Например, результатом первого измерения могла бы быть буква А, а последнего – буква Z. Буквы алфавита нужно как-то занумеровать. Нумерация зависит от размера алфавита. Например, 27-буквенный алфавит



можно было бы занумеровать тремя троичными цифрами, как мы делали в шифре Trifid в разделе 9.9. Корреляция могла бы существовать между любой троичной цифрой букв открытого текста и любой троичной цифрой букв шифртекста. В двух следующих разделах мы детально рассмотрим этот вопрос для 26-буквенного и 256-символьного алфавитов.

Линейность измеряется корреляцией между двумя величинами. Самой распространенной мерой корреляции является коэффициент Пирсона, предложенный английским математиком Карлом Пирсоном, основоположником биометрии. Эта работа была опубликована в 1895 году, хотя саму формулу опубликовал в 1844 году французский физик Огюст Браве, известный своими работами по кристаллографии.

Цель коэффициента корреляции – описать одним числом степень корреляции двух величин, причем это число не должно зависеть от единиц измерения или размеров величин.

Если между двумя величинами имеется линейная зависимость, то корреляция равна 1. Если между ними нет вообще никакой зависимости, то корреляция равна 0. Если зависимость обратная, то корреляция равна  $-1$ . Например, между числом орлов и числом решек, выпавших при 20 подбрасываниях монеты, зависимость обратная. Корреляция 0.8 свидетельствует о сильной линейной зависимости, а корреляция 0.2 – о том, что зависимость сильно нелинейна.

Чем приводить формулу, как в большинстве учебников, я лучше объясню, как и почему она работает. Это поможет использовать ее правильно и по делу.

Наша цель – сравнить две величины. Для этого сравнивается последовательность значений в наборе испытаний. Например, можно сопоставить стоимость волшебных ковров на Исфаханском базаре в Персии с их размером. Цена волшебного ковра зависит от многих факторов, в т. ч. от типа пряжи, от плотности узелков, сложности узора и, разумеется, от скорости полета.

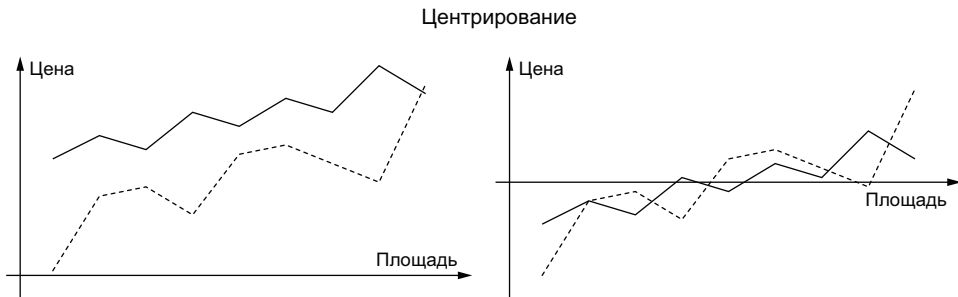
## ЦЕНТРИРОВАНИЕ

Первый шаг сопоставления величин – расположить их бок о бок, как если бы мы сравнивали их глазами. Иными словами, мы хотим исключить член  $+x$  из линейного соотношения  $P = mA + x$ , где  $P$  – цена, а  $A$  – площадь. На первый взгляд, достаточно было бы взять разности  $P_i - A_i$  и вычесть из  $P$  их среднее. Однако эта операция не имеет смысла, потому что  $P$  и  $A$  измеряются в разных единицах: площадь ковра – в квадратных *арсани* (приблизительно 1 метр), а цена ковра – в *туманах* (персидская валюта).

Из-за различных единиц измерения мы должны корректировать площади и цены независимо. Нужно взять среднюю цену и вычесть ее из всех цен, получив новые, скорректированные цены  $P'$ . Для вычисления средней цены  $\mu_p$  мы складываем цены на все ковры и де-

лим сумму на число ковров. Например, если цены равны 1000, 1200 и 1700 туманов, то в результате сложения и деления на 3 получаем среднюю цену 1300. Вычитаем 1300 из каждой цены и получаем скорректированные цены –300, –100 и 400. Как видите, сумма скорректированных цен равна нулю. В некотором смысле скорректированные цены центрированы относительно нуля.

Площади центрируются аналогично: складываем площади ковров и делим сумму на их количество – получаем среднюю площадь. Например, если площади равны 10, 12 и 17 квадратных арсани, то сумма площадей  $10+12+17$ , поделенная на 3, равна 13. Теперь вычитаем 13 из каждой площади и получаем скорректированные площади –3, –1 и 4. Сумма скорректированных площадей  $A'$  тоже равна нулю. Теперь скорректированные площади и цены центрированы относительно нуля и готовы к сравнению.



## МАСШТАБИРОВАНИЕ

Следующий шаг – привести цены и площади к общему масштабу. Цены выражены в туманах, площади – в квадратных арсани, а преобразования из туманов в квадратные арсани не существует. Это все равно, что переводить бушели в градусы Цельсия. Пирсон, а точнее Браве, воспользовались идеей *нормировки*, заимствованной из линейной алгебры.

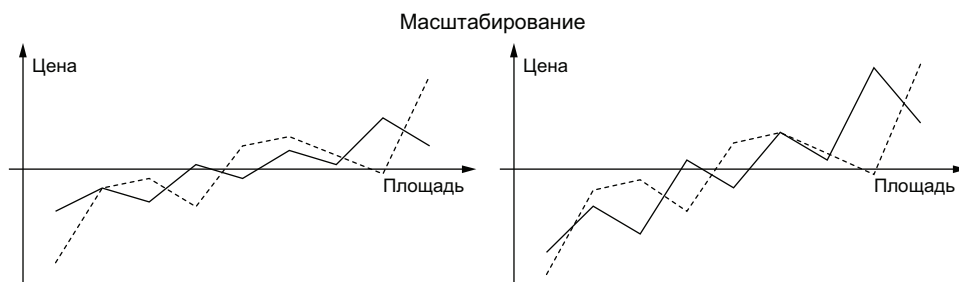
Предположим, что имеется вектор  $(a, b)$  и мы хотим найти вектор, направленный так же, но единичной длины. Любой вектор, кратный  $(a, b)$ , т. е. имеющий вид  $(ma, mb)$ , направлен так же. Умножение вектора на скаляр изменяет его длину, но не направление. Если поделить вектор на его длину, то новый вектор  $(a/L, b/L)$  будет иметь длину 1 и такое же направление, как у исходного. Заодно эта операция стирает информацию о единицах измерения. Допустим, что длина вектора измерена в метрах. Поделив вектор на его длину, мы поделили метры на метры. В результате получается просто безразмерное число. То же самое справедливо, когда вектор измеряется в туманах или квадратных арсани.

Длину вектора легко вычислить по теореме Пифагора  $L = \sqrt{a^2 + b^2}$ . Эта формула обобщается на любое число измерений,  $L = \sqrt{a^2 + b^2 + c^2 + \dots}$ .

Посмотрим, как это работает, на примере. Возьмем вектор (3,4). Его длина равна  $\sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$ . Нормированный вектор равен (3/5, 4/5). Таким образом, длина нормированного вектора равна  $\sqrt{(3/5)^2 + (4/5)^2} = \sqrt{9/25 + 16/25} = \sqrt{25/25} = 1$ , как и ожидалось. Все правильно.

$P$ ,  $A$ ,  $P'$  и  $A'$  – списки чисел, т. е. векторы. У них есть длины, как у любых векторов, и их можно нормировать. В геометрии вектор нормируется путем деления на длину. Длина любого нормированного вектора равна 1.

Чтобы нормировать  $P'$ , мы возводим в квадрат все скорректированные цены, вычисляем их сумму и извлекаем из суммы квадратный корень. Получается длина вектора  $P'$ . Делим скорректированные цены  $P'$  на эту длину и получаем нормированные цены  $P''$ . Аналогично вычисляем нормированные площади  $A''$ .



Итак, мы (1) центрируем цены и площади путем вычитания среднего, а затем (2) нормируем цены и площади путем деления на длину. В результате получаются стандартизованные списки цен и площадей такие, что сумма членов каждого списка равна 0, а сумма квадратов членов равна 1.

Вот теперь все готово для формулы. Умножаем каждый член нормированного списка цен на соответствующий член нормированного списка площадей, т. е. вычисляем произведения  $P_i'' \times A_i''$ . Складываем эти произведения. Результат называется коэффициентом корреляции. (В линейной алгебре это называется *скалярным*, или *внутренним*, *произведением* нормированного вектора цен и нормированного вектора площадей.)

Подвергнем эту идею реальной проверке. Допустим, что требуется проверить корреляцию между температурой по Цельсию и по Фаренгейту. Мы знаем, что эти величины связаны линейным соотношением  $F = 1.8C + 32$ , поэтому коэффициент корреляции должен быть равен 1. Предположим, что мы провели измерения в 11, 15, 19 и 23 часа и что температура по Цельсию была равна (14, 24, 6, 0), а по Фаренгейту – (57.2, 75.2, 42.8, 32). Средняя температура по Цельсию равна  $(14+24+6+0)/4 = 11$ , поэтому скорректированные температуры  $C'$  равны (3, 13, -5, -11). Соответственные скорректированные тем-

пературы по Фаренгейту  $F'$  равны (5.4, 23.4, -9, -19.8). Длина вектора  $C'$  равна 18. Поделив  $C'$  на 18, получим  $C''$ , вектор нормированных температур по Цельсию (3/18, 13/18, -5/18, -11/18). Длина вектора скорректированных температур по Фаренгейту  $F'$  равна 32.4, а вектор нормированных температур по Фаренгейту  $F''$  равен (3/18, 13/18, -5/18, -11/18).

Поэлементно умножаем  $C''$  на  $F''$  и складываем все четыре произведения, чтобы получить коэффициент корреляции. Сумма равна  $(3/18)^2 + (13/18)^2 + (-5/18)^2 + (-11/18)^2 = 1$ . Это подкрепляет утверждение о том, что описанная выше процедура – центрирование путем вычитания среднего, нормирование путем деления на длину и последующее суммирование попарных произведений – действительно дает коэффициент корреляции.

Подведем итог: для проверки на линейность следует вычислить коэффициент корреляции. В этом разделе показано, как это делается. Вычисление дает величину от -1 до +1. В таблице ниже приведена интерпретация коэффициента корреляции.

Сильно линейная	от 0.75 до 1.00 или от -0.75 до -1.00
Слабо линейная	от 0.50 до 0.74 или от -0.50 до -0.74
Слабо нелинейная	от 0.25 до 0.49 или от -0.25 до -0.49
Сильно нелинейная	от 0.00 до 0.24 или от -0.00 до -0.24

### 12.3.2 Линейность по основанию 26

Начнем исследование линейности с подстановок в 26-буквенном алфавите. Это может оказаться полезным при проектировании механического или электромеханического шифровального устройства или его моделировании. Каждый ротор такой машины выполняет подстановку в 26-буквенном алфавите. Для начала рассмотрим S-блок без ключа. В 26-буквенном алфавите возможно несколько форм линейности в зависимости от того, как пронумерованы буквы. Алфавит можно рассматривать тремя способами: как последовательность 26 букв, как массив букв размера  $2 \times 13$  или как массив букв размера  $13 \times 2$ . Соответственно, имеется три способа пронумеровать буквы, они показаны ниже и обозначены N1, N2, N3. При обсуждении этих схем нумерации используется модульная арифметика. Если хотите вспомнить, что это такое, перечитайте раздел 3.6.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
N1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
N2	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
N3	00	01	10	11	20	21	30	31	40	41	50	51	60	61	70	71	80	81	90	91	A0	A1	B0	B1	C0	C1

Схемы нумерации N2 и N3 следуют обычному соглашению об использовании букв A, B и C для представления цифр, больших 9. То есть используются первые 13 из 16 шестнадцатеричных цифр. В простейшем линейном шифре (шифре Беласо) ключ складыва-

ется с открытым текстом. При использовании в этом случае схемы нумерации N1 сложение производится по модулю 26. Если же используется схема N2, то первые цифры складываются по модулю 2, а вторые – по модулю 13. Наоборот, при использовании схемы N3 первые цифры складываются по модулю 13, а вторые – по модулю 2. В примерах ниже показано, как слово ТНЕ зашифровывается путем прибавления ключа J в каждой из трех схем.

N1			N2			N3			
Т	Н	Е	Т	Н	Е	Т	Н	Е	Открытый текст
19	7	4	16	07	04	91	31	20	Открытый текст в числовой форме
<u>+ 9</u>	<u>9</u>	<u>9</u>	<u>+09</u>	<u>09</u>	<u>09</u>	<u>+41</u>	<u>41</u>	<u>41</u>	Ключевая буква J
2	16	13	12	03	00	00	70	61	Шифртекст
С	Q	N	P	D	A	A	O	N	Шифртекст в символьной форме

Если алфавиты открытого текста, ключа и шифртекста пронумерованы с использованием схемы N1, то линейная подстановка, или линейное преобразование, принимает символ открытого текста  $p$  и преобразует его с использованием ключа  $k$  в символ шифртекста  $c = mp + f(k)$ , где  $m$  – множитель, взаимно простой с числом 26,  $f(k)$  – произвольная целочисленная функция, а арифметические операции выполняются по модулю 26. Например, если  $m = 5$ ,  $p = 10$ ,  $k = 3$  и  $f(k) = k^2 + 6$ , то  $c = 13$ , потому что  $5 \times 10 + 3^2 + 6 = 65 \equiv 13 \pmod{26}$ . Постоянная  $m$  и функция  $f(k)$  могут быть встроены в таблицу подстановки.

Если алфавиты открытого текста, ключа и шифртекста пронумерованы с использованием схемы N2, или  $2 \times 13$ , то либо первая, либо вторая, либо обе цифры могут быть линейными. Предположим, что линейны обе цифры. Тогда символ открытого текста  $p = a, b$  преобразуется ключом  $k$  в символ шифртекста  $c = ma + f(k), nb + g(k)$ , где  $m$  должно быть взаимно просто с 2, т. е.  $m = 1$ ,  $n$  взаимно просто с 13, а  $f(k)$  и  $g(k)$  – произвольные целочисленные функции. Арифметические операции выполняются по модулю 2 и 13 соответственно. Постоянные  $m, n$  и функции  $f(k), g(k)$  могут быть встроены в таблицу подстановки.

Если алфавиты открытого текста, ключа и шифртекста пронумерованы с использованием схемы N3, или  $13 \times 2$ , то либо первая, либо вторая, либо обе цифры могут быть линейными. Предположим, что линейны обе цифры. Тогда символ открытого текста  $p = a, b$  преобразуется ключом  $k$  в символ шифртекста  $c = ma + f(k), nb + g(k)$ , где  $m$  должно быть взаимно просто с 13,  $n$  взаимно просто с 2, т. е.  $n = 1$ , а  $f(k)$  и  $g(k)$  – произвольные целочисленные функции. Арифметические операции выполняются по модулю 13 и 2 соответственно. Постоянные  $m, n$  и функции  $f(k), g(k)$  могут быть встроены в таблицу подстановки.

Не требуется, чтобы открытый и шифртекст нумеровались одинаково. Корреляция может иметь место между любой цифрой от-

крытого текста и любой цифрой шифртекста при любой схеме нумерации. Эмили могла бы проверить все или некоторые комбинации, стремясь найти допускающую эксплуатацию слабость. Следовательно, проектировщик шифра обязан проверить все возможные схемы нумерации на предмет корреляции и убедиться, что таких слабостей не существует, или понять, какие контрмеры следует принять, чтобы помешать Эмили эксплуатировать имеющуюся слабость. Например, в чередующихся раундах блочного шифра можно использовать подстановки, имеющие разные слабости. В большинстве случаев одна подстановка нивелирует слабость другой. Конечно, это следует проверять, активно отыскивая линейные соотношения между открытым текстом и конечным шифртекстом, который формирует последний раунд.

Для проверки линейности подстановки нельзя применять коэффициент корреляции непосредственно, поскольку все подстановки производятся по модулю. Рассмотрим следующую подстановку с использованием схемы нумерации N1:

$p = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25$   
 $c = 0\ 2\ 4\ 6\ 8\ 10\ 12\ 14\ 16\ 18\ 20\ 22\ 24\ 1\ 3\ 5\ 7\ 9\ 11\ 13\ 15\ 17\ 19\ 21\ 23\ 25$

Здесь почти точно выполняется соотношение  $c = 2p$ , поэтому связь сильно линейна. Однако коэффициент корреляции между алфавитами открытого и шифртекста с такой схемой нумерации равен 0.55556, т. е. показывает, что подстановка всего лишь слабо линейна. При вычислении коэффициента корреляции нужно использовать следующее распределение, эквивалентное арифметике по модулю 26:

$p = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25$   
 $c = 0\ 2\ 4\ 6\ 8\ 10\ 12\ 14\ 16\ 18\ 20\ 22\ 24\ 27\ 29\ 31\ 33\ 35\ 37\ 39\ 41\ 43\ 45\ 47\ 49\ 51$

При такой схеме нумерации коэффициент корреляции равен 0.99987, что соответствует очень сильной линейности.

Этот пример иллюстрирует трудности использования коэффициента корреляции в криптографии. Мы всегда работаем по модулю размера алфавита. Чтобы найти правильную корреляцию, нужно прибавлять 26, затем 52, 78 и т. д. для схемы нумерации N1 или 13, 26, 39, ... для схем нумерации N2 и N3. В примере выше было очевидно, с какого места начать прибавление 26 – там, где нумерация шифртекста имеет вид **22 24 1 3**. Переход от 24 к 1 – ясная подсказка.

Но если алфавит шифртекста не настолько линейен, если в нем имеется небольшое количество скачков, то найти нужное место труднее. Например, следующая подстановка имеет коэффициент корреляции 0.3265, т. е. является умеренно нелинейной.

$p = 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16\ 17\ 18\ 19\ 20\ 21\ 22\ 23\ 24\ 25$   
 $c = 0\ 5\ 8\ 11\ 16\ 2\ 21\ 25\ 4\ 9\ 13\ 17\ 12\ 19\ 1\ 24\ 3\ 7\ 14\ 18\ 20\ 23\ 6\ 10\ 15\ 22$

Если исправить его, добавив кратные 26, как показано ниже:

p=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
c=	0	5	8	11	16	28	21	25	30	35	39	43	38	45	53	50	55	59	66	70	72	75	84	88	93	100

то коэффициент корреляции становится равен 0.9944, т. е. связь сильно линейная. Я использовал простое, двойное и жирное подчеркивание в тех местах, где к символам шифртекста прибавляется 26, 52 и 78 соответственно. Важно отметить, что 26 прибавляется к символу шифртекста 2, соответствующему символу открытого текста 5, но не прибавляется к следующим символам шифртекста, 21 и 25. Аналогично, 52 прибавляется к символу шифртекста 1, соответствующему символу открытого текста 14, но не прибавляется к следующему символу шифртекста, 24.

Определить, какое кратное 26 следует прибавлять, сравнительно легко, когда алфавит шифртекста близок к линейному. Если же он ведет себя особенно плохо, то задача существенно усложняется. Но... это не имеет значения. Если подстановка нелинейна, то больше нам ничего и не нужно знать. Не важно, чему равен коэффициент корреляции: 0.01 или 0.35. В любом случае корреляции недостаточно, чтобы Эмили могла ей успешно воспользоваться. Так что не тратьте время на вычисление точного значения.

Итак, мы рассмотрели случай отсутствия ключа. Теперь предположим, что ключ есть. Линейная подстановка будет иметь вид  $d(p) + f(k)$ , где  $p$  – открытый текст,  $k$  – ключ, а  $d$  и  $f$  – целочисленные функции. Сложение можно производить по любой из трех схем нумерации: N1, N2 или N3. В данном случае ключ не играет никакой роли при проверке линейности.  $f(k)$  – просто постоянная, прибавляемая к шифртексту. А прибавление постоянной не влияет на коэффициент корреляции, потому что она все равно вычитается, когда мы вычитаем среднее из каждого элемента списка (операция центрирования). Легко проверить, имеет ли подстановка  $S(k,p)$  вид  $d(p) + f(k)$ . Просто выберем два ключа  $k_1$  и  $k_2$  и вычислим разности  $S(k_1,0) - S(k_2,0)$ ,  $S(k_1,1) - S(k_2,1)$ ,  $S(k_1,2) - S(k_2,2)$ , .... Если S-блок имеет вид  $d(p) + f(k)$ , то все эти разности будут равны. Повторив эту проверку для всех возможных ключей, мы удостоверимся, что  $S(k,p)$  имеет нужную нам форму, и сможем выполнить проверку на линейность без участия ключа.

### 12.3.3 Линейность по основанию 256

Анализ линейности по основанию 26 – лишь подготовка к анализу по основанию 256, потому что в этом случае возможно два типа линейности. Назовем их *последовательной* и *конденсированной*. При последовательной линейности каждая группа битов представляет целое число. Например, 3-битовые группы 000, 001, 010, ..., 111 представляют числа 0, 1, 2, ..., 7. Обе формы линейности можно комбинировать для получения гибридной формы. Мы обсудим это в разделе 12.3.6.



Именно с последовательной формой линейности мы сталкивались при анализе основания 26. В этом случае могла иметь место корреляция между схемами нумерации N1, N2 и N3 в любой комбинации и в любом порядке, поэтому приходилось проверять на линейность много сочетаний. В случае основания 256 возможностей еще больше. Последовательная линейность может иметь место между любой группой битов в алфавите открытого текста и (или) ключа и любой группой битов в алфавите шифртекста. Эти группы битов необязательно должны быть одинакового размера. Группа четырех взятых из открытого текста битов, принимающая значения от 0 до 15, может быть сильно коррелирована с группой трех битов шифртекста, принимающей значения от 0 до 7, так что число возможных сочетаний значительно возрастает.

Мало того – четыре бита в 4-битовой группе могли бы быть любыми битами одного байта открытого текста. Биты 7, 2, 5, 1, взятые в этом порядке, ничем не хуже битов 1, 2, 3, 4. Линейная подстановка могла бы складывать эти четыре бита с четырьмя другими битами байта ключа по модулю 16. Количество возможных комбинаций неимоверно велико. Повторим, сумма любой группы битов, взятых в любом порядке, с символом ключа может быть линейно коррелирована с любой группой битов символа шифртекста, взятых в любом порядке. И все эти бесчисленные корреляции нужно проверять.

Но не спешите тянуться за таблеткой экседрина или рюмкой текилы – есть и хорошие новости. Возможно, все проверять и не придется. Если шифр не был спроектирован специально, чтобы передавать эти значения от раунда к раунду в неизменном виде, то корреляции с открытым текстом несущественны. Они настолько ослабевают с каждым последующим раундом, что по достижении последнего раунда уже не прослеживаются.

### 12.3.4 Включение закладки

Вы, наверное, обратили внимание на слово «возможно». Есть одно исключение – когда вы подозреваете наличие закладки, т. е. что шифр специально спроектирован так, чтобы люди, владеющие секретом, могли читать сообщения, не зная ключа. Например, национальное разведывательное управление могло бы снабдить своих агентов шифром с закладкой, чтобы иметь возможность следить за сообщениями и выявлять предателей.

Сменим амплуа. Предположим, что вы Z, шеф шпионской сети, которому поручено спроектировать такой шифр. Требуется построить шифр, который внешне ничем не отличается от стойкого блочного шифра, чтобы пользователи ничего не заподозрили. Например, шифр должен обладать свойством пятьдесят на пятьдесят, когда изменение одного бита ключа или шифртекста приводит к изменению примерно половины битов шифртекста без явных закономерностей. Если не все подстановки в вашем блочном шифре линейны, то на-



лицо верный признак стойкого блочного шифра. И вы хотите, чтобы ваш шифр по видимости обладал этим свойством.

Опишу один метод, позволяющий спрятать закладку в шифре. Поскольку он основан на последовательной нелинейности, назову его методом *последовательной закладки*, как и построенные с его помощью шифры. *Z* может читать сообщения, зашифрованные таким шифром, не зная ключа, но для человека, не знающего, как работает закладка, он неотличим от стойкого и безопасного блочного шифра. Метод состоит из тех частей: *маскировка*, *сокрытие* и *камуфлирование*.

## МАСКИРОВКА

В шифрах с последовательными закладками используются линейные подстановки шестнадцатеричных цифр. Каждый блок открытого текста и ключа рассматривается как последовательность 4-битовых шестнадцатеричных цифр. Операция шифрования – это сложение по модулю 16 шестнадцатеричных цифр блока сообщения и ключа. Пусть байт состоит из двух шестнадцатеричных цифр  $p_1$  и  $p_2$ , которые шифруются шестнадцатеричными цифрами ключа  $k_1$  и  $k_2$ . Линейная подстановка заменяет  $p_1$  и  $p_2$  на

$$\begin{aligned} q_1 &= ap_1 + bp_2 + ck_1 + dk_2 + e \text{ и} \\ q_2 &= fp_1 + gp_2 + hk_1 + ik_2 + j. \end{aligned}$$

Коэффициенты  $a, b, c, d, e, f, g, h, i, j$  могут быть любыми целыми числами от 0 до 15, а число  $ag - fb$  должно быть нечетным. Если шифр состоит из нескольких раундов, то эти 10 значений могут быть различны для каждого раунда.

Такой тип линейной замены Эмили легко обнаружит. В частности, младшие биты каждой шестнадцатеричной цифры в точности линейны, поэтому простой побитовый тест на линейность это сразу найдет. Чтобы избежать обнаружения, мы можем замаскировать шестнадцатеричные цифры. Сначала запишем их в каком-то перемешанном порядке, например:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Номер позиции
5	C	3	B	0	F	8	4	D	1	9	E	6	A	7	2	Перемешанные (замаскированные) цифры

Чтобы сложить две замаскированные шестнадцатеричные цифры, мы складываем их позиции в перемешанном списке и получаем позицию суммы в нем. Например, чтобы сложить **1+2**, мы находим, что цифра **1** находится в позиции 9 перемешанного списка, а цифра **2** в позиции F, поэтому вычисляем  $9+F \bmod 16$  и получаем 8. Сумма находится в позиции 8 списка. Это **D**, т. е. **1+2 = D**.

Аналогично, чтобы перемножить две замаскированные шестнадцатеричные цифры, мы перемножаем их позиции в перемешанном

списке и получаем позицию произведения в нем. Например, чтобы умножить  $2 \times 3$ , находим, что цифра **2** находится в позиции F, а цифра **3** в позиции 2, поэтому вычисляем  $F \times 2 \bmod 16$  и получаем E. Произведение находится в позиции E списка. Это **7**, т. е.  $2 \times 3 = 7$ .

По существу, маскировка – это простая подстановка шестнадцатеричных цифр. Если подстановка нелинейна, то между открытым и шифртекстом не будет линейной связи ни в одном бите. Обнаружить такого типа замаскированную линейность Эмили гораздо труднее, но чтобы по-настоящему обескуражить ее, мы можем скрыть замаскированные цифры.

## СОКРЫТИЕ

Если в качестве шестнадцатеричных цифр всегда используются биты 1–4 и 5–8 каждого байта блока и ключа, то у Эмили все еще есть шанс обнаружить линейность. Чтобы серьезно усложнить ей задачу, мы можем скрыть, какие биты каждого байта используются. Вместо того чтобы использовать биты (1, 2, 3, 4) открытого текста и ключа и помещать сумму в биты (1, 2, 3, 4) шифртекста, мы могли бы брать шестнадцатеричные цифры из битов (2, 7, 4, 1) открытого текста (именно в таком порядке) и битов (4, 8, 3, 5) ключа, а результат помещать в биты (8, 6, 1, 7) байта шифртекста. Годится любая комбинация 4 бит в любом порядке, лишь бы две шестнадцатеричные цифры каждого байта включали все восемь бит по одному разу.

Чтобы не было неясностей, поясню, что Сандра вовсе не должна извлекать биты из каждого байта, дешифровать замаскированную линейную подстановку, выполнять арифметические действия, а затем перепаковывать биты-результаты в другом порядке. Это было бы слишком медленно, и Эмили сразу поняла бы, что происходит. Все эти действия Сандра выполняет при построении таблицы подстановок. В процессе зашифровывания она просто использует байт ключа, чтобы выбрать из таблицы строку, а затем применяет подстановку к байту открытого текста. А маскировка и сокрытие уже встроены в таблицу подстановок.

## КАМУФЛИРОВАНИЕ

То, что описано выше, – просто сильно усложненный многоалфавитный шифр. Эмили могла бы вскрывать сообщения методами из раздела 5.8.3. Чтобы шифр с последовательной закладкой выглядел как стойкий блочный шифр, необходимо камуфлирование, скрывающее лежащий в его основе многоалфавитный шифр.

Можно, например, применять к блоку перестановку битов после каждого раунда. Тогда шифр будет выглядеть как подстановочно-перестановочная сеть (раздел 11.1). Чтобы сохранить скрытую линейность, образы 4 бит, составляющих каждую шестнадцатеричную цифру, должны принадлежать одному байту. Необязательно, чтобы они оказались в тех же позициях, что и раньше, они могут даже не

быть соседними, но обязаны находиться в одном байте. Иными словами, каждый входной байт разбивается на две шестнадцатеричные цифры, которые на следующем раунде оказываются в двух других байтах в переставленном порядке. К сожалению, если у Эмили есть доступ к опубликованным спецификациям шифра с последовательной закладкой, то она легко сможет распознать такой камуфляж.

Рассмотрим другую форму камуфлирования, обнаружить которую Эмили будет гораздо труднее. Этот метод заимствует идею из стандарта DES (раздел 11.2). Каждый блок шифра делится на две половины. В каждом раунде сначала левая половина используется как ключ для шифрования правой, а потом правая – как ключ для шифрования левой. Мы уже видели, как можно замаскировать и скрыть линейность внутри таблицы подстановок, поэтому воспользуемся этим, чтобы создать иллюзию стойкого блочного шифра.

Каждый раунд шифра будет состоять из четырех шагов. (1) Каждый байт из левой половины шифруется одним байтом ключа. (2) Каждый байт из правой половины шифруется одним байтом из левой половины, используемым в роли ключа. (3) Каждый байт из правой половины шифруется одним байтом ключа. (4) Каждый байт из левой половины шифруется одним байтом из правой половины, используемым в роли ключа.

Чтобы все это казалось суперстойким, каждый байт блока следует шифровать, используя разные байты ключа в каждом раунде, и байты из одной половины блока следует шифровать разными байтами из другой половины. Можете пойти еще дальше и перемешивать байты блока и ключа перед каждым раундом. Можете сделать ключ длиннее блока, чтобы усилить впечатление стойкости. Но шифр при этом останется линейным, потому что на каждом шаге каждого раунда линейность сохраняется.

## Память

Рассмотрим механизм работы шифра с последовательной закладкой. Каждый байт ключа, открытого текста и шифртекста содержит две шестнадцатеричные цифры. Каждая из них могла бы занимать любые 4 бита байта в любом порядке. Назовем этот упорядоченный набор из четырех битов *битовой конфигурацией* шестнадцатеричной цифры, а комбинацию двух шестнадцатеричных цифр в байте – *байтовой конфигурацией*. Конфигурация ключа обычно не изменяется, но байтовая конфигурация открытого и шифртекста могут меняться на любой стадии шифрования.

Для каждой подстановки имеется 6 битовых конфигураций: две для ключа, две для открытого текста и две для шифртекста. Для каждой шестнадцатеричной цифры перестановка 16 шестнадцатеричных значений может быть различна, поэтому существует также 6 перестановок шестнадцатеричных значений для каждой подстановки: две для ключа, две для открытого текста и две для шифртекста. Эта

комбинация шести конфигураций и шести перестановок определяет таблицу подстановок. Для каждой комбинации битовых конфигураций и перестановок нужна отдельная таблица подстановок.

Каждая таблица занимает 65 536 байт, поэтому потребный объем памяти может стать проблемой. В таком случае я рекомендую использовать не более двух байтовых конфигураций и для каждой битовой конфигурации не более двух различных перестановок, быть может, используя их попеременно в соседних раундах. Чтобы еще уменьшить объем памяти, можно рассмотреть возможность использования одной и той же перестановки всякий раз, как применяется одна и та же битовая конфигурация.

### 12.3.5 Конденсированная линейность

В большинстве случаев мы не встраиваем в свой шифр закладки, поэтому не интересуемся последовательной линейностью. Обратимся теперь ко второму типу линейности – конденсированной. В этом случае группа битов преобразуется – конденсируется – в один бит путем применения ко всем четырем операции ИСКЛЮЧАЮЩЕЕ ИЛИ. Таким образом, группы 000, 011, 101 и 110 конденсируются в 1. Любая группа битов открытого текста и (или) ключа потенциально может коррелировать с любой группой битов шифртекста для любого S-блока. Если в блочном шифре ИСКЛЮЧАЮЩЕЕ ИЛИ используется для комбинирования выходов S-блоков с остальной частью блока, то эта линейность может передаваться от раунда к раунду, и будет существовать линейная связь между начальным открытым текстом в первом раунде и конечным шифртекстом в последнем раунде. Проектировщик шифра должен либо избегать такого использования ИСКЛЮЧАЮЩЕГО ИЛИ, либо тщательно проверять, что S-блоки не содержат подобных линейностей.

Предположим, что S-блок принимает 8-битовый открытый текст и порождает 8-битовый шифртекст. Существует 255 способов выбрать группу битов из открытого текста и столько же способов выбрать группу битов из шифртекста. (Порядок битов не имеет значения, потому что  $a \oplus b = b \oplus a$ .) Всего получается  $255^2 = 65\,025$  пар групп, подлежащих проверке. В каждом случае проверяется наличие корреляции между 256 значениями открытого текста и 256 значениями шифртекста. Это легко сделать даже на персональном компьютере.

Если S-блок принимает 8-битовый открытый текст и 8-битовый ключ, а порождает 8-битовый шифртекст, то существует 65 535 различных способов выбрать группу битов из объединения открытого текста и ключа и прежние 255 способов выбрать группу битов из шифртекста. Всего получается  $65\,535 \times 255 = 16\,711\,425$  пар, подлежащих проверке. На ПК это займет довольно много времени, потому что в каждом вычислении коэффициента корреляции участвуют все 65 536 комбинаций открытого текста и ключа. То есть центрировать, масштабировать и сложить предстоит свыше  $10^{12}$  значений.

Самое время поговорить о том, как выполнить эти проверки эффективно. Существует несколько приемов, способных заметно ускорить процесс. (1) Для выбора комбинации битов из байта использовать маску. Например, если нам нужны биты 2, 4, и 7, то нужно использовать маску 01010010, в которой единицы расположены в позициях 2, 4 и 7. Применяв эту маску к каждому байту открытого текста с помощью операции И, мы выберем нужные биты. (2) Чтобы перепробовать все возможные комбинации битов, не нужно строить маски по одной за раз, лучше просто в цикле увеличивать значение маски от 1 до 255. (3) Для конденсации битов не нужно каждый раз пользоваться операциями сдвига и ИСКЛЮЧАЮЩЕГО ИЛИ. Сделайте это один раз и постройте таблицу конденсированных значений. После этого комбинацию битов можно использовать для поиска в этой таблице. Если имеется комбинация битов ключа и битов открытого текста, то их можно объединить ИСКЛЮЧАЮЩИМ ИЛИ, а результат конденсировать с помощью таблицы, так что понадобится один поиск в таблице вместо двух.

### 12.3.6 Гибридная нелинейность

Полноты ради отмечу, что существует гибридная форма, объединяющая последовательную и комбинированную линейность. Пусть каждый 8-битовый байт разбит на четыре 2-битовые группы. Эти группы могли бы быть последовательно линейными относительно сложения по модулю 4. Можно было бы конденсировать две или более групп, складывая их по модулю 4. То же самое можно было бы сделать с 3-битовыми группами по модулю 8 или с 4-битовыми группами по модулю 16.

Ограничимся 2-битовыми группами. Каждая группа могла бы состоять из двух бит, взятых из любого места байта. Например, байт можно было бы разложить на 4 группы: биты (6, 1), (4, 8), (2, 5) и (7, 3). Мы можем конденсировать несколько 2-битовых групп в одну, сложив их по модулю 4 или взяв любую другую линейную комбинацию по модулю 4. Например, если 2-битовыми группами являются A, B, C и D, то их можно было бы объединить в новую 2-битовую группу  $pA + qB + rC + sD + t \pmod{4}$ , где  $p, q, r, s$  и  $t$  – фиксированные целые числа от 0 до 3, причем по меньшей мере одно из  $p, q, r, s$  нечетно.

Эти типы конденсированных групп могли бы коррелировать с аналогичными гибридными группами битов шифртекста или с регулярными или конденсированными битовыми группами, взятыми из шифртекста. Если вы жаждете абсолютной уверенности, то следует проверить на корреляцию все возможные пары линейных групп, конденсированных групп и гибридных групп.

### 12.3.7 Конструирование S-блока

Существует три метода конструирования S-блоков с хорошими свойствами нелинейности: *метод циферблата*, *SkipMix* и *Meld8*.

## Метод циферблата

На листе бумаги нарисуйте большую окружность и равномерно расположите на ней буквы алфавита по часовой стрелке, как числа на часовом циферблате. Выберите начальную и вторую буквы и соедините их отрезком прямой. Затем выберите третью букву и соедините ее отрезком со второй и т. д. Определим *пролет* каждого отрезка как число позиций букв, которые нужно миновать при движении вдоль окружности по часовой стрелке от его начала к концу. Например, если в алфавите 26 букв, то пролет отрезка, соединяющего С с D, равен 1, а пролет отрезка, соединяющего D с С, равен 25. Чтобы сделать подстановку как можно более нелинейной, величины всех пролетов должны быть различны.

Делается это следующим образом. Для каждой буквы алфавита составим список всех букв, которые могут следовать за ней. В самом начале список для каждой буквы будет содержать все остальные буквы, т. е. получится 26 списков по 25 букв в каждом. Каждый раз, выбирая букву и добавляя ее в перемешанный алфавит, мы удаляем ее из всех списков. Если пролет от предыдущей буквы до этой равен  $s$ , то также удаляем из всех списков все остальные буквы с пролетом  $s$ . Например, предположим, что мы добавили в алфавит букву Р, а затем R. Пролет от Р до R равен 2, это позиции PQR. Поэтому из списка А мы удалили бы С, из списка В удалили бы D, из списка С – Е и т. д.

В конечном итоге некоторые списки опустеют. Если имеется только одна буква с пустым списком, то она должна будет стать последней буквой перемешанного алфавита. Если букв с пустыми списками две, то мы уперлись в тупик. Начните сначала, вернитесь на шаг назад и попробуйте еще раз. При выборе каждой следующей буквы, добавляемой в алфавит, берите букву с коротким списком, но не с пустым, если только это не последняя оставшаяся буква.

### Историческое отступление

Эта эвристика называется правилом Варнсдорфа по имени Х. К. Фон Варнсдорфа, который в 1823 году использовал ее для построения маршрута шахматного коня, проходящего через все поля доски по одному разу. Улучшенную версию с заглядыванием на два хода вперед предложил в 1965 году Айра Пол из Калифорнийского университета в Санта-Крузе.

Вот пример алфавита, построенного методом циферблата:

CHBOYEQFXGJZAVSDWUIMLTRNP

При проверке этого алфавита на линейность нужно рассмотреть пять различных схем нумерации: схема N1, первая и вторая цифры схемы N2, первая и вторая цифры схемы N3. Для каждой из них нужно вычислить корреляцию с теми же пятью схемами нумерации



стандартного латинского алфавита. Всего получается 25 корреляций. Мы хотим, чтобы каждый коэффициент корреляции находился между  $-0.5$  и  $+0.5$ . Еще лучше было бы сузить интервал: от  $-0.333$  до  $+0.333$ .

Ниже приведены результаты проверок, все 25 коэффициентов корреляции.

0.26632	0.14935	0.23985	0.26830	-0.05641
0.24891	0.16129	0.18143	0.26075	-0.20365
0.04386	-0.01814	0.12363	0.02451	0.28782
0.27645	0.15286	0.25324	0.27935	-0.07132
-0.17949	-0.06788	-0.22614	-0.19359	0.23077

Как видим, все коэффициенты заключены между  $-0.226$  и  $+0.288$ , причем 6 из них даже попали в диапазон от  $-0.1$  до  $+0.1$ , т. е. метод циферблата прекрасно строит нелинейные подстановки.

Но нет гарантии, что такой хороший результат будет получаться каждый раз. Все равно нужно проверять на линейность.

## SkipMix

Выше в этом разделе я упоминал, что алфавит можно строить с помощью алгоритма SkipMix (см. раздел 5.2) с генератором псевдослучайных чисел. Вообще говоря, выбор алфавита случайным образом не дает хорошей нелинейности, поэтому я опишу наилучший способ применения SkipMix более подробно. На этот раз для иллюстрации я возьму 256-символьный алфавит.

Как всегда, начинаем с выписывания всех 256 символов. Сгенерируем случайное число в диапазоне от 1 до 256 для выбора первого символа. Предположим, что это позиция 54 в алфавите. Удаляем этот символ из списка. Осталось 255 символов. Генерируем случайное число в диапазоне от 1 до 255. Пусть оно равно 231. Следующая позиция равна  $54 + 231 = 285$ . Поскольку она больше 255, вычитаем 255 и получаем 30. Удаляем символ в позиции 30 из списка. Мы выбрали два символа, и осталось 254, поэтому генерируем случайное число в диапазоне от 1 до 254. И так далее.

Результатирующий алфавит обладает хорошими свойствами нелинейности, потому что мы каждый раз генерировали случайное число в другом диапазоне. Это в какой-то мере аналогично выбору различных пролетов в методе циферблата. Ниже приведен пример 26-буквенного алфавита, сгенерированного этой версией SkipMix:

**DRWBMEFHNJCQZXTOILVAGUYPSK**

Проверить его можно так же, как предыдущий алфавит. Результаты таковы:

0.26838	0.33037	-0.11239	0.25608	0.15897
0.11314	0.16129	-0.09071	0.09891	0.20365

0.31523	0.34471	-0.04670	0.31860	-0.08223
0.24521	0.31470	-0.12798	0.23347	0.15283
0.32308	0.20365	0.24670	0.31585	0.07692

Хорошие результаты. Все коэффициенты корреляции заключены между  $-0.127$  и  $+0.344$ , причем пять из них попали в диапазон от  $-0.1$  до  $+0.1$ . Но все-таки они хуже, чем полученные методом циферблата.

## Метод MELD8

Этот метод, по существу, является специализированным генератором псевдослучайных чисел. Я буду предполагать, что язык программирования умеет работать с 64-битовыми целыми числами. В зависимости от способа их представления верхняя граница диапазона может быть равна  $2^{62}$  или  $2^{63}$ . На всякий случай буду считать, что  $2^{62}$ . Первый шаг – выбрать два числа: множитель  $m$  длиной от 24 до 26 бит и модуль  $N$  длиной от 35 до 37 бит. Модуль должен быть простым числом. Лучше, если  $m$  является первообразным корнем из  $N$ , но, поскольку я еще не объяснил, что это такое, просто сделаем  $m$  и  $N$  простыми.

Для проверки  $m$  и  $N$  перемножим их. Если результат больше  $2^{62} \approx 4.611 \times 10^{18}$ , то уменьшим  $m$  или  $N$ .

Прежде чем генерировать случайные числа, выберем начальное значение – какое-нибудь целое число от 2 до  $N-1$ . Умножим его на  $m$  по модулю  $N$ , это будет первое псевдослучайное число. Умножив его на  $m$  по модулю  $N$ , получим второе псевдослучайное число. И так далее. Мы построили последовательность случайных чисел в диапазоне от 1 до  $N-1$ . Воспользуемся ими для порождения алфавита.

Предположим, что длина  $N$  равна 36 битам. Пронумеруем их от 1 до 36, начиная со старших. Возьмем первые 8 бит каждого случайного числа, с номерами от 1 до 8. Удалим эти старшие биты и применим ИСКЛЮЧАЮЩЕЕ ИЛИ к ним и к следующим восьми битам, с номерами от 9 до 16. Эта операция называется Meld8. Ее задача – сделать последовательность символов нелинейной. Приведем пример:

1	8	9	16		36	Позиции битов
01100111	01101000	00101001	11010011	0001		36-битовое случайное число
	10100111					Операция Meld8
	11001111	00101001	11010011	0001		28-битовый результат

Следующий шаг – использовать 28-битовое случайное число для генерирования символа. Он зависит от того, какой алфавит мы строим: 26- или 256-символьный. В случае 26-символьного алфавита умножаем это число на 26 и делим на  $2^{28}$  (т. е. сдвигаем вправо на 28 позиций), чтобы получить следующий символ. В случае 256-символьного алфавита для получения следующего символа просто делим на  $2^{20}$ , т. е. сдвигаем на 20 позиций вправо.



Начнем с пустого алфавита и будем добавлять по одному символу. Если символ новый, дописываем его в конец алфавита. Дубликаты отбрасываются. Тот факт, что мы не берем последовательные случайные числа, дополнительно увеличивает нелинейность алфавита. Ниже приведен пример такого алфавита, сгенерированного для модуля  $N = 90392754973$ , множителя  $m = 23165801$  и начального значения  $s = 217934$ :

**ZEJBIRAFHGPYLVQMUTXOKSCNMD**

И вот результирующие коэффициенты корреляции:

<b>0.13983</b>	<b>0.17650</b>	<b>-0.06716</b>	<b>0.14196</b>	<b>-0.04615</b>
<b>0.16745</b>	<b>0.16129</b>	<b>0.01814</b>	<b>0.17084</b>	<b>-0.06788</b>
<b>-0.04934</b>	<b>0.03629</b>	<b>-0.17033</b>	<b>-0.05174</b>	<b>0.04112</b>
<b>0.12566</b>	<b>0.17084</b>	<b>-0.08441</b>	<b>0.12821</b>	<b>-0.05094</b>
<b>0.20000</b>	<b>0.06788</b>	<b>0.26726</b>	<b>0.19359</b>	<b>0.07692</b>

Все коэффициенты заключены между  $-0.170$  до  $+0.267$ , причем одиннадцать из них попали в диапазон от  $-0.1$  до  $+0.1$ . Это лучший из трех примеров, но было бы опрометчиво по одному примеру заключать, что Meld8 – наилучший метод. Всегда проверяйте.

### 12.3.8 S-блок с ключом

В разделе 12.3.7 мы имели дело с S-блоками без ключа. Они выполняли простую подстановку. Если ключ используется, то S-блок выполняет многоалфавитную подстановку общего вида (раздел 5.8.3). S-блок можно рассматривать как таблицу алфавитов, в которой каждая строка является перемешанным алфавитом. S-блоки можно сгенерировать, построив каждый из смешанных алфавитов с помощью метода циферблата, SkipMix, Meld8 или их комбинации.

При использовании метода циферблата или SkipMix нужно всякий раз брать новое случайное начальное значение. При использовании Meld8 допустимо все время применять один и тот же модуль, но начальные значения и множители должны быть разными. И обязательно проверяйте, проверяйте и еще раз проверяйте. Ваша цель – избежать любых линейных связей между комбинацией ключа с открытым текстом и шифртекстом. Если результаты неудовлетворительны, т. е. много коэффициентов корреляции выходят за пределы диапазона от  $-0.35$  до  $+0.35$ , то для устранения проблемы может оказаться достаточно заменить одну строку или переставить места-ми две строки таблицы алфавитов.

## 12.4 Диффузия

Второе свойство Шеннон назвал *диффузией*. Идея в том, что каждый бит или байт шифртекста должен зависеть от каждого бита или байта открытого текста и ключа.

Для иллюстрации вернемся к шифру Bifid Деластеля, основанному на квадрате Полибия. Если размер блока равен  $S$ , то каждая буква сообщения заменяется двумя пятеричными цифрами и эти цифры записываются в массив  $2 \times S$  по столбцам, а считываются по строкам. Затем пары цифр преобразуются обратно в буквы с помощью того же самого или другого квадрата Полибия.

Пусть размер блока равен 7, а буквы в блоке открытого текста обозначим A, B, C, D, E, F, G. Пусть эти буквы представляются цифрами aa, bb, cc, dd, ee, ff, gg. Я опустил нижние индексы, потому что не имеет значения, какая цифра первая, а какая вторая. Тогда блок будет иметь вид:

abcdefg	abcdefg	Записать по столбцам Считать по строкам
abcdefg	abcdefg	

Читая буквы из блока по строкам, мы получаем ab, cd, ef, ga, bc, de, fg. Заметим, что каждая буква шифртекста зависит от двух букв открытого текста. Первая буква шифртекста зависит от A и B, вторая – от C и D и т. д.

Теперь мне необходимо ввести специальную нотацию, чтобы показать, от каких именно букв открытого текста зависит каждая буква шифртекста. Если буква шифртекста зависит от букв открытого текста P, Q и R, то будем обозначать ее pqr. В этой нотации после повторного шифрования букв A, B, C, D, E, F, G блок примет вид:

ab	cd	ef	ga	bc	de	fg
ab	cd	ef	ga	bc	de	fg

Читая эти буквы по строкам, получаем abcd, efga, bcde, fgab, cdef, gabc, defg. Поскольку порядок цифр несуществен, то же самое можно было бы записать в виде abcd, aefg, bcde, abfg, cdef, abcg, defg. После двух шифрований каждая буква шифртекста зависит от четырех букв открытого текста.

Если зашифровать этот блок в третий раз шифром Bifid, то каждая буква шифртекста будет зависеть от всех семи букв открытого текста. Для шифра Bifid с размером блока 7 трех раундов шифрования достаточно для обеспечения полной диффузии. Если бы размер блока был равен 9, 11, 13 или 15, то понадобилось бы четыре раунда. (Напомним, что в шифре Bifid размер блока должен быть нечетным.)

В общем случае проверка на диффузию начинается с того, что каждый символ или бит открытого текста зависит только от себя самого. Если шифр оперирует целыми байтами или символами, то прослеживается байтовая диффузия. Если он оперирует цифрами – шестнадцатеричными или в какой-то другой системе счисления – либо отдельными битами, то диффузия прослеживается на уровне этих единиц. В случае шифра Bifid единицами являются координаты в квадрате Полибия, т. е. пятеричные цифры.

Для прослеживания диффузии нужно как-то представить множество единиц открытого текста и ключа, распространяющихся по раундам блочного шифра. Если единиц открытого текста немного, как в примере с шифром Bifid, то достаточно их просто перечислить. Но по мере увеличения количества единиц открытого текста, ключа и шифртекста возникает необходимость в более компактном представлении. Хорошая стратегия – создать двоичный вектор для каждой единицы шифртекста. Назовем его *вектором зависимостей*. Каждый элемент вектора зависимостей соответствует одному входу – единице открытого текста или ключа. Элемент равен 1, если единица шифртекста зависит от этой входной единицы, и 0 в противном случае.

Когда из двух или более входных единиц формируется выходная, их векторы зависимостей объединяются с помощью ИЛИ и образуют вектор зависимостей выходной единицы. Для иллюстрации снова рассмотрим пример с Bifid, пользуясь только что описанной нотацией. В начальный момент каждый символ зависит только от самого себя. Это можно представить векторами

**1000000 0100000 0010000 0001000 0000100 0000010 0000001**

После первого применения шифра Bifid каждая результирующая буква зависит от двух букв открытого текста. Первый выходной байт раунда 1 зависит от первых двух входных байтов раунда 1, результат применения ИЛИ к их векторам зависимостей **1000000V0100000** дает **1100000**. Вторая выходная буква зависит от третьей и четвертой букв открытого текста, применение ИЛИ к их векторам зависимостей **0010000V0001000** дает **0011000**. И так далее. Выход первого раунда представлен векторами

**1100000 0011000 0000110 1000001 0110000 0001100 0000011**

После второго раунда Bifid первая выходная буква зависит от первого и второго выходов первого раунда, поэтому мы применяем ИЛИ к их векторам зависимостей, **1100000V0011000**, и получаем **1111000**. Вторая выходная буква зависит от третьего и четвертого выходов первого раунда, поэтому применяем ИЛИ к их векторам зависимостей, **0000110V1000001**, и получаем **1000111**. И так далее. После двух раундов Bifid каждая буква зависит от четырех букв открытого текста, что можно представить векторами

**1111000 1000111 0111100 1100011 0011110 1110001 0001111**

После третьего раунда Bifid каждая выходная буква зависит от всех семи букв открытого текста в раунде 1, например **1111000V1000111 = 1111111**. Выход третьего раунда представлен векторами

**1111111 1111111 1111111 1111111 1111111 1111111 1111111**

Когда встречается S-блок, векторы зависимостей для выходных единиц формируются путем применения ИЛИ к векторам для каждого входа, который вносит вклад в данный выход. Рассмотрим несколько ситуаций, которые могут возникнуть в блочном шифре.

Если к двум единицам применяется ИСКЛЮЧАЮЩЕЕ ИЛИ, то вектор зависимостей для выходной единицы образуется применением ИЛИ к векторам для каждого входа. То же самое делается, когда несколько единиц объединяются с помощью какой-то комбинирующей функции, например **sxor** или **madd**.

Если единицы блока переставляются с помощью ключа, то каждая выходная единица оказывается зависима от всех единиц этого ключа, поэтому векторы для ключа объединяются операцией ИЛИ с векторами для каждой выходной единицы.

Предположим, что S-блок создан перемешиванием его алфавита с применением ключа. Если S-блок фиксированный или статический, скажем встроен в оборудование, то ключ перемешивания не участвует в процессе. Если S-блок переменный, быть может, перемешиваемый разными ключами при каждом шифровании, то его выходные единицы зависят от всех единиц ключа. Векторы для ключа объединяются с помощью ИЛИ с вектором для каждой выходной единицы.

