

## СЕРГЕЙ ПАНАСЕНКО



# АЛГОРИТМЫ ШИФРОВАНИЯ

# специальный справочник

270

## Классификация алгоритмов шифрования и методов их вскрытия

## Новейшая история симметричного шифрования

## Описание алгоритмов

**Сергей Панасенко**

# **АЛГОРИТМЫ ШИФРОВАНИЯ**

## специальный справочник

Санкт-Петербург

«БХВ-Петербург»

2009

УДК 681.3.06  
ББК 32.973.26-018.2  
П16

## Панасенко С. П.

П16 Алгоритмы шифрования. Специальный справочник. —  
СПб.: БХВ-Петербург, 2009. — 576 с.: ил.  
ISBN 978-5-9775-0319-8

Книга посвящена алгоритмам блочного симметричного шифрования. Данна общая классификация криптографических алгоритмов. Рассмотрено более 50 алгоритмов шифрования: история создания и использования, основные характеристики и структура, достоинства и недостатки. Описаны различные виды криптоаналитических атак на алгоритмы шифрования и на их реализации в виде программных или аппаратных шифраторов. Рассказано о конкурсах по выбору стандартов шифрования США и Евросоюза.

*Для специалистов в области информационных технологий,  
преподавателей, студентов и аспирантов*

УДК 681.3.06  
ББК 32.973.26-018.2

## Группа подготовки издания:

Главный редактор	Екатерина Кондукова
Зам. главного редактора	Игорь Шишигин
Зав. редакцией	Григорий Добин
Редактор	Ирина Иноземцева
Компьютерная верстка	Наталья Караваевой
Корректор	Наталья Першакова
Дизайн обложки	Елены Беляевой
Зав. производством	Николай Тверских

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 24.10.08.  
Формат 70×100<sup>1</sup>/16. Печать офсетная. Усл. печ. л. 46,44.  
Тираж 2000 экз. Заказ №

"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов  
в ГУП "Типография "Наука"  
199034, Санкт-Петербург, 9 линия, 12

# Оглавление

<b>Введение .....</b>	<b>1</b>
<b>Глава 1. Классификация алгоритмов шифрования и методов их вскрытия .....</b>	<b>3</b>
1.1. Криптографические алгоритмы .....	3
1.2. Категории алгоритмов шифрования.....	7
1.3. Структура алгоритмов симметричного шифрования .....	8
Алгоритмы на основе сети Фейстеля .....	9
Алгоритмы на основе подстановочно-перестановочных сетей .....	11
Алгоритмы со структурой «квадрат».....	11
Алгоритмы с нестандартной структурой .....	12
1.4. Режимы работы алгоритмов.....	12
Электронная кодовая книга .....	13
Сцепление блоков шифра .....	14
Обратная связь по шифртексту .....	15
Обратная связь по выходу .....	16
Другие режимы работы.....	18
1.5. Атаки на алгоритмы шифрования .....	18
Цели атак .....	18
Классификация атак .....	19
Количественная оценка криптоустойчивости алгоритмов шифрования .....	21
Криptoанализ модифицированных алгоритмов .....	22
1.6. Криптоаналитические методы, используемые в атаках .....	23
Метод «грубой силы».....	23
Атаки класса «встреча посередине» .....	25
Дифференциальный криптоанализ .....	27
Линейный криптоанализ .....	34
Метод бумеранга .....	38
Сдвиговая атака .....	40

Метод интерполяции.....	42
Невозможные дифференциалы .....	43
Заключение.....	44
<b>1.7. Атаки на шифраторы, использующие утечку данных</b>	
по побочным каналам .....	44
Атака по времени выполнения .....	45
Атаки по потребляемой мощности .....	46
Другие пассивные атаки .....	47
<b>1.8. Активные атаки на шифраторы, использующие утечку данных</b>	
по побочным каналам .....	48
Виды воздействий на шифратор .....	48
Дифференциальный анализ на основе сбоев .....	50
Противодействие активным атакам.....	52
<b>1.9. Криptoаналитические атаки на связанных ключах</b> .....	53
Расширение ключа.....	53
«Классическая» атака на связанных ключах.....	54
Атакуемые алгоритмы.....	57
Другие возможные проблемы процедуры расширения ключа .....	59
<b>Глава 2. Новейшая история симметричного шифрования</b> .....	61
<b>2.1. Конкурс AES</b> .....	61
Алгоритмы — участники конкурса .....	63
Достоинства и недостатки алгоритмов-финалистов .....	65
<b>2.2. Конкурс NESSIE</b> .....	69
<b>Глава 3. Описание алгоритмов</b> .....	73
<b>3.1. Алгоритм ГОСТ 28147-89</b> .....	73
Описание алгоритма .....	73
Режимы работы алгоритма .....	75
Криптостойкость алгоритма .....	79
Анализ таблиц замен .....	79
Модификации алгоритма и их анализ .....	80
Анализ полнораундового алгоритма .....	81
Заключение.....	81
<b>3.2. Алгоритм Aardvark</b> .....	82
<b>3.3. Алгоритм AES (Rijndael)</b> .....	84
Структура алгоритма .....	84
Процедура расширения ключа .....	88
Расшифровывание .....	89

## Оглавление

Отличия AES от исходного алгоритма Rijndael .....	91
Первичная оценка криптостойкости алгоритма Rijndael.....	92
Криptoанализ алгоритма Rijndael в рамках конкурса AES .....	93
Криptoанализ алгоритма после конкурса AES.....	95
Заключение.....	97
<b>3.4. Алгоритм Akelarre.....</b>	<b>97</b>
Структура алгоритма.....	98
Процедура расширения ключа .....	101
Криptoанализ алгоритма.....	102
<b>3.5. Алгоритм Anubis .....</b>	<b>103</b>
Структура алгоритма.....	103
Процедура расширения ключа .....	107
Достоинства и недостатки алгоритма.....	108
<b>3.6. Алгоритмы Bear, Lion и Lioness .....</b>	<b>108</b>
Алгоритм Bear.....	109
Алгоритм Lion.....	111
Алгоритм Lioness.....	112
Варианты использования .....	113
Криptoанализ алгоритмов .....	114
<b>3.7. Алгоритмы BEAST и BEAST-RK .....</b>	<b>114</b>
Криптостойкость алгоритма BEAST .....	116
Алгоритм BEAST-RK.....	117
<b>3.8. Алгоритм Blowfish .....</b>	<b>118</b>
Структура алгоритма.....	119
Процедура расширения ключа .....	121
Достоинства и недостатки алгоритма.....	122
<b>3.9. Алгоритм Camellia .....</b>	<b>123</b>
Структура алгоритма.....	123
Таблицы замен .....	127
Расшифровывание .....	128
Процедура расширения ключа .....	129
Криptoанализ алгоритма Camellia .....	132
<b>3.10. Алгоритм CAST-128 .....</b>	<b>134</b>
<b>3.11. Алгоритм CAST-256 .....</b>	<b>135</b>
Основные характеристики и структура алгоритма .....	135
Процедура расширения ключа .....	139
Достоинства и недостатки алгоритма.....	141
<b>3.12. Алгоритм Crypton.....</b>	<b>141</b>
Основные характеристики и структура алгоритма .....	141

Процедура расширения ключа .....	145
Достоинства и недостатки алгоритма.....	147
3.13. CS-Cipher.....	148
Структура алгоритма.....	149
Расшифровывание .....	153
Процедура расширения ключа .....	154
Достоинства и недостатки алгоритма.....	155
3.14. Алгоритм DEAL .....	156
Основные характеристики и структура алгоритма .....	156
Процедура расширения ключа .....	157
Достоинства и недостатки алгоритма.....	160
3.15. Алгоритм DES и его варианты.....	160
История создания алгоритма .....	161
Основные характеристики и структура алгоритма .....	162
Процедура расширения ключа .....	166
Криптостойкость алгоритма DES .....	168
Продолжение истории алгоритма .....	171
Алгоритм Double DES .....	172
Алгоритм 2-key Triple DES .....	173
Алгоритм 3-key Triple DES .....	174
Режимы работы Double DES и Triple DES .....	175
Режимы работы с маскированием.....	178
Алгоритм Quadruple DES .....	181
Алгоритм Ladder-DES .....	181
Алгоритм DESX .....	184
Алгоритм DES с независимыми ключами раундов.....	185
Алгоритм GDES .....	186
Алгоритм RDES .....	188
Алгоритмы s <sup>2</sup> DES, s <sup>3</sup> DES и s <sup>5</sup> DES .....	190
Алгоритм Biham-DES.....	191
Алгоритмы xDES <sup>1</sup> и xDES <sup>2</sup> .....	193
Другие варианты алгоритма DES.....	195
Заключение.....	195
3.16. Алгоритм DFC .....	195
Основные характеристики и структура алгоритма .....	195
Процедура расширения ключа .....	198
Достоинства и недостатки алгоритма.....	199
3.17. Алгоритм E2 .....	201
Структура алгоритма.....	201
Почему алгоритм E2 не вышел в финал конкурса AES.....	206

## Оглавление

3.18. Алгоритм FEAL .....	206
История создания алгоритма .....	206
Структура алгоритма .....	207
Процедура расширения ключа .....	209
Почему FEAL не используется.....	211
3.19. Алгоритм FROG .....	212
Основные характеристики и структура алгоритма .....	212
Процедура расширения ключа .....	214
Форматирование ключа .....	216
Достоинства и недостатки алгоритма.....	218
3.20. Алгоритм Grand Cru .....	219
Структура алгоритма.....	219
Расшифровывание .....	225
Процедура расширения ключа .....	226
Достоинства и недостатки алгоритма.....	229
3.21. Алгоритм Hierocrypt-L1 .....	229
Структура алгоритма.....	229
Процедура расширения ключа .....	234
Достоинства и недостатки алгоритма.....	239
3.22. Алгоритм Hierocrypt-3 .....	239
Отличия от Hierocrypt-L1 .....	239
Процедура расширения ключа .....	241
Криptoанализ алгоритма.....	246
3.23. Алгоритм HPC .....	246
Основные характеристики алгоритма .....	246
Структура раунда.....	247
Процедура расширения ключа .....	250
Достоинства и недостатки алгоритма.....	253
3.24. Алгоритм ICE.....	254
История создания алгоритма .....	254
Структура алгоритма.....	254
Варианты алгоритма.....	257
Процедура расширения ключа .....	257
Криptoанализ алгоритма.....	259
3.25. Алгоритм ICEBERG .....	259
Структура алгоритма.....	259
Процедура расширения ключа .....	263
Криptoанализ алгоритма.....	265

3.26. Алгоритмы IDEA, PES, IPES.....	265
Основные характеристики и структура .....	266
Процедура расширения ключа .....	268
Криптостойкость алгоритма.....	268
3.27. Алгоритм KASUMI .....	270
Структура алгоритма.....	271
Процедура расширения ключа .....	275
Использование алгоритма.....	277
Криptoанализ алгоритма.....	279
3.28. Алгоритм Khazad.....	282
Структура алгоритма.....	282
Процедура расширения ключа .....	284
Модификация алгоритма .....	285
Криptoанализ алгоритма Khazad .....	287
3.29. Алгоритмы Khufu и Khafre.....	288
История создания алгоритмов .....	288
Структура алгоритма Khufu.....	289
Процедура расширения ключа и таблицы замен .....	290
Алгоритм Khafre .....	292
Сравнение алгоритмов .....	293
Алгоритм Snefru.....	294
Криptoанализ алгоритмов .....	294
3.30. Алгоритм LOKI97 .....	295
История создания алгоритма .....	295
Основные характеристики и структура алгоритма .....	296
Процедура расширения ключа .....	299
Почему LOKI97 не вышел в финал конкурса AES.....	300
3.31. Алгоритм Lucifer .....	301
Вариант № 1 .....	301
Вариант № 2 .....	304
Вариант № 3 .....	308
Вариант № 4 .....	310
Криptoанализ вариантов алгоритма .....	311
3.32. Алгоритм MacGuffin .....	312
Структура алгоритма.....	313
Процедура расширения ключа .....	315
Криptoанализ алгоритма.....	316
3.33. Алгоритм MAGENTA .....	316
Структура алгоритма.....	317
Достоинства и недостатки алгоритма.....	319

## Оглавление

3.34. Алгоритм MARS.....	319
Структура алгоритма .....	320
Процедура расширения ключа .....	326
Достоинства и недостатки алгоритма.....	328
3.35. Алгоритм Mercy .....	328
Структура алгоритма .....	328
Процедура расширения ключа .....	333
Криптостойкость алгоритма .....	334
3.36. Алгоритмы MISTY1 и MISTY2 .....	334
Структура алгоритма MISTY1 .....	334
Расшифровывание .....	342
Процедура расширения ключа .....	344
Алгоритм MISTY2 .....	345
Алгоритм KASUMI.....	346
Криptoанализ алгоритмов MISTY1 и MISTY2.....	347
3.37. Алгоритм Nimbus .....	348
Структура алгоритма .....	348
Процедура расширения ключа .....	349
Достоинства и недостатки алгоритма.....	351
3.38. Алгоритм Noekeon .....	351
Структура алгоритма.....	352
Расшифровывание .....	356
Процедура расширения ключа .....	357
Криptoанализ алгоритма.....	357
3.39. Алгоритм NUSH .....	357
Структура алгоритма .....	357
Расшифровывание .....	362
Процедура расширения ключа .....	363
Криптостойкость алгоритма .....	363
128-битный вариант.....	364
256-битный вариант.....	368
Криptoанализ 128- и 256-битного вариантов алгоритма.....	373
3.40. Алгоритм Q .....	374
Структура алгоритма.....	374
Криptoанализ алгоритма.....	375
3.41. Алгоритм RC2.....	375
Структура алгоритма .....	376
Процедура расширения ключа .....	378
Расшифрование .....	380
Криптостойкость алгоритма .....	381

3.42. Алгоритм RC5.....	381
Структура алгоритма.....	382
Процедура расширения ключа .....	385
Криptoанализ алгоритма.....	386
Варианты RC5 .....	387
Продолжение истории алгоритма .....	390
3.43. Алгоритм RC6.....	391
Структура алгоритма.....	391
Процедура расширения ключа .....	393
Достоинства и недостатки алгоритма.....	394
3.44. Алгоритмы SAFER K и SAFER SK .....	394
Структура алгоритма SAFER K-64 .....	395
Расшифрование .....	399
Процедура расширения ключа .....	400
Криptoанализ алгоритма SAFER K-64.....	402
Алгоритм SAFER K-128.....	403
Алгоритмы SAFER SK-64, SAFER SK-128 и SAFER SK-40 .....	404
Криptoанализ алгоритмов SAFER SK-64 и SAFER SK-128.....	408
3.45. Алгоритм SAFER+ .....	408
Структура алгоритма.....	408
Расшифрование .....	412
Процедура расширения ключа .....	413
Криptoанализ алгоритма SAFER+ .....	415
Единое расширение ключа SAFER+.....	416
3.46. Алгоритм SAFER++ .....	418
Структура алгоритма .....	418
64-битный вариант SAFER++.....	421
Криptoанализ алгоритма SAFER++.....	423
Заключение.....	424
3.47. Алгоритм SC2000 .....	424
Структура алгоритма .....	424
Расшифрование .....	428
Процедура расширения ключа .....	429
Криptoанализ алгоритма.....	432
3.48. Алгоритм SERPENT.....	432
Структура алгоритма .....	433
Расшифрование .....	439

## Оглавление

Процедура расширения ключа .....	441
Криптостойкость алгоритма .....	442
3.49. Алгоритм SHACAL .....	442
Алгоритм SHACAL-1 .....	442
Алгоритм SHACAL-0 .....	445
Алгоритм SHACAL-2 .....	446
Криптоанализ алгоритма SHACAL-1 .....	449
Криптоанализ алгоритма SHACAL-2 .....	450
3.50. Алгоритмы SHARK и SHARK* .....	451
3.51. Алгоритм Sha-zam .....	454
Структура алгоритма .....	454
Процедура расширения ключа .....	456
Криптостойкость алгоритма .....	456
3.52. Алгоритм Skipjack .....	457
Структура алгоритма .....	457
Криптостойкость алгоритма .....	461
3.53. Алгоритм SPEED .....	462
Структура алгоритма .....	462
Расшифровывание .....	465
Процедура расширения ключа .....	466
Достоинства и недостатки алгоритма .....	468
3.54. Алгоритм Square .....	469
Структура алгоритма .....	469
Процедура расширения ключа .....	472
Криптостойкость алгоритма .....	473
3.55. Алгоритмы TEA, XTEA и их варианты .....	474
Алгоритм TEA .....	474
Криптоанализ алгоритма TEA .....	476
Алгоритм XTEA .....	478
Криптоанализ алгоритма XTEA .....	480
Алгоритм Block TEA .....	480
Алгоритм XXTEA .....	481
3.56. Алгоритмы Twofish и Twofish-FK .....	483
Структура алгоритма .....	483
Процедура расширения ключа .....	486
Алгоритм Twofish-FK .....	491
Достоинства и недостатки алгоритма .....	492

<b>Приложение. Таблицы замен.....</b>	<b>493</b>
П1. Алгоритм Blowfish .....	493
П2. Алгоритм Camellia.....	498
П3. Алгоритм CAST-128.....	500
П4. Алгоритм CAST-256.....	510
П5. Алгоритм Crypton 0 .....	515
П6. Алгоритм DES .....	517
П7. Алгоритм KASUMI .....	519
П8. Алгоритм MARS.....	522
П9. Алгоритм s <sup>2</sup> DES.....	525
П10. Алгоритм s <sup>3</sup> DES.....	527
П11. Алгоритм s <sup>5</sup> DES.....	529
<b>Литература .....</b>	<b>531</b>

# Введение

Те или иные способы защиты информации использовались людьми на протяжении тысячелетий. Но именно в течение нескольких последних десятилетий криптография — наука о защите информации — переживает невиданный доселе прогресс, обусловленный, как минимум, двумя важными факторами:

- бурное развитие вычислительной техники и ее повсеместное использование привело к тому, что теперь в подавляющем большинстве случаев криптография защищает именно компьютерную информацию;
- тогда как раньше криптография была уделом государственных структур, сейчас криптографические методы защиты могут использовать (и используют) обычные люди и организации, хотя бы для защиты своей собственной переписки от посторонних глаз.

То же самое касается и разработки криптографических алгоритмов — известно множество алгоритмов шифрования, и далеко не все из них разработаны «в недрах спецслужб» или научными институтами — встречаются весьма удачные и широко используемые алгоритмы, разработанные частными лицами.

Эта книга посвящена алгоритмам шифрования, а именно, алгоритмам блочного симметричного шифрования, которые, на взгляд автора, являются наиболее широко используемыми в современном компьютерном мире. В книге собраны описания более 50 алгоритмов шифрования (и еще около ста их вариантов), которые автор считает наиболее широко используемыми, наиболее удачными, обладающими наиболее полезными свойствами или просто наиболее интересными по своей структуре или истории.

Как известно, шифрование решает одну из основных проблем защиты информации — а именно, проблему обеспечения конфиденциальности данных. Известно также, что во все времена предпринимались усилия получить именно ту информацию, которая по той или иной причине защищена от посторонних. Поэтому немалая часть книги посвящена криptoаналитическим атакам,

т. е. методам, с помощью которых можно попытаться вскрыть зашифрованную информацию.

Книга состоит из трех глав и приложения.

В *главе 1* приведена общая классификация криптографических алгоритмов, описана структура алгоритмов шифрования и наиболее часто используемые режимы их применения. Там же описаны различные виды криptoаналитических атак на алгоритмы шифрования и на их реализации в виде программных или аппаратных шифраторов.

*Глава 2* посвящена краткому обзору новейшей истории алгоритмов шифрования. Она описывает прошедшие в течение последнего десятилетия конкурсы по выбору стандартов шифрования США и Евросоюза. Именно эти конкурсы, по мнению автора, очень сильно повлияли на современное развитие криптографии и криptoанализа.

*Глава 3* содержит описание алгоритмов шифрования, т. е., в основном, следующие сведения:

- историю создания и использования алгоритмов;
- основные характеристики и структуру алгоритмов, включая подробное описание используемых в алгоритмах преобразований;
- достоинства и недостатки алгоритмов;
- описание известных методов вскрытия алгоритмов.

Алгоритмы снабжены схемами, существенно упрощающими понимание структуры конкретного алгоритма. Стоит отметить, что описание большинства алгоритмов достаточно подробно для разработки на их основе программного или аппаратного шифратора.

В *Приложение 1* вынесены громоздкие таблицы, нахождение которых в основном тексте помешало бы его восприятию читателями.

Стоит сказать и о том, что при написании книги не использована какая-либо информация ограниченного пользования — абсолютно все источники информации об алгоритмах (за исключением ряда общедоступных книг и журнальных статей) найдены автором на открытых ресурсах сети Интернет.

Автор выражает благодарность своим коллегам по работе в ООО «АНКАД», многие из которых оказывали автору различное содействие при создании этой книги.

Автор будет признателен читателям за любые замечания по этой книге, в том числе критические. Жду ваших писем по адресу [serg@panasenko.ru](mailto:serg@panasenko.ru).



# Глава 1

## Классификация алгоритмов шифрования и методов их вскрытия

В этой главе дана общая классификация криптографических алгоритмов, приведены структуры алгоритмов шифрования и наиболее часто используемые режимы их применения. Здесь же описаны различные виды криптоаналитических атак на алгоритмы шифрования и на их реализации в виде программных или аппаратных шифраторов.

### 1.1. Криптографические алгоритмы

Для начала обсудим, какие же типы криптографических алгоритмов используются для защиты информации. Криptoалгоритмы, прежде всего, делятся на три категории (рис. 1.1) [7, 263]:

- *бесключевые* алгоритмы, которые не используют каких-либо ключей в процессе криптографических преобразований;
- *одноключевые* алгоритмы, использующие в своих вычислениях некий секретный ключ;
- *двухключевые* алгоритмы, в которых на различных этапах вычислений применяются два вида ключей: секретные и открытые.

Рассмотрим вкратце основные типы криptoалгоритмов.

- *Хэш-функции*. Выполняют «свертку» данных переменной длины в последовательность фиксированного размера — фактически это контрольное суммирование данных, которое может выполняться как с участием некоторого ключа, так и без него. Такие функции имеют весьма широкое применение в области защиты компьютерной информации, например:
  - для подтверждения целостности любых данных в тех случаях, когда использование электронной подписи (см. далее) невозможно (например, из-за большой ресурсоемкости) или является избыточным;

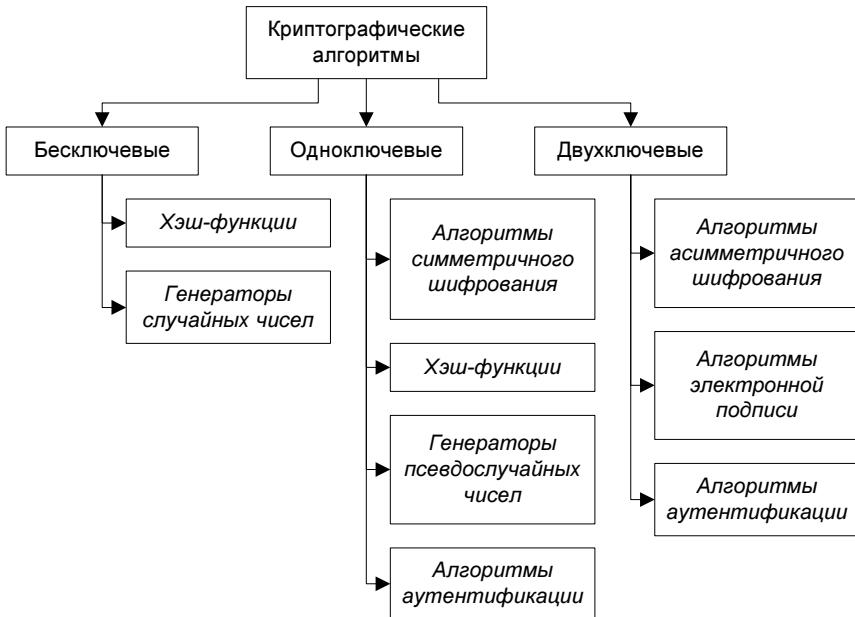


Рис. 1.1. Классификация криптографических алгоритмов

- в самих схемах электронной подписи — подписывается обычно хэш данных, а не все данные целиком;
  - в различных схемах аутентификации пользователей (см. далее).
- *Генераторы случайных чисел.* Случайные числа необходимы, в основном, для генерации секретных ключей шифрования, которые, в идеале, должны быть абсолютно случайными. Нужны они и для вычисления электронной цифровой подписи, и для работы многих алгоритмов аутентификации.
- *Алгоритмы симметричного шифрования* — алгоритмы шифрования, в которых для зашифровывания и расшифровывания используется один и тот же ключ, или ключ расшифровывания легко вычисляется из ключа зашифровывания и наоборот.

Симметричное шифрование бывает двух видов: блочное и потоковое.

- *Блоchное шифрование* — в этом случае информация разбивается на блоки фиксированной длины (например, 64 или 128 битов), после чего эти блоки поочередно шифруются. Причем в различных алгоритмах шифрования или даже в разных режимах работы одного и того же алгоритма блоки могут шифроваться независимо друг от друга или «со сцеплением» — когда результат зашифровывания текущего блока данных

## Классификация алгоритмов шифрования и методов их вскрытия

зависит от значения предыдущего блока или от результата зашифровывания предыдущего блока (см. разд. 1.4). Данная книга посвящена описанию криптоалгоритмов именно этой, наиболее обширной, категории алгоритмов шифрования.

- *Потоковое шифрование* — необходимо, прежде всего, в тех случаях, когда информацию невозможно разбить на блоки — скажем, некий поток данных, каждый символ которых должен быть зашифрован и отправлен куда-либо, не дожидаясь остальных данных, достаточных для формирования блока. Поэтому алгоритмы потокового шифрования шифруют данные побитно или посимвольно. Хотя стоит сказать, что некоторые классификации не разделяют блочное и потоковое шифрование, считая, что потоковое шифрование — это шифрование блоков единичной длины [263].

□ *Генераторы псевдослучайных чисел.* Не всегда возможно получение абсолютно случайных чисел — для этого необходимо наличие качественных аппаратных генераторов. Однако на основе алгоритмов симметричного шифрования можно построить очень качественный генератор псевдослучайных чисел.

□ *Алгоритмы аутентификации.* Позволяют проверить, что пользователь (или удаленный компьютер) действительно является тем, за кого себя выдает. Простейшая схема аутентификации — парольная — не требует наличия каких-либо криптографических ключей, но доказанно является слабой. А с помощью секретного ключа можно построить заметно более сильные схемы аутентификации. Пример аутентификации пользователя сервером (рис. 1.2):

**Этап 1.** Сервер генерирует случайное число

**Этап 2.** и отправляет его пользователю.

**Этап 3.** Пользователь зашифровывает полученное число секретным ключом и отправляет результат серверу.

**Этап 4.** Сервер расшифровывает полученные данные таким же секретным ключом

**Этап 5.** и сравнивает с исходным числом.

Равенство чисел означает, что пользователь обладает требуемым секретным ключом, т. е. ему удалось доказать свою легитимность.

□ *Алгоритмы асимметричного шифрования.* Применяют два вида ключей: открытый ключ для зашифровывания информации и секретный — для расшифровывания. Секретный и открытый ключи связаны между собой достаточно

сложным соотношением, главное в котором — легкость вычисления открытого ключа из секретного и невозможность (за ограниченное время при реальных ресурсах) вычисления секретного ключа из открытого при достаточно большой размерности операндов. Любая информация, зашифрованная общедоступным открытым ключом, может быть расшифрована только обладателем секретного ключа, из которого был вычислен данный открытый (рис. 1.3):

**Этап 1.** Пользователь В зашифровывает сообщение на открытом ключе пользователя А (который когда-либо передал его пользователю В).

