3.51. Алгоритм Sha-zam

Алгоритм шифрования Sha-zam разработан в 1999 г. тремя сотрудниками компании Lucent Technologies. Авторы алгоритма: Сарвар Пател (Sarvar Patel), Зульфикар Рамзан (Zulfikar Ramzan) и Ганеш Сундарам (Ganesh Sundaram).

Структура алгоритма

Алгоритм Sha-zam немного напоминает рассмотренные в *разд. 3.6* алгоритмы Bear, Lion и Lioness. Сходство в том, что Sha-zam также использует в качестве одного из преобразований алгоритм хэширования, а именно принятый в США стандарт хэширования SHA (Secure Hash Algorithm) в его 160-битном варианте SHA-1. Описание семейства алгоритмов SHA можно найти в тексте стандарта FIPS 180-2 [152].

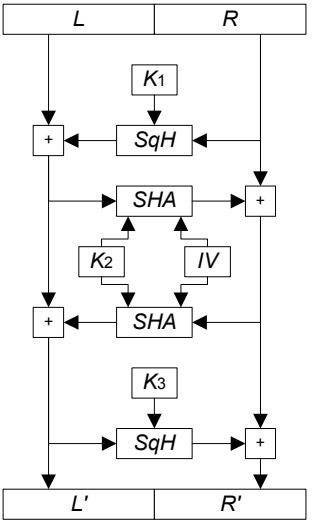

Рис. 3.195. Структура алгоритма Sha-zam

Структура алгоритма исключительно проста и состоит в выполнении следующих операций (рис. 3.195) [297]:

$$T1 = L + SqH_{k1}(R) \bmod 2^{160};$$
$$T2 = R + SHA(IV, T1, k2) \bmod 2^{160};$$
$$L' = T1 + SHA(IV, T2, k2) \bmod 2^{160};$$
$$R' = T2 + SqH_{k3}(L') \bmod 2^{160},$$

где:

- L и R — соответственно, левый и правый субблоки открытого текста по 160 битов каждый, т. е. алгоритм Sha-zam шифрует данные блоками по 320 битов;
- $T1$ и $T2$ — временные переменные;
- $k1$, $k2$ и $k3$ — фрагменты расширенного ключа (процедура расширения ключа будет описана далее); $k2$ имеет размер 352 бита, $k1$ и $k3$ — по 160 битов;
- IV — вектор инициализации — 160-битный дополнительный параметр алгоритма (см. далее);
- L' и R' — соответственно, левый и правый субблоки шифртекста;
- SqH — операция «квадратичного хэширования» (square hash), изобретенная авторами алгоритма и состоящая в выполнении следующего действия:

$$SqH_x(y) = (x+y)^2 \bmod p \bmod 2^{160},$$

где p — простое число, минимальное из простых чисел, бóльших 2^{160}.

Варьируя значение вектора инициализации, можно получать различные значения шифртекста для одного и того же открытого текста и одного и того же ключа шифрования. В данном случае IV используется в качестве начального значения
в алгоритме хэширования SHA (Initial hash value). Значение IV в алгоритме Sha-zam может быть зависимым от ключа шифрования, может быть константой, а может и не использоваться — в последнем случае в качестве начального берется стандартное значение, описанное в спецификации SHA-1 (5 значений по 32 бита) [152] — см. табл. 3.131 (указаны шестнадцатеричные значения).

Таблица 3.131

67452301
EFCDAB89
98BADCFE
10325476
C3D2E1F0

Расшифровывание данных выполняется с помощью обратных операций, применяемых в обратной последовательности:

$$T2 = R' - SqH_{k3}(L') \bmod 2^{160};$$
$$T1 = L' - SHA(IV, T2, k2) \bmod 2^{160};$$
$$R = T2 - SHA(IV, T1, k2) \bmod 2^{160};$$

$$L = T1 - SqH_{k1}(R) \bmod 2^{160}.$$

Процедура расширения ключа

Алгоритм Sha-zam использует ключи шифрования весьма необычного размера — 100 битов. На основе исходного ключа шифрования процедура расширения ключа вычисляет 672 бита расширенного ключа. Если же используется зависимость вектора инициализации от ключа шифрования, то данная процедура вырабатывает уже 832 бита ключевой информации.

Расширение ключа происходит таким образом:

1. В цикле по *i* от 1 до *m* выполняется следующее действие:

   $$s_i = SHA(s', C_i'),$$

   где:

   - s' — результат применения операции XOR к секретному ключу и старшим 100 битам значения *IV*, в качестве которого в процедуре расширения ключа используется приведенное выше стандартное значение;
   - C_i' — результат применения операции XOR к *i*-му 32-битному слову c_i ($c_0 ... c_m$ — дополнительный параметр алгоритма) и каждому четному 32-битному слову 512-битной константы *C* (не указана в спецификации алгоритма [297], отсчет слов константы начинается с младшего с номером 0);
   - *m* — количество циклов, необходимое для вычисления расширенного ключа требуемого размера, считая, что в каждой итерации цикла вычисляется 160 битов s_i.