Диффузию можно описать одним числом. Образует матрицу, составленную из векторов зависимостей для всех выходных единиц. Каждая строка матрицы будет представлять одну выходную единицу последнего раунда блочного шифра. Каждый столбец матрицы будет представлять одну входную единицу – ключ или открытый текст. Мерой, или *индексом, диффузии* называется доля элементов этой матрицы, равных 1. Если все элементы равны 1, то диффузия полная и ее индекс равен 1. Если S-блоки нелинейны и ключ длинный, то это является признаком стойкости шифра.

Диффузия – это еще не все. Существуют стойкие шифры с диффузией меньше 1. Примером может служить блочный шифр, в котором на каждом раунде используется свой ключ. Ключи ранних раундов могут достигать полной диффузии, но на поздних раундах и особенно на последнем это может быть не так. Тем не менее если обеспечивающие полную диффузию ключи содержат целевое количество битов, то шифр может быть уже безопасен, а ключи, дающие частичную диффузию, – просто дополнительная страховка.

Следующий пример поможет понять, как шифр может оказаться стойким, даже если диффузия неполная. Рассмотрим шифр с 12 раундами, на каждом из которых используются независимые 24-битовые ключи. В этом шифре для обеспечения полной диффузии требуется 6 раундов, т. е. после 6 раундов открытый ключ и ключ первого раунда полностью диффундировали. После 7 раундов полностью диффундировали открытый текст и ключи первого и второго раундов. И так далее. После 12 раундов полностью диффундировали открытый текст и ключи первых 7 раундов. При 24-битовых раундовых

ключах мы имеем 168 бит полностью диффундировавших ключей. Если целевая стойкость равна 128 битам, то цель уже с лихвой перекрыта. Частично диффундировавшие ключи раундов с восьмого по двенадцатый – дополнительный бонус.

## 12.5 Насыщение

Конфузия и диффузия – два столпа безопасности шифра. Для пущей уверенности в том, что блочный шифр покоится на надежном фундаменте, я предлагаю третий столп, который назвал *насыщением*. Диффузия показывает лишь, зависит данная выходная единица от данной входной единицы или нет. Насыщение измеряет, насколько сильна эта зависимость. Я покажу, как вычислить *индекс насыщения*, аналогичный индексу диффузии из предыдущего раздела. Насыщение – это, по существу, уточненная версия диффузии. В случае диффузии показатель зависимости может принимать только значения 0 или 1, а в случае насыщения – любое неотрицательное значение.

Объясню вкратце, в чем смысл насыщения. Предположим, что блочный шифр X состоит из нескольких раундов подстановки. В каждом раунде каждый байт сообщения объединяется операцией ИСКЛЮЧАЮЩЕЕ ИЛИ с одним байтом ключа, после чего к результату применяется простая подстановка. Предположим, что в каждом раунде используются разные байты ключа, так что каждый байт ключа применяется по одному разу для каждого байта блока. Насыщение шифра X будет небольшим, потому что каждый байт шифртекста зависит от каждого байта ключа только один раз. Чтобы увеличить насыщение, каждый выходной байт должен зависеть от каждого входного несколько раз.

Еще один пример поможет прояснить эту мысль. Представьте себе шифр, который оперирует 48-битовым блоком, рассматриваемым как шесть 8-битовых байтов. Каждый раунд шифра состоит из двух шагов: (1) блок циклически сдвигается влево на одну позицию, так что самый левый бит становится самым правым, а затем (2) к каждому из восьми байт применяется простая подстановка S. После первого раунда первый выходной байт **C1** зависит от последних 7 бит первого байта открытого текста и первого бита второго байта открытого текста:

P1	P2	P3	P4	P5	P6	Открытый текст, 6 байт
aaaaaaa	bbbbbbb	ccccccc	ddddddd	eeeeeee	ffffff	Открытый текст, 48 бит
<u>aaaaaaab</u>	<u>bbbbbbbc</u>	<u>cccccccd</u>	<u>ddddddde</u>	<u>eeeeeeef</u>	<u>fffffffa</u>	Циклический сдвиг влево на 1 бит
C1	C2	C3	C4	C5	C6	После подстановки

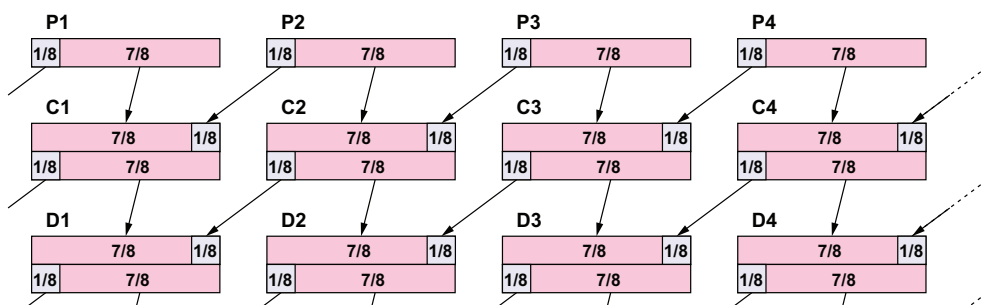
Символ шифртекста **C1** зависит от 7 бит байта открытого текста **P1** и 1 бита байта открытого текста **P2**. Можно сказать, что **C1** зависит на 7/8 от **P1** и на 1/8 от **P2**.

Рассмотрим второй раунд. Обозначим его выходы **D1 ... D6**.

<b>C1</b>	<b>C2</b>	<b>C3</b>	<b>C4</b>	<b>C5</b>	<b>C6</b>	Входы раунда 2
gggggggg	hhhhhhh	iiiiiii	jjjjjjj	kkkkkkk	lllllll	48 бит
gggggggh	hhhhhhi	iiiiij	jjjjjjk	kkkkkkkl	llllllg	Циклический сдвиг влево на 1 бит
<b>D1</b>	<b>D2</b>	<b>D3</b>	<b>D4</b>	<b>D5</b>	<b>D6</b>	После подстановки

Символ шифртекста **D1** зависит на 7/8 от **C1** и на 1/8 от **C2**. Но **C1** зависит на 7/8 от **P1** и на 1/8 от **P2**, а **C2** зависит на 7/8 от **P2** и на 1/8 от **P3**. Единственный вклад **P1** в **D1** косвенный, через **C1**. Поскольку **D1** на 7/8 зависит от **C1**, а **C1** на 7/8 зависит от **P1**, разумно будет сказать, что **D1** на 49/64 зависит от **P1**. По той же причине **D1** зависит на 1/64 от **P3**. Назовем эти величины *коэффициентами насыщения*, а само вычисление в случае, когда имеется только одна зависимость, – *вычислением S1*.

Диаграмма прояснит этот процесс.



А как насчет **P2**? Вклад **P2** в **D1** опосредован **C1** и **C2**. Будет разумно сказать, что **D1** на 7/8 зависит от **C1**, которое на 1/8 зависит от **P2**, и на 1/8 от **C2**, которое на 7/8 зависит от **P2**, и отсюда сделать вывод, что **D1** на  $(7/8)(1/8) + (1/8)(7/8) = 14/64$  зависит от **P2**. Это естественное вычисление приводит к более тонкой версии диффузии. Однако сумма всех вкладов в любую единицу всегда равна 1 и не может оказаться больше. Если повторить это вычисление много раз, то все величины диффузии будут сходиться к 1/48. Но это совсем не то, что пытается уловить понятие насыщения. Насыщение должно возрастать, когда вклад в единицу вносят несколько разных источников.

Если единица получает несколько вкладов, то для определения коэффициента насыщения используется другое вычисление. Предположим, что коэффициенты насыщения двух источников равны  $a$  и  $b$ , где  $a \geq b$ . Тогда объединенный коэффициент насыщения равен  $a+b/2$ . Если имеется три источника с коэффициентами насыщения  $a$ ,  $b$  и  $c$ , где  $a \geq b \geq c$ , то объединенный коэффициент равен  $a+b/2+c/4$ . В общем случае составляющие коэффициенты сортируются в порядке убывания –  $a \geq b \geq c \geq d \geq e \dots$  и объединенный коэффициент насыщения вычисляется по правилу:

2 коэффициента:  $a + b/2$

3 коэффициента:  $a + b/2 + c/4$

4 коэффициента:  $a + b/2 + c/4 + d/8$

5 коэффициентов:  $a + b/2 + c/4 + d/8 + e/16$

...

8 коэффициентов:  $a + b/2 + c/4 + d/8 + e/16 + f/32 + g/64 + h/128$

Назовем это вычисление при наличии нескольких зависимостей *вычислением S2*. Вычисление S1 применяется, когда источник один, а вычисление S2 – когда их несколько.

Может показаться, что вычисление S2 взято с потолка и даже выглядит эксцентрично, но оно обладает нужными для насыщения свойствами. Во-первых, результат возрастает, если единица зависит от нескольких предшественников. Это понятно, потому что  $a + b/2$  всегда больше  $a$ . Во-вторых, он возрастает не слишком быстро. При переходе к следующему раунду коэффициенты насыщения не могут увеличиться более чем вдвое. Действительно,  $a + a/2 + a/4 + \dots + a/2^n < 2a$  для любого  $n$ . Например,  $1 + 1/2 + 1/4 + 1/8 = 15/8 = 1.875$ .

В рассматриваемом случае, когда **D1** зависит от **P2**, входные коэффициенты равны  $7/8$  и  $1/8$ , так что объединенный коэффициент равен  $7/8 + (1/8)/2 = 15/16$ . Коэффициенты насыщения для выходной единицы можно организовать в виде вектора, как и величины диффузии. Таким образом, результирующий вектор насыщения для **D1** равен  $(49/64, 15/16, 1/64, 0, 0, 0)$ . Эти векторы можно затем собрать в матрицу насыщения. Тогда индексом насыщения называется наименьший коэффициент в матрице насыщения.

Рассмотрим более реалистичный шифр, который был предложен в литературе и, вполне возможно, использовался на практике. Я буду называть его *SFlip* (Substitute and Flip – подстановка с транспонированием). Это близкий родственник шифра Poly Triple Flip, описанного в разделе 11.7.5. Если вы не помните, что такое транспонирование матрицы, обратитесь к разделу 11.7. Шифр SFlip применяется к 8-байтовому блоку и состоит из нескольких раундов и завершающего шага. Каждый раунд включает два шага. (1) Ко всем восьми 8-битовым байтам применяется простая подстановка. (2) Битовая матрица  $8 \times 8$  транспонируется. На завершающем шаге к каждому из 8-битовых байтов еще раз применяется подстановка.

Для битовой матрицы  $8 \times 8$  необходима матрица зависимостей  $64 \times 64$ . Она слишком велика для печатной страницы, поэтому я продемонстрирую шифр в миниатюре. Будем использовать битовую матрицу  $3 \times 3$  с матрицей зависимостей  $9 \times 9$ . Мы проанализируем этот шифр дважды, с помощью диффузии и с помощью насыщения. Начнем с диффузии. Пометим биты в блоке текста и в матрице зависимостей следующим образом:



abc	abcdefghi
def	
ghi	

До первого раунда каждый бит зависит только от себя самого, поэтому матрица зависимостей имеет вид (1). После подстановки в первом раунде каждый бит зависит от всех трех бит в своем символе, поэтому матрица зависимостей имеет вид (2). После транспонирования в первом раунде матрица зависимостей принимает вид (3), а после подстановки во втором раунде – вид (4).

(1)	(2)	(3)	(4)
a00000000	abc000000	abc000000	abcdefghi
0b0000000	abc000000	000def000	abcdefghi
00c000000	abc000000	000000ghi	abcdefghi
000d00000	000def000	abc000000	abcdefghi
0000e0000	000def000	000def000	abcdefghi
00000f000	000def000	000000ghi	abcdefghi
000000g00	000000ghi	abc000000	abcdefghi
0000000h0	000000ghi	000def000	abcdefghi
00000000i	000000ghi	000000ghi	abcdefghi

Иными словами, в этот момент каждый бит шифртекста зависит от всех битов открытого текста. Это остается справедливым после транспонирования во втором раунде и после завершающей подстановки. Поэтому, опираясь только на вычисление зависимостей, мы приходим к выводу, что шифр становится безопасным уже после двух раундов. Но это неверно. Ади Шамир показал, что двух раундов недостаточно.

Теперь проанализируем шифр SFlip с помощью индекса насыщения. После подстановки в первом раунде каждый бит шифртекста зависит на 1/3 от каждого из трех соответствующих битов открытого текста. Матрица насыщения имеет вид (5). После транспонирования в первом раунде матрица насыщения принимает вид (6).

(5)	(6)
%%%000000	%%%000000
%%%000000	000%%%000
%%%000000	000000%%%
000%%%000	%%%000000
000%%%000	000%%%000
000%%%000	000000%%%
000000%%%	%%%000000
000000%%%	000%%%000
000000%%%	000000%%%



Подстановка во втором раунде делает каждый выходной бит зависимым от всех 9 бит открытого текста перед первым раундом. Коэффициент насыщения равен  $1/3 + (1/3)/2 + (1/3)/4 = 1/3 + 1/6 + 1/12 = 7/12 \approx 0.583$ . Все элементы матрицы насыщения равны этому числу, поэтому индекс насыщения равен  $7/12$ . Целевая величина индекса насыщения 1, хотя для пущей уверенности можно было бы задать и большую. Вот чему равен индекс после нескольких раундов:

Раунд	3×3	8×8
1	0.333	0.125
2	0.583	0.249
3	1.021	0.496
4	1.786	0.988
5	3.126	1.969

Итак, трех раундов достаточно для шифра 3×3, но для шифра 8×8 необходимо 5 раундов.

Обратимся теперь к ситуациям, когда выходная единица зависит от одной или нескольких входных единиц.

Если на вход S-блока подается и входной текст, и ключ, скажем  $p$  единиц открытого текста и  $k$  единиц ключа, то зависимость для каждого из его выходных единиц будет равна  $1/(p+k)$ . Например, если на вход подается 6 бит ключа и 4 бита открытого текста, то зависимость для каждого выходного бита будет равна  $1/10$ . Если входы S-блока сами зависят от более ранних входов, то для вычисления индекса насыщения следует использовать вычисление  $S1$  или  $S2$  (в зависимости от ситуации).

Аналогично, если две или более единиц объединяются с помощью ИСКЛЮЧАЮЩЕГО ИЛИ или какой-то другой комбинирующей функции, то при  $n$  входах зависимость равна  $1/n$ . Вычисление индекса насыщения такое же, как для S-блока с такими же входами.

Когда для перестановки используется  $k$ -битовый ключ, каждая входная единица перестановки имеет зависимость  $1/k$  от каждого из битов ключа и зависимость 1 от входного открытого текста. Предположим, что символ открытого текста перемещается перестановкой из позиции  $a$  в позицию  $b$ . Вектор насыщения для  $p$  после перестановки будет таким же, как до перестановки, за исключением столбцов, соответствующих битам ключа перестановки. В этих столбцах коэффициент насыщения будет определяться вычислением  $S1$  или  $S2$ .

Приведу пример. Предположим, что  $t$  – один из битов ключа перестановки. Если  $p$  не зависел от  $t$  до перестановки, т. е. в столбце  $t$  его вектора насыщения находился 0, то после перестановки значение в этом столбце будет равно  $1/k$ . С другой стороны, если бы  $p$  уже зависел от бита ключа  $t$ , то коэффициент насыщения определялся бы вычислением  $S2$ . Если бы коэффициент в столбце  $t$  был равен  $x$ , то после перестановки он стал бы равен  $x + 1/2k$ , если  $x \geq 1/k$ , или  $1/k + x/2$ , если  $x < 1/k$ .

Если при перемешивании алфавита или таблицы алфавитов для шага подстановки используется  $k$ -битовый ключ, то перемешанный алфавит или таблица имеют зависимость  $1/k$  от каждого бита ключа. При каждой подстановке символа с использованием этого алфавита выходной символ приобретает дополнительную зависимость  $1/k$  от каждого бита ключа. Это объединяется с зависимостями входного символа (и ключа подстановки, если таковой имеется) с помощью вычисления  $S1$  или  $S2$ , так что в результате получается коэффициент насыщения для выходного символа.

## Резюме

Блочный шифр будет невскрываемым на практике, если выполняется *каждое* из следующих правил:

- 1 размер блока достаточно велик. Современный стандарт – 16 символов, или 128 бит;
- 2 ключ достаточно велик. Современный стандарт – от 128 до 256 бит. Ключ должен быть не меньше блока, а лучше больше;
- 3 либо используются сильно нелинейные S-блоки, либо переменные таблицы подстановки, хорошо перемешанные длинным ключом;
- 4 индекс насыщения должен быть не меньше 1.

Как всегда, здоровая консервативность не повредит. Работайте с запасом. Делайте ключ длиннее и используйте больше раундов, чем необходимо, потому что компьютеры становятся все быстрее и постоянно обнаруживаются новые виды атак. В частности, можно задать целевой коэффициент насыщения больше 1, быть может 2, 3 или даже 5.

# 13

## Потоковые шифры

---

### *Краткое содержание главы:*

- генераторы псевдослучайных чисел;
- функции для комбинирования случайных чисел с сообщением;
- генерирование истинно случайных чисел;
- функции хеширования.

Потоковые шифры – противоположность блочным. Символы шифруются по мере поступления, обычно по одному. Основная идея – объединить поток символов сообщения с потоком символов ключа для порождения символов шифртекста. Эта парадигма хорошо подходит для непрерывных операций, когда сообщения постоянно шифруются и передаются одной стороной и постоянно принимаются и дешифрируются другой стороной, с короткими паузами на время смены ключей.

Мы уже встречались с несколькими потоковыми шифрами. Шифры с автоключом и бегущим ключом в разделе 5.9, роторные машины в разделе 5.10, подстановка Хаффмана в разделе 10.4 и шифры на основе сжатия текста в разделе 10.7 – все это примеры потоковых шифров.

## 13.1 Комбинирующие функции

В большинстве распространенных потоковых шифров используется одна единица ключа для шифрования одной единицы открытого текста. Единицами обычно являются буквы или байты, но можно использовать шестнадцатеричные цифры и даже биты. Единица ключа объединяется с единицей открытого текста с применением, по существу, тех же самых комбинирующих функций, что для пульсирующих шифров в разделе 11.8, только вместо предшествующей единицы используется единица ключа. Ниже перечислены аналогичные методы, где  $x_n$  обозначает  $n$ -ю единицу сообщения,  $k_n$  –  $n$ -ю единицу ключа,  $A$  и  $B$  – простые подстановки, а  $P$  – многоалфавитную подстановку общего вида. Подстановки  $A$ ,  $B$  и  $P$  должны быть перемешаны с помощью ключей, фиксированные или встроенные подстановки не допускаются.

<b>xor</b>	ИСКЛЮЧАЮЩЕЕ ИЛИ	$x_n = k_n \oplus x_n$
<b>sxor</b>	Подстановка, затем ИСКЛЮЧАЮЩЕЕ ИЛИ	Имеется три варианта: $x_n = A(k_n) \oplus x_n$ , $x_n = k_n \oplus B(x_n)$ и $x_n = A(k_n) \oplus B(x_n)$ . Таким образом, подставлять можно вместо $k_n$ , $x_n$ либо обоих. (Использование $A(k_n)$ вместо $k_n$ может помешать Эмили восстановить псевдослучайную последовательность при наличии известного открытого текста)
<b>xors</b>	ИСКЛЮЧАЮЩЕЕ ИЛИ, затем подстановка	$x_n = A(k_n \oplus x_n)$
<b>add</b>	Сложение	$x_n = k_n + x_n$ . Как всегда, сложение производится по модулю размера алфавита
<b>madd</b>	Умножить и сложить	Также называется <i>линейной заменой</i> . $x_n = rk_n + x_n$ или $k_n + qx_n$ , или $rk_n + qx_n$ , где $r$ – любое целое число, а $q$ – нечетное число. (Если размер алфавита отличен от 256, то $q$ должно быть взаимно простым с этим размером)
<b>sadd</b>	Подстановка, затем сложение	$x_n = A(k_n) + x_n$ , или $k_n + B(x_n)$ , или $A(k_n) + B(x_n)$
<b>adds</b>	Сложение, затем подстановка	$x_n = A(k_n + x_n)$
<b>poly</b>	Многоалфавитная подстановка общего вида	$x_n = P(k_n, x_n)$

Поскольку операции **xor** и **sxor** могут раскрывать информацию о своих операндах, я рекомендую использовать вместо них **xors**, тогда простая подстановка, выполняемая после ИСКЛЮЧАЮЩЕГО ИЛИ, т. е.  $A(k_n \oplus x_n)$ , маскирует волновые формы.

Для шифрования текущего символа в потоковом шифре может использоваться один или несколько предыдущих символов. Вариантов много. Один из примеров –  $P(k_n \oplus x_{n-i}, x_n)$  для некоторого малого целого числа  $i$ . Для шифрования первых  $i$  символов такому шифру требуется вектор инициализации. Поточковый шифр можно укрепить переключением между несколькими комбинирующими функциями, например периодически переключаясь между тремя формами **sadd** или **madd** или периодически меняя множители  $r$  и  $q$  в **madd**.

## 13.2 Случайные числа

Длинные ключи, применяемые в потоковых шифрах, могут происходить из нескольких источников.

- Список чисел, повторенный нужное число раз. Этот метод был стандартным на протяжении XVI–XIX веков.
- Могут порождаться математическим процессом. Такие числа называются *псевдослучайными*, потому что рано или поздно последовательность начинает повторяться, в отличие от истинно случайных чисел, которые не повторяются никогда. Процесс, порождающий такие числа, называется *генератором псевдослучайных чисел*, ГПСЧ (англ. *pseudorandom number generator* – PRNG).
- Это могут быть истинно случайные числа, порождаемые каким-то физическим процессом, например гамма-излучением взорвавшейся звезды. Такие процессы обычно слишком медленны для криптографических целей, поэтому собираются на протяжении некоторого времени и сохраняются в памяти для последующего использования. То есть они могут собираться непрерывно, а использоваться, только когда нужно отправить сообщение.

В книгах и статьях по криптографии часто пишут, что для безопасного шифра нужны истинно случайные числа. В пользу своего аргумента авторы говорят, что математически доказано, что одноразовый блокнот с истинно случайным ключом невозможно вскрыть. Это, конечно, правда, при условии что для любой единицы открытого текста  $p$  и любой единицы шифртекста  $c$  существует единица ключа  $k$ , которая преобразует  $p$  в  $c$ , т. е.  $S(k, p) = c$ . Истинно случайного ключа достаточно, чтобы сделать одноразовый блокнот невскрываемым. Однако каждому, кто изучал логику, известно, что достаточность условия не означает его необходимости и наоборот.

Например, чтобы целое число было простым, необходимо, чтобы оно было больше 1. Но этого недостаточно, потому что число 4 больше 1, но простым не является. Чтобы целое число было составным, достаточно, чтобы оно было полным квадратом, большим 1. Но это условие не является необходимым, потому что число 6 составное, но не квадрат.

Требовать от ключа одноразового блокнота истинной случайности совершенно излишне. Чтобы одноразовый блокнот был невскрываемым, ключ должен быть *непредсказуемым*, или, как говорят, *криптостойким*. Если ключ истинно случаен, то сколько бы единиц ключа ни знала Эмили, она все равно не сможет определить остальные единицы. От непредсказуемого ключа требуется лишь, чтобы определение дополнительных единиц было вычислительно не осуществимой задачей. Точнее, объем работы, который должна проделать Эмили, чтобы определить еще одну единицу ключа, должен превышать  $2^k$ , где  $k$  – выбранный размер ключа в битах. Да, если поток битов ключа

ча – гамма – лишь псевдослучайный, то математически доказать не-вскрываемость шифра невозможно, но практического значения это не имеет.

Ниже в этой главе я опишу несколько схем обеспечения криптостойкости генераторов псевдослучайных чисел и одну схему, CG5, которая выглядит криптостойкой, но на поверку таковой не является (см. раздел 13.13).

Во всех встречавшихся выше потоковых шифрах для порождения гаммы можно использовать генераторы псевдослучайных чисел, поэтому мы рассмотрим разнообразные ГПСЧ, начиная с классических методов, разработанных в 1950-х годах. В них используется небольшое начальное значение, или *начальное состояние*, и математическая функция, которая генерирует следующее состояние по текущему, называемому *вектором состояния*. В этой функции чаще всего используются сложение, умножение и ИСКЛЮЧАЮЩЕЕ ИЛИ. Эти генераторы применяются и по сей день благодаря высокой скорости и простоте реализации.

Каждый генератор порождает последовательность целых чисел, которая рано или поздно начинает повторяться – период зависит от начального значения. Можно также построить повторяющуюся последовательность, в которой больше никогда не встречается начальное значение, например 1, 2, 3, 4, 5, 4, 5, 4, 5, 4, 5, ..., но ни один из генераторов, рассматриваемых в этой книге, таким поведением не обладает. Период ограничен размером вектора состояния. Например, генератор, для которого вектор состояния состоит из трех 31-битовых целых чисел, не может иметь период длиннее  $2^{93}$ .

### 13.3 Мультипликативный конгруэнтный генератор

У мультипликативного конгруэнтного ГПСЧ два параметра: множитель  $m$  и модуль  $p$ . Отправляясь от начального значения  $s$ , последовательность псевдослучайных чисел  $x_n$  генерируется согласно рекуррентному соотношению

$$\begin{aligned}x_0 &= s, \\ x_n &= mx_{n-1} \bmod p \text{ для } n = 1, 2, 3, \dots\end{aligned}$$

Иными словами, для получения следующего псевдослучайного числа нужно умножить предыдущее число на  $m$  и взять остаток от деления на  $p$ . Начальным значением может быть любое целое число 1, 2, 3, ...,  $p - 1$ . Модуль  $p$  почти всегда выбирается простым, потому что простые числа порождают самые длинные периоды. Выбор  $p$  часто зависит от размера регистров используемого процессора. Если регистры 32-битовые, то обычно выбирают простое число  $2^{31} - 1 =$

2 147 483 647. Первый ГПСЧ этого класса опубликовал в 1949 году специалист по теории чисел из Калифорнийского университета в Беркли Деррик Г. Лемер (не путайте с Дерриком Н. Лемером, тоже из Беркли, – его отцом).

Множитель  $m$  следует выбирать осторожно. Период мультипликативного конгруэнтного генератора может быть любым делителем  $p - 1$ . Поскольку  $p$  простое и, надо думать, значительно больше 2, то  $p - 1$  будет четным, поэтому при очень плохом выборе  $m$ , в частности  $p - 1$ , период мог бы оказаться равным 2. Множитель, имеющий максимально возможный период, а именно  $p - 1$ , называется *первообразным корнем* из  $p$ . Это означает, что при делении на  $p$  все числа  $m, m^2, m^3, \dots, m^{p-1}$  дают различные остатки. Чтобы период мультипликативного конгруэнтного генератора был максимальным, лучше выбрать в качестве  $m$  первообразный корень.

По счастью, это легко сделать. В среднем чуть меньше  $3/8$  чисел в диапазоне от 2 до  $p - 2$  являются первообразными корнями из  $p$ . Точная доля называется постоянной Артина в честь Эмиля Артина, австрийского математика, бежавшего из нацистской Германии в 1937 году и завершившего свою карьеру в Принстоне. Она приближенно равна 0.373956. Если мы умеем раскладывать  $p - 1$  на множители, то легко проверить, является ли данный множитель  $m$  первообразным корнем из  $p$ . Мы знаем, что период  $m$  должен быть делителем  $p - 1$ , поэтому для начала разложим  $p - 1$  на множители. Предположим, что  $a, b, c, d$  – различные простые множители  $p - 1$ . Тогда нужно только проверить числа  $m^{(p-1)/a} \pmod{p}$ ,  $m^{(p-1)/b} \pmod{p}$ ,  $m^{(p-1)/c} \pmod{p}$  и  $m^{(p-1)/d} \pmod{p}$ . Если ни одно из них не равно 1, то  $m$  – первообразный корень. Например, если  $p = 13$ , то различными простыми множителями  $p - 1 = 12$  являются 2 и 3, так что нужно лишь проверить показатели степени  $12/2$  и  $12/3$ , т. е.  $m^6$  и  $m^4$ . Так, 5 не является первообразным корнем из 13, потому что  $5^4 = 625 \equiv 1 \pmod{13}$ .

Имеются эффективные способы вычислить  $m^x$  путем последовательного возведения в квадрат. Например, чтобы вычислить  $m^{21}$ , можно было бы последовательно вычислить  $m^2, m^4, m^8, m^{16}, m^{20}, m^{21}$ , выполнив всего 6 умножений. Можно еще повысить эффективность, если использовать ранее вычисленные произведения для вычисления следующей степени. Например, если следующим предстоит проверить значение  $m^{37}$ , то можно было бы вычислить  $m^{32}, m^{36}, m^{37}$ , выполнив все 3 умножения. Более эффективно производить все вычисления по модулю  $p$ , а не вычислять огромное число  $m^{21}$  и брать остаток от деления в самом конце. Существуют и более сложные схемы, в которых количество умножений на 10–15 % меньше, но если это делается всего несколько раз, то дополнительные усилия не оправданы.

При использовании мультипликативного конгруэнтного ГПСЧ важно понимать, что именно сами вычисленные им числа обладают свойствами случайности. Чтобы преобразовать выход  $R$  генератора в целое число в диапазоне от 0 до  $N-1$ , нужно вычислить  $[RN/p]$ , где

$\lfloor x \rfloor$ , «целая часть  $x$ », или «пол  $x$ » – результат округления  $x$  с недостатком до ближайшего целого числа. Например,  $\lfloor 27 \rfloor = 27$  и  $\lfloor 27.999 \rfloor = 27$ . Выражение  $\lfloor RN/p \rfloor$  несколько смещено в сторону меньших значений, т. е. оно дает меньшие значения чаще, чем большие. Однако когда  $p$  много больше  $N$ , скажем  $p > 1000N$ , это несущественно с точки зрения криптографии.

### Историческое отступление

Кстати говоря, обозначения  $\lfloor x \rfloor$  и  $\lceil x \rceil$  («потолок  $x$ », равный результату округления  $x$  с избытком до ближайшего целого, так что  $\lceil 27.001 \rceil = 28$ ) предложил Кеннет Иверсон, создавший язык программирования APL, в 1962 году. APL стал первым интерактивным языком программирования. В наши дни интерактивность считается само собой разумеющейся. Вы нажимаете клавишу или щелкаете мышкой, и компьютер что-то делает. Никто не задумывается о том, что эту идею кто-то должен был изобрести. А до этого стандартная модель использования компьютера была иной – подать колоду перфокарт устройству чтения и через несколько часов получить стопку бумаги с напечатанными результатами.

**ПРЕДУПРЕЖДЕНИЕ** Не используйте  $R \bmod N$  в качестве случайного числа.  $R \bmod N$  может быть значительно смещено в сторону меньших значений. Например, если модуль  $p = 11$  и  $N = 7$ , то  $R \bmod 7$  принимает 11 значений: 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, т. е. 0, 1, 2 и 3 генерируются в два раза чаще, чем 4, 5 и 6.

Мультипликативный конгруэнтный генератор будет обладать хорошими свойствами случайности, если  $m > \sqrt{p}$ . Лучше, если еще и мультипликативный обратный элемент  $m' > \sqrt{p}$ . Это означает, что число битов  $m$  должно быть не меньше половины числа битов  $p$ . Мы хотим, чтобы  $p$  было как можно больше, чтобы у генератора был длинный период и чтобы  $m$  было большим – тогда генератор будет порождать случайные числа. Насколько большим? Размеры  $m$  и  $p$  ограничены размером регистров процессора. Если превысить размер регистра, то придется расплачиваться скоростью работы.

Каждое псевдослучайное число  $x_n$  генерируется путем умножения предыдущего числа  $x_{n-1}$  на  $m$ . Число  $x_{n-1}$  может иметь столько же битов, сколько  $p$ , т. е. если  $p$  состоит из  $b$  бит, то и  $x_{n-1}$  тоже может состоять из  $b$  бит. Поскольку  $m$  должно иметь по меньшей мере  $b/2$  бит, произведение  $mx_{n-1}$  может иметь  $3b/2$  бит. Если размер регистра 63 бита, то  $b$  не может превышать  $2/3$  от 63, т. е. 42, а значит,  $m$  может содержать не более 21 бита. Хорошо бы сделать  $m$  больше, чем  $\sqrt{p}$ . Разумный компромисс – 25 бит для  $m$  и 38 бит для  $p$ . Тогда период мог бы доходить до  $2^{38}$ .

Чтобы генератор был непредсказуемым, необходимо, чтобы были одинаковы частоты сгенерированных единиц, частоты пар единиц,



троек, четверок и т. д. На практике не имеет смысла заходить дальше восьмерок, в крайнем случае десятков байт. Если требуется абсолютная уверенность, поделите размер ключа на размер сгенерированных единиц. Например, если размер ключа 128 бит и ППСЧ порождает 4-битовые шестнадцатеричные цифры, то можете потребовать, чтобы была одинакова частота  $n$ -кортежей для всех  $n$ , не больших 32. (Но человек, который и вправду так поступит, очевидно, страдает навязчивым неврозом и нуждается в лечении.) Даже для 4-битовых случайных чисел стремление идти дальше кортежей, содержащих 16, максимум 20 элементов (т. е. 64 или 80 бит), нельзя назвать ни необходимым, ни полезным.

Эмили понадобится более  $2^{64}$  (соответственно  $2^{80}$ ) байт открытого текста, чтобы воспользоваться неравными частотами. Даже если Сандра никогда не меняет ключ, все равно Эмили вряд ли удастся набрать столько материала. Приведу конкретный пример, чтобы было понятнее: допустим, что спутник передает на землю телеметрию по лучу со скоростью 1 МБ/с. Допустим также, что данные передаются с использованием двух разных гамм одновременно и что Эмили знает ключ одной из них. Так вот, притом что она получает пары открытый текст / шифртекст со скоростью 1 МБ/с, чтобы набрать  $2^{64}$  байт, ей понадобится 585 000 лет. Даже если данные передают 1000 спутников и все пользуются одним и тем же ключом, все равно понадобится 585 лет.

Если частоты  $n$ -кортежей одинаковы для всех значений  $n$ , то ваш генератор истинно случайный. Вы изобрели математический алгоритм генерирования истинно случайных чисел. Поздравляю. Отправляйтесь за филдсовской премией.

Чтобы частоты кортежей были равны для кортежей вплоть до длины  $n$ , в общем случае необходимо, чтобы начальное значение само было кортежем длины не менее  $n$ . Для мультипликативных конгруэнтных генераторов частоты одиночных единиц и пар единиц распределены равномерно, но для частот троек это уже не так, а при  $n > 3$  распределение очень далеко от равномерного: большинство частот равны 0.

Вскрыть равномерный конгруэнтный шифр нетрудно, если известно несколько символов открытого текста и шифр легко позволяет определить случайный выход по паре открытый текст / шифртекст, т. е. если в качестве комбинирующей функции используется **xor**, **add** или **madd**. Например, если для получения байта шифртекста к байтам открытого текста и ключа применяется ИСКЛЮЧАЮЩЕЕ ИЛИ, то Эмили нужно только применить ИСКЛЮЧАЮЩЕЕ ИЛИ к байтам открытого и шифртекста, чтобы получить байт ключа.

Если модуль генератора состоит из 31 или 32 бит, то Эмили может перебрать все  $2^{31}$  или  $2^{32}$  возможных начальных значений; это осуществимо даже на ПК. Известные символы открытого текста понадобились бы только для проверки. Если модуль больше, скажем 48 или 64 бита, то первые 2 или 4 известных символа открытого текста

используются, чтобы ограничить область поиска. Первый случайный выход ограничивает текущее состояние генератора узким диапазоном, составляющим  $1/256$  часть от полного диапазона. Вторым известным символом открытого текста дает второй выход, который составляет  $1/256$  этого диапазона и т. д.

Таким образом, простой мультипликативный конгруэнтный генератор не является криптостойким. Можно взять значительно больший модуль и использовать методы перемножения больших целых чисел, например Карацубы или Тоома–Кука, но тогда пришлось бы пожертвовать быстродействием, свойственным этому классу генераторов. Существуют более быстрые способы создания криптостойких генераторов, поэтому мы здесь не будем рассматривать методы умножения больших целых чисел.

## 13.4 Линейный конгруэнтный генератор

*Линейный конгруэнтный генератор* – обобщение мультипликативного конгруэнтного генератора. Он добавляет постоянный член в рекуррентную формулу. Отправляясь от начального значения  $s$ , последовательность случайных чисел  $x_n$  генерируется следующим рекуррентным соотношением:

$$\begin{aligned}x_0 &= s, \\ x_n &= (mx_{n-1} + c) \bmod P \text{ для } n = 1, 2, 3, \dots\end{aligned}$$

Иными словами, чтобы получить следующее псевдослучайное число, мы умножаем предыдущее число на  $m$ , прибавляем  $c$ , а затем берем остаток от деления суммы на  $P$ . Начальным значением может быть любое целое число  $1, 2, 3, \dots, P - 1$ . Период генератора будет максимальным, если выполнены три условия:

- 1  $c$  и  $P$  – взаимно простые числа;
- 2 для любого простого  $p$ , являющегося делителем  $P$ ,  $m$  имеет вид  $pk+1$ ;
- 3 если  $P$  кратно 4, то  $m$  имеет вид  $4k+1$ ,

где  $k$  – целое число. Это так называемые *условия Халла–Добелла*, именованные так в честь Т. Э. Халла и А. Р. Добелла из университета Британской Колумбии, которые опубликовали их в 1962 году.

Например, предположим, что  $P = 30$ , т. е. раскладывается в произведение простых чисел  $2 \times 3 \times 5$ . Тогда  $m - 1$  должно быть кратно 2, 3 и 5. Иначе говоря,  $m$  должно быть равно 1. Поэтому если  $s = 1$  и  $c = 7$ , то последовательность псевдослучайных чисел будет иметь вид 1, 8, 15, 22, 29, .... Это арифметическая прогрессия, в которой нет ничего случайного. По этой причине в качестве модуля  $P$  обычно выбирается степень простого числа, чаще всего двойки. Трудно найти значения  $m$ ,  $c$  и  $P$ , порождающие хорошие свойства случайности.

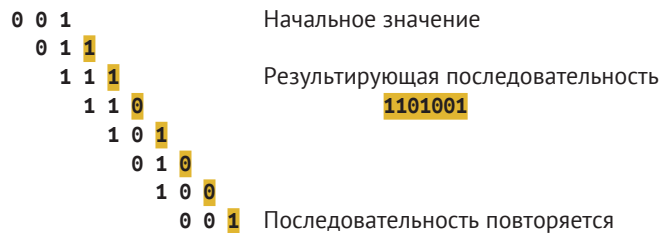
Однако у линейных конгруэнтных генераторов есть одно полезное применение. Если нужен генератор с очень длинным периодом, то можно сложить выходы двух или более линейных конгруэнтных генераторов, модулями которых являются степени различных простых чисел. Так мы получим генератор с хорошими свойствами случайности и периодом, равным произведению модулей. Например, предположим, что мы сложили выходы следующих трех ГПСЧ. Я выбрал все три модуля максимально большими, но еще уместяющимися в 32-битовое машинное слово, а множители и постоянные – удовлетворяющими условиям Халла–Добелла. В остальном их выбор произволен.

$$\begin{aligned}x_{n+1} &= (10000001x_n + 1234567) \bmod 2^{31}, \\y_{n+1} &= (1212121y_n + 7654321) \bmod 3^{19}, \\z_{n+1} &= (43214321z_n + 777777) \bmod 5^{13}.\end{aligned}$$

Пусть  $w_n = (x_n + y_n + z_n) \bmod 2^{31}$ . Выберем старший байт  $w_n$ , сдвинув его вправо на 23 позиции, т. е.  $v_n = w_n / 2^{23}$ . Последовательность  $v_n$  обладает хорошими свойствами случайности, при условии что (1) по меньшей мере один из трех множителей и мультипликативно обратный ему больше квадратного корня из соответствующего ему модуля и (2) ни один из остальных двух множителей не равен 1 или  $P - 1$ . Период последовательности  $v_n$  равен  $2^{31}3^{19}5^{13} = 3.0468 \times 10^{27}$ .

## 13.5 Цепной XOR-генератор

Простейший *цепной XOR-генератор* работает со строкой битов, например 10111. Основная идея такова: применить ИСКЛЮЧАЮЩЕЕ ИЛИ к первому и последнему битам, удалить первый бит и добавить новый бит в конец строки, т. е.  $x_i = x_{i-1} \oplus x_{i-n}$ . Поскольку  $n$ -битовая строка может принимать  $2^n$  значений и поскольку строка, состоящая только из нулей, порождает последовательность нулей, то самый длинный период цепного XOR-генератора равен  $2^n - 1$ . Рассмотрим простой пример с 3-битовыми строками.



После семи шагов начальная строка 001 повторяется, так что период этого генератора равен 7. Такой генератор называется *полно-*

периодным. Цепной XOR-генератор имеет полный период при  $n = 2, 3, 4, 6, 7, 15, 22$ . При  $n = 37$  генератору не хватает 0.00057 % до полного периода. То есть 99.99943 % всех 37-битовых значений образует один большой цикл, а остальные принадлежат коротким циклам. Для каких-то целей  $n = 37$  может оказаться хорошим выбором. Для большинства значений  $n$  имеется несколько повторяющихся последовательностей битов, одни короткие, другие длинные. Сумма их длин равна  $2^n - 1$ . О единственном периоде можно говорить только в случае полнопериодных генераторов. В противном случае будет несколько циклов разной длины.

Допустим, что нам нужен генератор с периодом длиннее  $2^{22}$ , но мы не хотим допускать даже 0.00057-процентного шанса получить более короткий цикл. Что делать? Один из вариантов – попробовать другие генерирующие функции. Вместо  $x_i = x_{i-1} \oplus x_{i-n}$  попробуем рекуррентное соотношение  $x_i = x_{i-1} \oplus x_{i-j} \oplus x_{i-k} \oplus x_{i-n}$  для значений  $j$  и  $k$  таких, что  $1 < j < k < n$ . Есть неплохие шансы на то, что некоторые из этих генераторов будут иметь полный период. Отметим, однако, что соотношение  $x_i = x_{i-1} \oplus x_{i-j} \oplus x_{i-n}$  с тремя членами не может породить полнопериодный генератор. Количество членов должно быть четным.

Какой бы генератор ни выбрать, результатом является последовательность битов. Чтобы получить псевдослучайную последовательность байтов, будем брать группы битов, т. е. биты 1–8, 9–16, 17–24 и т. д. Следовательно, нужно сгенерировать 8 бит для каждого байта. Но есть более быстрый способ. Вместо того чтобы применять операцию ИСКЛЮЧАЮЩЕЕ ИЛИ к одиночным битам, будем применять ее к байтам. По существу, мы параллельно выполняем 8 отдельных битовых генераторов. Так мы получаем целый байт за одну операцию. Если язык программирования поддерживает, можно использовать 32-битовые слова и получить 4 байта за раз.

Любую из комбинирующих функций, перечисленных в разделе 13.1, можно использовать для комбинирования псевдослучайного потока с открытым текстом и таким образом получить шифр. Если Сандра выберет функцию **xor**, **add** или **madd**, то Эмили легко вскроет шифр при наличии достаточного объема открытого текста. Она без труда определит случайные выходы, соответствующие символам открытого текста. Это позволит реконструировать участок гаммы. Затем этот участок можно будет расширить вперед и назад и реконструировать всю гамму, просто выполнив ИСКЛЮЧАЮЩЕЕ ИЛИ.

Но есть прием, позволяющий Сандре поставить Эмили в тупик. Предположим, что генератор порождает последовательность 32-битовых слов, которые Сандра разбивает на четыре отдельных байта. Вместо того чтобы всегда начинать со старшего бита, Сандра могла бы каждый раз начинать с другой позиции. Эквивалентно, Сандра могла бы сдвигать 32-битовое слово циклически влево или вправо и менять величину сдвига. Например, ABCDEF после циклического сдвига на две позиции становится равным CDEFAB. Величины сдви-

гов могут образовывать периодическую последовательность чисел от 0 до 31. В результате Эмили не сможет сопоставить последовательные выходы генератора и реконструировать гамму.

## 13.6 Цепной аддитивный генератор

*Цепные аддитивные генераторы*, называемые также *генераторами Фибоначчи с запаздыванием*, похожи на цепные XOR-генераторы, только вместо ИСКЛЮЧАЮЩЕГО ИЛИ используется операция сложения. Сложение выполняется по модулю  $2^w$ , где  $w$  – размер слова в битах:  $x_i = (x_{i-1} + x_{i-n}) \bmod 2^w$ . Типичные значения  $w$  – 15, 31, 63 в случае сложения со знаком и 16, 32, 64 в случае сложения без знака. По-другому рассматривать операцию по модулю  $2^w$  можно как игнорирование переноса из старшего разряда.

Поскольку при сложении возникает перенос из одного разряда в следующий, период старшего бита в два раза больше периода младшего. Период младшего бита в каждом слове совпадает с периодом XOR-генератора с теми же начальными значениями. Причина в том, что сложение – то же самое, что ИСКЛЮЧАЮЩЕЕ ИЛИ с переносом. Если период младшего бита в цепном аддитивном генераторе равен  $P$ , то период старшего бита равен  $2^{w-1}P$ .

Цепные аддитивные генераторы – простой способ получить более длинный период ценой небольших дополнительных усилий. Просто найдите цепной XOR-генератор с длинным периодом, хорошо бы полнопериодный, а затем увеличьте ширину с одного бита до полного слова. Как и в случае мультипликативных конгруэнтных генераторов, наиболее случайная часть последовательности выходов – старшая. В качестве последовательности псевдослучайных байтов используйте только восемь старших бит каждого слова.

И снова для объединения псевдослучайного потока с открытым текстом и, следовательно, порождения шифра можно использовать любую из комбинирующих функций, описанных в разделе 13.1.

## 13.7 Сдвиговый XOR-генератор

Еще один класс ГПСЧ – сдвиговые XOR-генераторы, придуманные Джорджем Марсалье из Флоридского университета, который известен прежде всего разработкой комплекта тестов на случайность Diehard. В этих генераторах используются два оператора для работы с целыми числами:

- $\ll$  *сдвиг влево*. Например,  $80 \ll 2$  сдвигает целое число 80 на два бита влево, так что получается значение 320;
- $\gg$  *сдвиг вправо*. Например,  $80 \gg 2$  сдвигает целое число 80 на два бита вправо, так что получается значение 20.

Биты, выдвигаемые из старшего и младшего разрядов машинного слова, теряются. Например,  $25 \gg 1$  равно 12, а не 12.5. Сравните эти операции с циклическими сдвигами  $\lll$  и  $\ggg$ , при которых биты, выдвигаемые с одного конца, вдвигаются в другой. Например, если 32-битовое машинное слово состоит из шестнадцатеричных цифр 12345678, то  $12345678 \lll 4 = 23456781$ , а  $12345678 \ggg 12 = 67812345$ , поскольку каждая шестнадцатеричная цифра содержит 4 бита. Если слово хранится в более широком регистре процессора, то неиспользуемые биты обнуляются.

В этом классе имеется несколько разных генераторов. Величины и направления сдвигов следует тщательно выбирать, чтобы генератор имел длинный период. Ниже приведено два примера генератора *Xorshift*, предложенного Марсальей. Они имеют длинные периоды, обладают хорошими свойствами случайности, хотя не проходят некоторые особо требовательные тесты на случайность. В каждом генераторе для порождения следующего члена последовательности выполняется три комбинированных шага ИСКЛЮЧАЮЩЕГО ИЛИ и сдвигов влево-вправо-влево. В переменной  $y$  хранятся промежуточные значения. Любое положительное число может служить подходящим начальным значением.

32-битовый генератор. Период  $2^{32}-1$ .

$$\begin{aligned} y &= x_n \oplus (x_n \lll 13) \\ y &= y \oplus (y \ggg 17) \\ x_{n+1} &= y \oplus (y \lll 5) \end{aligned}$$

64-битовый генератор. Период  $2^{64}-1$ .

$$\begin{aligned} y &= x_n \oplus (x_n \lll 13) \\ y &= y \oplus (y \ggg 7) \\ x_{n+1} &= y \oplus (y \lll 17) \end{aligned}$$

## 13.8 FRand

Генератор FRand, или Fast Random (быстрый генератор случайных чисел), – мое собственное изобретение. В нем используется массив  $S$  двоичных слов ширины  $W$ , т. е. в  $W$  младших битах каждого слова массива хранится целое число без знака. Период зависит от значений  $S$  и  $W$ . Я обнаружил, что наилучшие результаты дает  $W = 29$ , а при  $S = 40$  и  $S = 64$  получаются очень длинные периоды. Массив начальных значений можно рассматривать как битовую матрицу  $40 \times 29$ . В каждой строке хранится одно начальное слово, а каждый столбец представляет один бит начального слова.

При  $S = 40$  период равен  $2^{1160} - 2^{40} \approx 1.566 \times 10^{349}$  для подходящих начальных значений. Начальное значение является *подходящим*, если по крайней мере одно из 40 начальных слов не состоит из одних нулей или одних единиц. У этого генератора есть слабое место. Если

начальный массив состоит почти целиком из нулей, то генератор может порождать десятки и даже сотни последовательных выходов, в основном равных нулю. В экстремальном случае, когда начальный массив содержит 1159 нулей и всего одну единицу, понадобится минимум 1120 циклов, прежде чем в каждом столбце появится хотя бы одна единица.

Желательно, чтобы начальные значения содержали много единиц и нулей без явных закономерностей. Чтобы получить пригодный массив, можно, например, взять мнемонический или числовой ключ, записанный в кодировке UTF-8, и вычислить на его основе 1160-битовый хеш. Подходящая функция хеширования имеет вид:

$$\begin{aligned} x_1 &= x_1 + 19x_{40} \bmod 2^{29}, \\ x_n &= x_n + 19x_{n-1} \bmod 2^{29} \text{ для } n = 2, 3, 4, \dots, 40 \text{ (первый полный проход),} \\ x_1 &= x_1 + 19x_{40} \bmod 2^{29}, \\ x_n &= x_n + 19x_{n-1} \bmod 2^{29} \text{ для } n = 2, 3, 4, \dots, 10 \text{ (второй частичный проход).} \end{aligned}$$

После инициализации генератора псевдослучайная последовательность генерируется по следующей рекуррентной формуле:

$$\begin{aligned} n &= n + 1 \\ x_n &= x_n \oplus x_{n-1} \end{aligned}$$

В конце каждого прохода по начальному массиву, когда  $n$  принимает значение 40, индекс сбрасывается в 1 и генерируется следующее псевдослучайное число  $x_1 = (x_1 \oplus x_{40}) \ggg 1$ . То есть первое 29-битовое слово  $x_1$  циклически сдвигается на одну позицию вправо.

Эта псевдослучайная последовательность проходит многие тесты на случайность, но немного не дотягивает до криптостойкости. Чтобы последовательность была безопасной, нужно брать последовательные выходные байты из разных частей 29-битового слова. И для выбора этой части можно использовать саму псевдослучайную последовательность. Предположим, что следующие три псевдослучайных выхода равны  $a$ ,  $b$  и  $c$ . Положим  $s = a \bmod 25$ . Если  $s$  принадлежит диапазону от 0 до 21, то сдвинем  $b$  вправо на  $s$  позиций и возьмем младшие 8 бит. В этом случае генерируются только  $a$  и  $b$ . Что касается  $c$ , то оно будет сгенерировано для следующего псевдослучайного числа. Если  $s > 21$ , то сдвиг на  $s$  позиций вправо оставил бы меньше 8 бит. В этом случае игнорируем  $a$  и берем  $s = b \bmod 22$ . Сдвигаем  $c$  вправо на  $s$  позиций и берем 8 младших бит в качестве случайного выхода. В виде псевдокода описанные действия выглядят так:

```
s = xn+1 mod 25
if s ≤ 21 then
    r = (xn+2 >> s) and FF (младшие 8 бит r)
else
    s = xn+2 mod 22
    r = (xn+3 >> s) and FF (младшие 8 бит r)
```



В этой процедуре для порождения каждого безопасного бита ключа используется в среднем 2.12 псевдослучайного выхода. Байт ключа в половине случаев происходит из выходов с нечетными номерами, а в половине – с четными. Генератор переключается между четом и нечетом примерно раз в 8 циклов, нерегулярно.

## 13.9 Вихрь Мерсенна

Вихрь Мерсенна – ГПСЧ с самым длинным периодом. Его разработали в 1997 году Макото Мацумото и Такудзи Нишимура из Хиросимского университета. Назван он в честь французского теолога Марина Мерсенна (1588–1648), хорошо известного своей работой по простым числам вида  $2^n - 1$  и сыгравшего важную роль в распространении работ Галилея, Декарта, Паскаля, Ферма и других ученых.

Вихрь характеризуется весьма достойными свойствами случайности, хотя и не проходит некоторые тесты на случайность. Он гораздо медленнее других генераторов случайных чисел, описанных в этой главе. Важен он прежде всего своим гигантским периодом, равным простому числу Мерсенна  $2^{19937} - 1$ , открытому в 1971 году Брайантом Такерманом из IBM Research, Йорктаун, штат Нью-Йорк. Лаборатория IBM Research была так горда этим открытием, что поместила надпись « $2^{19937} - 1$  is prime» на фирменные канцелярские принадлежности и на свой почтовый штампель.

Как и FRand, вихрь Мерсенна демонстрирует слабость, если начальное значение почти целиком состоит из нулей. В этом случае до получения байтов, похожих на случайные, может пройти много циклов. Обычно требуется пропустить первые 10 000 или даже 50 000 циклов, прежде чем использовать порождаемые байты для дела. С другой стороны, в пакете FRand имеется функция, которая инициализирует генератор, не требуя холостого выполнения начальных циклов.