**Этап 2.** Пользователь А расшифровывает сообщение своим секретным ключом.

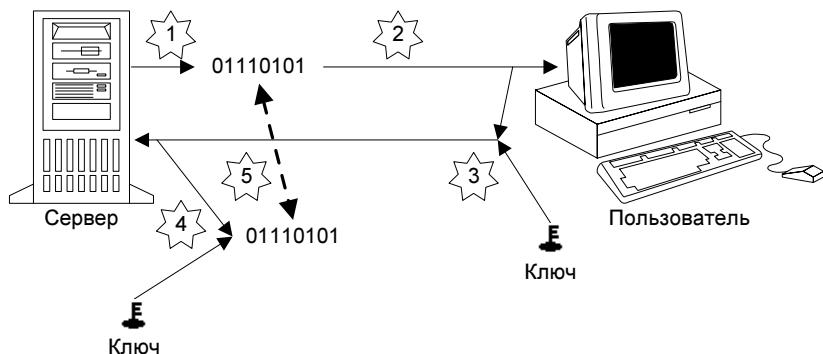


Рис. 1.2. Пример аутентификации пользователя сервером

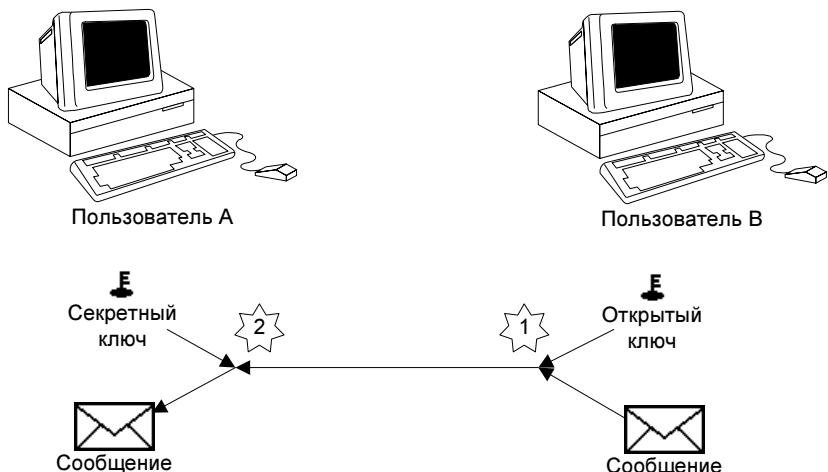


Рис. 1.3. Асимметричное шифрование

## Классификация алгоритмов шифрования и методов их вскрытия

- Алгоритмы электронной подписи. Используют секретный ключ для вычисления электронной цифровой подписи данных, а вычисляемый из него открытый — для ее проверки.

Как и асимметричное шифрование, это двухключевые алгоритмы с таким же простым вычислением открытого ключа из секретного и практической невозможностью обратного вычисления. Однако назначение ключей является совершенно другим:

- секретный ключ используется для вычисления электронной подписи;
- открытый ключ необходим для ее проверки.

При соблюдении безопасного хранения секретного ключа, никто, кроме его владельца, не в состоянии вычислить верную электронную подпись какого-либо электронного документа.

## 1.2. Категории алгоритмов шифрования

Шифрование — это основной криптографический метод защиты информации, обеспечивающий ее конфиденциальность [14, 21].

Шифрование информации — это преобразование открытых данных в зашифрованные и наоборот. Первая часть этого процесса называется зашифровыванием, вторая — расшифрованием.

Шифрование можно представить в виде следующих формул:

$$C = E_{k1}(M);$$

$$M' = D_{k2}(C).$$

где:

- $M$  (message) — открытая информация, в литературе по защите информации часто обозначается словосочетанием *открытый текст*;
- $C$  (cipher text) — полученный в результате зашифровывания *шифртекст* (или *криптоGRAMМА*);
- $E$  (encryption) — функция зашифровывания, выполняющая криптографические преобразования над исходным текстом;
- $k1$  (key) — параметр функции  $E$ , называемый *ключом зашифровывания*;
- $M'$  — информация, полученная в результате расшифровывания;
- $k2$  — ключ, с помощью которого выполняется расшифровывание информации;
- $D$  (decryption) — функция расшифровывания, выполняющая обратные зашифровыванию криптографические преобразования над шифртекстом.

В стандарте ГОСТ 28147-89 (см. разд. 3.1) понятие «ключ» определено следующим образом: «Конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования, обеспечивающее выбор одного преобразования из совокупности всевозможных для данного алгоритма преобразований» [4]. Иными словами, ключ является уникальным элементом, с помощью которого можно варьировать результат работы алгоритма шифрования: один и тот же открытый текст при использовании различных ключей будет зашифрован по-разному.

Для того чтобы результат операций зашифровывания и последующего расшифровывания совпал с исходным сообщением (т. е. для получения  $M' = M$ ), необходимо одновременное выполнение двух условий:

- функция расшифровывания  $D$  должна соответствовать функции зашифровывания  $E$ ;
- ключ расшифровывания  $k_2$  должен соответствовать ключу зашифровывания  $k_1$ .

При отсутствии верного ключа  $k_2$  получить исходное сообщение  $M' = M$  невозможно, если для зашифровывания использовался криптографически стойкий алгоритм шифрования. Понятие *криптоустойчивости* описано в разд. 1.5.

Алгоритмы шифрования можно разделить на две категории.

- Алгоритмы симметричного шифрования. Определяются следующим соотношением ключей зашифровывания и расшифровывания:

$$k_1 = k_2 = k,$$

т. е. функциями  $E$  и  $D$  используется один и тот же ключ шифрования.

- Алгоритмы асимметричного шифрования, в которых ключ зашифровывания  $k_1$  вычисляется из ключа  $k_2$  таким образом, что обратное вычисление невозможно; например, по следующей формуле:

$$k_1 = a^{k_2} \bmod p,$$

где  $a$  и  $p$  — параметры используемого алгоритма шифрования.

## 1.3. Структура алгоритмов симметричного шифрования

Подавляющее большинство современных алгоритмов шифрования работают весьма схожим образом: над шифруемым текстом выполняется некое преобразование с участием ключа шифрования, которое повторяется определенное число раз (раундов). При этом по виду повторяющегося преобразования алгоритмы шифрования принято делить на несколько категорий. Здесь также

## Классификация алгоритмов шифрования и методов их вскрытия

существуют различные классификации, приведу одну из них. Итак, по своей структуре алгоритмы шифрования классифицируются следующим образом [9, 50, 187, 284, 395].

### Алгоритмы на основе сети Фейстеля

Сеть Фейстеля (Feistel network) подразумевает разбиение обрабатываемого блока данных на несколько субблоков (чаще всего — на два), один из которых обрабатывается некоей функцией  $f$  и накладывается на один или несколько остальных субблоков. На рис. 1.4 приведена наиболее часто встречающаяся структура алгоритмов на основе сети Фейстеля.

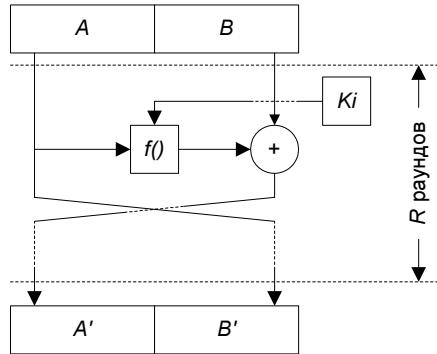


Рис. 1.4. Сеть Фейстеля

Дополнительный аргумент функции  $f$ , обозначенный на рис. 1.4 как  $K_i$ , называется *ключом раунда*. Ключ раунда является результатом обработки ключа шифрования процедурой расширения ключа, задача которой — получение необходимого количества ключей  $K_i$  из исходного ключа шифрования относительно небольшого размера (в настоящее время достаточным для ключа симметричного шифрования считается размер 128 битов). В простейших случаях процедура расширения ключа просто разбивает ключ на несколько фрагментов, которые поочередно используются в раундах шифрования; существенно чаще процедура расширения ключа является достаточно сложной, а ключи  $K_i$  зависят от значений большинства битов исходного ключа шифрования.

Наложение обработанного субблока на необработанный чаще всего выполняется с помощью логической операции «исключающее или» (Exclusive OR, XOR), как показано на рис. 1.4. Достаточно часто вместо XOR здесь используется сложение по модулю  $2^n$ , где  $n$  — размер субблока в битах. После наложения субблоки меняются местами, т. е. в следующем раунде алгоритма обрабатывается уже другой субблок данных.

Такая структура алгоритмов шифрования получила свое название по имени Хорста Фейстеля (Horst Feistel) — одного из разработчиков алгоритма шифрования Lucifer (см. разд. 3.31) и разработанного на его основе алгоритма DES (Data Encryption Standard) — бывшего (но до сих пор широко используемого) стандарта шифрования США (см. разд. 3.15). Оба этих алгоритма имеют структуру, аналогичную показанной на рис. 1.4. Среди других алгоритмов, основанных на сети Фейстеля, можно привести в пример отечественный стандарт шифрования ГОСТ 28147-89 (см. разд. 3.1), а также другие весьма известные алгоритмы: RC5 (см. разд. 3.42), Blowfish (см. разд. 3.8), TEA (см. разд. 3.55), CAST-128 (см. разд. 3.10) и т. д.

На сети Фейстеля основано большинство современных алгоритмов шифрования — благодаря множеству преимуществ подобной структуры, среди которых стоит отметить следующие:

- алгоритмы на основе сети Фейстеля могут быть сконструированы таким образом, что для зашифровывания и расшифровывания может использоваться один и тот же код алгоритма — разница между этими операциями может состоять лишь в порядке применения ключей  $K_i$ ; такое свойство алгоритма наиболее полезно при его аппаратной реализации или на платформах с ограниченными ресурсами; в качестве примера такого алгоритма можно привести ГОСТ 28147-89;
- алгоритмы на основе сети Фейстеля являются наиболее изученными — таким алгоритмам посвящено огромное количество криптоаналитических исследований, что является несомненным преимуществом как при разработке алгоритма, так и при его анализе.

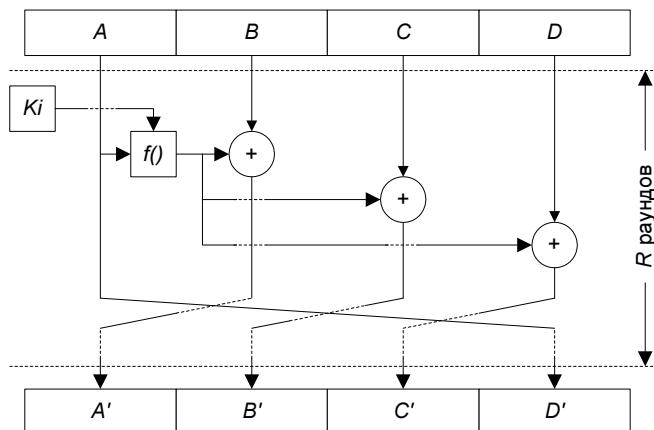


Рис. 1.5. Расширенная сеть Фейстеля

## Классификация алгоритмов шифрования и методов их вскрытия

Существует и более сложная структура сети Фейстеля, пример которой приведен на рис. 1.5. Такая структура называется *обобщенной* или *расширенной* сетью Фейстеля и используется существенно реже традиционной сети Фейстеля. Примером такой сети Фейстеля может служить алгоритм RC6 (см. разд. 3.43).

## Алгоритмы на основе подстановочно-перестановочных сетей

В отличие от сети Фейстеля, *SP-сети* (Substitution-permutation network, подстановочно-перестановочная сеть) обрабатывают за один раунд целиком шифруемый блок. Обработка данных сводится, в основном, к заменам (когда, например, фрагмент входного значения заменяется другим фрагментом в соответствии с таблицей замен, которая может зависеть от значения ключа  $K_i$ ) и перестановкам, зависящим от ключа  $K_i$  (упрощенная схема показана на рис. 1.6). Впрочем, такие операции характерны и для других видов алгоритмов шифрования, поэтому, на мой взгляд, название «подстановочно-перестановочная сеть» является достаточно условным.

SP-сети являются гораздо менее распространенными, чем сети Фейстеля; в качестве примера SP-сетей можно привести алгоритмы Serpent (см. разд. 3.48) или SAFER+ (см. разд. 3.45).

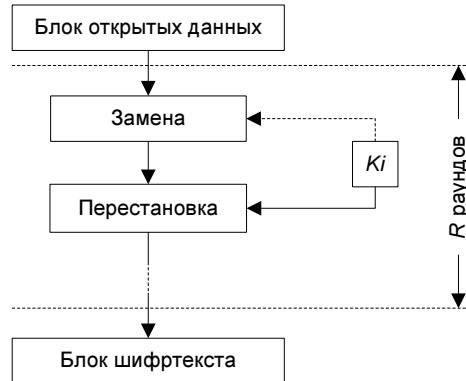


Рис. 1.6. SP-сеть

## Алгоритмы со структурой «квадрат»

Для структуры «квадрат» (Square) характерно представление шифруемого блока данных в виде двумерного байтового массива. Криптографические преобразования могут выполняться над отдельными байтами массива, а также над его строками или столбцами.

Эта структура получила свое название от алгоритма Square (*см. разд. 3.54*), который был разработан в 1996 г. Винсентом Риджменом (Vincent Rijmen) и Джоан Деймен (Joan Daemen) — будущими авторами алгоритма Rijndael (*см. разд. 3.3*), ставшего в США новым стандартом шифрования AES после победы на открытом конкурсе (*см. разд. 2.1*). Алгоритм Rijndael также имеет Square-подобную структуру; кроме того, в качестве примера можно привести алгоритмы SHARK (более ранняя разработка Риджмена и Деймен, *см. разд. 3.50*) и Crypton (*см. разд. 3.12*). Недостатком алгоритмов со структурой «квадрат» является их недостаточная изученность, что не помешало алгоритму Rijndael стать новым стандартом шифрования США.

## Алгоритмы с нестандартной структурой

Изобретательность безгранична, поэтому как-либо классифицировать все возможные варианты алгоритмов шифрования представляется сложным, т. е. существуют такие алгоритмы, которые невозможно причислить ни к одному из перечисленных типов. В качестве примера алгоритма с нестандартной структурой можно привести уникальный по своей структуре алгоритм FROG (*см. разд. 3.19*), в каждом раунде которого по достаточно сложным правилам выполняется модификация двух байтов шифруемых данных.

Строгие границы между описанными выше структурами не определены, поэтому достаточно часто встречаются алгоритмы, причисляемые различными экспертами к разным типам структур. Например, алгоритм CAST-256 (*см. разд. 3.11*) относится его автором к SP-сети, а многими экспертами называется расширенной сетью Фейстеля. Другой пример — алгоритм HPC (*см. разд. 3.23*), называемый его автором сетью Фейстеля, но относимый экспертами к алгоритмам с нестандартной структурой [50, 187].

## 1.4. Режимы работы алгоритмов

В 1980 г. в США был принят стандарт [151], определяющий режимы работы алгоритма DES — стандарта шифрования США (*см. разд. 3.15*). Можно сказать, что этот стандарт уточнял подробности реализации DES для различных применений.

В стандарте были определены следующие режимы работы алгоритма DES:

- электронная кодовая книга ECB (Electronic Code Book);
- сцепление блоков шифра CBC (Cipher Block Chaining);
- обратная связь по шифртексту CFB (Cipher Feed Back);
- обратная связь по выходу OFB (Output Feed Back).

## Классификация алгоритмов шифрования и методов их вскрытия

Рассмотренные далее режимы не привязаны к конкретному алгоритму — фактически любые алгоритмы блочного симметричного шифрования могут быть использованы (и используются) в данных режимах работы.

### Электронная кодовая книга

Наиболее простым из них является режим ECB. Суть его состоит в том (рис. 1.7), что каждый блок шифруемых данных «прогоняется» через алгоритм шифрования отдельно и независимо от других блоков:

$$C_i = E_k(M_i),$$

где:

- $E_k$  — функция зашифровывания на ключе  $k$ ;
- $M_i$  и  $C_i$  — блоки открытого текста и соответствующие им блоки шифртекста.

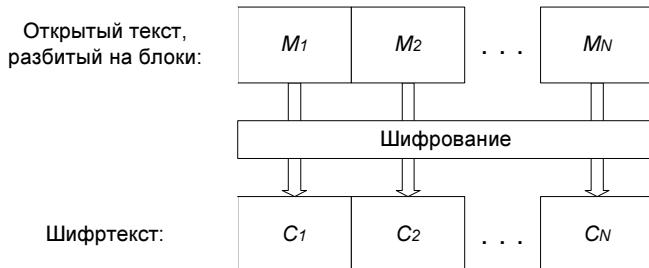


Рис. 1.7. Режим ECB

Аналогично выполняется и расшифровывание — блоки шифртекста обрабатываются поочередно и независимо.

Режим ECB плох тем, что если открытый текст содержит какое-либо количество блоков с одинаковым содержимым (например, большой массив, проинициализированный нулевыми или единичными битами), то и шифртекст будет содержать такое же количество одинаковых блоков. Это непозволительно, поскольку дает криptoаналитику информацию о структуре зашифрованной информации, что может существенно облегчить вскрытие алгоритма шифрования (т. е. получение открытого текста из зашифрованного или вычисление ключа шифрования). Поэтому режим ECB должен использоваться только для шифрования ключей друг на друге в многоключевых схемах [14]. Допускается также шифрование небольших фрагментов данных при условии их неповторяемости [263].

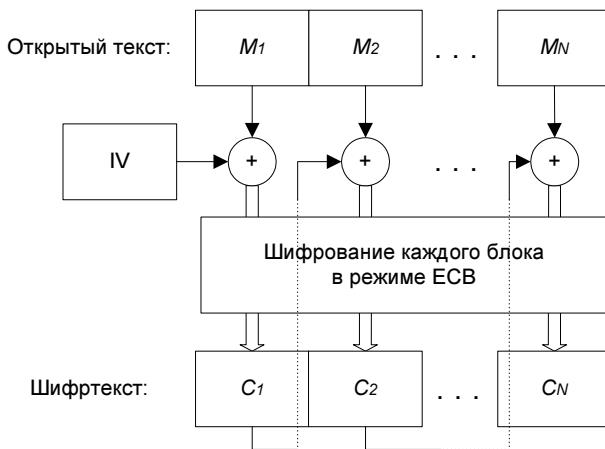
Есть и еще один недостаток — в режиме ECB алгоритм шифрует данные только поблочно. При необходимости, например, зашифровать алгоритмом DES 8 битов придется дополнить эти данные до 64-битного размера блока, зашифровать блок целиком, а при расшифровывании отбросить дополнительные биты. Такое не всегда возможно.

Достоинство режима ECB — простота реализации по сравнению с другими режимами. Однако, по словам известного криптолога Брюса Шнайера (Bruce Schneier), «из-за своей простоты в большинстве существующих коммерческих программ используется режим ECB, хотя этот режим наиболее уязвим к вскрытию» [28].

## Сцепление блоков шифра

Существенно более стойким является режим CBC. В этом режиме (рис. 1.8) перед шифрованием каждого  $i$ -го блока шифруемых данных выполняется его побитовое сложение по модулю 2 с результатом шифрования предыдущего,  $(i-1)$ -го, блока. То есть для режима CBC шифрование выглядит так:

$$C_i = E_k(M_i \oplus C_{i-1}).$$



**Рис. 1.8.** Режим CBC

Видно, что результат шифрования каждого блока зависит не только от содержимого шифруемого блока, но и от всех предыдущих блоков открытого текста. При шифровании первого блока открытого текста вместо результата

## Классификация алгоритмов шифрования и методов их вскрытия

зашифровывания предыдущего блока используется вектор инициализации (Initialization Vector, IV):

$$C_1 = E_k(M_1 \oplus IV).$$

Варьируя значение вектора инициализации, можно получать различные шифртексты для одинаковых открытых текстов. Естественно, при расшифровывании шифртекста, полученного в режиме CBC, вектор инициализации должен быть известен. Это справедливо и для рассмотренных далее режимов CFB и OFB.

Аналогично предыдущему режиму, данные, размер которых меньше 64-битного блока, придется перед обработкой дополнить до 64 битов.

Режим CBC используется непосредственно для зашифровывания данных, в том числе в больших объемах. Кроме того, последний блок шифртекста  $C_N$  может использоваться для контроля целостности сообщений, поскольку его значение зависит от содержимого всех блоков открытого текста, вектора инициализации и ключа:

$$C_N = f(M_1, M_2, \dots, M_N, IV, k).$$

## Обратная связь по шифртексту

Режим CFB более сложен в реализации, чем два предыдущих. Шифрование данных в этом режиме выполняется следующим образом (рис. 1.9):

1. Вектор инициализации IV записывается в регистр 1.
2. 64-битное содержимое регистра 1 зашифровывается, результат помещается в регистр 2.
3. Из регистра 2 выбираются  $L$  левых битов ( $1 \leq L \leq 64$ ), которые накладываются операцией XOR на  $L$ -битный блок шифруемого текста. Результат этого шага —  $L$ -битный блок шифртекста.
4. Содержимое регистра 1 сдвигается влево на  $L$  битов.
5. Регистр 1 дополняется справа  $L$ -битным блоком шифртекста. При необходимости продолжить шифрование данных шаги 2–5 повторяются в цикле до обработки всех шифруемых данных.

Фактически в режиме CFB алгоритм шифрования на основе вектора инициализации, ключа шифрования и предыдущих блоков шифртекста генерирует псевдослучайную последовательность, которая накладывается на открытый текст. Поскольку операция XOR обладает следующим свойством:

$$(x \oplus y) \oplus y = x$$

для любых значений  $x$  и  $y$ , то для расшифровывания следует сгенерировать такую же последовательность. Не имея ключа и значения вектора инициализации, такую же последовательность сгенерировать невозможно.

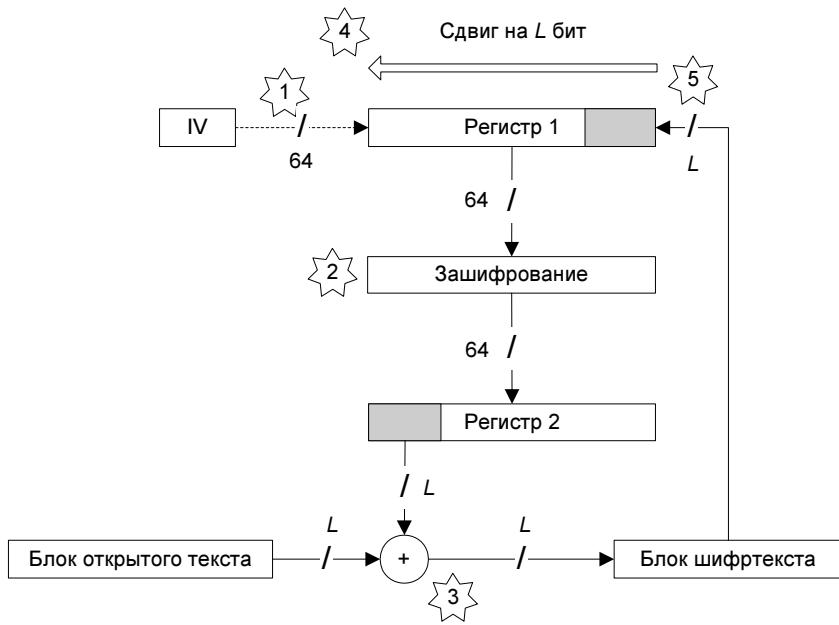


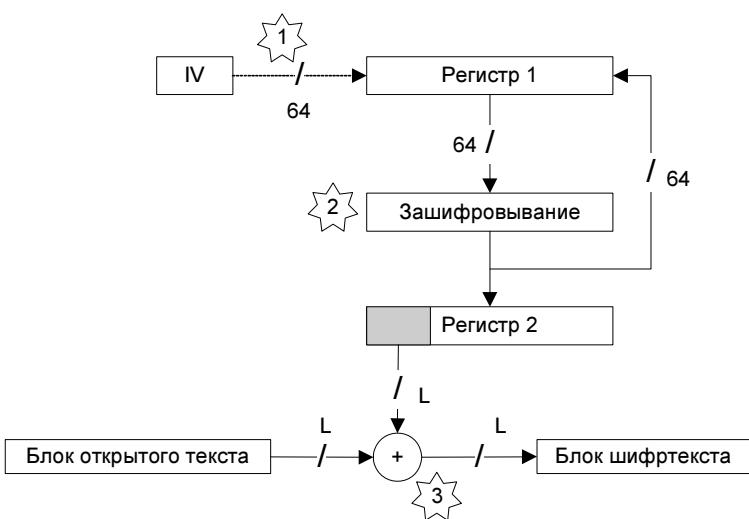
Рис. 1.9. Режим CFB

## Обратная связь по выходу

Режим шифрования OFB похож на предыдущий (рис. 1.10):

1. Вектор инициализации IV записывается в регистр 1.
2. Содержимое регистра 1 зашифровывается, результат помещается в регистры 1 и 2.
3. Из регистра 2 выбираются  $L$  левых битов, которые накладываются операцией XOR на  $L$ -битный блок шифруемого текста, в результате получается  $L$ -битный блок шифртекста. При необходимости продолжить шифрование шаги 2 и 3 повторяются в цикле.

Здесь приведена более поздняя версия режима OFB (описанная в стандарте ISO 10116); согласно [151] в регистр 1 «возвращался» не полностью 64-битный результат зашифровывания регистра 1, а младшие  $L$  битов (что больше напоминает режим CFB); исходный вариант режима OFB считается менее криптографически стойким, чем модифицированный [28, 263].



**Рис. 1.10. Режим OFB**

Существует и более простой режим OFB — *режим счетчика* [263], в котором обратная связь вообще отсутствует, а содержимое регистра 1 перед каждым его зашифровыванием увеличивается на 1.

Аналогично режиму CFB, для расшифровывания необходимо сгенерировать ту же последовательность и наложить ее на шифртекст.

Отличие от предыдущего режима лишь в том, что значение регистра 1 заменяется в цикле его же зашифрованным содержимым. Отсюда проистекает важное свойство режима OFB — накладываемая на шифруемый текст последовательность зависит только от значения вектора инициализации и ключа шифрования. То есть шифрующую последовательность можно сгенерировать заранее и, например, использовать ее в периоды максимальной нагрузки на шифрующий компьютер или устройство, восполняя последовательность в фоновом режиме. Такая возможность весьма важна для серверных компонент распределенных систем, поскольку позволяет существенно увеличить их пиковую производительность.

Еще одно важное отличие режима OFB от остальных состоит в том, что при возникновении ошибки в одном бите шифртекста после расшифровывания возникает ошибка только в одном бите расшифрованных данных, тогда как в остальных режимах ошибочно расшифруются один или два блока.

Стоит сказать и о том, что в режимах OFB и CFB алгоритмы блочного симметричного шифрования можно использовать в качестве потокового шифра (см. разд. 1.1), установив  $L$  равным размеру символов потока (например, 1 или 8).

## Другие режимы работы

Существуют и другие режимы работы, большинство которых являются незначительной модификацией описанных выше и применяются существенно реже [28].

Кроме того, при многократном шифровании используется множество специфических режимов работы. Многократное шифрование подробно рассмотрено на примере алгоритмов Double DES и Triple DES в разд. 3.15.

## 1.5. Атаки на алгоритмы шифрования

Рассмотрим стойкость алгоритмов шифрования к разнообразным криптоаналитическим действиям злоумышленника, направленным на их вскрытие [6, 263]. Эта характеристика является важнейшей для алгоритмов шифрования и называется *криптостойкостью*. Криптостойкость алгоритма обычно измеряется временем, которое необходимо на его вскрытие при неких фиксированных ресурсах, имеющихся в распоряжении у злоумышленника. Например, известно, что если у злоумышленника есть миллион процессоров, каждый из которых может перебрать миллион ключей алгоритма DES в секунду, то на полный перебор всех вариантов 56-битного ключа DES уйдет 20 часов. А это означает, что криптостойкости алгоритма DES недостаточно для защиты каких-либо долговременных данных, но его вполне можно применять для шифрования какой-либо срочной информации, раскрытие которой через те же 20 часов уже не страшно.

## Цели атак

Атакуя алгоритм шифрования, злоумышленник может преследовать следующие цели:

- нахождение открытого текста, имея его в зашифрованном виде, но не имея секретного ключа;
- нахождение самого секретного ключа.

В первом случае злоумышленнику необходимо какое-либо конкретное зашифрованное сообщение; достигнув же второй цели, т. е. получив секретный ключ, злоумышленник сможет читать все сообщения, зашифрованные на нем, что несравненно опаснее. Успешное получение злоумышленником секретного ключа называется *полным раскрытием* алгоритма шифрования.

Злоумышленник может иметь целью и нахождение ключа, эквивалентного секретному. *Эквивалентными* называются ключи, которые являются различ-

## Классификация алгоритмов шифрования и методов их вскрытия

ными, но приводят к одному и тому же результату шифрования. При успешном нахождении такого ключа злоумышленнику уже не нужен настоящий секретный ключ — все необходимое он расшифрует с помощью эквивалентного.

Стоит сказать и о том, что достижение перечисленных выше целей может оказаться неосуществимым при имеющихся у злоумышленника ресурсах. В этом случае злоумышленник может стремиться к *частичному раскрытию* секретного ключа или открытого сообщения. Частичное раскрытие секретного ключа может, например, помочь злоумышленнику значительно сузить область перебора секретных ключей.

## Классификация атак

Атаки на алгоритмы шифрования принято классифицировать в зависимости от того набора информации, который имеет злоумышленник перед осуществлением своей атаки. Прежде всего криptoаналитические атаки можно разделить на две категории.