2. Необходимое количество битов расширенного ключа набирается из последовательности $h(s_1)...h(s_m)$, где $h()$ — преобразование, не описанное в спецификации алгоритма Sha-zam; функция $h()$ должна заменять 160-битное значение другим, например, возведением s_i в фиксированную степень по модулю 2^{160}. В принципе, эта функция может и не применяться совсем, а в качестве расширенного ключа можно использовать значения $s_1 ... s_m$.

На самом деле размер 100 битов является только рекомендуемым. В качестве начального значения описанного генератора псевдослучайных чисел могут использоваться и ключи большего или меньшего размера.

Криптостойкость алгоритма

Несмотря на достаточно интересную структуру алгоритма Sha-zam, он не вызвал внимания со стороны криптоаналитиков. Какие-либо работы, посвященные доказательству криптостойкости этого алгоритма (за исключением собственно описания алгоритма [297], часть которого посвящена данному вопросу, и другой работы авторов алгоритма — [298]) или поиску уязвимостей в нем, не получили широкой известности.

3.52. Алгоритм Skipjack

Алгоритм Skipjack интересен по многим причинам.

- Данный алгоритм разработан Агентством Национальной Безопасности США для шифрования в специальных случаях [28].
- Фактически алгоритм Skipjack представлял собой еще один стандарт шифрования США — разработанный в 1987 г. и существующий одновременно со стандартом DES. Принципиальное различие этих алгоритмов состоит в том, что DES являлся открытым стандартом, алгоритм был опубликован и полностью открыт, что позволило всем заинтересованным специалистам оценить стойкость алгоритма. В отличие от DES, алгоритм Skipjack был долгое время засекречен, его описание стало доступно на сайте института NIST только в 1998 г. [358].
- Брюс Шнайер в [28] утверждает, что алгоритм является секретным не для повышения его надежности, а для того, чтобы его нельзя было использовать в сторонних реализациях — подразумевалось использование Skipjack только в аппаратных реализациях Clipper и Fortezza. Микросхема Clipper получила скандальную известность благодаря тому, что в ней реализован механизм депонирования ключей. Это означает, что любой ключ шифрования какого-либо пользователя (использующего реализацию Skipjack в микросхеме Clipper) может быть легко вскрыт правительственными чиновниками США, получившими соответствующий ордер. Аналогичный механизм существует и в криптоплате Fortezza. Настойчивое навязывание Clipper пользователям со стороны АНБ стало причиной многочисленных протестов против использования данной системы со стороны различных американских правозащитных организаций. Следствием этого стал факт ограниченного распространения алгоритма Skipjack и его аппаратных реализаций.

Структура алгоритма

Алгоритм Skipjack шифрует данные блоками по 64 бита и использует 80-битный ключ шифрования. В процессе шифрования обработка данных производится по 16-битным словам, т. е. входной блок данных разбивается на 4 слова, обозначаемые w_1, w_2, w_3 и w_4. Выполняются 32 раунда преобразований, причем алгоритм предполагает два варианта функций раундов (функция *A* и функция *B*), в каждом раунде задействована только одна из них [358].

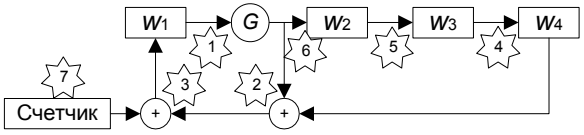

Рис. 3.196. Функция *A*

Функция *A* представлена на рис. 3.196:

1. Над словом w_1 выполняется операция *G*, которая представляет собой зависящее от ключа преобразование (подробно описана далее).
2. Результат предыдущего шага складывается операцией XOR со значением слова w_4.
3. Результат предыдущего шага складывается операцией XOR с текущим значением счетчика (см. далее), результат этой операции становится новым значением слова w_1.
4. Текущее значение w_3 замещает старое значение w_4.
5. Текущее значение w_2 замещает старое значение w_3.
6. Результат шага 1 становится новым значением w_2.
7. Значение счетчика увеличивается на 1.

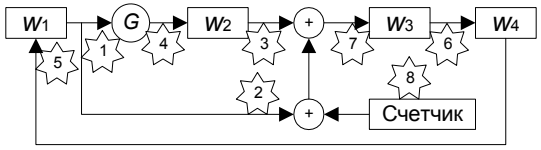

Рис. 3.197. Функция *B*

Функция *B* представлена на рис. 3.197:

1. Над словом w_1 выполняется операция *G*.
2. Значение слова w_1 складывается операцией XOR с текущим значением счетчика.
3. Результат предыдущего шага складывается операцией XOR со значением слова w_2.
4. Результат шага 1 становится новым значением слова w_2.
5. Текущее значение w_4 замещает старое значение w_1.
6. Текущее значение w_3 замещает старое значение w_4.
7. Результат шага 3 становится новым значением слова w_3.
8. Значение счетчика увеличивается на 1.