## 13.10 Регистры сдвига с линейной обратной связью

Регистр сдвига с линейной обратной связью (РСЛОС, англ. LFSR) – любимец инженеров-электротехников, поскольку его очень просто реализовать в виде цифровой схемы. В РСЛОС используется массив битов  $x_1, x_2, \dots, x_n$ . Следующий бит генерируется применением ИСКЛЮЧАЮЩЕГО ИЛИ к нескольким предыдущим, например:

$$x_{n+1} = x_n \oplus x_{n-i} \oplus x_{n-j} \oplus x_{n-k}$$



с тремя обратными связями. Количество обратных связей, конечно, необязательно должно быть равно 3, но для нечетных значений период обычно получается гораздо длиннее, чем для четных.

У этого РСЛОС было бы  $k+1$  бит в предположении, что  $i < j < k$ . После генерирования каждого нового бита младший бит выдвигается, а новый помещается на место старшего, поэтому регистр в каждый момент времени содержит последние  $k+1$  бит псевдослучайной последовательности.

Очевидный недостаток РСЛОС – неспешность, потому что для порождения каждого псевдослучайного байта им требуется 8 циклов. РСЛОС также самые слабые из генераторов псевдослучайных чисел в силу своей линейности. Если Эмили известен какой-то открытый текст и если она сможет определить соответствующие биты ключа, то сможет реконструировать всю псевдослучайную последовательность, просто решив систему линейных уравнений. А определить биты ключа Эмили сможет, если Сандра использовала комбинирующую функцию **xor**, **add** или **madd**.

По этой причине псевдослучайные выходы обычно пропускаются через нелинейную подстановку и только потом объединяются с открытым текстом. Это можно делать побитово или побайтово. Нелинейные побитовые подстановки возможны, потому что на каждом цикле в регистре доступно  $k+1$  бит. Биты, используемые как входы нелинейной функции, называются *отводами* и могут браться из любого места регистра. Использование нелинейных функций затрудняет Эмили определение битов ключа.

На эту роль подходит, в частности, *мажоритарная функция*. Она принимает значение 1, если большинство входных битов равны 1, и 0 в противном случае. Если имеется 3 входных бита A, B и C, то мажоритарная функция имеет вид  $AB \vee BC \vee CA$ , где  $\vee$  – булева функция ИЛИ. Мажоритарная функция определена для любого нечетного числа входов, 3, 5, 7, .... Возможное развитие этой идеи – использовать девять отводов и три схемы 3-битовых мажоритарных функций. На входы каждой схемы подается три бита из девяти. После этого три выходных бита пропускаются через четвертую мажоритарную функцию.

Побайтовые подстановки естественно возникают, если в качестве комбинирующей функции используется **sxor**, **sadd** или **poly**. Построение таких нелинейных подстановок подробно обсуждается в разделе 12.3. Можно объединить побитовые и побайтовые подстановки. Каждый из восьми бит выходного байта генерируется с помощью отводов и нелинейной битовой функции, а затем все восемь однобитовых выходов этих схем подаются на вход побайтовой подстановки.

Подумаем, что должна сделать Эмили, чтобы вскрыть шифр РСЛОС. Предположим, что Сандра использует 40-битовый аппаратный РСЛОС и отводы из позиций 3, 6 и 9 подаются на вход мажоритарной схемы M. Предположим также, что по наивности она исполь-

зовала для комбинирования функцию **xor**. Допустим, что Эмили знает несколько символов открытого текста, а значит, и последовательность выходных битов. Для каждого известного бита множество значений трех отводов, подаваемых на вход *M*, сужается до четырех из восьми возможных. Если бит равен 0, то три отвода должны давать 000, 001, 010 или 100, а если бит равен 1, то 011, 101, 110 или 111.

После четырех циклов в три отвода было подано 12 бит, и для них существует  $4^4 = 256$  возможных комбинаций. Это значимое уменьшение по сравнению с общим числом комбинаций  $2^{12} = 4096$ . Более того, с точки зрения Эмили, бит, который первоначально находился в позиции 3, теперь занимает позицию 6, а бит, находившийся в позиции 6, переместился в позицию 9. Это значит, что некоторые из 12-битовых комбинаций можно исключить. Если они различны, то исключается больше комбинаций. Каждый дополнительный известный выходной бит еще больше уменьшает число возможных комбинаций битов в регистре сдвига.

Пример поможет разобраться. Предположим, что Сандра использует 40-битовый РСЛОС с тремя отводами, которые подаются на вход мажоритарной функции для порождения выходного бита. Предположим также, что Эмили все знает об устройстве и знает, что сообщение отправлено из генерального штаба и, стало быть, начинается с букв *GHQ* (General Headquarters). Это дает ей 24 бита известного открытого текста. Применив к ним и соответствующим битам шифртекста ИСКЛЮЧАЮЩЕЕ ИЛИ, она получит 24 выходных бита устройства. Для каждого из них имеется 4 возможные комбинации трех входных битов, порождающие известное значение. Всего получается 72 возможных значения битов в трех позициях отводов. Поскольку биты РСЛОС на каждом цикле сдвигаются на одну позицию, часть этих комбинаций битов может совпадать, поэтому число различных комбинаций меньше.

Какие уроки может извлечь Сандра из этого краткого анализа? (1) Регистр сдвига нужно делать длинным, желательно не меньше 128 бит. (2) Отводы должны далеко отстоять друг от друга. (3) Не следует располагать отводы равномерно. Наш выбор битов 3, 6, 9 на редкость неудачен. (4) Использовать комбинирующую функцию, которая затрудняет противнику определение битов ключа. Не использовать **xor**, **add** или **madd** в качестве комбинирующей функции. Лучше брать **xors** и **adds**, а самый лучший выбор – **poly**.

## 13.11 Оценивание периода

Если вы криптограф-любитель, то, возможно, захотите спроектировать собственный генератор псевдослучайных чисел. В этой книге ничего не говорится о том, как тестировать ГПСЧ, потому что это обширная тема, но давайте хотя бы посмотрим, как можно оценить пе-

риод вашего генератора. Метод зависит от размера вектора состояния (раздел 13.2).

Если вектор состояния мал, скажем 31 бит, то можно просто выполнить  $2^{31}$  циклов и посмотреть, когда начнется повторение. К сожалению, может случиться, что начальное значение никогда не повторится. Но и на этот случай есть прием. Создайте два экземпляра своего ГПСЧ и инициализируйте их одним и тем же начальным значением  $S$ . Затем выполняйте первый экземпляр по одному шагу за раз, а второй – по два шага. Предположим, что после 3000 циклов оба экземпляра пришли к одному и тому же вектору состояния. Это означает, что  $R_{3000} = R_{6000}$ , так что период вашего генератора равен 3000, по крайней мере с начальным значением  $S$ .

Если вектор состояния больше, скажем 64 бита, то выполнить  $2^{64}$  циклов генератора практически невозможно. Но все равно можно оценить период по выборке. Создадим таблицу, содержащую, к примеру,  $T = 1\,000\,000$  элементов. В  $N$ -м элементе хранится номер цикла, в котором генератор порождает значение  $N$ . В начальный момент все элементы равны нулю, поскольку еще не было порождено ни одного значения. Выберем начальное значение в диапазоне от 1 до  $T - 1$  и выполним сколько-то циклов генератора, скажем  $G = 1\,000\,000\,000$ . Если порожденное в цикле значение  $N$  меньше  $T$ , то записываем порядковый номер цикла в  $N$ -й элемент таблицы. Если находившееся в этом элементе значение не равно 0, значит, имеет место повторение и мы знаем период. Например, если значение 12795 порождается в цикле 33000, а затем в цикле 73500, то период генератора для данного начального значения равен  $73500 - 33000 = 40500$ .

Если не найдено ни одного повторения, то можно оценить период, исходя из того, сколько из  $T$  значений было порождено. Если  $E$  элементов таблицы отлично от нуля, то доля порожденных элементов равна  $E/T$ . Поскольку было выполнено  $G$  циклов генератора, период можно оценить как  $G/(E/T) = GT/E$ .

При обсуждении цепного генератора цифр (раздел 4.5.1) мы видели, что генератор может иметь разные циклы – как короткие, так и длинные. Необходимо оценить период генератора несколько раз, с разными начальными значениями. Можно порекомендовать такую стратегию. Сначала берем начальное значение 1. В качестве второго начального значения берем наименьшее число, не сгенерированное при первой попытке. Третьим начальным значением будет наименьшее число, не сгенерированное ни при первой, ни при второй попытке. Чтобы реализовать эту стратегию, нужно просто не обнулять таблицу между попытками. Если при 20–100 попытках оценки периода получаются близкими, есть уверенность, что для большинства начальных значений генератор имеет длинный период.

## 13.12 Укрепление генератора

Один из методов укрепления ГПСЧ – использовать *генератор выбора*, который отделяет операцию генерирования чисел от операции выбора чисел. Для этого будем хранить в массиве  $N$  чисел, например 32, 64 или 256. Разрядность всех чисел должна совпадать с разрядностью желаемых случайных выходов. Например, если вы хотите генерировать случайные байты, то массив должен содержать 8-битовые числа. Сначала выполняется  $N$  циклов ГПСЧ для порождения начальных чисел, которые помещаются в массив в том порядке, в котором сгенерированы. Затем ГПСЧ перезапускается с новым начальным значением и используется для порождения последовательности псевдослучайных чисел в диапазоне от 1 до  $N$ , которые служат для выбора элемента массива. Выбранный элемент становится следующим псевдослучайным числом и заменяется новым числом, порожденным ГПСЧ.

Это означает, что первое, третье, пятое, ... случайные числа используются для выбора, а второе, четвертое, шестое, ... – для замены чисел в массиве. Хотя использовать два экземпляра одного и того же ГПСЧ с разными начальными значениями удобно, период при этом не увеличится. Улучшенная стратегия – использовать два разных генератора с взаимно простыми периодами. Тогда период объединенного генератора будет равен произведению периодов. Например, если числа генерируются мультипликативным конгруэнтным генератором с периодом  $2^{31} - 1$ , а выбираются линейным конгруэнтным генератором с периодом  $2^{31}$ , то период объединенного генератора будет равен  $2^{62} - 2^{31} \approx 4.612 \times 10^{18}$ .

Период  $4.612 \times 10^{18}$  достаточно длинный для криптографических применений, но генератор с выбором все еще не является криптостойким, потому что Эмили может перебрать всю последовательность селектора и опробовать все  $2^{31}$  начальных значений. При наличии достаточного объема открытого текста это могло бы дать ей последовательность выходов первого генератора, и этого должно хватить для его вскрытия.

Исправить ситуацию можно несколькими способами. (1) Использовать комбинирующую функцию **xors**, **adds** или **poly**, это затруднит Эмили задачу определения случайных выходов. (2) Сделать селектор (генератор выбора) больше, скажем 63 бита вместо 31. (3) Сделать начальное значение селектора больше, например увеличив множитель и (или) аддитивную постоянную, т. е.  $m$  и  $c$  в порождающей функции  $x_{n+1} = (mx_n + c) \bmod P$ . (4) Использовать описанные в следующем разделе методы для конструирования селектора с более длинным периодом.

## 13.13 Комбинирование генераторов

Генераторы псевдослучайных чисел можно комбинировать разными способами для получения более длинных периодов, улучшения свойств случайности или повышения криптостойкости. Обычно эти улучшения сопровождают друг друга. Не нужно жертвовать одним, чтобы добиться другого. При увеличении периода, как правило, улучшается и случайность. Существует два класса комбинированных генераторов: фиксированные и переменные комбинации.

### ФИКСИРОВАННЫЕ КОМБИНАЦИИ

В случае фиксированной комбинации имеется несколько ГПСЧ, желательно со взаимно простыми периодами. Это могут быть мультипликативные конгруэнтные, линейные конгруэнтные или XOR-генераторы со сдвигом. Выходы генераторов можно объединять побитово или побайтово. Один из побитовых методов – взять фиксированное количество битов от каждого генератора и подать их на вход какой-то комбинирующей функции. Например, можно взять от каждого из восьми генераторов старший бит или от каждого из четырех генераторов два старших бита. Затем к этим 8 битам применяется сильно нелинейная подстановка. Шаг подстановки мешает Эмили разделить выходы генераторов и вскрыть их по отдельности.

Пример побайтового метода – взять старшие байты от каждого генератора и объединить их путем сложения по модулю 256 или применения ИСКЛЮЧАЮЩЕГО ИЛИ. Два генератора можно скомбинировать, перемножив их выходы и взяв средние 8 бит произведения. Еще один подход – взять линейную комбинацию вида  $(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4) \bmod 256$ , где  $x_1, x_2, x_3$  и  $x_4$  – 8-битовые выходы четырех ГПСЧ, а коэффициенты  $a_1, a_2, a_3$  и  $a_4$  могут быть произвольными нечетными числами от 1 до 255. Для каждого сообщения коэффициенты могут быть разными.

Например, все четыре ГПСЧ могли бы быть мультипликативными конгруэнтными генераторами с простым модулем  $2^{31} - 1$  и разными, но фиксированными множителями. Четыре 31-битовых начальных значения плюс четыре 7-битовых коэффициента дают комбинированное начальное значение длиной 152 бита.

Три ГПСЧ можно скомбинировать с помощью операции циклического сдвига  $\ggg$  (раздел 13.7). Если используется 32-битовый беззнаковый генератор, то 32-битовые выходы можно скомбинировать операцией  $x_1 + (x_2 \ggg 11) + (x_3 \ggg 21) \bmod 232$ . Оптимальные величины сдвига – на  $1/3$  и  $2/3$  32-битового регистра. Если требуется более трех генераторов, то делайте величины сдвигов как можно более равномерными. Например, при пяти генераторах величины сдвигов должны быть равны  $1/5, 2/5, 3/5$  и  $4/5$  размера слова с округлением до ближайшего целого.

Еще один фиксированный генератор, *CyGen*, комбинирует два генератора, *C* и *G*, посредством циклического сдвига. *C* может быть любого размера, но размер *G* должен быть 32 или 64 бита. В каждом цикле мы берем соответственно 5 или 6 бит *C*, они определяют величину сдвига. Затем выход *G* циклически сдвигается влево на это число позиций, и результат становится выходом *CyGen*. При этом Эмили не сможет реконструировать *G* по последовательности его выходов.

Необязательно ограничиваться только линейными комбинациями. Например, три генератора можно скомбинировать по формуле  $x_n + y_n z_n$ , или  $x_n + y_n^2 + z_n z_{n-1} z_{n-3}$ , или еще что-то в этом роде. По меньшей мере один член суммы должен быть линейным. Возможности безграничны, и, конечно же, вы можете переключаться с одного метода на другой.

## ПЕРЕМЕННЫЕ КОМБИНАЦИИ

Примером переменной комбинации может служить генератор выбора из раздела 13.11. Но этот раздел я хотел бы начать поучительной историей. Описанный ниже комбинированный генератор CG5 кажется непробиваемо безопасным, однако на поверку таковым не является.

В комбинированном генераторе CG5 используется 5 мультипликативных конгруэнтных генераторов с разными множителями и разными 32-битовыми простыми модулями. Назовем эти генераторы *G0*, *G1*, *G2*, *G3* и *SEL*. (Альтернативно *SEL* мог бы быть линейным конгруэнтным генератором или XOR-генератором со сдвигом с периодом  $2^{31}$ .) Порождающие генераторы *G0*–*G3* используются для порождения псевдослучайных чисел, а селектор *SEL* – для выбора того из них, который будет порождать следующее псевдослучайное число. Точнее, 2 старших бита *SEL* определяют, какой из *G0*–*G3* использовать. Предположим, что *SEL* генерирует биты 10 и, стало быть, выбирает *G2*. Затем выполняется один цикл *G2*, и его выход становится очередным выходом CG5. Комбинированный генератор будет иметь период примерно  $2^{155}$  и обладать хорошими свойствами случайности, но ... не будет криптостойким. И вот почему.

Предположим, что Эмили располагает достаточным объемом открытого текста, и рассмотрим первые 17 выходов CG5. По меньшей мере пять из них должны быть порождены одним и тем же генератором. (Если бы каждый из четырех генераторов породил не более четырех выходов, то всего их было бы максимум 16, а не 17.) Существует всего 6188 способов выбрать 5 элементов из 17. Эмили может перебрать их все. Получается примерно  $1.33 \times 10^{13}$  комбинаций размещения с начальным значением, подлежащих проверке, но это число можно значительно уменьшить. Эмили знает старшие 8 бит каждого из пяти выбранных выходов. Вместо того чтобы начинать с первого из 17 выходов, она должна начать с первого из пяти вы-



бренных выходов. Тогда ей придется проверить только  $2^{23}$  значений вместо  $2^{31}$ . В результате объем работы сокращается до  $5.19 \times 10^{10}$  комбинаций, что уже вполне выполнимо. Комбинированный генератор CG5 небезопасен.

Рассмотрим более безопасный генератор. Я назову его *Gen5*. Как и прежде, в комбинированном генераторе используется пять мультипликативных конгруэнтных генераторов с разными множителями и разными 31-битовыми модулями. Модули и множители фиксированы и обеспечивают хорошие свойства случайности. На этот раз я назову генераторы G1, G2, G4, G8 и SEL. Используются только четыре старших бита селектора SEL. Единица в первом бите означает выбор G1, единица во втором – выбор G2, единица в третьем – выбор G4, единица в четвертом – выбор G8. Если выбрано менее двух генераторов, то выполняется еще один цикл SEL, чтобы произвести новый выбор. Используется только 11 из 16 возможных 4-битовых выходных значений SEL, поэтому дополнительный цикл SEL выполняется в 5/16 случаев, два дополнительных цикла – в 25/256 случаев и т. д.

Если выбрано два или более генераторов, то выполняется один цикл каждого, после чего их выходы складываются по модулю  $2^{31}$  и порождается псевдослучайный выход Gen5. Невыбранные генераторы не выполняются, так что все четыре генератора работают асинхронно. В результате выход немного смещен в сторону меньших чисел, но недостаточно для того, чтобы Эмили смогла этим воспользоваться. Если статистическое смещение вас все-таки тревожит, то либо (1) игнорируйте старший бит и используйте биты 2–9 суммы в качестве выходного байта, либо (2) воспользуйтесь операцией Meld8 (раздел 12.3.7). То есть сформируйте выходной байт генератора, применив ИСКЛЮЧАЮЩЕЕ ИЛИ к битам 1–8 и 9–16 суммы.

\* Эмили больше не может изолировать какой-либо из четырех генераторов. Может показаться, что Эмили теоретически способна выделить одну из шести пар  $G_i + G_j$ , где  $i$  и  $j$  могут принимать значения 1, 2, 4 или 8. Такую пару можно было бы рассматривать как один генератор, и затем, возможно, удастся отделить  $G_i$  от  $G_j$ . Рассмотрим этот подход пристальнее. Чтобы найти начальные значения  $G_i$  и  $G_j$ , необходимо по меньшей мере 9 случайных выходов Gen5. Так как каждая пара встречается всего в 1/11 случаев, Эмили придется рассмотреть 89 или более символов сообщения. Действительно, статистически пять комбинаций трех или четырех генераторов встречаются чаще, чем 6 комбинаций двух генераторов.

Существует приблизительно  $6.356 \times 10^{11}$  размещений 9 элементов из 89, поэтому для Эмили было бы эффективнее просто перебрать  $2^{31} \approx 2.147 \times 10^9$  начальных значений генератора SEL. Это позволит ей найти следующие 10 позиций для всех шести пар  $G_i + G_j$ . Также это позволит подсчитать, сколько раз  $G_i$  и  $G_j$  использовались до каждого

из этих вхождений. Например, предположим, что G2 + G4 встречается в 14-м цикле Gen5. Может оказаться, что из этих 14 циклов G2 использовался в шести, а G4 – в девяти. Теперь Эмили знает сумму шестого выхода G2 и девятого выхода G4.

Если Эмили сможет собрать 10 таких выходных значений, например для G2 + G4, то сможет и определить начальные значения обоих генераторов, выполнив  $2^{31+31-8} = 2^{54} = 1.801 \times 10^{16}$  попыток. Это необходимо проделать для каждого из  $2^{31}$  начальных значений SEL, т. е. всего предстоит выполнить  $2^{85} \approx 3.869 \times 10^{25}$  операций. Это гораздо меньше, чем  $2^{155}$  попыток, необходимых для вскрытия Gen5 полным перебором, но намного превосходит целевой показатель –  $2^{128}$  попыток. Этот генератор получает оценку 9. \*\*

Теперь все готово к решающему удару. Опишу улучшенную версию Gen5, которую буду называть *GenX*. GenX состоит из двух частей: генератора псевдослучайных чисел и шифра. ГПСЧ генерирует последовательность 10-битовых псевдослучайных выходов, а шифр объединяет байты ключа  $k_n$  с байтами сообщения  $x_n$  для порождения шифртекста. Это выводит шифр за пределы шифров со 128-битовым ключом.

Генератор GenX – это просто расширенная версия Gen5. В нем используется четыре порождающих генератора, G1, G2, G4 и G8, и генератор выбора SEL. Старшие 4 бита SEL применяются для выбора комбинации от 2 до 4 порождающих генераторов. Для каждого из выбранных генераторов выполняется один цикл, и выходы складываются по модулю  $2^{31}$ ; обозначим получившуюся сумму G. Старшие 10 бит G объединяются с помощью ИСКЛЮЧАЮЩЕГО ИЛИ со следующими 10 битами G для порождения 10-битового результата, который разбивается на 8-битовый байт ключа  $k_n$  и два управляющих бита  $c_n$ .

Шифр GenX объединяет байт ключа  $k_n$  с байтом сообщения  $x_n$  в соответствии с управляющими битами  $c_n$ , применяя хорошо перемешанную ключом подстановку S. Управляющие биты  $c_n$  определяют, какую комбинирующую функцию использовать для каждого байта открытого текста. Вот, например, как можно интерпретировать управляющие биты:

- 00 заменить  $x_n$  на  $S(k_n) + x_n$ ,
- 01 заменить  $x_n$  на  $k_n + S(x_n)$ ,
- 10 заменить  $x_n$  на  $S(k_n) + S(x_n)$ ,
- 11 заменить  $x_n$  на  $S(k_n + x_n)$ .

Все суммы вычисляются по модулю 256. Шифр GenX получает оценку 10. Ключами этого шифра являются пять 31-битовых начальных значений G1, G2, G4, G8 и SEL плюс ключ перемешивания перестановки S, например методом SkipMix.



## 13.14 *Истинно случайные числа*

Все рассмотренные выше методы генерирования случайных чисел порождают псевдослучайные числа. Во всех известных мне книгах, где обсуждаются случайные числа, авторы не устают твердить, что программным путем невозможно получить истинно случайные числа. А все потому, что они ограничили себя слишком узким кругом допустимых методов. В этом разделе я покажу программно реализованный работоспособный метод массового порождения истинно случайных чисел.

Все описанные в литературе методы порождения истинно случайных чисел опираются на какое-то физическое явление – космические лучи, тепловые шумы, вибрацию, радиоактивный распад и т. д. Эти методы работают слишком медленно для криптографических целей.

Но можно порождать истинно случайные числа с помощью трехшагового процесса. (1) Создать большой массив истинно случайных чисел, взятых из природы. (2) Сделать распределение их вероятностей равномерным. (3) Генерировать случайные числа путем выборки и комбинирования чисел из этого массива. В следующих разделах подробно объясняется каждый шаг.

Природа полна случайностей. Форма, цвет и положение каждого листочка на любом дереве случайны. Это результат ветров, прохождения солнечного света через листву, поступления питательных веществ от корневой системы, упавших на листья капель дождя и градин, активности питающихся листьями насекомых, птиц и белок, землетрясений и многих других факторов. Возьмите любую волну в океане, любое растение или камень в пустыне, любую рябь на реке, любое облако и раковину на пляже – их размер, форма, цвет, местоположение, ориентация, а иногда и скорость будут случайны.

Иногда случайность можно уловить, просто сфотографировав местность. Даже из дома выходить необязательно. Просто рассыпьте пригоршню попкорна на поверхность с каким-то рисунком. Можете также взять для этой цели сделанные самостоятельно фотографии своих знакомых и мест, где вы побывали. Есть фотографии, скачанные вами с сайтов и электронных писем. Еще сотни изображений были загружены на ваш компьютер операционной системой и приложениями. А миллиарды их имеются в сети и доступны с помощью браузера. Эксперимента ради я придумал несуществующее слово, ZRMWKNV, и поискал изображения с таким словом. Нашлось больше тысячи сайтов, и некоторые содержали сотни изображений.

### 13.14.1 *Линейное суммирование с запаздыванием*

В любом файле изображения найдется немало случайности, особенно если разрешение высокое, а распределение значений байтов да-

леко от равномерного и независимого. Распределение можно сгладить с помощью линейного суммирования с запаздыванием, когда все изображение, включая заголовки, рассматривается как одна длинная строка байтов длины  $L$ . Приведу пример:

$$\begin{aligned}x_n &= (7x_n + 31x_{n-40} + 73x_{n-1581}) \bmod 256 \text{ для } n = 1, 2, 3, \dots, L. \\x_n &= (27x_n + 231x_{n-137} + 109x_{n-10051}) \bmod 256 \text{ для } n = 1, 2, 3, \dots, L. \\x_n &= (241x_n + 19x_{n-64} + 165x_{n-2517}) \bmod 256 \text{ для } n = 1, 2, 3, \dots, L.\end{aligned}$$

Индексы, как всегда, заворачивают. Трех проходов достаточно, но, если хотите, можете увеличить их число. Делать распределение слишком равномерным не стоит, потому что тогда оно перестанет быть случайным. Если собираетесь воспользоваться упрощенной версией, например  $x_n = (x_n + x_{n-179}) \bmod 256$ , то понадобится пять проходов. На каждом проходе используйте разные запаздывания.

В коэффициентах 7, 31 и т. д. нет ничего специального. Я выбрал их произвольно. Это могут быть любые нечетные числа от 1 до 255. Запаздывания (40, 1581 и т. д.) на каждом проходе следует выбирать так, чтобы одно запаздывание было намного больше другого. Например, одно запаздывание может быть порядка  $\sqrt[3]{L}$ , а другое порядка  $\sqrt[3]{L^2}$ . Так, если размер файла изображения равен 1 000 000 байт, то запаздывания можно выбрать равными порядка 100 и 10 000, точнее меньшее запаздывание в диапазоне от 50 до 200, а большее в диапазоне от 5000 до 20 000. Если после линейного суммирования с запаздыванием выполняется еще и простая подстановка с ключом, то Эмили будет труднее реконструировать файл изображения.

### 13.14.2 Наложение изображений

Еще один способ построить истинно случайную последовательность – взять два изображения и наложить одно на другое с помощью какой-нибудь комбинирующей функции, например **xor** или **add** (см. раздел 13.1). Хороший метод – выполнить один проход линейного суммирования с запаздыванием для каждого изображения до комбинирования и еще один проход по результирующему изображению после комбинирования. Три изображения можно скомбинировать побитово, применив нелинейную мажоритарную функцию (раздел 13.10), а также по одному проходу линейного суммирования с запаздыванием до и после комбинирования. Этот метод пригоден даже тогда, когда все три изображения разного размера. Совместите одно короткое изображение с левым концом самого длинного, а другое – с правым концом, как показано на рисунке ниже.



Там, где перекрываются только два изображения, сложите их побайтово по модулю 256, а там, где перекрываются все три, используйте мажоритарную функцию или линейную комбинацию по модулю 256, например  $c_n = (113x_n + 57y_n + 225z_n) \bmod 256$ . Коэффициенты могут быть любыми нечетными числами от 1 до 255.

Другой подход к выравниванию изображений – продолжить короткие изображения путем повторения. В примере выше размер изображения  $x$  – 22 байта, а изображения  $y$  – 33 байта. Изображение  $x$  можно продолжить до 33 байт, повторив первые 11 байт. При таком подходе мажоритарную функцию можно использовать во всех 33 позициях. На практике изображения содержат миллионы байт.

### 13.15 Обновление случайных байтов

Ну, хорошо. Теперь у нас есть таблица  $T$ , содержащая несколько миллионов случайных байтов. Они истинно случайные, потому что даже если бы Эмили знала все байты  $T$ , кроме одного, это не помогло бы ей определить отсутствующий байт. У Сандры и у Ривы есть копии. Что дальше? Очевидно, что мы не можем повторять этот процесс всякий раз, как нужно будет отправить сообщение.

С пользой применить  $T$  можно, например разделив ее на ключи для блочного шифра. Из миллиона случайных байтов можно выкроить 62 500 ключей по 128 бит. В конечном итоге весь миллион будет потрачен. Если Сандра работает со стойким блочным шифром, то это несущественно. Она может использовать ключи повторно, при условии что Эмили не в состоянии определить, какие сообщения были зашифрованы одним и тем же ключом. Но, разумеется, для потокового шифра повторно использовать ключи невозможно.

Предположим, что Сандра не хочет рисковать и использовать несколько раз один и тот же ключ. Возможное решение – обновить список случайных чисел. Сандра могла бы наложить другое изображение, но тогда у Ривы должна быть его точная копия. Проблему можно было бы решить, если бы отображения скачивались с сайта, к которому у Ривы и Сандры есть доступ. Это неплохая стратегия в ситуации, когда высок риск перехвата передаваемых ключей.

Другой способ – обновить матрицу  $T$  с помощью линейного суммирования с запаздыванием (раздел 13.14.1). Назовем обновленную таблицу  $T_1$ . Теперь Сандра должна передать только 9 коэффициентов и 6 запаздываний, после чего получит очередные 62 500 ключей. В предположении, что каждый коэффициент занимает 1 байт, а каждое запаздывание 2 байта, Сандре нужно передать всего 21 байт для создания  $T_1$ . А далее для выбора ключа сообщения нужно передать только его смещение от начала  $T_1$ . Для этого достаточно двух байт, потому что все смещения кратны 16. По исчерпанию  $T_1$  мы строим  $T_2$ , выбрав новые коэффициенты и запаздывания, и т. д.

В разделах 13.5 и 13.6 использовались линейные функции, чтобы увеличить период генератора. В данном случае никакого периода нет, а значит, нет и такого ограничения. Вот два примера возможных нелинейных функций:

$$\begin{aligned}x_n &= (ax_n + bx_{n-i} + S(x_{n-j})) \bmod 256 \text{ для } n = 1, 2, 3, \dots, L, \\x_n &= (ax_n + bx_{n-i} + E(x_{n-j}x_{n-k})) \bmod 256 \text{ для } n = 1, 2, 3, \dots, L.\end{aligned}$$

Здесь индексы заворачивают, а и b – нечетные числа от 1 до 255, а i, j, k – целые числа от 1 до L-1. S может быть как фиксированной нелинейной подстановкой, так и переменной подстановкой, перемешанной ключом. Функция E(x) определена следующим образом:

$$E(x) = x + (x \gg 8) + (x \gg 16) + (x \gg 24) + \dots = \lfloor 256x/255 \rfloor.$$

Вычисляя  $E(x_{n-j}x_{n-k}) \bmod 256$ , мы, по сути дела, складываем отдельные байты  $x_{n-j}x_{n-k}$ . Это более стойкая операция, чем просто использование  $x_{n-j}x_{n-k}$ , потому что  $x_{n-j}x_{n-k}$  в 3/4 случаев четное.

Альтернативно Сандра могла бы получать ключи из T следующим образом: взять один байт, пропустить три, взять следующий байт, пропустить два, взять два байта, пропустить четыре и т. д., следуя какой-то периодической схеме. Пропуски могут быть малыми, так чтобы 2 или 3 пропуска можно было бы закодировать одним байтом ключа. Если Эмили каким-то образом получит доступ к источнику случайности T, то, возможно, сумеет определить последовательность малых пропусков. Чтобы предотвратить это, можно сочетать пропуски с прибавлением некоторой последовательности чисел, также периодической, к выбранным байтам по модулю 256. Безопаснее, когда количество пропусков и количество слагаемых – взаимно простые числа, например 12 пропусков и 11 слагаемых. При таком подходе каждый ключ сообщения будет содержать два байта для кодирования начальной точки, 6 байт – для 12 пропусков и 11 слагаемых – всего 20 байт, или 160 бит. Этот метод можно было бы назвать *Skip & Add* (пропустить и сложить).

В такого типа системах важно, чтобы Эмили не могла практически реконструировать T. Например, со временем Эмили могла бы набрать открытые тексты многочисленных сообщений и восстановить их ключи. Если она также знает, где в T эти ключи находятся, быть может, потому что Сандра передает эту информацию Риве в каждом сообщении, то сможет реконструировать части T. По этой причине саму T никогда не следует использовать в качестве хранилища ключей. T нужно сохранять для конструирования  $T_1, T_2, \dots$ , а уже из них нарезать ключи сообщений. Хранение T защитит Сандру и Риву от потери или повреждения  $T_1$ . T можно было бы назвать *базовым* ключом, а  $T_1, T_2, \dots$  – *производными* ключами.

Даже если бы Эмили могла каким-то образом реконструировать  $T_1$  или  $T_2$ , она не сможет по ним восстановить исходную таблицу T,

потому что та истинно случайна. Если бы Эмили перебирала все возможные комбинации коэффициентов и запаздываний, ничто не подсказало бы ей, какая из этих квинтильонов строк является правильной случайной строкой Т.

## 13.16 Синхронизированные гаммы

В случае криптографии с секретным ключом Сандра и Рива должны использовать один и тот же ключ. Обычно это означает, что либо (1) зашифрованный ключ передается вместе с сообщением, либо (2) обе стороны хранят список ключей и выбирают из него конкретный ключ в зависимости от даты, времени суток или еще какого-то внешнего фактора. Есть и третий метод, относящийся только к потокковым шифрам.

Сандра и Рива могли бы использовать синхронизированную гамму. Это значит, что обе непрерывно генерируют одни и те же гаммы. Шифруя сообщение, Сандра начинает со следующего байта ключа в своей гамме, который должен в точности совпадать со следующим байтом в гамме Ривы. Получив сообщение, Рива должна начать с того же места в своей гамме. Сандра и Рива должны начинать генерирование с одинаковым начальным значением и в одно и то же время. Синхронизированный метод наиболее полезен, когда между сторонами имеется прямой кабель или соединение в пределах прямой видимости или когда обе стороны получают сообщения беспроводным способом от одного и того же передатчика. Он хорошо подходит для передачи оцифрованной речи на небольшое расстояние.

Если сообщения передаются по сети со значительными задержками в узлах или ретрансляционных пунктах, особенно по сети с коммутацией пакетов, где части сообщения могут приходить по разным путям и должны собираться в точке приема, то отправитель должен указывать временную метку начала передачи, например в заголовке сообщения.

Поскольку требуется время для зашифрования сообщения Сандрой и для его передачи Риве, может показаться, что Рива вынуждена генерировать случайные ключи на несколько микросекунд позже, чем Сандра. И то же самое верно для передачи сообщений в обратном направлении.

Выйти из этого тупика можно несколькими способами. Например, Сандра могла бы начинать сообщения только в определенных циклах генерирования потока псевдослучайных чисел, например на каждом 100 000-м цикле. Тогда Рива, получив сообщение в цикле 123 456 789 123, будет знать, что генерирование ключа было начато в цикле 123 456 789 000. Если сообщение было получено близко к циклу, кратному 100 000, например 123 456 701 234, то Рива могла бы попробовать 123 456 700 000 и 123 456 600 000. Риве придется

хранить последние два набора по 100 000 псевдослучайных чисел. Величину 100 000 можно скорректировать в меньшую или большую сторону в зависимости от быстродействия ГПСЧ и времени передачи между сторонами.

Остался один вопрос – как Риве определить начало и конец зашифрованного сообщения. Если канал связи находится в состоянии простоя, когда не передаются ни нули, ни единицы, то проблемы вообще не возникает. В противном случае будем предполагать, что во время простоя передается постоянный поток нулей. В таком случае мы будем добавлять лишний единичный бит до и после сообщения, как если бы оно было заключено в кавычки, и потребуем, чтобы перед началом сообщения было передано как минимум 64 нуля. Шансы, что в обычном сообщении может встретиться подряд 64 нуля, пренебрежимо малы. (Отметим также, что среднее время между сообщениями в реальности будет больше 50 000 циклов; 64 цикла – это просто худший случай.) Таким образом, обнаружив единицу после 64 нулей, Рива может быть уверена, что это начало следующего сообщения, а обнаружив единицу, за которой следует 64 или более нулей, она знает, что это конец сообщения.

## 13.17 Функции хеширования

Функции хеширования, или хеш-функции, не являются шифрами, но тесно связаны с шифрами и часто применяются в криптографии. В этом разделе я обсужу два применения хеш-функций и для каждого случая представлю один пример.

Хеш-функции часто применяются при поиске. Предположим, что имеется список людей – клиентов, пациентов или студентов – и в нем требуется часто производить поиск сведений о человеке. Хеширование позволяет осуществить быстрый поиск – имя человека преобразуется в число, которое ищется напрямую в таблице. Например, имя «John Smith» могло бы быть преобразовано в число 2307, а в записи таблицы с индексом 2307 хранились бы сведения о Джоне Смите.

Опишем хеш-функцию, предназначенную для этой цели. Для каждой буквы  $L$  алфавита случайным образом выберем двоичное значение  $R(L)$  некоторого фиксированного размера, скажем 32 бита. Для хеширования имени просто применим ИСКЛЮЧАЮЩЕЕ ИЛИ к буквам имени и соответствующим им 32-битовым числам. Слабость этой схемы в том, что анаграммам сопоставляются одинаковые хеш-значения. Например, ARNOLD, ROLAND и RONALD хешируются в одно и то же число. Чтобы решить эту проблему, будем циклически сдвигать хеш-значения влево на одну позицию после добавления каждой буквы. Операция описывается такой формулой:

$$H_n = (H_{n-1} \oplus R(M_n)) \lll 1.$$



Обозначим финальное хеш-значение  $H$ . Его можно преобразовать в индекс  $I$  для таблицы имен размера  $T$ :  $I = \lfloor HT/2^{32} \rfloor$ . Например, если имя хешируется в значение 917354668, а в таблице 5000 записей, то индекс будет равен  $\lfloor 917354668 \times 5000 / 4294967296 \rfloor = \lfloor 1067.94 \rfloor = 1067$ . Назовем этот метод хеширования *Hash32*.

Один и тот же индекс может соответствовать нескольким именам. Существует несколько способов разрешения таких конфликтов, например завести отдельную таблицу для хранения дубликатов, или хешировать имя второй раз, чтобы выбрать другую запись, или связывать дубликаты в список.

Хеш-функции применяются также для аутентификации сообщений. В этом случае хешируется все сообщение и порождается длинное хеш-значение, допустим 16-байтовое. Это хеш-значение следует отправить Риве способом, исключающим манипуляции, например прибегнув к услугам третьей стороны, которая хранит хеш-значения вместе с временными метками. Затем Рива хеширует сообщение и сравнивает хеш-значения. Если они различаются, то сообщение, вероятно, было изменено. Используемая для этой цели хеш-функция должна быть устроена так, чтобы Эмили не могла модифицировать сообщение, не изменив хеш-значения. То есть Эмили не сможет найти другое сообщение с точно таким же хеш-значением. Аналогично Сандра не может изменить сообщение и заявить, что именно его она и отправляла, потому что хеш-значения не совпадут.

Для построения такой хеш-функции мы будем использовать 4 сильно нелинейные подстановки,  $A$ ,  $B$ ,  $C$  и  $D$ . Они даже могут быть общеизвестны. Следует приложить усилия к тому, чтобы все четыре подстановки были сильно нелинейными и минимально коррелированными. Основная операция – объединение каждого байта сообщения с четырьмя предыдущими байтами с применением комбинирующей функции **xors**, т. е. мы выполняем ИСКЛЮЧАЮЩЕЕ ИЛИ, а затем к результату применяем простую подстановку. Будем считать, что  $H$  – копия сообщения  $M$ , т. е. сообщение не уничтожается в процессе хеширования. Каждый символ копии  $H_n$  хешируется по формуле

$$H_n = A(B(C(D(H_n) \oplus H_{n-1}) \oplus H_{n-4}) \oplus H_{n-16}).$$

При этом каждый байт хеша зависит от каждого предшествующего байта, и от него, в свою очередь, зависит каждый следующий за ним байт.

Для хеширования необходим вектор инициализации (раздел 11.10), чтобы вычислить хеш-значение первых 16 байт сообщения. Для этой цели можно использовать копию первых 16 байт. То есть в начальный момент байты от  $H_{-15}$  до  $H_0$  совпадают с байтами от  $H_1$  до  $H_{16}$ , а те, в свою очередь, с байтами  $M_1$ – $M_{16}$  сообщения. Имея вектор инициализации, мы можем распространить хеш от  $H_1$  до  $H_L$ , где  $L$  – длина сообщения.

При этом последние несколько байтов хешируются относительно слабо. Быть может, Эмили удастся изменить их без особых усилий. Решить проблему можно, продолжив хеширование и после того, как уже достигнут конец сообщения. Для этого после хеширования первых 16 байт мы сохраняем их для последующего использования. Дойдя до конца сообщения, мы дописываем эти 16 байт и продолжаем хеширование до конца расширенного сообщения. Последние 16 байт и становятся хеш-значением всего сообщения. Назовем этот метод *Hash128*.

На некоторых компьютерах быстрее хешировать сообщение по четыре байта за раз, пользуясь машинными командами для работы с 32-битовыми числами. Сообщение и хеш-значения рассматриваются как списки  $L$  32-битовых слов, а не как последовательности 4L байтов. Массив хешей  $H$  первоначально является копией сообщения. Если длина сообщения не делится нацело на 4, то в конец дописывается от одного до трех байт, чтобы получилось полное слово. Копии первых двух слов  $H$  добавляются в начало сообщения, т. е.  $H_{-1} = H_1$  и  $H_0 = H_2$ . После хеширования первых четырех слов они дописываются в конец сообщения.

В следующей хеш-функции, называемой *HashPQ*, используются два простых числа,  $P = 2^{32} - 5 = 4294967291$  и  $Q = 2^{32} - 17 = 4294967279$ , а также магический множитель  $R = 77788888$ , являющийся первообразным корнем одновременно  $P$  и  $Q$ . Операция хеширования имеет вид:

$$H_n = H_n + (RH_{n-1} \bmod P) + (RH_{n-2} \bmod Q) \text{ для } n = 1, 2, 3, \dots, L+4.$$

Если сумма больше  $2^{32} - 1$ , то значение усекается до 32 бит – старшие биты просто игнорируются. Таким образом, мы задаром получаем операцию по модулю  $2^{32}$ . Последние четыре слова массива  $H$  и являются 16-байтовым хеш-значением. Функция *HashPQ* потребляет меньше памяти, чем *Hash128*, потому что не нуждается в четырех простых подстановках.

*Hash32*, *Hash128* и *HashPQ* обладают свойством, необходимым любой хорошей функции хеширования, – изменение любого входного бита или комбинации входных битов приводит к изменению примерно половины выходных битов. Все три функции работают быстро и могут быть вычислены за один проход слева направо.



# 14

## Одноразовый блокнот

---

### **Краткое содержание главы:**

- шифры типа одноразового блокнота;
- шифр Вернама, аппроксимирующий одноразовый блокнот;
- алгоритм распределения ключей Диффи–Хеллмана;
- построение больших простых чисел, необходимых для алгоритма Диффи–Хеллмана и криптографии с открытым ключом.

Самым известным потоковым шифром является *одноразовый блокнот*. Многие авторы употребляют этот термин в ограниченном смысле, имея в виду только побайтовое применение ИСКЛЮЧАЮЩЕГО ИЛИ к открытому тексту и гамме. Исторически это неправильно. Первый шифр типа одноразового блокнота был опубликован в 1882 году Фрэнком Миллером, банкиром из Сакраменто, штат Калифорния. Он хотел сэкономить деньги, уменьшив длину телеграфных сообщений. В телеграфном коде Миллера использовались пятизначные цифровые группы для обозначения слов и фраз, часто встречающихся в деловой корреспонденции. Для обеспечения секретности Миллер предложил шифр, сводившийся к прибавлению трехзначного числа к каждой пятизначной группе. Кодовые значения были достаточно малы, так что сумма никогда не превышала 99999, т. е. все коды были меньше 99000. Таким образом, первый одноразовый блокнот был десятичным, а не двоичным.

Систему, давшую одноразовому блокноту это название, изобрел криптограф Вернер Кунце из немецкого Управления радиотехнической разведки Pers Z S в 1922 году. Система Кунце была основана на стандартном дипломатическом коде, состоящем из групп по 5 цифр. Как и в шифре Миллера, в шифре Кунце кодовые группы складывались с ключевыми. Сложение производилось поразрядно без переноса, т. е.  $33333 + 56789$  равнялось 89012, а не 90122. Для распределения ключей использовались блокноты по 50 листов, каждый из которых содержал 8 строк по 6 групп в каждой. Страница блокнота применялась для шифрования одного сообщения, а затем выбрасывалась. Отсюда и название. В ходе дальнейших усовершенствований появились водорастворимые чернила и бумага, обеспечивающие быстрое уничтожение.

Другой вариант одноразового блокнота изобрел Лео (Леопольд Самуэль) Маркс, британский писатель и сценарист (фильм «Подглядывающий»), в 1940 году. Он широко использовался британскими шпионами. В шифре Маркса вместо цифр применялись буквы. Для получения буквы шифртекста отправитель складывал букву ключа с буквой открытого текста по модулю 26. Другими словами, одноразовый блокнот Маркса был не чем иным, как шифром Беласо со случайным ключом. Профессор MIT Клод Шеннон изобрел тот же шифр между 1940 и 1945 годом, а советский ученый, специалист по теории информации, Владимир Котельников изобрел его вариант в 1941 году или раньше, но детали до сих пор засекречены. И Шеннон, и Котельников математически доказали, что одноразовый блокнот вскрыть невозможно. И по сей день это единственный метод шифрования, невскрываемость которого строго доказана.

Поскольку в одноразовом блокноте Миллера 1882 года и в блокноте Кунце 1922 года в качестве комбинирующей функции использовалось сложение десятичных цифр, а в блокноте Маркса 1940 года – сложение по модулю 26, вряд ли кто-то рискнет утверждать, что одноразовые блокноты ограничены только применением операции ИСКЛЮЧАЮЩЕЕ ИЛИ к открытому тексту и ключу. Перечислим определяющие свойства одноразового блокнота.

- 1 Ключ должен быть не короче сообщения.
  - 2 Ключ неотличим от истинно случайной последовательности.
  - 3 Каждый символ или блок ключа объединяется с одним символом или блоком того же размера открытого текста.
  - 4 Ключ используется только один раз.
- Любой шифр, удовлетворяющий этим четырем условиям, является одноразовым блокнотом. Но чтобы доказать невскрываемость одноразового блокнота, необходимо еще одно, более сильное условие:
- 5 Любой символ открытого текста с равной вероятностью преобразуется в любой символ шифртекста.

А теперь взглянем на ставший уже достоянием истории шифр, основанный на ИСКЛЮЧАЮЩЕМ ИЛИ, который тесно связан с системами типа одноразового блокнота.

## 14.1 Шифр Вернама

К 1918 году многие дипломатические миссии отказались от отправки и приема сообщений телеграфистами, поскольку затем текст нужно было печатать вручную. Вместо этого сообщения записывались на катушки перфоленты в 5-битовом коде Бодо, изобретенном в 1870 году французским инженером Эмилем Бодо, или в коде Мюррея–Бодо, изобретенном новозеландским журналистом Дональдом Мюрреем в 1901 году. (Не буду вдаваться в детали этих кодов, поскольку они несколько раз менялись между 1870 и 1950-ми годами, когда компания Вестерн Юнион прекратила их использовать. Коды в стиле Бодо полностью вышли из употребления после 1963 года, когда им на смену пришел код ASCII.) Важная особенность состояла в том, что оператор набивал сообщение на бумажную перфорированную ленту с пятью рядами отверстий, с которой их можно было передать и распечатать на принимающей стороне без дальнейшего участия человека.

Как и код Морзе, коды Бодо и Мюррея–Бодо не предполагали никакой секретности. Любой желающий мог прочесть сообщение прямо с ленты. Вплоть до 1918 года для обеспечения секретности шифровальщик должен был сначала зашифровать сообщение вручную, затем оно распечатывалось и дешифровалось другим шифровальщиком на принимающей стороне. Нужен был какой-то метод для ускорения этого процесса. И тут на сцене появился Вернам.

*Шифр Вернама* разработал Гилберт Сэндфорд Вернам из компании AT&T Bell Labs в 1918 году по просьбе Джозефа О. Моборна из службы связи сухопутных войск США. Идея была простой и остроумной. Человек, как и раньше, набивал сообщение на ленту, но передавался результат объединения кода символа с кодом ключа с помощью ИСКЛЮЧАЮЩЕГО ИЛИ. Коды ключа считывались с отдельной перфоленты, на которую была набита случайная на первый взгляд последовательность символов. На принимающей стороне переданные символы снова объединялись ИСКЛЮЧАЮЩИМ ИЛИ с точной копией этой ленты, в результате чего производилось дешифрирование. На каждой ленте было набито 1000 квазислучайных символов, так что для длинных сообщений ключ повторялся через каждые 1000 символов.

На рисунке ниже показаны две ленты – с сообщением и с ключом, устройство чтения ленты, электрические схемы для применения ИСКЛЮЧАЮЩЕГО ИЛИ к ключу и открытому тексту и перфоратор на принимающей стороне, которая может находиться далеко от передающей. В другой конфигурации перфоратор можно заменить принтером или передатчиком.

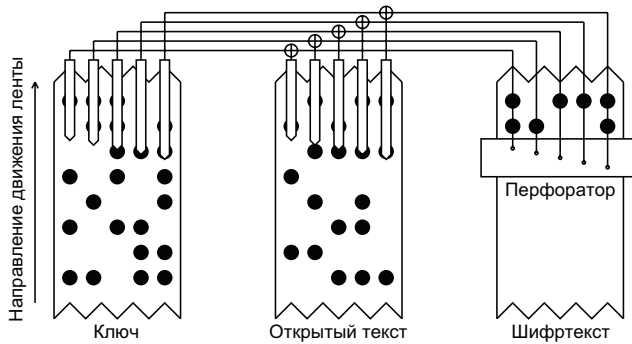


Рисунок мой собственный, т. к. я не сумел найти изображения машины Вернама, быть может, потому что она была засекречена.

Я назвал ленты с ключами «квазислучайными», потому что они набивались человеком на клавиатуре устройства, напоминающего пишущую машинку, предшественнике Friden Flexowriter. В результате символы, расположенные в центре клавиатуры, использовались чаще, чем расположенные по краям. У человека плохо получается порождать случайные числа или символы. Но для 1918 года это был очень стойкий шифр.

Во многих источниках шифр Вернама ошибочно называют одноразовым блокнотом, возможно, потому что это был первый шифр, в котором ИСКЛЮЧАЮЩЕЕ ИЛИ применялось к сообщению и ключу, записанным в двоичном виде. Однако шифр Вернама не был одноразовым блокнотом, потому что ключ повторялся с фиксированным периодом 1000 символов. А настоящий одноразовый блокнот изобрел Миллер 36 годами раньше, и в оригинальной версии использовалась десятичная система.

Активно работающее посольство могло отправлять 100 и более шифрограмм в день. Для корреспонденции с несколькими другими посольствами нужно было несколько наборов лент. Ленты для сообщений из Вашингтона в Берлин не следовало применять для шифрования трафика из Берлина в Вашингтон. Все ленты маркировались 6-значными порядковыми номерами. Перед отправкой каждого сообщения номер ленты передавался в *открытом* виде, т. е. незашифрованным. Операторы должны были помнить, какая лента предназначена для какого посольства, и вести учет использованным и подлежащим уничтожению лентам. В каждом посольстве нужно было поддерживать достаточный запас лент.

Вскоре Вернам предложил вторую версию, в которой использовались две ленты, объединявшиеся ИСКЛЮЧИТЕЛЬНЫМ ИЛИ с открытым текстом. На одной ленте было 1000 символов, на другой 999, т. е. период составлял 999 000 символов. Одну и ту же пару лент можно

было использовать целый день, достаточно было просто начинать шифрование каждого сообщения с новой позиции на ленте. Если у посольства было, к примеру, 100 лент, то в разные дни можно было использовать различные их комбинации, пока бумага не приходила в негодность.

Легко видеть, что двухленточные машины Вернама можно обобщить на 3 или 4 ленты. Но, насколько мне известно, до этого дело не дошло, потому что ленточные машины вскоре были заменены роторными (см. раздел 5.10).

## 14.2 *Запас ключей*

У одноразового блокнота есть серьезная проблема – поддержание достаточного запаса ключей. Бумажные ленты еще годятся для 10 станций, каждая из которых отправляет до 100 сообщений в день, но для 100 станций, отправляющих по 1000 сообщений в день, этот метод непригоден.

Во многих книгах и статьях по криптографии описывается следующий порочный круг. Сандра и Рива решили обмениваться сообщениями с помощью одноразового блокнота. У обеих имеются копии длинного случайного ключа. Они используют его участок за участком, пока ключ не закончится. Теперь им нужен другой случайный ключ. Сандра может выбрать его и отправить Риве, однако его необходимо зашифровать, чтобы скрыть от Эмили. Простейший способ зашифровать ключ – воспользоваться одноразовым блокнотом, поэтому, чтобы зашифровать новый ключ, нужен еще один ключ той же длины. Сандра, конечно, может выбрать его и отправить Риве, но ведь и этот ключ нужно зашифровать. Значит, понадобится еще один ключ. И так далее до бесконечности.

К решению проблемы можно подойти с двух сторон. Во-первых, случайную гамму можно обновлять, как описано в разделе 13.15, например методом суммирования с запаздыванием. Скажем, один раз в сутки или с другой частотой, по согласованию сторон, можно порождать новый ключ из базового. Во-вторых, сами суточные ключи необязательно использовать напрямую для шифрования сообщений. Вместо этого по суточным ключам строятся ключи сообщений. В таком случае, даже восстановив какой-нибудь ключ сообщения, Эмили все равно окажется в двух шагах от восстановления базового ключа. В следующих разделах мы опишем некоторые методы порождения ключей сообщений.