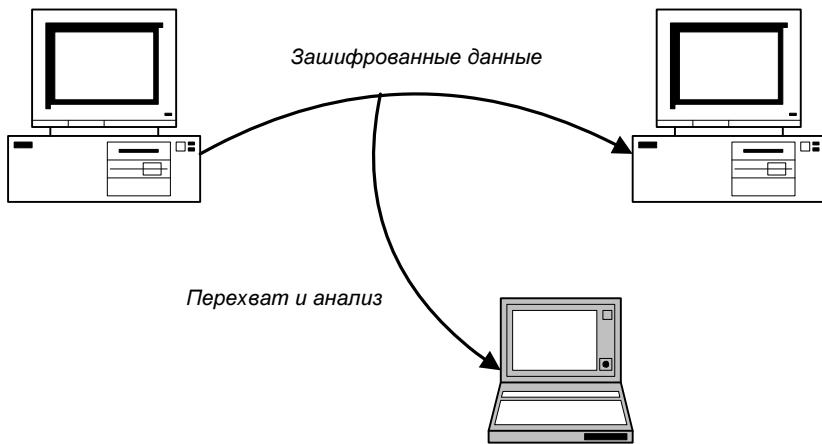


Рис. 1.11. Пассивный перехват зашифрованных данных

**Категория 1.** У криptoаналитика есть только возможность пассивного прослушивания некоего канала, по которому пересылаются зашифрованные данные (рис. 1.11). В результате у злоумышленника есть лишь набор шифртекстов, зашифрованных на определенном ключе. Такая атака называется атакой *с известным шифртекстом*. Она наиболее сложна, но этот вариант

атаки наиболее распространен, поскольку он является самым «жизненным» — в подавляющем большинстве реальных случаев криptoаналитик не имеет возможности получить больше данных.



**Рис. 1.12.** Активное воздействие на шифратор

**Категория 2.** Предполагает, что у криptoаналитика есть некое шифрующее устройство с прошитым ключом шифрования, который и является целью атаки. Таким устройством может быть, например, криптографическая смарт-карта. Криptoаналитик может выполнять с шифратором определенные (допускаемые шифратором и его техническим окружением, а также тактическими условиями осуществления атаки) действия для получения требуемой ему информации, например, «прогонять» через шифратор какие-либо открытые тексты для получения соответствующих им шифртекстов (рис. 1.12). В зависимости от данных, которые криptoаналитик может «добыть» у шифратора, существуют следующие виды атак [28, 263].

- *Атака с известным открытым текстом.* Предполагает наличие у криptoаналитика некоторого количества пар текстов, каждая из которых представляет собой открытый текст и соответствующий ему шифртекст.
- *Атака с выбранным открытым текстом.* У криptoаналитика есть возможность выбора открытых текстов для получения соответствующих им шифртекстов (как это может быть полезно криptoаналитику, будет рассмотрено далее).

## Классификация алгоритмов шифрования и методов их вскрытия

- *Адаптивная атака с выбором открытого текста.* Криптоаналитик может не просто выбирать открытые тексты для зашифровывания, но и делать это многократно, с учетом результатов анализа ранее полученных данных.
- *Атака с выбором шифртекста.* Криптоаналитик может выбирать шифртексты и, прогоняя их через шифратор, получать путем расшифровывания соответствующие им открытые тексты.
- *Адаптивная атака с выбором шифртекста.* По аналогии с описанными ранее атаками ясно, что криптоаналитик может многократно выбирать шифртексты для их расшифровывания с учетом предыдущих результатов.

Теоретически возможности криптоаналитика могут и не ограничиваться перечисленными выше; более серьезные варианты воздействия криптоаналитика на шифратор будут рассмотрены в разд. 1.8.

## Количественная оценка криптостойкости алгоритмов шифрования

Криптостойкость является количественной характеристикой алгоритмов шифрования — для вскрытия конкретного алгоритма шифрования при определенных условиях (в том числе определенным криптоаналитическим методом) требуется определенное количество ресурсов. Ресурсами в данном случае являются:

- количество информации, необходимое для осуществления атаки, — например, количество пар известных или выбранных текстов;
- время, необходимое для осуществления атаки; обычно измеряется в количестве тестовых операций шифрования атакуемым алгоритмом, выполнение которых при соблюдении остальных необходимых условий позволит, например, вычислить ключ шифрования;
- память, необходимая для хранения используемой при атаке информации; также является немаловажной характеристикой, поскольку многие атаки могут предъявлять весьма существенные требования к памяти.

Сообщество этих трех величин характеризует конкретную атаку на конкретный алгоритм шифрования. А лучшая (для которой требуется минимальный набор ресурсов) из возможных атак на алгоритм характеризует его криптостойкость.

Здесь и далее подразумевается, что сам алгоритм шифрования атакующему известен — неизвестен только ключ. Подавляющее большинство криптоаналитических методов (которые будут рассмотрены далее) основаны на доскональном знании криптоаналитиком атакуемого алгоритма. Существует и еще

одна немаловажная характеристика алгоритма шифрования — насколько шифртексты, полученные с его помощью, отличаются от случайной последовательности. Причем данная характеристика может быть выражена количественно в тех же трех описанных выше типах ресурсов.

## Криптоанализ модифицированных алгоритмов

Существует немало алгоритмов шифрования, которые являются криптографически стойкими. В известной работе [95] понятие *стойкого* (strong) алгоритма определено так:

- алгоритм является криптографически стойким, если не существует каких-либо методов его вскрытия, кроме метода «грубой силы» (brute force), который будет рассмотрен далее;
- кроме того, размер ключа алгоритма является достаточно большим для того, чтобы использование метода «грубой силы» стало невозможным при текущем уровне развития вычислительной техники.

Однако, например, бывает необходимо сравнить между собой два или более криптографически стойких алгоритма шифрования (как, например, на открытом конкурсе по выбору AES, нового стандарта шифрования США — см. разд. 2.1). В этом случае используют другую характеристику (скорее качественную, чем количественную) — запас криптостойкости (security margin).

Известно, что подавляющее большинство современных алгоритмов шифрования состоит из определенного количества раундов, в каждом из которых повторяются одни и те же (или схожие) преобразования над шифруемыми данными. Для определения запаса криптостойкости анализируют алгоритм с *усеченным* количеством раундов — т. е. модификацию исследуемого алгоритма, в которой количество раундов уменьшено по сравнению с конкретным предусмотренным в алгоритме количеством раундов. Запас криптостойкости можно определить как соотношение исходного количества раундов исследуемого алгоритма к максимальному количеству раундов его модификаций, не являющихся криптографически стойкими.

Другой вариант определения запаса криптостойкости — анализ модификаций исследуемого алгоритма с незначительными изменениями структуры раунда. Один из наиболее ярких примеров — вскрытие алгоритма Skipjack-3XOR при наличии всего  $2^9$  выбранных открытых текстов и соответствующих им шифртекстов выполнением всего около миллиона тестовых операций шифрования (см. разд. 3.52). По современным меркам, благодаря данной атаке Skipjack-3XOR можно считать весьма слабым алгоритмом, а ведь он отличается от известного и достаточно распространенного алгоритма шифрования

## Классификация алгоритмов шифрования и методов их вскрытия

Skipjack всего лишь тем, что из структуры последнего удалены 3 конкретных операции XOR из предусмотренных алгоритмом Skipjack 320 (!) подобных операций. Соответственно, были (однако недоказанные) предположения о недостаточном запасе криптостойкости у алгоритма Skipjack. Впрочем, в случае анализа алгоритмов с подобными модификациями запас криптостойкости можно рассматривать только как качественную характеристику, имеющую достаточно косвенное отношение к исследуемому алгоритму шифрования.

## 1.6. Криptoаналитические методы, используемые в атаках

В предыдущем разделе были классифицированы атаки на алгоритмы симметричного шифрования. Теперь опишем существующие криptoаналитические методы, переходя от более простых к более сложным.

### Метод «грубой силы»

Метод «грубой силы» (brute-force attack) предполагает перебор всех возможных вариантов ключа шифрования до нахождения искомого ключа.

Пусть размер ключа шифрования в битах равен  $b$ . Соответственно, существует  $2^b$  вариантов ключа. Криptoаналитик должен перебрать все возможные ключи, т. е. (если рассматривать  $b$ -битную последовательность как число) применить в качестве ключа значение 0, затем 1, 2, 3 и т. д. до максимально возможного ( $2^b - 1$ ). В результате ключ шифрования обязательно будет найден, причем в среднем такой поиск потребует  $2^b/2$ , т. е.  $2^{b-1}$  тестовых операций шифрования.

Понятно, что необходим какой-либо критерий корректности найденного ключа. С атакой с известным открытым текстом все достаточно просто — при тестировании каждого ключа  $K_x$  шифртекст  $C$  расшифровывается (в результате получается некое значение  $M'$ ) и сравнивается с соответствующим ему открытым текстом  $M$ ; совпадение  $M' = M$  говорит о том, что искомый ключ найден.

Несколько сложнее с атакой на основе шифртекста. В этом случае необходимо наличие какой-либо дополнительной информации об открытом тексте, например, следующей.

- Если открытый текст является осмысленным текстом на каком-либо языке, перехваченный шифртекст должен иметь достаточный размер для однозначного расшифровывания в осмысленный текст (минимально достаточ-

ный для этого размер называется *точкой единственности*). В основополагающей для современных симметричных криптосистем работе [26] точка единственности для английского языка теоретически определена как 27 букв. Если сообщение короче, то при переборе возможно появление нескольких различных осмысленных текстов, каждому из которых соответствует некий кандидат в искомые ключи. При невозможности перехвата дополнительных шифртекстов невозможно определить, какое из осмысленных сообщений является верным, если это не ясно из контекста.

- Если открытый текст состоит из бинарных данных, необходима какая-либо информация о том, что он из себя представляет. Если перехватывается архив, то при переборе ключей каждое значение  $M'$  должно рассматриваться как возможный заголовок архива. При другом потенциальном  $M$  это может быть PE-заголовок исполняемого файла для Windows, заголовок графического файла и т. д.
- Стоит отметить, что многие средства шифрования информации внедряют в формат зашифрованного объекта контрольную сумму открытого текста для проверки его целостности после расшифровывания (см. подробное описание в [14]). Это может быть, например, имитоприставка, вычисляемая согласно отечественному криптостандарту ГОСТ 28147-89 (см. разд. 3.1), или просто CRC32. Главное, что такая контрольная сумма может быть идеальным эталоном при криptoанализе, вполне подходящим для определения верного ключа.

Защита от атак, выполняемых методом «грубой силы», весьма проста — достаточно лишь увеличить размер ключа. Понятно, что увеличение размера ключа на 1 бит увеличит возможное количество ключей (и среднее время атаки) в 2 раза.

Несмотря на простоту атаки методом «грубой силы», существуют различные методы улучшения ее эффективности, например, следующие.

- Атака методом «грубой силы» простейшим образом распараллеливается: при наличии, скажем, миллиона компьютеров, участвующих в атаке, ключевое множество делится на миллион равных фрагментов, которые распределяются между участниками атаки [95].
- Скорость перебора ключей может быть во много раз увеличена, если в переборе участвуют не компьютеры общего назначения, а специализированные устройства. В [95] приводится пример микросхемы ORCA компании AT&T (технология ПЛИС), которая способна перебрать до 30 миллионов ключей DES (подробную информацию см. в разд. 3.15) в секунду. В той же работе [95] рассмотрено применение для перебора ключей специализированных микросхем, каждая из которых способна перебрать

## Классификация алгоритмов шифрования и методов их вскрытия

до 200 миллионов ключей DES в секунду. Следует учесть, что приведенные в [95] оценки даны на конец 1995 г. — сейчас перебор может осуществляться несравненно быстрее.

Все эти методы были опробованы на американском стандарте шифрования DES.

Понятно, что с развитием вычислительной техники требования к размеру ключа шифрования постоянно возрастают. В той же работе [95] был рекомендован 90-битный размер ключа в качестве абсолютно безопасного (причем с 20-летним запасом) на конец 1995 г. Сейчас подавляющее большинство алгоритмов шифрования используют ключи размером от 128 битов, что считается безопасным с примерно 80-летним запасом [131].

Современная вычислительная техника не позволяет «в лоб» атаковать 128-битный ключ полным перебором. Однако атаки методом «грубой силы» часто используются в контексте других атак — например, с помощью дифференциального криптоанализа (этот метод будет описан далее) сужается область возможных ключей, после чего выполняется перебор оставшихся вариантов (один из вариантов подобной атаки описан, например, в работе [59]).

## Атаки класса «встреча посередине»

Как было сказано выше, алгоритм шифрования считается стойким, если атака методом «грубой силы» против него неэффективна, а более быстрых возможностей вскрыть алгоритм не существует [95].

Любые методы, способные вскрыть алгоритм шифрования быстрее, чем полный перебор ключей, эксплуатируют те или иные конструктивные недостатки алгоритма или его реализации. Начнем обзор таких методов с атак класса *встреча посередине* (meet-in-the-middle).

Простейший пример подобной атаки — вскрытие любого алгоритма шифрования, представляющего собой двойное шифрование данных с помощью какого-либо «одинарного» алгоритма. Рассмотрим вкратце алгоритм Double DES (подробную информацию см. в разд. 3.15), представляющий собой двойное шифрование обычным алгоритмом DES (рис. 1.13):

$$C = DES_{k_{2/2}}(DES_{k_{1/2}}(M)),$$

где  $k_{1/2}$  и  $k_{2/2}$  — половины двойного ключа алгоритма Double DES, каждая из которых представляет собой обычный 56-битный ключ DES.

Double DES решает основную проблему алгоритма DES (56-битный ключ шифрования слишком короток, как было показано выше) — 112-битный ключ Double DES невозможно вскрыть полным перебором. Однако вскрытие

Double DES легко выполняется атакой на основе известных открытых текстов. Предположим, у криptoаналитика есть открытый текст  $M1$  и результат его зашифровывания —  $C1$ . Он может выполнить следующую последовательность действий (рис. 1.14):

1. Выполняется зашифровывание  $DES_{kx}(M1)$  на всем ключевом множестве ( $kx = 0\dots2^{56} - 1$ ) с записью результатов в некоторую таблицу.
2. Производится расшифровывание  $DES^{-1}_{ky}(C1)$  также на всем ключевом множестве; результаты расшифровывания сравниваются со всеми записями в таблице, сформированной на шаге 1.
3. Если какой-либо результат, полученный на шаге 2, совпал с одним из результатов шага 1, то можно предположить, что нужный ключ найден, т. е. найдены соответствующие совпадающему результату  $kx = k_{1/2}$  и  $ky = k_{2/2}$ .

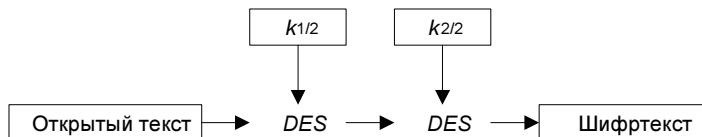


Рис. 1.13. Алгоритм Double DES

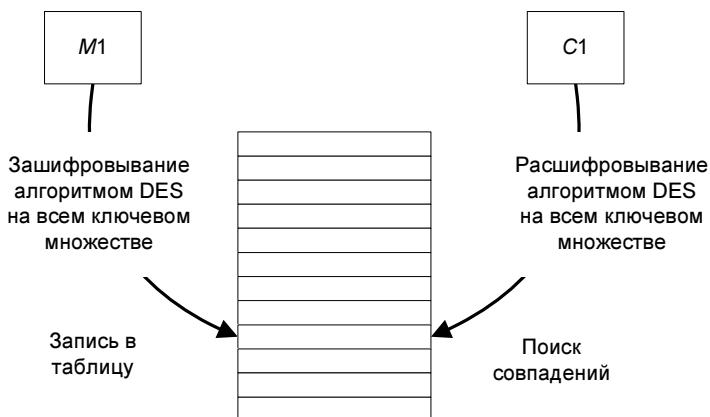


Рис. 1.14. Пример атаки «встреча посередине»

Следует учесть, что таких совпадений может быть много — порядка  $2^{48}$  согласно [3]. Для «уточнения» правильного ключа из этих примерно  $2^{48}$  вариантов достаточно наличия еще одной пары текстов:  $M2$  и  $C2$ . Криptoаналитик может использовать их абсолютно так же, как и первую пару ( $M1, C1$ ),

но перебор вариантов  $k_x$  и  $k_y$  осуществляется уже только по совпадениям первого перебора. В результате будет найден единственно верный ключ (вероятность повторного совпадения ничтожно мала — около  $2^{-16}$  [3]; в этом случае используется третья пара —  $M_3$  и  $C_3$  — при ее наличии).

В результате воздействия данной атаки сложность вычисления ключа Double DES всего в 2 раза выше, чем полный перебор ключей обычного DES, что несравненно меньше  $2^{112}$  возможных вариантов «двойного» ключа. Алгоритм Double DES, собственно, и не используется, а упоминается лишь в иллюстрациях к атаке «встреча посередине» (см., например, [3] или [14]).

Атака «встреча посередине» была изобретена в 1981 г. известными криптологами Ральфом Мерклем (Ralph C. Merkle) и Мартином Хеллманом (Martin E. Hellman) именно в применении к алгоритму Double DES [266]. Кроме того, эта атака (но в заметно более сложном исполнении) применима и к одному из вариантов «тройного» DES (Triple DES) [266].

Впоследствии атаки данного класса были неоднократно использованы для анализа различных алгоритмов шифрования; наиболее показательные примеры относятся к следующим алгоритмам шифрования:

- атака «встреча посередине» позволяет вычислить 192-битный ключ алгоритма DEAL (см. разд. 3.14) выполнением порядка  $2^{166}$  тестовых операций шифрования; для вычисления 256-битного ключа необходимо порядка  $2^{222}$  операций [47]; и то, и другое не является практически осуществимым, но доказывает невысокий запас криптостойкости алгоритма DEAL;
- столь же непрактичная атака возможна против еще одного участника конкурса AES — SAFER+ (см. разд. 3.45) — для его вскрытия необходимо  $2^{240}$  операций шифрования (при 256-битном ключе) [198].

Как видно, в отличие от Double DES, атака малоэффективна против более современных алгоритмов шифрования, что не удивительно. Однако, как и атака методом «грубой силы», атака «встреча посередине» нередко применима в комбинации с другими атаками, а также для оценки запаса криптостойкости модифицированных версий алгоритмов и алгоритмов с усеченным количеством раундов.

## Дифференциальный криptoанализ

Этот метод атак на алгоритмы шифрования был изобретен в 1990 г. известными израильскими криптологами Эли Бихамом (Eli Biham) и Эди Шамиром (Adi Shamir) и опубликован в их работе [75]. Однако не менее известный криптолог Брюс Шнайер в своей книге [28] утверждает, что дифференциальный

криптоанализ был открыт существенно раньше, но не появлялся в открытой печати. Тем не менее, именно Бихам и Шамир до сих пор считаются изобретателями дифференциального криптоанализа.

Работа [75] посвящена атаке с помощью дифференциального криптоанализа на алгоритм DES (см. разд. 3.15). Кратко опишем функцию раунда алгоритма DES  $f(b, k)$  (рис. 1.15):

1. Над 32-битным входным значением  $b$  выполняется расширяющая перестановка (EP), которая дает 48-битное значение  $be$ .

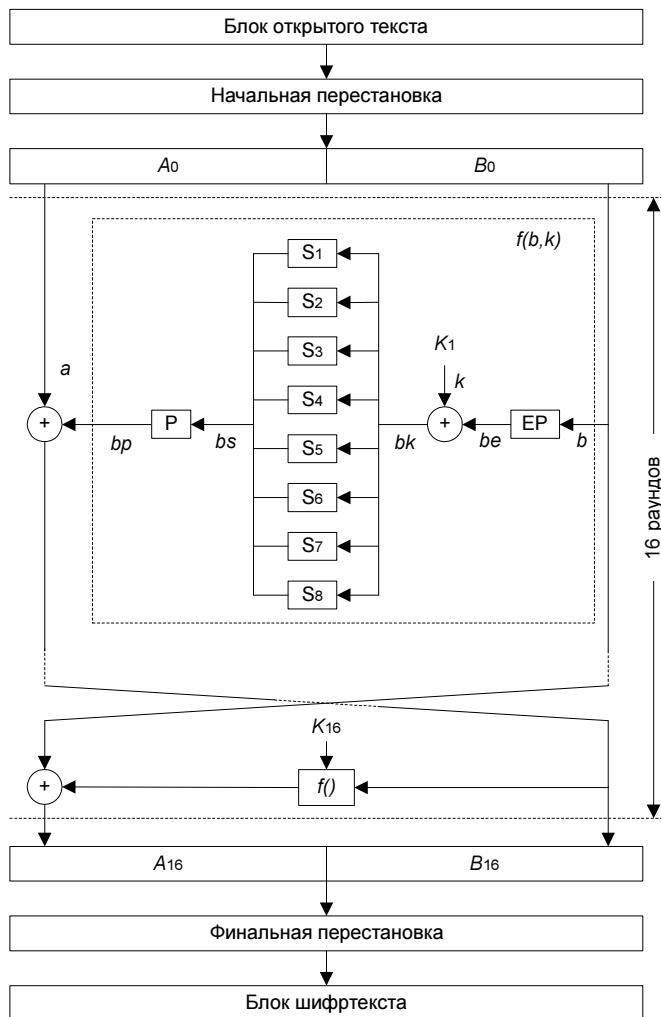


Рис. 1.15. Структура алгоритма DES

## Классификация алгоритмов шифрования и методов их вскрытия

2.  $be$  складывается с ключом раунда  $K_i$  операцией XOR (обозначим результат этой операции как  $bk$ ).
3.  $bk$  разбивается на 8 фрагментов по 6 битов, каждый из которых прогоняется через соответствующую таблицу замен ( $S_1 \dots S_8$ ). Каждая таблица включает в себя 4 строки, содержащие по 16 значений от 0 до 15. Входное значение интерпретируется следующим образом: два крайних бита формируют номер строки (от 0 до 3), из которой выбирается число, расположеннное в столбце, номер которого соответствует значению четырех остальных битов входа. Например, при двоичном входе 101100 (десятичное число 44) выбирается значение шестой ячейки второй строки.
4. Полученные в результате замен 4-битные значения объединяются в 32-битный субблок (обозначим его как  $bs$ ), после чего над ними выполняется еще одна перестановка ( $P$ ), которая завершает работу функции  $f$ .

Теперь рассмотрим собственно дифференциальный криптоанализ. Этот метод основан на анализе пар открытых текстов, между которыми существует определенная разность (difference). При анализе алгоритмов разность текстов может быть определена различным образом. Для DES разность открытых текстов  $M_1$  и  $M_2$  определяется как операция XOR между данными текстами:

$$\Delta = M_1 \oplus M_2.$$

Дифференциальный криптоанализ использует множество пар текстов с определенной разностью, анализ которых позволяет выделить некий ключ (или его фрагмент), который является искомым ключом либо однозначно, либо с наибольшей (по сравнению с другими возможными ключами) вероятностью.

Выполняется такой анализ следующим образом. Предположим, имеется два открытых текста, которые на входе в функцию  $f$  какого-либо раунда алгоритма имеют разность  $\Delta_b$  (рис. 1.15):

$$\Delta_b = b_1 \oplus b_2.$$

Разность  $\Delta_{be} = be_1 \oplus be_2$ , т. е. разность после обработки  $b_1$  и  $b_2$  расширяющей перестановкой  $EP$ , весьма легко определить, поскольку:

$$\Delta_{be} = EP(b_1) \oplus EP(b_2) = EP(b_1 \oplus b_2) = EP(\Delta_b).$$

Наложение фрагмента ключа операцией XOR вообще не меняет разность, т. е.:

$$\Delta_{bk} = \Delta_{be}.$$

Аналогично преобразованию  $EP$ , легко вычислить разность  $\Delta_{bp}$  после перестановки  $P$ :

$$\Delta_{bp} = P(bs_1) \oplus P(bs_2) = P(bs_1 \oplus bs_2) = P(\Delta_{bs}).$$

Таким образом, единственной операцией из выполняемых функцией  $f$ , существенно влияющей на значение разности, остается табличная замена. Значение  $\Delta_{bs}$  зависит не только от разности  $\Delta_{bk}$ , но и от конкретных входных значений  $bk_1$  и  $bk_2$ . Здесь-то и проявляется влияние наложенного предыдущей операцией ключа шифрования.

Как было сказано выше, таблицы замен меняют 6-битное входное значение на 4-битное. Это означает, что любой входной разности  $\Delta_{bk,n}$  (где  $n$  — номер таблицы) соответствует  $2^6 = 64$  возможных пар входных значений (обозначим их  $bk_{1,n}$  и  $bk_{2,n}$ ). Соответственно, любой выходной разности  $\Delta_{bs,n}$  соответствует  $2^4 = 16$  возможных пар  $bs_{1,n}$  и  $bs_{2,n}$ . Дифференциальный криптоанализ алгоритма DES эксплуатирует тот факт, что, во-первых, каждое конкретное значение  $\Delta_{bk,n}$  приводит не ко всем возможным 16 значениям  $\Delta_{bs,n}$ , а во-вторых, данные значения  $\Delta_{bs,n}$  имеют весьма различную вероятность. Бихам и Шамир в качестве примера рассматривают таблицу  $S_1$  и ее входную разность  $\Delta_{bk,1} = 34$  (здесь и далее значения разностей указываются в шестнадцатеричном виде). Возможные значения  $\Delta_{bs,1}$  и их вероятности (в количестве значений пар  $bk_{1,1}$  и  $bk_{2,1}$  из 64 возможных, которые приводят к данному значению  $\Delta_{bs,1}$ ) приведены в табл. 1.1 [75].

**Таблица 1.1**

Значение $\Delta_{bs,1}$	Вероятность
1	8
2	16
3	6
4	2
7	12
8	6
D	8
F	6
Остальные	0
Всего	64

Как с помощью всего этого можно определить ключ, рассмотрим на простейшем примере. Предположим, что в распоряжении криптоаналитика имеется пара текстов с  $\Delta_{bk,1} = 34$ . Кроме того, у криптоаналитика есть соответ-

## Классификация алгоритмов шифрования и методов их вскрытия

ствующие им шифртексты (снова предположим, что для их зашифровывания использовался однораундовый DES) с  $\Delta_{bs,1} = 4$ . Входные значения  $S1$  при таких условиях известны [75] — это значения 13 и 27 ( $bk_{1,1} = 13$ , а  $bk_{2,1} = 27$ , или наоборот; здесь также указаны шестнадцатеричные значения). Получается, что криптоаналитик знает значения  $be_1$  и  $be_2$  (поскольку ему известны открытые тексты), а также два варианта значений  $bk_{1,1}$  и  $bk_{2,1}$ . В результате криптоаналитик может простейшей операцией XOR вычислить 2 возможных варианта первых шести битов  $K_1$ . Выбрать из двух вариантов правильный криптоаналитику поможет вторая пара открытых текстов, если таковая у него есть. Даже если такой пары нет, криптоаналитик существенно сузил область возможных значений ключа: в  $2^5$  раз.

Понятно, что однораундовый DES не является реальным объектом для криптоанализа. Для вскрытия алгоритмов с большим количеством раундов используют *характеристики* — такие разности открытых текстов, которые с высокой вероятностью вызывают определенные разности в получаемых шифртекстах [28]. В качестве примера рассмотрим следующую трехраундовую характеристику алгоритма DES (рис. 1.16) [75]:

1. В первом раунде на вход функции  $f$  подаются два субблока с разностью  $\Delta_b = 60\ 00\ 00\ 00$ . Разность подобрана таким образом, чтобы после перестановки  $EP$  (наложение фрагмента ключа, как было сказано выше, не влияет на разность) она была нулевой на входе всех таблиц замен, кроме  $S1$ , на входе которой разность  $\Delta_{bk,1} = 0C$ . Эта разность обеспечивает выходную разность  $\Delta_{bs,1} = 0E$  с вероятностью  $14/64$ , которая после перестановки  $P$  (с учетом нулевой разности на выходе остальных таблиц) приводит к разности  $\Delta_{bp} = 00\ 80\ 82\ 00$ . Как видно, эта разность совпадает с разностью необработанных субблоков, т. е.  $\Delta_a$ ; в результате наложения результата функции  $f$  на необработанный субблок получается нулевая разность на входе второго раунда.
2. Нулевая разность  $\Delta_b$  на входе функции  $f$  во втором раунде дает также нулевую разность на выходе. Следовательно (рис. 1.16), разность на входе функции  $f$  третьего раунда полностью эквивалентна таковой в первом раунде.
3. Получается, что разности третьего раунда и их вероятности аналогичны первому раунду. Таким образом, данная характеристика обеспечивает после трех раундов выходную разность  $\Delta = 00\ 80\ 82\ 00\ 60\ 00\ 00\ 00$  (равную входной разности) со следующей вероятностью:

$$p = \frac{14}{64} * 1 * \frac{14}{64} = \left(\frac{14}{64}\right)^2 \approx 0,048.$$

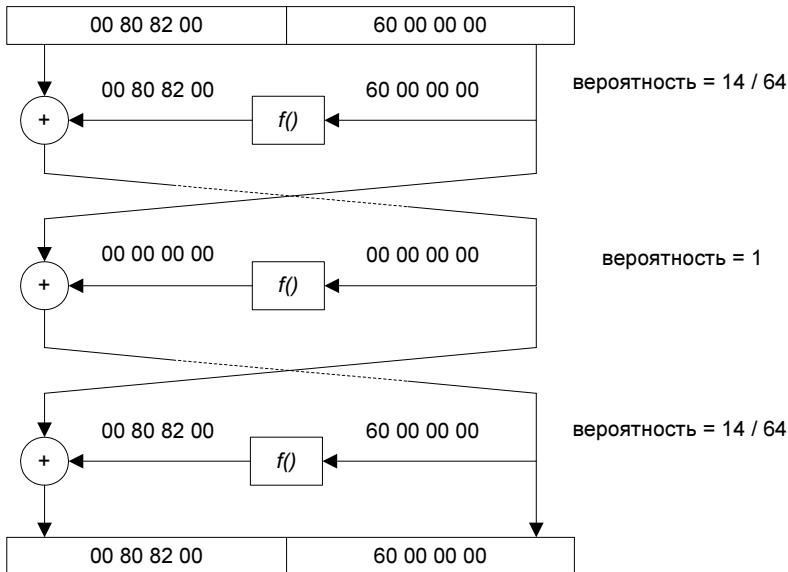


Рис. 1.16. Пример трехраундовой характеристики

Характеристики используются следующим образом (на примере *трехраундового DES* и приведенной выше характеристики):

1. Генерируется необходимое количество выбранных открытых текстов (пары которых соответствуют выбранной характеристике) и соответствующих им шифртекстов. Количество пар с требуемой разностью оценивается в [75] как «некоторое число» (several), умноженное на  $p^{-1}$ .
2. Формируется таблица, содержащая возможные варианты искомого фрагмента ключа шифрования (как было показано выше для однораундового DES). Верное значение должно повторяться для каждого анализируемого текста, поскольку именно оно использовалось для его зашифровывания.