В процессе зашифровывания 8 раз выполняется функция *A*, затем 8 раз выполняется функция *B*, затем эти 16 раундов повторяются, т. е.:

$$P \to (8 * A) \to (8 * B) \to (8 * A) \to (8 * B) \to C,$$

где *P* и *C* — соответственно, открытый текст и результат его зашифровывания.

Перед выполнением этих 32 операций значение счетчика устанавливается в 1.

Расшифровывание производится путем выполнения обратных операций в обратной последовательности. Перед расшифровыванием счетчик устанавливается в 32.

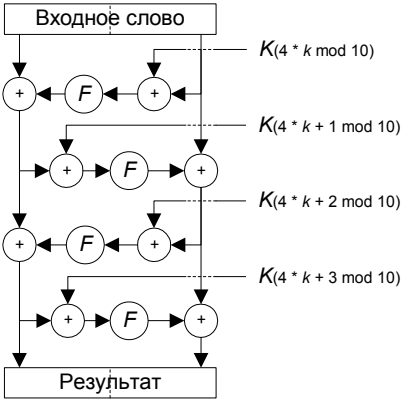

Рис. 3.198. Операция *G*

Операция *G* представляет собой 4-раундовую сеть Фейстеля, в которой в каждом раунде выполняются следующие действия (рис. 3.198):

1. Обрабатываемый в текущем раунде байт входного слова (правый в нечетных раундах или левый — в четных) складывается операцией XOR с байтом ключа шифрования K_i, индекс которого определяется по формуле:

$$i = 4 * k + r \bmod 10,$$

где:

- k — номер текущего раунда алгоритма;
- r — номер текущего раунда операции G.

2. Результат предыдущего шага замещается согласно таблице замен F (табл. 3.132); старший нибл входного значения определяет номер строки (начиная с нуля), а младший нибл — номер столбца, на пересечении которых находится выходное значение.

Таблица 3.132

A3	D7	09	83	F8	48	F6	F4	B3	21	15	78	99	B1	AF	F9
E7	2D	4D	8A	CE	4C	CA	2E	52	95	D9	1E	4E	38	44	28
0A	DF	02	A0	17	F1	60	68	12	B7	7A	C3	E9	FA	3D	53
96	84	6B	BA	F2	63	9A	19	7C	AE	E5	F5	F7	16	6A	A2
39	B6	7B	0F	C1	93	81	1B	EE	B4	1A	EA	D0	91	2F	B8
55	B9	DA	85	3F	41	BF	E0	5A	58	80	5F	66	0B	D8	90
35	D5	C0	A7	33	06	65	69	45	00	94	56	6D	98	9B	76
97	FC	B2	C2	B0	FE	DB	20	E1	EB	D6	E4	DD	47	4A	1D
42	ED	9E	6E	49	3C	CD	43	27	D2	07	D4	DE	C7	67	18
89	CB	30	1F	8D	C6	8F	AA	C8	74	DC	C9	5D	5C	31	A4
70	88	61	2C	9F	0D	2B	87	50	82	54	64	26	7D	03	40
34	4B	1C	73	D1	C4	FD	3B	CC	FB	7F	AB	E6	3E	5B	A5
AD	04	23	9C	14	51	22	F0	29	79	71	7E	FF	8C	0E	E2
0C	EF	BC	72	75	6F	37	A1	EC	D3	8E	62	8B	86	10	E8
08	77	11	BE	92	4F	24	C5	32	36	9D	CF	F3	A6	BB	AC
5E	6C	A9	13	57	25	B5	E3	BD	A8	3A	01	05	59	2A	46

3. Результат предыдущего шага накладывается операцией XOR на другой байт входного слова.

Процедура расширения ключа у данного алгоритма отсутствует — операция *G* использует байты исходного ключа шифрования по мере необходимости без их предварительной обработки. Однако в спецификации алгоритма Skipjack [358] определен также протокол KEA (Key Exchange Algorithm) — протокол выработки общего симметричного ключа на основе секретного ключа отправителя информации и открытого ключа получателя. Данный протокол базируется на известном алгоритме вычисления общего ключа Диффи—Хеллмана. Очевидно, что 80-битный ключ шифрования может быть как вычислен с помощью алгоритма KEA, так и задан напрямую, что традиционно для алгоритмов симметричного шифрования.

Криптостойкость алгоритма

В то время, пока алгоритм Skipjack был засекречен, у пользователей и специалистов возникало множество вопросов по поводу его криптостойкости: пусть в реализациях алгоритма присутствует механизм депонирования ключей, позволяющий уполномоченным лицам вычислить сессионный ключ; но достаточно ли надежен сам секретный алгоритм? Нет ли в нем каких-либо случайных или преднамеренно внедренных разработчиками уязвимостей?