Все методы преследуют две цели: либо (1a) метод должен каждый день генерировать достаточно ключевого материала, чтобы никакие два ключа сообщений не перекрывались, либо (1b) у Эмили не должно быть возможности определить, какие участки ключей перекрываются, и (2) Эмили не должна иметь возможности реконструировать участки производных ключей или базового ключа.

### 14.2.1 Возвращение ключей в оборот

Суточный ключ выводится из базового, как описано в разделе 13.14. Последовательные участки суточного ключа используются для генерирования ключей сообщений, например путем несложного шифрования. Достаточно простой подстановки с ключом. Я рекомендую оставлять между соседними ключами промежутки случайной ширины, скажем от 1 до 32 байт. По достижении конца суточного ключа производится возврат в начало с использованием одного прохода линейного суммирования с запаздыванием (раздел 13.14.1), чтобы продлить срок его службы в те дни, когда сообщений особенно много. Наглядно это можно представить так: всякий раз после отправки сообщения его ключ и следующий за ним промежуток перемещаются из начала суточного ключа в его конец, после чего обновляются с помощью линейного суммирования с запаздыванием. Сандра и Рива должны делать это синхронно.

Это хорошо работает, когда количество сообщений мало и вероятность, что Сандра и Рива отправят сообщения друг другу одновременно, невелика. Если трафик интенсивный, то лучше использовать два базовых и два суточных ключа, один для сообщений от Сандры к Риве, а другой – от Ривы к Сандре.

### 14.2.2 Комбинированный ключ

Для каждого сообщения длины  $L$  из суточного ключа берутся три сегмента длины  $L$ . Назовем их  $x$ ,  $y$ ,  $z$ , а их начальные позиции в суточном ключе обозначим  $p_x$ ,  $p_y$  и  $p_z$ . Если какая-то позиция оказывается близко к концу суточного ключа, этот сегмент может заворачивать в начало. Каждый байт ключа сообщения формируется в виде линейной комбинации соответствующих байтов  $x$ ,  $y$  и  $z$ . То есть

$$k_n = (ax_n + by_n + cz_n) \bmod 256,$$

где коэффициенты  $a$ ,  $b$  и  $c$  могут быть произвольными нечетными числами от 1 до 255. Значения  $a$ ,  $b$ ,  $c$  и позиций  $p_x$ ,  $p_y$ ,  $p_z$  для всех сообщений должны быть различны. Их можно согласовать заранее или зашифровывать и отправлять вместе с сообщением.

### 14.2.3 Ключ выбора

Для каждого сообщения длины  $L$  берутся два непересекающихся сегмента, начинающихся в случайно выбранных позициях суточного ключа. Первый сегмент – *селектор*,  $s$ , длины  $L$ . Второй сегмент – *склад*,  $x$ , длины 256. Чтобы зашифровать  $n$ -й символ сообщения  $m_n$ , мы сначала берем соответствующий байт селектора  $p = s_n$ . Он выбирает позицию на складе, из которой берется байт ключа:  $k_n = x_p$ . Байт ключа  $k_n$  объединяется с байтом сообщения  $m_n$  одной из комбинирующих функций, например **xors** или **adds**.

После того как байт ключа  $k_n$  был использован,  $x_p$  заменяется на складе числом  $(ax_p + b) \bmod 256$ . Коэффициенты  $a$  и  $b$  должны удовлетворять условиям Халла–Добелла (раздел 13.4):  $a \equiv 1 \pmod{4}$  и  $b \equiv 1 \pmod{2}$ . Таким образом, каждая из 256 позиций склада,  $x$ , становится отдельным линейным конгруэнтным генератором псевдослучайных чисел. Коэффициенты  $a$  и  $b$  могут быть одинаковыми для всех 256 позиций, но могут и различаться. Например, можно использовать две разные пары значений  $a$  и  $b$  и выбирать одну из них по какому-то фиксированному принципу. Но сколько бы ни было пар значений, они должны быть различны для разных сообщений.

Обновлять склад можно и по-другому: заменять  $x_p$  на  $(ax_p + bx_p - 1) \bmod 256$ , где  $a$  и  $b$  – произвольные нечетные числа от 1 до 255. Можно также в качестве замены брать  $(ax_p + bx_p - i) \bmod 256$ , где  $i$  – произвольное целое от 2 до 255.

Поскольку существует всего 8192 возможных значения пары  $a$  и  $b$ , причем значения  $a = 1$  следует избегать, от дублирования никуда не деться. Но это не проблема, коль скоро Эмили не может сказать, какая пара использовалась при передаче каждого сообщения. Важно, чтобы Эмили не могла набрать несколько сообщений с заведомо одинаковыми значениями  $a$  и  $b$ . Недостаток использования индикаторов (см. ниже) в том, что противник может собрать несколько сообщений с одним индикатором и будет знать, что эти сообщения зашифрованы одним и тем же ключом.

## 14.3 Индикаторы

В классической криптографии один ключ часто использовался на протяжении длительного времени, иногда месяцами или годами. В наши дни ключи обычно меняются после шифрования каждого сообщения. В случае одноразового блокнота ключ сообщения должен использоваться ровно один раз. В противном случае Эмили могла бы сдвинуть одно сообщение относительно другого и, воспользовавшись индексом совпадения (раздел 5.7), обнаружить перекрытие.

При наличии двустороннего трафика умеренной интенсивности Сандра и Рива могли бы воспользоваться небольшой книгой, где перечислены ключи, которые следует использовать, скажем в определенное время суток и в определенные дни недели. До появления компьютеров было принято нумеровать сообщения. Номер можно было зашифровать и передать вместе с сообщением. Сандра и Рива могли бы по номеру найти ключ в книге.

Шифровальная книга становится непригодной, когда трафик интенсивный или в обмене сообщениями принимает участие несколько сторон. Это верно, даже если заменить книгу компьютерным файлом. Решением проблемы может служить использование индикаторов. *Индикатором* называется информация, отправляемая вместе с сообщением и позволяющая получателю определить ключ.



Когда-то давно индикатором был сам ключ, скрытый внутри сообщения. Например, ключом могла быть третья группа символов сообщения или первые символы первых восьми групп. Чуть более хитрая уловка – средняя цифра второй группы говорит, какая группа является ключом. Очевидная проблема такого рода индикаторов заключается в том, что, изучив систему, Эмили сможет читать все сообщения. Но, даже не зная всех деталей, Эмили может перепробовать все группы символов и понять, есть ли среди них ключ. Найдя таким порядком несколько ключей, она сможет вывести закономерность.

Более безопасный способ – зашифровать ключ и использовать его в качестве индикатора. Так поступали немцы во Второй мировой войне в машинах Энигма. Для шифрования ключа сообщения была предусмотрена специальная настройка, которая менялась каждый день. Сначала на Энигме выставлялась суточная настройка, определявшая главный ключ. Затем оператор случайным образом выбирал и вводил ключ сообщения. Во избежание ошибок при передаче ключ вводился еще раз. В обоих случаях ключ сообщения шифровался главным суточным ключом. В польской *Бомбе* это двойное шифрование ключа сообщения было использовано для определения ключей. (*Криптологическая бомба* – электромеханическое устройство, изобретенное польским главным криптографом Марианом Режевски в 1938 году для вскрытия немецких сообщений, зашифрованных Энигмой.) Когда немцы поняли, в чем дело, эта практика была прекращена, и поляки ослепли – они больше не могли читать зашифрованные сообщения. Алан Тьюринг предвидел эту проблему и спроектировал свою *Бомбу* для работы с *зацепками* (crib); так он называл вероятные фрагменты открытого текста. Французская машина для вскрытия Энигмы также называлась бомбой, возможно, по названию замороженного десерта *bombe glacée* (ледяная бомба), по форме напоминавшего купол.

В разделе 14.2 описано несколько методов генерирования ключей сообщения по суточному ключу. В каждом из них используется небольшой набор параметров, например коэффициенты в формуле линейного суммирования с запаздыванием или позиция внутри суточного ключа. Эти наборы параметров идеально подходят на роль индикаторов.

## 14.4 Алгоритм распределения ключей Диффи–Хеллмана

Ну и хватит о классических методах. Поговорим о чем-то более современном. Алгоритм распределения ключей Диффи–Хеллмана был изобретен в 1976 году Мартином Хеллманом, профессором Стэнфордского университета, и Бэйли Уитфилдом Диффи, его ассистентом, позже работавшим в компании Sun Microsystems. Концепцию,



лежащую в основе криптографии с открытым ключом, изобрел в 1974 году Ральф Меркл, тогда студент старшего курса Калифорнийского университета в Беркли.

Важная особенность алгоритма Диффи–Хеллмана заключается в том, что Сандра и Рива могут выработать безопасный ключ шифрования, даже если Эмили перехватывает все сообщения, которыми они обмениваются. Чтобы настроить обмен, Сандра и Рива должна согласовать большое простое число  $P$  и первообразный корень из этого числа  $w$ . Или же Сандра может просто выбрать  $P$  и  $w$  и передать их Риве.  $P$  и  $w$  можно передавать в открытом виде. Напомним, что найти первообразные корни нетрудно (см. раздел 13.3). Для большинства простых чисел по крайней мере одно из чисел 2, 3, 5 и 7 является первообразным корнем.

Сандра выбирает секретный показатель степени  $s$  и вычисляет  $x = w^s \bmod P$ . Это значение она отправляет Риве, при этом сохраняет  $s$  у себя. Рива выбирает секретный показатель степени  $r$  и вычисляет  $y = w^r \bmod P$ . Это значение она отправляет Сандре, при этом сохраняет  $r$  у себя. Теперь Сандра может вычислить значение  $y^s \bmod P$ , равное  $w^{rs} \bmod P$ , а Рива – значение  $x^r \bmod P$ , равное  $w^{sr} \bmod P$ . Поскольку  $w^{rs} = w^{sr}$ , Сандра и Рива вычислили одно и то же число, которое могут использовать в качестве ключа шифрования или разделить на несколько ключей шифрования. Эффективный способ возведения в степень описан в разделе 13.3.

Некоторые авторы (и Википедия) описывают алгоритм распределения ключей Диффи–Хеллмана как метод криптографии с открытым ключом. Они толкуют о комбинировании открытых и закрытых ключей Сандры и Ривы. Это не так. В алгоритме Диффи–Хеллмана нет никаких открытых ключей. Даже если считать показатели степени  $r$  и  $s$  ключами, оба они секретны.

Предположим, что Эмили перехватила все сообщения, которыми обмениваются Сандра и Рива. Тогда Эмили знает  $P$ ,  $w$ ,  $x$  и  $y$ , т. е.  $w^s \bmod P$  и  $w^r \bmod P$ , но не знает ни  $s$ , ни  $r$ , ни  $w^{rs} \bmod P$ . Определение  $w^{rs} \bmod P$  называется *задачей Диффи–Хеллмана*. Неизвестно, эквивалентна ли она определению  $r$  и  $s$ , но считается, что эти задачи одинаково трудные. Определение  $s$  и  $r$  по известным  $P$ ,  $w$  и либо  $x$ , либо  $y$  называется *задачей дискретного логарифмирования*. Известно, что она очень трудна. Считается, что если  $P$ ,  $r$  и  $s$  достаточно велики, то она вычислительно неразрешима. Специалисты спорят по поводу того, насколько большим должно быть  $P$ , но все сходятся на том, что от 300 до 600 десятичных цифр достаточно. В некоторых реализациях  $P$  может включать до 1234 десятичных цифр, т. е. 4096 бит. Показатели степени  $r$  и  $s$  могут быть гораздо меньше. Эксперты рекомендуют от 40 до 150 десятичных цифр.

Алгоритм *Сильвера–Полига–Хеллмана*, названный по именам авторов Роланда Сильвера, Стивена Полига и Мартина Хеллмана, позволяет легко решить задачу дискретного логарифмирования, если все множители  $P - 1$  малы. Он решает задачу отдельно для каждого

множителя. Поэтому Сандра должна следить за тем, чтобы простое число  $P$  было *безопасным*, т. е. содержало хотя бы один большой множитель, скажем  $q > 10^{35}$ . В идеале Сандра должна выбирать  $P$  вида  $2Q + 1$ , где  $Q$  – тоже простое число. Соответствующее  $Q$  называется простым числом Софи Жермен в честь французской специалистки по теории чисел Мари-Софи Жермен, занимавшейся также акустикой и теорией упругости. Еще лучше, если  $Q - 1$  и  $Q + 1$  имеют большие простые множители. В следующем разделе мы явно построим такое  $Q$ , что  $Q - 1$  имеет большой простой множитель. С большой вероятностью и  $Q + 1$  имеет большой простой множитель, просто потому что  $Q$  очень велико. Числа, все простые множители которых малы, называются *гладкими*. Среди больших чисел они встречаются редко.

#### \*14.4.1 Построение больших простых чисел, старый подход

Традиционный метод построения больших простых чисел, описанный на многих сайтах, начинается со случайного выбора нечетного числа  $N$  нужного размера и проверки его на простоту. Сначала опробуется несколько сотен малых простых чисел. Если  $N$  делится на одно из них, то оно не простое. Пробуем следующее. Этот предварительный тест полезен, потому что выполняется быстро. Далее применяется вероятностный тест на простоту. Самый распространенный из них – тест *Миллера–Рабина*, был предложен Гэри Л. Миллером и Майклом О. Рабином. Пусть  $N - 1 = 2^h d$ , где  $d$  нечетно. То есть  $2^h$  – наибольшая степень 2, на которую делится  $N - 1$ . Первый шаг – выбрать основание  $b$  в диапазоне от 2 до  $N - 2$  и проверить, что  $b^d \equiv 1 \pmod{N}$ . Если это так, то  $N$  проходит тест. Если нет, проверяем, верно ли, что  $b^{2^i d} \equiv -1 \pmod{N}$ ,  $b^{4^i d} \equiv -1 \pmod{N}$  и т. д. Продолжаем до тех пор, пока  $2^g d$  меньше  $2^h d$ . Если такое  $g$  найдено, то  $N$  проходит тест, а  $b$  называется *свидетелем* простоты  $N$ . Если нет, то  $N$  заведомо не является простым и надо пробовать другое число.

Если  $N$  проходит тест, все равно с вероятностью  $1/4$   $N$  может быть составным. Если нужно снизить эту вероятность до  $1/2^{128}$ , то нужно выполнить 64 теста Миллера–Рабина с разными основаниями  $b$ . К сожалению, гарантии мы все равно не получим. Тест Миллера–Рабина неправильно определяет числа Кармайкла как простые. Это составные числа, для которых любое  $b$  является свидетелем их простоты. Они были открыты Робертом Кармайклом из Иллинойского университета в 1910 году. Вот несколько первых чисел Кармайкла: 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041. Но у этих чисел обычно имеются небольшие простые множители, поэтому если число проходит 64 теста Миллера–Рабина и при этом не делится ни на одно из первых нескольких сотен простых чисел, то с очень большой вероятностью оно простое.

Существует хороший метод нахождения простых чисел заданного размера, но он не гарантирует, что найденное число будет безопас-

ным, и к тому же гораздо медленнее метода, описанного в этом разделе. Если  $S$  – требуемый размер, то количество испытаний, необходимое для нахождения простого числа такого размера, имеет порядок  $\ln(S)$ . Так что для нахождения 500-значного простого числа потребуется примерно  $\ln(10^{500}) \approx 1151$  испытание, и для каждого из них нужно будет выполнить 64 теста Миллера–Рабина и несколько сотен делений. Использование описанного в этом разделе метода может сэкономить много часов или даже недель машинного времени в зависимости от мощности компьютера и желаемого размера простых чисел.

#### 14.4.2 Построение больших простых чисел, новый подход

Для нахождения большого простого числа можно начать с любого целого  $N$  и пробовать  $2N + 1$ ,  $2N + 3$ ,  $2N + 5$ , ..., проверяя каждое, пока не встретится простое число. Чуть лучшие результаты получаются, если пробовать  $6N + 1$ ,  $6N + 5$ ,  $6N + 7$ ,  $6N + 11$ ,  $6N + 13$ , .... При этом из проверки исключаются числа, кратные 2 и 3. Можно также пробовать  $30N + 1$ ,  $30N + 7$ ,  $30N + 11$ ,  $30N + 13$ , ..., чтобы исключить кратные 2, 3 и 5. Аналогично для  $2 \times 3 \times 5 \times 7 = 210$  и т. д.

Проверить заданное число  $N$  на простоту можно несколькими способами. Самый простой – прямой перебор делителей. Чтобы проверить, является ли число  $N$  простым, пробуем разделить его на каждое простое число, не превышающее  $\sqrt{N}$ . Если хотя бы одно из них делит  $N$ , то  $N$  составное, иначе простое. Перебор делителей полезен для  $N$ , не превышающих  $10^{12}$ , максимум  $10^{14}$ , но потом требует слишком много времени. Большинство других тестов на простоту вероятностные, они могут лишь сказать, что число *вероятно* является простым.

Но есть один тест, который позволяет точно сказать, что число простое: целое число  $N > 1$  является простым, если имеет первообразный корень. Напомню (см. раздел 13.3), что  $g$  называется первообразным корнем из  $N$ , если  $g^{N-1} \bmod N = 1$  и  $g^{(N-1)/p} \bmod N \neq 1$  для любого  $p$ , делящего  $N - 1$ . Чтобы проверить  $N$  на простоту, нужно только вычислить  $g^x \bmod N$  для значений  $x = N - 1$  и  $x = (N - 1)/p$ , где  $p$  – делитель  $N - 1$ . Назовем этот метод *тестом на простоту по первообразным корням*, или для краткости *тестом по корням*. Его придумал в 1876 году французский математик Эдуард Люка – тот самый, который ввел в оборот термин *число Фибоначчи* (раздел 3.4). Люка умер в 1891 году в результате трагического несчастного случая.

Достаточно проверить, являются ли первообразными корнями 2, 3, 5, 7, 11 и 13. Если  $N$  имеет хотя бы один первообразный корень, то с очень высокой вероятностью одно из этих чисел будет таковым. А если ни одно из них первообразным корнем не является, то не тратьте время на проверку других значений, а лучше сразу переходите к следующему кандидату на роль простого числа.

Проблема теста Люка по корням в том, что нужно разложить  $N - 1$  на множители, а если  $N$  содержит 300 и более цифр, то эффективно

сделать это невозможно, по крайней мере без помощи квантового компьютера. Потому-то этот тест и не упоминается в книгах и на сайтах, где обсуждается проверка на простоту.

Но это препятствие можно обойти. Напомним, что наша цель состоит не в том, чтобы отыскать общий способ проверки на простоту, а в том, чтобы получить одно простое число и взять его в качестве модуля в алгоритме Диффи–Хеллмана. Поэтому вместо того чтобы *искать* простое число, мы можем его *сконструировать*.

Фокус заключается в том, чтобы выбрать  $N - 1$  с известными множителями. Например, можно было бы выбрать  $N - 1$  вида  $2^n$ , тогда  $N$  имело бы вид  $2^n + 1$ . Единственным простым множителем  $N - 1$  было бы число 2. Чтобы найти простые числа вида  $2^n + 1$ , нужно только найти такое число  $b$ , что  $b^{N-1} \bmod N = 1$  и  $b^{(N-1)/2} \bmod N \neq 1$ . Я рекомендую попробовать  $b = 2, 3, 5, 7, 11$  и  $13$ . Если ни одно из них не является первообразным корнем, то пропустите  $N = 2^n + 1$  и посмотрите, не будет ли простым  $N = 2^{n+1} + 1$ . Поиск в сети сообщит вам, что среди чисел такого вида простыми являются 3, 5, 17, 257 и 65537. Неизвестно, есть ли еще какие-нибудь, хотя на решение этого вопроса потрачены тысячи часов машинного времени. Эти пять простых чисел называются числами Ферма в честь французского математика Пьера де Ферма, знаменитого своей заметкой на полях книги по поводу уравнения  $a^n + b^n = c^n$ .

## План

Прежде чем переходить к деталям, я хочу в общих чертах описать метод построения большого простого числа  $P$ . Метод должен удовлетворять следующим условиям:

- 1  $P - 1$  должно иметь большой простой множитель, так чтобы  $P$  было безопасным;
- 2 для любого кандидата на роль  $P$  вероятность оказаться простым числом должна быть велика, чтобы минимизировать число проверок на простоту;
- 3  $P - 1$  должно иметь мало различных простых множителей, чтобы каждая проверка на простоту занимала как можно меньше времени.

Любой поиск большого простого числа будет включать проверку сотен, а то и тысяч кандидатов. Обозначим ожидаемое число проверок  $E$ . Наш подход заключается в том, чтобы искать каждого кандидата на роль  $P - 1$  в виде произведения двух чисел,  $sK$ . Коэффициент  $s$  будет пробегать последовательность относительно небольших чисел, как правило, сравнимых с  $E$  по порядку величины. Ядро  $K$  будет либо большим простым числом, либо произведением степеней максимум двух простых чисел  $p^a q^b$ , где хотя бы одно из  $p$  и  $q$  – большое простое число. Сначала рассмотрим выбор коэффициентов, а затем выбор ядра.

## Коэффициенты

Простейший способ выбрать коэффициенты – перебирать простые числа по одному. Поскольку коэффициенты должны быть четными, будем пробовать удвоенные простые числа:  $2 \times 2$ ,  $2 \times 3$ ,  $2 \times 5$ ,  $2 \times 7$ , .... Этот метод, который мы назовем *PickPrimes*, минимизирует число различных простых множителей  $sK$ . Как  $s$ , так и  $K$  имеет не более двух различных простых множителей. Однако *PickPrimes* не сокращает количество необходимых проверок.

Другой способ выбора коэффициентов – искать их в виде  $p^a q^b$ ,  $p^a q^b r^c$  и т. п. Здесь  $p$ ,  $q$  и  $r$  – малые простые числа, например 2, 3 и 5 или 2, 5 и 7. (Ниже в этом разделе мы встретим случай, когда 3 необходимо опустить.) При таком подходе  $P$  не может быть кратно 2, 3 и 5, что существенно увеличивает шансы  $P$  оказаться простым. Если вы решите воспользоваться этим методом, то имеет смысл заранее вычислить и отсортировать список коэффициентов.

## Ядро

Ядро  $K$  должно иметь хотя бы один простой множитель,  $R$ . Я рекомендую выбирать  $R$  не менее  $2^{128} \approx 3.4 \times 10^{38}$ . Если противник располагает квантовым компьютером, то  $R$  должно быть не меньше  $2^{256} = 1.16 \times 10^{77}$ . А откуда же взять эти простые числа? Если вы готовы согласиться на 30-значные числа, то кое-что можно найти на сайте [bigprimes.org](http://bigprimes.org).

Если вы ожидаете, что будете генерировать много больших или очень больших простых чисел, то придется приложить усилия самостоятельно. Подготовьтесь, заранее построив таблицу простых чисел разного размера. Назовем ее *PrimeTab*. Сохраните *PrimeTab*, чтобы в тот момент, когда потребуются дополнительные простые числа, не повторять все заново. Можете для начала поместить в таблицу 25 простых чисел, меньших 100. Возможно, вы помните их наизусть, так что просто наберите в программе. Далее можете, если хотите, сгенерировать несколько простых чисел длиной от 3 до 12 цифр, скажем по 2–3 числа каждого размера, воспользовавшись методом перебора делителей. Я рекомендую внести в процесс элемент случайности, чтобы не строить одни и те же простые числа при каждом использовании этого метода (и чтобы все читатели, использующие его, не получали одинаковые числа). На этом этапе *PrimeTab* могла бы содержать примерно 50 простых чисел.

## Конструирование $R$ (метод малых шагов)

Теперь начнем конструировать  $R$ , большой простой множитель  $Q - 1$ . Можно делать малыми шагами, отыскивая простые числа, каждое из которых немного больше последнего найденного, или рвануть вперед одним гигантским прыжком. Если вы рассчитываете сгенери-

ровать много больших простых чисел, то рекомендую остановиться на малых шагах – тогда в PrimeTab накопится много чисел для последующего использования. Для иллюстрации обоих методов будем конструировать  $R$  малыми шагами, а  $Q$  – одним скачком.

Предположим, что PrimeTab содержит  $k$  простых чисел,  $p_1 < p_2 < p_3 < \dots < p_k$ . Для построения следующего простого числа начните с выбора из таблицы любых двух чисел,  $p_i$  и  $p_j$ . Обозначим  $g$  произведение  $p_i p_j$ . Если  $g < p_k$ , то можно было бы взять  $i$  или  $j$  побольше, чтобы не генерировать слишком много малых простых чисел. Конечно, сколько-то малых простых чисел нужно, поэтому я рекомендую брать большее  $i$  или  $j$ , когда  $p_i p_j < p_k^{2/3}$ . Сначала с помощью теста Люка проверим, является ли простым  $R = 2g + 1$ . Это легко, т. к. мы знаем, что единственными простыми множителями  $R - 1$  являются  $2$ ,  $p_i$  и  $p_j$ . Если  $2g + 1$  не простое, пробуем  $4g + 1$ ,  $6g + 1$ ,  $10g + 1$ , ... применяя метод PickPrimes для выбора коэффициентов. Когда количество цифр перевалит за 20, для поиска каждого простого числа может потребоваться 50 или более попыток.

## УМЕНЬШЕНИЕ КОЛИЧЕСТВА ПРОВЕРОК

Когда числа становятся очень большими, мы сможем сэкономить время, проверяя, делится ли каждый кандидат  $ng + 1$  на много малых простых чисел, и только потом переходить к поиску первообразного корня. Например, можно было бы проверить, что  $ng + 1$  не делится ни на одно из первых 100 простых чисел. Эту проверку можно значительно ускорить, если заранее вычислить  $x_i = g \bmod p_i$  для каждого из первых 100 простых чисел. Затем вместо того чтобы вычислять  $(ng + 1) \bmod p_i$ , где  $g$  может содержать несколько сотен цифр, мы вычисляем  $(nx_i + 1) \bmod p_i$ , где  $x_i$  содержит всего от одной до трех цифр. То есть пробное деление на  $(g \bmod p_i)$  производится только один раз, а не по одному разу для каждого значения  $n$ . Назовем этот метод *PrimeCheck*.

Метод PrimeCheck работает, потому что простые числа-кандидаты выбираются последовательно. Прodelать то же самое для традиционного метода поиска больших простых чисел не получится, потому что в нем кандидаты выбираются случайно. Поэтому пробное деление на малые простые числа в PrimeCheck выполняется гораздо быстрее, а раз так, то можно использовать больше малых простых чисел, скажем 300 вместо 100, и тем самым уменьшить число попыток.

Как и раньше, если ни одно из чисел 2, 3, 5, 7, 11, 13 не является первообразным корнем из  $ng + 1$ , то кандидата можно пропустить и перейти к следующему значению  $n$ , пока не найдется очередное простое число. В этом методе для каждого кандидата выполняется 6 проверок, а не 64, как в традиционном, т. е. он в 10 с лишним раз быстрее. Каждое найденное простое число добавляйте в PrimeTab.



## КОНСТРУИРОВАНИЕ $P$ И $Q$ (МЕТОД БОЛЬШОГО СКАЧКА)

Предположим, что наша цель – найти 300-значное простое число Софи Жермен. Будем продолжать заполнение таблицы PrimeTab, пока в ней не окажется хотя бы одно простое число, скажем  $R > 2^{128}$ . Теперь мы готовы сгенерировать 300-значное простое число одним большим скачком. Для начала выберем целевое  $T$  требуемого размера, например  $T = 10^{300}$ . Можно добиться, чтобы  $P$  было сколь угодно близко к целевому значению, но для алгоритма Диффи–Хеллмана это необязательно. Достаточно, чтобы  $T$  было минимально желательного размера.

Следующий шаг – нахождение  $Q$ . Напомним, что  $Q$  должно удовлетворять трем условиям:  $Q$  должно быть простым,  $Q - 1$  должно быть кратным большому простому числу  $R$  и  $P = 2Q + 1$  тоже должно быть простым. Стратегия поиска  $Q$  заключается в том, чтобы выбрать какое-то начальное число  $t$  с известными простыми делителями и пробовать  $2t + 1$ ,  $4t + 1$ ,  $6t + 1$ ,  $10t + 1$ , ..., применяя метод PickPrimes.

**ПРЕДУПРЕЖДЕНИЕ** Если выбрать  $t$  кратным 3, то  $Q$  будет иметь вид  $3x + 1$ . Тогда  $P = 2Q + 1 = 6x + 3$  будет кратно 3. Так что при  $t$ , кратном 3,  $P$  не может быть простым.

Поскольку  $T$  – минимальное значение  $P$ , а  $Q$  близко к  $P/2$ , то  $t$  должно быть близко к  $T/2$ . Для построения  $t$  начнем с наибольшего простого числа в PrimeTab, а именно  $R$ . Возьмем наибольшую степень  $R$ , меньшую  $T/2$ , скажем  $R^r$ . Например, если  $T$  равно  $10^{300}$ , а  $R$  порядка  $10^{40}$ , то  $T/2 = 5 \times 10^{299}$ , так что  $r$  равно 7. Следовательно,  $R^r$  порядка  $10^{280}$ . Переход от  $10^{40}$  прямо к  $10^{280}$  – действительно гигантский скачок. Это  $R^r$  намного меньше  $5 \times 10^{299}$ , поэтому положим  $t = R^r S$ , где  $S$  порядка  $5 \times 10^{19}$ . Когда  $S < 10^{12}$ , можно воспользоваться перебором делителей для нахождения следующего простого числа, большего  $S$ . Если оно равно  $S'$ , то  $t$  равно  $R^r S'$ . Когда  $S > 10^{12}$ , можно положить  $S'$  равным произведению какого-то простого числа из таблицы PrimeTab и выбранного вами простого числа, меньшего  $10^{12}$ , или сделать  $S'$  квадратом либо кубом простого числа. Предположим последнее. В нашем примере  $S \approx 5 \times 10^{19}$ . Квадратный корень из него равен приблизительно 7 071 067 812. Следующее большее простое число равно  $U = 7\,071\,067\,851$ . Поэтому  $t$  будет равно  $R^7 U^2$ .

Итак, теперь мы построили  $t$  и знаем все его простые множители. Можно приступить к поиску  $Q$  путем проверки  $2t + 1$ ,  $4t + 1$ ,  $6t + 1$ ,  $10t + 1$ , ... с помощью теста по корням. Для случайно выбранного числа  $N$  вероятность оказаться простым приближенно равна  $1/\ln(N)$ . Если  $N$  порядка  $10^{300}$ , то  $\ln(N) \approx 690$ . Это значит, что понадобится примерно 690 попыток, чтобы найти простое число вида  $nt+1$ . Необходимо также, что  $P$  было простым, а вероятность этого тоже приближенно равна  $1/690$ . Это значит, что всего понадобится  $690^2 =$



476 100 попыток, чтобы найти такие числа  $Q = nt+1$  и  $P = 2Q+1$ , которые одновременно будут простыми. Очень много.

Эти проверки занимают время, поэтому ценна любая идея, позволяющая уменьшить их количество. В данном случае мы можем воспользоваться естественным обобщением PrimeCheck. Для каждого простого  $p_i$  вычисляем  $x_i = t \bmod p_i$ , как и прежде. Для каждого значения  $n$  проверяем, делится ли  $nx_i + 1$  на  $p_i$ , чтобы убедиться, что  $Q$  не является кратным  $p_i$ , а также проверяем, делится ли  $2(nx_i + 1) + 1 = 2nx_i + 3$  на  $p_i$ , чтобы убедиться, что  $P$  не является кратным  $p_i$ . Таким образом, мы получаем от списка  $x_i$  двойную выгоду.

## СЕКРЕТНЫЕ ПРОСТЫЕ ЧИСЛА

В некоторых шифрах требуется использовать секретное простое число, известное только вам и вашим корреспондентам. Для построения такого числа можно использовать методы, описанные в этом разделе, но нужна уверенность, что противник не сможет повторить те же шаги и раскрыть ваше число. Я рекомендую две меры предосторожности. (1) При инициализации таблицы PrimeTab вместо 2–3 простых чисел длиной от 3 до 12 цифр выберите случайным образом 5–10 простых чисел длиной от 3 до 14 цифр. Ваша цель – поместить в PrimeTab по меньшей мере 100 начальных простых чисел. (2) Применяйте метод малых шагов для построения  $P$ ,  $Q$  и  $R$ , желательно использовать как минимум 100 шагов в дополнение к начальным простым числам.

## Точный РАЗМЕР

Метод большого скачка для конструирования простых чисел легко модифицировать для нахождения чисел точно заданного размера. Приведу пример. Пусть требуется найти простое число от  $10^{300}$  до  $1.1 \times 10^{300}$ . Выберем  $r$  немного меньшим, чем  $10^{300}/2000000$ , т. е.  $5 \times 10^{294}$ . Воспользуемся методом PickPrimes, но начнем с простых чисел, больших 1000000, т. е. 1000003, 1000033, 1000037, 1000039, .... Для уменьшения количества проверок будем использовать метод PrimeCheck.

Существует примерно 6700 простых чисел между 1 000 000 и 1 100 000 и примерно одно из каждых 690 чисел между  $10^{300}$  и  $1.1 \times 10^{300}$  простое, поэтому мы почти наверняка найдем простое число требуемого размера. Вероятность легко вычислить. Вероятность, что любое наперед заданное число в интересующем нас диапазоне не простое, равна 689/690. Вероятность, что все 6700 выбранных чисел не простые, равна  $(689/690)^{6700} \approx 0.00006$ . Следовательно, вероятность успеха равна 99.994 %. \*\*

# 15

## Матричные методы

---

### Краткое содержание главы:

- шифры, в которых используется умножение на матрицу целых чисел или на матрицу элементов кольца;
- шифры, в которых используется умножение на большие и малые числа;
- решение сравнений первой степени;
- построение колец и обратимых матриц.

Матрицы находят полезные применения в криптографии, потому что с их помощью можно шифровать сколько угодно большие блоки текста за одну операцию. Обычно каждый блок сообщения рассматривается как вектор байтов, т. е. целых чисел по модулю 256.



Если Сандра использует матрицу для шифрования сообщения, то Рива должна использовать обратную матрицу для его дешифиро-

вания. Начнем обсуждение матричных методов с техники обращения матриц.

## 15.1 Обращение матрицы

Существует несколько способов решения матричного уравнения вида  $C = AP$ , когда открытый текст известен. Поскольку Эмили знает  $P$  и  $C$ , но не знает  $A$ , она может решить уравнения относительно  $A$ , умножив обе части справа на  $P'$ ; тогда  $CP' = APP' = A$ . Следовательно, Эмили нужно обратить  $P$ . У Ривы обратная задача. Она знает  $A$ , но не знает  $P$ , поэтому должна обратить  $A$ . Умножая обе части слева на  $A'$ , она получает  $A'C = A'AP = P$ .

У описанного здесь метода есть то преимущество, что обратная матрица получается непосредственно, без промежуточного шага обратной подстановки, необходимого в других методах. Идея метода заключается в том, чтобы расположить заданную матрицу бок о бок с единичной, так чтобы образовалась матрица  $n \times 2n$  двойной ширины. Левая часть приводится к единичной матрице с помощью одних лишь элементарных операций над строками. Эти операции применяются к каждой строке двойной матрицы, так что левая половина преобразуется из исходной матрицы в единичную, а правая – из единичной в обратную исходной.

Под элементарными понимаются следующие операции над строками: (1) умножение строки на число, отличное от нуля; (2) перестановка двух строк; (3) вычитание одной строки, умноженной на число, из другой строки.

Алгоритм преобразует элементы исходной матрицы в элементы единичной матрицы по одному, начиная с левого верхнего угла и спускаясь вниз по самому левому столбцу. Затем то же самое делается для второго столбца и т. д. Если в какой-то момент алгоритм заходит в тупик, т. е. все элементы активного столбца кратны 2 или 13, то матрица необратима. Если такое происходит с Сандрой, то ей следует попробовать другую матрицу  $A$ . Часто достаточно прибавить 1 к какому-то элементу нижней строки. Если такое приключается с Эмили, значит, ей нужно еще  $n$  элементов открытого текста. Это даст ей матрицу размера  $(n+1) \times n$ . После применения описанного алгоритма обратная матрица окажется в правом верхнем углу размера  $n \times n$  матрицы двойной ширины.

Приведем пример для случая  $3 \times 3$ . Эта матрица применяется к 26-буквенному английскому алфавиту, поэтому элементами матрицы являются целые числа по модулю 26. Ни дроби, ни отрицательные числа не возникают. Поскольку все делается по модулю 26, всякий элемент, не кратный ни 2, ни 13, имеет мультипликативный обратный. Это позволяет преобразовать первый ненулевой элемент в каждой строке в 1 и, стало быть, без труда решить, какое кратное строки вычитать из любой другой строки. Исходная матрица равна

$$\begin{pmatrix} 13 & 4 & 11 \\ 6 & 15 & 1 \\ 10 & 9 & 22 \end{pmatrix}.$$

Для перехода к матрице двойной ширины дописываем справа единичную матрицу  $3 \times 3$ :

$$\begin{pmatrix} 13 & 4 & 11 & 1 & 0 & 0 \\ 6 & 15 & 1 & 0 & 1 & 0 \\ 10 & 9 & 22 & 0 & 0 & 1 \end{pmatrix}.$$

Проблема возникает сразу же, потому что первые элементы во всех строках необратимы. Я сделал это специально, чтобы продемонстрировать весьма полезный трюк. Первый элемент в строке 1 кратен 13, но не кратен 2. Первый элемент в строке 2 кратен 2, но не кратен 13. Если просто прибавить строку 2 к строке 1, то первый элемент станет равен 19; он не кратен ни 2, ни 13, а стало быть, обратим. Проблема решена.

$$\begin{pmatrix} 19 & 19 & 12 & 1 & 1 & 0 \\ 6 & 15 & 1 & 0 & 1 & 0 \\ 10 & 9 & 22 & 0 & 0 & 1 \end{pmatrix}.$$

Обратным элементом для 19 по модулю 26 является 11, потому что  $19 \times 11 = 209 \equiv 1 \pmod{26}$ . Умножим первую строку на 11, чтобы сделать первый элемент матрицы равным 1:

$$\begin{pmatrix} 1 & 1 & 2 & 11 & 11 & 0 \\ 6 & 15 & 1 & 0 & 1 & 0 \\ 10 & 9 & 22 & 0 & 0 & 1 \end{pmatrix}.$$

Теперь мы можем закончить работу со столбцом 1, для чего вычтем первую строку, умноженную на 6, из второй, и первую строку, умноженную на 10, из третьей. Тогда первые элементы второй и третьей строки станут равны 0:

$$\begin{pmatrix} 1 & 1 & 2 & 11 & 11 & 0 \\ 0 & 9 & 15 & 12 & 13 & 0 \\ 0 & 25 & 2 & 20 & 20 & 1 \end{pmatrix}.$$

Перейдем ко второй строке. Первый ненулевой элемент во второй строке – 9. Обратным к нему является 3, потому что  $9 \times 3 = 27 \equiv$

1 mod 26. Умножим вторую строку на 3, чтобы первый элемент в ней стал равен 1:

$$\begin{pmatrix} 1 & 1 & 2 & 11 & 11 & 0 \\ 0 & 1 & 19 & 10 & 13 & 0 \\ 0 & 25 & 2 & 20 & 20 & 1 \end{pmatrix}.$$

Теперь, чтобы покончить со вторым столбцом, нужно вычесть вторую строку из первой и вычесть вторую строку, умноженную на 25, из третьей. Обратите внимание, как левая часть матрицы двойной ширины постепенно преобразуется в единичную матрицу.

$$\begin{pmatrix} 1 & 0 & 9 & 1 & 24 & 0 \\ 0 & 1 & 19 & 10 & 13 & 0 \\ 0 & 0 & 21 & 4 & 7 & 1 \end{pmatrix}.$$

Почти готово. Первый ненулевой элемент в третьей строке – 21. Обратным к нему является 5, поэтому умножаем нижнюю строку матрицы на 5.

$$\begin{pmatrix} 1 & 0 & 9 & 1 & 24 & 0 \\ 0 & 1 & 19 & 10 & 13 & 0 \\ 0 & 0 & 1 & 20 & 9 & 5 \end{pmatrix}.$$

Чтобы покончить с третьим столбцом, нужно вычесть третью строку, умноженную на 9, из первой и третью строку, умноженную на 19, из второй.

$$\begin{pmatrix} 1 & 0 & 0 & 3 & 21 & 7 \\ 0 & 1 & 0 & 20 & 24 & 9 \\ 0 & 0 & 1 & 20 & 9 & 5 \end{pmatrix}.$$

Всё! Теперь в левой половине находится единичная матрица, а в правой – матрица, обратная исходной. Это легко проверить, умножим исходную матрицу на обратную к ней. Результатом должна быть единичная матрица – и так оно и есть.

$$\begin{pmatrix} 13 & 4 & 11 \\ 6 & 15 & 1 \\ 10 & 9 & 22 \end{pmatrix} \begin{pmatrix} 3 & 21 & 7 \\ 20 & 24 & 9 \\ 20 & 9 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

## 15.2 Матрица перестановки

Начнем с очень простого матричного метода, а именно *матрицы перестановки*. Это квадратная матрица, в каждой строке и каждом столбце которой находится ровно одна единица. Все остальные элементы матрицы равны 0. Чтобы переставить блок из 10 букв, мы можем представить его в виде матрицы размера  $1 \times 10$  и умножить справа на матрицу перестановки размера  $10 \times 10$ . В результате получится матрица  $1 \times 10$  с переставленными буквами.

Чтобы переместить букву из позиции 2 блока в позицию 5, следует записать 1 в элемент на пересечении второй строки и пятого столбца. Следующая матрица перестановки  $4 \times 4$  изменяет блок ABCD на BADC:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Сами по себе матрицы перестановки не особенно полезны, но если имеется матрица  $M$  для выполнения подстановки блока и матрица перестановки  $T$ , то обе операции можно выполнить за один шаг, используя матрицу  $MT$  вместо  $M$ .

## 15.3 Шифр Хилла

Самый ранний шифр, основанный на матрицах, – *шифр Хилла*, предложенный в 1929 году Лестером С. Хиллом из Хантерского колледжа и опубликованный в журнале «American Mathematical Monthly». Похожий шифр изобрел в 1924 году тогда еще подросток Джек Левин, впоследствии работавший в Каролинском колледже. Он опубликовал его в 1926 году в журнале «Flynn's Weekly», специализирующемся на низкопробных детективах. Колонку, посвященную криптографии, в нем вел М. Э. Охавер, изобретатель фракционированного кода Морзе (раздел 4.4). Кстати говоря, именно в «Flynn's Weekly» Кенделл Фостер Кроссен, также упоминавшийся в разделе 4.4, опубликовал многие свои рассказы. На протяжении всей своей карьеры Левин пытался принизить шифр Хилла и продвинуть свой собственный.

Шифр Хилла применяется к 26-буквенному алфавиту, в котором буквы занумерованы от 0 до 25 в перемешанном порядке. То есть перед матричной операцией выполняется простая подстановка. Буквы открытого текста берутся блоками по 3. Один блок образует вектор-столбец, т. е. матрицу  $P$  размера  $3 \times 1$ . Этот вектор-столбец умножается слева на матрицу  $A$  размера  $3 \times 3$ , к результату прибав-

ляется вектор-столбец  $B$ , и получается вектор шифртекста  $C$ . В матричной нотации это записывается в виде  $C = AP + B$ , где сложение и умножение производятся по модулю 26. Наконец, мы преобразуем числа обратно в буквы, пользуясь тем же соответствием между буквами и цифрами.

К сожалению, многие авторы под *шифром Хилла* понимают его ослабленную версию. Чтобы устранить эту путаницу, пронумеруем несколько версий шифра Хилла. Хилл-0 – самая слабая версия. Используется стандартный английский алфавит без перемешивания, и вектор  $B$  опускается, т. е.  $C = AP$ . Шифр Хилл-1 немного более стойкий. Алфавит по-прежнему не перемешан, но вектор  $B$  ненулевой. Хилл-2 – та версия, которая была предложена самим Хиллом. Алфавит перемешан, и вектор  $B$  ненулевой. Версия Хилл-3 еще более стойкая, в ней для преобразования букв в цифры используется один перемешанный алфавит, а для обратного преобразования цифр в буквы – другой. Это уже сравнимо с шифром Bifid с сопряженной матрицей из раздела 9.6.1.

Оригинальный шифр самого Хилла, Хилл-2, был, по существу, шифром с засекреченным методом. Преобразование букв в цифры и обе матрицы  $A$  и  $B$  были фиксированы. Никакого ключа не предполагалось. Всякий, кто знал метод, мог читать сообщения с той же легкостью, что и полномочный получатель. В большинстве книг и сайтов, где обсуждается шифр Хилла, перемешанный алфавит игнорируется, а все внимание уделяется матричным операциям. Это было бы допустимо, если бы использовалось фиксированное преобразование букв в цифры, поскольку известное перемешивание алфавита легко исключается.

Сначала рассмотрим версию Хилл-0, где  $A$  – неизвестная матрица  $n \times n$ , а вектор  $B$  равен 0, т. е.  $C = AP$ . Это самое обычное умножение матриц, которое мы видели в разделе 11.3. Рива может дешифровать сообщение, умножив шифртекст на матрицу  $A'$ , обратную  $A$ . Обратная матрица обладает свойством  $AA' = A'A = I$ , где  $I$  – единичная матрица (элементы на диагонали равны 1, а все остальные равны 0). Единичная матрица сродни числу 1 в обычном умножении, где  $1 \times N = N \times 1 = N$  для любого числа  $N$ . В случае матриц соответствующее тождество имеет вид  $IA = AI = A$  для любой квадратной матрицы  $A$ . Эмили тоже сможет дешифровать сообщение, если сумеет определить  $A'$ .

Версия шифра Хилла с  $B = 0$  уязвима к атаке с известным открытым текстом. Если Эмили известно  $n^2$  символов открытого текста, то она может организовать их в виде матрицы  $n \times n$ . Назовем эту матрицу  $P$ , а соответствующую матрицу шифртекста  $C$ . Тогда  $C = AP$ , где  $C$ ,  $A$  и  $P$  – матрицы целых чисел по модулю 26 размера  $n \times n$ . Решить это матричное уравнение можно несколькими способами. Один из них описан в разделе 15.1.1.

Если вектор  $B$  ненулевой, то Эмили просто нужно узнать еще  $n$  символов открытого текста, и тогда она сможет исключить  $B$  из уравне-



ния. Дополнительные известные символы образуют вектор-столбец  $P_2$ , а соответствующие символы шифртекста – столбец  $C_2$ . Эти векторы можно вычесть из обеих частей уравнения:  $(C - C_2) = A(P - P_2)$ . Форма этого уравнения такая же, как у уравнения  $C = AP$ , и решается оно точно так же – обращением матрицы  $P - P_2$ . Здесь мы вычитаем вектор-столбец  $n \times 1$  из матрицы  $n \times n$ ; для этого первый элемент вектора вычитается из каждого элемента первой строки, второй – из каждого элемента второй строки и т. д.:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} - \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} a-x & b-x & c-x \\ d-y & e-y & f-y \\ g-z & h-z & i-z \end{pmatrix}.$$

Предположим, что открытый текст неизвестен. Но и тогда версию Хилл-0 можно вскрыть. Будем по-прежнему считать, что секретная матрица имеет размер  $3 \times 3$  и  $C = AP$ . Умножая слева на матрицу  $A'$ , обратную к  $A$ , получаем  $P = A'C$ . В каждом блоке сообщения первый символ блока открытого текста зависит только от верхней строки  $A$ . Возможностей всего  $26^3 = 17\,576$ , их легко перебрать. Каждая комбинация букв в верхней строке определяет буквы в позициях 1, 4, 7, ... открытого текста. Для всех таких комбинаций подсчитаем частоты букв.

Эти частоты можно сравнить со стандартными частотами букв в английском тексте, применив метод высоких пиков, описанный в разделе 5.9.1. Возьмем комбинации с наилучшим совпадением, скажем 1 %, или 175 лучших комбинаций. Сделаем то же самое для второй и третьей букв каждого блока, используя вторую и третью строки обратной матрицы  $A'$ . Это даст 175 правдоподобных комбинаций для каждой из трех строк. Теперь можно попробовать скомбинировать эти комбинации и получить возможные реконструкции всего сообщения. Нужно перебрать всего лишь  $175^3 \approx 5.36 \times 10^6$  комбинаций. Комбинация для первой строки дает первую букву в каждом блоке, для второй строки – вторую букву, а для третьей – третью букву, так что мы имеем все буквы блока.

Теперь можно воспользоваться частотами триграмм и определить наиболее вероятный открытый текст. Используйте все триграммы – не только совпадающие с трехбуквенными блоками, но и пересекающие границы блоков. Процесс такой же, как в разделах 5.10 и 8.2, поэтому не буду повторяться. Если удовлетворительный результат получить не удалось, вернитесь в начало и возьмите 2 % или 350 лучших комбинаций для каждой буквы.

Шифр Хилл-1 с секретной матрицей перестановки  $3 \times 3$  и секретным аддитивным вектором  $3 \times 1$ , но без перемешивания алфавита, получает оценку 3. Если производится подстановка, перемешанная ключом, до и после матричных операций, то оценка повышается до 5. Его можно вскрыть, как подстановочный триграммный

шифр общего вида. Оценка повышается с увеличением размера матрицы. Чтобы достичь оценки 10, размер матрицы должен быть не меньше  $8 \times 8$ , а матричную операцию Хилл-3 следует применять дважды. Опишем шаги такого шифра. (1) Преобразовать сообщение в числовую форму с помощью нелинейной подстановки с ключом. (2) Умножить каждый блок на матрицу и прибавить вектор-столбец. (3) Применить к числам вторую нелинейную подстановку. (4) Умножить каждый блок на вторую матрицу и прибавить второй вектор-столбец. (5) Преобразовать результат обратно в буквы, применив третью нелинейную подстановку с ключом. Обе матрицы можно фиксировать, но три ключа для перемешивания алфавитов и оба вектора-столбца следует изменять для каждого сообщения. Назовем этот шифр *DoubleHill*.

## 15.4 Шифр Хилла, компьютерные версии

Шифр Хилла оказался слишком трудным для шифрования вручную. Хилл сконструировал также механическое устройство для шифрования и дешифрирования. Это было сделано во исполнение патентного законодательства того времени, которое разрешало патентовать машину, но не математический алгоритм. И все равно шифр почти не нашел практического применения.

Но теперь, с наступлением эры компьютеров, шифр Хилла снова стал практически полезным. Умножение матриц – детская игра для компьютера. И работать он может не с матрицами  $3 \times 3$ , а с матрицами  $10 \times 10$ . Вместо 9 известных символов открытого текста Эмили понадобится 100, чтобы организовать атаку с открытым текстом. Это почти нереально, разве что путем шпионажа или перехвата сообщений на поле боя. Шифр Хилла с секретной матрицей  $10 \times 10$  и стандартным алфавитом получает оценку 6. А с перемешанной ключом подстановкой до и после умножения матриц – оценку 8.

Шифр Хилла можно дополнительно укрепить, если завести несколько матриц и выбирать какую-то для каждого блока периодически либо случайно. Размеры матриц и блоков открытого текста могут изменяться. Поскольку операция умножения матриц не коммутативна, мы почти наверняка получим разные результаты при умножении на матрицу слева или справа. Блок открытого текста следует рассматривать как вектор-столбец при умножении слева и как вектор-строку при умножении справа. Это значит, что шифр можно сделать еще более стойким, если умножать с разных сторон, меняя их случайно или периодически. Переменные матрицы, переменные размеры блоков, переменные стороны – можете выбрать что-то одно или все сразу.

Можно также сочетать с шифром Хилла перестановку, однако не всякая перестановка повышает стойкость. Предположим, что ис-

пользуется версия Хилл-0 или Хилл-1 и после умножения на матрицу переставляются буквы в каждом блоке. Это то же самое, что использовать шифр Хилла с разными матрицами-множителями. Обозначим перестановку  $T$ . Применение  $T$  после шифра Хилл-1 дает  $C = T(AP + B) = (TA)P + (TB)$ . И чего же мы достигли? Просто вместо матрицы  $A$  используется  $TA$ , а вместо вектора  $B$  –  $TB$ . Эмили может вскрыть этот шифр, имея известный открытый текст, и даже не узнает, что была какая-то перестановка. Если вы хотите использовать перестановку с шифрами Хилл-0 или Хилл-1, то нужно переставлять буквы между блоками или разные буквы в разных блоках.

Как ни странно, ситуация точно такая же при использовании шифров Хилл-2 и Хилл-3. Дело в том, что простая подстановка и перестановка коммутируют. Если  $S$  – произвольная простая подстановка,  $T$  – произвольная перестановка, а  $M$  – произвольное сообщение, то  $S(T(M)) = T(S(M))$  и потому  $ST = TS$ . Следовательно, для любой версии шифра Хилла, если вы собираетесь добавить шаг перестановки, обязательно переставляйте буквы между блоками или разные буквы в разных блоках, меняя их периодически или псевдослучайно.

Еще одна идея – умножать сообщение на матрицы с обеих сторон. Как уже отмечалось, блок текста следует рассматривать как вектор-столбец при умножении на матрице слева и как вектор-строку при умножении справа. Допустим, что используются матрицы  $3 \times 3$  и аддитивная матрица  $B = 0$ . При умножении на матрицу с одной стороны выражение для каждого символа шифртекста состоит из трех членов, в формировании каждого из которых участвует один символ открытого текста и один элемент матрицы. Если умножать с обеих сторон, то в выражении для каждого символа шифртекста будет девять членов, и каждый формируется с участием одной буквы открытого текста и произведения двух элементов матриц. Поэтому коэффициенты при буквах открытого текста квадратичные. Из 81 возможного квадратичного коэффициента в этих выражениях могут встречаться 27.