Версии алгоритма с большим количеством раундов (включая полную 16-раундовую) вскрываются различными путями (см., например, [75] и [78]).

- Использованием характеристик, распространяющихся на большее количество раундов (но с меньшей вероятностью). Например, в [75] приведена 5-раундовая характеристика для DES с вероятностью 0,000095.
- Параллельное использование двух и более различных характеристик.
- Использование итеративных характеристик. Пример итеративной характеристики для DES приведен на рис. 1.17. Итеративные характеристики могут «размножаться» на требуемое количество раундов (конечно, также

## Классификация алгоритмов шифрования и методов их вскрытия

с потерей в вероятности), поскольку их структура подразумевает, что в выходной разности значения  $\Delta_a$  и  $\Delta_b$  меняются местами относительно входной разности (что необходимо для «размножения» характеристики, поскольку субблоки в DES меняются местами в конце раунда).

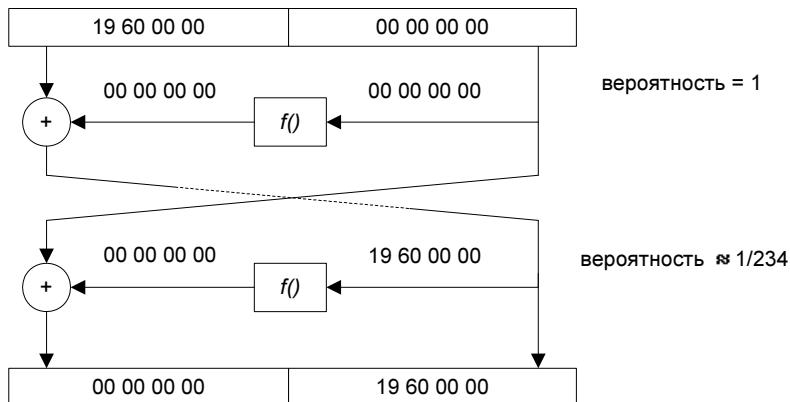


Рис. 1.17. Пример итеративной характеристики

Поиск характеристик с достаточной вероятностью является творческим процессом. На данный момент автору не известны какие-либо методы, позволяющие алгоритмизировать поиск характеристик для различных алгоритмов шифрования с высокой вероятностью. Стоит сказать и о том, что современные алгоритмы обладают весьма различной криптостойкостью к дифференциальному криптоанализу.

Если дифференциальный криптоанализ и был известен еще в середине 1970-х гг. (см. выше), то явно не всем разработчикам алгоритмов шифрования — многие из более поздних алгоритмов могут быть вскрыты этим методом. Рассмотрим наиболее показательные примеры.

- Известный алгоритм RC2 (см. разд. 3.4.1) вскрывается дифференциальным криптоанализом при наличии  $2^{59}$  выбранных открытых текстов [224].
- Не менее известный и широко используемый алгоритм RC5 (а именно, его основной вариант — RC5-32/12/16 — см. разд. 3.4.2) еще менее стоек к дифференциальному криптоанализу — он вскрывается при наличии  $2^{44}$  выбранных открытых текстов [90]. Существуют и другие варианты алгоритма RC5, также подверженные дифференциальному криптоанализу — подробную информацию см. в разд. 3.4.2.

- «Ускоренный» вариант алгоритма ICE — Thin-ICE (см. разд. 3.24) — вскрывается с вероятностью 25 % при наличии  $2^{23}$  выбранных открытых текстов; при увеличении количества текстов до  $2^{27}$  вероятность вскрытия алгоритма возрастает до 95 % [377].
- Один из вариантов алгоритма DES — Generalized DES (GDES) с 16 раундами и размером блока 256 битов — вскрывается при наличии всего 6 (!) выбранных открытых текстов [75]. Некоторые другие варианты алгоритма DES (DESX, DES с независимыми подключами,  $s^2$ DES и др.) также подвержены дифференциальному криптоанализу — подробную информацию см. в разд. 3.15.

Кроме того, известно множество работ, посвященных вскрытию дифференциальному криптоанализом версий различных алгоритмов с усеченным количеством раундов.

## Линейный криптоанализ

Линейный криптоанализ изобрел японский криптолог Мицуру Мацуи (Mitsuru Matsui). Предложенный им в 1993 г. метод изначально был направлен на вскрытие того же алгоритма DES [256]. Впоследствии линейный криптоанализ был распространен и на другие алгоритмы [326]. Однако есть источники (например, [395]), которые утверждают, что линейный криптоанализ был изобретен для вскрытия алгоритма FEAL (см. разд. 3.18) в 1992 г., а уже затем перенесен на DES.

Смысл линейного криптоанализа состоит в нахождении соотношений следующего вида [28, 256]:

$$P_{i1} \oplus P_{i2} \oplus \dots \oplus P_{ia} \oplus C_{j1} \oplus C_{j2} \oplus \dots \oplus C_{jb} = K_{k1} \oplus K_{k2} \oplus \dots \oplus K_{kc}, \quad (1.1)$$

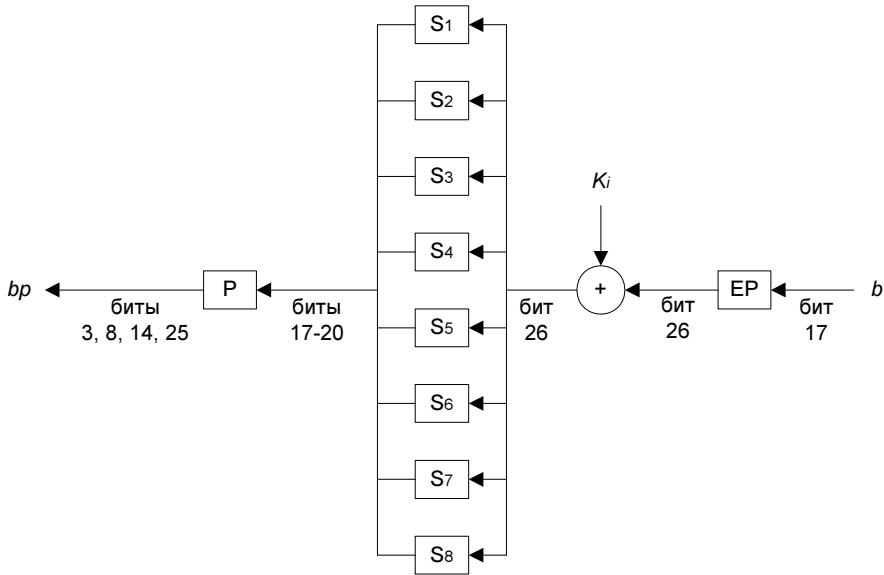
где  $P_n$ ,  $C_n$  и  $K_n$  —  $n$ -е биты открытого текста, шифртекста и ключа соответственно.

Для произвольно выбранных битов открытого текста, шифртекста и ключа вероятность  $P$  справедливости такого соотношения составляет около  $1/2$ . В том случае, если криптоаналитику удается найти такие биты, при которых вероятность  $P$  заметно отличается от  $1/2$ , данным соотношением можно воспользоваться для вскрытия алгоритма.

Так же, как и в дифференциальном криптоанализе, сначала криптоаналитик находит некое однораундовое соотношение, затем пытается распространить его на большее количество раундов, в результате находит соотношение для полнораундового варианта анализируемого алгоритма. Стоит отметить, что,

## Классификация алгоритмов шифрования и методов их вскрытия

в отличие от дифференциального криптоанализа, существуют алгоритмы поиска полезных соотношений; два подобных алгоритма изначально были описаны Мицуру Мацуи в [256], другие появились позже (см., например, [180]).



**Рис. 1.18.** Наиболее эффективное однораундовое соотношение для алгоритма DES

На рис. 1.18 представлено наиболее эффективное однораундовое соотношение для алгоритма DES [28, 256]. Данное соотношение использует то свойство таблицы  $S_5$ , что второй входной бит таблицы равен результату применения операции XOR над всеми четырьмя выходными битами с вероятностью  $3/16$  (т. е. смещение в  $5/16$  относительно вероятности  $1/2$ ), т. е. (применив нотацию, использованную ранее при описании дифференциального криптоанализа):

$$(bk_5)_2 = (bs_5)_1 \oplus (bs_5)_2 \oplus (bs_5)_3 \oplus (bs_5)_4,$$

что эквивалентно

$$bk_{26} = bs_{17} \oplus bs_{18} \oplus bs_{19} \oplus bs_{20}. \quad (1.2)$$

Шнайер в [28] пишет, что данное свойство таблицы  $S_5$  заметил еще Эди Шамир в 1985 г., однако именно Мацуи удалось его использовать для атаки на алгоритм. Дальнейшее расширение данного свойства таково:

1. Как показано на рис. 1.18, соотношение (1.2) распространяется на целый раунд алгоритма; в результате чего (с учетом перестановок) получается следующее однораундовое соотношение:

$$b_{17} \oplus bp_3 \oplus bp_8 \oplus bp_{14} \oplus bp_{25} = (K_i)_{26},$$

где  $i$  — номер раунда.

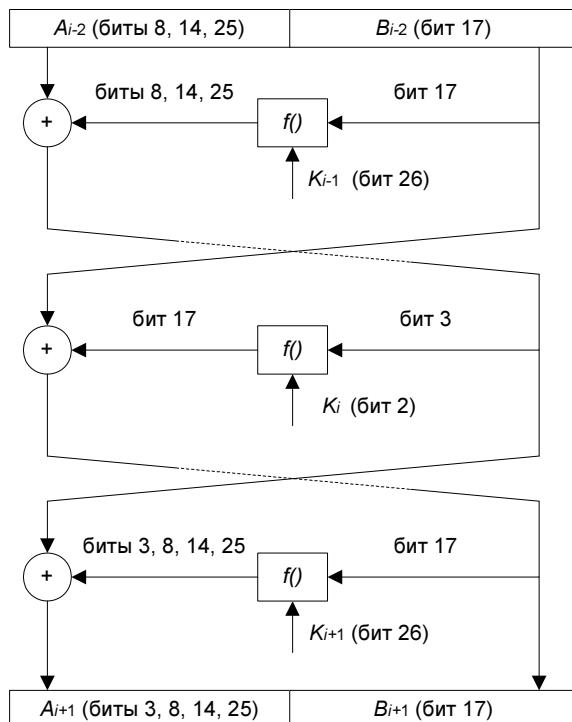


Рис. 1.19. Пример трехраундового соотношения

2. Весьма похоже на дифференциальный криптоанализ соотношение распространяется на несколько раундов. Пример такого соотношения показан на рис. 1.19 (соотношение (1.1) справедливо для тех битов, номера которых указаны на рисунке); его вероятность смешена относительно  $1/2$  на  $0,0061$  [28]. А для полнораундового DES известно соотношение, выполняющееся с вероятностью  $1/2 + 2^{-24}$  [86].

## Классификация алгоритмов шифрования и методов их вскрытия

3. С использованием максимально эффективного соотношения выполняется анализ имеющихся пар «открытый текст — шифртекст» с целью найти наиболее вероятные значения определенных битов ключа шифрования.

Весьма часто линейный криптоанализ используется в совокупности с атакой методом «грубой силы» — определенные биты ключа находятся с помощью линейного криптоанализа, после чего выполняется исчерпывающий поиск по возможным значениям остальных битов. Подобным образом алгоритм DES вскрывается при наличии  $2^{43}$  известных открытых текстов, что существенно эффективнее дифференциального криптоанализа [28, 263].

Стоит отметить и то, что для дифференциального криптоанализа требуются обычно выбранные открытые тексты, тогда как метод линейного криптоанализа «довольствуется» известными открытими текстами, что существенно увеличивает область применения этого метода. Однако, если это возможно, и в линейном криптоанализе в ряде случаев бывает весьма полезно использовать выбранные открытые тексты вместо известных. В частности, для DES существует методика, позволяющая существенно уменьшить количество требуемых данных и вычислений использованием выбранных открытых текстов [366].

Линейный криптоанализ имеет одно весьма полезнейшее свойство: при определенных обстоятельствах соотношение (1.1) может быть преобразовано к следующему:

$$C_{j1} \oplus C_{j2} \oplus \dots \oplus C_{jb} = K_{k1} \oplus K_{k2} \oplus \dots \oplus K_{kc}.$$

В этом соотношении полностью отсутствуют биты открытого текста, т. е. с помощью линейного криптоанализа можно построить атаку на основе только шифртекста (см. разд. 1.5), что еще больше расширяет область применения линейного криптоанализа, поскольку атака, для которой требуется только перехваченный шифртекст, является наиболее практической. В [256] Мацуи описывает такую атаку на полнорундовый DES, которая срабатывает при допущении, что соответствующий перехваченным шифртекстам открытый текст представляет собой текст на английском языке в ASCII-кодировке. Для этой атаки криптоаналитику потребуется порядка  $2^{54}$  шифртекстов.

Помимо DES и упомянутого выше алгоритма FEAL, известны и другие алгоритмы, подверженные линейному криптоанализу, например (отметим, что линейному криптоанализу подвержено существенно меньше известных алгоритмов, чем дифференциальному):

- линейный криптоанализ действует против алгоритма RC5 (см. разд. 3.42) в случае, если искомый ключ шифрования попадает в класс слабых ключей [179];

- алгоритмы NUSH (см. разд. 3.39) и Noekeon (см. разд. 3.38), известные своим участием в конкурсе криптоалгоритмов Евросоюза NESSIE (см. разд. 2.2), также вскрываются методом линейного криптоанализа [220, 398];
- некоторые варианты алгоритма DES (DESX, DES с независимыми подключами и Biham-DES) вскрываются линейным криптоанализом — подробную информацию см. в разд. 3.15.

## Метод бумеранга

Метод бумеранга (boomerang attack) предложен в 1999 г. известнейшим криптоаналитиком — профессором университета Беркли (Калифорния, США) Дэвидом Вагнером (David Wagner). Этот метод, фактически, является усилением дифференциального криптоанализа и состоит в использовании квартета (т. е. четырех вместо двух) открытых текстов и соответствующих им шифртекстов, связанных следующими соотношениями [386] (рис. 1.20):

1. Предположим, что алгоритм шифрования можно логически разделить на две примерно равные по трудоемкости части. Например,  $N$ -раундовый алгоритм с раундами схожей структуры делится на две части по  $N/2$  раундов.
2. Обозначим процедуру зашифровывания первой части алгоритма как  $E_0()$ . Для формирования квартета выберем два открытых текста  $P$  и  $P'$ , разность которых составляет некую величину  $\Delta$ . В результате обработки этих текстов функцией  $E_0()$  получаем разность  $\Delta^*$ , т. е. (считая, что разность определяется операцией XOR):

$$E_0(P) \oplus E_0(P') = \Delta^*.$$

3. Продолжим зашифровывание текстов  $P$  и  $P'$  путем применения к ним процедуры зашифровывания второй части алгоритма, которую обозначим  $E_1()$ . В результате получим два шифртекста —  $C$  и  $C'$ :

$$C = E_1(E_0(P));$$

$$C' = E_1(E_0(P')).$$

4. В этом случае разность между  $C$  и  $C'$  криптоаналитика не интересует, однако данные шифртексты используются для получения двух других выбранных шифртекстов:  $D$  и  $D'$ , связанных с предыдущими определенной разностью  $\nabla$ :

$$C \oplus D = C' \oplus D' = \nabla.$$

5. Затем формирование квартета продолжается в обратную сторону (от чего и происходит название метода): к  $D$  и  $D'$  применяется функция

## Классификация алгоритмов шифрования и методов их вскрытия

«полу-расшифровывания»  $E_1^{-1}()$ , в результате чего получается некая разность  $\nabla^*$ :

$$E_1^{-1}(D) \oplus E_0(P) = E_1^{-1}(D') \oplus E_0(P') = \nabla^*$$

(стоит отметить, что  $E_1^{-1}(C) = E_0(P)$ ).

6. «Окончательное» расшифровывание шифртекстов  $D$  и  $D'$  дает в результате два открытых текста, обозначаемых  $Q$  и  $Q'$ :

$$Q = E_0^{-1}(E_1^{-1}(D));$$

$$Q' = E_0^{-1}(E_1^{-1}(D')).$$

При этом Вагнер в [386] доказывает, что разность между полученными таким образом открытыми текстами  $Q$  и  $Q'$  полностью аналогична разности между исходными текстами  $P$  и  $P'$ , т. е.:  $\Delta$

$$Q \oplus Q' = \Delta.$$

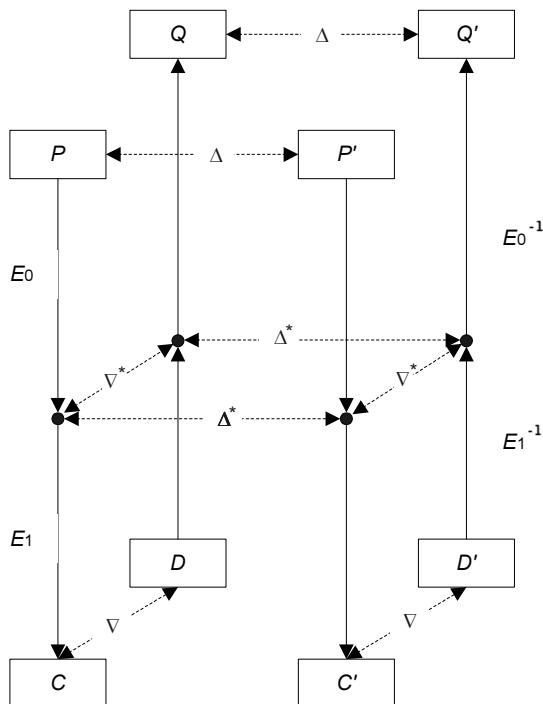


Рис. 1.20. Метод бумеранга

Описанные выше соотношения четырех открытых текстов и четырех шифртекстов позволяют атаковать некоторые из алгоритмов, стойких к «классическим»

скому» дифференциальному криптоанализу. В ряде случаев (см. [386]) метод бумеранга позволяет существенно сократить объем требуемых для атаки данных (по сравнению с дифференциальным криптоанализом). Кроме того, атака применима к алгоритмам с гетерогенной структурой раундов.

Недостаток метода по сравнению с дифференциальным криптоанализом весьма серьезен: как видно из приведенного выше алгоритма формирования квартета, метод бумеранга представляет собой атаку с аддитивным выбором открытых текстов и шифртекстов, т. е. атаку, наиболее сложно применимую на практике.

Метод бумеранга может быть применен против достаточно большого количества известных алгоритмов шифрования, среди которых стоит отметить алгоритмы-участники конкурса AES (см. разд. 2.1) CAST-256 (см. разд. 3.11), MARS (см. разд. 3.34) и Serpent (см. разд. 3.48). Впрочем, два последних алгоритма вскрываются только в вариантах с уменьшенным количеством раундов [195].

## Сдвиговая атака

*Сдвиговая атака* (slide attack) предложена в 1999 г. Алексом Бирюковым (Alex Biryukov) и упоминавшимся выше Дэвидом Вагнером [92].

Уникальность атаки состоит в том, что ее успешность не зависит от количества раундов атакуемого алгоритма. Однако с помощью сдвиговой атаки можно вскрыть только те алгоритмы, раунды которых являются идентичными. Предположим, есть некий многораундовый шифр, в каждом раунде которого выполняется функция  $F(x, k)$ , где  $x$  — выходное значение предыдущего раунда (или открытый текст для первого раунда алгоритма), а  $k$  — ключ раунда; предположим также, что ключи всех раундов являются идентичными. В этом случае для атаки используется следующая пара открытых текстов:

- некий случайно выбранный текст  $P$ ;
- второй текст —  $P'$ , представляющий собой результат однораундового преобразования текста  $P$ , т. е.:

$$P' = F(P, k).$$

Ясно, что соответствующие таким открытым текстам шифртексты (обозначим их  $C$  и  $C'$  для  $P$  и  $P'$  соответственно) связаны между собой аналогичным соотношением (рис. 1.21):

$$C' = F(C, k).$$

Как видно на рис. 1.21, открытые тексты и соответствующие им шифртексты «сдвинуты» относительно друг друга на 1 раунд атакуемого алгоритма, что

## Классификация алгоритмов шифрования и методов их вскрытия

и предопределило название атаки (которое, как сказано в [92], предложено Брюсом Шнайером).

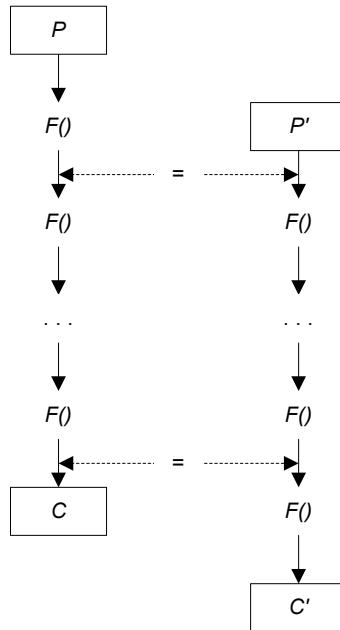


Рис. 1.21. Сдвиговая атака

Множество алгоритмов шифрования построено по тому принципу, что функция раунда является слабой (но легко реализуемой и нетребовательной к ресурсам), а криптостойкость алгоритма шифрования определяется ее многократным повторением. Поэтому, имея две пары текстов ( $P, P'$  и  $C, C'$ ), связанные между собой лишь одним раундом шифрования, криptoаналитик способен (с помощью различных методов, описанных в этой главе) получить значение ключа раунда  $k$ . В данном случае это можно считать полным раскрытием алгоритма.

Соответственно, основная проблема атакующего состоит в нахождении двух открытых текстов,  $P$  и  $P'$ , связанных приведенным выше соотношением (предположим, что у злоумышленника есть доступ к шифратору с искомым ключом; однако невозможно «заставить» шифратор выполнить лишь 1 раунд шифрования). Авторы атаки предлагают осуществлять поиск нужной пары следующим образом:

1. Зафиксировав некоторое значение  $P$ , выполнить полный перебор остальных открытых текстов до нахождения нужного значения  $P'$ . Согласно

парадоксу дней рождения (birthday paradox, подробно описан, например, в [28]), требуемый текст  $P'$  будет найден после перебора порядка  $2^{n/2}$  текстов, где  $n$  — размер шифруемого блока в битах.

2. «Правильная» пара  $P$  и  $P'$  может быть определена следующим образом:

- для  $P$  и каждого текста — кандидата в  $P'$  (обозначим такой как  $P^*$ ) — получаются соответствующие шифртексты —  $C$  и  $C^*$ ;
- из соотношения  $P^* = F(P, k_1)$  определяется некое значение ключа раунда  $k_1$ ; соответственно, из соотношения  $C^* = F(C, k_2)$  определяется некое значение  $k_2$ ;
- совпадение  $k_1$  и  $k_2$  означает, что требуемое значение  $P' = P^*$  найдено, а  $k_1 = k_2$  и есть искомое значение ключа раунда  $k$ .

Еще раз стоит отметить, что количество раундов алгоритма в данном случае никак не влияет на успешность его вскрытия.

С помощью сдвиговой атаки был полностью раскрыт алгоритм шифрования TREYFER, предложенный в 1997 г. для применения в смарт-картах и других устройствах с ограниченными ресурсами. Данный алгоритм имеет 32 раунда шифрования и 64-битный размер блока; в каждом раунде абсолютно идентично используется 64-битный ключ шифрования. Для атаки на TREYFER требуется около  $2^{32}$  известных открытых текстов и  $2^{44}$  тестовых операций шифрования [92]. Кроме того, сдвиговая атака была применена к модифицированным вариантам известных алгоритмов DES (вариант 2K-DES) и Blowfish (см. разд. 3.8) [92]; однако атака нераспространима на их полноценные версии.

Несколько позже (в 2000 г.) вышла еще одна работа Бирюкова и Вагнера [91], в которой сдвиговая атака была усиlena и распространена на алгоритмы, в которых функции раундов не совсем идентичны, но обладают существенным сходством. Усиленная атака была использована для взлома нескольких вариантов алгоритма DES, а также 20-раундового модифицированного отечественного стандарта шифрования ГОСТ 28147-89 (см. разд. 3.1).

## Метод интерполяции

*Метод интерполяции* (interpolation attack) предложен в 1997 г. датскими криптологами Томасом Якобсеном (Thomas Jakobsen) и Ларсом Кнудсеном (Lars R. Knudsen) и описан в их работе [185].

Атаке подвержены алгоритмы шифрования, использующие достаточно простые алгебраические операции, в результате чего криptoаналитик может построить некий полином, определяющий соотношение между шифртекстом и открытым текстом.

## Классификация алгоритмов шифрования и методов их вскрытия

Авторы атаки доказали, что если число ненулевых коэффициентов полинома (обозначим его  $n$ ) не превышает  $2^m$ , где  $m$  — размер блока шифруемых данных в битах, то возможна атака на этот алгоритм шифрования, которая находит алгоритм, эквивалентный расшифровыванию (или зашифровыванию) данных искомым секретным ключом. Для такой атаки требуется выполнение  $n$  тестовых операций шифрования при наличии  $n$  известных открытых текстов.

В [185] Кнудсен и Якобсен показали, как с помощью метода интерполяции вскрыть алгоритмы KN и PURE (последний из алгоритмов, однако, изобретен ими же для пояснения на нем данного метода), а также модифицированного варианта алгоритма SHARK (см. разд. 3.50).

## Невозможные дифференциалы

В 1998 г. изобретателями дифференциального криptoанализа Эли Бихамом и Эди Шамиром совместно с упоминавшимся выше Алексом Бирюковым были опубликованы две работы ([63] и [64]), в которых был предложен принципиально новый вариант дифференциального криptoанализа, использующий *невозможные дифференциалы* (*impossible differentials*).

Если сравнивать невозможные дифференциалы с классическим дифференциальным криptoанализом (см. выше), то видно, что использование невозможных дифференциалов представляет собой дифференциальный криptoанализ «от противного»: данный метод использует дифференциалы с нулевой вероятностью. Процесс вскрытия алгоритма с помощью невозможных дифференциалов кратко описывается так:

1. Выбирается нужное количество пар открытых текстов с требуемой разностью; получаются соответствующие им шифртексты.
2. Выполняется анализ полученных данных, в рамках которого все варианты ключа шифрования, приводящие к невозможным дифференциалам, считаются неверными и отбрасываются.
3. В результате остается единственный возможный вариант ключа или некое подмножество ключевого множества, не приводящие к невозможным дифференциалам; в случае подмножества может быть использован его полный перебор для нахождения верного ключа.

Дифференциалы с нулевой вероятностью могут быть заменены дифференциалами с минимальной вероятностью — действия криptoаналитика при этом аналогичны описанным выше [366].

Невозможные дифференциалы могут быть применены для вскрытия усеченных (по количеству раундов) версий таких известных алгоритмов шифрования, как IDEA (см. разд. 3.26), Khufu и Khafre (см. разд. 3.29), Skipjack (см. разд. 3.52), MISTY и KASUMI (см. разд. 3.36 и 3.27).

## Заключение

Это далеко не все из применяемых сейчас криптоаналитических методов. Однако автор надеется, что дал представление об основных атаках на алгоритмы блочного симметричного шифрования.