Скорее всего, стремясь пресечь слухи о возможных слабостях алгоритма, в 1993 г. АНБ передало спецификацию алгоритма и сопроводительную документацию к нему пяти известным экспертам в области криптологии для изучения алгоритма и опубликования отчета, который не должен был содержать описания внутренней реализации алгоритма, однако должен был дать понять (не в последнюю очередь благодаря авторитету изучавших алгоритм экспертов) заинтересованным лицам, что алгоритм Skipjack является криптографически стойким.

Отчет экспертов [105] появился в июле 1993 г. и содержал, в частности, следующие выводы:

- благодаря 80-битному ключу алгоритма Skipjack (сравнивая с 56-битным ключом DES), учитывая предполагаемую скорость развития вычислительной техники, крайне незначителен риск взлома алгоритма перебором всех возможных вариантов ключа (атака методом «грубой силы») в ближайшие 30–40 лет;
- отсутствует серьезный риск взлома алгоритма более эффективными криптоаналитическими методами, включая метод дифференциального криптоанализа;

- криптостойкость алгоритма Skipjack не зависит от секретности самого алгоритма.

Причем по поводу второго из приведенных выводов эксперты сделали оговорку, что за предоставленное им время серьезный анализ алгоритма провести невозможно, поэтому вывод базируется только на изучении ими критериев и процесса разработки и тестирования алгоритма, проведенных в АНБ.

Полномасштабный криптоанализ алгоритма Skipjack начался уже после опубликования его спецификации в 1998 г. В том же году вышла работа ряда специалистов из Израиля [61], в которой, в частности, было отмечено свойство несимметричности ключа шифрования Skipjack, незначительно снижающее трудоемкость полного перебора ключей. В этой же работе было представлено несколько атак на усеченные версии алгоритма с неполным количеством раундов и другими изменениями. Стоит отметить одну из опубликованных атак, действовавшую против варианта Skipjack, в котором не было всего трех операций XOR по сравнению со стандартной версией — в раундах № 4, 16 и 17. Такая версия алгоритма получила название Skipjack-3XOR. Интересно, что удаление всего трех операций XOR из 320 подобных операций приводит к полной слабости алгоритма — в этом случае ключ вскрывается при наличии 2^9 пар блоков открытого текста и шифртекста выполнением всего около миллиона операций шифрования.

В том же году авторы предыдущей работы представили новый вид дифференциального криптоанализа [63], основанного на поиске ключа «от противного»: если попытка расшифровывания двух шифртекстов на каком-либо ключе приводит к такому соотношению между результатами их расшифровывания, которое невозможно в принципе, то данный ключ является неверным. Такая технология криптоанализа может быть полезна, в частности, для существенного сужения области полного перебора ключей (см. подробное описание криптоанализа с использованием невозможных дифференциалов в *разд. 1.6*). Однако атаке оказались подвержены только усеченные версии алгоритма.

Были предприняты и более поздние попытки криптоанализа алгоритма Skipjack (например, [166, 225]), однако все они оказались неспособными взломать полноценную и полнораундовую версию алгоритма. При этом многие криптоаналитики высказывали мнение, что успешность атак на усеченные версии алгоритма говорит о его потенциальной слабости, что, впрочем, не доказано.

3.53. Алгоритм SPEED

Алгоритм блочного симметричного шифрования SPEED предложен в 1997 г. австралийским криптологом Юлианом Женом (Yuliang Zheng). Аббревиатура SPEED обозначает Secure Package for Encrypting Electronic Data, т. е. «пакет безопасности для шифрования электронных данных».

Структура алгоритма

Алгоритм SPEED фактически представляет собой целое семейство алгоритмов, в котором основные параметры являются переменными. В этом SPEED похож на существенно более известный алгоритм RC5 (*см. разд. 3.42*) — как и в RC5, в алгоритме SPEED параметризуются следующие величины [407]:

- размер блока шифруемых данных W может принимать значения 64, 128 или 256 битов;
- размер ключа шифрования L принимает любое, кратное 16, значение в диапазоне от 48 до 256 битов включительно;
- количество раундов преобразований R может быть не менее 32 и должно быть кратно 4.

Независимо от количества раундов, шифрование выполняется в 4 фазы, которые обозначаются как $P1...P4$ (рис. 3.199). В каждой фазе выполняется $R/4$ раундов преобразований. Структура раунда приведена на рис. 3.200.