Для шифров Хилл-0 и Хилл-1 Эмили по-прежнему может решить эти уравнения, зная открытый текст. Есть легкий и трудный способы. Трудный заключается в том, чтобы использовать 18 символов известного открытого текста для решения 18 квадратных уравнений относительно 18 неизвестных элементов двух матриц  $3 \times 3$ . Удаchi вам!

Легкий способ – рассматривать каждый из 27 квадратичных коэффициентов как отдельную переменную. Тогда вместо квадратных уравнений с 18 переменными мы получим линейные с 27 переменными. Не важно, как эти 27 переменных получены из 18 элементов матриц, будем рассматривать их как независимые единицы. Поскольку теперь неизвестных 27, Эмили нужно знать 27, а не 18 букв открытого текста. Маловероятно, но возможно, особенно если она перехватила несколько сообщений и знает, что они зашифрованы

одним и тем же ключом. Например, предположим, что Эмили знает, что все сообщения, отправленные из Швеции, заканчиваются словом STOCKHOLM. Поскольку STOCKHOLM, скорее всего, начинается в разных позициях трехбуквенных блоков, три разных сообщения могут дать ей 27 положений известных букв. Эти 27 уравнений легко решить и найти все 27 коэффициентов.

Далее можно будет без труда решить 27 одночленных квадратных уравнений и найти 18 элементов матрицы – но к чему эти хлопоты? Все связи между буквами открытого текста и буквами шифртекста выражены в терминах 27 квадратичных коэффициентов. А уж как эти коэффициенты получены, Эмили знать необязательно.

Случай Хилл-1 мало чем отличается от Хилл-0. Имеется 36 неизвестных, так что Эмили нужно получить 36 символов открытого текста. В остальном процесс решения точно такой же. Ничего подобного для шифров Хилл-2 и Хилл-3 не существует. Их лучше вскрывать как подстановочные триграммные шифры.

Можно еще повысить стойкость варианта с умножением на матрицы с двух сторон, если использовать матрицы разного размера. Приведу два примера такой техники. В первом случае умножение слева производится на матрицы  $3 \times 3$ , а умножение справа на матрицы  $4 \times 4$ . Поскольку матрицы  $3 \times 3$  зацепляются с матрицами  $4 \times 4$ , как показано на рисунке ниже, назовем такую конфигурацию *зубчатой* (Butthead).

$3 \times 3$	$3 \times 3$	$3 \times 3$	$3 \times 3$
$4 \times 4$	$4 \times 4$	$4 \times 4$	$4 \times 4$

В результате получается блок из 12 символов. Поскольку каждая правая матрица  $4 \times 4$  пересекается с двумя левыми матрицами  $3 \times 3$ , каждый символ шифртекста зависит от шести, а не от четырех символов открытого текста. При такой конфигурации для получения каждого символа шифртекста нужно всего 7 умножений, так что этот метод работает очень быстро. Если перемешанные алфавиты секретны, но матрицы известны, то шифр Butthead получает оценку 6. Если секретны и перемешанные алфавиты, и матрицы, то оценка повышается до 8. И оценка равна 10, если используются матрицы размера  $6 \times 6$  и  $7 \times 7$  или больше. Разумеется, выбранные Сандрой размеры матриц должны быть взаимно простыми числами.

Еще одна рекомендуемая конфигурация шифра с умножением на матрицы с двух сторон – *кирпичная стена* (Brick Wall). В этом случае размеры матриц одинаковы, но сами матрицы смещены на половину ширины, как в кирпичной кладке. См. рисунок ниже.

4x4		4x4		4x4		4x4	
2x2	4x4		4x4		4x4		2x2

Заметим, что границы матриц никогда не совмещаются. В такой конфигурации нет блочной структуры; иначе можно сказать, что все сообщение представляет собой один блок. Поскольку каждая правая матрица  $4 \times 4$  пересекает две левые, каждый символ шифртекста зависит от восьми символов открытого текста. Этого достаточно для обеспечения высокой стойкости.

Если бы мы использовали настоящие матрицы  $2 \times 2$  для первого и последнего блоков, то эти блоки оказались бы слабыми и уязвимыми. Кроме того, понадобились бы матрицы  $1 \times 1$  и  $3 \times 3$  для сообщений некратной длины. Лучше всюду использовать матрицы  $4 \times 4$ . На следующих двух рисунках показано, как это можно сделать для сообщения длины 13. В первом случае мы видим размещение левых матриц, причем последняя матрица выровнена с правым концом сообщения.

A	B	C	D	E	F	G	H	I	J	K	L	M
A	B	C	D									
				E	F	G	H					
								I	J	K	L	
								J	K	L	M	

А здесь показано размещение правых матриц со смещением на два символа. Первая и последняя матрицы  $4 \times 4$  выровнены с концами сообщения.

A	B	C	D	E	F	G	H	I	J	K	L	M
A	B	C	D									
		C	D	E	F							
						G	H	I	J			
								J	K	L	M	

При таком методе расположения последняя левая и последняя правая матрицы выравниваются с границами сообщения. Этого можно избежать, завернув матрицу в начало сообщения, как на рисунке ниже.

A	B	C	D	E	F	G	H	I	J	K	L	M
A	B	C	D									
		C	D	E	F							
						G	H	I	J			
										K	L	M A

Если используется простая подстановка, перемешанная ключом, до умножения на матрицу слева, и другая подстановка после умно-

жения на матрицу справа, а сами матрицы размера  $6 \times 6$  или более держатся в секрете, то шифр Brick Wall получает оценку 10.

Поскольку для обращения матрицы нужно приложить некоторые усилия, быть может, имеет смысл использовать фиксированные матрицы для умножения слева и справа. Это ослабляет шифр, но такое ослабление можно компенсировать добавлением третьей простой подстановки между двумя шагами умножения на матрицу. Матрицы могут быть любого четного размера, начиная от  $6 \times 6$ . По аналогии с шифром Хилла назовем этот шифр *Эверест*<sup>1</sup> (Everest). Шифр Everest получает оценку 10.

## 15.5 Умножение больших целых чисел

Умножение больших целых чисел аналогично умножению матриц в одном важном отношении: при умножении матриц каждый элемент произведения равен сумме произведений двух элементов, взятых из обеих матриц. При умножении больших целых чисел каждая цифра произведения равна сумме произведений двух цифр, по одной из каждого сомножителя. Именно по этой причине я поместил этот раздел в главу о матрицах.

128-битовый блок можно рассматривать как 16 байт или 16 цифр в 256-ичной системе счисления или как 128-битовое целое. При перемножении двух таких 256-ичных целых чисел требуется произвести 256 умножений и 256 сложений (включая переносы). Все гораздо быстрее и проще, если язык программирования позволяет умножить два 32-битовых числа без знака и получить 64-битовое число без знака. Тогда понадобится всего 16 умножений и 16 сложений. А если язык еще и поддерживает умножение 64-битовых чисел с получением 128-битового произведения, то задача дополнительно упрощается.

Существуют и более быстрые методы умножения очень больших чисел, например Карацубы и Тоома–Кука, но выигрыш от их использования при умножении 128-битовых или даже 256-битовых чисел слишком мал, так что я не буду вдаваться в детали. Некоторые языки программирования берут на себя все заботы о таких деталях, так что пользователю вообще не о чем беспокоиться.

А теперь рассмотрим шифр Mult128, в котором сообщение разбивается на 128-битовые блоки и каждый блок умножается на секретное 128-битовое целое  $M$  по модулю  $2^{128}$ . Иными словами, используется только младшая половина 256-битового произведения, а старшая половина отбрасывается. Это означает, что некоторые промежуточные произведения не нужно вычислять, потому что они вносят вклад только в старшую половину. Рива может прочитать со-

---

<sup>1</sup> Игра слов: по-английски Hill – холм. – Прим. перев.

общение, умножив шифртекст на матрицу  $M'$ , обратную  $M$  по модулю  $2^{128}$ . Посмотрим, как такую матрицу найти.

### 15.5.1 Умножение и деление сравнений

Раньше я обещал, что все требуемые сведения из математики буду сообщать по мере необходимости. Вот и настал такой момент. Методы вычисления обратной матрицы опираются на умножение линейных сравнений. Прежде чем показать, как это делается, рассмотрим пример, чтобы понять, в чем проблема. (Часть этого примера уже излагалась в разделе 3.6. Быть может, сейчас стоит перечитать этот раздел.)

Не все сравнения одинаковы. Сильные сравнения имеют единственное решение, слабые – много решений. Чем сильнее сравнение, тем больше информации оно дает. Рассмотрим разные сравнения, начав с самого сильного.

$5x \equiv 1 \pmod{12}$ . Единственное решение  $x \equiv 5 \pmod{12}$ .

$10x \equiv 8 \pmod{12}$ . Два решения  $x \equiv 2, 8 \pmod{12}$ .

$9x \equiv 3 \pmod{12}$ . Три решения  $x \equiv 3, 7, 11 \pmod{12}$ .

$8x \equiv 4 \pmod{12}$ . Четыре решения  $x \equiv 2, 5, 8, 11 \pmod{12}$ .

$6x \equiv 6 \pmod{12}$ . Шесть решений  $x \equiv 1, 3, 5, 7, 9, 11 \pmod{12}$ .

Причина такого различия в том, что во всех сравнениях  $ax \equiv b \pmod{n}$ , кроме первого, параметры  $a$ ,  $b$  и  $n$  имеют общий множитель. В сравнении  $10x \equiv 8 \pmod{12}$  этот общий множитель равен 2, и сравнение имеет два решения. В сравнении  $9x \equiv 3 \pmod{12}$  параметры 9, 3 и 12 имеют общий множитель 3, и решений тоже три. И так далее. Чем больше общий множитель, тем больше решений и тем слабее сравнение.

Если  $a$ ,  $b$  и  $n$  имеют общий множитель  $d$ , то сравнение можно разделить на  $d$ . Например, для  $9x \equiv 3 \pmod{12}$  общий множитель равен 3. Деление на 3 дает сравнение  $3x \equiv 1 \pmod{4}$ . Его можно решить в уме. Решение имеет вид  $x \equiv 3 \pmod{4}$ . Проверяется тривиально:  $3 \times 3 = 9 \equiv 1 \pmod{4}$ . Теперь можно вернуться к исходному сравнению по модулю 12; первое решение равно  $3 \pmod{12}$ , а два других получаются прибавлением  $12/3 = 4$ , т. е. равны  $7 \pmod{12}$  и  $11 \pmod{12}$ .

Резюмирую: если  $a$ ,  $b$  и  $n$  имеют общий делитель  $d$ , то сравнение имеет  $d$  различных решений. Первое решение имеет вид  $a/d \equiv b/d \pmod{n/d}$ , а остальные отстоят друг от друга на  $n/d$ .

Рассмотрим еще две ситуации. Снова предположим, что  $ax \equiv b \pmod{n}$  и что  $a$  и  $n$  имеют общий множитель, который не делит  $b$ , например  $3x \equiv 7 \pmod{30}$ . Такое сравнение не имеет решений. Предположим, с другой стороны, что  $a$  и  $b$  имеют общий множитель  $d$ , который не делит  $n$ . Тогда можно разделить  $a$  и  $b$  на  $d$ . Например, если  $10x \equiv 25 \pmod{37}$ , то  $2x \equiv 5 \pmod{37}$ . Это сравнение можно решить



в уме, прибавив 37 к 5, так что получится  $2x \equiv 42 \pmod{37}$ . Деление на 2 дает  $x \equiv 21 \pmod{37}$ .

$a$  и  $b$  можно не только делить, но и умножать на постоянную,  $m$ . При этом  $m$  не должно иметь общих множителей с  $n$ . Иными словами,  $m$  должно быть обратимо по модулю  $n$ . В противном случае мы сделаем сравнение слабее и потеряем часть информации. Например, пусть дано сравнение  $9x \equiv 3 \pmod{12}$ . Это слабое сравнение, имеющее 3 решения. Если умножить  $a$  и  $b$  на 2, то сравнение примет вид  $18x \equiv 6 \pmod{12}$ , что эквивалентно сравнению  $6x \equiv 6 \pmod{12}$ , имеющему 6 решений. Слабое сравнение стало еще слабее.

Сравнения по одному и тому же модулю можно складывать и вычитать. Пусть имеются сравнения  $ax \equiv b \pmod{n}$  и  $cx \equiv d \pmod{n}$ . Тогда  $(a+c)x \equiv b+d \pmod{n}$  и  $(a-c)x \equiv b-d \pmod{n}$ . Это можно использовать для усиления системы слабых сравнений. Рассмотрим, к примеру, сравнения  $9x \equiv 3 \pmod{12}$  и  $8x \equiv 4 \pmod{12}$ . Первое имеет три решения, второе – четыре. Сложив их, получим сравнение  $17x \equiv 7 \pmod{12}$ , которое сводится к  $5x \equiv 7 \pmod{12}$  и имеет единственное решение,  $x \equiv 11 \pmod{12}$ . Можно поступить еще умнее и вычесть одно сравнение из другого, тогда получится  $(9-8)x \equiv (3-4) \pmod{12}$ , т. е. мы сразу получаем  $x \equiv 11 \pmod{12}$ .

## \*15.6 Решение линейных сравнений

Теперь, когда мы знаем, как производить действия над сравнениями, не ослабляя их, можно заняться задачей о решении линейного сравнения  $ax \equiv b \pmod{m}$ , где  $a$ ,  $b$  и  $m$  – заданные постоянные, а  $x$  – неизвестное значение. В частном случае  $b = 1$   $x$  является обратным к  $a$  по модулю  $m$ . В большинстве учебников упоминается только один метод – *расширенный алгоритм Евклида*. (Алгоритм Евклида обычно приписывают Теэтету Афинскому, который жил на сто лет раньше Евклида.) Это очень хороший метод. И он вполне может применяться, когда модуль мал или имеет несколько различных малых простых множителей. Его, безусловно, стоит рекомендовать, когда разложение на множители неизвестно или достаточно велика вероятность наличия малых множителей.

Однако в криптографии есть всего два часто встречающихся случая, когда требуется вычислять обратное значение по модулю: когда модуль простой и когда он является степенью 2. В этом разделе описывается более прямой метод.

### 15.6.1 Приведение сравнения

Основной метод решения сравнения  $ax \equiv b \pmod{m}$  – повторяющееся уменьшение коэффициента при  $x$ . Самый простой метод называется *ResM*. Идея в том, чтобы умножить сравнение на такое целое

число  $n$ , что  $a(n-1) < m \leq an$ . Найти его можно, округлив частное  $m/a$  с избытком до целого, так что 2.0000 остается равным 2, но 2.0001 становится равным 3. В результате приведения по модулю  $m$  коэффициент становится меньше.

Начнем с простого примера применения ResM, чтобы понять, как он работает, – так будет проще следить за дальнейшим обсуждением уже не столь простых вещей. Рассмотрим сравнение  $38x \equiv 55 \pmod{101}$ . Мы знаем, что  $101/38 = 2.658$ , поэтому умножим обе части сравнения на 3 и приведем по модулю 101:

$$3 \times 38x \equiv 3 \times 55 \pmod{101}, \text{ или } 114x \equiv 165 \pmod{101}, \\ \text{что приводится к } 13x \equiv 64 \pmod{101}.$$

Заметим, что коэффициент при  $x$  уменьшился с 38 до 13. Повторим приведение.  $101/13 = 7.769$ , поэтому умножаем сравнение на 8:

$$8 \times 13x \equiv 8 \times 64 \pmod{101}, \text{ или } 104x \equiv 512 \pmod{101}, \\ \text{что приводится к } 3x \equiv 7 \pmod{101}.$$

Мы почти у цели.  $101/3 = 33.667$ , поэтому умножаем последнее сравнение на 34, в результате чего коэффициент при  $x$  приводится к 1:

$$34 \times 3x \equiv 34 \times 7 \pmod{101}, \text{ или } 102x \equiv 238 \pmod{101}, \\ \text{что приводится к } x \equiv 36 \pmod{101}.$$

Этот результат можно проверить, подставив  $x = 36$  в исходное сравнение,  $38x \equiv 55 \pmod{101}$ . Имеем  $38 \times 36 \equiv 55 \pmod{101}$ , или  $1368 \equiv 55 \pmod{101}$ , это действительно так. Правильный ответ  $x \equiv 36 \pmod{101}$ .

### 15.6.2 Правило половины

Рассмотрим небольшое усовершенствование, которое я назову *правилом половины*. Примерно в половине случаев дробная часть  $m/a$  меньше  $1/2$  и в половине случаев больше. Положим  $q = m/a$ . Тогда в половине случаев  $q$  ближе к  $a$ , а в половине –  $(q+1)a$ .

Поясню на числовом примере. Пусть  $m = 101$ ,  $a = 40$ . Тогда  $q = 101/40 = 2.525$ . Дробная часть 0.525 больше  $1/2$ . Если умножить 40 на 2, то результат, 80, будет на 21 меньше, чем 101. А если умножить 40 на 3, то результат, 120, будет на 19 больше, чем 101. Таким образом,  $40 \times 3$  ближе к 101, чем  $40 \times 2$ . Следовательно,  $n = 3$  – наилучший множитель.

Теперь предположим, что  $a = 41$ . Тогда  $m/a = 101/41 = 2.463$ . На этот раз дробная доля 0.463 меньше  $1/2$ . При этом  $41 \times 2 = 82$  на 19 меньше, чем 101. А  $41 \times 3 = 123$  на 22 больше, чем 101. Поэтому  $41 \times 2$  ближе к 101, чем  $41 \times 3$ . Следовательно, в этом случае наилучшим множителем является  $n = 2$ .

Итак, если дробная часть  $q = m/a$  меньше  $1/2$ , то на ближе к  $m$  при округлении  $q$  с недостатком, а если она больше  $1/2$ , то на ближе к  $m$  при округлении  $q$  с избытком. Воспользовавшись нотацией функций пола и потолка (раздел 13.3), мы можем записать это так: если  $\text{frac}(q) < 1/2$ , выбирать  $n = [q]$ , а если  $\text{frac}(q) > 1/2$ , выбирать  $n = [q]$ . На первый взгляд, все просто, но есть одно осложнение. Когда  $n = [q]$ , на больше  $m$ , поэтому мы приводим сравнение путем вычитания из него кратных  $m$ , как делали в начале этого раздела. Когда  $n = [q]$ , на меньше  $m$ , поэтому сравнение приводится путем вычитания его из кратных  $m$ .

Начнем со сравнения

$$41x \equiv 90 \pmod{101}.$$

Поскольку  $101/41 = 2.463$ , умножаем сравнение на 2:

$$82x \equiv 180 \pmod{101}.$$

Вычитаем его из кратных 101, а именно

$$101x \equiv 202 \pmod{101}.$$

Поскольку  $101 - 82 = 19$  и  $202 - 180 = 22$ , получаем

$$19x \equiv 22 \pmod{101}.$$

Чтобы проиллюстрировать, чего мы достигли, решим сравнение с применением и без применения правила половины:

Без правила половины	С правилом половины
$135x \equiv 77 \pmod{1009}$	$135x \equiv 77 \pmod{1009}$
$71x \equiv 616 \pmod{1009}$	$64x \equiv 470 \pmod{1009}$
$56x \equiv 159 \pmod{1009}$	$15x \equiv 457 \pmod{1009}$
$55x \equiv 1003 \pmod{1009}$	$4x \equiv 660 \pmod{1009}$
$36x \equiv 895 \pmod{1009}$	$x \equiv 165 \pmod{1009}$
$35x \equiv 730 \pmod{1009}$	
$6x \equiv 990 \pmod{1009}$	
$5x \equiv 825 \pmod{1009}$	
$x \equiv 165 \pmod{1009}$	

Без правила половины приведение потребовало восьми шагов, а с правилом половины – всего четырех. Это соотношение меняется в зависимости от коэффициентов и модуля, но 8:4 – довольно типичный результат. Метод ResM с правилом половины будем называть *ResMH*.

### 15.6.3 Лесенка

Если целые числа очень большие, то это все равно медленно, потому что приходится умножать и делить большие числа. Избежать этого помогает метод лесенки. В нем на каждом шаге используется два сравнения. Вместо того чтобы умножать коэффициент при  $x$  на возрастающие числа, чтобы приблизить его к модулю, метод лесенки предлагает умножать коэффициент в каждом сравнении на малое число, дабы приблизить его к предыдущему коэффициенту. Чтобы начать процесс, необходимо дополнительное сравнение. Мы возьмем для этой цели сравнение  $mx \equiv m \pmod{m}$ , эквивалентное  $0x \equiv 0 \pmod{m}$ .

Рассмотрим пример, в котором участвуют относительно большие числа:

$$\begin{array}{ll} 28338689x \equiv 28338689 \pmod{28338689} & \text{Искусственное начальное} \\ & \text{сравнение} \\ 6114257x \equiv 90926 \pmod{28338689} & \text{Сравнение, подлежащее} \\ & \text{решению} \end{array}$$

Поскольку  $28338689/6114257 \approx 4.635$ , умножаем обе части на 5 и вычитаем одно сравнение из другого:

$$\begin{array}{l} 6114257x \equiv 90926 \pmod{28338689}, \\ 2232596x \equiv 454630 \pmod{28338689}. \end{array}$$

Здесь  $6114257/2232596 \approx 2.739$ , поэтому умножаем на 3 и вычитаем:

$$\begin{array}{l} 2232596x \equiv 454630 \pmod{28338689}, \\ 583531x \equiv 1272964 \pmod{28338689}. \end{array}$$

Продолжая в том же духе, получаем:

$$\begin{array}{l} 101528x \equiv 4637226 \pmod{28338689}, \\ 25637x \equiv 26550392 \pmod{28338689}, \\ 1020x \equiv 16548275 \pmod{28338689}, \\ 137x \equiv 9585163 \pmod{28338689}, \\ 61x \equiv 6129512 \pmod{28338689}, \\ 15x \equiv 25664828 \pmod{28338689}, \\ x \equiv 16824956 \pmod{28338689}. \end{array}$$

Во всех приведенных выше примерах модуль был простым числом. Если модуль составной, то ситуация усложняется. Я не буду рассматривать здесь все сложности. Для криптографии наиболее важен случай, когда модуль является степенью 2, например  $2^{32}$  или  $2^{128}$ . В этом случае модуль, выбираемый на каждом шаге, должен быть

нечетным. Поэтому вместо округления к ближайшему целому мы всегда округляем к нечетному числу. Например, и 3.14, и 3.99 следует округлять до 3. Метод ResMH с лесенкой будем называть *ResMHL*.

### 15.6.4 Цепные дроби

Если имеется два или более линейных сравнений, то коэффициент при  $x$  можно уменьшить гораздо быстрее, применив метод *цепных* (или *непрерывных*) *дробей*. Цепная дробь – это способ аппроксимировать десятичное число дробью с произвольной точностью. Рассмотрим десятичное число  $R = 0.13579$ . Оно заключено между  $1/7$  и  $1/8$ . Точнее,  $R \approx 1/7.3643$ . Это можно записать как  $\frac{1}{7+}0.3643$ . Обратите внимание на знак  $+$  в знаменателе. Он означает, что следующее далее число прибавляется к знаменателю, а не ко всей дроби  $1/7$ .

Дробь  $0.3643$  можно аппроксимировать числом  $1/2.745$ , или  $\frac{1}{2+}0.745$ , так что теперь  $R$  принимает вид  $\frac{1}{7+} \frac{1}{2+}0.745$ . Здесь  $0.745$  очень близко к  $3/4$ , так что аппроксимацию можно записать в виде  $\frac{1}{7+} \frac{1}{2+} \frac{3}{4}$ . Чтобы вернуться к обыкновенной дроби, следует произвести действия в обратном порядке:

$$\frac{1}{7+} \frac{1}{2+} \frac{3}{4} = \frac{1}{7+} \frac{1}{2 \frac{3}{4}} = \frac{1}{7+} \frac{1}{11/4} = \frac{1}{7+} \frac{4}{11} = \frac{1}{7 \frac{4}{11}} = \frac{1}{81/11} = \frac{11}{81}.$$

Дробь  $11/81$  приближенно равна  $0.13580$ , т. е. отличается от  $0.13579$  всего на  $0.00001$ . Как видите, этот метод дает очень хорошие приближения.

Рассмотрим снова пример из раздела 15.4.3:

$$6114257x \equiv 90926 \pmod{28338689}.$$

В качестве второго сравнения воспользуемся описанным выше приемом  $0 = 0$ :

$$28338689x \equiv 28338689 \pmod{28338689}.$$

Здесь  $6114257/28338689$  разлагается в цепную дробь следующим образом:

$$\frac{1}{4+} \frac{1}{1+} \frac{1}{1+} \frac{1}{1+} \frac{1}{2+} \frac{1}{1+} \frac{1}{4+} \frac{1}{1+} \frac{1}{2+} \frac{1}{1+} \frac{1}{24+} \frac{1}{7+} \frac{1}{2+} \frac{1}{4+} \frac{1}{15+}.$$

Хорошее эвристическое правило для получения близкой аппроксимации – остановиться непосредственно перед большим знаменателем, в данном случае 24. Оборвав цепную дробь перед 24, получим

$$\frac{1}{4+} \frac{1}{1+} \frac{1}{1+} \frac{1}{1+} \frac{1}{2+} \frac{1}{1+} \frac{1}{4+} \frac{1}{1+} \frac{1}{2+} \frac{1}{1+},$$

или 241/1117.

Умножая коэффициент 6114257 в первом сравнении на 1117, а коэффициент 28338689 во втором сравнении на 241 и вычитая одно из другого, получаем:

$$\begin{array}{rcl} 6829625069x & \equiv & 101564342 \pmod{28338689} \\ 6829624049x & \equiv & 0 \pmod{28338689} \\ \hline 1020x & \equiv & 101564342 \pmod{28338689} \\ 1020x & \equiv & 16548275 \pmod{28338689} \end{array}$$

Это уменьшает коэффициент при  $x$  с 6114257 до 1020, т. е. примерно в 5994 раза. Таким образом, применение метода цепных дробей резко уменьшает число шагов по сравнению с другими методами. Однако при его использовании возможны сложности, потому что коэффициент на предыдущем шаге может оказаться гораздо больше, чем на следующем, как, например, при переходе от 6829625069 к 1020. Для балансирования коэффициентов можно чередовать шаги методом цепных дробей и методом половины. \*\*

## 15.7 Шифры на основе больших целых чисел

На основе умножения больших целых чисел можно построить много шифров. В разделе 15.3 описан шифр Mult128, подразумевающий разбиение сообщения на 128-битовые блоки. Каждый блок рассматривается как 128-битовое целое и умножается на секретное 128-битовое целое  $M$  по модулю  $2^{128}$ . Чтобы биты хорошо перемешались, все байты множителя должны быть отличны от нуля. Но все равно этот шифр слабый, потому что младшие  $n$  бит каждого блока шифртекста зависят только от младших  $n$  бит открытого текста и младших  $n$  ключа  $M$ . В результате шифрование младшего байта оказывается простой подстановкой. Выполнение простой подстановки до и после умножения не устраняет эту слабость. Аналогично к двум младшим байтам применяется подстановка биграмм, а к трем младшим байтам – подстановка триграмм. Шифр Mult128 получает оценку 3.

Простой и очень быстрый способ исправить проблему младшего байта – объединить старший байт с младшим, например с помощью комбинирующей функции **xors** или **adds**. В этом случае оценка повышается до 5. Еще лучше объединить старшие 8 байт с младшими 8 байтами с помощью **xors** или **adds**. Тогда оценка повысится до 7. Вот пример:

<b>ABCDEFGH</b>	Первые 8 букв используются как ключ
<b>ABCDEFGH I J K L M N O P</b>	Блок открытого текста
<b>ABCDEFGH; 8i=W?6}</b>	После объединения, но перед умножением

Одним из способов укрепить шифр могла бы стать перестановка 16 байт, но при наличии достаточного объема шифртекста Эмили смогла бы определить позицию, в которой изменения минимальны, и пришла бы к выводу, что это переставленный младший байт. Шифр *Mult128* с перестановкой получает оценку 4.

Гораздо более стойкий подход – умножить, переставить и снова умножить. Перестановка должна перемещать слабые младшие байты в старшую половину блока. Вот несколько примеров подходящих перестановок: (1) изменение порядка байтов на противоположный, (2) обмен младшей и старшей половин блока, (3) чередование байтов из младшей и старшей половин в обратном порядке. Если байты нумеруются от старших к младшим и номера записаны в шестнадцатеричном виде, цифрами от 0 до F, то эти три перестановки можно представить в виде:

Обращение	Перестановка половин	Чередование
<b>FEDCBA9876543210</b>	<b>89ABCDEF01234567</b>	<b>F7E6D5C4B3A29180</b>

Если язык программирования позволяет работать с блоком и как с набором 32-битовых слов, и как с набором байтов, то, быть может, быстрее будет изменить порядок всех четырех слов, что приводит к такой перестановке:

**CDEF89AB45670123**

Этот шифр, который мы назовем *MPM128*, получает оценку 7.

Если включить в данный процесс шаги перестановки, то стойкость взлетит до небес. Пусть  $S_1$ ,  $S_2$ ,  $S_3$  и  $S_4$  – четыре независимые хорошо перемешанные ключом перестановки,  $P$  – фиксированная перестановка 5BF4AE39D28C1706, а  $M_1$ ,  $M_2$ ,  $M_3$  – операции умножения на три секретных 128-битовых ключа. Тогда шифр  $S_1M_1PS_2M_2S_3PM_3S_4$ , называемый *Tiger*, получает оценку 10.

## 15.8 Умножение на малое число

Мини-версию шифра *Mult128* можно реализовать с помощью обычного 32-битового умножения без знака. 128-битовый блок рассматривается как четыре 32-битовых целых числа. Каждое из этих целых чисел умножается на секретное 32-битовое целое по модулю  $2^{32}$ . Все четыре множителя должны быть нечетными, иначе последующее дешифрирование будет невозможно. Получается 32-битовый шифр. Чтобы получить 128-битовый шифр, четыре отдельных



4-байтовых произведения можно рассмотреть как 16-байтовый блок и перемешать с применением следующей фиксированной перестановки 16-байтового ключа (раздел 7.6):

**3E9472D8B61CFA50**

За этой перестановкой следует второй шаг умножения, в котором 16-байтовый блок снова рассматривается как четыре 32-битовых целых. Множители могут быть теми же самыми или новыми. Далее следует вторая перестановка и еще один раунд умножения, так что всего имеется три раунда умножения и два раунда перестановки. Этот шифр, называемый Mult32, получает оценку 7. Он гораздо быстрее всех вариантов шифра Mult128.

Будем рассматривать 16 байт 128-битового блока как байтовую матрицу  $4 \times 4$ . Четыре байта в любой строке этой матрицы можно рассматривать как 32-битовое целое. Обычно 4 байта целого читаются слева направо, т. е. самый левый байт считается старшим. Однако можно читать их и в противоположном порядке, считая самый левый байт младшим. Возьмем шестнадцатеричное число 01020304. Если умножить его на шестнадцатеричное число 01010101 по модулю  $2^{32}$  обычным образом, то получится 0A090704. Если же умножить число, записанное в обратном порядке, 04030201, на то же 01010101 по модулю  $2^{32}$ , то получится 0A060301.

Точно так же четыре байта в любом столбце можно рассматривать как 32-битовое целое, читая его сверху вниз или снизу вверх. Назовем два горизонтальных направления Восток и Запад, а два вертикальных – Север и Юг. Умножение строк и столбцов на нечетные 32-битовые числа по модулю  $2^{32}$  в порядке Восток, Север, Запад, Юг приводит к очень тщательному перемешиванию. Для этого необходимо шестнадцать 32-битовых множителей. Общий размер ключа равен  $16 \times 31 = 496$  бит, а не  $16 \times 32 = 512$ , потому что множители должны быть нечетными. Этот шифр, который можно было бы назвать *Compass* (компас), получает оценку 8.

Чтобы увеличить оценку до 10, добавьте один или несколько раундов подстановки, например Восток, Север, подстановка, Запад, Юг. Еще лучше добавить несколько подстановок, например Восток, подстановка, Север, Запад, подстановка, Юг. Назовем этот вариант *CompassS*. Даже при использовании фиксированных, но сильно нелинейных подстановок шифр *CompassS* получает оценку 10.

Другой способ использования умножения на малое целое число – *циклическое умножение*. Занумеруем байты в каждой 32-битовой строке 1, 2, 3, 4 слева направо, т. е. от старшего байта к младшему. Умножим это число на нечетное целое по модулю  $2^{32}$ . Переместим байт 1 в конец – 2, 3, 4, 1. Снова умножим на нечетное целое по модулю  $2^{32}$ . Повторим еще дважды, так чтобы каждый байт побывал в каждой позиции по одному разу. То есть байты берутся в порядке 1234, 2341, 3412 и, наконец, 4123. Это следует сделать для каждой из

четырёх строк байтовой матрицы  $4 \times 4$ . Всего получается 16 умножений и 12 циклических сдвигов.

Затем те же операции применяются к столбцам. Итого имеем 32 умножения и 24 циклических сдвига. Этот шифр с циклическим умножением получает оценку 8. У него может быть до 32 различных 32-битовых множителей, используемых в качестве ключей.

Описанные в этом разделе методы можно различными способами сочетать с методами из раздела 15.4. Приведу лишь один пример, который назову *Mat36*. Разобьём сообщение на блоки по 36 символов, рассматриваемые как девять 32-битовых целых. Они образуют матрицу целых чисел по модулю  $2^{32}$  размера  $3 \times 3$ . Эту матрицу будем умножать на секретную обратимую матрицу размера  $3 \times 3$ . Если просто умножать справа на другую матрицу  $3 \times 3$ , то младшие байты 9 целых чисел будут зашифрованы слабо. Вместо этого мы циклически сдвинем весь 36-байтовый блок влево на 16 позиций, а затем умножим его справа на вторую секретную обратимую матрицу целых чисел размера  $3 \times 3$ . Шифр *Mat36* получает оценку 8.

## 15.9 Умножение по модулю $P$

Если умножение производится по модулю  $2^n$ , то младшие байты оказываются зашифрованы слабо и, чтобы эту слабость устранить, приходится устраивать танцы с бубнами. Эта проблема не возникает, когда умножение производится по модулю простого числа  $P$ . При условии что множитель велик, каждый бит произведения зависит от каждого бита открытого текста. Но возникает другая проблема. Предположим, что выбрано простое число  $P < 2^n$  и множитель  $M$  такой, что  $1 < M < P$ . Это позволяет безопасно умножать значения от 0 до  $P - 1$  на  $M$  по модулю  $P$ , так что Рива может их дешифровать, умножив на число  $M'$ , обратное  $M$ .

Однако значение открытого текста 0 останется неизменным и значения открытого текста от  $P$  до  $2^n - 1$  нельзя умножать безопасно, так как результат может оказаться неоднозначным. Например, при умножении 3 и  $P+3$  на  $M$  по модулю  $P$  получится одинаковый результат, поскольку  $3M \equiv PM + 3M \pmod{P}$ . А это значит, что Рива не сможет сказать, что было в сообщении: 3 или  $P+3$ . Следовательно, значения от  $P$  до  $2^n - 1$  нужно оставлять без изменения. Для этого определим следующую функцию  $\text{modp}$ :

$$\begin{aligned} \text{modp}(x) &= Mx \bmod P, \text{ если } x < P, \\ \text{modp}(x) &= x, \quad \text{если } x \geq P. \end{aligned}$$

Для решения проблемы Сандра могла бы, например, применить ИСКЛЮЧАЮЩЕЕ ИЛИ к секретному значению и открытому тексту. Это приводит к семейству шифров по модулю  $P$ . Остановимся на зна-

чении  $n = 64$  (блок размером 8 байт), простом модуле  $P = 2^{64} - 59 = 18446744073709551557$  и множителе  $M = 39958679596607489$ , тоже простом.

Чтобы зашифровывать 64-битовый блок открытого текста  $B$ , Сандра выбирает секретную 64-битовую постоянную  $C_1$  в качестве ключа и вычисляет  $x = \text{modp}(C_1 \oplus B) + C_1$ . Это первый шифр в семействе. Назовем его *PMod1*. Он получает оценку 5. Второй шифр, *PMod2*, представляет собой две итерации *PMod1*, с использованием второй 64-битовой постоянной,  $C_2$ :

$$\begin{aligned}x_1 &= \text{modp}(C_1 \oplus B) + C_1, \\x_2 &= \text{modp}(C_2 \oplus x_1) + C_2.\end{aligned}$$

Шифр *PMod2* получает оценку 7. Третий шифр в этом семействе, *PMod3*, включает три итерации и получает оценку 9:

$$\begin{aligned}x_1 &= \text{modp}(C_1 \oplus B) + C_1, \\x_2 &= \text{modp}(C_2 \oplus x_1) + C_2, \\x_3 &= \text{modp}(C_3 \oplus x_2) + C_3.\end{aligned}$$

Четвертый член семейства, *PMod4*, получает оценку 10. Полный размер ключа в этом случае равен 256 байтам, в четыре раза больше размера блока:

$$\begin{aligned}x_1 &= \text{modp}(C_1 \oplus B) + C_1, \\x_2 &= \text{modp}(C_2 \oplus x_1) + C_2, \\x_3 &= \text{modp}(C_3 \oplus x_2) + C_3, \\x_4 &= \text{modp}(C_4 \oplus x_3) + C_4.\end{aligned}$$

Все сложения производятся по модулю  $2^n$ , а не  $P$ .

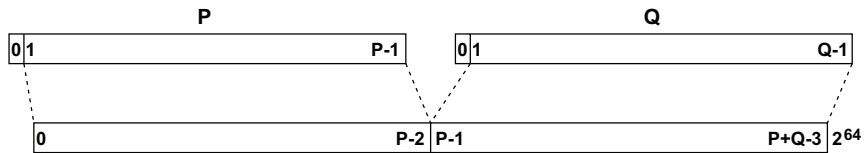
Все четыре шифра *PModX* работают очень быстро, потому что сложение, умножение и деление по модулю 64-битовых целых непосредственно поддерживаются большинством языков программирования. В некоторых компьютерах эти операции вообще выполняются одной командой. Поэтому шифр *PModX* работает с 4 или 8 байтами как с одной единицей, а не обрабатывает каждый 4-битовый блок по отдельности, как в шифре DES. Этот класс шифров идеален для шифрования, реализованного программно. *PMod2* безопаснее DES из-за гораздо большего размера ключа, а *PMod4* безопаснее 3DES.

Есть и другой подход к умножению по модулю  $P$ , не страдающий от проблемы неизменившихся значений. Идея в том, чтобы разбить диапазон целых чисел от 0 до  $2^{64} - 1$  на два отдельных диапазона с разными простыми модулями и множителями. Выберем два простых числа  $P$  и  $Q$  таких, что  $P + Q = 2^{64} + 2$ . Существует приблизительно  $10^{16}$  таких пар, так что подобрать нужную легко, например  $P = 9228410438352162389$  и  $Q = 9218333635357389229$ . Выберем также два больших множителя  $M < P$  и  $N < Q$ . Нетривиальная часть заклю-

чается в том, что нужно сдвигать каждый диапазон, чтобы умножались только числа в диапазоне от 1 до  $P-1$  или от 1 до  $Q-1$ . Для этого переопределим функцию  $\text{modp}$ .

$$\begin{aligned}\text{modp}(x) &= ((x+1)M \bmod P) - 1, & \text{если } x \leq P-2, \\ \text{modp}(x) &= ((x-P+2)N \bmod Q) + P - 2, & \text{если } x > P-2.\end{aligned}$$

При такой функции  $\text{modp}$  шифры PMod1–PMod4 работают так же, как и раньше, и получают такие же оценки. На рисунке ниже показано, как разбивается диапазон от 0 до  $2^{64} - 1$ .



## 15.10 Изменение основания

Изменение основания системы счисления тесно связано с умножением на большое целое число. Для очень больших чисел изменение основания – медленная операция, поэтому наилучшая стратегия – разбить сообщение на блоки и преобразовывать каждый блок отдельно. Изменение основания размывает границу между байтами сообщения.

Есть два метода преобразования чисел из одной системы счисления в другую: от младших разрядов к старшим и наоборот. Большинство из нас изучают эти методы в школе, но потом забывают. Чтобы освежить память читателей, ниже продемонстрированы оба. Сначала мы преобразуем число  $1A87$  одиннадцатеричной системы в десятичную, начиная с младших разрядов, а потом обратно в одиннадцатеричную, начиная со старших разрядов.

В первом случае мы последовательно делим число на новое основание. Каждый остаток становится новой цифрой преобразованного числа. Вот шаги преобразования числа  $1A87$ :

$1A87/7$  дает 312 с остатком 4;  
 $312/7$  дает 49 с остатком 5;  
 $49/7$  дает 7 с остатком 4;  
 $7/7$  дает 1 с остатком 0.

Таким образом,  $1A87_{11} = 10454_7$ .

Во втором случае мы последовательно умножаем старшие цифры на старое основание и прибавляем новую цифру. Вот шаги преобразования числа  $10454$ :

$1 \times 7 + 0$  дает 7;  
 $7 \times 7 + 4$  дает 49;  
 $49 \times 7 + 5$  дает 312;  
 $312 \times 7 + 4$  дает 1A87.

Если у вас возникли сомнения по поводу правильности вычислений, вспомните, что 7, 49, 312 и 1A87 записаны в одиннадцатеричной системе.

На базе изменения основания системы счисления можно построить много красивых шифров. Например, 16-байтовый блок можно также рассматривать как 16-значное число в 256-ичной системе счисления. Преобразуем его в другую систему, скажем по основанию 263. Затем переставим 263-ичные цифры или выполним подстановку или сделаем то и другое сразу. Получившееся число можно было бы преобразовать в 277-ичную систему и проделать то же самое. И напоследок вернуться в 256-ичную систему. Для хранения результата потребуется 17-байтовое число. Использовать можно любое основание от 256 до 362. Напомню, что начальную цифру 0 в любом промежуточном результате нельзя отбрасывать, потому что она необходима Риве для дешифрирования шифртекста.

Если каждое последующее основание лишь немного больше предыдущего, то на каждом этапе понадобится одинаковое количество цифр. Количество цифр увеличивается только на последнем шаге, когда число преобразуется в 256-ичную систему. Основания необязательно должны быть простыми числами.

Опишу идею блочного шифра на базе изменения основания системы счисления. Начинаем с секретной хорошо перемешанной ключом простой подстановки  $S$ . Эта подстановка применяется к байтам, т. е. к целым числам от 0 до 255.  $S$  можно продолжить на основания, большие 256, если оставлять каждую цифру, большую 255, без изменения. Это позволяет обойтись без отдельной таблицы подстановки для каждого возможного основания. Выберем три основания  $B_1$ ,  $B_2$  и  $B_3$ , удовлетворяющие условию  $256 < B_1 < B_2 < B_3 < 363$ . Понадобится также три перестановки 16 элементов  $T_1$ ,  $T_2$  и  $T_3$ . Элементами являются целые числа, которые могут достигать  $B_3 - 1$ , так что для записи каждого понадобится больше одного байта.

Блочный шифр *3Base* включает следующие шаги: (1) подстановка  $S$ ; (2) преобразование в систему по основанию  $B_1$ ; (3) подстановка  $S$ ; (4) перестановка  $T_1$ ; (5) преобразование в систему по основанию  $B_2$ ; (6) подстановка  $S$ ; (7) перестановка  $T_2$ ; (8) преобразование в систему по основанию  $B_3$ ; (9) подстановка  $S$ ; (10) перестановка  $T_3$ ; (11) преобразование в систему по основанию 256; (12) подстановка  $S$ . Блок шифра состоит из 16 элементов на всех шагах вплоть до 11-го. А на шаге 12 размер блока увеличивается до 17 байт. Шифр *3Base* получает оценку 10.

## \*15.11 Кольца

Кольцо – это абстрактная версия целых чисел. То есть кольцо – это множество, элементы которого можно складывать и умножать, как целые числа. Вы уже знакомы с несколькими кольцами: целых чисел, рациональных чисел, вещественных чисел, целых по фиксированному модулю, а возможно, также комплексных и алгебраических чисел. Не столь хорошо известны кольца полиномов с коэффициентами, принадлежащими некоторому кольцу, матриц с элементами, принадлежащими некоторому кольцу, а также чисел вида  $a+b\sqrt{13}$ ,  $a+b\sqrt[3]{7}+c\sqrt[3]{49}$  и  $a+b\sqrt{2}+c\sqrt{3}+d\sqrt{6}$ , где  $a$ ,  $b$ ,  $c$  и  $d$  могут быть целыми, рациональными или числами по фиксированному модулю.

Прежде чем обсуждать использование колец в криптографии, перечислим формальные аксиомы кольца. Сложение в кольце обозначается знаком  $+$  ( $a+b$ ), а умножение – простым расположением элементов рядом ( $ab$ ).

- Для любых элементов кольца  $a$  и  $b$  сумма  $a+b$  и произведение  $ab$  также являются элементами кольца (замкнутость).
- Для любых элементов кольца  $a$ ,  $b$  и  $c$  имеет место равенство  $a+(b+c) = (a+b)+c$  (ассоциативность сложения).
- Для любых элементов кольца  $a$  и  $b$  имеет место равенство  $a+b = b+a$  (коммутативность сложения).
- Существует элемент кольца, обозначаемый  $0$ , такой, что  $0+a = a+0 = a$  для любого элемента кольца  $a$  (нейтральный элемент относительно сложения, или аддитивный нейтральный элемент).
- Для любого элемента кольца  $a$  существует элемент, обозначаемый  $-a$ , такой, что  $a+(-a) = (-a)+a = 0$  (обратимость относительно сложения).
- Для любых элементов кольца  $a$ ,  $b$  и  $c$  имеет место равенство  $a(bc) = (ab)c$  (ассоциативность умножения).
- Для любых элементов кольца  $a$ ,  $b$  и  $c$  имеют место равенства  $a(b+c) = ab+ac$  и  $(a+b)c = ac+bc$  (дистрибутивность).
- Существует элемент кольца, обозначаемый  $1$ , такой, что  $1a = a1 = a$  для любого элемента кольца (нейтральный элемент относительно умножения, или мультипликативный нейтральный элемент).

При сложении с аддитивным обратным элементом скобки обычно опускаются, т. е.  $(-a)+b$  записывается в виде  $-a+b$ , а  $a+(-b)$  – в виде  $a-b$ .

Отметим, что операция умножения в кольце не обязана быть коммутативной. Но если это так, то кольцо называется *коммутативным*. Все приведенные выше примеры колец коммутативны. Если у элемента кольца  $a$  имеется мультипликативный обратный  $a'$  такой, что  $aa' = 1$ , то  $a$  называется *обратимым*. При работе с конечными кольцами рекомендуется попробовать попарно перемножить все элементы, определить, какие элементы обратимы, и составить их таблицу, к которой можно будет быстро обратиться.

Простой способ применить арифметику колец к шифрованию – объединить пульсирующий шифр из раздела 11.8 и линейное суммирование с запаздыванием из раздела 13.14.1. Выберем кольцо **R13** с элементами вида  $a+b\sqrt{13}$ , где  $a$  и  $b$  – шестнадцатеричные цифры, т. е. целые по модулю 16. Две шестнадцатеричные цифры  $a$  и  $b$  образуют байт, представляющий некоторый символ. Например, буква  $X$  в кодировке ASCII представлена шестнадцатеричным числом 58, которому соответствует элемент кольца  $5+8\sqrt{13}$ .

Сумма двух элементов кольца **R13**,  $a+b\sqrt{13}$  и  $c+d\sqrt{13}$ , равна  $(a+c)+(b+d)\sqrt{13}$ , а произведение  $(ac+13bd)+(ad+bc)\sqrt{13}$ , причем все операции сложения производятся по модулю 16. Например, если  $x = 2+3\sqrt{13}$  и  $y = 4+5\sqrt{13}$ , то  $x+y = 6+8\sqrt{13}$  и  $xy = 11+6\sqrt{13}$ .

Для комбинации пульсирующего шифра и линейного суммирования с запаздыванием, которую можно было бы назвать *Lag Ripple*, мы заменяем  $x_n$  выражением  $ax_n + bx_{n-i} + cx_{n-j}$ , где коэффициенты  $a$ ,  $b$  и  $c$  – элементы кольца, в данном случае **R13**, а запаздывания  $i$  и  $j$  – малые целые числа, например 2 и 5. Открытый текст можно было бы разбить на блоки, скажем по 16 байт, но если сообщение короткое, то шифр можно применить сразу ко всему сообщению. Предположим именно этот случай. Тогда шифрование можно записать в виде:

$$x_n = ax_n + bx_{n-2} + cx_{n-5} \text{ для } n = 1, 2, 3, \dots, L.$$

Здесь  $a$  – обратимый элемент **R13**,  $b$  и  $c$  – произвольные элементы **R13**, а  $L$  – длина сообщения. Арифметические операции выполняются в кольце. Можете считать это вариантом комбинирующей функции **madd** из раздела 11.8. Для шифрования первых нескольких байтов применяется обычное заворачивание.

В случае одного прохода с известными фиксированными запаздываниями шифр *Lag Ripple* получает оценку 2, поскольку всего существует  $256^3$  возможных комбинаций коэффициентов. Если шифр сопровождается простой подстановкой до и после, то оценка повышается до 5. При трех проходах с разными коэффициентами и запаздываниями оценка достигает 6.

В шифре *Triple Ripple* используется 3 прохода с секретной простой подстановкой, перемешанной ключом, перед каждой фазой пульсации и после последней фазы. На каждом проходе секретные коэффициенты и запаздывания изменяются. Дополнительно на каждом проходе можно начинать фазу пульсации в разных позициях сообщения, с заворачиванием. Шифр *Triple Ripple* получает оценку 10.

## 15.12 Матрицы над кольцом

В разделах 15.1 и 15.2 мы рассматривали шифр Хилла, в котором каждый блок сообщения рассматривался как вектор целых чисел, и этот вектор умножался на целочисленную матрицу по модулю 26



или 256. В числах 26 и 256 нет ничего особенного. Символы сообщения можно представить элементами любого кольца. Если символов больше, чем элементов в кольце, можно использовать пары или тройки элементов, так же как пары целых чисел от 1 до 5 использовались в квадрате Полибия (раздел 9.1) для представления 25-буквенного алфавита.

Предположим, что мы использовали матрицу над кольцом **R13** с элементами вида  $a+b\sqrt{13}$ . Если рассматривать блок открытого текста как вектор из 32 шестнадцатеричных цифр, а не как 16 байт и выписать выражение для каждой цифры произведения матриц, то мы увидим, что каждая шестнадцатеричная цифра шифртекста является линейной комбинацией цифр открытого текста. Поэтому использование матрицы  $16 \times 16$  над кольцом **R13** эквивалентно использованию матрицы  $32 \times 32$  над кольцом шестнадцатеричных цифр, т. е. целых по модулю 16. Следовательно, такой шифр по-прежнему уязвим к атаке с известным открытым текстом. Для атаки потребовалось бы как минимум 256 байт открытого текста. Вероятно,  $16 \times 17 = 272$  байт было бы достаточно.

Отправитель легко может отразить такую атаку, применив простую подстановку с ключом до и после умножения матриц. Можно было бы также построить собственное кольцо, неизвестное никому, кроме полномочных корреспондентов. Если кольцо удастся сохранить в секрете, то никто не сможет организовать атаку против вашего матричного шифра.

## 15.13 Построение кольца

Кольцо, содержащее  $N$  элементов, называется кольцом *порядка*  $N$ . Оно представляется двумя таблицами  $N \times N$  – своими таблицами сложения и умножения. Построение кольца производится в два этапа. Для демонстрации построим кольцо из 8 элементов. Начнем с таблицы сложения. Из аксиом кольца мы знаем о существовании двух элементов – аддитивного нейтрального элемента 0 и мультипликативного нейтрального элемента 1. Суммы  $0+a$  и  $a+0$  известны для всех  $a$ . Это дает верхнюю строку и левый столбец таблицы сложения.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	-	-	-	-	-	-	-
2	2	-	-	-	-	-	-	-
3	3	-	-	-	-	-	-	-
4	4	-	-	-	-	-	-	-
5	5	-	-	-	-	-	-	-
6	6	-	-	-	-	-	-	-
7	7	-	-	-	-	-	-	-

Перейдем ко второй строке. Наша стратегия заключается в том, чтобы взять первую сумму, которой еще не присвоено значение, присвоить его, а затем сделать все возможные выводы относительно других сумм, пользуясь аксиомой ассоциативности. Пусть мы хотим, чтобы  $1+1 = 2$ ,  $2+1 = 3$ ,  $3+1 = 4$  и  $4+1 = 0$ . Применяя аксиому ассоциативности, мы можем заполнить левую верхнюю часть таблицы. Например, можно найти  $2+2$ , потому что  $2+2 = (1+1)+2 = 1+(1+2) = 1+3 = 4$ .

<b>+</b>	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	0	-	-	-
2	2	3	4	0	1	-	-	-
3	3	4	0	1	2	-	-	-
4	4	0	1	2	3	-	-	-
5	5	-	-	-	-	-	-	-
6	6	-	-	-	-	-	-	-
7	7	-	-	-	-	-	-	-

Поскольку  $4+1 = 0$ , отсюда следует, что аддитивным обратным элементом к 1 является 4, а обратным к 4 – 1. Аналогично 2 является обратным к 3, а 3 – обратным к 2. А какое значение следует присвоить  $5+1$ ? Это не может быть 0, потому что 4 и 5 не могут быть одновременно обратными 1. И 1 не может быть, потому что тогда 5 было бы равно 0. Это никак не может быть 2, потому что  $5+1 = 1+1$  означало бы, что  $5 = 1$ . Аналогично  $5+1$  не может быть равно ни 3, ни 4. И не может быть равно 5, потому что  $5+5 = 5$  означает, что  $5 = 0$ . Остается две возможности:  $5+1 = 6$  или  $5+1 = 7$ . Обе имеют одинаковое право на существование, поэтому положим  $5+1 = 6$ . Отсюда вытекает, что  $6+1 = 7$  и  $7+1 = 5$ . Это значит, что  $5+1+1+1 = 5$ . Следовательно,  $1+1+1 = 0$ . Но мы уже знаем, что  $1+1+1 = 3$ , и получается, что  $3 = 0$ . Невозможно. Мы зашли в тупик. Полагать  $4+1 = 0$  нельзя.