### 1.7. Атаки на шифраторы, использующие утечку данных по побочным каналам

Все описанные в предыдущем разделе атаки объединяло одно — они использовали некоторые уязвимости структуры атакуемого алгоритма. То есть данные атаки, фактически, рассматривали алгоритм шифрования только с теоретической точки зрения — как некое математическое преобразование, с помощью которого из открытого текста получается шифртекст.

Однако не стоит забывать и о том, что для практического использования алгоритмы шифрования так или иначе должны быть реализованы в некоторой компьютерной программе или аппаратном шифраторе. И в том, и в другом случае у криптоаналитика появляются дополнительные возможности для атак на алгоритмы шифрования. Данные возможности проистекают из того факта, что любой шифратор в процессе своей работы дает криптоаналитику возможность для съема различной побочной информации. Причем интенсивность утечки такой информации и ее полезность для криптоаналитика зависят как от структурных особенностей самого алгоритма шифрования, так и от особенностей его программной или аппаратной реализации: насколько разработчик шифратора старался уменьшить возможность утечки информации и вообще учитывал возможность подобных атак.

Все атаки, использующие утечку данных по побочным каналам (*side-channel attacks*), можно разделить на две категории [307]:

- пассивные (*passive*) атаки, в процессе реализации которых криптоаналитик лишь пассивно «прослушивает» шифратор с целью получения необходимой информации для ее последующего анализа;
- активные (*active*) атаки, для проведения которых криптоаналитик производит различные воздействия на шифратор с целью заставить его работать в нештатном режиме, вследствие чего шифратор может выдавать существенно больше полезной информации по побочным каналам.

В этом разделе рассмотрим подробно пассивные атаки. Активные атаки будут рассмотрены далее.

## Атака по времени выполнения

Начало подобным атакам положили работы Пола Кохера (Paul Kocher), прежде всего, статья [228], в которой была представлена атака по времени выполнения (timing attack). Данная атака использует тот факт, что на некоторых аппаратных платформах для выполнения определенных операций требуется различное количество тактов процессора. Результат — различное время выполнения операций. Соответственно, криptoаналитик путем высокоточных замеров времени выполнения шифратором определенных операций может сделать какие-либо предположения о значении определенных фрагментов секретного ключа.

В отчете [283] приведена следующая классификация используемых в криптографических алгоритмах операций по степени их подверженности атакам по времени выполнения:

- не подвержены атакам по времени выполнения (т. е. выполняются за одинаковое количество тактов на всех платформах) операции табличной замены, сдвига и вращения на фиксированное число битов, а также логические операции;
- в ряде случаев атаки по времени выполнения могут быть проведены против алгоритмов, в которых присутствуют операции сложения или вычитания;
- наиболее проблемными с данной точки зрения являются операции умножения, деления, возведения в степень, а также сдвиги и вращения на переменное число битов.

Одним из наиболее показательных алгоритмов, против которых может быть проведена атака по времени выполнения, является алгоритм RC5 (подробную информацию см. в разд. 3.42).

Рассмотрим раунд этого алгоритма (см. рис. 3.154). Как видно, в раунде RC5 присутствуют операции вращения на переменное число битов, что приводит к появлению неких, зависящих от обрабатываемых данных и значения секретного ключа, временных характеристик. Это делает атаку на RC5 теоретически возможной. Среди других алгоритмов, подверженных атаке по времени выполнения, Кохер в [228] упоминает такие известные алгоритмы, как IDEA (см. разд. 3.26), Blowfish (см. разд. 3.8) и DES (см. разд. 3.15).

В качестве «противоядия» от атак по времени выполнения Кохер предлагает следующее [228].

- Обеспечить выполнение шифратором операций строго за одно и то же количество тактов процессора независимо от значений операндов. Однако, это сопряжено с техническими сложностями; кроме того, уменьшает

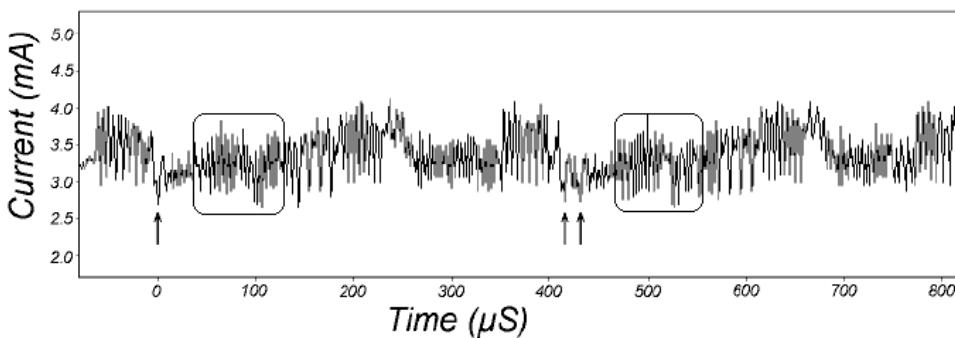
быстродействие алгоритма, поскольку время выполнения операций в этом случае будет приведено к максимально возможному.

- Различным образом маскировать время выполнения операций: использовать случайные временные задержки, выполнять произвольные зашумляющие операции, внедрять в алгоритм различные случайные величины и т. д. Все это также приводит к уменьшению быстродействия алгоритма, поэтому наилучшим вариантом противодействия таким атакам является отсутствие в алгоритме шифрования операций, время выполнения которых зависит от обрабатываемых данных.

## Атаки по потребляемой мощности

Еще одна из пассивных атак предложена также Полом Кохером при участии ряда других сотрудников американской компании Cryptography Research: атака по потребляемой мощности (SPA, simple power analysis) [229]. Данная атака во многом похожа на предыдущую, однако высокоточному замеру подлежит мощность, потребляемая шифратором в процессе выполнения криптографических преобразований.

В работе [229] доказывается, что такой высокоточный замер мощности, достаточный для криптоанализа, вполне возможен. В качестве доказательства, в частности, приводится результат замера мощности, потребляемой шифратором в процессе выполнения второго и третьего раундов алгоритма DES (иллюстрация из работы [229] приведена на рис. 1.22).



**Рис. 1.22.** Пример высокоточного замера потребляемой мощности

На рисунке отчетливо видны операции вращения ключевых регистров процедуры расширения ключа DES (данная процедура подробно описана в разд. 3.15): перед вторым раундом вращение выполняется на 1 бит, а перед

## Классификация алгоритмов шифрования и методов их вскрытия

третьим — на два бита (см. стрелочки на рис. 1.22). Кроме того, отчетливо видны различия между данными раундами (например, см. выделенные рамками области) — эти различия могут проистекать, в том числе, из-за того, что в этих раундах используются различные биты ключа. Соответственно, они могут быть проанализированы криptoаналитиком с целью получения какой-либо информации о ключе шифрования.

В качестве противодействия SPA в [229] предлагаются различные методы зашумления — аналогично атакам по времени выполнения.

Усилиением SPA является дифференциальный криptoанализ по потребляемой мощности DPA (differential power analysis), также предложенный в [229]. Данная атака является более эффективной, чем SPA, но и более сложно реализуемой на практике.

## Другие пассивные атаки

В 2000 г. была предложена атака, использующая высокоточные измерения электромагнитного излучения шифратора, возникающего в процессе шифрования (см., например, [158]). Аналогично SPA и DPA, рассматриваются два варианта такой атаки: SEMA (simple electromagnetic analysis) и DEMA (differential electromagnetic analysis). Авторы статьи [158] в качестве атакуемых устройств избрали криптографические смарт-карты, реализующие алгоритмы DES, RSA и COMP128. Во всех трех случаях атака методом DEMA позволила вычислить значения ключей шифрования.

Еще одна атака основана на предположении, что шифраторы должны каким-либо образом сообщать о возникновении ошибочных ситуаций при расшифровывании данных. Простейший вариант — попытка расшифровывания неверным ключом. Сам факт ошибки расшифровывания может в определенных случаях дать криptoаналитику некоторую полезную информацию (причем, в общем случае, сообщение шифратора об ошибке может быть достаточно развернутым и информативным). Практически реализуемая атака, основанная на сообщениях об ошибках (error message leakage), была предложена известным криптологом Сержем Воденэ (Serge Vaudenay) в [383].

Успех описанных атак зависит от множества различных факторов, в частности:

- содержит ли атакуемый алгоритм операции, критичные с точки зрения утечки информации по побочным каналам;
- насколько реализация алгоритма в программном или аппаратном шифраторе учитывает данные атаки и защищена от подобной утечки информации.

Наиболее часто атакуемыми оказываются криптографические смарт-карты, т. е. достаточно низкоскоростные шифраторы, работающие в условиях огра-

ниченных ресурсов. В данном случае для проведения атаки нет необходимости в столь высокоточных замерах, какие были бы необходимы, например, для атаки на программный шифратор, работающий на современном персональном компьютере. Помимо упомянутой статьи [158], весьма показательной в данном случае является работа [136], посвященная атаке по времени выполнения на смарт-карту CASCADE.

Информация, полученная с помощью атак с использованием утечки данных по побочным каналам, может использоваться криptoаналитиком в контексте других атак, например, для сужения области возможных значений ключа шифрования. Однако, как показано в [158], возможен и идеальный для криptoаналитика вариант: полное раскрытие алгоритма (т. е. вычисление ключа шифрования) только за счет пассивной side-channel-атаки.

## **1.8. Активные атаки на шифраторы, использующие утечку данных по побочным каналам**

При проведении рассмотренных выше пассивных атак криptoаналитик только «прослушивает» шифратор для получения требуемой ему информации.

Активные же атаки подразумевают различные специфические воздействия на шифратор с целью нарушения его нормального функционирования, в результате чего шифратор может давать сбои в процессе своей работы. Такие наведенные криptoаналитиком ошибки в работе шифратора могут дать ему существенно больше информации, полезной для дальнейшего анализа.

Независимо от вида воздействия на шифратор, подобные атаки называются *атаками на основе сбоев* (fault attacks); рассмотрим их в данном разделе.

### **Виды воздействий на шифратор**

Заставить шифратор работать неверно можно множеством различных способов. Наиболее действенными в свете проведения атак на основе сбоев считаются следующие воздействия на шифратор (рис. 1.23) [79, 97, 307, 359]:

- изменение напряжения питания шифратора, существенно превосходящее допустимые пределы (spike attack);
- изменение тактовой частоты шифратора, также выходящее за допустимые рамки (glitch attack);
- высокоточное облучение шифратора с помощью лазера, ультрафиолетовым, рентгеновским или каким-либо другим излучением (optical, radiation attacks);

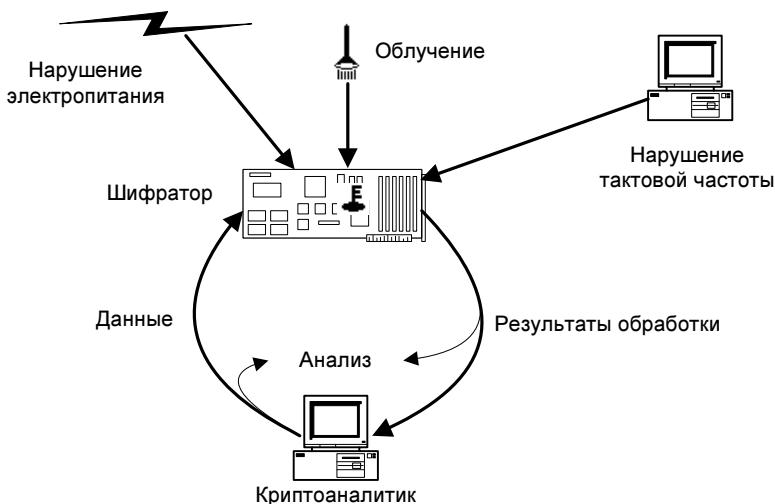


Рис. 1.23. Некоторые виды активного воздействия на шифратор

- высокоточное наведение электромагнитного поля или локальный нагрев определенной области шифратора (electromagnetic, heating attacks);
- внесение изменений в конструкцию шифратора, например, нарушение определенных электрических контактов.

Кроме того, атаки на основе сбоев классифицируются также по тому признаку, насколько атакующий может контролировать следующие факторы [97, 307]:

- местоположение сбоя (например, конкретный бит обрабатываемых данных);
- время возникновения сбоя (например, номер раунда алгоритма шифрования, при выполнении которого происходит сбой);
- количество битов, подверженных сбою;
- вид сбоя: инверсия значения бита или его сброс (в 0 или 1 в зависимости от технологических особенностей шифратора и вида воздействия [359]).

Ясно, что чем больший контроль над данными факторами атаки имеет криптоаналитик, тем более действенной является атака на шифратор. Считается, что наибольший контроль перечисленных факторов атакующему дает наведение сбоев с помощью высокоточного облучения шифратора [97, 359] или наведения электромагнитного поля [97].

Идеальной мишенью для атак на основе сбоев являются криптографические смарт-карты, которые, фактически, находятся в полном распоряжении их

владельца: криptoаналитик, таким образом, может применить к смарт-карте все перечисленные виды воздействий с целью определения прошитого в смарт-карте секретного ключа.

## Дифференциальный анализ на основе сбоев

Основоположниками атак на основе сбоев являются три специалиста американской компании Bellcore: Дэн Боне (Dan Boneh), Ричард ДеМилло (Richard DeMillo) и Ричард Липтон (Richard Lipton). В своей работе [100], вышедшей в 1996 г., они предложили данный вид атак и описали некоторые из возможных атак на основе сбоев на ряд асимметричных криптосистем.

Предложенную в [100] идею существенно развили и распространили на алгоритмы симметричного шифрования известнейшие израильские криptoаналитики Эли Бихам и Эди Шамир, которые в своей работе [79] предложили *дифференциальный криptoанализ на основе сбоев* (differential fault analysis, DFA).

Бихам и Шамир в качестве атакуемого алгоритма выбрали DES, однако впоследствии данная атака была распространена на другие алгоритмы шифрования. Суть атаки на DES методом дифференциального криptoанализа на основе сбоев состоит в следующем:

1. Предполагается, что в процессе шифрования возникает инверсия одного бита в правой части регистра, содержащего текущее значение шифруемого блока данных (т. е. в  $B_i$  — см. рис. 1.24; более подробное описание алгоритма DES см. в разд. 3.15). Точное расположение бита, в котором возникает сбой, а также номер раунда, в процессе которого бит инвертируется, атакующему не известны. Кроме того, авторы атаки не конкретизируют воздействие на шифратор, приводящее к подобному сбою.
2. Атакующий «прогоняет» через шифратор один и тот же текст дважды, воздействуя на шифратор в одном из этих случаев. В результате у криptoаналитика появляется два шифртекста, представляющих собой один и тот же открытый текст, зашифрованный на одном и том же ключе, но только один из этих шифртекстов является корректным, другой же содержит описанную выше ошибку.
3. Сравнивая два полученных шифртекста, атакующий пытается определить номер раунда, в котором возник сбой:
  - если ошибка возникла в раунде 16, то в правой половине шифртекстов будет различаться только 1 бит (здесь и далее не принимается в расчет финальная перестановка алгоритма DES), в левой же различаться будут биты, соответствующие выходным битам той таблицы замен, входное значение которой содержит ошибочный бит (таких таблиц может быть и две);

## Классификация алгоритмов шифрования и методов их вскрытия

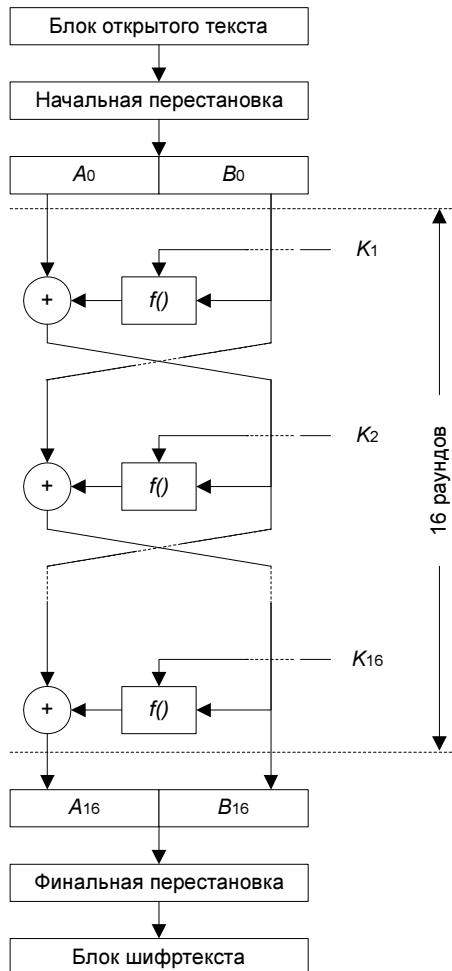


Рис. 1.24. Структура алгоритма DES

- ошибка в раунде 15 также достаточно легко определяется: в этом случае различия в правой половине шифртекстов (которые аналогичны различиям в их левой половине в случае, если ошибка возникла в раунде 16) состоят в битах, соответствующих выходным битам одной или двух таблиц замен; проанализировав различия в левой и правой частях шифртекстов, достаточно легко определить, действительно ли ошибка возникла в раунде 15;
- несколько сложнее, но возможно определить случай, когда ошибка возникает в раунде 14;

- в случае, если ошибка возникает в более ранних раундах, данная пара шифртекстов является непригодной для анализа и отбрасывается.
4. К полученным шифртекстам применяется технология дифференциального криптоанализа (см. разд. 1.6). Наиболее простой вариант возникает в случае, если сбой возник в раунде 16: дифференциальный криптоанализ позволяет легко вычислить значения 6 битов ключа последнего раунда, т. е.  $K_{16}$ .
  5. Для нахождения значений остальных битов  $K_{16}$  аналогичным образом «прогоняются» через шифратор и анализируются дополнительные тексты. Согласно [79], для полного вычисления ключа последнего раунда достаточно от 50 до 200 шифртекстов.
  6. Согласно процедуре расширения ключа алгоритма DES,  $K_{16}$  содержит 48 из 56 битов исходного ключа шифрования алгоритма DES. Остальные биты можно легко вычислить перебором возможных 256 вариантов или анализом отброшенных ранее шифртекстов с ошибками на более ранних раундах.

Как видно, для атаки на основе сбоев требуется несравненно меньше данных, чем для собственно дифференциального криптоанализа алгоритма DES и других алгоритмов (см. разд. 1.6). Однако модель атаки является достаточно строгой и сложно реализуемой на практике. Тем не менее, этот метод криптоанализа является весьма мощным по сравнению со многими из рассмотренных ранее. А в работе [79] Бихам и Шамир продемонстрировали, как дифференциальный криптоанализ на основе сбоев может быть распространен на другие модели атак.

Помимо DES, Бихам и Шамир перечислили в работе [79] несколько алгоритмов, подверженных дифференциальному криптоанализу на основе сбоев: IDEA (см. разд. 3.26), RC5 (см. разд. 3.42), FEAL (см. разд. 3.18), Khufu и Khafre (см. разд. 3.29), Blowfish (см. разд. 3.8), а также Triple DES и DES с независимыми подключами (см. разд. 3.15). Несколько видов атак данным методом предложено, например, в [97], [141] и [163] против текущего стандарта шифрования США — алгоритма AES (см. разд. 3.3).

## Противодействие активным атакам

К сожалению, какого-либо универсального «противоядия» против описанных здесь методов воздействия на шифратор не существует. Однако значительно усложнить проведение атак на основе сбоев против аппаратного шифратора можно, в частности, следующими способами [41]:

- внедрение в шифратор детекторов различных воздействий (например, детекторов изменения напряжения, частоты питания и/или синхронизации,

## Классификация алгоритмов шифрования и методов их вскрытия

освещенности и т. д.), которые при обнаружении воздействия выполняли бы блокировку шифратора;

различного рода пассивное экранирование шифратора, устранение которого приводило бы к выходу шифратора из строя;

различные виды дублирования вычислений со сравнением результатов.

Для программных шифраторов также предлагаются методы защиты [41]:

использование контрольного суммирования фрагментов данных с периодической проверкой в процессе вычислений или различные контрольные вычисления;

дублирование вычислений со сравнением результатов;

внедрение в программу случайных избыточных вычислений.

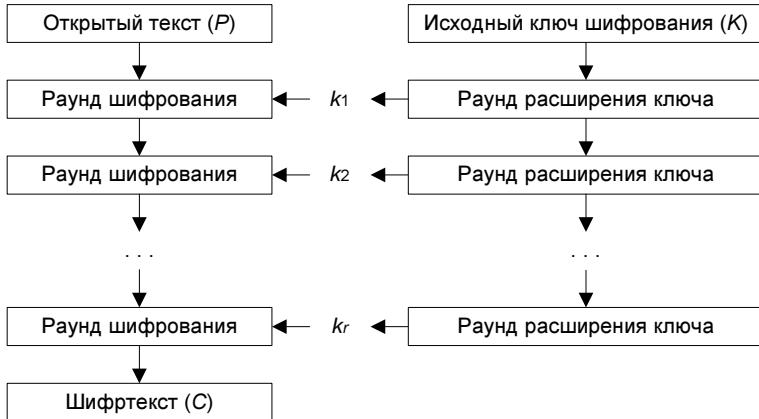
Ясно, что подобные методы приводят к удорожанию шифратора и/или снижению его быстродействия, однако видно, что цель в данном случае оправдывает средства.

## 1.9. Криptoаналитические атаки на связанных ключах

В завершение обзора современных криptoаналитических методов рассмотрим семейство атак на алгоритмы шифрования, использующих *связанные ключи* (related-keys attacks).

### Расширение ключа

Для начала стоит сказать о том, что весьма редко встречаются алгоритмы шифрования, которые используют ключ шифрования (или его фрагменты) в «чистом» виде (таким алгоритмом является, например, отечественный стандарт шифрования ГОСТ 28147-89 — см. разд. 3.1). Подавляющее большинство алгоритмов шифрования выполняют существенную модификацию исходного ключа шифрования для его последующего использования в процессе преобразований. Такая модификация называется *расширением ключа* (key extension, key schedule); существуют примеры алгоритмов, в которых процедура расширения ключа является исключительно сложной по сравнению с собственно шифрованием, среди них стоит упомянуть алгоритмы НРС и FROG (см. разд. 3.23 и 3.19). Название процедуры определяется тем фактом, что исходный ключ шифрования обычно имеет размер существенно меньший совокупности подключей, используемых в раундах алгоритма, т. е. *расширенного ключа*.



**Рис. 1.25.** Назначение процедуры расширения ключа

Получается, что алгоритм шифрования можно логически разделить на два субалгоритма: собственно шифрующие преобразования и процедура расширения ключа (рис. 1.25).

К процедуре расширения ключа предъявляется немало требований, целью которых является повышение криптостойкости и других характеристик алгоритма, например:

- весьма желательно, чтобы процедура расширения ключа могла вычислять ключи «на лету» (on-the-fly), т. е. параллельно с шифрующими преобразованиями: это позволит как распараллеливать вычисления в многопроцессорных системах, так и не тратить память для хранения всего расширенного ключа при шифровании в условиях ограниченных ресурсов;
  - во многих применениях алгоритмов симметричного шифрования (например, в сетевом шифраторе, использующем различные ключи для шифрования данных по различным направлениям, или при использовании алгоритмов шифрования для построения хэш-функций) часто приходится менять ключи в шифраторе; соответственно, весьма сложная процедура расширения ключа не позволит использовать алгоритм шифрования в этих случаях;
  - степень подверженности алгоритма атакам на связанных ключах также весьма зависит от процедуры расширения ключа.

## «Классическая» атака на связанных ключах

«Классическая» атака на связанных ключах, предложенная Эли Бихамом в работе [56], во многом похожа на сдвиговую атаку, описанную в разд. 1.6.

## Классификация алгоритмов шифрования и методов их вскрытия

Основное различие состоит в том, что сдвиговая атака использует два открытых текста  $P$  и  $P'$ , связанных между собой следующим соотношением:

$$P' = F(P, k_1),$$

где  $F$  — функция раунда, а  $k_1$  — подключ первого раунда, тогда как атака на связанных ключах использует весьма похожее соотношение между ключами.

Итак, предположим, что искомый ключ шифрования  $K$  в результате его обработки процедурой расширения ключа  $Ex(K)$  дает следующую последовательность подключей:

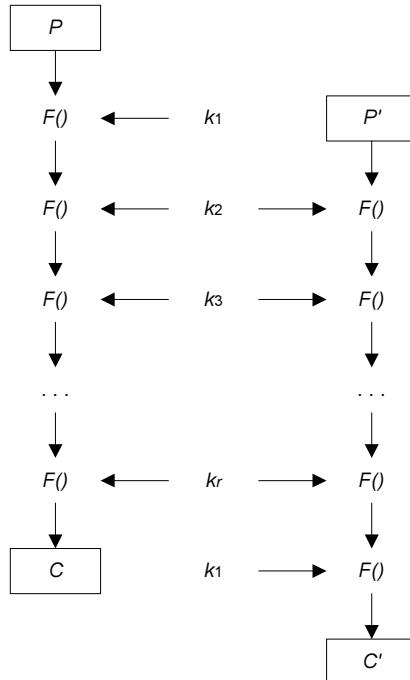
$$Ex(K) = \{k_1, k_2, \dots, k_{r-1}, k_r\},$$

где  $r$  — количество раундов алгоритма шифрования.

Предположим также, что есть некий ключ  $K'$ , расширение которого дает следующую последовательность:

$$Ex(K') = \{k_2, k_3, \dots, k_r, k_1\},$$

т. е. последовательность подключей, формируемая на основе ключа  $K'$ , циклически сдвинута относительно последовательности искомого ключа  $K$  на 1 раунд (рис. 1.26).



**Рис. 1.26.** Подключи в «классической» атаке на связанных ключах

Суть атаки состоит в следующем [56, 119]:

1. Снова предположим, что криптоаналитику известно  $2^{n/2}$  пар  $(P, C)$  открытых текстов и соответствующих им шифртекстов (зашифрованных на искомом ключе  $K$ ), где  $n$  — размер блока шифруемых данных в битах.
2. Кроме того, криптоаналитику известно столько же ( $2^{n/2}$ ) пар текстов  $(P', C')$ , полученных уже с использованием ключа  $K'$ , связанного с ключом  $K$  приведенным выше соотношением.
3. Для каждого соотношения текстов  $(P, C)$  и  $(P', C')$  криптоаналитик решает систему уравнений:

$$F(P, k^*) = P';$$

$$F(C, k^*) = C'.$$

Криптоаналитик не может знать заранее, какой из  $2^{n/2}$  текстов  $P'$  соответствует конкретному тексту  $P$ . Однако вероятность того, что два случайных текста будут соответствовать требуемому соотношению, равна  $2^{-n}$ ; соответственно, согласно парадоксу дней рождения (см. [28]), требуемая пара  $(P, P')$  должна быть найдена, в среднем, после анализа не более, чем  $2^{n/2+1}$  текстов.

Весьма вероятно, что некий ключ  $k^*$ , удовлетворяющий приведенным выше уравнениям, и есть искомый подключ  $k_1$ . В зависимости от процедуры расширения ключа знание  $k_1$  может дать криптоаналитику достаточно много информации об искомом ключе  $K$ . Например, расширение ключа алгоритма LOKI89 [110] построено таким образом, что  $k_1$  представляет собой просто левую 32-битную половину ключа  $K$ , т. е. после вычисления  $k_1$  для нахождения  $K$  достаточно хотя бы перебора всего  $2^{32}$  вариантов.

Необходимое условие для проведения данной атаки состоит в том, что функция раунда  $F$  атакуемого алгоритма должна быть достаточно слабой для вычисления  $k^*$ , что, однако, можно сказать применительно к большинству современных алгоритмов шифрования.

В зависимости от структуры самого алгоритма шифрования трудоемкость атаки может быть существенно меньше описанного выше общего случая. В частности, для алгоритмов, основанных на сбалансированной сети Фейстеля (см. разд. 1.3), раунд алгоритма модифицирует данные, например, следующим образом:

$$\{L_{i+1}, R_{i+1}\} = \{R_i, L_i \oplus F(R_i)\},$$

где  $L$  и  $R$  — соответственно, левая и правая половины обрабатываемого блока данных, а  $i$  — номер раунда. Ясно, что в этом случае поиск требуемых пар  $(P, P')$  существенно упрощается [119].

## Классификация алгоритмов шифрования и методов их вскрытия

Описанная выше атака на связанных ключах не выглядит практической. Легко согласиться с тем, что она требует слишком большого количества предположений. Данная атака довольно долго считалась достаточно мощной, но сугубо теоретической. Однако сейчас многие эксперты считают, что атаки на связанных ключах могут иметь практическое применение. В работе [197] отмечается, что основное условие для атаки выглядит достаточно странно: криptoаналитик может писать ключ (т. е. модифицировать искомый неизвестный ключ требуемым образом), но не может его читать. Тем не менее, некоторые реализации криptoалгоритмов или сетевых протоколов (например, протоколов аутентификации или обмена ключами) могут быть атакованы с использованием связанных ключей. В любом случае, в работе [197] категорически рекомендуется принимать в расчет криptoанализ на связанных ключах при разработке алгоритмов шифрования и их реализаций. Там же приведен пример реализации протокола защищенного обмена ключами, вскрываемого с использованием связанных ключей при условии, что лежащий в основе данного протокола алгоритм шифрования подвержен атакам на связанных ключах. И там же отмечается, что атаки на связанных ключах могут быть весьма опасны и в случае использования алгоритмов симметричного шифрования для построения функций хэширования (см. разд. 1.1).