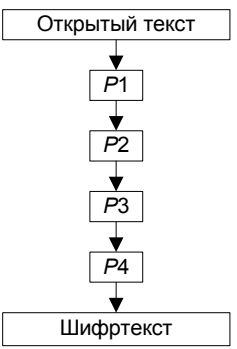

Рис. 3.199. Структура алгоритма

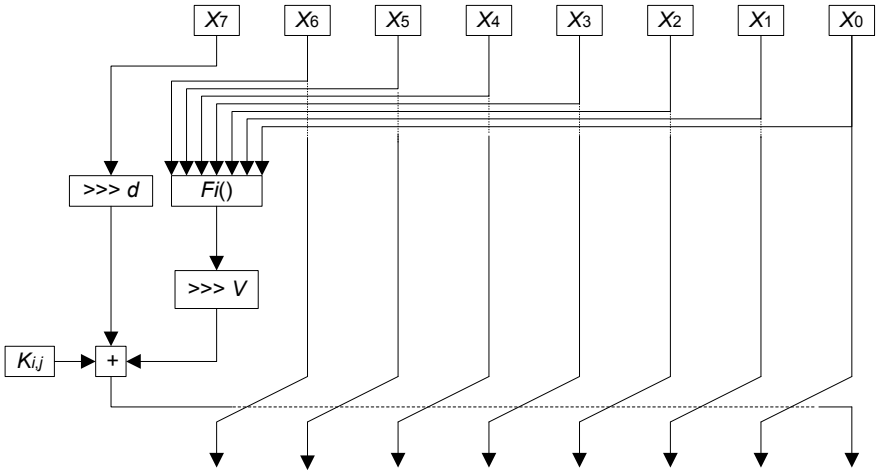

Рис. 3.200. Раунд алгоритма

Как видно из рисунка, блок данных делится на 8 фрагментов $X_0 ... X_7$, каждый из которых имеет размер $W/8$ битов. Над этими фрагментами выполняются следующие действия:

1. Фрагменты $X_0 ... X_6$ обрабатываются функцией $Fi()$. Здесь i — это номер текущей фазы. Функции $Fi()$ вычисляют $W/8$-битное значение на основе семи $W/8$-битных фрагментов таким образом:

$$F1(X_0 ... X_6) = X_6 X_3 \oplus X_5 X_1 \oplus X_4 X_2 \oplus X_1 X_0 \oplus X_0;$$

$$F2(X_0 ... X_6) = X_6 X_4 X_0 \oplus X_4 X_3 X_0 \oplus X_5 X_2 \oplus X_4 X_3 \oplus X_4 X_1 \oplus X_3 X_0 \oplus X_1;$$

$$F3(X_0 ... X_6) = X_5 X_4 X_0 \oplus X_6 X_4 \oplus X_5 X_2 \oplus X_3 X_0 \oplus X_1 X_0 \oplus X_3;$$

$$F4(X_0 ... X_6) = X_6 X_4 X_2 X_0 \oplus X_6 X_5 \oplus X_4 X_3 \oplus X_3 X_2 \oplus X_1 X_0 \oplus X_2,$$

   где запись $X_n X_m$ обозначает применение побитовой логической операции «и» к указанным операндам.

2. Результат предыдущего шага циклически сдвигается вправо (на рис. 3.200 вращение вправо обозначено как $>>>$) на переменное число битов v (вращение на переменное число битов — еще одно сходство SPEED с алгоритмом RC5). v определяется как значение старших $\log_2(W/8)$ битов (которыми являются биты 1…3 для $W = 64$, биты 4…7 для $W = 128$ или биты 11…15 для $W = 256$) результата сложения левых и правых $W/16$ битов выходного значения предыдущего шага, т. е. значения V:

$$V = (F >> (W/16)) + F,$$

где:

- F — результат предыдущего шага;
- $>>$ — операция побитового сдвига вправо.

3. Значение фрагмента X_7 циклически сдвигается вправо на фиксированное число битов *d*, которое определено так:

$$d = W/16 - 1.$$

4. Результат шага 2 складывается с результатом шага 3 и $W/8$-битным фрагментом расширенного ключа $K_{i,j}$ по модулю $2^{W/8}$. Здесь j в индексе фрагмента ключа обозначает номер раунда в рамках текущей фазы. Процедура расширения ключа будет описана далее.

5. Фрагменты $X_0 ... X_6$ сдвигаются на 1 фрагмент влево, т. е.:

$$X_{n+1} = X_n \text{ для } n = 6...0.$$

6. Результат шага 4 становится новым значением фрагмента X_0.

Данный алгоритм можно представить в виде несбалансированной сети Фейстеля (source-heavy unbalanced Feistel network), в которой в каждом раунде результат обработки субблока большего размера ($X_0 ... X_6$) некоей функцией $H()$, представляющей собой совокупность описанных выше шагов 1, 2 и 4, накладывается на субблок меньшего размера — X_7. Такое представление раунда алгоритма показано на рис. 3.201.

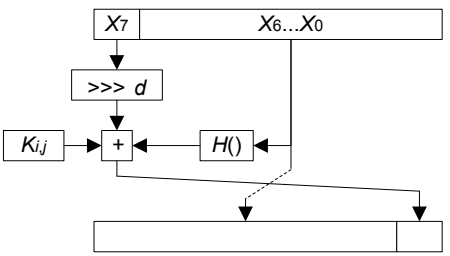

Рис. 3.201. Альтернативное представление раунда алгоритма

Автор алгоритма рекомендует использовать следующие минимальные значения размера ключа и количества раундов для различных значений *W* (табл. 3.133).