В чем же ошибка? Цикл  $1+1+1+1+1 = 0$  состоит из 5 членов. Длина любого такого цикла в кольце порядка  $N$  должна быть делителем  $N$ . Поскольку 5 не делит 8, довести до конца построение таблицы сложения оказалось невозможно. У нас есть три возможности: сделать длину цикла равной 2, 4 или 8. Если выбрать длину цикла 8, то получим кольцо целых чисел по модулю 8. При длине цикла 2 сложение будет совпадать с ИСКЛЮЧАЮЩИМ ИЛИ. Поскольку наша цель – получить новое кольцо, остается только длина цикла 4. Таблица сложения должна иметь вид:

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	0	7	4	5	6
2	2	3	0	1	6	7	4	5
3	3	0	1	2	5	6	7	4
4	4	7	6	5	2	1	0	3
5	5	4	7	6	1	0	3	2
6	6	5	4	7	0	3	2	1
7	7	6	5	4	3	2	1	0

Теперь таблицу умножения можно составить, пользуясь аксиомой дистрибутивности. Например,  $2 \times 2 = 2 \times (1+1) = 2+2 = 0$ .

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	0	2	2	0	2	0
3	0	3	2	1	6	5	4	7
4	0	4	2	6	4	0	6	2
5	0	5	0	5	0	5	0	5
6	0	6	2	4	6	0	4	2
7	0	7	0	7	2	5	2	5

Назовем это кольцо **R8**. Оно коммутативно, поскольку  $ab = ba$  для всех элементов  $a$  и  $b$ . Заметим, что 1 и 3 – единственные элементы **R8**, имеющие мультипликативный обратный, и что каждый из них обратен самому себе.

Два кольца заслуживают особого внимания – гауссовы целые числа и кватернионы.

### 15.13.1 Гауссовы целые числа

Гауссовыми целыми числами называются числа вида  $a+bi$ , где  $a$  и  $b$  – целые числа, а  $i$  – мнимая единица,  $\sqrt{-1}$ . Иначе говоря, гауссовы целые числа – это комплексные числа, у которых вещественная и мнимая части – целые. Для целей криптографии  $a$  и  $b$  должны быть целыми по модулю 16. Таким образом, гауссово число  $a+bi$  можно использовать для представления шестнадцатеричного числа  $ab$ . Например, буква X, имеющая шестнадцатеричный ASCII-код 58, представляется гауссовым целым  $5+8i$ .

Гауссовы целые числа складываются и умножаются по следующим правилам:

$$\begin{aligned}(a+bi) + (c+di) &= (a+c) + (b+d)i, \\ (a+bi) \times (c+di) &= (ac-bd) + (ad+bc)i,\end{aligned}$$

где для применений в криптографии сложение и умножение производятся по модулю 16.

### 15.13.2 Кватернионы

Кватернионы изобрел в 1843 году ирландский математик Уильям Роуэн Гамильтон из Тринити-колледжа в Дублине, Королевский астроном Ирландии. Они понадобились ему для описания движений вращающегося тела. Кватернионы – это числа вида  $a+bi+cj+dk$ , где  $a, b, c$  и  $d$  – обыкновенные числа,  $a, i, j$  и  $k$  – абстрактные единицы. Определяющими для кватернионов соотношениями являются  $i^2 = j^2 = k^2 = ijk = -1$ . Из них можно вывести правила умножения:

$$\begin{aligned}ij &= k & ji &= -k \\ jk &= i & kj &= -i \\ ki &= j & ik &= -j\end{aligned}$$

Умножение кватернионов не коммутативно. Кватернионы часто выступают в роли канонического примера некоммутативного кольца.

Кватернионы широко используются в физике, например для представления точек на сферической поверхности и вращений твердого тела. Их можно адаптировать и для криптографических целей, если считать, что  $a, b, c, d$  – целые числа по модулю 16 или 256. Тогда каждый кватернион будет представлять 2 или 4 символа сообщения.

Другой способ использования кватернионов – считать, что  $a, b, c, d$  – целые по модулю  $2^{32}$ . Мы можем использовать секретное хорошо перемешанное множество 5-, 6- или 8-битовых кодов символов, так что каждый коэффициент будет представлять 6, 5 или 4 символа соответственно. Таким образом, весь кватернион будет представлять 24, 20 или 16 символов сообщения. Зашифровать кватернион сообщения  $M$  можно, умножив его слева или справа на секретный множитель-кватернион. Поскольку умножение кватернионов не коммутативно, гораздо более стойкий шифр получится, если умножать и слева, и справа – АМВ. Как и при умножении обычных чисел, младший байт каждой компоненты оказывается самым слабым, поэтому рекомендуется после первого умножения циклически сдвинуть весь 16-байтовый блок влево на 16 позиций. Тот же самый набор кодов символов можно использовать для преобразования произведения обратно в стандартную кодировку ASCII, но лучше взять разные наборы кодов, предпочтительно с кодами разного размера.

Этот метод, который мы назовем *Qmult*, получает оценку 10. Для дешифрирования сообщения Рива должна умножить слева на кватернион, обратный  $A'$ , а справа на кватернион, обратный  $B'$ . Кватернион, обратный  $a+bi+cj+dk$ , имеет вид  $(a-bi-cj-dk)/(a^2+b^2+c^2+d^2)$ . Поскольку все операции производятся по модулю  $2^{32}$ ,  $a^2+b^2+c^2+d^2$  будет иметь мультипликативный обратный элемент, если является нечетным, т. е. когда нечетны один или три коэффициента.

## 15.14 Нахождение обратимых матриц

Чтобы можно было использовать матрицу в шифре типа Хилла, эта матрица должна быть обратимой. Но отыскать обратимые матрицы зачастую трудно. Если количество обратимых элементов в кольце равно  $i$ , а всего элементов  $r$ , то вероятность, что случайная матрица  $n \times n$  над этим кольцом будет обратимой, равна  $(i/r)^n$ . Для кольца **R8**  $i/r = 2/8 = 1/4$ . (Тут налицо резкий контраст с матрицами над кольцом рациональных или вещественных чисел, где у каждого элемента, кроме 0, имеется мультипликативный обратный, а потому почти любая матрица обратима.) Если матрицы малы, то обычно найти обратимую матрицу удастся, случайно выбирая элементы и пробуя все возможные значения последнего или, в худшем случае, двух последних элементов. Благодаря использованию одного или двух последних элементов мы можем свести матрицу к двум нижним строкам и избежать полного сведения в каждой попытке.

Я решил не рассматривать в этой книге определители, потому что не знаю ни одного их применения в криптографии. Однако для читателей, знакомых с определителями, скажу, что матрица обратима, если значение ее определителя является обратимым элементом кольца. В частности, матрица над кольцом целых чисел обратима, только если ее определитель равен  $+1$  или  $-1$ .

Для больших матриц нахождение обратимой матрицы может оказаться вычислительно неразрешимой задачей. Вместо этого обратимую матрицу нужно *построить*. Начинаем с построения множества матриц желаемого размера, имеющих одну из двух специальных форм: треугольную или блочно-диагональную. Ниже приведены примеры четырех типов треугольных матриц  $4 \times 4$ .

Верхнетреугольная

$$\begin{pmatrix} a & b & c & d \\ 0 & e & f & g \\ 0 & 0 & h & i \\ 0 & 0 & 0 & j \end{pmatrix}$$

Нижнетреугольная

$$\begin{pmatrix} a & 0 & 0 & 0 \\ b & c & 0 & 0 \\ d & e & f & 0 \\ g & h & i & j \end{pmatrix}$$

Верхне-антитреугольная

$$\begin{pmatrix} a & b & c & d \\ e & f & g & 0 \\ h & i & 0 & 0 \\ j & 0 & 0 & 0 \end{pmatrix}$$

Нижне-антитреугольная

$$\begin{pmatrix} 0 & 0 & 0 & a \\ 0 & 0 & b & c \\ 0 & d & e & f \\ g & h & i & j \end{pmatrix}$$

В верхнетреугольной матрице ненулевые элементы находятся только на главной диагонали и выше нее, а все остальные равны нулю. В нижнетреугольной матрице ненулевые элементы находятся только на главной диагонали и ниже нее, а все остальные равны нулю. В верхнеантитреугольной матрице ненулевые элементы находятся только на антидиагонали и выше нее, а все остальные равны нулю. В нижнеантитреугольной матрице ненулевые элементы находятся только на антидиагонали и ниже нее, а все остальные равны нулю.

Треугольная матрица обратима, если обратимы все диагональные элементы. Антитреугольная матрица обратима, если обратимы все антидиагональные элементы. Найти обратные матрицы легко с помощью техники, описанной в разделе 15.1.1. Для верхнетреугольной и нижнеантитреугольной матрицы процесс приведения следует выполнять справа налево.

Обратимую матрицу общего вида можно построить из треугольных путем перемножения. Но делать это нужно аккуратно. Произведение двух верхнетреугольных матриц также является верхнетреугольной матрицей, а произведение двух нижнетреугольных – нижнетреугольной. Антитреугольные матрицы таким свойством не обладают. Правильный подход – включить в произведение матрицы всех четырех типов. Если треугольные матрицы обозначить  $A, B, C, D$ , а обратные к ним  $A', B', C', D'$ , то обратной к произведению  $ABCD$  будет матрица  $D'C'B'A'$ .

Помимо треугольных, к процессу построения обратимых матриц можно привлечь блочно-диагональные матрицы. Ниже приведен пример такой матрицы размера  $5 \times 5$ . Ее можно назвать матрицей типа 2,3, потому что она состоит из матриц  $2 \times 2$  и  $3 \times 3$ , расположенных вдоль главной диагонали:

$$\begin{pmatrix} a & b & 0 & 0 & 0 \\ c & d & 0 & 0 & 0 \\ 0 & 0 & e & f & g \\ 0 & 0 & h & i & j \\ 0 & 0 & k & l & m \end{pmatrix}.$$

Произведением двух блочно-диагональных матриц является матрица такого же вида.

Преимущество блочно-диагональных матриц в том, что обрабатывать можно каждый блок в отдельности. Если затем расположить найденные таким образом обратные матрицы вдоль диагонали, то получится матрица, обратная к исходной. Нахождение обратной матрицы  $16 \times 16$  может оказаться практически неразрешимой задачей, но найти четыре обратимые матрицы  $4 \times 4$  не так уж сложно. Продолжить построенную обратимую блочно-диагональную матрицу

цу до полной можно, перемножив несколько блочно-диагональных матриц разного типа или добавив в произведение обратимые треугольные матрицы.

Давайте, за работу. Постройте собственную обратимую блочно-диагональную матрицу, состоящую из самых крупных блоков, которые сможете найти, и добавьте четыре треугольные матрицы, по одной каждого вида. Окончательная обратимая матрица будет произведением всех пяти матриц. \*\*



# 16

## Трехпроходный протокол

---

### **Краткое содержание главы:**

- трехпроходный протокол на основе возведения в степень;
- трехпроходный протокол на основе умножения матриц;
- трехпроходный протокол на основе двустороннего умножения матриц.

В разделах 2.2 и 2.3 было сказано, что в современной криптографии можно выделить три ветви: с секретным ключом, с открытым ключом и с персональным ключом. До сих пор описывались только методы криптографии с секретным ключом. Криптография с открытым ключом описывается во многих книгах, и здесь мы ее затрагивать не будем. А эта глава посвящена менее известной дисциплине – криптографии с персональным ключом. Иногда ее называют *бесключевой* криптографией, поскольку стороны не нуждаются ни в передаче, ни в разделении ключей.

Основная идея криптографии с персональным ключом заключается в том, что у обоих корреспондентов, Сандры и Ривы, имеется свой персональный ключ. Он никогда не передается и не разделяется с кем-то еще, даже друг с другом, поэтому у Эмили нет никакой возможности узнать персональные ключи путем подключения к проводным линиям, перехвата широковестьельных сообщений или какой-либо иной формы подслушивания. У криптографии

с персональным ключом имеется важнейшее достоинство – ничего не нужно подготавливать заранее. Не нужен ни секретный ключ, ни защищенные каналы связи для обмена ключами. Сообщения можно передавать по открытым каналам. Отсутствуют серверы ключей и прочая инфраструктура.

Криптография с персональным ключом основана на *трехпроходном протоколе*, который изобрел Ади Шамир из израильского института Вейцмана в 1975 году. Для его иллюстрации я сочинил небольшую историю:

*Жил-был король, влюбившийся в королеву соседней страны. Чтобы завоевать ее сердце, король решил отправить ей бесценную жемчужину. Была у короля шкатулка, которую невозможно взломать, и крепкий навесной замок. Но как передать королеве ключ? Если у гонца будет и шкатулка, и ключ, то он сможет открыть шкатулку и похитить жемчужину. Король мог бы отправить ключ со вторым гонцом, но опасался, что гонцы сговорятся о встрече на дороге и украдут жемчужину вместе. Королева предложила остроумное решение.*

*Пусть король запрет шкатулку на свой замок и отправит ее королеве. Она навесит на нее свой замок и отправит обратно королю. Король отперет свой замок своим ключом и отправит королеве шкатулку, закрытую только на ее замок. Королева сможет отпереть шкатулку своим ключом и забрать жемчужину.*

Здесь два замка – метафоры двух шифрований, а два ключа – метафоры соответствующих дешифрирований. Сообщение шифруется функцией отправителя, отправляется получателю, шифруется функцией получателя и возвращается отправителю, который дешифрирует его своей функцией и отправляет обратно получателю, который дешифрирует его своей функцией. Таким образом, сообщение передается три раза, отсюда и название – *трехпроходный протокол*.

\* Разберем эту процедуру подробнее. Обозначим сообщение  $M$ , функции шифрования и дешифрирования Сандры –  $S$  и  $S'$ , а функции шифрования и дешифрирования Ривы –  $R$  и  $R'$ . На первом проходе Сандра шифрует сообщение  $M$  своей функцией шифрования  $S$  и отправляет  $SM$  Риве. На втором проходе Рива шифрует сообщение  $SM$  своей функцией шифрования  $R$  и отправляет дважды зашифрованное сообщение  $RSM$  обратно Сандре. На третьем проходе Сандра применяет свою функцию дешифрирования  $S'$  к сообщению  $RSM$  и получает  $S'RSM$ . Предполагается, что эта операция снимет шифрование  $S$ . Но так будет, только если либо функции  $R$  и  $S$ , либо  $S'$  и  $R$  коммутируют, т. е.  $S'RSM = RS'SM = RM$ . Это позволит Риве снять свое шифрование и прочесть сообщение.

Итак, чтобы трехпроходный протокол работал, нам необходимо найти коммутативную функцию шифрования или две функции шифрования, коммутирующие друг с другом. Сходу я могу предло-

жить три коммутативные функции шифрования: сложение, умножение и ИСКЛЮЧАЮЩЕЕ ИЛИ. Легко представить себе схему, в которой длина ключа совпадает с длиной сообщения, а само шифрование заключается в побайтовом сложении ключа с сообщением, или побайтовом перемножении байтов ключа и сообщения, или применении ИСКЛЮЧАЮЩЕГО ИЛИ к сообщению и ключу. Все это простые варианты одноразового блокнота.

Ни одна из этих схем не безопасна. Если Эмили удастся получить все три зашифрованных сообщения, она сможет легко снять шифрование. Если функцией является сложение, то три сообщения имеют вид  $M+S$ ,  $M+S+R$  и  $M+R$ . Сложив первое и третье сообщения и вычтя из суммы второе, она получит  $(M+S)+(M+R)-(M+S+R) = M$ , т. е. в точности сообщение  $M$ . Тот же метод работает, когда функцией шифрования является умножение. Тогда три сообщения – это  $(M \times S)$ ,  $(M \times R)$  и  $(M \times S \times R)$ . Результатом операции  $(M \times S) \times (M \times R) \div (M \times S \times R)$  снова является  $M$ . Если для шифрования применяется ИСКЛЮЧАЮЩЕЕ ИЛИ, то найти  $M$  еще проще, т. к. ИСКЛЮЧАЮЩЕЕ ИЛИ – самообратная функция. Если просто применить ко всем трем зашифрованным сообщениям ИСКЛЮЧАЮЩЕЕ ИЛИ, то получится исходное сообщение:  $(M \oplus S) \oplus (M \oplus R) \oplus (M \oplus S \oplus R) = M$ .

Коммутирующими функциями шифрования являются подстановка и перестановка. Они также небезопасны. Поскольку Эмили будет видеть сообщение до и после перестановки, ей не составит никакого труда вычислить эту перестановку.

Таким образом, нам нужна пара коммутирующих функций шифрования  $S$  и  $R$  такая, чтобы Эмили не могла определить  $M$ , даже зная  $SM$ ,  $RSM$  и  $RM$ .

## 16.1 Метод Шамира

Для решения этой проблемы Шамир применил операцию возведения в степень. Пусть  $p$  – большое простое число длиной, скажем, от 300 до 600 десятичных цифр. Сандра выбирает показатель степени для шифрования  $s$ . Соответствующий показатель степени для дешифрирования  $s'$  должен быть таким, чтобы  $ss' \equiv 1 \pmod{p-1}$ . Из малой теоремы Ферма следует, что если  $0 < a < p$ , то  $a^{p-1} \equiv 1 \pmod{p}$ . В разделе 14.4.2 описано, как выбирать простое  $p$ , а в разделе 15.4 – как найти  $s'$ . Аналогично Рива выбирает свои показатели степени для дешифрирования,  $r$  и  $r'$ . Обе функции шифрования коммутируют, потому что  $(M^s)^r = M^{sr} = M^{rs} = (M^r)^s$ .

Сандра вычисляет  $(M^s \bmod p)$  и отправляет результат Риве. Рива вычисляет  $(M^{sr} \bmod p)$  и отправляет Сандре. Сандра вычисляет  $(M^{srs'} \bmod p) = (M^r \bmod p)$  и отправляет Риве, которая наконец-то вычисляет  $(M^{r'r} \bmod p) = M$  и получает исходное сообщение.

Этот метод считается безопасным, потому что для нахождения  $s$  или  $r$  необходимо решить задачу дискретного логарифмирования,

а в разделе 14.4 мы говорили, что эта задача является вычислительно сложной. Неизвестно ни одного практически осуществимого алгоритма ее решения.

Этот метод работает очень медленно. Все операции возведения в степень и приведения по модулю больших чисел требуют большого объема вычислений. В следующем разделе описана одна попытка решения данной проблемы.

## 16.2 Метод Мэсси–Омуры

Метод Мэсси–Омуры предложили Джеймс Мэсси из Высшей технической школы Цюриха и Джим К. Омура из Калифорнийского университета в Лос-Анджелесе в 1982 году. (В патенте имя написано как Джимми Омура. Мы с ним учились на одном курсе в МИТ, хотя я его не помню.) Система Мэсси–Омура по существу совпадает с системой Шамира, только модуль имеет вид  $2^k$ . Вычеты по модулю  $2^k$  можно вычислить, просто взяв младшие  $k$  бит числа. Это гораздо быстрее, чем вычисление вычета по модулю  $p$ , для чего нужно выполнять длинное деление 300- или 600-значных чисел.

Вопрос о том, какой метод быстрее, в течение нескольких лет живо обсуждался в публикациях Ассоциации по вычислительной технике (АСМ) и Института инженеров по электротехнике и электронике (IEEE).

## 16.3 Дискретный логарифм

Безопасность алгоритма распределения ключей Диффи–Хеллмана, трехпроходного протокола Шамира и метода Мэсси–Омуры зависит от трудности решения задачи дискретного логарифмирования. Есть три популярных алгоритма ее решения: полный перебор, эффективный до  $10^{12}$ , алгоритм больших и малых шагов Дэниэла Шенкса<sup>1</sup>, эффективный до  $10^{18}$ , и ро-алгоритм Джона Полларда, эффективный до  $10^{22}$ . Однако нам нужен алгоритм, способный работать с числами порядка  $10^{300}$ . Чтобы прочувствовать, насколько трудна эта задача, рассмотрим комбинированный метод ее решения. На домашнем ПК сделать это не получится. Нужен мейнфрейм с большим объемом памяти или сеть из многих совместно работающих ПК. Впрочем, можете пропустить этот раздел и просто принять на веру, что задача дискретного логарифмирования очень трудна.

---

<sup>1</sup> Этот алгоритм был впервые предложен в 1962 году советским математиком Александром Гельфондом и заново открыт Дэниэлом Шенксом в 1972 году. – *Прим. перев.*

### 16.3.1 Логарифмы

Начнем с обсуждения того, как люди вычисляли обыкновенные логарифмы до появления компьютеров. Один из методов заключался в том, чтобы взять число, например  $b = 1.000001$ , и упорно вычислять его последовательные степени. Так мы нашли бы, что степень  $b^{693148}$  – ближайшая к 2, а  $b^{2302586}$  – ближайшая к 10. А это значит, что значение  $\log_{10}(2)$  очень близко к  $693148/2302586 \approx 0.3010302$ . Более точное значение 0.3010300, так что этот метод дает очень хорошую аппроксимацию.

То же самое можно сделать в кольце, например целых чисел по простому модулю  $p$ . Предположим, что Сандра отправляет сообщение  $6 \bmod 13$ , а Рива возвращает в ответ сообщение  $7 \bmod 13$ . Эмили хочет узнать, какой показатель степени Рива использовала при шифровании. Но вместо степеней 1.000001 мы использовали бы первообразный корень по модулю 13, например 2. При таком малом модуле Эмили легко может перебрать все степени 2 по модулю 13.

1	2	3	4	5	6	7	8	9	10	11	12	N
2	4	8	3	6	12	11	9	5	10	7	1	$2^N \pmod{13}$

Теперь Эмили знает, что Сандра отправила  $2^5$ , а Рива в ответ отправила  $2^{11}$ . Итак,  $(2^5)^r \equiv 2^{5r} \equiv 2^{11} \pmod{13}$ . Это значит, что  $5r \equiv 11 \pmod{12}$ . Это сравнение можно решить в уме. Действительно,  $11+12 = 23$ ,  $23+12 = 35$ . Но 35 кратно 5, а именно равно  $5 \times 7$ , а это значит, что  $r$  должно быть равно 7. Это можно проверить с помощью ручного калькулятора:  $6^7 = 279936 \equiv 7 \pmod{13}$ . Сандра отправила 6, Рива в ответ отправила 7, так что все сходится.

### 16.3.2 Степени простых чисел

Полный перебор – это, конечно, возможный вариант для Эмили, но при больших  $p$  он не годится. Попробуем идею, лежащую в основе ро-алгоритма Полларда. Первый шаг – сгенерировать несколько последовательностей степеней по модулю  $p$  и изучить повторения. Эмили может проверять одновременно несколько первообразных корней, по одному на каждом процессорном ядре. Будем считать, что ядер два. Если  $b$  – первообразный корень по модулю  $p$ , то Эмили может вычислять на одном ядре последовательность  $b^2, b^3, b^4, b^5, \dots \pmod{p}$ , а на другом  $b^2, b^4, b^8, b^{16}, \dots \pmod{p}$ . Это дает Эмили две отдельные последовательности степеней для каждого используемого ей первообразного корня.

Помимо вычисления первообразных корней, Эмили может проверять и непосредственно. Сандра отправляет SM, а Рива отправляет в ответ RSM. Эмили может построить последовательности  $(SM)^2, (SM)^3, (SM)^4, (SM)^5, \dots$  и  $(SM)^2, (SM)^4, (SM)^8, (SM)^{16}, \dots$  и аналогично для RSM. Это дает Эмили еще четыре последовательности степеней.

Помимо этих регулярных последовательностей степеней, она может генерировать и нерегулярные. Обычно они называются *случайными блужданиями*. Один из способов – возвести последнюю сгенерированную степень в квадрат, а затем умножить на одну из ранее вычисленных степеней. Предыдущую степень можно выбирать случайно или брать средний элемент списка. Например, предположим, что уже вычислены степени  $x$ ,  $x^2$ ,  $x^4$ ,  $x^8$  и  $x^{16}$ . Для вычисления следующей степени она могла бы возвести  $x^{16}$  в квадрат, получив  $x^{32}$ , а затем умножить, скажем, на  $x^2$  и получить  $x^{34}$ . Для получения следующей степени нужно было бы возвести в квадрат  $x^{34}$  и результат  $x^{68}$  умножить на другой элемент списка, скажем  $x^8$  – получится  $x^{76}$ . И так далее.

Еще одна форма случайного блуждания получается при использовании двух или трех базовых простых чисел, каждое из которых должно быть первообразным корнем. Начнем с произведения этих чисел. Для генерирования следующего произведения Эмили случайно выбирает одно из базовых простых чисел и умножает на него. Чем больше последовательностей Эмили сгенерирует, тем скорее она начнет получать результаты.

### 16.3.3 Коллизия

Ну хорошо, нагенерировала Эмили кучу последовательностей. А дальше что? Она ищет число, встретившееся в двух списках. Это называется *коллизией*, или *столкновением*. Допустим, оказалось, что  $3^{172964} \equiv 103^{4298755} \pmod{p}$ . Это позволит выразить 103 в виде степени 3 по модулю  $p$ , решив сравнение  $172964r \equiv 4298755 \pmod{p-1}$ . Метод решения таких сравнений описан в разделе 15.4. Накопив достаточно коллизий, Эмили может выстроить цепочку, например:  $RSM \equiv 19^a$ ,  $19 \equiv 773^b$ ,  $773 \equiv 131^c$ , ...,  $103^y \equiv (SM)^z$ . Умножив все компоненты по модулю  $p-1$ , она получит, что  $RSM \equiv (SM)^r \pmod{p}$ . Это показатель степени  $r$  в функции шифрования Ривы. Эмили вскрыла шифр!

На самом деле все не так просто. Если  $p$  – 300-значное простое число, то Эмили придется вычислить порядка  $10^{150}$  степеней, прежде чем начнут появляться коллизии. Если бы она располагала миллионном процессоров, непрерывно вычисляющих эти степени со скоростью 1 000 000 в секунду, то смогла бы сгенерировать  $3 \times 10^{19}$  степеней в год. А значит, до получения хоть каких-нибудь результатов прошло бы примерно  $10^{150}$  лет и гораздо больше – до выстраивания цепочки. И ко всему прочему понадобилось бы несколько хранилищ по  $10^{150}$  байт.

### 16.3.4 Факторизация

Вместо того чтобы искать коллизии, Эмили могла бы при каждом генерировании новой степени попробовать разложить на множители (факторизовать) ее вычет по модулю  $p$ . Предположим, что ей удалось

факторизовать вычет  $97^a \pmod p$  и найти, что  $97^a \equiv 11^b 29^c 83^d \pmod p$ . Она может решить это сравнение относительно 97. Пусть мультипликативный обратный элемент для  $a$  по модулю  $p-1$  равен  $a'$ . Возведем это сравнение в степень  $a'$ . Имеем  $97^{aa'} \equiv 97 \equiv (11^b 29^c 83^d)^{a'} \pmod p$ . После перемножения всех показателей степени и приведения их по модулю  $p-1$  получаем  $97 \equiv 11^e 29^f 83^g \pmod p$  для некоторых значений  $e, f$  и  $g$ . (Длина каждого фактического значения могла бы достигать 300 цифр, даже если  $p$  содержит 300 цифр.) Имея выражение для одного из базовых простых чисел, в данном случае 97, Эмили может подставить его во все факторизованные произведения – как те, что уже найдены, так и те, что будут найдены позже.

Эмили не сможет факторизовать вычет каждой степени. Факторизовать 300-значное число очень трудно, в смысле – занимает много времени. Лучшая стратегия – выбрать фиксированное базовое множество  $F(B)$  простых чисел, скажем всех простых чисел, не превышающих  $B = 10^6$  или, быть может,  $B = 10^7$ .  $F(B)$  называется *базой множителей*. Пробуем разложить каждую степень, пользуясь только простыми числами из базы множителей. Числа, которые удается факторизовать таким образом, называются *B-гладкими*. По мере роста чисел доля B-гладких чисел становится все меньше. Среди 300-значных чисел B-гладких совсем мало. Когда Эмили находит какой-то множитель, еще не факторизованная часть числа сокращается. Если после перебора всех простых чисел в базовом множестве какая-то часть числа осталась нефакторизованной, то дальнейшие попытки следует прекратить. Эффективнее отбросить эту степень и перейти к следующей.

Вот что должна делать Эмили: продолжать генерировать произведения и факторизовать их вычеты по модулю  $p$ . Оставлять только B-гладкие числа и отбрасывать остальные. Проверять коллизии среди B-гладких чисел. При обнаружении коллизии решать сравнение относительно наибольшего простого числа в произведении, чтобы для выражения каждого произведения требовалось все меньше и меньше базовых простых чисел. Для решения этой задачи она может зарезервировать один или несколько процессоров.

Предположим, что  $q^n$  – степень простого числа, и пусть ее вычет по модулю  $p$  равен  $x$ . Пробуем разложить  $x$  на простые множители из базового множества  $B$ . Если  $x$  не B-гладкое, то пробуем факторизовать числа  $x+p, x+2p, x+3p, \dots$ . Факторизовать 301- или 302-значное не намного сложнее, чем 300-значное. Зададим фиксированное число таких попыток, скажем 10 для каждого вычета.

В процессе генерирования степеней Эмили должна уделять особое внимание произведениям  $SM$  и  $RSM$ . Напомним, что цель всего этого – найти такой показатель степени  $r$ , что  $(SM)^r \equiv RSM \pmod p$ . Сделать это она не сможет, пока не разложит  $SM$  и  $RSM$  в произведение степеней базовых простых чисел. Для начала она должна построить много последовательностей степеней  $SM$  и  $RSM$ . Успешно отыскав такое выражение, она ищет простые числа в выражении, которое



еще не было представлено в терминах степеней меньших простых чисел. Далее она обращает внимание на эти числа. И продолжает, пока SM и RSM не будут выражены в виде степеней одного простого числа. Теперь она может найти  $g$ , применив методы из раздела 15.3.2.

### 16.3.5 Оценки

Предположим, что Эмили использует  $10^6$  базовых простых чисел, т. е. не превышающих  $B = 15\,485\,863$ . Чтобы выразить их все в терминах одного простого числа, потребуется  $10^6$  сравнений. Для их хранения нужна матрица показателей степени размера  $10^6 \times 10^6$ . Первоначально эта матрица разрежена, но в процессе поиска решения постепенно заполняется, поэтому методы работы с разреженными матрицами не принесут пользы. Каждый показатель степени – 300-значное число. Значит, потребуется порядка  $10^{15}$  байт, или один *петабайт*. На момент написания книги (март 2022) самый большой в мире суперкомпьютер Summit в Ок-Риджской национальной лаборатории был оснащен 2.76 петабайта адресуемой памяти.

Время работы, очевидно, зависит от того, сколько времени нужно для поиска  $B$ -гладких чисел. Плотность  $B$ -гладких чисел описывается функцией де Брёйна  $\Psi(p, B)$ , которая равна числу  $B$ -гладких чисел, меньших  $p$ . Ее изучал голландский математик Николас Говерт де Брёйн. Значение  $\Psi(x, x^{1/u})$  хорошо аппроксимируется функцией  $x\rho(u)$ , где  $\rho(u)$  – функция Дикмана, предложенная статистиком страхового общества Карлом Дикманом. Функция Дикмана  $\rho(u)$  аппроксимируется функцией  $u^{-u}$ . В нашем случае  $x = 10^{300}$  и  $x^{1/u} = 15\,485\,863$ , так что  $u = 41.725$ . Таким образом, для нахождения каждого  $B$ -гладкого числа потребуется  $41.725^{41.725} \approx 4.08 \times 10^{67}$  попыток.

Всего для нахождения  $10^6$   $B$ -гладких чисел потребуется больше  $10^{73}$  попыток. Факторизация каждого числа может потребовать  $10^6$  операций перебора делителей, т. е. всего  $10^{79}$  таких операций. Поскольку числа 300-значные, количество операций при каждом переборе делителей будет равно какому-то кратному 300. Ну, пусть всего  $10^{82}$  операций. Это гораздо лучше, чем  $10^{150}$  операций в методе коллизий, но все равно за пределами возможностей современных компьютеров.

Итак, мы показали, что 300 цифр более чем достаточно в обозримом будущем, быть может на ближайшие 20–30 лет. Все может измениться по мере разработки квантовых компьютеров, но пока 300 цифр обеспечивают безопасность.

## 16.4 Матричный трехпроходный протокол

В методах Шамира и Мэсси–Омуры, применяемых в трехпроходном алгоритме, используется возведение в степень. Но есть и другой подход на основе матриц. Мы уже встречались с ним при обсуждении шифра Хилла в разделе 15.1. Сообщение разбивается на блоки.

Каждый блок рассматривается как вектор целых чисел по модулю 256. Этот вектор умножается слева или справа на обратимую квадратную матрицу целых чисел по модулю 256. Для трехпроходной версии Сандра будет хранить матрицу  $S$  для шифрования и обратную к ней матрицу  $S'$  для дешифрирования, а Рива – аналогичные матрицы  $R$  и  $R'$ . Это матрицы не над множеством целых чисел по модулю 256, а над кольцом  $\mathbf{R}$ , содержащим 256 элементов, и символы сообщения интерпретируются как элементы этого кольца. Обозначим блок сообщения  $M$ , так что Сандра отправляет Риве  $SM$ , а Рива в ответ отправляет  $RSM$ . Сандра дешифрирует это сообщение с помощью  $S'$  и получает  $S'RSM = RM$ . Теперь Рива может дешифрировать его с помощью  $R'$  и получить  $R'RM = M$ .

Нетривиальная часть – сделать так, чтобы  $S'RSM = RM$ . Умножение матриц не коммутативно, поэтому Сандра и Рива должны выбирать специальные матрицы  $S$  и  $R$ , коммутирующие друг с другом. Уточню:  $S$  и  $R$  не являются коммутативными матрицами. Если выбрать случайную матрицу  $X$ , то почти наверняка  $SX \neq XS$  и  $RX \neq XR$ . Это очень важно, так что я повторю:  $S$  и  $R$  – не коммутативные матрицы. Но они коммутируют между собой.

### 16.4.1 Коммутативное семейство матриц

Сандре и Риве понадобится большой запас матриц, чтобы Эмили не могла просто перебрать их все. Это значит, что нужно большое коммутативное семейство матриц  $\mathbf{F}$ , из которого будут выбираться матрицы для каждого блока сообщения.

**ПРИМЕЧАНИЕ**  $\mathbf{F}$  – это именно коммутативное семейство матриц, а не семейство коммутативных матриц. Важно понимать, что коммутативно семейство, а не сами матрицы. Почти все матрицы в  $\mathbf{F}$  не будут коммутативными. Они коммутируют друг с другом, но не с другими матрицами.

Самый простой способ построить коммутативное семейство – начать с произвольной обратимой матрицы  $F$  и взять ее степени,  $F^0, F^1, F^2, F^3, \dots$ , где  $F^0$  – единичная матрица  $I$ , а  $F^1 = F$ . Будем называть  $F$  *порождающей матрицей* семейства  $\mathbf{F}$ .

Сандре и Риве понадобятся разные матрицы для каждого блока сообщения, в противном случае Эмили смогла бы решить систему линейных уравнений  $R(SM_i) = RSM_i$  при наличии достаточного набора блоков сообщений  $M_i$  с известным открытым текстом.

### 16.4.2 Мультипликативный порядок

Чтобы семейство  $\mathbf{F}$  было большим, нужно найти или построить порождающую матрицу  $F$  высокого мультипликативного порядка. Это значит, что наименьшее целое  $n > 0$  такое, что  $F^n = I$ , должно быть

велико, по меньшей мере  $10^{25}$ , но лучше – больше. Если матрица  $F$  обратима, такое  $n$  обязательно существует, и матрицей  $F'$ , обратной к  $F$ , будет  $F^{n-1}$ . Метод нахождения обратимых матриц описан в разделе 15.8. Определение мультипликативного порядка  $F$  сродни искусству. Очевидно, что не имеет практического смысла просто вычислять последовательные степени  $F$ , пока не окажется, что  $F^n = I$ , уж во всяком случае не тогда, когда  $n > 10^{25}$ . И все же это можно сделать.

Чтобы найти мультипликативный порядок, начнем с матриц  $1 \times 1$ , т. е. элементов кольца. Определим мультипликативный порядок этих элементов. Это легко сделать прямым перебором, поскольку максимально возможное значение  $n$  равно 255. Вероятные значения – 2, 3, 7, 15, 31, 63, 127 и 255. Мультипликативные порядки матриц большего размера часто оказываются кратны этим числам.

Предположим, что мультипликативные порядки элементов кольца оказались равны 2, 7 и 31. При рассмотрении матриц  $2 \times 2$  сначала возведем каждую матрицу  $A$  в степень, являющуюся произведением порядков отдельных элементов, например  $2^4 7^2 3^1 = 24304$ . Затем переберем степени  $B = A^{24304}$ . Предположим, оказалось, что  $B^{52} = I$ . Теперь мы точно знаем, что мультипликативный порядок  $m$  матрицы  $A$  делит  $x = 24304 \times 52 = 2^6 7^2 13 \times 31$  и является кратным  $2^6 13$ . Далее следует посмотреть, не равны ли  $I$  матрицы  $A^{x/7}$  и  $A^{x/31}$ . Если  $A^{x/7} = I$ , то можно попробовать  $A^{x/49}$ . В данном случае наибольшим мультипликативным порядком может оказаться  $2^6 7 \times 13 \times 31$ .

Далее переходим к матрицам  $3 \times 3$ . Если в мультипликативных порядках матриц  $2 \times 2$  не встречалось никаких множителей, кроме 2, 3, 7, 13 и 31, то хорошим начальным показателем степени может стать  $x = 2^8 7^2 13^2 31^2$ . Переберем последовательные степени  $B = A^x$  и повторим процесс снижения показателя. По мере увеличения размера матриц мультипликативный порядок может оказаться слишком большим для прямого перебора. В таком случае придется угадывать, какие новые простые множители появятся.

Наблюдайте закономерности в последовательности мультипликативных порядков. Тут вам понадобятся таланты детектива. Например, предположим, что встречаются  $2^5 - 1$ ,  $2^6 - 1$ ,  $2^9 - 1$  и  $2^{12} - 1$ . Напрямую вы их не увидите, потому что не все они простые:  $2^6 - 1 = 63 = 3^2 7$ ,  $2^9 - 1 = 511 = 7 \times 73$ ,  $2^{12} - 1 = 4095 = 3^2 5 \times 7 \times 13$ . Но обнаружение 13 среди простых множителей позволяет высказать догадку, что «настоящим» множителем может быть  $2^{12} - 1$ , а обнаружение 73 – сильный довод в пользу того, что множителем может оказаться  $2^9 - 1$ . Если встретились  $2^5 - 1$ ,  $2^6 - 1$ ,  $2^9 - 1$  и  $2^{12} - 1$ , то можно ожидать, что скоро встретится и  $2^{15} - 1$ . Если все они встретились, то, поскольку каждый делится на 7, мультипликативный порядок будет делиться на  $7^4$ .

### 16.4.3 Максимальный порядок

Цель Сандры – сделать семейство  $\mathbf{F}$  максимально большим, чтобы ей и Риве было из чего выбрать матрицы  $S$  и  $R$ . Есть полезный

прием – наблюдать за мультипликативными порядками сомножителей. Например, если мультипликативный порядок  $A$  равен  $19m$ , а мультипликативный порядок  $B$  равен  $23m$ , то мультипликативный порядок  $AB$  мог бы быть равным  $19 \times 23m = 437m$ . Если это не так, то мультипликативный порядок  $437m$  могли бы иметь матрицы  $A'B$  или  $AB'$ .

По возможности Сандра должна стремиться к тому, чтобы у порождающей матрицы  $F$  мультипликативный порядок был большим простым числом, скажем  $m > 10^{35}$ , дабы предотвратить атаку Силвера–Полига–Хеллмана (раздел 14.4). Сандре придется факторизовать  $2^n - 1$  для различных  $n$ , чтобы найти значения, имеющие большие простые множители, а затем найти порождающую матрицу, мультипликативный порядок которой делится на одно из таких  $2^n - 1$ , пробуя матрицы возрастающего размера.

#### 16.4.4 Атаки Эмили

Предположим, что Сандра выбрала  $F$  и  $\mathfrak{f}$  и отправила сообщение Риве. Поскольку Сандра и Рива общаются по открытому каналу, например через интернет, будем предполагать, что Эмили знает  $F$ ,  $\mathfrak{f}$ ,  $SM$ ,  $RSM$  и  $RM$ . Ее цель – найти либо  $R$ , либо  $S$ , так что шансы удваиваются. Сосредоточимся на нахождении  $R$ . Эмили знает две вещи об  $R$ . Во-первых, ей известны значения  $SM$  и  $RSM$ , что дает систему  $n$  линейных уравнений с  $n^2$  неизвестными элементами  $R$ . Во-вторых, она знает, что  $R$  принадлежит семейству  $\mathfrak{f}$ , поэтому должна коммутировать с  $F$ , т. е.  $RF = FR$ . Если кольцо  $\mathbf{R}$  коммутативно, то это дает ей недостающие  $n(n-1)$  линейных уравнений относительно  $n^2$  элементов  $R$ .

Эта идея работает, потому что левая часть матричного уравнения  $RF = FR$  порождает суммы членов вида  $rf$ , где  $r$  – неизвестный элемент  $R$ , а  $f$  – известный элемент  $F$ . Правая же часть порождает суммы членов вида  $fr$ . Поскольку кольцо коммутативно, левые члены  $rf$  можно преобразовать к виду  $fr$  и объединить с правыми членами для образования линейных уравнений.

Итак, мы имеем  $n^2$  линейных уравнений с  $n^2$  неизвестными, и кажется, что решить эту систему и найти  $R$  – детская задача. Однако не все так просто. Вспомните (раздел 15.3.1), что сравнения бывают слабыми и сильными. И то же самое относится к линейным уравнениям над любым конечным кольцом, размер которого не является простым числом. Чем больше простых множителей в размере кольца, тем вероятнее появление слабых уравнений. В нашем случае размер кольца равен  $2^8$ , т. е. простых множителей восемь, поэтому многие линейные уравнения, скорее всего, окажутся слабыми. Если кольцо  $\mathbf{R}$  выбрано хорошо, то типичный размер матриц мог бы быть равен  $30 \times 30$ , а если плохо, то  $128 \times 128$  или даже  $256 \times 256$ . Но даже если кольцо выбрано хорошо и половина уравнений сильная, следует ожидать, что система из  $30 \times 30 = 900$  уравнений будет иметь как

минимум  $2^{450}$  решений. На практике число решений гораздо больше, потому что могут быть уравнения, имеющие 4, 8 или 16 решений.

Для Эмили есть хорошая новость. Она может решать уравнения относительно  $R'$ , а не  $R$ , и любое из найденных  $2^{450}$  или более решений будет обращением  $R$ , что позволит получить сообщение в виде  $R'RM = M$ .

### 16.4.5 Некоммутативное кольцо

Похоже, Сандру и Риву постигла неудача. Эмили победила.

Возможный ответ на эту атаку со стороны Сандры и Ривы – использовать некоммутативное кольцо. Примерами таких колец являются матрицы и кватернионы (раздел 15.7.2). Элементами матрицы вполне могут быть матрицы, а коэффициентами кватернионов – матрицы или кватернионы. Но то и другое плохо. Чтобы получить высокий мультипликативный порядок, матрицы должны быть очень большими.

Лучше пойти по другому пути и сконструировать собственное кольцо  $N$ , применяя методы из раздела 15.7. Выбирать кольцо нужно так, чтобы в нем было много элементов, обладающих следующими свойствами: (1) обратимы, (2) имеют высокий мультипликативный порядок и (3) не коммутативны. Отыскать кольцо, удовлетворяющее всем этим условиям, нелегко. Например, кольцо, все элементы которого имеют максимальный мультипликативный порядок (255 для кольца из 256 элементов), вообще не может иметь некоммутативных элементов. Если бы удалось найти кольцо, в котором половина элементов обратима, половина имеет мультипликативный порядок, равный примерно половине размера кольца, а половина не коммутативна, то нам и этого было бы достаточно. Достичь всех трех целей одновременно невозможно, но можно превзойти одни и подойти близко к другим.

В некоммутативном кольце матричное уравнение  $RF = FR$  уже нельзя линеаризовать, потому что равенство  $rf = fr$  в общем случае не имеет места. Вместо этого мы получаем систему *билинейных* уравнений. Члены билинейного уравнения имеют вид  $axb$ , где  $a$  и  $b$  – элементы кольца, а  $x$  – искомая переменная. Для решения линейных уравнений существует простой систематический подход, например метод исключения Гаусса, но для билинейных это уже не так. Не существует даже общего метода решения такого простого уравнения, как  $ax + xb = c$  с одной переменной  $x$ . Поэтому решить билинейные уравнения над кольцом «невозможно».

### 16.4.6 Решение билинейных уравнений

А теперь я покажу, как все же решить билинейные уравнения. Хитрость в том, чтобы изменить представление элементов кольца  $N$ . Мы уже встречали несколько таких примеров. Элементы кольца **R13**

представляются в виде  $a + b\sqrt{13}$ , гауссовы целые числа – в виде  $a + bi$ , кватернионы – в виде  $a+bi+cj+dk$ . Здесь  $i, j$  и  $k$  – абстрактные единицы, произведения которых определяют поведение кольца, а  $a, b, c$  и  $d$  – коммутативные элементы кольца. Кватернионы могут быть не коммутативны, потому что умножение единиц не коммутативно, т. е.  $ij \neq ji$ ,  $ik \neq ki$  и  $jk \neq kj$ . В случае гауссовых чисел мнимая единица всего одна, поэтому они коммутативны.

Идея в том, чтобы *линеаризовать* билинейные уравнения, отыскав подходящее представление некоммутативного кольца  $N$ . Сделать это просто. Для начала разобьем элементы  $N$  на два множества,  $A$  и  $B$ , где  $A$  содержит элементы, имеющие представления, а  $B$  – остальные элементы. В начальный момент  $A$  пусто, а  $B$  содержит все элементы кольца. Переместим все коммутативные элементы в множество  $A$ . Эти элементы представляют сами себя. Они образуют член «а» в представлении. Любой из оставшихся обратимых элементов выберем в качестве единицы  $i$ . Возьмем все элементы кольца, которые можно представить в виде  $a+bi$ , где  $a$  и  $b$  – коммутативные элементы, и переместим их из множества  $B$  в множество  $A$ . Пока что все элементы  $A$  по-прежнему коммутативны.

Множество  $B$  не может оказаться пустым, потому что  $N$  не коммутативно. Мы уже отмечали, что кольцо с единственной единицей, подобное кольцу гауссовых чисел, обязано быть коммутативным. Поэтому возьмем еще какой-нибудь обратимый элемент из множества  $B$  и назовем его второй единицей,  $j$ . На этот раз мы берем все элементы, которые можно представить в виде  $a+bi+cj$ , и перемещаем их из  $B$  в  $A$ . В множестве  $B$  могут еще оставаться элементы. В таком случае описанные шаги следовало бы повторить, но для простоты предположим, что (1) хватает всего двух единиц; (2) все элементы кольца можно представить в виде  $a+bi+cj$ , где  $i$  и  $j$  – абстрактные единицы; (3)  $a, b$  и  $c$  – коммутативные элементы кольца  $N$ . На практике количество единиц может зависеть от выбора  $i$  и  $j$ , поэтому следует попробовать несколько раз, стараясь добиться минимального числа. Это важно, потому что чем больше единиц, тем больше уравнений получится после линеаризации. А поскольку время, необходимое для решения системы линейных уравнений, пропорционально кубу их числа, это далеко не безразлично. Вернемся к матричному уравнению  $RF = FR$  и представим элементы кольца в виде  $a+bi+cj$ . Незвестные элементы  $R$  будут иметь вид  $x+yi+zj$ , где  $x, y$  и  $z$  – неизвестные коммутативные элементы кольца. Теперь член произведения матриц  $RF$  можно записать в виде

$$(x+yi+zj)(a+bi+cj) = ax + bxi + cxj + ayi + byi^2 + cyij + azj + bzji + czj^2,$$

где  $i^2, j^2, ij$  и  $ji$  представимы в виде линейных комбинаций  $1, i$  и  $j$ , например  $d+ei+fj$ . Конечно, фактические представления зависят от выбора кольца и его элементов  $i$  и  $j$ .



То же самое следует проделать с членами в произведении FR. В итоге вместо 900 уравнений с 900 неизвестными мы получим 2700 с 2700 неизвестными. И количество ложных решений возрастет с  $2^{450}$  до  $2^{1350}$ . Это плохая новость для Эмили. Ложные решения не помогут ей восстановить сообщение.

### 16.4.7 Слабые элементы

В семействе  $\mathbf{F}$  имеются слабые элементы, например диагональные или треугольные матрицы, которые Эмили легко может обратить. Такие элементы не следует использовать в качестве ключей. При выборе матрицы из  $\mathbf{F}$  проверяйте, что выше и ниже главной диагонали есть хотя бы один ненулевой элемент. Чтобы ускорить проверку, сравнивайте с нулем только  $X_{12}$ ,  $X_{13}$  и  $X_{23}$ , а также  $X_{21}$ ,  $X_{31}$  и  $X_{32}$ . Если хотя бы в одной тройке все элементы равны нулю, отбросьте  $X$  и выберите другую матрицу. Процент отброшенных матриц будет пренебрежимо мал.

### 16.4.8 Как сделать побыстрее

Пока что не ясно, есть ли у матриц какие-нибудь преимущества перед возведением в степень. Для выбора матрицы  $S$  или  $R$  из семейства  $\mathbf{F}$  требуется возводить порождающую матрицу  $F$  в большую степень. И чем же это лучше или быстрее возведения в большую степень большого целого числа? Разница в подготовке. В методах Шамира и Мэсси–Омуры Сандра и Рива должны возвести в большую степень число, полученное от другой стороны. Поскольку заранее это число неизвестно, они не могут принять никаких дополнительных мер, ускоряющих возведение в степень.

Но в методе на основе матриц порождающая матрица  $F$  известна заранее. И Сандра, и Рива могут заранее вычислить какие-то степени  $F$  и сохранить их; тогда для вычисления новой степени  $F$  можно будет ограничиться всего одним или двумя умножениями матриц. Для начала можно сгенерировать 16 матриц  $F, F^2, F^4, F^8, \dots, F^{32768}$ , для чего понадобится 15 операций умножения.

Но если бы они на этом и остановились, то Эмили могла бы сделать то же самое. У нее был бы тот же базовый набор матриц, что у Сандры и Ривы, поэтому она смогла бы без труда определить их матрицы шифрования,  $S$  и  $R$ . Чтобы предотвратить это, Сандра и Рива должны рандомизировать свои наборы матриц. Для этого они случайным образом выбирают две из своих матриц и перемножают их. Это произведение заменит одну из двух выбранных матриц в базовом наборе. Сандра и Рива делают это независимо. Ни одна из них не знает, какие степени  $F$  выбрала другая сторона.

Эту операцию замены следует повторить на этапе инициализации многократно, скажем 1000 раз, так чтобы набор матриц у каждой стороны был совершенно случаен. Если вам кажется, что 1000 слишком



много, вспомните, что в методе Шамира используется 300-значное простое число и что каждое возведение в степень требует примерно 1000 умножений и 1000 приведений по модулю. Сандра и Рива должны также хранить обращения своих матриц. При каждом перемножении двух степеней  $F$  нужно перемножать также соответствующие степени  $F'$ , чтобы потом не приходилось обращать степени.

Этот этап подготовки, генерирование базового набора, выполняется только один раз, до отправки первого сообщения. Имея такой набор порождающих матриц, мы можем вычислить матрицу для отправки сообщения, выполнив всего одно умножение для самой матрицы и еще одно для обратной к ней. Мы случайно выбираем две разные матрицы  $F^a$  и  $F^b$  из базового набора, перемножаем их, получая  $F^{a+b}$ , а затем заменяем  $F^a$  на  $F^{a+b}$ , чтобы каждый раз генерировалась другая матрица.

Я обнаружил, что при использовании такой техники матричный метод с матрицей  $30 \times 30$  примерно в 2100 раз быстрее методов Шамира и Мэсси–Омуры на основе возведения в степень с 1024-битовым модулем.

## 16.5 *Двусторонний трехпроходный протокол*

В описанном выше методе умножать на матрицу можно слева или справа, т. е. зашифрованное сообщение может принимать вид  $SM$  или  $MS$ . Но можно также умножать с обеих сторон. В таком случае сообщение разбивается на блоки по  $n^2$  символов и выбирается два независимых коммутативных семейства матриц  $n \times n$ ,  $\mathbf{F}$  и  $\mathbf{G}$ , с порождающими матрицами  $F$  и  $G$ . Сандра будет шифровать сообщение матрицами  $S$  из  $\mathbf{F}$  и  $T$  из  $\mathbf{G}$ , а Рива – перешифровывать с помощью матриц  $R$  из  $\mathbf{F}$  и  $Q$  из  $\mathbf{G}$ .

Сандра отправляет Риве зашифрованное сообщение  $SMT$ . Рива перешифровывает его и отправляет обратно сообщение  $RSMTQ$ . Сандра снимает свое шифрование, умножая на матрицы  $S'$  и  $T'$ , и отправляет  $S'RSMTQT' = RMQ$  Риве, которая дешифрирует его своими обратными матрицами  $R'$  и  $Q'$ , поскольку  $R'RMQQ' = M$ . Двусторонний метод непрактичен для коротких сообщений из-за большого размера блока, но для длинных сообщений он гораздо быстрее одностороннего метода, потому что число символов в каждом блоке равно  $n^2$ , а не  $n$ . Для матриц  $30 \times 30$  ускорение может быть 15-кратным, т. е. примерно в 30 000 раз быстрее метода Шамира или Мэсси–Омура.