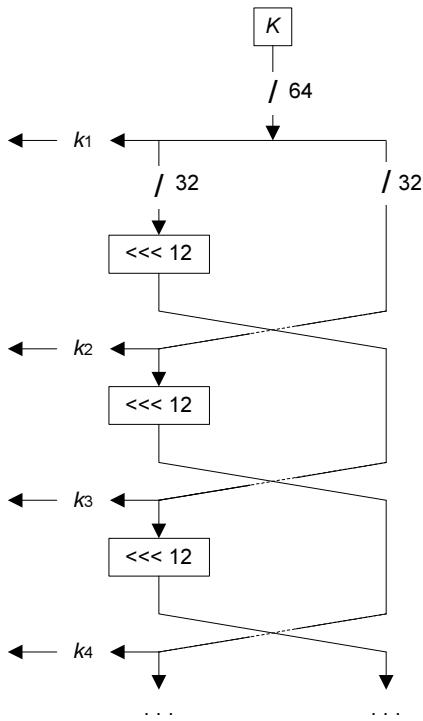
## Атакуемые алгоритмы

Вернемся к алгоритму LOKI89. Схема процедуры расширения ключа этого алгоритма приведена на рис. 1.27 (ключ также используется для входного и выходного *отбеливания* — наложения ключа на шифруемые данные до и после остальных преобразований, что не показано на рисунке; знаком <<< на рис. 1.27 обозначен циклический сдвиг влево на указанное число битов). Как видно из рисунка, все подключи нечетных раундов наследуют одни и те же биты левой половины ключа  $K$ ; аналогично, правая половина ключа формирует все подключи четных раундов. Чрезмерная простота процедуры расширения ключа делает алгоритм подверженным описанной выше атаке на основе связанных ключей [56]:

- для успешной атаки на основе выбранных открытых текстов требуется всего  $2^{17}$  тестовых операций шифрования;
- в случае использования известных открытых текстов сложность атаки возрастает до порядка  $2^{32}$  операций.

В следующей версии алгоритма LOKI — LOKI91 [108] — процедура расширения ключа была несколько модифицирована, однако алгоритм остался подверженным «классической» атаке на связанных ключах: для ее осуществления требуется  $2^{32}$  (выбранные открытые тексты) или  $2^{48}$  (известные

открытые тексты) операций [56]. Еще одна атака, использующая слабость процедуры расширения ключа алгоритма LOKI91, была предложена Ларсом Кнудсеном в работе [210].



**Рис. 1.27.** Процедура расширения ключа алгоритма LOKI89

Согласно [56], данной атакой вскрывается также один из вариантов алгоритма Lucifer (*см. разд. 3.31*); для его вскрытия на основе выбранных открытых текстов требуется около  $2^{32}$  операций.

Другой вариант атаки на связанных ключах описан в [213]. Атака использует весьма интересную особенность алгоритма шифрования SAFER K-64 [251]: для почти каждого ключа  $K$  существует ключ  $K'$  (отличающийся от  $K$  значением одного байта), такой, что для достаточно большого множества открытых текстов (до  $2^{28}$  из возможных  $2^{64}$ ) после 6 раундов шифрования (SAFER K-64 выполняет 6 раундов и финальное преобразование данных) результаты шифрования каждого из текстов на ключах  $K$  и  $K'$  абсолютно эквивалентны. Финальное преобразование делает результаты шифрования различными, но только в одном байте шифртекста. Эта особенность позволяет вычислить

## Классификация алгоритмов шифрования и методов их вскрытия

8 битов 64-битного ключа шифрования алгоритма SAFER K-64 на основе от  $2^{44}$  до  $2^{47}$  выбранных открытых текстов.

## Другие возможные проблемы процедуры расширения ключа

Существуют и другие потенциальные уязвимости, вносимые в алгоритм шифрования неудачно спроектированной процедурой расширения ключа, в частности [197]:

- *нестойкость* к атакам класса «встреча посередине», описанным в разд. 1.6, поскольку данные атаки эксплуатируют тот факт, что первая часть шифрующих преобразований атакуемого алгоритма использует иной набор битов ключа, нежели вторая часть;
- *слабые ключи* — ключи, зашифровывание с использованием которых эквивалентно расшифровыванию, или обладающие другими характеристиками, существенно упрощающими вскрытие;
- *эквивалентные ключи* — различные ключи, но дающие один и тот же результат зашифровывания на некоем подмножестве открытых текстов;
- *классы ключей* — возникают, когда криптоаналитик может относительно легко определить подмножество ключевого множества, которому принадлежит искомый ключ, и, соответственно, уменьшить таким образом область полного перебора ключа.

В работах [197] и [200] описаны различные атаки на известные алгоритмы шифрования, использующие слабости процедур расширения ключа.





## Глава 2

# Новейшая история симметричного шифрования

Глава посвящена краткому обзору новейшей истории алгоритмов шифрования. Она описывает прошедшие в течение последнего десятилетия конкурсы по выбору стандартов шифрования США и Евросоюза. Именно эти конкурсы очень сильно повлияли на современное развитие криптографии и криptoанализа.

## 2.1. Конкурс AES

Алгоритм шифрования DES (подробную информацию см. в разд. 3.15) являлся стандартом симметричного шифрования США с 1979 г. и до принятия нового стандарта в виде алгоритма AES (см. разд. 3.3).

Фактически первые реальные шаги по замене стандарта шифрования были сделаны в 1997 г., когда Институт стандартов и технологий США NIST (National Institute of Standards and Technology) объявил о проведении открытого конкурса алгоритмов шифрования, победитель которого должен был стать новым стандартом симметричного шифрования США. В конкурсе могли принять участие любые организации или частные лица, в том числе находящиеся за пределами США. И действительно, список участников конкурса оказался весьма разнообразен; среди авторов алгоритмов-претендентов были:

- всемирно известные криптологи, например, интернациональный коллектив авторов алгоритма Serpent — Росс Андерсон (Ross Anderson), Эли Бихам и Ларс Кнудсен;
- организации, давно работающие в данной области и обладающие как богатым опытом разработки криптоалгоритмов, так и сильнейшими специалистами, например, американская фирма Counterpane — автор алгоритма Twofish;
- всемирно известные корпорации, обладающие большим научным потенциалом, например, немецкий телекоммуникационный гигант Deutsche Telekom с алгоритмом MAGENTA;

- образовательные учреждения, известные своими достижениями в области криптографии, например, Католический Университет г. Лювен (Katholieke Universiteit Leuven), Бельгия, с алгоритмом Rijndael;
- и наоборот, организации, малоизвестные до проведения конкурса AES, например, фирма Tecnología Apropriada (автор алгоритма FROG) из латиноамериканского государства Коста-Рика.

NIST установил всего два обязательных требования к алгоритмам — участникам конкурса [36]:

- 128-битный размер блока шифруемых данных;
- не менее трех поддерживаемых алгоритмом размеров ключей шифрования: 128, 192 и 256 битов.

Однако несравненно больше было «рекомендательных» требований к будущему стандарту шифрования США. Поскольку удовлетворить обязательные требования было достаточно просто, анализ алгоритмов и выбор из них лучшего производился именно по его соответствуанию «необязательным» характеристикам. «Пожелания» института NIST были, в частности, таковы [36]:

- алгоритм должен быть стойким против криptoаналитических атак, известных на момент проведения конкурса;
- структура алгоритма должна быть ясной, простой и обоснованной, что, во-первых, облегчало бы изучение алгоритма специалистами, а во-вторых, гарантировало бы отсутствие внедренных авторами алгоритма «закладок» (т. е., в данном случае, особенностей архитектуры алгоритма, которыми теоретически могли бы воспользоваться его авторы в злонамеренных целях);
- должны отсутствовать слабые и эквивалентные ключи (см. разд. 1.9);
- скорость шифрования данных должна быть высокой на всех потенциальных аппаратных платформах — от 8-битных до 64-битных;
- структура алгоритма должна позволять распараллеливание операций в многопроцессорных системах и аппаратных реализациях;
- алгоритм должен предъявлять минимальные требования к оперативной и энергонезависимой памяти;
- не должно быть ограничений на использование алгоритма, в частности, в различных стандартных режимах работы (см. разд. 1.4), в качестве основы для построения хэш-функций (см. разд. 1.1), генераторов псевдослучайных последовательностей и т. д.

Эти требования оказались достаточно жесткими и частично противоречавшими друг другу; каждый конкретный алгоритм из участвовавших в конкурсе

представлял собой некий найденный авторами компромисс между предъявленными требованиями.

Заявки от участников конкурса принимались в течение полутора лет, после чего все присланные на конкурс алгоритмы изучались и обсуждались как в самом институте NIST, так и всеми желающими. Стоит сказать, что в NIST пришло немало отзывов, все они находятся в открытом доступе и их можно посмотреть на web-сайте института (<http://www.nist.gov>).

## Алгоритмы — участники конкурса

Всего в конкурсе приняли участие 15 алгоритмов шифрования [33, 284], информация о которых приведена в табл. 2.1.

*Таблица 2.1*

Алгоритмы	Авторы или организация	Основные результаты анализа
CAST-256	Entrust Technologies, Inc.	В алгоритме не обнаружено уязвимостей. Однако скорость шифрования у этого алгоритма невысока; кроме того, у него высокие требования к оперативной и энергонезависимой памяти
Crypton	Future Systems, Inc.	Алгоритм без явных недостатков. Однако Crypton уступает по всем характеристикам похожему на него алгоритму Rijndael. Эксперты конкурса сочли, что в финале конкурса Crypton однозначно проиграет алгоритму Rijndael, поэтому не выбрали его во второй раунд конкурса
DEAL	Richard Outerbridge, Lars Knudsen	Множество недостатков: наличие под множеств слабых ключей, подверженность дифференциальному криптоанализу, отсутствие усиления при использовании 192-битных ключей по сравнению со 128-битными
DFC	Ecole Normale Supérieure	Достоинство алгоритма: очень высокая скорость шифрования на 64-битных платформах. При этом алгоритм весьма медленно шифрует на остальных платформах, а также имеет классы слабых ключей
E2	NTT — Nippon Telegraph and Telephone Corporation	Аналогично алгоритму CAST-256, в E2 не найдено уязвимостей, но требования к энергонезависимой памяти слишком высоки

Таблица 2.1 (продолжение)

Алгоритмы	Авторы или организация	Основные результаты анализа
FROG	TecApro Internacional S.A.	Алгоритм с наиболее оригинальной структурой и с наибольшим количеством недостатков: наличие уязвимостей, низкая скорость шифрования и высокие требования к ресурсам
HPC	Richard Schroeppel	Аналогично алгоритму DFC, HPC очень быстро шифрует на 64-битных платформах, но весьма медленно на остальных. Кроме того, сложная и медленная процедура расширения ключа ограничивает возможные применения алгоритма, а наиболее сложная из всех представленных на конкурс алгоритмов структура раунда усложняет анализ алгоритма и делает недоказуемым отсутствие закладок
LOKI97	Lawrie Brown, Josef Pieprzyk, Jennifer Seberry	Низкая скорость шифрования, высокие требования к ресурсам, наличие уязвимостей
Magenta	Deutsche Telekom AG	Алгоритм подвержен атакам методами линейного и дифференциального криptoанализа; медленная скорость шифрования
MARS	IBM	У алгоритма отсутствуют серьезные недостатки, за исключением низкой скорости шифрования на ряде платформ, не имеющих аппаратной поддержки требуемых операций, и некоторых других несущественных недостатков. Алгоритм был незначительно модифицирован в течение первого раунда; измененный вариант вышел в финал конкурса
RC6	RSA Laboratories	RC6 имеет весьма похожие на MARS проблемы с реализацией на некоторых платформах. Эксперты посчитали это незначительным недостатком — алгоритм вышел в финал конкурса
Rijndael	Joan Daemen, Vincent Rijmen	Каких-либо недостатков у алгоритма не обнаружено; алгоритм вышел в финал конкурса
SAFER+	Cylink Corporation	Алгоритм подвержен ряду атак и имеет низкую скорость выполнения

Таблица 2.1 (окончание)

Алгоритмы	Авторы или организация	Основные результаты анализа
Serpent	Ross Anderson, Eli Biham, Lars Knudsen	Выявлены некоторые незначительные недостатки, алгоритм вышел в финал конкурса
Twofish	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson	Из недостатков эксперты отметили сложность алгоритма, затрудняющую его анализ. Алгоритм вышел в финал конкурса

В результате первого этапа конкурса были выбраны 5 алгоритмов без явно выраженных недостатков: MARS, RC6, Rijndael, Serpent и Twofish. Началось детальное изучение именно этих алгоритмов (что называлось «вторым этапом» или «финалом» конкурса AES), продолжавшееся еще год с небольшим.

## Достоинства и недостатки алгоритмов-финалистов

Результаты проделанной аналитиками работы по изучению алгоритмов-финалистов NIST сформулировал в виде отчета [283]. Данный отчет содержит как результаты анализа алгоритмов, так и обоснование критериев, по которым выполнялась оценка. На основе отчета [283] можно кратко сформулировать сравнительные оценки пяти алгоритмов — финалистов конкурса AES по основным критериям в виде табл. 2.2.

Таблица 2.2

№	Критерий	Serpent	Twofish	MARS	RC6	Rijndael
1	Криптостойкость	+	+	+	+	+
2	Запас криптостойкости	++	++	++	+	+
3	Скорость шифрования при программной реализации	—	±	±	+	+
4	Скорость расширения ключа при программной реализации	±	—	±	±	+
5	Смарт-карты с большим объемом ресурсов	+	+	—	±	++
6	Смарт-карты с ограниченным объемом ресурсов	±	+	—	±	++
7	Аппаратная реализация (ПЛИС)	+	+	—	±	+

Таблица 2.2 (окончание)

№	Критерий	Serpent	Twofish	MARS	RC6	Rijndael
8	Аппаратная реализация (специализированная микросхема)	+	±	—	—	+
9	Защита от атак по времени выполнения и потребляемой мощности	+	±	—	—	+
10	Защита от атак по потребляемой мощности на процедуру расширения ключа	±	±	±	±	—
11	Защита от атак по потребляемой мощности на реализации в смарт-картах	±	+	—	±	+
12	Возможность расширения ключа «на лету»	+	+	±	±	±
13	Наличие вариантов реализации (без потерь в совместимости)	+	+	±	±	+
14	Возможность параллельных вычислений	±	±	±	±	+

Стоит прокомментировать некоторые строки приведенной таблицы.

- Криптостойкость всех алгоритмов-финалистов оказалась достаточной — в процессе исследований не было обнаружено каких-либо практически реализуемых атак на полноценные и полнораундовые версии алгоритмов. В данном случае криptoаналитики обычно исследуют варианты алгоритмов с усеченным количеством раундов, либо с некоторыми внесенными изменениями, незначительными, но ослабляющими алгоритм. Под запасом криптостойкости (security margin) эксперты NIST подразумевают соотношение полного (предусмотренного в спецификациях алгоритмов) количества раундов и максимального из тех вариантов, против которых действуют какие-либо криptoаналитические атаки. Например, с помощью дифференциально-линейного криptoанализа вскрывается 11-раундовый Serpent [68], тогда как в оригинальном алгоритме выполняется 32 раунда. Эксперты NIST в отчете [283] предупредили, что данная оценка является весьма поверхностью и не может быть значимой при выборе алгоритма — победителя конкурса, но, тем не менее, отметили, что запас криптостойкости у Rijndael и RC6 несколько ниже, чем у остальных алгоритмов-финалистов.

- В пп. 5–8 приведена сравнительная оценка возможности и эффективности реализации алгоритмов в перечисленных устройствах.
- В пп. 9–11 имеется в виду, насколько операции, выполняемые конкретным алгоритмом, могут быть подвержены анализу указанным методом. При этом принималось в расчет то, что операции могут быть модифицированы с целью усложнения криptoанализа за счет потери в скорости шифрования (например, проблемное в этом смысле вращение на переменное число битов может принудительно выполняться за равное число тактов — т. е. максимально возможное для данной операции; именно подобные меры противодействия атакам по времени исполнения и потребляемой мощности рекомендует их изобретатель Пол Кохер [228]).
- Из описаний алгоритмов видно, что все они поддерживают расширение ключа «на лету» (т. е. подключи могут генерироваться непосредственно в процессе шифрования — по мере необходимости), однако только Serpent и Twofish поддерживают такую возможность без каких-либо ограничений.
- Под наличием вариантов реализации (*implementation flexibility*) имеется в виду возможность по-разному реализовывать какие-либо операции алгоритма, оптимизируя их под конкретные цели. Наиболее показательными в этом смысле являются варианты процедуры расширения ключа алгоритма Twofish (см. разд. 3.56), позволяющие оптимизировать реализацию алгоритма в зависимости, прежде всего, от частоты смены ключа.

Сформулируем основные достоинства и недостатки каждого из алгоритмов-финалистов, не ставших победителями конкурса [283] (табл. 2.3).

Таблица 2.3

Алгоритм	Достоинства	Недостатки
Serpent	<p>Простая структура алгоритма облегчает его анализ с целью нахождения возможных уязвимостей.</p> <p>Serpent эффективно реализуем аппаратно и в условиях ограниченных ресурсов.</p> <p>Serpent легко модифицируется с целью защиты от атак по времени выполнения и потребляемой мощности (однако за счет снижения скорости)</p>	<p>Является самым медленным из алгоритмов-финалистов в программных реализациях.</p> <p>Процедуры зашифровывания и расшифровывания абсолютно различны, т. е. требуют раздельной реализации.</p> <p>Распараллеливание вычислений при шифровании алгоритмом Serpent реализуемо с ограничениями</p>

Таблица 2.3 (продолжение)

Алгоритм	Достоинства	Недостатки
Twofish	<p>Twofish эффективно реализуем аппаратно и в условиях ограниченных ресурсов.</p> <p>Зашифровывание и расшифровывание в алгоритме Twofish практически идентичны.</p> <p>Является лучшим из алгоритмов-финалистов с точки зрения поддержки расширения ключа «на лету».</p> <p>Несколько вариантов реализации позволяют оптимизировать алгоритм для конкретных применений</p>	<p>Сложность структуры алгоритма затрудняет его анализ.</p> <p>Сложная и медленная процедура расширения ключа.</p> <p>Относительно сложно защищается от атак по времени выполнения и потребляемой мощности.</p> <p>Распараллеливание вычислений при шифровании алгоритмом Twofish реализуемо с ограничениями</p>
MARS	<p>Зашифровывание и расшифровывание в алгоритме MARS практически идентичны</p>	<p>Исключительно сложная структура алгоритма с раундами различных типов затрудняет как анализ алгоритма, так и его реализацию.</p> <p>Возникают проблемы при программной реализации на тех платформах, которые не поддерживают 32-битное умножение и вращение на переменное число битов.</p> <p>Алгоритм MARS не может быть эффективно реализован аппаратно и в условиях ограниченных ресурсов.</p> <p>Сложно защищается от атак по времени выполнения и потребляемой мощности.</p> <p>MARS хуже других алгоритмов-финалистов поддерживает расширение ключей «на лету».</p> <p>Распараллеливание вычислений при шифровании алгоритмом MARS реализуемо с ограничениями</p>

Таблица 2.3 (окончание)

Алгоритм	Достоинства	Недостатки
RC6	<p>Простая структура алгоритма облегчает его анализ. Кроме того, алгоритм унаследовал часть преобразований от своего предшественника — алгоритма RC5, тщательно проанализированного до конкурса AES.</p> <p>Самый быстрый из алгоритмов-финалистов на 32-битных платформах.</p> <p>Зашифровывание и расшифровывание в алгоритме RC6 практически идентичны</p>	<p>Скорость шифрования при программной реализации сильно зависит от того, поддерживает ли платформа 32-битное умножение и вращение на переменное число битов.</p> <p>RC6 сложно реализуем аппаратно и в условиях ограниченных ресурсов.</p> <p>Достаточно сложно защищается от атак по времени выполнения и потребляемой мощности.</p> <p>Недостаточно полно поддерживает расширение ключей «на лету».</p> <p>Распараллеливание вычислений при шифровании алгоритмом RC6 реализуемо с ограничениями</p>

В результате победителем конкурса стал алгоритм Rijndael [283]; ему было присвоено название AES [32], под именем которого он уже достаточно широко реализован и, видимо, по широте распространения обойдет своего предшественника — алгоритм DES.

Выбор экспертов не кажется удивительным — в таблице с характеристиками алгоритмов отчетливо видно, что практически по всем характеристикам Rijndael, как минимум, не уступает остальным алгоритмам-финалистам.

Что же касается алгоритмов Serpent, Twofish, MARS и RC6, то видно, что они практически равнозначны по совокупности характеристик, за исключением алгоритма MARS, имеющего существенно больше недостатков, в том числе алгоритм практически нереализуем в условиях ограниченных ресурсов.

## 2.2. Конкурс NESSIE

Европейский конкурс криптоалгоритмов NESSIE стартовал несколько позже конкурса AES — в феврале 2000 г. NESSIE — аббревиатура от New European Schemes for Signature, Integrity, and Encryption, «новые европейские алгоритмы подписи, обеспечения целостности и шифрования». Как видно из

названия, цели конкурса NESSIE были заметно шире, чем у конкурса AES. В рамках конкурса NESSIE рассматривались алгоритмы следующих категорий [375, 376]:

- блочное симметричное шифрование (на конкурс принято 17 алгоритмов);
- потоковое шифрование (6 алгоритмов);
- вычисление кодов аутентификации сообщений (Message Authentication Code — MAC, 2 алгоритма);
- хэширование (1 алгоритм);
- асимметричное шифрование (5 алгоритмов);
- электронная цифровая подпись (7 алгоритмов);
- идентификация (1 алгоритм).

Как и на конкурсе AES, алгоритмы-участники конкурса были присланы практически со всего мира. Причем абсолютным лидером по количеству рассмотренных на конкурсе алгоритмов явилась Япония — из 39 участников конкурса в Японии было разработано 8.

В отличие от конкурса AES, единственной экспертной организацией в котором был институт NIST, организатором конкурса NESSIE был интернациональный консорциум из следующих организаций, известных своими исследованиями в области криптологии:

- Католический Университет г. Лювен, Бельгия — координатор проекта;
- высшее учебное заведение Ecole Normale Supérieure, Париж, Франция;
- университет Royal Holloway, Лондон, Великобритания;
- корпорация Siemens AG, Германия;
- Технологический Институт Technion, Хайфа, Израиль;
- университет г. Берген, Норвегия.

Еще одно принципиальное отличие NESSIE (касательно алгоритмов блочного шифрования) от конкурса AES состояло в том, что не был установлен какой-либо конкретный размер блока шифруемых данных, поэтому в конкурсе рассматривались 64-, 128-, 160- и 256-битные блочные шифры. Общий список алгоритмов блочного шифрования, участвовавших в конкурсе, приведен в табл. 2.4.

Таблица 2.4

№	Алгоритм	Размеры блока в битах	Авторы (организация*, страна)
1	CS-Cipher	64	Жак Стерн (Jacques Stern), Серж Воденэ (Serge Vaudenay) — CS Communication & Systemes, Франция
2	Hierocrypt-L1	64	Кенджи Охкума (Kenji Ohkuma), Фумихико Сано (Fumihiko Sano), Хирофуми Муратани (Hirofumi Muratani), Масахико Мотояма (Masahiko Motoyama), Шиничи Кавамура (Shinichi Kawamura) — Toshiba Corporation, Япония
3	Hierocrypt-3	128	
4	IDEA	64	Ксуеджа Лай (Xuejia Lai), Джеймс Мэсси (James L. Massey) — Ascom Systec Ltd., Швейцария
5	Khazad	64	Пауло Баррето (Paulo Sergio L. M. Barreto) — Бразилия,
6	Anubis	128	Винсент Риджмен (Vincent Rijmen) — Бельгия
7	MISTY1	64	Мицуру Мацуи (Mitsuru Matsui) и др. — Mitsubishi Electric Corporation, Япония
8	Nimbus	64	Алексис Уорнер Мачадо (Alexis Warner Machado) — Бразилия
9	NUSH	64, 128, 256	Лебедев Анатолий Николаевич, Волчков Алексей Анатольевич — «ЛАН Крипто», Россия
10	SAFER++	64, 128	Джеймс Мэсси (James L. Massey) — Швейцария, Гурген Хачатрян (Gurgen H. Khachatrian) — США, Мельсик Курегян (Melsik K. Kuregian) — Армения
11	Grand Cru	128	Йохан Борст (Johan Borst) — Бельгия
12	Noekeon	128	Джоан Деймен (Joan Daemen), Михаэль Петерс (Michael Peeters), Жиль Ван Аске (Gilles Van Assche), Винсент Риджмен (Vincent Rijmen) — Бельгия
13	Q	128	Лесли МакБрайд (Leslie McBride) — США
14	RC6	128 и более	Рональд Ривест (Ronald R. Rivest), Мэттью Робшоу (Matthew J. B. Robshaw), Рэймонд Сидни (Raymond M. Sidney), Икван Лайза Ин (Yiqun Lisa Yin) — RSA Security Inc., США

Таблица 2.4 (окончание)

№	Алгоритм	Размеры блока в битах	Авторы (организация*, страна)
15	SC2000	128	Такэши Шимояма (Takeshi Shimoyama), Хитоши Янами (Hitoshi Yanami), Казухиро Йокояма (Kazuhiro Yokoyama), Масахико Таненака (Masahiko Takenaka), Коуичи Ито (Kouichi Itoh), Джун Яджима (Jun Yajima), Наоя Тории (Naoya Torii), Хидема Танака (Hidema Tanaka) — Fujitsu Laboratories LTD., Япония
16	Camellia	128	Казумаро Аоки (Kazumaro Aoki), Тецуя Ичикава (Tetsuya Ichikawa), Масаюки Канда (Masayuki Kanda), Мицуру Мацуи (Mitsuru Matsui), Шихо Мориай (Shiho Morai), Джунко Накаджима (Junko Nakajima), Тошио Токита (Toshio Tokita) — Nippon Telegraph and Telephone Corporation и Mitsubishi Electric Corporation, Япония
17	SHACAL	160	Хелена Хандшух (Helena Handschuh), Давид Начаш (David Naccache) — Gemplus, Франция

\* Организация – владелец прав на алгоритм (если таковая существует).

Еще одна аналогия с конкурсом AES — два раунда конкурса. Причем их назначение полностью аналогично конкурсу AES: в первом раунде конкурса NESSIE были рассмотрены все алгоритмы, из которых во второй раунд были выбраны те из них, у которых не было явно выраженных недостатков. Соответственно, во втором раунде из финалистов были выбраны лучшие.

Многие материалы, касающиеся конкурса NESSIE, доступны на его домашней странице (<http://www.cryptonessie.org>), которая поддерживается Католическим Университетом г. Лювен (<http://www.cosic.esat.kuleuven.be>).



## Глава 3

# Описание алгоритмов

Эта глава содержит описание алгоритмов шифрования, т. е., в основном, следующие сведения о каждом из приведенных алгоритмов:

- историю создания и использования алгоритма;
- основные характеристики и структуру алгоритма, включая подробное описание используемых в алгоритме преобразований;
- достоинства и недостатки алгоритма;
- описание известных методов вскрытия алгоритма.

Алгоритмы снабжены схемами, существенно упрощающими понимание структуры алгоритма. Стоит отметить, что описание большинства алгоритмов достаточно подробно для разработки на их основе программного или аппаратного шифратора.

### 3.1. Алгоритм ГОСТ 28147-89

Этот алгоритм является обязательным для применения в качестве алгоритма шифрования в государственных организациях РФ и ряде коммерческих [17, 18].

#### Описание алгоритма

Схема алгоритма ГОСТ 28147-89 показана на рис. 3.1 [4]. Как видно, схема этого алгоритма достаточно проста, что однозначно упрощает его программную или аппаратную реализацию.

Алгоритм ГОСТ 28147-89 шифрует информацию блоками по 64 бита, которые разбиваются на два субблока по 32 бита ( $N1$  и  $N2$ ). Субблок  $N1$  определенным образом обрабатывается, после чего его значение складывается

со значением субблока  $N_2$  (сложение выполняется по модулю 2), затем субблоки меняются местами. Такое преобразование выполняется определенное количество раундов: 16 или 32 в зависимости от режима работы алгоритма (описаны далее). В каждом раунде выполняются следующие операции:

1. Наложение ключа. Содержимое субблока  $N_1$  складывается по модулю  $2^{32}$  с частью ключа  $K_x$ .