Таблица 3.133

W	64	128	256

L	≥ 64	≥ 64	≥ 64
R	≥ 64	≥ 48	≥ 48

Возможно использование значений, меньших указанных, но только в случае, если алгоритм применяется в качестве основы функции хэширования данных (*см. разд. 1.1*).

Расшифровывание

При расшифровывании фазы алгоритма выполняются в обратном порядке, т. е. для $K_{i,j}$ и $Fi()$ $i = 4...1$. В каждой фазе раунды также нумеруются в обратном порядке, т. е. от $(R/4-1)$ до 0. И, наконец, в каждом раунде выполняются обратные раунду зашифровывания операции, т. е. (рис. 3.202):

1. Все фрагменты циклически вращаются на 1 фрагмент вправо.
2. Фрагменты $X_0...X_6$ обрабатываются функцией $Fi()$.
3. Результат предыдущего шага вращается вправо на *v* битов.
4. Побитовый комплемент (на рис. 3.202 операция его получения обозначена как ~) фрагмента расширенного ключа $K_{i,j}$, фрагмент X_7 и побитовый комплемент результата шага 3 складываются по модулю $2^{W/8}$.
5. Результат предыдущего шага вращается влево на *d* битов и становится новым значением фрагмента X_7.

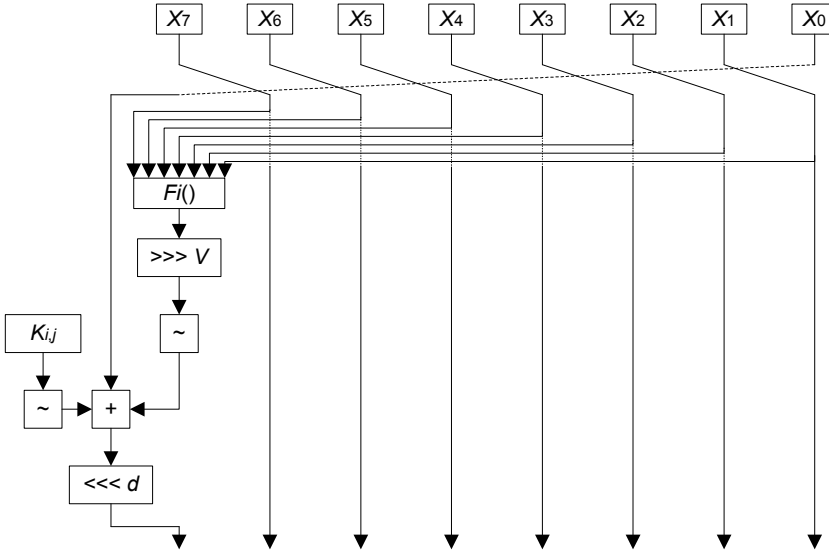

Рис. 3.202. Раунд расшифровывания

Процедура расширения ключа

Задача процедуры расширения ключа состоит в получении из исходного L-битного ключа K необходимого количества (R) фрагментов расширенного ключа размером $W/8$ битов для использования по одному в R раундах алгоритма.

Прежде всего создается массив переменных $kb_0 ... kb_{Z-1}$, где каждая из переменных kb_n имеет размер 16 битов, а количество элементов массива (Z) напрямую зависит от размера блока алгоритма и количества раундов:

- $Z = R/2$ для $W = 64$;
- $Z = R$ для $W = 128$;
- $Z = 2R$ для $W = 256$.

Первые $L/16$ элементов массива ($kb_0 ... kb_{L/16-1}$) инициализируются соответствующими битами ключа K. Помимо этого массива, в процедуре расширения ключа принимают участие три переменных состояния $S_0 ... S_2$, каждая из которых также имеет размер 16 битов, а начальное значение определяется следующим образом:

$$S_0 = Q_0;$$

$$S_1 = Q_1;$$

$$S_2 = Q_2,$$

где $Q_0 ... Q_2$ — константы, значение которых зависит от размера ключа шифрования. Эти константы набираются из массива из 42 констант, образованного дробной частью числа $\sqrt{15}$ (табл. 3.134 — указаны шестнадцатеричные значения).

Таблица 3.134

DF7B	D629	E9DB	362F	5D00	F20F	C3D1
1FD2	589B	4312	91EB	718E	BF2A	1E7D
B257	77A6	1654	6B2A	0D9B	A9D3	668F
19BE	F855	6D98	022D	E4E2	D017	EA2F
7572	C3B5	1086	480C	3AA6	9CA0	98F7
D0E4	253C	C901	55F3	9BF4	F659	D76C

Для $L = 48$ в качестве $Q_0 ... Q_2$ используются первые 3 константы данного массива, для $L = 64$ — следующие 3 константы и т. д. до максимального размера ключа в 256 битов, при расширении которого в качестве $Q_0 ... Q_2$ берутся 3 последние константы массива.