Эмили предстоит найти две матрицы одновременно. Пусть Эмили перехватила матрицы  $X$ ,  $Y$  и  $Z$ , т. е.  $X = SMT$ ,  $Y = RSMTQ$  и  $Z = RMQ$ . Эмили знает, что  $Y = RXQ$  и  $Z = S'YT'$ . На первый взгляд кажется, что Эмили должна будет решить большую систему квадратных уравнений над некоммутативным кольцом  $N$ , что гораздо труднее, чем решение системы линейных или билинейных уравнений. Однако

после умножения этих уравнений соответственно на  $R'$ ,  $Q'$ ,  $S$  и  $T$  они принимают вид  $R'Y = XQ$ ,  $YQ' = RX$ ,  $SZ = YT'$  и  $ZT = S'Y$ . После раскрытия скобок эти матричные уравнения дают билинейные уравнения. А как решать билинейные уравнения, мы видели в разделе 16.4.6.

Эмили сможет восстановить  $M$ , если сумеет найти  $R'$  и  $Q'$  или  $S'$  и  $T'$ . Она может выбрать, какие из четырех уравнений решать: первые два или последние два. Продолжим пример с матрицами  $30 \times 30$  и займемся решением уравнения  $R'Y = XQ$ . В  $R'$  имеется 900 неизвестных и еще 900 в  $Q$ . Это матричное уравнение дает 900 билинейных уравнений с 1800 неизвестными. Эмили также знает, что  $R'$  принадлежит  $\mathfrak{F}$ , а  $Q$  принадлежит  $\mathfrak{G}$ , так что  $R'F = FR'$  и  $QG = GQ$ . Каждое из этих условий дает еще  $30 \times 29 = 870$  билинейных уравнений. Всего Эмили имеет 2640 билинейных уравнений с 1800 неизвестными. Эти уравнения можно линеаризовать, изменив представление элементов кольца.

В итоге получаем 7920 линейных уравнений с 5400 неизвестными. Если уравнений больше, чем неизвестных, то система называется *переопределенной*. В процессе приведения системы уравнений Эмили лишние уравнения просто пропадают. То есть многие строки матрицы  $7920 \times 5400$  будут состоять из одних нулей. Их можно сдвинуть в нижнюю часть матрицы и игнорировать. Поскольку двусторонние уравнения образуют переопределенную систему, они сильнее односторонних. С другой стороны, и неизвестных в два раза больше. Не ясно, какой метод более стойкий. Можете отдать предпочтение двустороннему, просто потому что он намного быстрее. \*\*

# 17

## Коды

---

### *Краткое содержание главы:*

- идеи для конструирования кодов.

На протяжении столетий, несмотря на прогресс в области шифров, затем шифровальных машин, а теперь и компьютерной криптографии, военные всегда полагались на коды. Можно предположить, что даже в наши дни военные все еще используют коды как резервный вариант на случай, если электронные устройства выйдут из строя или прервется подача питания.

В большинстве кодов буквы, слоги, слова или фразы заменяются группами символов фиксированного размера, обычно 3, 4 или 5 десятичными цифрами либо 3 или 4 буквами. Встречаются также коды переменной длины. Принято выделять два типа кодов: одинарные и двойные. В случае одинарного кода слова и фразы выписываются в алфавитном порядке, а кодовые группы присваиваются в числовом порядке, хотя и не подряд, так чтобы для поиска слов и кодовых групп можно было использовать один и тот же список. Слабость этого кода очевидна. Если противник прознал, что 08452 означает CANCEL (пушка), то он будет знать, что коды, близкие к 08452, должны соответствовать таким словам, как CAMOUFLAGE, CAMPAIGN, CANCEL, CANINE, CANVAS, CAPITAL, CAPITULATE, CAPSIZE, CAPTAIN и т. д.

В случае двойного кода кодовые группы сопоставляются в случайном порядке. Кодовая книга содержит два отдельных списка: один со словами и фразами в алфавитном порядке, другой с кодовыми группами в числовом порядке. В прошлом на составление кодовых книг уходили месяцы, и обходилось это очень дорого. Поэтому государственные службы могли пользоваться одним и тем же кодом годами, что резко снижало его эффективность. Начиная с 1960-х годов задачу компилирования двойного кода взял на себя компьютер, и занимает это секунды.

Компиляторы кодов применяли множество приемов, делающих код более безопасным. Для часто встречающихся слов и фраз включалось много эквивалентных кодовых групп, или синонимов. Так, код ВМФ мог содержать от 10 до 20 кодовых групп для слова «ship» (корабль), армейский код – много кодовых групп для слова «artillery» (артиллерия), а дипломатический код – для слова «treaty» (договор). В кодах встречалось много групп-пустышек. Целые секции сообщения могли быть абсолютно пустыми, т. е. не нести никакой информации. Некоторые кодовые группы могли принимать несколько значений в зависимости от некоторого индикатора, например последней цифры предыдущей группы.

Иногда кодовые книги печатаются в две колонки. Код берется из левой или правой колонки в зависимости от индикатора. Например, если текущая кодовая группа начинается четной цифрой, то следующая кодовая группа берется из левой колонки, иначе из правой.

## 17.1 Джокер

Джокер – предложенный мной стиль кодирования. Если читатель захочет придумать собственный код, то джокер может стать источником полезных идей. Основная идея заключается в том, что в каждой кодовой группе одна буква или цифра отличается от остальных. Так, в группе из 5 символов четыре символа могут нести какой-то смысл, а пятый, джокер, присутствует только для внесения хаоса. Даже одного null-символа достаточно, чтобы серьезно затруднить задачу противнику, но специальная буква или цифра позволяет сделать гораздо больше.

Предположим, что код состоит из групп по 5 цифр. Четыре цифры – это сам код, а одна – джокер. Для начала будем считать, что в первой кодовой группе каждого сообщения джокер всегда занимает среднюю позицию. Допустим также, что мы работаем с двухколонной кодовой книгой и у кодов в левой и правой колонках значения совершенно различны. Например, 0022 в левой колонке может означать «rescue» (спасите), а 0022 в правой колонке – «engine» (двигатель).

Аналогично могло бы быть две колонки значений джокера, чтобы джокер мог переместиться в другую позицию или в другую колонку.

Ниже перечислены возможные значения джокера. Их гораздо больше десяти. Можете выбрать десятку по своему усмотрению. Или использовать две колонки и взять 20 значений джокера. Или использовать буквы вместо цифр и выбрать 26 значений джокера.

- Начиная со следующей группы джокер перемещается на одну позицию влево.
- Начиная со следующей группы джокер перемещается на одну позицию вправо.
- Начиная со следующей группы джокер перемещается в позицию 1.
- Начиная со следующей группы джокер перемещается в позицию 2 и т. д.
- Только в следующей группе джокер находится в позиции 1.
- Только в следующей группе джокер находится в позиции 2 и т. д.
- Переключиться на левую колонку кодов.
- Переключиться на правую колонку кодов.
- Переключиться на противоположную колонку кодов.
- Только для следующего кода использовать противоположную колонку.
- Для следующих двух кодов использовать противоположную колонку и т. д.
- Далее следует группа-пустышка.
- Вторая после текущей группа является пустышкой.
- Следующие две группы – пустышки и т. д.
- В следующей группе код – пустышка, но джокер настоящий.
- В следующей группе код настоящий, но джокер – пустышка.
- Поменять порядок следующих двух групп.
- Прибавить 1111 к коду в следующей группе (сложение без переноса).
- Прибавить 3030 к коду в следующей группе (сложение без переноса) и т. д.
- Если следующий код четный, прибавить 2222, иначе вычесть 2222 (без переноса).
- Прибавить 1 к джокеру в следующей группе (без переноса).
- Прибавить 2 к джокеру в следующей группе (без переноса) и т. д.
- Прибавить этот 4-значный код к следующему 4-значному коду. Джокер исключается.
- Читать цифры следующего кода в обратном направлении, например 1075 в действительности означает 5701.
- Игнорировать следующие коды, пока не встретится код, начинающийся с 0.
- Следующая группа является специальным индикатором.

Пример сложения без переноса см. в разделе 4.6.

Специальные индикаторы нуждаются в более подробном объяснении. В специальном индикаторе все пять цифр кодовой группы служат специальной цели, например сообщают, где будет джо-

кер или какую колонку использовать. Так, специальный индикатор 13152 мог бы означать, что в следующих пяти группах джокер будет находиться в позициях 1, 3, 1, 5, 2 именно в таком порядке. Специальный индикатор также мог бы сообщить, из какой колонки брать следующие пять кодов: нечетная цифра означала бы, что из левой, а четная – что из правой. Специальный индикатор 10384 означал бы тогда, что в следующих пяти группах коды берутся из левой, правой, левой, правой и правой колонок.

Еще одно возможное применение специального индикатора – задать числа, прибавляемые к кодам в следующих пяти группах. Например, можно было бы установить такое соответствие между цифрами специального индикатора и значениями, прибавляемыми к 4-значным кодам:

1 1000	5 1100	9 0111
2 0100	6 0110	0 1111
3 0010	7 0011	
4 0001	8 1110	

Эти значения следовало бы прибавлять без переноса, т. е. сложение производится по модулю 10.

Джокер всегда задает, какое действие следует предпринять в следующей группе или нескольких группах, но никак не в текущей или предыдущей. Так, не следует наделять джокер семантикой «отменить предыдущий джокер». Если джокер задает действие, распространяющееся на несколько последующих групп, следите за тем, чтобы действия двух джокеров не конфликтовали. Например, не должно быть ситуации, когда джокер в группе 20 говорит «следующие три кода берутся из левой колонки», а джокер в группе 21 – «следующие три кода берутся из правой колонки».

Еще один прием – использовать вместо джокера букву от А до Е. Буква может находиться в любой позиции и переопределять ожидаемую позицию джокера. Буква А означает, что следующий джокер будет находиться в позиции 1, В – в позиции 2 и т. д. Можно также наделить семантикой буквы F, G и H. Например, F могло бы означать, что код следует брать из противоположной колонки, а следующий джокер будет в позиции 1.

Не используйте букву I, если есть хотя бы малейший шанс перепутать ее с цифрой 1. Я люблю использовать букву J в роли суперджокера. Она означает, что все последующее – пустышки. Можете передавать мусор на протяжении 10, 20 или 100 кодовых групп, вогнав Эмили в ступор. Или можете использовать пустышки, чтобы отправить дезинформирующее сообщение, например: «Высадка в Нормандии перенесена на 10 июня, пляж Юта».

# 18

## Квантовые компьютеры

---

### Краткое содержание главы:

- свойства квантовых компьютеров;
- использование квантовых компьютеров для связи;
- использование квантовых компьютеров для распределения ключей;
- использование квантовых компьютеров для решения задач оптимизации;
- использование квантовых компьютеров для дешифрования блочных шифров;
- следующий шаг – ультракомпьютеры.

Сейчас, когда я пишу эту книгу, квантовые компьютеры находятся в состоянии младенчества. Во всем мире насчитывается не более 20 квантовых компьютеров, и ни один не содержит более 50 кубитов, или квантовых битов. Я понимаю, что к моменту выхода книги многое или даже все из написанного устареет или окажется неверным. Математика, применяемая в квантовой механике и квантовых вычислениях, по большей части выходит далеко за рамки этой книги, поэтому во многих случаях я просто называю квантовые методы и алгоритмы, не объясняя, как они работают.

В основе квантовых вычислений лежит *квантовый бит*, или *кубит*. У кубита есть два *базисных состояния*, обозначаемых  $|0\rangle$  и  $|1\rangle$ , они со-



ответствуют состояниям 0 и 1 обыкновенного бита в традиционном компьютере. Нотация  $|1\rangle$  называется *бра-кет*. Когда угловая скобка находится слева,  $\langle 0|$ , говорят о состоянии *бра*, поэтому  $\langle 0|$  читается «бра-0». Когда угловая скобка находится справа, говорят о состоянии *кет*, поэтому  $|1\rangle$  читается «кет-1». Эту нотацию предложил английский физик, нобелевский лауреат Пол Адриен Морис Дирак.

Обыкновенный бит в традиционном компьютере имеет вполне определенное значение: 0 или 1. Никаких других значений, в частности промежуточных, быть не может. Бит не может принимать сразу несколько значений, а иногда не быть равным ни 0, ни 1. Физическое устройство, например намагниченный участок поверхности, можно переключить из одного состояния в другое с помощью электрического тока или магнитного поля. Возможен краткий переходный период, но устройство не может остаться в каком-то промежуточном или смешанном состоянии.

## 18.1 Суперпозиция

С другой стороны, у кубита нет никакого значения до тех пор, пока не будет произведено измерение или наблюдение. В этот момент значение может быть равно 0 или 1. Базисное состояние  $|0\rangle$  означает, что с вероятностью 1.0 значение равно 0, а базисное состояние 1 – что оно равно 1. В общем случае состояние кубита является *суперпозицией* обоих базисных состояний  $\alpha|0\rangle + \beta|1\rangle$ , где  $\alpha$  и  $\beta$  – комплексные числа такие, что  $|\alpha|^2 + |\beta|^2 = 1$ . Вероятность, что результатом измерения этого кубита будет 0, равна  $|\alpha|^2$ , а вероятность, что оно окажется равным 1, равна  $|\beta|^2$ . Нотация  $|\alpha|$  обозначает *модуль*  $\alpha$ . Модуль комплексного числа  $a + bi$  равен  $\sqrt{a^2 + b^2}$ . Поскольку результат измерения имеет вероятностный характер, измерение двух кубитов в одинаковых состояниях может давать разные результаты. В суперпозиции может находиться сколько угодно состояний.

Если квантовое состояние  $x$  состоит из нескольких кубитов, скажем  $x_1, x_2, x_3$ , то состояние  $\langle x|$  представляется в виде вектора-строки  $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$ , в котором черточка над каждым компонентом означает *комплексное сопряжение*. Для комплексного числа  $\alpha = a + bi$  комплексно сопряженным называется число  $\bar{\alpha} = a - bi$ . При этом произведение  $\alpha\bar{\alpha} = a^2 + b^2 = |\alpha|^2$ . С другой стороны, состояние  $|y\rangle$  представляется вектором-столбцом

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}.$$

Поскольку в этом примере вектор-строка – матрица  $1 \times 3$ , а вектор-столбец – матрица  $3 \times 1$ , их можно перемножить. Произведением

матриц, обозначаемым  $\langle x|y \rangle$ , является матрица  $1 \times 1$  с единственным элементом, равным скалярному произведению  $\vec{x} \cdot \vec{y}$ . Таким образом,  $\langle x|y \rangle$  – скаляр. (Если что-то не понятно, перечитайте раздел 11.3.)

Поскольку к любым двум состояниям можно применить суперпозицию, а сами состояния могут, в свою очередь, являться суперпозицией других состояний, всякий кубит может являться суперпозицией произвольного числа состояний.

Суперпозиция состояний недолговечна. Малейшие возмущения, например колебания температуры или механическая вибрация, могут вывести кубит из состояния суперпозиции и вернуть в одно из базисных состояний. Это явление называется *декогеренцией*. Неустойчивость – главное препятствие на пути конструирования больших и надежных квантовых компьютеров. В частности, в процессе измерения когерентность кубита нарушается, и он оказывается в наблюдаемом базисном состоянии. По той же причине кубит нельзя скопировать, т. е. для этого необходимо наблюдение.

Если вам трудно воспринять идею коэффициента, являющегося комплексным числом, то, возможно, поможет следующее объяснение. Представьте точку  $(a,b)$  в декартовой системе координат. Отрезок прямой, соединяющий начало координат  $(0,0)$  с точкой  $(a,b)$ , – это вектор. У него есть модуль и направление. В случае суперпозиции двух состояний эти векторы складываются по правилам аналитической геометрии, т. е. в точности так, как складываются комплексные числа. Именно поэтому вероятности и представляются комплексными числами. После сложения векторов коэффициенты необходимо перенормировать, чтобы было выполнено условие  $|\alpha|^2 + |\beta|^2 = 1$ . Перенормировку можно устранить, если  $\alpha$  и  $\beta$  описываются в терминах углов с применением тригонометрических формул для суммы и разности углов.

Над кубитами можно производить элементарные логические операции и таким образом строить квантовые схемы. Примером может служить условная функция НЕ, CNOT, применяемая к двухбитовым кубитам  $|xy\rangle$ . CNOT определена как  $|xy\rangle$ , если  $x = 0$ , и  $|xy'\rangle$ , если  $x = 1$ . Иными словами, первый бит не изменяется, а второй вычисляется как ИСКЛЮЧАЮЩЕЕ ИЛИ обоих битов.

## 18.2 Квантовая запутанность

Помимо суперпозиции, у частиц есть еще одно квантово-механическое свойство – *запутанность*. Группа частиц называется *запутанной*, если имеется корреляция между каким-то свойством одной частицы и таким же свойством другой. Например, у электронов есть свойство *спин*. Спины нескольких частиц относительно определенной оси, например оси  $x$ , могут быть коррелированы. Также запутанной может быть поляризация в группе фотонов. Запутанность может

сохраняться, даже когда частицы удалены друг от друга на большое расстояние. Поэтому запутанность можно использовать для связи.

Процесс начинается созданием запутанной пары частиц. Один из способов добиться этого – пропустить лазерный луч через кристалл специального типа. В результате часть высокоэнергетических фотонов расщепляется на пару низкоэнергетических фотонов. Некоторые из таких фотонных пар будут запутаны, хотя выход очень низкий, порядка одного на миллиард. Следующий шаг – доставить эти запутанные фотоны туда, где Сандра и Рива будут заниматься передачей и приемом. На большие расстояния фотоны обычно передаются по волоконно-оптическому кабелю, хотя можно переместить их и физически, используя дефекты кристаллической решетки.

Когда Сандра будет готова отправить сообщение, она организует взаимодействие своего фотона со специальным подготовленными дополнительными фотонами, которые называются *анциллами*. В результате взаимодействия ее фотон переходит в желаемое состояние, которое она хочет передать. Это приводит к тому, что запутанный фотон, который может быть удален на много километров, переходит в дополнительное состояние. Распространено предположение, что это происходит мгновенно, но на самом деле изменение распространяется со скоростью света. Информация не может передаваться мгновенно, что бы ни писали на эту тему поколения фантастов.

Наконец, Рива измеряет свой запутанный фотон и определяет однокбитовое сообщение – или не определяет, поскольку это вероятный процесс. Иногда это называют *квантовой телепортацией*, но этот термин употребляется учеными, которые в детстве перечитали слишком много научной фантастики. Предполагается, что такая связь защищена от прослушивания, потому что если Эмили измерит фотон, он подвергнется декогеренции, и Сандра с Ривой смогут это обнаружить.

Но в этом рассуждении есть два изъяна. (1) Эмили может быть все равно, обнаружили подслушивание или нет. Коль скоро она заполучила информацию, уже не важно, знают ли Сандра и Рива о том, что она знает. (2) Целью Эмили может быть не сбор информации, а нарушение связи. Эмили, возможно, не узнает планов боя, но их не узнает и Рива. На самом деле если Сандра и Рива обнаружат, что Эмили подслушивает, то, наверное, станут использовать квантовый канал связи реже, что Эмили вполне устраивает.

## 18.3 Исправление ошибок

Поскольку квантовые события имеют вероятностный характер, частота ошибок у квантовых компьютеров гораздо выше, чем у традиционных. Поэтому необходимы средства обнаружения и исправления ошибок. Для классических компьютеров существуют коды,

обнаруживающие и исправляющие ошибки. В них для обнаружения ошибок используются дополнительные биты, например к каждому байту можно добавить бит четности, который обычно равен результату применения ИСКЛЮЧАЮЩЕГО ИЛИ ко всем восьми битам данных. При таком определении результирующий 9-битовый байт всегда должен быть четным. Если он оказался нечетным, значит, произошла ошибка, но в каком именно бите, неизвестно.

Для традиционных компьютеров простейший код, исправляющий ошибки, – код «2 из 3». Имеется три копии каждого бита. Если произошла ошибка в одном бите, то значения двух его копий будут правильными. Если шанс ошибки в одном бите равен, скажем,  $1$  к  $10^7$ , то такое использование мажоритарного значения снижает вероятность ошибки до  $3$  к  $10^{14}$  – существенное улучшение. Использовать 3 бита для представления одного дорого, но есть несколько алгоритмов кодирования, например коды Хэмминга и сверточные коды, требующие меньшего числа дополнительных битов, причем некоторые коды могут обнаруживать и исправлять ошибки в нескольких битах. Безошибочность связи – непереносимое условие в современной криптографии, где изменение даже одного бита может сделать сообщение не поддающимся расшифровке.

Такого типа обнаружение и исправление ошибок в квантовых компьютерах невозможны. Эти коды предполагают возможность скопировать значение бита и проверить четность. С кубитами это не пройдет, потому что измерение его значения приводит к декогеренции. Попытки реализовать проверку квантовых ошибок обычно опираются на использование дополнительных кубитов. Кубиты, предназначенные для обнаружения и исправления ошибок, расположены вместе с информационными кубитами в плоской решетке. Все это вместе называется *поверхностным кодом*.

В настоящее время квантовая коррекция ошибок – всего лишь теория. Еще никому не удалось построить реальное устройство. Необходимость в дополнительных битах исправления ошибок резко увеличивает число кубитов в практически полезном квантовом компьютере. Поскольку частота квантовых ошибок велика, до создания реальных квантовых компьютеров, по-видимому, еще очень далеко. Помните об этом, читая описания различных квантовых алгоритмов в последующих разделах.

## 18.4 Измерение

Измерить поляризацию фотона – дело непростое. Задумайтесь, как вы стали бы измерять поляризацию светового луча. Вы пропустили бы луч через поляризационный фильтр и наблюдали бы его яркость. Затем начали бы медленно вращать фильтр, пока яркость прошедшего через него света не достигла бы максимума. В этот момент

фильтр совмещен с поляризацией луча, и угол его поворота можно измерить.

Но у Ривы такой возможности нет. Она имеет дело всего с одним фотоном. Он проходит через ее фильтр или кристалл, и она либо детектирует вспышку, либо нет. Если ее фильтр ориентирован не так же, как излучатель Сандры, то шансы получить такое же состояние, как у Сандры, зависят от относительного угла. Например, если ее детектор повернут на  $90^\circ$  относительно излучателя Сандры, то у нее ровно 50 % шансов получить такое же значение кубита.

Сандра может решить проблему, отправив сразу пачку фотонов. Рива может пропустить эти фотоны через различные фильтры и измерить яркость каждого образца с помощью фотодетектора и вольтметра и точно вычислить угол поляризации. Далее она производит измерения при таком угле и с очень высокой вероятностью получает то же базисное состояние, что у Сандры. Возможность использовать квантовые компьютеры в криптографии в конечном счете зависит от способности различать мельчайшие градации поляризации.

## 18.5 Квантовый трехэтапный протокол

Мы подготовили сцену для *трехэтапного квантового протокола*, который в 2006 году предложил Субхаш Как из Оклахомского университета в Стиллуотере. Трехэтапный протокол Кака основан на той же схеме с тремя сообщениями, что и другие трехпроходные алгоритмы, рассмотренные в разделах 16.1, 16.2 и 16.4. В квантовой версии операция шифрования – это поворот поляризации на случайный угол вокруг выбранной пространственной оси. Сандра и Рива должны согласовать ось, в противном случае повороты не будут коммутировать. (1) Сандра отправляет фотон, повернутый на ее случайный угол  $\phi$ , (2) Рива поворачивает фотон на свой секретный угол  $\psi$  и отправляет Сандре фотон, повернутый на угол  $\phi + \psi$ , (3) Сандра поворачивает фотон в обратном направлении на угол  $-\phi$  и отправляет Риве фотон, повернутый на ее угол  $\psi$ . Рива снимает этот поворот и читает кубит. Попытавшись измерить любой из повернутых кубитов, Эмили не будет знать, правильный ли угол установлен в ее детекторе, а значит, не будет знать вероятность получения правильного значения.

Применяя этот метод, Сандра и Рива должны часто изменять свои углы, желательно для каждого бита. В противном случае Эмили сможет просто выбирать случайный угол и пытаться читать каждое сообщение. Если выбранный ей угол близок к правильному, то она будет получать правильные значения 80, а то 90 % бит. Этого может хватить для прочтения сообщения. Если повезет, она сможет прочитать примерно 25 % сообщений. Отметим, что угол, отличающийся от истинного на  $180^\circ$ , тоже устроит Эмили, потому

что так она получит от 80 % до 90 % инвертированных битов – тоже неплохо.

## 18.6 Квантовое распределение ключей

Существуют алгоритмы квантового распределения ключей, аналогичные алгоритму Диффи–Хеллмана. Самый известный из них – *BB84*, названный по именам авторов, Чарльза Х. Беннета из IBM Research и Жилия Брассара из Монреальского университета. В алгоритме используется 4 кубита, которые позволяют обнаруживать и исправлять ошибки из-за шума в канале связи. Возмущения, приносимые Эмили, рассматриваются как дополнительный шум в канале, поэтому не нуждаются ни в обнаружении, ни в исправлении.

Из этой работы вытекает, что несколько слабо запутанных частиц можно скомбинировать и получить меньшее число сильно запутанных частиц.

## 18.7 Алгоритм Гровера

*Криптографический алгоритм Гровера*, предназначенный для вскрытия блочных шифров с секретным ключом, в частности DES и AES, на квантовых компьютерах, разработал в 1996 году Лов Кумар Гровер из Bell Labs на основе собственного алгоритма квантового поиска в файле. В нем каждое вычисление функции шифрования рассматривается как один доступ для чтения неотсортированной базы данных. Алгоритм уменьшает ожидаемое число вычислений с  $K$  до  $\sqrt{K}$ , где  $K$  – число возможных ключей. По сути дела, размер ключа уменьшается с  $n$  до  $n/2$  бит.

Алгоритм Гровера с высокой вероятностью находит ключ  $k$ , для которого  $E(k, p) = c$ , где  $E$  – функция шифрования,  $p$  – открытый текст, а  $c$  – шифртекст. Алгоритм требует один блок известного открытого текста для каждого такого ключа. Специалист по квантовой физике, мало что знающий о криптографии, может сделать вывод, что для защиты от алгоритма Гровера нужно просто удвоить размеры всех криптографических ключей. Но это было бы неэффективно, потому что пришлось бы увеличить количество раундов блочного шифра. Например, в AES со 128-битовым ключом используется 10 раундов, а в AES с 256-битовым ключом – 14 раундов.

Более дешевая альтернатива – расширить ключи, добавив до и после основного шифрования какой-нибудь простой и быстрый шифр, например простую подстановку. Ключи для перемешивания двух простых алфавитов подстановки могут достигать 1684 бит каждый (см. раздел 5.2), поскольку число перестановок каждого алфавита равно  $256!$ , что близко к  $2^{1684}$ . Простая перестановка также

может увеличить размер ключа, но не так существенно, потому что  $16!$  приближенно равно всего  $2^{44}$ . Если вы решите воспользоваться перестановкой, то можете переставлять блоки по два за раз, потому что  $32!$  приближенно равно  $2^{118}$ , т. е. размер ключа увеличивается намного.

Читатели этой книги наверняка поняли, что алгоритм Гровера можно обмануть такими элементарными приемами, как использование null-символов, использование разных ключей для каждого блока, сцепление блоков или сжатие сообщения. Это значит, что если предпослать блочному шифру сжатие, например перемешанный код Хаффмана (раздел 4.2.1), то обе цели – увеличение ключа и сжатие – будут достигнуты одновременно. Недостаток перемешанного кода Хаффмана в том, что он изменяет размер блока. Пожалуй, лучше использовать подстановку Хаффмана (раздел 10.4) или Поста (раздел 10.5) до и после блочного шифра.

## 18.8 Уравнения

Прежде чем обсуждать следующую тему, квантовую имитацию отжига, нужно сказать несколько слов об уравнениях. Многие шифры можно выразить в виде системы уравнений. Шифр Беласо можно записать в виде  $C = P + K$ , где  $C$  – шифртекст,  $P$  – открытый текст, а  $K$  – ключ; все целые числа по модулю 26. Шифр Хилла – это система линейных уравнений. Шифры Плейфера и Two-Square можно выразить как уравнения в пятеричной системе счисления.

### 18.8.1 Перестановки

Перестановки легко записываются в виде систем равенств. Например, столбцовую перестановку

<b>E</b>	<b>X</b>	<b>A</b>	Открытый текст:	<b>EXAMPLE</b>
<b>M</b>	<b>P</b>	<b>L</b>	Шифртекст:	<b>EMEXPAL</b>
<b>E</b>				

можно выразить в виде  $c_1 = m_1$ ,  $c_2 = m_4$ ,  $c_3 = m_7$ ,  $c_4 = m_2$ ,  $c_5 = m_5$ ,  $c_6 = m_3$ ,  $c_7 = m_6$ , где  $m_i$  – символы открытого текста, а  $c_j$  – символы шифртекста.

Логические функции можно преобразовать в числовые уравнения следующим образом:

**not**  $x \rightarrow 1-x$   
**x or**  $y \rightarrow x+y-xy$   
**x and**  $y \rightarrow xy$   
**x xor**  $y \rightarrow x+y-2xy$



### 18.8.2 Подстановки

Подстановки можно привести к форме уравнения с помощью трехшагового процесса. Сначала представим каждый бит шифртекста булевым выражением, содержащим биты ключа и открытого текста. Например, рассмотрим следующую подстановку, которая принимает однобитовый ключ  $K$  и двухбитовый открытый текст  $AB$  и порождает двухбитовый шифртекст  $XY$ .

$K$	$AB$	$XY$	Булевы входы
0	00	01	$\bar{K}\bar{A}\bar{B}$
0	01	11	$\bar{K}\bar{A}B$
0	10	00	$\bar{K}A\bar{B}$
0	11	01	$\bar{K}AB$
1	00	10	$K\bar{A}\bar{B}$
1	01	00	$K\bar{A}B$
1	10	10	$KA\bar{B}$
1	11	11	$KAB$

Здесь  $\bar{K}\bar{A}\bar{B}$  означает  $K = 0$ ,  $A = 0$  и  $B = 0$ ,  $\bar{K}\bar{A}B$  –  $K = 0$ ,  $A = 0$  и  $B = 1$  и т. д. Бит шифртекста  $X$  теперь можно записать в виде  $X = \bar{K}\bar{A}\bar{B} + \bar{K}\bar{A}B + \bar{K}A\bar{B} + KAB$ . Аналогичное выражение можно выписать для  $Y$ .

### 18.8.3 Карты Карно

Карты Карно используются для приведения, или упрощения, этих выражений. Это второй шаг. Идея принадлежит Морису Карно из Bell Labs, высказавшему ее в 1953 году, и заключается в том, чтобы изобразить множество всех возможных  $n$ -битовых входов в виде  $n$ -мерного куба  $2 \times 2 \times 2 \times \dots \times 2$ . Закрасим каждую ячейку, в которой выходной бит равен 1. Ниже показана карта для выходного бита  $X$ . Аналогичную карту можно построить для  $Y$ .

	$\bar{A}\bar{B}$	$A\bar{B}$	$AB$	$\bar{A}B$
$\bar{K}$				
$K$				

Обратите внимание, как обозначены столбцы в этой карте. При переходе от одной ячейки к другой слева направо на каждом шаге изменяется только один бит, это относится даже к переходу от столбца 4 к столбцу 1. Такая конфигурация называется *кодом Грэя*. Коды Грэя были предложены Фрэнком Грэем из Bell Labs в 1947 году. Код Грэя легко построить, дописывая по одному биту. Например, чтобы расширить этот 2-битовый код Грэя до 3-битового, мы сначала выпишем четыре пары  $A, B$  в порядке  $\bar{A}\bar{B}$ ,  $A\bar{B}$ ,  $AB$ ,  $\bar{A}B$  и допишем

$\bar{C}$  в конец каждой пары, а затем выпишем те же пары, но инвертированные, и допишем в конец каждой  $C$ . Бит  $C$  изменяется всего два раза, после четвертой и после восьмой группы.

Визуально карты Карно позволяют оптимизировать логику, когда число битов не превышает 6, три по горизонтали и три по вертикали, для чего строится карта  $8 \times 8$ . Когда число битов больше 6, лучше написать программу. На каждом шаге добавляется наибольший прямоугольный блок, который умещается в закрашенную область и покрывает хотя бы одну новую ячейку, которая не была покрыта ранее. Каждое измерение блока должно быть степенью 2, поэтому его объем также будет степенью 2. Если имеется несколько блоков максимального размера, то выбирается тот, который покрывает наибольшее число еще не покрытых ячеек. Так продолжается, пока не будут закрашены все ячейки.

В примере с  $K, A, B$  имеется два блока  $1 \times 2$  в закрашенной области,  $KA$  и  $K\bar{B}$ . Каждый покрывает две ячейки. Поскольку вместе они покрывают 3 ячейки, необходимы оба. Для покрытия остается только ячейка  $KAB$ . Поэтому приведенное выражение для  $X$  имеет вид  $KA + K\bar{B} + KAB$ .

Третий шаг выражения подстановки в виде системы уравнений заключается в том, чтобы заменить функции **and**, **or** и **not** в этих выражениях арифметическими выражениями, следуя сформулированным выше правилам.

## 18.8.4 Промежуточные переменные

Если мы попытаемся записать каждый бит шифртекста в сложном блочном шифре типа AES одним выражением, то размер этого выражения будет экспоненциально возрастать с каждым следующим раундом. Иногда эту проблему называют причиной, по которой невозможно использовать уравнения для вскрытия блочных шифров. Чувствительная. Эту проблему можно устранить с помощью промежуточных переменных. Будем считать выходы каждого раунда отдельным набором переменных.

Входы первого раунда – ключ, открытый текст и вектор(ы) сцепления – это *независимые* переменные. Любой из этих битов может изменяться независимо от остальных. Выходы каждого раунда или каждого этапа внутри раунда – *зависимые* переменные. К ним относятся и векторы сцепления для следующего блока. Их значения полностью определены значениями независимых переменных. Невозможно изменить один из этих битов, не изменив никакую другую переменную.

## 18.8.5 Известный открытый текст

Предположим, что Эмили располагает каким-то объемом открытого текста. Для простоты пусть это будет  $n$ -битовый блок сообщения.

Ее цель – воспользовавшись известным открытым текстом и перехваченным шифртекстом, определить ключ. Допустим, что Эмили нашла выражение каждого бита шифртекста в терминах открытого текста, ключа и, возможно, вектора сцепления. Обозначим  $E_i$  выражение  $i$ -го бита, и пусть  $c_i$  –  $i$ -й бит шифртекста. Для любого ключа  $K$  Эмили может измерить различие между шифртекстом, который получился бы при шифровании известного открытого текста ключом  $K$ , и перехваченным шифртекстом. Для этого она вычисляет функцию

$$D(K) = (E_1 - c_1)^2 + (E_2 - c_2)^2 + (E_3 - c_3)^2 + \dots + (E_n - c_n)^2.$$

Если найден правильный ключ, то  $D(K)$  будет равно 0. Здесь  $D(K)$  называется *целевой функцией*, или просто *оценкой*.

## 18.9 Минимизация

Введение целевой функции превращает задачу нахождения правильного ключа в задачу минимизации. Цель – минимизировать значение функции  $D(K)$ . Квантовые компьютеры работают, потому что квантовая система всегда стремится перейти в состояние с наименьшей энергией. Если квантовый компьютер можно сконфигурировать так, что кубиты или группы кубитов представляют значения переменных, а энергия системы соответствует значению целевой функции, то состояние с наименьшей энергией будет соответствовать минимуму целевой функции. Если такая конфигурация достижима, то квантовые компьютеры смогут решать широкий спектр реальных задач, в т. ч. вскрывать шифры.

Начнем с того, что заменим биты ключа вещественными числами. В конечном итоге они должны принять значения 0 или 1, но удобно на время поиска допустить и другие значения переменных. Зададим начальные значения, например положим все биты равными 0.5 или каким-то случайным значениям в диапазоне от 0 до 1, а затем будем изменять их, стараясь уменьшить величину  $D(K)$  и довести ее до 0.

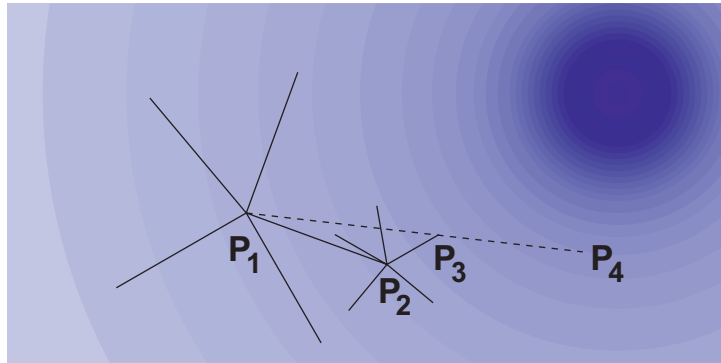
Для традиционных компьютеров разработано много методов оптимизации, но мы рассмотрим только три. При попытке применить эти алгоритмы к поиску криптографических ключей потребуется большой объем известного открытого текста. Как минимум его размер должен в три раза превышать размер ключа.

### 18.9.1 Восхождение на вершину

Метод *восхождения на вершину*, называемый также *методом наискорейшего спуска* или *градиентным методом*, – один из самых старых методов оптимизации. Идея заключается в том, чтобы, начав с какой-то точки  $P_1$ , рассмотреть несколько случайных точек, уда-

ленных от нее на одно и то же расстояние. Из них выберем точку  $P_2$ , которая дает наибольшее улучшение – в данном случае наименьшее значение  $D(K)$ . Затем уточним направление, для чего рассмотрим случайные точки в окрестности  $P$ . Расстояния от  $P_2$  до любой из этих точек будут значительно меньше, чем от  $P_1$  до  $P_2$ . Назовем следующую точку  $P_3$ . Прямая, соединяющая  $P_1$  и  $P_3$ , определяет направление поиска. Наконец, найдем на этой прямой точку  $P_4$ , доставляющую минимум  $D(K)$ . Поиск повторяется с  $P_4$  в качестве начальной точки. В процессе поиска величины шагов при переходе от  $P_i$  к  $P_{i+1}$  увеличиваются всякий раз, как обнаружено улучшение, и уменьшаются, если улучшения не было.

Такой поиск хорошо работает, если форма пространства поиска напоминает одиночную гору в  $n$ -мерном пространстве или одну высокую гору, окруженную большим числом более низких вершин. Но может потерпеть полный провал, если топография более сложная, т. е. изобилует локальными минимумами. На рисунке ниже чем темнее цвет, тем лучше оценка.

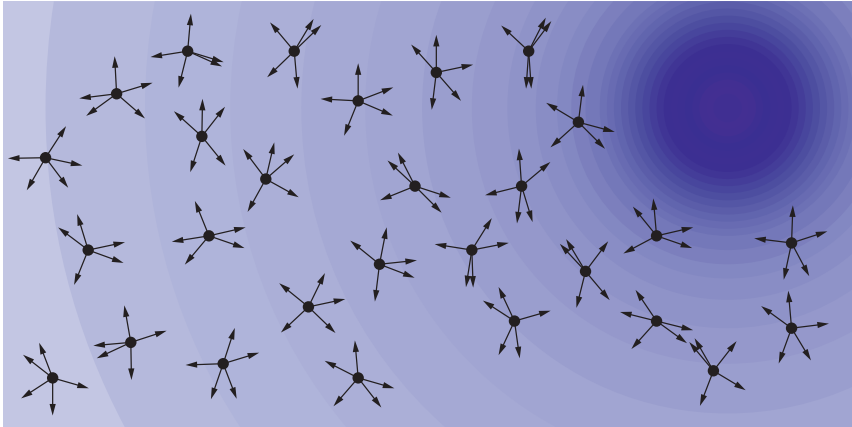


### 18.9.2 Тысяча вершин

*Тысяча вершин* – идея, которую я использовал, чтобы набрать победные баллы в различных конкурсах головоломок, в которых принимал участие в 1970-х годах. Позже я принялся было за написание статьи для компьютерных журналов с подробным описанием этого метода поиска, но погряз в деталях, пытаюсь охарактеризовать типы целевых функций, для которых он оказывается лучше других методов. В 1990-х годах метод был открыт заново под названием *рой частиц*.

Изобразим пространство поиска как горную гряду с большим числом пиков, долин и хребтов. Теперь представим себе эскадрилью самолетов, которые летят над местностью и сбрасывают сотни альпинистов на парашютах. Иными словами, начальных точек будет много, все они исследуются одновременно. Альпинисты разглядывают ближайшие точки и оценивают, выше они или ниже. Есть два варианта действий. (1) Можно брать только лучшую из этих точек

и двигаться туда. Если лучших точек нет, уменьшаем размер шага и пробуем снова. Если, скажем, три попытки подряд завершились неудачно, то сбрасываем нового альпиниста, который начнет со случайной точки. (2) Сохранять несколько точек, для которых зафиксировано улучшение. Можно считать, что команда альпинистов разделилась на несколько партий, исследующих разные маршруты. Лучше не брать все улучшающие оценку решения, потому что тогда все альпинисты быстро соберутся в немногих районах.



Первоначально я хотел хранить все решения в пирамидальной структуре, так чтобы на вершине всегда оказывалось худшее решение. Тогда нужно взять его и попытаться улучшить. Но это оказалось неэффективно, потому что много времени тратилось на улучшение плохих решений, которые в конечном итоге отбрасывались. Если, наоборот, всегда выбирать наилучшее решение, то все альпинисты соберутся на одной вершине. Лучшая стратегия – выбирать следующего альпиниста случайным образом. Аналогично, когда имеется несколько улучшающих оценку решений, не всегда выгодно выбирать из них наилучшее. Иногда лучше выбрать какое-то из них случайно.

### 18.9.3 Имитация отжига

*Имитация отжига* – популярный метод оптимизации, в основном потому что легко реализуется. Мы начинаем со случайной точки в пространстве поиска и рассматриваем точку поблизости от нее. Если это решение оказывается лучше, то мы переходим в новую точку с вероятностью  $V$ , а если хуже – то с вероятностью  $W$ .

Главная особенность имитации отжига в том, что эти вероятности изменяются в процессе поиска. Первоначально вероятности отвергнуть хорошее решение или принять плохое задаются довольно высокими, скажем отвергать 40 % лучших решений и принимать 30 %

худших, т. е.  $B = 0.6$  и  $W = 0.3$ . С течением времени, скажем после 1000 шагов, вероятность отвергнуть хорошее решение уменьшается. Например, на втором этапе можно отвергать 20 % лучших решений и принимать 15 % худших. Спустя еще некоторое время, скажем после 2000 шагов, мы будем отвергать только 10 % лучших решений и принимать 7 % худших.

Этот процесс называется имитацией отжига, потому что напоминает процесс отжига, применяемый в металлургии, когда металл сначала раскаляется, а затем медленно охлаждается. При этом изменяется кристаллическая структура металла, в результате чего уменьшается твердость, но увеличиваются пластичность и ковкость, так что он легче поддается обработке. При имитации отжига начальные высокие вероятности отбрасывания лучших решений и принятия худших аналогичны начальной высокой температуре металла, а постепенное уменьшение этих вероятностей – его медленному охлаждению. В описаниях метода имитации отжига этапы пошагового изменения вероятностей часто называют *уменьшением температуры*. Хочу дать несколько советов, основанных на моем собственном опыте работы с имитацией отжига.

- Не имеет смысла продвигаться слишком медленно. Будет напрасной тратой времени сначала отвергать 40 % лучших решений, затем 39 %, потом 38 % и т. д. На каждом этапе частота принятия/отбрасывания должна составлять от 1/2 до 2/3 предыдущей частоты. Например, 40 % на первом этапе, затем 20 %, 10 %, 5 % и 3 %. Или 40 % на первом этапе, потом 25 %, 15 %, 10 %, 6 %, 4 % и, наконец, 2.5 %.
- Пяти этапов обычно достаточно.
- Не стоит начинать с частоты принятия 50 %. Выбирайте начальное значение между 60 % и 75 %.
- Не стоит доходить до 0 %. Улучшение будет более значимым, если на последнем этапе принимается от 2 % до 3 % худших решений.
- Останавливайтесь, когда ничего не происходит. Возможно, вы планировали 1000 попыток на каждом этапе, но если после 100 попыток изменения прекратились, останавливайтесь.
- Процент лучше делать зависящим от величины улучшения. Например, на первом этапе можно принимать 60 % изменений, улучшающих оценку на 1 %, 75 % изменений, улучшающих оценку на 2 %, и 90 % изменений, улучшающих оценку на 3 % и более.
- Экспериментируйте. Все задачи оптимизации различны. Пробуйте варьировать количество этапов, количество попыток на каждом этапе, темп изменения вероятностей, размер шага и связь между степенью улучшения и процентной долей принятия лучших решений.

Методы восхождения на вершину, тысячи вершин и имитации отжига можно комбинировать между собой.

## 18.10 Квантовая имитация отжига

Для квантовых компьютеров предложено несколько методов имитации отжига. В них используются такие квантовые явления, как суперпозиция, чтобы выполнять много операций поиска параллельно. Однако в каждой попытке требуется вычислять целевую функцию в выбранной точке. Квантовые компьютеры не предназначены для вычисления выражений. На данный момент не существует способов параллельного вычисления этих функций квантовыми средствами. Квантовый компьютер может обращаться за этим к обычному, но тогда утрачивается параллелизм. Пока что квантовый поиск не продемонстрировал никакого ускорения по сравнению с поиском на традиционных компьютерах.

## 18.11 Квантовая факторизация

Само существование криптосистемы RSA с открытым ключом опирается на трудность факторизации больших целых чисел. Два больших целых числа  $A$  и  $B$  легко перемножить и найти произведение  $AB$ , но обратить этот процесс и найти множители большого целого числа очень трудно. Трудность факторизации большого числа сравнима с трудностью вычисления дискретного логарифма (раздел 16.3), и для решения обеих задач используются сходные методы.

Эту безопасность потенциально может поставить под сомнение алгоритм Шора факторизации больших чисел. Исторически это первый квантовый алгоритм, он был предложен Питером Шором из MIT в 1994 году. Если его удастся успешно реализовать, то либо от RSA придется отказываться, либо значительно увеличивать размер модуля, быть может, до миллионов битов. На данный момент с помощью алгоритма Шора удалось факторизовать число  $15 = 3 \times 5$  в 2001 году и число  $21 = 3 \times 7$  в 2012 году. Такими темпами можно ожидать, что где-то в 2023 году будет разложено на множители число  $35 = 5 \times 7$ . Но юмор в сторону, могут пройти еще десятилетия, прежде чем алгоритм Шора станет реальной угрозой безопасности RSA.

## 18.12 Ультракомпьютеры

Квантовые компьютеры не предназначены для вычисления выражений – сегодня. Но давайте предположим, что это всего лишь техническая проблема. Допустим, что со временем появятся гибридные компьютеры, объединяющие вычислительную мощь суперкомпьютеров с параллелизмом квантовых компьютеров. Будем называть их *ультракомпьютерами*.



Какие действия может предпринять Сандра сегодня, чтобы во всеоружии встретить час, когда у Эмили появится ультракомпьютер? Кое-какие идеи можно почерпнуть из способа, которым мы обошли алгоритм Гровера (раздел 18.6). Мы увеличили размер ключа, чтобы выйти за пределы возможностей этого алгоритма. Это можно повторить и для ультракомпьютеров. Можно увеличить количество неизвестных, так что предполагаемой мощности ультракомпьютера не хватит для их нахождения. Рассмотрим два аспекта этого подхода: подстановку и генерирование случайных чисел.

Этим алгоритмам понадобятся очень большие ключи шифрования. Просто примем на веру, что в будущем мире, где существуют ультракомпьютеры, работать с такими ключами станет возможно.

### 18.12.1 Подстановка

Если подстановка не определена каким-то математическим правилом, то для ее определения нужна таблица подстановки. Все элементы этой таблицы известны Сандре, но неизвестны Эмили. Каждый элемент можно рассматривать как математическую переменную. В начальный момент все переменные могут иметь произвольные значения. Если Эмили сможет узнать какие-то из них, то множество возможных значений остальных переменных сужается, но первоначально вместо любого символа можно подставить любой другой.

Цель Сандры – превзойти возможности ультракомпьютера. В многоалфавитном шифре общего вида размер таблицы алфавитов равен  $26 \times 26$  при ручном использовании и  $256 \times 256$  для компьютера. Таким образом, число неизвестных равно  $2^{16} = 65\,536$ . Но нет никаких причин ограничиваться таблицей с 256 строками. Если у Эмили имеется ультракомпьютер, то логично предположить, что и у Сандры есть быстрый компьютер с большой памятью. Сандра могла бы использовать таблицу алфавитов с 1024 строками с 10-битовыми ключами, или 4096 строками с 12-битовыми ключами, или даже с 65 536 строками с 16-битовыми ключами. Это потребовало бы  $2^{24} = 16\,777\,216$  байт внутренней памяти для хранения таблицы подстановки, что вполне доступно современным персональным компьютерам. К тому же использование 16-битового ключа для 8-битовой подстановки дает весьма желательную избыточность. Назовем таблицу алфавитов с  $2^{24}$  элементами *Tab24*. В каждой строке *Tab24* хранится отдельный ключ перемешивания. Если его длина равна 256 битам, то вся таблица занимает  $256 \times 65\,536 = 16\,777\,216$  бит – таков размер ключа шифра.

Сандра также вполне в состоянии использовать полную таблицу биграмм. Для хранения таблицы биграмм  $256 \times 256$  с 8-битовым ключом для выбора строки (фактически слоя) потребовалось бы  $2^{25} = 33\,554\,432$  байта внутренней памяти. И это практически возможно даже сегодня. Но если бы таблица имела 65 536 слоев, а длина каждого ключа составляла 16 бит, то понадобился бы гораздо более мощный компьютер.

Вспомним, однако, что эти таблицы подстановки должны храниться в секрете и быть абсолютно случайными. Даже если они генерируются каким-то алгоритмом, определение начального состояния и параметров генератора должно выходить далеко за пределы возможностей ультракомпьютера Эмили. Некоторые подходящие методы см. в разделе 13.13.

### 18.12.2 Случайные числа

Методы из раздела 13.13 – неплохое начало, но чтобы создать генератор псевдослучайных чисел, способный противостоять ультракомпьютеру, мы объединим идею генератора выбора из раздела 13.11 с техникой обновления генератора из раздела 13.15.

В *ультрагенераторе* UG используется три массива, A, B и C. Массивы A и B содержат по 65 536 элементов – 24-битовых целых чисел. Массив C содержит  $2^{24} = 16\,777\,216$  8-битовых целых чисел. Все три массива можно инициализировать по фотографиям природы, как описано в разделе 13.14.2. У Сандры и Ривы должны быть одинаковые массивы. Генератор порождает 8-битовый выход в каждом цикле. Цикл n состоит из следующих шагов:

- 1 вычислить  $x = (A_n + A_{n-103} + A_{n-1071}) \bmod 16777216$  и заменить  $A_n$  на x;
- 2 вычислить  $x = x \bmod 65536$  и положить  $y = B_x$ .  
Выходом ультрагенератора UG в этом цикле будет  $C_y$ ;
- 3 заменить  $B_x$  на  $(B_x + B_{x-573} + B_{x-2604}) \bmod 16777216$ ;
- 4 заменить  $C_y$  на  $(C_y + C_{y-249} + C_{y-16774}) \bmod 256$ .

Индексы вычисляются по модулю 65536 или 16777216. В величинах запаздываний 103, 1071, ..., 16774 нет ничего специального. Я не проверял, порождают ли эти индексы особенно длинные периоды. Когда начальные массивы так велики, даже вырожденные периоды будут очень длинными. Можно использовать любые комбинирующие функции из раздела 13.1, например **madd**, или суммирование с запаздыванием, описанное в разделе 13.14.1.

Для обновления этих случайных чисел обоих методов из раздела 13.14 по отдельности недостаточно, если у противника имеется ультракомпьютер, но их можно объединить и таким образом получить стойкую функцию обновления. При каждом обновлении понадобится новый случайный массив R, содержащий 65536 или больше 24-битовых целых. Обозначим  $L_A, L_B, L_C, L_R$  длины массивов A, B, C, R.

**Шаг 1.** Объединить R с A, B и C.

Заменить  $A_n$  на  $(A_n + R_n) \bmod 16777216$  для  $n = 1, 2, 3, \dots, L_A$ .

Заменить  $B_n$  на  $(B_n + R_n) \bmod 16777216$  для  $n = 1, 2, 3, \dots, L_B$ .

Заменить  $C_{an}$  на  $(C_{an} + R_n) \bmod 256$  для  $n = 1, 2, 3, \dots, L_R$ .

Здесь  $a = \lfloor L_C/L_R \rfloor - 1$ . Здесь  $\lfloor LC/LR \rfloor$  (читается «целая часть» или «пол»  $L_C/L_R$ ) обозначает наибольшее целое число, не превосходящее  $L_C/L_R$ . Например,  $\lfloor 8/3 \rfloor = 2$ , а  $\lfloor 9/3 \rfloor = 3$ . Смысл использования  $C_{an}$  вместо  $C_n$  – равномерно «размазать» байты  $R$  по массиву  $C$ .

#### Шаг 2. Распространить изменения.

Заменить  $A_n$  на  $(A_n + A_{n-229} + A_{n-6141}) \bmod 16777216$  для  $n = 1, 2, 3, \dots, LA$ .

Заменить  $B_n$  на  $(B_n + B_{n-503} + B_{n-3829}) \bmod 16777216$  для  $n = 1, 2, 3, \dots, LB$ .