Ключ шифрования алгоритма ГОСТ 28147-89 имеет размерность 256 битов, а  $K_x$  — это его 32-битная часть, т. е. 256-битный ключ шифрования представляется в виде конкатенации 32-битных подключей (рис. 3.2):

$$K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7.$$

В процессе шифрования используется один из этих подключей — в зависимости от номера раунда и режима работы алгоритма.

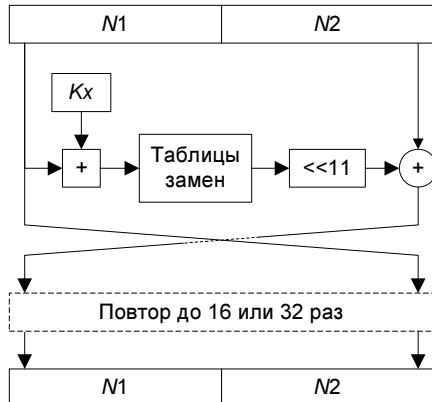


Рис. 3.1. Схема алгоритма ГОСТ 28147-89

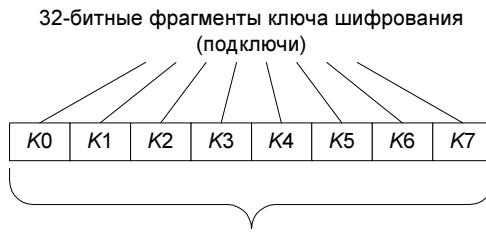


Рис. 3.2. Ключ шифрования алгоритма ГОСТ 28147-89

2. Табличная замена. После наложения ключа субблок  $N_1$  разбивается на 8 частей по 4 бита, значение каждой из которых по отдельности заменяется в соответствии с таблицей замены для данной части субблока. Табличные замены (Substitution box, S-box) часто используются в современных алгоритмах шифрования, поэтому стоит рассмотреть их подробнее.

Табличная замена используется таким образом: на вход подается блок данных определенной размерности (в этом случае — 4-битный), числовое представление которого определяет номер выходного значения. Например, имеем S-box следующего вида:

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1.

Пусть на вход пришел 4-битный блок «0100», т. е. значение 4. Согласно таблице, выходное значение будет равно 15, т. е. «1111» (0 заменяется на 4, 1 — на 11, значение 2 не изменяется и т. д.).

Как видно, схема алгоритма весьма проста, что означает, что наибольшая нагрузка по шифрованию данных ложится на таблицы замен. К сожалению, алгоритм обладает тем свойством, что существуют «слабые» таблицы замен, при использовании которых алгоритм может быть раскрыт криptoаналитическими методами. К числу слабых относится, например, таблица, в которой выход равен входу [16]:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.

3. Побитовый циклический сдвиг влево на 11 битов.

## Режимы работы алгоритма

Алгоритм ГОСТ 28147-89 имеет 4 режима работы:

- режим простой замены;
- режим гаммирования;
- режим гаммирования с обратной связью;
- режим выработки имитоприставок.

Эти режимы несколько отличаются от общепринятых (описанных в разд. 1.4), поэтому стоит рассмотреть их подробнее.

Данные режимы имеют различное назначение, но используют одно и то же описанное выше шифрующее преобразование.

## Режим простой замены

В режиме простой замены для зашифровывания каждого 64-битного блока информации просто выполняются 32 описанных выше раунда. 32-битные подключи используются в следующей последовательности:

- $K0, K1, K2, K3, K4, K5, K6, K7, K0, K1$  и т. д. — в раундах с 1-го по 24-й;
  - $K7, K6, K5, K4, K3, K2, K1, K0$  — в раундах с 25-го по 32-й.

Расшифровывание в режиме простой замены производится совершенно так же, но с несколько другой последовательностью применения подключей:

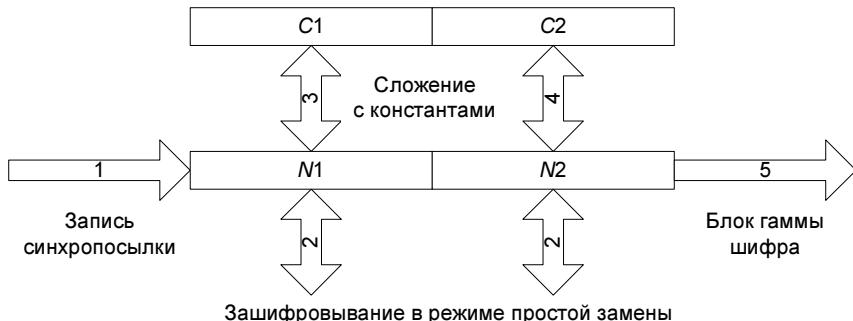
- $K0, K1, K2, K3, K4, K5, K6, K7$  — в раундах с 1-го по 8-й;
  - $K7, K6, K5, K4, K3, K2, K1, K0, K7, K6$  и т. д. — в раундах с 9-го по 32-й.

Аналогично стандартному режиму ECB, по причине раздельного шифрования блоков режим простой замены категорически не рекомендуется использовать для шифрования собственно данных; он должен использоваться только для шифрования других ключей шифрования в многоключевых схемах.

## Режим гаммирования

В режиме гаммирования (рис. 3.3) каждый блок открытого текста побитно складывается по модулю 2 с блоком гаммы шифра размером 64 бита. Гамма шифра — это специальная последовательность, которая вырабатывается с помощью описанных выше преобразований следующим образом:

1. В регистры  $N1$  и  $N2$  записывается их начальное заполнение — 64-битная величина, называемая «синхропосылкой» (синхропосылка, практически, является аналогом вектора инициализации в режимах CBC, CFB и OFB).



**Рис. 3.3.** Режим гаммирования

## Описание алгоритмов

2. Выполняется зашифровывание содержимого регистров  $N1$  и  $N2$  (в данном случае — синхропосылки) в режиме простой замены.
3. Содержимое  $N1$  складывается по модулю  $(2^{32} - 1)$  с константой  $C1 = 2^{24} + 2^{16} + 2^8 + 4$ , результат сложения записывается в регистр  $N1$ .
4. Содержимое  $N2$  складывается по модулю  $2^{32}$  с константой  $C2 = 2^{24} + 2^{16} + 2^8 + 1$ , результат сложения записывается в регистр  $N2$ .
5. Содержимое регистров  $N1$  и  $N2$  подается на выход в качестве 64-битного блока гаммы шифра (т. е. в данном случае  $N1$  и  $N2$  образуют первый блок гаммы).
6. Если необходим следующий блок гаммы (т. е. необходимо продолжить зашифровывание или расшифровывание), выполняется возврат к шагу 2.

Для расшифровывания аналогичным образом выполняется выработка гаммы, затем снова применяется операция XOR к битам зашифрованного текста и гаммы.

Для выработки той же самой гаммы шифра у пользователя, расшифровывающего криптоGRAMМУ, должен быть тот же самый ключ и то же значение синхропосылки, которые применялись при зашифровывании информации. В противном случае получить исходный текст из зашифрованного не удастся.

В большинстве реализаций алгоритма ГОСТ 28147-89 синхропосылка не является секретным элементом, однако синхропосылка может быть так же секретна, как и ключ шифрования. В этом случае можно считать, что эффективная длина ключа алгоритма (256 битов) увеличивается еще на 64 бита синхропосылки, которую можно рассматривать как дополнительный ключевой элемент.

## Режим гаммирования с обратной связью

В режиме гаммирования с обратной связью в качестве заполнения регистров  $N1$  и  $N2$ , начиная со 2-го блока, используется не предыдущий блок гаммы, а результат зашифровывания предыдущего блока открытого текста (рис. 3.4). Первый же блок в данном режиме генерируется полностью аналогично предыдущему.

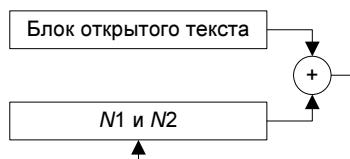


Рис. 3.4. Выработка гаммы шифра в режиме гаммирования с обратной связью

## Режим выработки имитоприставки

Имитоприставка — это криптографическая контрольная сумма, вычисляемая с использованием ключа шифрования и предназначенная для проверки целостности сообщений. Для ее вычисления существует специальный режим алгоритма ГОСТ 28147-89.

Генерация имитоприставки выполняется следующим образом:

1. Первый 64-битный блок информации, для которой вычисляется имитоприставка, записывается в регистры  $N1$  и  $N2$  и зашифровывается в сокращенном режиме простой замены, в котором выполняются первые 16 раундов из 32.
2. Полученный результат суммируется по модулю 2 со следующим блоком информации с сохранением результата в  $N1$  и  $N2$ .
3.  $N1$  и  $N2$  снова зашифровываются в сокращенном режиме простой замены и т. д. до последнего блока информации.

Имитоприставкой считается 64-битное результирующее содержимое регистров  $N1$  и  $N2$  или его часть. Чаще всего используется 32-битная имитоприставка, т. е. половина содержимого регистров. Этого достаточно, поскольку, как и любая контрольная сумма, имитоприставка предназначена, прежде всего, для защиты от случайных искажений информации. Для защиты же от преднамеренной модификации данных применяются другие криптографические методы — в первую очередь электронная цифровая подпись (см. разд. 1.1).

Имитоприставка используется следующим образом:

1. При зашифровывании какой-либо информации вычисляется имитоприставка открытого текста и посыпается вместе с шифртекстом.
2. После расшифровывания имитоприставка снова вычисляется и сравнивается с присланной.
3. Если вычисленная и присланная имитоприставки не совпадают — шифртекст был искажен при передаче или использовались неверные ключи при расшифровывании.

Имитоприставка особенно полезна для проверки правильности расшифровывания ключевой информации при использовании многоключевых схем.

Имитоприставка — это некоторый аналог кода аутентификации сообщений MAC, вычисляемого в режиме CBC; отличие состоит в том, что при вычислении имитоприставки не используется синхропосылка, тогда как при вычислении MAC используется вектор инициализации.

## Криптостойкость алгоритма

В 1994 г. описание алгоритма ГОСТ 28147-89 было переведено на английский язык и опубликовано [300]; именно после этого стали появляться результаты его анализа, выполненного зарубежными специалистами; однако в течение значительного времени не было найдено каких-либо атак, приближающихся к практически осуществимым [91].

По мнению автора [16], высокая стойкость алгоритма ГОСТ 28147-89 достигается за счет следующих факторов:

- большой длины ключа — 256 битов; вместе с секретной синхропосылкой эффективная длина ключа увеличивается до 320 битов;
- 32 раундов преобразований; уже после 8 раундов достигается полный эффект рассеивания входных данных: изменение одного бита блока открытого текста повлияет на все биты блока шифртекста, и наоборот, т. е. существует многократный запас стойкости.

Рассмотрим результаты криptoанализа алгоритма ГОСТ 28147-89.

## Анализ таблиц замен

Поскольку таблицы замен в стандарте [4] не приведены, в ряде работ (например, в [115]) высказывается предположение, что «компетентная организация» может выдать как «хорошие», так и «плохие» таблицы замен. Однако в [28] известнейший эксперт Брюс Шнайер называет такие предположения «слухами». Ясно, что криптостойкость алгоритма во многом зависит от свойств используемых таблиц замен, соответственно, существуют слабые таблицы замен (пример см. выше), применение которых может упростить вскрытие алгоритма. Тем не менее, возможность использования различных таблиц замен кажется весьма достойной идеей, в пользу которой можно привести два следующих факта из истории стандарта шифрования DES (подробности см. в разд. 3.15):

- атаки с помощью как линейного, так и дифференциального криptoанализа алгоритма DES используют конкретные особенности таблиц замен; при использовании других таблиц криptoанализ придется начинать сначала;
- были предприняты попытки усилить DES против линейного и дифференциального криptoанализа путем использования более стойких таблиц замен; такие таблицы, действительно более стойкие, были предложены, например, в алгоритме  $s^5$ DES [206]; но, увы, заменить DES на  $s^5$ DES было невозможно, поскольку таблицы замен жестко определены в стандарте [150], соответственно, реализации алгоритма наверняка не поддерживают возможность смены таблиц на другие.

В ряде работ (например, [14], [16] и [115]) ошибочно делается вывод о том, что секретные таблицы замен алгоритма ГОСТ 28147-89 могут являться частью ключа и увеличивать его эффективную длину (что несущественно, поскольку алгоритм обладает весьма большим 256-битным ключом). Однако в работе [334] доказано, что секретные таблицы замен могут быть вычислены с помощью следующей атаки, которая может быть применена практически:

1. Устанавливается нулевой ключ и выполняется поиск «нулевого вектора», т. е. значения  $z = f(0)$ , где  $f()$  — функция раунда алгоритма. Этот этап занимает порядка  $2^{32}$  операций шифрования.
2. С помощью нулевого вектора вычисляются значения таблиц замен, что занимает не более  $2^{11}$  операций.

## Модификации алгоритма и их анализ

В работе [293] проведен криптоанализ модификаций алгоритма ГОСТ 28147-89:

- алгоритма GOST-H, в котором, относительно оригинального алгоритма, изменен порядок использования подключей, а именно в раундах с 25-го по 32-й подключи используются в прямом порядке, т. е. точно так же, как и в предыдущих раундах алгоритма;
- 20-раундового алгоритма GOST $\oplus$ , в раунде которого для наложения ключа используется операция XOR вместо сложения по модулю  $2^{32}$ .

По результатам анализа сделан вывод о том, что GOST-H и GOST $\oplus$  слабее исходного алгоритма ГОСТ 28147-89, поскольку оба имеют классы слабых ключей. Стоит отметить, что в части криптоанализа GOST $\oplus$  работа [293] слово в слово повторяет раздел, посвященный криптоанализу алгоритма ГОСТ 28147-89, вышедшей в 2000 г. известной работы [91] (без каких-либо ссылок на оригинал). Это ставит под сомнение профессионализм авторов работы [293] и остальные ее результаты.

Весьма интересная модификация алгоритма предложена в работе [402]: таблицы  $S_1 \dots S_8$  обязательно должны быть различными; в каждом раунде алгоритма должна выполняться их перестановка по определенному закону. Данная перестановка может быть зависимой от ключа шифрования, а может быть и секретной (т. е. являться частью ключа шифрования большего размера по сравнению с исходным 256-битным ключом). Оба этих варианта, по мнению их авторов, существенно усиливают стойкость алгоритма против линейного и дифференциального криптоанализа.

И еще одна модификация, связанная с таблицами замен, приведена в работе [168], в которой анализируется один из возможных методов вычисления таблиц

замен на основе ключа шифрования. Авторы работы сделали вывод, что подобная зависимость ослабляет алгоритм, поскольку приводит к наличию слабых ключей и к некоторым потенциальным уязвимостям алгоритма.

## Анализ полнораундового алгоритма

Существуют атаки и на полнораундовый ГОСТ 28147-89 без каких-либо модификаций. Одна из первых открытых работ, в которых был проведен анализ алгоритма, — широко известная работа [197] — посвящена атакам, использующим слабости процедуры расширения ключа ряда известных алгоритмов шифрования. В частности, полнораундовый алгоритм ГОСТ 28147-89 может быть вскрыт с помощью дифференциального криптоанализа на связанных ключах, но только в случае использования слабых таблиц замен. 24-раундовый вариант алгоритма (в котором отсутствуют первые 8 раундов) вскрывается аналогичным образом при любых таблицах замен, однако сильные таблицы замен (например, приведенная в [28]) делают такую атаку абсолютно непрактичной.

Отечественные ученые А. Г. Ростовцев и Е. Б. Маховенко в 2001 г. в работе [22] предложили принципиально новый метод криптоанализа (по мнению авторов, существенно более эффективный, чем линейный и дифференциальный криптоанализ [23]) путем формирования целевой функции от известного открытого текста, соответствующего ему шифртекста и искомого значения ключа и нахождения ее экстремума, соответствующего истинному значению ключа. Они же нашли большой класс слабых ключей алгоритма ГОСТ 28147-89, которые позволяют вскрыть алгоритм с помощью всего 4-х выбранных открытых текстов и соответствующих им шифртекстов с достаточно низкой сложностью. Криптоанализ алгоритма продолжен в работе [23].

В 2004 г. группа специалистов из Кореи предложила атаку, с помощью которой, используя дифференциальный криптоанализ на связанных ключах, можно получить с вероятностью 91,7 % 12 битов секретного ключа [227]. Для атаки требуется  $2^{35}$  выбранных открытых текстов и  $2^{36}$  операций шифрования. Как видно, данная атака, практически, бесполезна для реального вскрытия алгоритма.

## Заключение

В открытой литературе можно найти довольно мало работ, посвященных криптоанализу алгоритма ГОСТ 28147-89. Это особенно заметно по сравнению с огромным числом работ, посвященных криптоанализу стандартов шифрования DES (см. разд. 3.15) и AES (см. разд. 3.3) или криптоанализу

некоторых широко используемых алгоритмов шифрования, например, IDEA (см. разд. 3.26) или SAFER (см. разд. 3.44). По результатам открытых работ можно сделать вывод о весьма высокой криптостойкости отечественного стандарта шифрования.

## 3.2. Алгоритм Aardvark

Алгоритм Aardvark является продолжением и усилением идеи, заложенной в семейство алгоритмов Bear, Lion и Lioness (см. разд. 3.6), состоящей в том, что алгоритм шифрования представляет собой комбинацию функции хэширования и генератора псевдослучайных последовательностей. Алгоритм разработан в 1996 г.

Автор алгоритма — Пэт Морин (Pat Morin) из Университета Carleton (Канада). Aardvark имеет весьма необычную структуру. Кроме того, необычно и то, что алгоритм имеет переменный и, фактически, неограниченный размер блока шифруемых данных. И еще одна особенность, которую можно считать существенным недостатком алгоритма, — размер шифртекста всегда несколько больше размера шифруемого открытого текста, что существенно уменьшает потенциальную область применения алгоритма.

Как было сказано выше, в качестве шифрующих преобразований алгоритм Aardvark использует следующие [271]:

- алгоритм хэширования  $H()$ , результатом которого является значение размером не более 512 битов; в качестве функции  $H()$  может быть использован, например, алгоритм SHA (Secure Hash Algorithm, стандарт хэширования США) [152];
- алгоритм ключевого хэширования  $H'()$ ;
- алгоритм генерации псевдослучайных чисел, в качестве которого может использоваться, например, потоковый шифр SEAL [332].

Фактически Aardvark является шаблоном для построения на его основе конкретных алгоритмов шифрования, поскольку он использует множество параметров, не определенных в спецификации алгоритма. Структура алгоритма приведена на рис. 3.5. Зашифровывание данных алгоритмом Aardvark выполняется следующим образом:

1. Первый субблок шифртекста  $C_1$  вычисляется путем хэширования функцией  $H()$  блока открытого текста  $P$ .
2.  $C_1$  обрабатывается функцией ключевого хэширования  $H'()$ , в качестве второго параметра которой используется секретный ключ алгоритма. Стоит обратить внимание на тот факт, что размер секретного ключа в описании

## Описание алгоритмов

алгоритма [271] не ограничивается. Кроме того, там не указана и конкретная функция ключевого хэширования, однако в качестве примера приведен следующий вариант:

$$H'(X, K) = SHA(K \bullet X \bullet K),$$

где  $\bullet$  — операция конкатенации.

Именно такая функция ключевого хэширования используется в упомянутом выше алгоритме Bear.

3. Результат предыдущего шага используется в качестве начального значения функции  $S()$  — генератора псевдослучайных чисел, который вырабатывает на его основе последовательность того же размера, что и размер шифруемого блока данных  $P$ . Эта последовательность накладывается на  $P$  с помощью побитовой логической операции «исключающее или» (XOR), в результате чего получается второй субблок шифртекста  $C2$ .

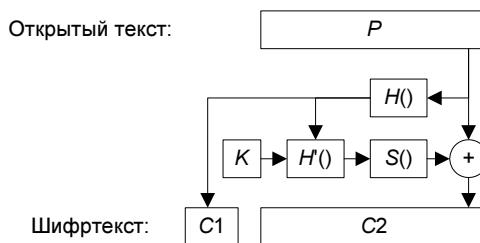


Рис. 3.5. Зашифровывание данных алгоритмом Aardvark

Таким образом, зашифровывание данных алгоритмом Aardvark можно представить в виде следующих формул:

$$C1 = H(P);$$

$$C2 = P \oplus S(H'(C1, K)).$$

Расшифровывание выполняется еще проще (рис. 3.6):

$$P = C2 \oplus S(H'(C1, K)).$$

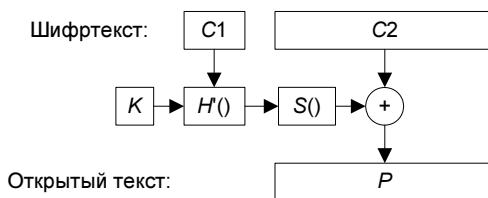


Рис. 3.6. Расшифровывание данных алгоритмом Aardvark

Интересной особенностью алгоритма Aardvark является и то, что в его структуру заложена возможность проверки целостности данных [271] — после расшифровывания можно выполнить следующую операцию:

$$C1' = H(P),$$

затем сравнить  $C1'$  с  $C1$ . Эквивалентность этих значений будет подтверждением корректности полученного значения  $P$ .

Как было сказано выше, размер блока алгоритма не конкретизирован. Однако стоит учесть, что с точки зрения быстродействия алгоритм имеет смысл лишь тогда, когда размер  $P$  намного больше размера  $C1$ , т. е. размера выходного значения используемой функции хэширования. Автор алгоритма рекомендует шифровать данные блоками размером от 4 Кбайт до 1 Мбайт.

Алгоритм Aardvark является существенно более быстрым, чем его предшественники — алгоритмы Bear и Lion — зашифровывание данных алгоритмом Aardvark выполняется на 20–70 % быстрее, чем упомянутыми алгоритмами (в зависимости от размера блока в указанных выше пределах) [271].

Тем не менее, Aardvark не вызвал столь большого интереса криптологов, как алгоритмы Bear и Lion. Какие-либо работы, посвященные криptoанализу данного алгоритма, не получили широкой известности, поэтому криптостойкость алгоритма Aardvark не определена.

### 3.3. Алгоритм AES (Rijndael)

Название AES присвоено алгоритму Rijndael — победителю конкурса AES. Конкурс AES подробно описан в разд. 2.1, а здесь рассмотрим структуру алгоритма и результаты его криptoанализа.

#### Структура алгоритма

Алгоритм AES представляет блок данных в виде двумерного байтового массива размером  $4 \times 4$ . Все операции производятся над отдельными байтами массива, а также над независимыми столбцами и строками.

В каждом раунде алгоритма выполняются следующие преобразования (рис. 3.7) [132, 153]:

1. Операция SubBytes, представляющая собой табличную замену каждого байта массива данных согласно табл. 3.1 (рис. 3.8).

## Описание алгоритмов

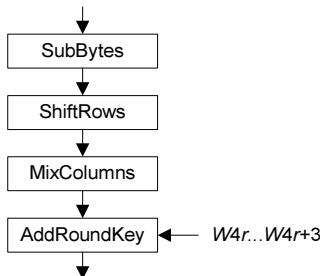


Рис. 3.7. Раунд алгоритма

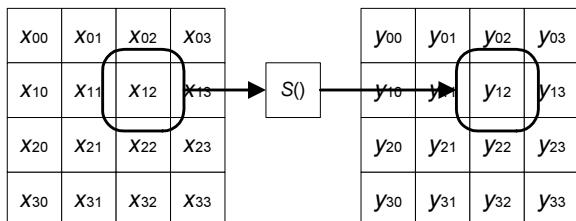


Рис. 3.8. Операция SubBytes

Таблица 3.1

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

Таблица меняет входное значение 0 на 63 (шестнадцатеричное значение), 1 — на 7C, 2 — на 77 и т. д.

Вместо данной табличной замены можно выполнить эквивалентную ей комбинацию двух операций:

- вычисление мультипликативной обратной величины от входного значения в конечном поле GF( $2^8$ ); обратной величиной от 0 является 0;
- выходное значение  $b$  вычисляется следующим образом:

$$b_i = a_i \oplus a_{i+4 \bmod 8} \oplus a_{i+5 \bmod 8} \oplus a_{i+6 \bmod 8} \oplus a_{i+7 \bmod 8} \oplus c_i,$$

где:

- ◊  $n_i$  обозначает  $i$ -й бит величины  $n$ ;
- ◊  $a$  — результат предыдущей операции;
- ◊  $c$  — шестнадцатеричная константа 63.

2. Операция ShiftRows, которая выполняет циклический сдвиг влево всех строк массива данных, за исключением нулевой (рис. 3.9). Сдвиг  $i$ -й строки массива (для  $i = 1, 2, 3$ ) производится на  $i$  байтов.



Рис. 3.9. Операция ShiftRows

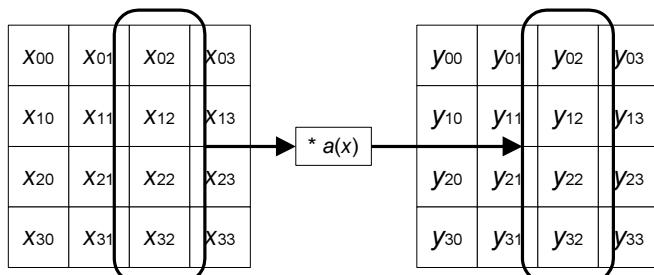


Рис. 3.10. Операция MixColumns

## Описание алгоритмов

3. Операция MixColumns. Выполняет умножение каждого столбца массива данных (рис. 3.10), который рассматривается как полином в конечном поле  $GF(2^8)$ , на фиксированный полином  $a(x)$ :

$$a(x) = 3x^3 + x^2 + x + 2.$$

Умножение выполняется по модулю  $x^4 + 1$ .

4. Операция AddRoundKey. Выполняет наложение на массив данных материала ключа. А именно, на  $i$ -й столбец массива данных ( $i = 0 \dots 3$ ) побитовой логической операцией «исключающее или» (XOR) накладывается определенное слово расширенного ключа  $W_{4r+i}$ , где  $r$  — номер текущего раунда алгоритма, начиная с 1 (процедура расширения ключа будет описана далее). Операция AddRoundKey представлена на рис. 3.11.

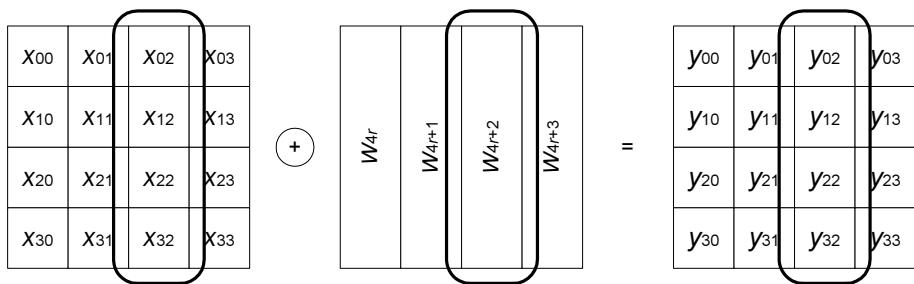


Рис. 3.11. Операция AddRoundKey

Зависимость количества раундов алгоритма  $R$  от размера ключа приведена в табл. 3.2.

Таблица 3.2

Размер ключа в битах	Количество раундов
128	10
192	12
256	14

Перед первым раундом алгоритма выполняется предварительное наложение материала ключа с помощью операции AddRoundKey, которая производит наложение на открытый текст первых четырех слов расширенного ключа  $W_0 \dots W_3$ .

Последний же раунд отличается от предыдущих тем, что в нем не выполняется операция MixColumns.

## Процедура расширения ключа

Алгоритм AES использует ключи шифрования трех фиксированных размеров: 128, 192 и 256 битов. В зависимости от размера ключа конкретный вариант алгоритма AES может обозначаться как AES-128, AES-192 и AES-256 соответственно [153].

Задача процедуры расширения ключа состоит в формировании нужного количества слов расширенного ключа для их использования в операции AddRoundKey. Как было сказано выше, под «словом» здесь понимается 4-байтный фрагмент расширенного ключа, один из которых используется в первичном наложении материала ключа и по одному — в каждом раунде алгоритма. Таким образом, в процессе расширения ключа формируется  $4 * (R + 1)$  слов.

Расширение ключа выполняется в два этапа, на первом из которых производится инициализация слов расширенного ключа (обозначаемых как  $W_i$ ): первые  $Nk$  ( $Nk$  — размер исходного ключа шифрования  $K$  в словах, т. е. 4, 6 или 8) слов  $W_i$ , т. е.  $i = 0 \dots (Nk - 1)$ , формируются их последовательным заполнением байтами ключа (рис. 3.12).