Затем на основе проинициализированных элементов массива ($kb_0 ... kb_{L/16-1}$) формируются остальные его элементы ($kb_{L/16} ... kb_{Z-1}$). Для этого в цикле по n от $L/16$ до $Z-1$ выполняются раунды процедуры расширения ключа, каждый из которых состоит из следующих действий (рис. 3.203):

1. Переменные состояния $S_0 ... S_2$ обрабатываются функцией $G()$, которая определена таким образом:

$$G(S_0 ... S_2) = S_0 S_1 \oplus S_1 S_2 \oplus S_0 S_2.$$

2. 16-битный результат предыдущей операции вращается влево на 5 битов.
3. Результат предыдущего шага складывается по модулю 2^{16} со значением S_2 и проинициализированным ранее элементом массива $kb_0 ... kb_{Z-1}$, индекс которого определяется как $n \bmod (L/16)$.
4. Выполняется сдвиг переменных состояния $S_0 ... S_2$:

$$S_2 = S_1,$$
$$S_1 = S_0.$$

5. Результат шага 3 становится значением kb_n и новым значением переменной S_0.

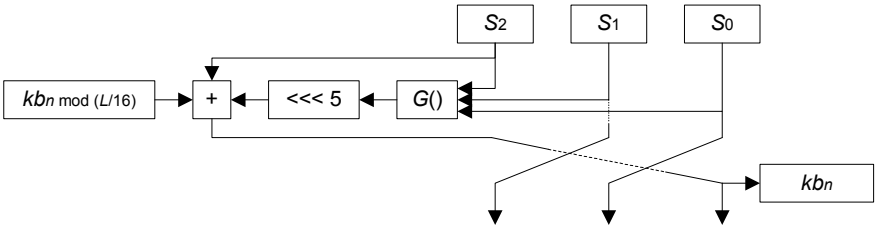

Рис. 3.203. Раунд процедуры расширения ключа

В завершение процедуры расширения ключа массив 16-битных переменных $kb_0 ... kb_{Z-1}$ преобразуется в массив фрагментов расширенного ключа из $R\ W/8$-битных элементов $K_{i,j}$, которые используются в раундах шифрования, как было показано выше.

Достоинства и недостатки алгоритма

При разработке алгоритма автор руководствовался следующими критериями: алгоритм должен быть простым и компактным в реализации, иметь высокое быстродействие и достаточную криптостойкость. Изначально алгоритм создает впечатление, что его автор добился вышеперечисленных целей. Однако в 1998 г. несколько известных криптологов: Крис Холл (Chris Hall), Джон Келси, Винсент Риджмен, Брюс Шнайер и Дэвид Вагнер — опубликовали статью, посвященную криптоанализу алгоритма SPEED. Их исследования показали, что SPEED имеет ряд потенциальных слабостей и, как результат, атаки, использующие найденные уязвимости, могут быть реализованы на практике [169]:

- эксперты обнаружили ряд принципиальных недостатков в структуре раунда алгоритма; в частности, весьма неудачен выбор значения *v* в шаге 3 раунда: это значение зависит от выходного значения функции $Fi()$, т. е. от самóй сдвигаемой величины, что существенно сужает область выходных значений шага 3;
- алгоритм подвержен дифференциальному криптоанализу;
- алгоритм подвержен атакам на связанных ключах.

Эксперты предположили, что противодействовать найденным атакам можно увеличением рекомендованного автором количества раундов алгоритма (см. выше). Однако существенное увеличение количества раундов отрицательно сказалось на производительности алгоритма: проведенное экспертами сравнение производительности показало, что алгоритм SPEED с увеличенным количеством раундов уступает в быстродействии таким извест-

ным и криптографически стойким алгоритмам шифрования, как Blowfish и RC5-32/16, в 2–12 раз (в зависимости от конкретных значений W и R), т. е. криптографически стойкий вариант алгоритма SPEED является весьма медленным.

В результате алгоритм SPEED не получил широкой известности.

3.54. Алгоритм Square

Алгоритм Square интересен, прежде всего, по двум изложенным далее причинам.

- Этот алгоритм разработан теми же специалистами, которые впоследствии разработали алгоритм Rijndael (*см. разд. 3.3*), т. е. Винсентом Риджменом и Джоан Деймен.
- Мало того, именно структура алгоритма Square легла в основу алгоритма Rijndael. Структура алгоритма являлась весьма нетрадиционной для современных алгоритмов симметричного шифрования данных — это справедливо как для 1997 г., когда был разработан алгоритм Square, так и для 2000 г., когда при подведении итогов конкурса AES эксперты отмечали, что «в основе алгоритма Rijndael лежит нетрадиционная парадигма, поэтому алгоритм может содержать скрытые уязвимости». Это не помешало алгоритму Rijndael стать новым стандартом шифрования США, а та самая нетрадиционная структура сейчас называется «квадрат» (Square) по названию алгоритма, в котором она была впервые применена. Кстати, в конкурсе AES участвовал и еще один алгоритм со Square-подобной структурой — алгоритм Crypton (*см. разд. 3.12*), разработанный совсем другими авторами.