Заменить  $C_n$  на  $(C_n + C_{n-754} + C_{n-25887}) \bmod 256$  для  $n = 1, 2, 3, \dots, LC$ .

Эти 2 шага следует повторить 3 или более раз. Как обычно, индексы заворачивают по достижении конца.

Кстати говоря, не требуется, чтобы длина массива  $C$  была степенью 2.  $L_C$ , к примеру, могла бы быть равна 77777777, и тогда массивы  $A$ ,  $B$  и  $R$  должны были бы содержать целые по модулю 77777777, а в вычислениях выше модуль 16777216 следовало бы заменить на 77777777. Единственные ограничения на величину  $L_C$  – сколько памяти вы готовы выделить и практические проблемы с распределением такого большого ключа.

Оба метода, подстановку и генерирование случайных чисел, можно объединить и получить тем самым сколько угодно блочных и потоковых шифров, которые устоят против ультракомпьютеров. В следующих двух разделах приведено по одному примеру шифра каждого типа.

### 18.12.3 Ультраподстановочный шифр US-A

При написании этого раздела я испытывал сильное искушение задать гигантский размер блока, скажем 65 536 или даже 16 777 216. Но из того, что криптография должна измениться в эру ультракомпьютеров, еще не следует, что изменятся и типы сообщений. Сообщения длиной менее 100 символов по-прежнему будут в ходу, и было бы крайне неэффективно дополнять их до размера 65 536 или более байт.

Назовем пример ультраподстановочного шифра US-A. Этот шифр работает с блоками длиной 32 байта, или 256 бит. Каждый блок можно рассматривать либо как 32 отдельных байта, либо как битовый массив  $16 \times 16$ . В шифре US-A 15 раундов, состоящих из трех этапов: подстановка, перестановка строк и транспонирование массива. По завершении всех раундов выполняется заключительный шаг подстановки.

На всех 16 шагах подстановки используется таблица подстановки Tab24, в которой должно храниться 16 бит ключа для каждого символа, а всего  $16 \times 32 = 512$  бит на раунд или 8192 бита для 15 раундов плюс заключительная подстановка. Вторым этапом каждого из

15 раундов является перестановка каждой строки. Это может быть простой циклический сдвиг строки, требующий всего 4 бит на строку, т. е. 64 бита на раунд, а всего 960 бит.

Более стойкий вариант перестановки битов – таблица перестановок, содержащая, скажем, 256 различных перестановок, например перестановок по ключу (раздел 7.6). Каждая строка матрицы  $16 \times 16$  переставляется отдельно. Каждая перестановка строки определяется 16 шестнадцатеричными цифрами. Например, цифры **5A3F1E940B-2D68C7** означают, что первый бит перемещается в позицию 5, второй бит – в позицию A, т. е. 10, и т. д. Перестановка каждой строки выбирается из таблицы с помощью 8-битового ключа, что требует  $8 \times 16 = 128$  бит на раунд или 1920 бит на все 15 раундов.

Третий этап каждого раунда – транспонирование битового массива, т. е. перестановка бита в позиции  $(i, j)$  с битом в позиции  $(j, i)$ . Это описано в разделе 11.7, а быстрый метод транспонирования массива – в разделе 11.2.3. На этом этапе ключ не нужен.

Соберем все вместе. Шифр US-A требует 8192 бита для ключей подстановки и, скажем, 1920 бит для перестановок с ключом – всего 10 112 бит ключа. Это гораздо меньше, чем 65 536 бит, необходимых, чтобы противостоять ультракомпьютеру. Но не пугайтесь. Не забывайте, что для подстановки у нас есть таблица алфавитов Tab24, использующая 16 777 216 бит ключа для перемешивания своих 65 536 строк, не говоря уже о тех битах, которые использовались для перемешивания таблицы перестановки.

Ну, а чтобы еще укрепить шифр, я рекомендую использовать сцепление блоков в режиме **PP** (открытый текст с открытым текстом), описанном в разделе 11.10.

### 18.12.4 Ультрапотоковый шифр US-B

Подстановку Tab24, описанную в разделе 18.11.1, и генератор псевдослучайных чисел из раздела 18.11.2 можно объединить, получив тем самым невероятно стойкий шифр, который мы назовем *US-B*. В этом шифре имеется предварительный шаг, который выполняется до шифрования и призван придать открытому тексту случайный вид. Обозначим само сообщение  $M$ , а его длину  $L_M$ . Предварительный шаг таков:

$M_n$  заменяется на  $(M_n + 19M_{n-1} + 7M_{n-16}) \bmod 256$  для  $n = 1, 2, 3, \dots, L_{M+16}$ .

Дополнительные 16 итераций с заворачиванием нужны для того, чтобы перемешать первые 16 символов сообщения. Этот шаг не увеличивает криптостойкость, но мешает Эмили понять, что она нашла правильный ключ.

Каждый 16-битовый ключ  $K_n$  генерируется из двух последовательных выходных байтов генератора случайных чисел,  $x$  и  $y$ , по формуле  $256x + y$ . (Или можно хранить в массиве  $S$  16-битовые целые, но за

это придется расплачиваться удвоением объема потребной памяти.) Ключ  $K_n$  используется для шифрования символа сообщения  $M_n$  в таблице подстановки  $Tab24$  символом  $Tab24(K_n, M_n)$ . Иначе говоря, замена  $M_n$  берется из строки  $K_n$  таблицы алфавитов.

Вы, наверное, распознали в этом шифре многоалфавитный шифр общего вида с большой таблицей алфавитов и случайным ключом. Напомню, что по-французски многоалфавитный шифр называется *Le Chiffre Indéchiffrable*<sup>1</sup>. Благодаря ультрагенератору UG многоалфавитный шифр US-B наконец-то стал и вправду недешифрируемым, даже в эпоху ультракомпьютеров. Мы пришли к Невскрываемой Криптографии.

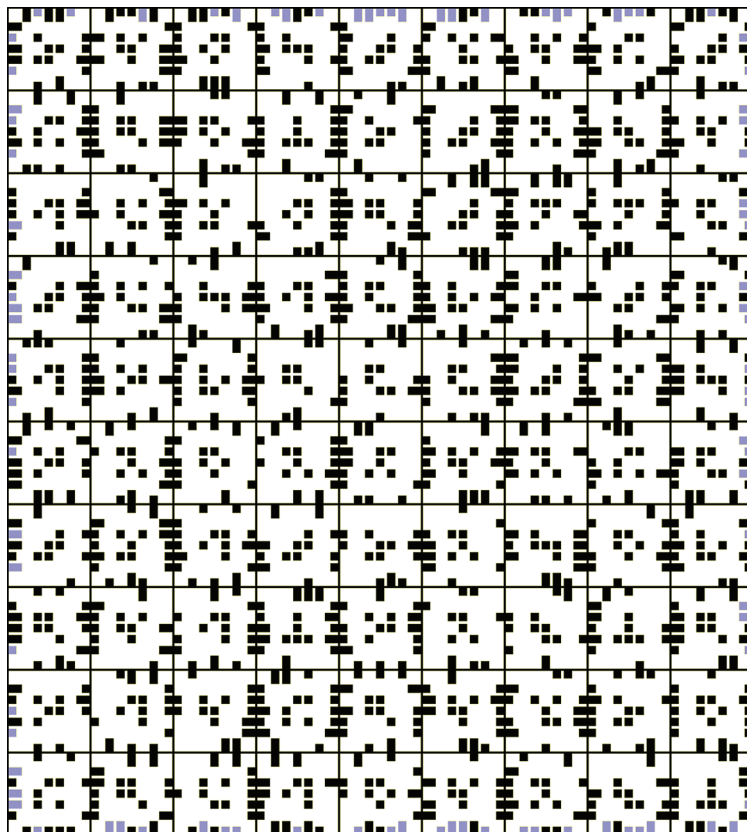
---

<sup>1</sup> Шифр, не поддающийся дешифрированию. – Прим. перев.

# Развлечения

---

На рисунке ниже изображены четыре разных шифра, от S1 до S4. Каждый шифр – простая одноалфавитная подстановка. Ваша задача – определить тип, например код Морзе, и затем вскрыть шифр. Все открытые тексты написаны на стандартном английском языке, заглавными буквами, без пробелов и знаков препинания. Длина текстов от 75 до 90 букв. Все тексты начинаются в левом верхнем углу. Первые три шифра читаются слева направо по строкам. Последний шифр читается по часовой стрелке вдоль границы.



Все методы описаны в этой книге. Отличается только визуальное представление. Вам предстоит определить, какие факторы важны в каждом шифре, например: высота, ширина, положение или цвет. Можете отправлять свои ответы для получения признания, следуя инструкциям, спрятанным где-то в книге.

Вот еще несколько головоломок, в которых используются шифры, наиболее популярные среди любителей математических развлечений. Все тексты написаны на стандартном английском языке, встречается несколько имен собственных. По традиции, буквы объединены в группы по пять, не обращая внимания на точки. Если хотите еще кучу такого рода шифров, вступайте в Американскую ассоциацию криптограмм по адресу [cryptogram.org](http://cryptogram.org). Можете отправлять свои ответы для получения признания, следуя инструкциям, изложенным в открытом виде, но спрятанным где-то в книге. Дополнительные криптограммы можно найти по адресу [www.contestcen.com/crypt.htm](http://www.contestcen.com/crypt.htm).

### F1: ШИФР БЕЛАСО (РАЗДЕЛ 5.8.1)

HZRRJ GHEEM ZZHXU AYNLJ GYXCV LRRDL UMIEE PHMET PIPWA ELZOC  
BNPBK SHSLV GQVLP AIVBM LVFLB RLOHX BNZUH MATSM LHVTI ZRH

### F2: ШИФР ВИЖЕНЕРА (РАЗДЕЛ 5.8.2)

MGGAP AGXCD IFDAZ GZFSH OODAZ HGYBS HZNEB KBQAZ BBCGF ADRDZ  
KDVXZ LFTYZ ZGYVW JVXUH MBYBN TRLRZ HGWJZ IJAAI EGUOD ADWAQ  
ADAGS ADA

### F3: СТОЛБЦОВАЯ ПЕРЕСТАНОВКА (РАЗДЕЛ 7.2)

DUSEL CQTNT ACNLH HLTME AEOEO RLRES TEHNT TAERW AEGLR EDAEE  
TEYEH UBSHE OVAAE HRDCI INHWE SFTEA LYWIR TIIOT BITRD AEBRT  
NATTT ENLRU HDHTE AE

### F4: ШИФР ПЛЕЙФЕРА (РАЗДЕЛ 9.2)

EIWDU WJHYL BHFBK NWVKY TKHDE WVBXF GKTDV XXHIY DHNZT ZHDAR  
HAYEG SLHXB CIDPH YEIWP

### F5: ШИФР VIGID (РАЗДЕЛ 9.6)

Размер блока 7, тема – садоводство.

SZSAPNF RBHBKNV OAAVCBI LFIUUD IRFTPNZ SBLANBA GEPNEAX  
ONNAMLB GFRMEUV LIMASUT BFUIEZM CBBRHTI LHROSUV ALSEOET  
FHWBTXL UWRBIKL TUHTIEI IFIGOKP

### F6: ОДНОРАЗОВЫЙ БЛОКНОТ (ГЛАВА 14)

Для каждой буквы сообщения генерируется случайное число. Если число четное, то к букве прибавляется X по модулю 26, иначе прибавляется Y по модулю 26.



UVTUQ JYRMV GJVSI FTZEL YFIJY JVAVI JZETV YMJKF IGFEA RIZVK  
DNRXK ZIJKI TRJXY NHRKV URSFL YWNKK PWNAV YLEUI JIDVF WXFLF

Конечно, одноразовый блокнот – не любительский шифр. Я включил его в главу «Развлечения», дабы показать, что некоторые шифры на основе одноразового блокнота на практике можно вскрыть. Сумеете ли вы найти значения X и Y, не перебирая все 676 комбинаций?

### F7: Многоалфавитный шифр общего вида (раздел 5.9.3)

Чтобы было повеселее, в следующем примере взяты символы из нескольких языков общим числом больше 26. Однако каждый алфавит состоит из 26 букв и сообщение написано на стандартном английском языке.

У у з ф F 9 Δ 3 Σ η Н Δ Ъ Ё д F Ё з 0 X U F y Ё S F P y W a F a Ё  
Ш Э Ф P Ъ Ё h Q P 5 a T 9 F Ф 0 Σ 9 F Z Ё d H Ф y W Z y Ф 9 5 P 5  
P 5 P W y Σ X 9 d 5 d Ё 5 F h Σ Ё 0 z 5 P Э W a Z Δ 0 Ё H Ф P y Ф  
W 9 d U Ё z z P 3 5 F a P 5 0 W Ё F 5 F η y Ф S Ё h H H d 5 H a P  
5 0 5 0 H d Ф 9 9 Δ d Σ 0 z 3 z 0 W F d Ё 9 d F Σ Ё z W 9 H 0 W 0

# Задачи

---

Эти криптограммы представлены в виде задач для читателей. Метод не раскрывается. Вы сами должны определить метод и решить криптограмму. Для решительного, мотивированного и знающего любителя приведенных сведений достаточно. Во всех случаях сообщение написано на английском языке. Текст не содержит грамматических и других ошибок. Я не прикладывал никаких специальных усилий, чтобы исказить стандартные для английского языка частоты букв и контактов.

Все шифры одношаговые. Методы не комбинируются, в частности отсутствуют сочетания подстановки с перестановкой. Все эти шифры имеют оценку 3.

## С1: задача 1

Этот шифр вскрывается с помощью карандаша и бумаги. Открытый текст содержит 250 заглавных букв без пробелов между словами и знаком препинания.

```
LIUJE IETOJ TUUIL PICLO RNETH SEVWP GRHJS OIMTO ETEPI CETBE  
OOKIP AHOSA GRHJO AHETB AUTTI RAHTV NENAH TTUTG ICOSI YHNFN  
ENAAС OGNET JTGUA FMEE EHITR OAHET SHHNW TJTOE EGRHJ NETHT  
GUTTO HTTRP HNCON OIEIO AHETB ALCOW TJSEV CMPIC SIYOF SPMHN  
VNSWE AONES TSSEV OSAMY ITDPI CSLAU UIYFS PMHNI OOSCA RHTRR
```

## С2: задача 2

Этот шифр можно решить вручную, но для работы с шестнадцатеричными числами проще прибегнуть к помощи компьютера. Открытый текст состоит из 200 символов в разных регистрах с пробелами и знаками препинания.

```

4CB1BAB35A 68C7BAA966 6947C49FA6 F509C4B144 4F48864F03
F3C68DD25E 4F468653A6 F509C4B144 4F48864F04 8F6B537F01
F06829B286 8974E37F12 6F87BDA94F 8D3E24DCF1 F3F8E64D66
02F9C06553 8879B6CF1C 8969B9B286 F529BBAA46 F247B014FE
8975CACF1B 8968E32D41 8969B6D246 0147CDFF12 8D35C9A94D
4F55C4FCE4 6AEA864F0E B24696D0ED 0E4691D0ED 6C99536B01
F110E33D5D 6C49BAAB42 6AF886495A 4F7424FCE1 8D2AE364ED
F0C7BF2955 8BEBBC4CDC2 8954E3295B 4F4FBAB342 8879BAE64D

```

### С3: задача 3

Этот шифр допускает ручное шифрование и дешифрирование. Открытый текст состоит из 180 заглавных букв без пробелов и знаков препинания.

```

ZNQXI VAKSG UZONV ALQPR EMYNN WBXXS NPPYB DQPIP KSYEC RXKVE
CGQZI NHIRA NLTSO VGRXH NQVBU EBORK IWOPK SWZIJ EMJTA YNVWD
AUMLP VZIQM XZRMJ CXJKM OMONN UXIPL JWESX CRMJT QRKBL TQVBL
TACSA GUPKC QKIIU LTJFT QPZFB KVBUE V

```



группами битов фиксированной либо переменной длины, символами кода Морзе, цифрами в любой системе счисления или элементами математического кольца. Все символы можно переставлять и перегруппировывать. Группы можно снова подставлять, сжимать, умножать на целые числа или матрицы, преобразовать в другую систему счисления или сцеплять. Части одних групп можно использовать для шифрования частей других групп.

**Поставщики криптографических услуг.** Компании, предоставляющие услуги зашифрованной связи, обычно применяют собственные алгоритмы. Они могут применять любые методы, рекомендованные выше разработчикам, но должны гарантировать, что шифр отвечает всем критериям, сформулированным в главе 12. Должны использоваться большие блоки и длинные ключи. Шифр должен быть сильно нелинейным, иметь хорошую диффузию и насыщение.

Если поставщик услуг пользуется стандартным алгоритмом, например 3DES или AES, то до и после стандартного шифра должна применяться секретная подстановка или перестановка с ключом.

**Банки.** К банкам и финансовым компаниям предъявляется требование использовать AES для любых взаимодействий, чтобы они могли обмениваться информацией между собой, а также с Федеральной резервной системой, Комиссией по ценным бумагам и биржам, Налоговым управлением и другими государственными агентствами. Банки также широко используют криптографию с открытым ключом для выработки ключей шифрования, а также аутентификации и верификации.

**Вооруженные силы и дипломатические службы.** Вооруженные силы США и госдепартамент обязаны следовать рекомендациям АНБ и использовать 256-битовый AES. Это требование закона. Оно соблюдается на ПК, ноутбуках и даже смартфонах с помощью аппаратной или программной реализации AES. Однако вооруженные силы и разведка действуют в таких местах и условиях, где компьютеры и сотовые телефоны могут не работать, а телефон, оснащенный AES, может вызывать подозрения или считаться незаконным. Во многих странах владение любым криптографическим оборудованием, литературой или рабочим продуктом – преступление. Кроме того, иностранные армии и дипломатические службы могут не доверять никакому оборудованию или программному обеспечению для AES, потому что они могут исходить только от АНБ или поставщиков, регулируемых АНБ.

По этим причинам у армии и разведки имеются также резервные коды и шифры на случай невозможности использовать электронное криптографическое оборудование. Для применения в боевой обстановке подходят такие ручные шифры, как диагональный Bifid, TwoSquare+1, Two Square с рябью и Playfair TwoSquare. Еще одна идея – использовать Bifid или Two Square, сопровождаемый кусочной перестановкой.

**Большие файлы.** Для очень больших файлов гораздо эффективнее использовать потоковый шифр, чем блочный. Нужно генерировать поток псевдослучайных чисел и объединять его с файлом данных, имитируя одноразовый блокнот. В качестве ГПСЧ можно использовать Xorshift, FRand или Gen5, а в качестве комбинирующей функции – **xors**, **adds** или **poly**. Или же можно использовать GenX для генерирования и объединения одновременно.

# Предметный указатель

---

## Символы

---

<<< циклический сдвиг влево, 257  
<<< циклический сдвиг вправо, 257, 264  
<< сдвиг влево, 256  
>> сдвиг вправо, 256  
 $|0\rangle$  квантовое базисное состояние 0, 346  
 $|\alpha|$  модуль комплексного числа, 347  
 $\wedge$  булев оператор И, 43  
 $\vee$  булев оператор ИЛИ, 43  
 $\oplus$  булев оператор ИСКЛЮЧАЮЩЕЕ  
ИЛИ, 43  
3Base шифр, 316  
- аддитивное обращение, 47, 317, 320  
! факториал, 43  
 $\lfloor x \rfloor$  пол  $x$ , 251  
 $\lceil x \rceil$  потолок  $x$ , 251

## A

---

add функция, 255, 269  
AddRoundKey, 198  
adds функция, 209, 211, 263  
ADFGVX шифр, 151  
AES (Advanced Encryption Standard), 37,  
196, 198, 352, 372  
and логическая функция, 43  
APL язык программирования, 251  
ASCII код, 68, 107, 318

## B

---

B-гладкие числа, 332  
Bicyclic 8×8 шифр, 160  
Bifid шифр, 148, 157, 188, 237, 367, 371  
    диагональный, 151  
    с сопряженной матрицей, 150  
BitBlock MA шифр, 168  
BitBlock SA шифр, 168  
BitCycle Substitution шифр, 161  
Blackout перестановочный шифр, 116  
Brick Wall шифр, 301  
Butthead шифр, 301

## C

---

CG5 генератор случайных чисел, 265  
CNOT функция, 348  
Cryptomenysis Patefacta, 111  
CSP-488 шифровальное устройство, 129  
CyGen, 265

## D

---

De compendis cifri, 36  
Deep Crack, 193  
De Occultis Literarum Notis, 34  
DES (Data Encryption Standard), 36, 39,  
190, 200



быстрая перестановка битов, 194  
Double DES, 192  
Triple DES (3DES), 193  
Diehard, комплект тестов на  
случайность, 256

---

**E**

Electronic Frontier Foundation, 193  
EmbedBits, 108

---

**F**

Four Square шифр, 146, 157  
FRand генератор случайных чисел, 257

---

**H**

Hash32, 275  
Hash128, 275  
HashPQ, 275

---

**I**

IBM, 36, 42, 125, 189, 259, 352

---

**L**

La cifra del Sig., 71  
Le Chiffre Indéchiffrable, 72, 365  
Lucifer шифр, 189  
LZ77, 178  
LZ78, 178  
LZW, 179

---

**M**

М-94 шифровальное устройство, 129  
madd функция, 239, 252, 255  
Mat36 шифр, 313  
Meld8 метод, 232  
MixColumns, 198  
MPM128 шифр, 311

---

**N**

not логическая функция, 43  
null-дополнение, 104  
null-символы, 96, 110, 114, 134, 140, 158,  
164, 177, 184, 195, 353

---

**O**

or логическая функция, 43

---

**P**

PickPrimes, 288  
poly функция, 209  
Post64 шифр, 174  
PostDL шифр, 176  
PostOv шифр, 174

---

**R**

Rijndael шифр, 198

---

**S**

S-блок, 189  
    конструирование, 232  
    с ключом, 236  
sadd функция, 260  
ShiftRows, 198  
SkipMix перемешивание алфавита, 68,  
232  
Steganographia, 30  
SubBytes, 198  
Summit компьютер, 93  
sxor функция, 239

---

**T**

The Military Cipher of Commandant  
Bazeries, 60  
Tiger шифр, 311  
Traité Élémentaire de Cryptographie  
(Деластель), 146

---

**X**

xor логическая функция, 43  
xor функция, 252, 255, 269  
xors комбинирующая функция, 209,  
211, 261, 274

---

**A**

Автоключ, 72, 85, 246, 371  
Агентство национальной безопасности  
(АНБ), 39

- Адаптивное кодирование, 178  
 Аддитивное обращение, 47  
 Адлеман Лен, 31  
 Алгоритм  
   больших и малых шагов, 329  
   восхождение на вершину, 356  
   высоких пиков, 81, 298  
   Гровера, 352  
   имитации отжига, 358  
   Касиски, 73  
   Лемпеля–Зива–Уэлча, 179  
   Мэсси–Омуры трехпроходный, 329, 339  
   наискорейшего спуска, 356  
   ро Полларда, 330  
   Сильвера–Полига–Хеллмана, 284  
   тест прироста, 124  
   трехпроходный протокол, 327  
   тысяча вершин, 357  
   цепной аддитивный генератор, 256  
   цепной генератор цифр, 56  
   BB84, 352  
   ResM, 305  
   ResMH, 307  
   ResMHL, 309  
   SkipMix, 68, 232, 234  
   Xorshift, 257  
 Алфавитный порядок, 101, 342  
 Альберти Леон Баттиста, 36  
 Антидиагональный, 151  
 Анцилла, 349  
 Ардженти семья, 67, 96  
 Арифметическое кодирование  
   адаптивное, 184  
   общие сведения, 181  
 Арифметическое шифрование, 183  
 Артина постоянная, 250  
 Асимметричная криптография, 32  
 Атака  
   с известным открытым текстом, 297  
   со встречей посередине, 193, 212  
   с подлогом, 37  
   с противником в середине, 37  
 Аутентификация, 37, 274, 372
- Б**
- База множителей, 332  
 Базери шифр типа 4, 59, 142, 201, 371  
 Базери Этьен, 59  
 Базисное состояние, 346  
 Базовый ключ, 271  
 Байт, 42  
 Байтовая конфигурация, 230  
 Баркер Уэйн Дж., 117  
 Беласо Джован Баттиста, 71  
 Беласо шифр, 155, 223, 353, 371  
   вскрытие, 78  
   описание, 71  
   таблица алфавитов, 85, 95  
 Беннет Чарльз Х., 352  
 Бесключевая криптография, 32  
 Биграмма, 52  
   подстановка, 100  
 Билинейное уравнение, 337  
 Битовая конфигурация, 230  
 Битовое фракционирование  
   описание, 158  
   Cyclic  $8 \times N$  шифр, 159  
 Бит определение, 41  
 Блочные шифры, 188  
   инволютивный шифр.  
   См. *Инволютивный шифр*  
   подстановки переменной длины, 204  
   подстановочно-перестановочная  
   сеть, 189  
   пульсирующие, 205  
   сцепление блоков. См. *Сцепление  
   блоков*  
   укрепление, 212  
   умножение матриц, 196  
   фиксированная подстановка  
   и подстановка с ключом, 200  
 AES, 198  
 DES, 191  
   3DES, 193  
   быстрая перестановка битов, 194  
   неполные блоки, 195  
   Double DES, 192  
 Бодо код, 278  
 Бодо Эмиль, 278  
 Большие простые числа  
   конструирование  $P$  и  $Q$ , 290  
   коэффициенты, 288  
   новый подход к построению, 286  
   секретные простые числа, 291  
   точный размер, 291  
   традиционный подход  
   к построению, 285

уменьшение количества  
проверок, 289  
ядро, 288

Браве Огюст, 220  
Бра-кет нотация, 347  
Брассар Жиль, 352  
Булев оператор, 43  
Бумажная полоска, 36, 119, 126, 130  
Буонафальче Августо, 36  
Бухгольц Вернер, 42  
Бэббидж Чарльз, 73

## В

Вайль Альфред, 54  
Вектор инициализации (IV), 56  
Вектор-столбец, 196  
Вектор сцепления, 208  
Вернама шифр, 278  
Вернам Гилберт Сэндфорд, 278  
Вероятное слово, 87, 130, 140, 149, 184  
Взаимно простые числа, 46  
Виженер Блез де, 72  
Внутренние отводы, 210  
Возвращение ключей в оборот, 281  
Волочение слова, 87  
Восхождение на вершину, 356  
Восьмироторная машина, 92  
Вскрытие кода, 28  
Вскрытый шифр, 29  
Вычет, 46

## Г

Гамильтон Уильям Роуэн, 322  
Гамма, 33  
Гауссово целое число, 321  
Гейденберг Иоганн, 30  
Генератор  
    выбора, 263  
    псевдослучайных чисел (ГПСЧ), 248  
    случайных чисел, 54, 249  
        вектор состояния, 249  
        вихрь Мерсенна, 259  
        комбинирование, 264  
        линейное суммирование  
        с запаздыванием, 268  
        линейный конгруэнтный, 253  
        мультипликативный  
        конгруэнтный, 249

наложение изображений, 269  
начальное значение, 56  
обновление случайных байтов, 270  
оценивание периода, 261  
регистр сдвига с линейной  
    обратной связью, 259  
сдвиговый XOR-генератор, 256  
синхронизированные гаммы, 272  
укрепление, 263  
ультракомпьютер, 362  
функции хеширования, 273  
цепной аддитивный, 256  
цепной генератор цифр, 56  
цепной XOR-генератор, 254  
FRand, 257

Гибридная нелинейность, 232  
Гистограмма, 70  
Гладкие числа, 285  
Градиентный метод, 356  
Гражданская война в США, 116  
Гровер Лов Кумар, 352  
Грэй Фрэнк, 354  
Грэя код, 354  
Гутенберга проект, 65

## Д

Даймен Йоан, 198  
Двоичная система счисления, 45  
Двойная столбцовая перестановка, 117  
Двойное шифрование, 95  
Де Брёйна функция, 333  
Де Брёйн Николас Говерт, 333  
Декогеренция, 348  
Делатель Феликс-Мари, 142, 146, 148,  
    152  
Делитель, 46  
Дешифрирование, 28  
Джефферсона цилиндрический  
    шифр, 128  
        вскрытие при наличии известных  
        слов, 131  
        вскрытие при наличии только  
        шифртекста, 132  
Джефферсон Томас, 35  
Джокер, код, 343  
Дикмана функция, 333  
Дикман Карл, 333  
Дирак Пол А. М., 347

Дискретного логарифмирования  
задача, 284, 329  
коллизии, 331  
логарифмы, 330  
оценки, 333  
степени простых чисел, 330  
факторизация, 331  
Диффи Уитфилд, 283  
Диффи–Хеллмана алгоритм  
распределения ключей, 283  
    построение больших простых чисел,  
    новый подход, 286  
    построение больших простых чисел,  
    традиционный подход, 285  
Диффи–Хеллмана задача, 284  
Диффузия, 236, 372  
    индекс, 239  
Длинные ключи, 215  
Добавление null-битов, 103  
Добелл А. Р., 253  
Дополнение, 104

## Е

---

Единичная матрица, 297

## З

---

Закладка, 227  
Замаскированное сообщение, 30  
Запас безопасности, 29, 173  
Запас ключей, 280  
Запутанность, 348  
Зацепка, 87, 283  
Зив Якоб, 178  
Зигзаговый шифр, 35  
Значения истинности, 42  
Зубчатая конфигурация, 301

## И

---

Иверсон Кеннет, 251  
Избыточный ключ, 216  
Имитация отжига, 359  
    квантовая, 360  
Инвариант, 202  
Инволютивный шифр, 201  
    инволютивная многоалфавитная  
    подстановка, 202

    инволютивная перестановка, 202  
    инволютивная подстановка, 202  
    инволютивный блочный шифр, 203  
    Poly Triple Flip, 204  
Индекс совпадения, 76  
Индикаторы, 282  
Институт Куранта, 171  
Инфраструктура открытых ключей, 37  
ИСКЛЮЧАЮЩЕЕ ИЛИ оператор, 43  
Истинно случайные числа, 55, 268

## К

---

Как Субхаш, 351  
Камуфлирование, 229  
Кандела Розарио, 60  
Кардано Джироламо, 85  
Кармайкла числа, 285  
Кармайкл Роберт, 285  
Карно карта, 354  
Карно Морис, 354  
Карта отличий, 102  
Касиски метод, 73  
Квантовая телепортация, 349  
Квантовая факторизация, 360  
Квантовый бит, 346  
Квантовый компьютер, 346  
    алгоритм Гровера, 352  
    запутанность, 348  
    измерение, 350  
    исправление ошибок, 349  
    квантовое распределение  
    ключей, 352  
    квантовый трехэтапный  
    протокол, 351  
    минимизация, 356  
    суперпозиция, 347  
    ультракомпьютер.  
    См. *Ультракомпьютеры*  
    уравнения, 353  
Квантовый поиск в файле, 352  
Кватернион, 322, 337  
Керкгоффс Огюст, 61  
Кирпичная стена, конфигурация, 301  
Класс вычетов, 46  
Ключ, 28  
Ключевая фраза, 28  
Ключевое слово, 28  
Ключевой текст, 86

- Ключ  
  объединения, 106  
  символа, 33  
  сообщения, 33  
Книга абака, 44  
Код, 28  
Коды, исправляющие ошибки, 350  
Коды, обнаруживающие ошибки, 350  
Коллизия, 331  
Кольцо, 317  
  гауссовых целых чисел, 321  
  кватернионов, 322  
  коммутативное, 317  
  матрицы над кольцом, 318  
  некоммутативное, 337  
  построение, 319  
  R8, 323  
  R13, 318  
Комбинирование генераторов, 264  
Комбинированный ключ, 281  
Комбинирующие функции, 209, 211, 247, 255  
Коммутативное семейство матриц, 334  
Комплексное сопряжение, 347  
Комплексное число, 317, 347  
Композиция шифров, 58  
Конденсированная линейность, 231  
Контрольная сумма, 39  
Конфузия, 217  
  включение закладки, 227  
  гибридная нелинейность, 232  
  конденсированная линейность, 231  
  конструирование S-блока, 232  
  коэффициент корреляции, 219  
  линейность по основанию 26, 223  
  линейность по основанию 256, 226  
  S-блоки с ключом, 236  
Котельников Владимир, 277  
Коэффициент корреляции, 219  
Криптоанализ, 28  
Криптографический  
инструментарий, 48  
  генератор случайных чисел, 54  
  комбинирование, 58  
  перестановка, 52  
  подстановка, 50  
  система оценивания шифров, 49  
  фракционирование, 53  
Криптография, 27  
  блочные и потоковые шифры, 33  
  виды, 30  
  инструментарий, 48  
  контрмеры, 94  
  механические и цифровые шифры, 33  
  невскрываемые шифры, 28  
  подстановочные шифры, 61  
  предварительные сведения, 41  
  причины для построения  
  собственного шифра, 38  
  симметричная и асимметричная, 32  
  с секретным ключом, 37  
  терминология, 27  
Криптологическая бомба, 283  
Криптология, 28  
Криптостойкость, 248  
Кубит, 346  
Кук Уильям, 137  
Кульбак Соломон, 117  
Кунце Вернер, 277  
Кусочное обращение, 59
- Л**
- Левин Джек, 296  
Леводиагональный, 151  
Лемпель Абрахам, 178  
Лемпеля–Зива алгоритм сжатия  
текста, 178  
Лемпеля–Зива подстановочный  
шифр, 181  
Лемпеля–Зива–Уэлча метод, 179  
Лео (Леопольд Самуэль) Маркс, 277  
Лесенка, 308  
Либри Гильельмо, 45  
Линейная подстановка, 224, 228  
Линейное преобразование, 224  
Линейное сравнение, 305  
Линейное суммирование  
с запаздыванием, 268  
Линейное уравнение, 69, 218, 260, 300,  
334, 340, 353  
Линейность, 218  
  по основанию 26, 223  
  по основанию 256, 226  
Линейный конгруэнтный  
генератор, 253  
Линейный шифр, 218  
Логический оператор, 43

Логическое значение, 42

Ложный путь, 30

Люка Эдуард, 286

## М

Мажоритарная функция, 260, 269

Маркер конца сообщения, 106

Марсалья Джордж, 256

Масштабирование (статистика), 221

Матрица, 196

Матрица двойной ширины, 293

Матрица перестановки, 296

Матричные методы

кольца. См. *Кольцо*

нахождение обратимых матриц, 323

решение линейных сравнений, 305

лесенка, 308

правило половины, 306

приведение сравнения, 305

цепные дроби, 309

Матричный трехпроходный  
протокол, 333

атаки, 336

двусторонний, 340

коммутативное семейство

матриц, 334

максимальный порядок, 335

мультипликативный порядок, 334

некоммутативное кольцо, 337

решение билинейных уравнений, 337

скорость работы, 339

слабые элементы, 339

Мацумото Макото, 259

Менеджер ключевых слов, 216

Меркл Ральф, 284

Мерсенна вихрь (ГПСЧ), 259

Мерсенн Марин, 259

Метод перекрытия для коротких  
блоков, 196

Метод роя частиц, 357

Миллера–Рабина тест, 285

Миллер Гэри Л., 285

Миллер Фрэнк, 276

Минимизация, 356

имитация отжига, 358

метод восхождения на вершину, 356

метод тысячи вершин, 357

Многоалфавитная подстановка, 70

вскрытие, 83

инволютивная, 202

индекс совпадения, 76

метод Касиски, 72

шифр Беласо, 71

шифр Виженера, 81

Многоалфавитный шифр общего  
вида, 206, 361, 368

Множественные анаграммы, 126

Модульная арифметика, 46

Моном-биномный шифр, 165

Морзе код, 54, 72, 164

Морзе Сэмюэль Ф. Б., 137

Мультиплексный шифр, 128

Мультипликативное обращение, 47

Мультипликативный конгруэнтный  
генератор, 249

Мэсси Джеймс, 329

Мюррей Дональд, 278

Мюррея–Бодо код, 278

## Н

Наибольший общий делитель, 47

Наименьшее общее кратное, 95

Наложение изображений, 269

Насыщение, 240

вычисление  $S_1$ , 241

вычисление  $S_2$ , 242

индекс, 240

коэффициенты, 241

Натуральный, 149

Научная нотация, 46

Национальное бюро стандартов, 37, 190

Национальный институт стандартов и  
технологий (NIST), 37

Начальное значение, 56

Начальное состояние, 249

Небель Фриц, 151

Невидимое сообщение, 30

Невскрываемость, определение, 28

Независимые переменные, 355

Нелинейная подстановка, 234, 260, 264,  
271, 299

Непериодический ключ, 33

Неподвижная точка, 202

Непредсказуемый ключ, 248

Нишимура Такудзи, 259

Номенклатор, 70

Нормировка, 221

## О

Облачное хранилище, 38  
Обратимая матрица, 313, 323  
Обратная матрица, 297  
Обратный ключ, 31  
Обращение матрицы, 293  
Объединение нескольких сообщений, 105  
Одноалфавитная подстановка, 50, 62, 366  
Одноразовый блокнот, 276  
Оле Имануил, 73  
Омофоническая подстановка, 50  
Омофоны, 50  
Омура Джим К., 329  
Отвод, 210  
Открытый ключ, 31  
Открытый текст, 27  
Охавер М. Э., 53, 163, 296

## П

Первообразный корень, 250  
Перебор делителей, 286  
Перемешанный алфавит, 72, 81  
Перемешивание алфавита, 67  
Переопределенная система уравнений, 341  
Перестановка, 109  
    двойная столбцовая, 117  
    деление пополам, 125  
    маршрутная, 109  
    множественные анаграммы, 126  
    селекторная, 121  
    с ключом, 122  
    слов, 116  
    со случайными числами, 120  
    столбцовая, 111  
    с циклическим сдвигом, 118  
    Cysquare, 115  
Перестановка битов, 191, 229, 364  
    быстрая, 194  
Перешифрование, 58  
Период ГПСЧ, 250, 261, 264  
Периодический ключ, 107  
Период случайной последовательности, 55

Персональный ключ, 32  
Перфокарта, 35  
Пиксель, 101  
Пирсона коэффициент корреляции, 80, 220  
Пирсон Карл, 80, 220  
Плейфера шифр, 137  
    вскрытие, 139  
    укрепление, 140  
Плейфер Бэрон Лайон, 137  
Поверхностный код, 350  
Подстановочно-перестановочная сеть, 189  
Подстановочные шифры, 50  
    автоключ, 85  
    бегущий ключ, 86  
    индекс совпадения, 76  
    метод Касиски, 72  
    многоалфавитная подстановка, 70  
    моделирование роторных машин, 88  
    номенклаторы, 70  
    перемешивание алфавита, 67  
    простая подстановка, 62  
    шифр Беласо, 71  
    шифр Виженера, 81  
Подходящее начальное значение, 57, 257  
Показатель степени, 45  
Полибий, 53  
Полибия квадрат, 136  
    квадраты  $6 \times 6$ , 152  
    шифр Плейфера, 137  
    шифр Bifid, 148  
    шифр Four Square, 146  
    шифр Three Square, 143  
    шифр Two Square, 142  
Полиг Стивен, 284  
Поллард Джон, 329  
Полнопериодный генератор, 255  
Полунатуральный, 149  
Порождающая матрица, 334  
Последовательная линейность, 226  
Последовательной закладки метод, 228  
Поста таг-системы, 171  
    короткие и длинные  
    перемещения, 177  
    несколько алфавитов, 176  
    таги одинаковой длины, 172  
    таги разной длины, 174



Постшифрование, 212  
 Пост Эмиль Леон, 171  
 Потокковые шифры, 246, 363  
     Вихрь Мерсенна, 259  
     генерирование случайных чисел, 248  
     истинно случайные числа, 268  
     комбинирование генераторов, 264  
     комбинирующие функции, 247  
     линейный конгруэнтный генератор, 253  
     мультипликативный конгруэнтный генератор, 249  
     оценивание периода, 261  
     регистр сдвига с линейной обратной связью, 259  
     сдвиговый XOR-генератор, 256  
     синхронизированные гаммы, 272  
     укрепление генератора, 263  
     цепной аддитивный генератор, 256  
     цепной XOR-генератор, 254  
     FRand, 257  
 Правило половины, 306  
 Предшифрование, 212  
 Прерванный ключ, 96  
 Префиксное свойство, 51  
 Прозрачность, 143, 145  
 Производный ключ, 271  
 Пропуски, 68  
 Простая (одноалфавитная) подстановка, 62  
 Простые числа, 46  
     безопасные, 285  
     новый способ построения, 286  
     традиционный способ построения, 285  
 Протокол, 32, 327  
 Прямоугольная сетка букв, 156  
 Псевдослучайные числа, 55, 248  
 Пятьдесят на пятьдесят свойство, 214

## Р

Рабин Майкл О., 285  
 Развертка ключа, 192, 198  
 Распределение  
     гласных, 62  
     согласных, 62  
     частот, 70  
 Расширенный алгоритм Евклида, 305

Раундовый ключ, 198, 239  
 Регистр сдвига с линейной обратной связью (РСЛОС), 259  
 Режевски Мариан, 283  
 Режим сцепления  
     СС, 209  
     СК, 211  
     СР, 209  
     ИК, 211  
     РС, 209  
     РК, 211  
     РР, 209, 364  
 Ривест Рональд, 31  
 Ротор, 36, 89  
 Роторная машина, 280  
     моделирование, 88  
 Рэйман Винсент, 198

## С

Самообратный шифр, 201  
 Сандвича техника, 212  
 Сверточный код, 350  
 Свидетель простоты, 285  
 Секретный ключ, 31, 37, 48, 272  
 Селекторная перестановка, 121  
 Сен-Сира линейка, 72  
 Сжатие текста, 51, 178  
     адаптивное арифметическое кодирование, 184  
     арифметическое кодирование, 181  
     метод Лемпеля–Зива, 178  
     метод Лемпеля–Зива–Уэлча, 179  
 Сильвер Роланд, 284  
 Скалярное произведение, 197, 222, 348  
 Скитала, 34  
 Скользящее окно, 182  
 Скрытое сообщение, 30  
 Словарь, 179  
 Сложение без переноса, 58  
 Служба разведки сигналов, 117  
 Случайное блуждание, 331  
 Случайные числа, 234, 248, 263, 270, 280, 362  
 Соккрытие, 229  
 Соккрытие сообщений  
     в изображениях, 101  
 Софи Жермен простые числа, 285  
 Сравнимость по модулю, 46

Стеганография, 30  
Стейджер Энсон, 116  
Столбцовая перестановка, 111, 151, 158, 191, 204, 353, 367  
    двойная, 117  
    слов, 116  
    с циклическим сдвигом, 118  
Столбцовое перемешивание, 67  
Суперкомпьютер, 49, 92, 333  
Суперпозиция, 347  
Суффиксное свойство, 172  
Схема контактов, 63, 83  
Сцепление блоков, 208  
    блоки переменной длины, 211  
    внутренние отводы, 210  
    зашифрованное сцепление, 210  
    многоалфавитное сцепление, 209  
    неполные блоки, 211  
    режим сцепления, 211  
    сцепление ключей, 211  
    сцепление с запаздыванием, 210

---

## Т

Таблица  
    алфавитов, 71  
    контактов букв, 65  
    подстановки, 90, 154  
Тайнопись, 27, 35  
Такерман Брайант, 259  
Тест на простоту по корням, 286  
Тест прироста для перестановок, 124  
Тожественный шифр, 204  
Транспонирование квадратной матрицы, 201  
Транспонирование матрицы, 195, 204  
Трехпроходный протокол, 326  
    метод Мэсси–Омуры, 329  
    метод Шамира, 328  
Триггерное событие, 97  
Триграмма, 52  
Тритемий Иоганн, 30, 71  
Тьюринг Алан, 171, 283

---

## У

Уитстона криптограф, 137  
Уитстон Чарльз, 72, 137  
Ультракомпьютер, 360

    подстановка, 361  
    случайные числа, 362  
    ультрагенератор UG, 362  
    ультраподстановочный шифр US-A, 363  
    ультрапотоковый шифр US-A, 364  
Умножение больших чисел  
    метод Карацубы, 303  
    метод Тоома–Кука, 303  
Умножение матриц, 196, 296, 299  
Уодсорт Деций, 137  
Управление радиотехнической разведки, Германия, 277  
Условная вероятность, 87  
Уэлч Терри, 179

---

## Ф

Факториал, 43  
Факторизация, 46, 331  
    квантовая, 360  
Фалькoner Джон, 111  
Фейстель Хорст, 189  
Ферма малая теорема, 328  
Ферма простые числа, 287  
Фибоначчи (Леонардо Пизанский), 44  
Фибоначчи последовательность, 166  
Фракционирование, 53, 135  
    битовое, 158  
    квадрат Полибия. См. *Полибия квадрат*  
    переменной длины, 163  
    повышение стойкости блоков, 161  
    прямоугольные сетки, 156  
    шестнадцатеричное, 157  
    шифр Three Cube, 154  
    шифр Trifid, 152  
    шифр Two Square, 142  
Фракционированный код Морзе, 54  
Фридман Уильям Ф., 76

---

## Х

Халла–Добелла условия, 253  
Халл Т. Э., 253  
Хаффмана коды, 51, 168  
Хаффмана подстановка, 168  
Хаффман Дэвид А., 51  
Хейханен Рейно, 165

Хеллман Мартин, 283  
Хеширования функции, 273  
Хилла шифр, 32, 296, 353  
Хилл Лестер С., 296  
Хитт Паркер, 129  
Хэмминга код, 350

## Ц

Цезаря шифр, 50, 218  
Целевая функция, 356  
Центрирование (статистика), 220, 222  
Цепной аддитивный генератор, 256  
Цепной XOR-генератор, 254  
Цепные дроби, 309  
Циклический сдвиг, 118  
Циклическое умножение, 312  
Циферблата метод, 232

## Ч

Чапман Нойес, 44  
Частота букв, 51, 62, 76  
Частота пар букв, 62

## Ш

Шамир Ади, 31  
Шенкс Дэниэл, 329  
Шеннон Клод, 217, 277  
Шербиус Артур, 36  
Шифровальная микросхема, 189  
Шифрование, 28  
Шифртекст, 28  
Шифры  
    3Base, 316  
    3DES, 193, 372  
    5858, 100  
    арифметическое шифрование, 183  
    Базери типа 4, 59, 142, 201, 371  
    бегущий ключ, 86, 371  
    Беласо, 71, 76, 78, 81, 223, 277, 353, 367, 371  
    Вернама, 278  
    Виженера, 72, 78, 81, 367  
    Виженера автоключ, 72  
    ВИК, 165  
    гибридные, 33, 145, 226  
    двойной Плейфер, 141

двойной столбцовый Bifid, 151  
диагональный Bifid, 151  
дисковый, 35  
заборный, 35  
зачем создавать собственный, 38  
зигзаговый, 35  
маршрутная перестановка, 52, 109, 371  
машинный однороторный, 90  
многоалфавитная подстановка, 100, 118, 206, 236, 247  
моном-биномные, 165  
мультиплексный, 128, 130  
невскрываемые, 28  
номенклатор, 70  
одноалфавитный  
    подстановочный, 50, 62, 366  
    одноразовый блокнот, 33, 248, 276, 328, 368  
    омофоническая подстановка, 50, 99, 105  
    перестановка битов, 191, 194, 229, 364  
    перестановка кусочного обращения, 157, 201  
    перестановка с ключом, 122  
    перестановочный, 35, 52, 59, 109, 142, 157, 189, 201  
    Плейфера, 72, 137, 139, 145, 188  
    Плейферова рябь, 141  
    подстановка биграмм, 100, 108, 143, 190, 202, 310  
    подстановка Лемпеля–Зива, 181  
    подстановка Поста, 171, 177  
    подстановка триграмм, 100  
    подстановка Хаффмана, 168, 171, 246, 353  
    Полибиева рябь, 136  
    последовательная закладка, 228  
    поточковый, 33, 246, 270, 276, 363, 373  
    простой подстановочный, 50, 58, 62  
    пульсация с ключом, 207, 210  
    пульсирующие, 205  
    пульсирующий шифр madd, 216  
    расширительная шахматная доска, 165  
    с автоключом, 72  
    система оценивания, 49  
    тождественный, 204  
    ультраподстановочный US-A, 363

- ультрапотоковый US-B, 364
  - Хилл-0, 297
  - Хилл-1, 297, 298
  - Хилл-2, 297
  - Хилла, 296
  - Цезаря, 30, 50, 218
  - циклическое умножение, 312
  - цилиндр Джефферсона, 35, 128, 200
  - Эверест, 303
  - ADFGVX, 151
  - AES, 37, 196, 198, 352, 372
  - Bicyclic  $8 \times 8$ , 160
  - Bifid, 157, 188
  - Bifid с сопряженной матрицей, 150, 297
  - BitBlock MA, 168
  - BitBlock SA, 168
  - BitCycle Substitution, 161
  - Blackout, 116
  - Brick Wall, 301
  - Butthead, 301
  - Compass, 312
  - CompassS, 312
  - Cycle Hex, 157
  - Cyclic  $8 \times N$ , 159
  - Cysquare, 115
  - DES, 36, 125, 191, 230
  - Double Cyclic  $8 \times N$ , 160
  - DoubleHill, 299
  - EmbedBits, 108
  - Four Square, 146, 157, 188
  - FR-Actionated Morse, 188
  - GenX, 267
  - Hex Rectangle, 159
  - Lag Ripple, 318
  - Lucifer, 189
  - Mat36, 313
  - Merge8, 107
  - Morse3, 164
  - MPM128, 311
  - Mult32, 312
  - Mult128, 303, 310
  - Mult128 с перестановкой, 311
  - Nofair, 140
  - Null5, 105
  - Nullfair, 140
  - Piecewise Hex, 158
  - Playfair+1, 140
  - Playfair ThreeSquare, 145
  - Playfair TwoSquare, 143, 146, 372
  - PMod1, 314
  - PMod2, 314
  - PMod3, 314
  - PMod4, 314
  - poly, 208
  - PolyHuff, 176
  - PolyPlayfair, 141
  - PolyPost, 176
  - Poly Triple Flip, 204, 242
  - Post64, 174
  - PostDL, 176
  - PostOv, 174
  - Rijndael, 198
  - sadd, 208
  - SFlip (Substitute and Flip), 242
  - Three Cube, 154
  - Three Cube Plus, 155
  - Three Cube Super, 156
  - Three Square, 143, 152, 157, 188
  - Tiger, 311
  - Trifid, 152, 164, 188, 371
  - Trig3, 99
  - Triple Ripple, 318
  - Two Square, 142, 152, 157, 188, 353, 371
  - TwoSquare+1, 143
  - Two Square с рябью, 143, 372
  - Two Square B, 143
  - VLA, 205
  - VLB, 205
  - XAESX, 201
  - XDESX, 201
  - Шор Питер, 38, 360
- Э**
- 
- Эверест шифр, 303
  - Экспоненциальная нотация, 45
  - Элиас Питер, 181
  - Энигма шифровальная машина, 36, 77, 283

Книги издательства «ДМК ПРЕСС»  
можно купить оптом и в розницу  
в книготорговой компании «Галактика»  
(представляет интересы издательств  
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38, оф. 10;  
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью),  
по которому должны быть высланы книги;  
фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: <http://www.galaktika-dmk.com/>.

Фрэнк Рубин

### **Криптография с секретным ключом**

Главный редактор *Мовчан Д. А.*  
dmkpress@gmail.com

Зам. главного редактора *Сенченкова Е. А.*

Перевод *Слинкин А. А.*

Корректор *Синяева Г. И.*

Верстка *Чаннова А. А.*

Дизайн обложки *Мовчан А. Г.*

Гарнитура PT Serif. Печать цифровая.

Усл. печ. л. 31,36. Тираж 200 экз.

Веб-сайт издательства: [www.dmkpress.com](http://www.dmkpress.com)

# Криптография с секретным ключом

От шифра Цезаря времен Римской империи до шифровальной машины «Энигма», применявшейся во Второй мировой войне, секретные сообщения оказывали влияние на ход истории. В наши дни криптография с секретным ключом — становой хребет всей современной архитектуры вычислений. Будучи правильно спроектированы, эти алгоритмы практически эффективны. А некоторые не-вскрываются даже с применением суперкомпьютеров и квантовых технологий!

В книге объясняется, как создавать шифры с секретным ключом — от простых, для которых хватает карандаша и бумаги, до очень сложных, применяемых в современной компьютерной криптографии. Вы научитесь эффективно шифровать большие файлы с помощью быстрых потоковых шифров, узнаете об альтернативах шифру AES и сможете избежать шифров, которые только кажутся стойкими. А для развлечения предлагается вскрыть несколько несложных мини-шифров.

*Для профессиональных инженеров, специалистов по информатике и криптографов-любителей.*

*Знания сложной математики не требуется.*

## Вы научитесь:

- конструировать 30 невскрываемых шифров;
- измерять стойкость шифров и гарантированно обеспечивать их безопасность;
- включать в шифр необнаруживаемую закладку;
- противостоять гипотетическим ультракомпьютерам будущего.

Фрэнк Рубин занимается криптографией больше 50 лет. Имеет степень магистра математики и доктора информатики.

«Раскройте в себе внутреннего агента секретной службы, начните проектировать и создавать шифры мирового класса. А эта книга вам поможет!»

*Рой Принс, Salesforce*

«Эта книга — ключ к миру криптографии. И это не секрет!»

*Крис Карделл, FIS*

«Фантастика! Тут и история, и методы криптографии. Никогда не думал, что мне это понадобится — пока не прочел эту книгу!»

*Мэтью Харвелл, AbacusNext*

Интернет-магазин: [www.dmkpress.com](http://www.dmkpress.com)

Оптовая продажа: КТК «Галактика»  
[books@aliants-kniga.ru](mailto:books@aliants-kniga.ru)



ISBN 978-5-97060-748-0



9 785970 607480 >