Структура алгоритма

Алгоритм Square шифрует данные блоками по 128 битов, длина ключа также составляет 128 битов. 128-битный блок данных представляется в виде двумерного байтового массива (таблицы) размером 4×4 — отсюда и название алгоритма. Текущее значение байтов массива в спецификации алгоритма называется *состоянием* (state). Над состоянием выполняется 8 раундов преобразований, каждый из которых состоит из следующих операций [130]:

1. Линейное преобразование θ, выполняющееся раздельно над каждой строкой таблицы (рис. 3.204):

$$\theta : b_{i,j} = c_j a_{i,0} \oplus c_{j-1} a_{i,1} \oplus c_{j-2} a_{i,2} \oplus c_{j-3} a_{i,3},$$

где:

- $a_{i,j}$ — текущее значение байта состояния, принадлежащего i-й строке и j-му столбцу;
- $b_{i,j}$ — новое значение байта состояния;
- c_n — набор констант, определенных в спецификации алгоритма;
- умножение выполняется по модулю 2^8.

2. Нелинейное преобразование, представляющее собой табличную замену (рис. 3.205):

$$\gamma : b_{i,j} = S(a_{i,j}).$$

Таблица, реализующая замену, приведена в табл. 3.135 (указаны шестнадцатеричные значения).

То есть значение 0 заменяется на B1, 1 — на CE и т. д.

Таблица 3.135

b1	ce	c3	95	5a	ad	e7	02	4d	44	fb	91	0c	87	a1	50
cb	67	54	dd	46	8f	e1	4e	f0	fd	fc	eb	f9	c4	1a	6e
5e	f5	cc	8d	1c	56	43	fe	07	61	f8	75	59	ff	03	22
8a	d1	13	ee	88	00	0e	34	15	80	94	e3	ed	b5	53	23
4b	47	17	a7	90	35	ab	d8	b8	df	4f	57	9a	92	db	1b
3c	c8	99	04	8e	e0	d7	7d	85	bb	40	2c	3a	45	f1	42
65	20	41	18	72	25	93	70	36	05	f2	0b	a3	79	ec	08
27	31	32	b6	7c	b0	0a	73	5b	7b	b7	81	d2	0d	6a	26
9e	58	9c	83	74	b3	ac	30	7a	69	77	0f	ae	21	de	d0
2e	97	10	a4	98	a8	d4	68	2d	62	29	6d	16	49	76	c7
e8	c1	96	37	e5	ca	f4	e9	63	12	c2	a6	14	bc	d3	28
af	2f	e6	24	52	c6	a0	09	bd	8c	cf	5d	11	5f	01	c5
9f	3d	a2	9b	c9	3b	be	51	19	1f	3f	5c	b2	ef	4a	cd
bf	ba	6f	64	d9	f3	3e	b4	aa	dc	d5	06	c0	7e	f6	66
6c	84	71	38	b9	1d	7f	9d	48	8b	2a	da	a5	33	82	39
d6	78	86	fa	e4	2b	a9	1e	89	60	6b	ea	55	4c	f7	e2

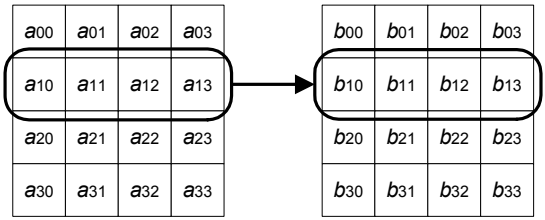

Рис. 3.204. Операция θ

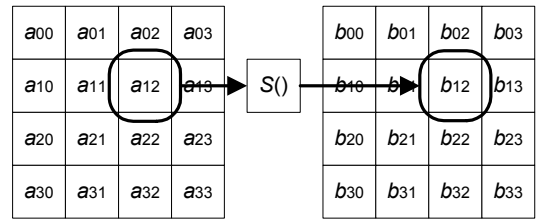

Рис. 3.205. Операция γ

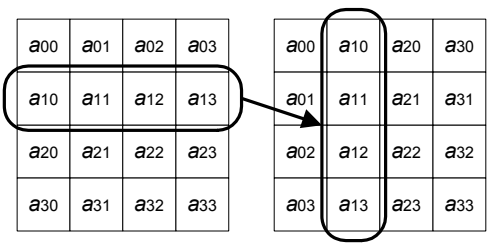

Рис. 3.206. Операция π

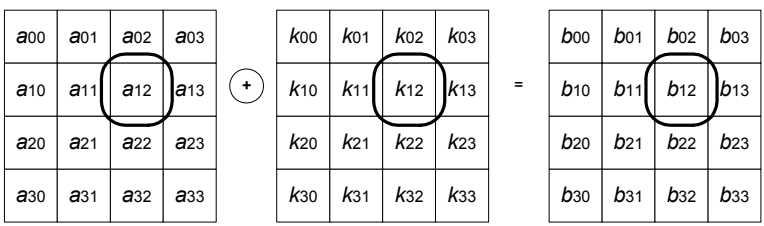

Рис. 3.207. Операция σ