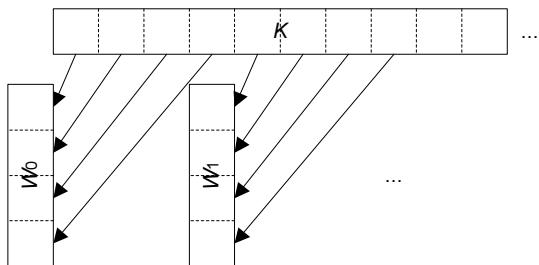


Рис. 3.12. Инициализация первых слов расширенного ключа

Последующие слова  $W_i$  формируются такой последовательностью операций для каждого  $i = Nk \dots (4 * (R + 1) - 1)$ :

1. Инициализируется временная переменная  $T$ :

$$T = W_{i-1}.$$

2. Эта переменная модифицируется таким образом:

- если  $i$  кратно  $Nk$ , то:

$$T = SubWord(RotWord(T)) \oplus RC_{i/Nk};$$

операции SubWord и RotWord будут описаны далее, а константы  $RC_n$  представляют собой слова, в которых все байты, кроме первого, являются нулевыми, а первый байт имеет значение  $2^{n-1} \bmod 256$ ;

- если  $Nk = 8$  и  $(i \bmod Nk) = 4$ , то:

$$T = SubWord(T);$$

- в остальных случаях модификация переменной  $T$  не выполняется.

3. Формируется  $i$ -е слово расширенного ключа:

$$W_i = W_{i-Nk} \oplus T.$$

Операция SubWord выполняет над каждым байтом входного значения табличную замену, которая была описана выше — см. операцию SubBytes.

Операция RotWord побайтно вращает входное слово на 1 байт влево.

Как видно, процедура расширения ключа является достаточно простой по сравнению со многими другими современными алгоритмами шифрования. Процедура расширения ключа имеет также несомненное достоинство в том, что расширение ключа может быть выполнено «на лету» (on-the-fly), т. е. параллельно с зашифровыванием данных.

Авторы алгоритма в [132] пишут также, что не следует задавать напрямую расширенный ключ — программная или аппаратная реализация алгоритма должна именно получать исходный ключ шифрования  $K$  и выполнять процедуру расширения ключа. Здесь стоит снова вспомнить алгоритм DES — известно, что DES с независимо задаваемыми ключами раундов оказался слабее против некоторых атак, чем исходный алгоритм DES (см. разд. 3.15).

## Расшифровывание

Расшифровывание выполняется применением обратных операций в обратной последовательности. Соответственно, перед первым раундом расшифровывания выполняется операция AddRoundKey (которая является обратной самой себе), накладывающая на шифртекст четыре последних слова расширенного ключа, т. е.  $W_{4R} \dots W_{4R+3}$ .

Затем выполняется  $R$  раундов расшифровывания, каждый из которых осуществляет следующие преобразования (рис. 3.13):

1. Операция InvShiftRows производит циклический сдвиг вправо трех последних строк массива данных на то же количество байтов, на которое выполнялся сдвиг операцией ShiftRows при зашифровывании.
2. Операция InvSubBytes производит побайтно обратную табличную замену, которая приведена в табл. 3.3.

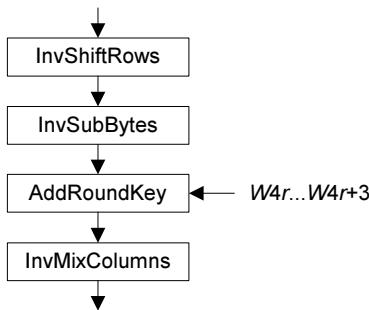


Рис. 3.13. Раунд расшифровывания

Таблица 3.3

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

Такую табличную замену можно выполнить, применив к входному байту преобразование, обратное второму действию альтернативной операции SubBytes (см. описание SubBytes), после чего вычислить мультиплексивную обратную величину от результата предыдущей операции в конечном поле  $GF(2^8)$ .

3. Операция AddRoundKey, как и при зашифровывании, выполняет наложение на обрабатываемые данные четырех слов расширенного ключа  $W_{4r} \dots W_{4r+3}$ . Однако нумерация раундов  $r$  при расшифровывании производится в обратную сторону — от  $(R - 1)$  до 0.
4. Операция InvMixColumns выполняет умножение каждого столбца массива данных аналогично прямой операции MixColumns, однако, умножение производится на полином  $a^{-1}(x)$ , определенный следующим образом:

$$a^{-1}(x) = Bx^3 + Dx^2 + 9x + E.$$

Аналогично зашифровыванию, последний раунд расшифровывания не содержит операцию InvMixColumns.

## Отличия AES от исходного алгоритма Rijndael

Алгоритм Rijndael позволяет шифровать данные не только 128-битными блоками, но и блоками по 192 или 256 битов. Таким образом, алгоритм AES, фактически, имеет лишь одно принципиальное отличие от Rijndael: он предусматривает использование только 128-битных блоков данных. Рассмотрим изменения в приведенном выше описании алгоритма AES, связанные с другими размерами блоков.

- Обрабатываемые данные могут представляться не только в виде массива размером  $4 \times 4$ , но и  $4 \times 6$  или  $4 \times 8$  для 192- и 256-битных блоков соответственно.
- Количество раундов  $R$  алгоритма Rijndael определяется табл. 3.4 в зависимости не только от размера ключа, но и от размера блока.

**Таблица 3.4**

Размер ключа в битах	Размер блока в битах		
	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

- Количество битов сдвига строк таблицы также зависит от размера блока (табл. 3.5).

Таблица 3.5

Номер строки	Количество битов сдвига		
	128	192	256
1	1	1	1
2	2	2	3
3	3	3	4

- Поскольку для 192- и 256-битного блоков увеличивается количество столбцов массива данных до 6 и 8 соответственно, в операции AddRoundKey участвуют уже 6 или 8 слов расширенного ключа вместо четырех. Следовательно, в  $r$ -м раунде алгоритма выполняется наложение слов расширенного ключа  $W_{Nb \cdot r}, \dots, W_{Nb \cdot r + 3}$ , где  $Nb$  — количество столбцов массива данных.
- В связи с вышесказанным, изменяется и процедура расширения ключа, однако изменение состоит лишь в том, что эта процедура должна выработать  $Nb \cdot (R + 1)$ , а не  $4 \cdot (R + 1)$  слов расширенного ключа (что, впрочем, остается справедливым для 128-битного блока).

## Первичная оценка криптостойкости алгоритма Rijndael

Первичная оценка криптостойкости алгоритма Rijndael была приведена авторами алгоритма в его спецификации [132], представленной на конкурс AES. Согласно оценкам авторов, Rijndael не подвержен следующим видам криптоаналитических атак:

- у алгоритма отсутствуют слабые ключи, а также возможности его вскрытия с помощью атак на связанных ключах (описание криптоаналитических атак, упомянутых здесь и далее, приведено в разд. 1.5–1.9);
- к алгоритму не применим дифференциальный криптоанализ;
- алгоритм не атакуем с помощью линейного криптоанализа и усеченных дифференциалов;
- Square-атака (специфичная атака на алгоритмы со структурой «квадрат», к которым относится и AES; подробно описана в [130]) также не применима к алгоритму Rijndael;
- алгоритм не вскрывается методом интерполяции.

## Криптоанализ алгоритма Rijndael в рамках конкурса AES

Сначала рассмотрим те результаты криптоанализа, которые были учтены экспертами при выборе алгоритм — победителя конкурса AES. Прежде всего, было найдено несколько атак на усеченные (с уменьшенным количеством раундов) версии алгоритма Rijndael [283]:

- в упомянутой выше первичной оценке криптостойкости были приведены некоторые атаки на усеченные версии Rijndael [132], из которых стоит отметить Square-атаку на 6-раундовую версию алгоритма, для которой необходимо  $2^{32}$  выбранных открытых текстов и  $2^{72}$  операций шифрования.
- Square-атака на 6-раундовую версию Rijndael была незначительно усиlena Эли Бихамом и Натаном Келлером (Nathan Keller) [73], которые также предложили атаку методом невозможных дифференциалов на 5-раундовую версию алгоритма; для нее требуется  $2^{29.5}$  выбранных открытых текстов и  $2^{31}$  операций; атака с помощью невозможных дифференциалов рядом корейских специалистов была распространена на 6-раундовую версию Rijndael [117]: для данной атаки требуется  $2^{91.5}$  выбранных открытых текстов и  $2^{122}$  операций шифрования;
- впоследствии Square-атака была расширена на 7 раундов алгоритма Rijndael; однако данная атака весьма непрактична: для ее реализации требуется  $2^{32}$  выбранных открытых текстов и от  $2^{184}$  до  $2^{208}$  операций шифрования [244];
- авторы алгоритма Twofish (финалиста конкурса AES) с участием ряда других специалистов продолжили усиление Square-атаки против алгоритма Rijndael: новая Square-атака позволяла вскрыть 6-раундовый Rijndael выполнением  $2^{44}$  операций, а 7-раундовый — от  $2^{155}$  до  $2^{172}$  операций шифрования при том же требуемом количестве выбранных открытых текстов [147]; кроме того, новая атака позволяет вскрыть 7- и 8-раундовые (последнюю — только при использовании 256-битного ключа) версии Rijndael при наличии от  $2^{119}$  до  $2^{128}$  выбранных открытых текстов выполнением, соответственно,  $2^{120}$  или  $2^{204}$  операций;
- та же команда криптологов предложила атаку на связанных ключах на 9-раундовую версию Rijndael с 256-битным ключом, которой необходимо  $2^{77}$  выбранных открытых текстов, зашифрованных на 256 связанных ключах, и  $2^{224}$  операций шифрования.

С учетом того, что в алгоритме Rijndael выполняется, как минимум, 10 раундов, запас криптостойкости алгоритма экспертами был признан адекватным [283]. С этим, однако, согласны далеко не все специалисты. В частности, в материалах [51, 215, 348, 350] достаточным запасом криптостойкости считается двукратное увеличение количества раундов алгоритма по сравнению с максимально атакуемым (однако стоит отметить, что далеко не все атаки на Rijndael с уменьшенным количеством раундов являются осуществимыми на практике [372]). В [51] достаточно убедительно утверждается, что алгоритм, выбранный в качестве стандарта AES, должен оставаться криптографически стойким до 2100 г. Ясно, что за почти сто лет будет сделано огромное количество попыток вскрытия AES («AES будет наиболее привлекательной мишенью для лучших криptoаналитиков мира» [350]), некоторые из которых приведут к увеличению количества вскрываемых раундов. Судя по последнему десятилетию, появятся и принципиально новые виды криptoаналитических атак. Поэтому, считая формально, согласно [51] и [348], Rijndael должен выполнять, как минимум, 18 раундов.

Авторы [348] утверждают, что в алгоритме Rijndael сделан чрезмерный акцент на быстродействие в ущерб криптостойкости, и рекомендуют даже 24 раунда, а не 18. Ясно, что в этом случае Rijndael по быстродействию будет заметно проигрывать другим алгоритмам — финалистам конкурса AES. Они же считают, что успешные атаки на алгоритм Rijndael происходят именно благодаря его чрезвычайно простой структуре. Стоит сказать, что документ [348] также написан авторами «конкурирующего» с Rijndael алгоритма Twofish; в нем настолько убедительно доказывается преимущество Twofish над остальными финалистами конкурса (аналогичные доказательства приведены и в работе [350]), что после его прочтения выбор Rijndael (а не Twofish) в качестве стандарта AES начинает удивлять.

В рамках исследований в течение первого раунда конкурса AES было также отмечено [284], что Rijndael имеет ряд потенциальных уязвимостей при реализации данного алгоритма в смарт-картах.

- Rijndael может быть подвержен атаке по потребляемой мощности, нацеленной на его процедуру расширения ключа [80], что, однако, не доказано авторами [80] Эли Бихамом и Эди Шамиром.
- Весьма интересное исследование [114] на данную тему было проведено специалистами исследовательского центра компании IBM. Они выполнили реализацию алгоритма Twofish для типичной смарт-карты с CMOS-архитектурой и проанализировали возможность атаки на данный алгоритм с помощью дифференциального анализа потребляемой мощности (DPA — см. разд. 1.7). Оказалось, что атаке подвержена процедура входного обес-

ливания алгоритма Twofish, в результате чего можно вычислить ключ шифрования данного алгоритма. Атака была (теоретически) распространена на остальные алгоритмы — участники конкурса AES. Аналогично алгоритму Twofish, с помощью DPA атакуется предварительное наложение материала ключа, выполняемое в алгоритме Rijndael, после чего вычисляется ключ шифрования целиком. Авторы [114] утверждают, что среди алгоритмов — финалистов конкурса AES алгоритмы MARS и RC6 подвержены этой атаке несколько менее, чем Twofish, Rijndael и Serpent.

Эти потенциальные уязвимости не повлияли на выбор алгоритма Rijndael в качестве стандарта AES, поскольку, во-первых, остальные алгоритмы-финалисты не принципиально лучше в данном контексте, а во-вторых, существуют различные методы противодействия атакам по потребляемой мощности (в частности, описаны в [114]), которые, однако, усложняют реализацию и ухудшают быстродействие алгоритма.

Среди других исследований можно отметить работу [277], в которой утверждается, что алгоритм Rijndael не обеспечивает достаточное рассеивание (распространение влияния одного бита открытого текста на несколько битов шифртекста) данных (это указано и в [350], где предложены конкретные меры усиления алгоритма), однако конкретных атак в [277] предложено не было. Дискуссия между авторами [277] и авторами Rijndael продолжилась [133, 276], однако эта потенциальная проблема также не повлияла на выбор алгоритма-победителя.

## Криptoанализ алгоритма после конкурса AES

Как и предполагали эксперты, после принятия алгоритма Rijndael в качестве стандарта AES попытки вскрытия этого алгоритма существенно усилились.

Можно сказать, что криptoанализ алгоритма AES стал развиваться, в основном, в следующих четырех направлениях.

Во-первых, были предприняты попытки усиления «классических» атак или применения других известных атак к данному алгоритму. Например, работа [84] описывает атаку методом бумеранга на 6-раундовую версию алгоритма со 128-битным ключом, для выполнения которой требуется  $2^{39}$  выбранных открытых текстов,  $2^{71}$  шифртекстов с аддитивным выбором и  $2^{71}$  операций шифрования.

Во-вторых, имело место применение различных методов криptoанализа на связанных ключах, в частности:

- в работе [406] предложено несколько вариантов атак на связанных ключах на 7- и 8-раундовый AES-192 с использованием невозможных дифференциалов;

- комбинация метода бумеранга и связанных ключей предложена в работе [70]: 9-раундовый AES-192 атакуется при наличии  $2^{79}$  выбранных открытых текстов, каждый из которых шифруется на 256 связанных ключах, выполнением  $2^{125}$  операций шифрования; для атаки на 10-раундовый AES-256 требуется  $2^{114,9}$  выбранных открытых текстов (включая зашифровывание на 256 связанных ключах) и  $2^{171,8}$  операций; данная атака использует слабость процедуры расширения ключа, состоящую в ее недостаточной нелинейности;
- эта атака была усиlena в работе [202], в которой, в частности, предлагается атака на 10-раундовый алгоритм AES-192; для новой атаки требуется  $2^{125}$  выбранных открытых текстов (на 256 связанных ключах) и  $2^{146,7}$  операций.

Несмотря на то, что предложенные атаки на связанных ключах являются весьма непрактичными, настораживает тот факт, что атаке подвержены уже 10 из 12 раундов алгоритма AES-192 (и это после всего 5 лет после принятия стандарта AES!) — возникает опасение, что эксперты (указывающие на недостаточность раундов в алгоритме Rijndael) были правы и полнораундовый алгоритм AES будет вскрыт существенно раньше, чем предполагали эксперты института NIST.

В-третьих, многие исследования были посвящены алгебраической структуре алгоритма Rijndael, например:

- в работе [156] найдены линейные соотношения в таблице замен Rijndael (т. е. в единственном нелинейном элементе алгоритма); однако, как и в других аналогичных работах, никаких-либо практических возможностей использования этого свойства не предложено;
- как показано в работе [149], зашифровывание с помощью Rijndael можно выразить относительно (особенно по сравнению с другими «серезными» алгоритмами шифрования) простой формулой; авторы не нашли практического применения данной формулы, но предположили, что она будет использована в реальных атаках в течение ближайших примерно 20 лет;
- в работе [275] показано, что вскрытие алгоритма AES эквивалентно решению системы квадратичных уравнений в конечном поле  $GF(2^8)$ .

Попытки использования алгебраических свойств алгоритма для его вскрытия были названы «алгебраическими атаками» [121]. Стоит отметить, что были и работы с попытками доказательства того факта, что простая структура алгоритма AES не ухудшает его криптостойкости, например, [361].

В-четвертых, больше всего исследований было посвящено атакам, использующим информацию, полученную по побочным каналам:

- во многих работах содержатся примеры успешного вскрытия различных реализаций полнорундового алгоритма AES с помощью атак по времени выполнения (например, [49]), потребляемой мощности (например, [248]) и атак на основе сбоев (например, [299]); автор [49] (эта работа описывает успешное вскрытие сервера OpenSSL, использующего для шифрования алгоритм AES), в частности, считает, что эксперты NIST при выборе Rijndael в качестве AES допустили весьма серьезную ошибку, посчитав, что время выбора значения из таблицы замен является константной величиной в конкретной реализации; вывод автора таков: выбор Rijndael, скорее всего, был ошибочным;
- не меньше исследований (например, [124, 349]) посвящено безопасным (т. е. защищенным от утечки данных по побочным каналам) программным или аппаратным реализациям AES.

Довольно большой список работ, посвященных side-channel-атакам на AES и методам защиты от них, можно найти, например, на ресурсе [373], посвященном криптоанализу AES.

## Заключение

Итак, на май 2007 г., по прошествии всего 6 лет после принятия алгоритма Rijndael в качестве стандарта AES, криптоаналитики весьма серьезно продвинулись во вскрытии данного алгоритма:

- предложена теоретическая атака уже на 10 раундов (из 12) алгоритма AES-192;
- существует множество примеров вскрытия реализаций алгоритма AES с помощью side-channel-атак.

Не правда ли, все это производит впечатление, что эксперты NIST могли ошибиться в выборе алгоритма — победителя конкурса AES?

## 3.4. Алгоритм Akelarre

Алгоритм Akelarre разработан коллективом испанских криптографов. Его отличительной особенностью является то, что структура алгоритма, фактически, представляет собой комбинацию преобразований, использованных в двух более ранних алгоритмах, хорошо зарекомендовавших себя с точки зрения криптостойкости: IDEA и RC5.

Алгоритм шифрует данные блоками по 128 битов. Как и в алгоритме RC5, часть основных параметров алгоритма являются переменными: может изменяться количество раундов алгоритма  $R$ , а ключ шифрования может иметь любой размер, кратный 64 битам (оптимальным считается 128-битный ключ).

## Структура алгоритма

Структура алгоритма (рис. 3.14) [148, 222] весьма похожа на структуру алгоритма IDEA. Разница, прежде всего, в основных преобразованиях, используемых в каждом раунде алгоритма, а также в размере слова: 32 бита вместо 16-битного слова у IDEA — шифруемый 128-битный блок разбивается на 4 субблока  $A$ ,  $B$ ,  $C$  и  $D$  по 32 бита, над которыми и выполняются криптографические преобразования.

Алгоритм состоит из начального преобразования,  $R$  раундов и финального преобразования.

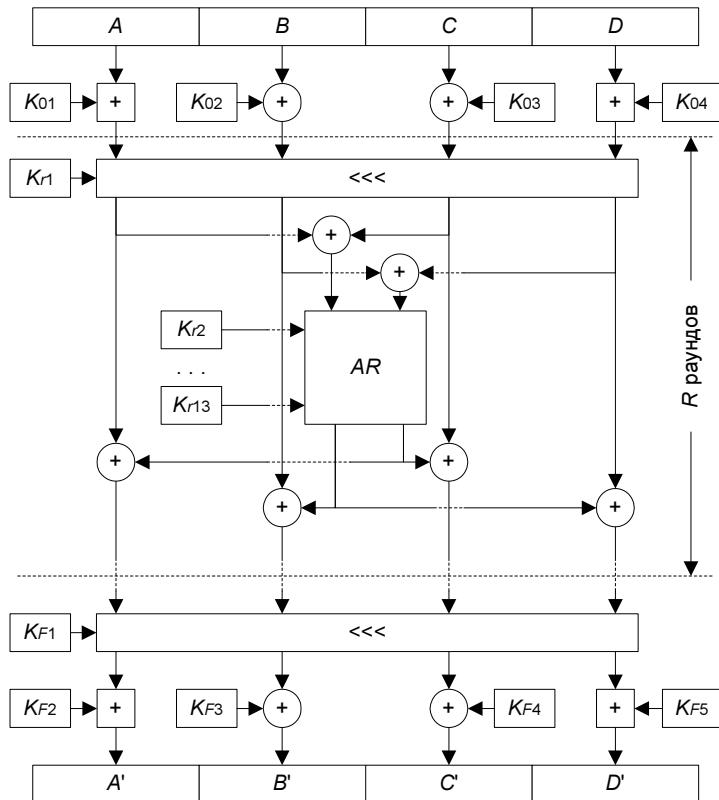


Рис. 3.14. Структура алгоритма Akelarre

Начальное преобразование представляет собой наложение фрагментов расширенного ключа  $K_{01}...K_{04}$  на субблоки:

$$A = A + K_{01} \bmod 2^{32};$$

$$B = B \oplus K_{02};$$

$$C = C \oplus K_{03};$$

$$D = D + K_{04} \bmod 2^{32}.$$

В каждом раунде алгоритма  $r$  выполняются следующие преобразования:

- Субблоки  $A, B, C, D$  объединяются в 128-битный блок, над которым выполняется циклический сдвиг на переменное число битов, определяемое семью младшими битами фрагмента ключа  $K_{r1}$  (именно применяемые в данном алгоритме операции циклического сдвига на переменное число битов считаются заимствованными у алгоритма RC5).
- 128-битный блок снова разбивается на 4 фрагмента, после чего вычисляются следующие промежуточные величины:

$$T1 = A \oplus C;$$

$$T2 = B \oplus D.$$

- Значения  $T1$  и  $T2$ , а также 12 фрагментов ключей  $K_{r2}...K_{r13}$  подаются на вход AR-модуля (addition-rotation structure), выполняющего операции сложения и циклического сдвига на переменное число битов. AR-модуль будет подробно описан далее.
- Выходные значения раунда формируются следующим образом:

$$A = A \oplus T2';$$

$$B = B \oplus T1';$$

$$C = C \oplus T2';$$

$$D = D \oplus T1',$$

где  $T1'$  и  $T2'$  — выходные значения AR-модуля.

AR-модуль предполагает выполнение следующих операций (на примере  $T1$ , аналогичные операции выполняются над  $T2$ , см. упрощенную схему на рис. 3.15):

- 31 старший бит  $T1$  циклически сдвигается влево на величину, определяемую текущим значением 5 младших битов  $T2$ .
- Результат предыдущего шага складывается с фрагментом ключа раунда  $K_{r8}$ :

$$T1 = T1 + K_{r8} \bmod 2^{32}.$$

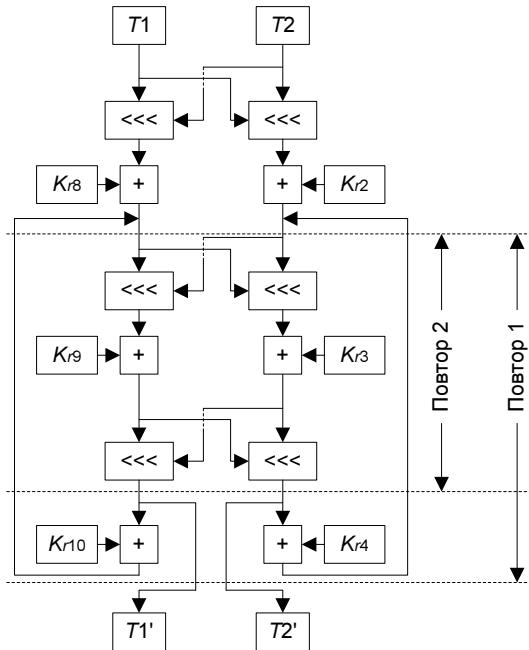


Рис. 3.15. AR-модуль

3. 31 младший бит  $T1$  циклически сдвигается влево аналогично шагу 1.
4.  $T1$  складывается с фрагментом ключа раунда  $K_{r9}$  аналогично шагу 2.
5. 31 старший бит  $T1$  циклически сдвигается влево аналогично шагу 1.
6.  $T1$  складывается с фрагментом ключа раунда  $K_{r10}$  аналогично шагу 2.
7. Шаги 3–6 повторяются с использованием  $K_{r11}$  и  $K_{r12}$ . Затем повторяются шаги 3–5 с использованием фрагмента ключа раунда  $K_{r13}$ . Результат последнего повторения шага 5 становится выходным значением  $T1'$ .

Аналогичным образом обрабатывается  $T2$ , но с использованием фрагментов расширенного ключа  $K_{r2}...K_{r7}$  вместо  $K_{r8}...K_{r13}$ .

Финальное преобразование состоит из циклического сдвига 128-битного блока влево на количество битов, определяемое значением семи младших битов фрагмента ключа  $K_{F1}$ , после чего выполняются следующие действия:

$$A' = A + K_{F2} \bmod 2^{32};$$

$$B' = B \oplus K_{F3};$$

$$C' = C \oplus K_{F4};$$

$$D' = D + K_{F5} \bmod 2^{32}.$$

## Процедура расширения ключа

Как было сказано выше, алгоритм может использовать ключ любого размера, кратного 64 битам. Однако основной размер ключа — 128 битов, поэтому стоит рассмотреть процедуру расширения именно 128-битного ключа. Данная процедура выполняется следующим образом (рис. 3.16):

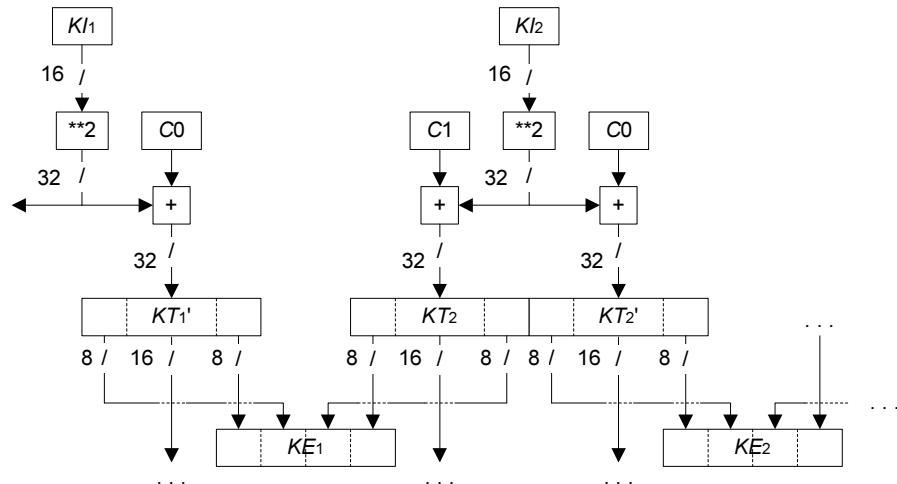


Рис. 3.16. Расширение ключа

- Ключ шифрования разбивается на 8 фрагментов по 16 битов  $KI_1 \dots KI_8$ .
- Каждый 16-битный фрагмент возводится в квадрат с получением 32-битного значения, которое складывается с константами  $C0$  и  $C1$  следующим образом:

$$KT_i = {KI_i}^2 + C1 \bmod 2^{32};$$

$$KT'_i = {KI_i}^2 + C0 \bmod 2^{32}.$$

Константы  $C0$  и  $C1$  определены так (указаны шестнадцатеричные значения):

$$C0 = A49ED284;$$

$$C1 = 735203DE.$$

- 8 младших и 8 старших битов временных значений  $KT_i$  и  $KT'_i$  формируют фрагменты предварительного расширенного ключа  $KE_1 \dots KE_8$ :  $KE_i$  — результат конкатенации следующих величин:

- 8 младших битов  $KT_i'$ ;

- 8 старших битов  $KT_i'$ ;
  - 8 младших битов  $KT_{(i \bmod 8)+1}$ ;
  - 8 старших битов  $KT_{(i \bmod 8)+1}$ .
4. Средние 16 битов временных значений обрабатываются аналогично  $KI_1 \dots KI_8$  для получения новых значений фрагментов предварительного расширенного ключа (с незначительными отличиями — см. рис. 3.16).
5. Ключи  $K_{01} \dots K_{04}$ ,  $K_{r1} \dots K_{r13}$  (для каждого раунда) и  $K_{F1} \dots K_{F5}$  заполняются поочередно вычисляемыми фрагментами  $KE_1 \dots KE_8$ .

Процедура расширения ключа выглядит достаточно сложной. Однако данная процедура имеет существенный недостаток — по получаемым в результате криptoаналитических вычислений частям ключей раундов можно восстановить соответствующие им биты ключа шифрования; данное свойство эксплуатируется в описанных далее атаках на алгоритм.

## Криптоанализ алгоритма

Алгоритм Akelarre был представлен в 1996 г., а уже в следующем году Нильс Фергюсон (Niels Ferguson) и Брюс Шнайер описали атаку на Akelarre, позволяющую вскрыть алгоритм на основе не более ста выбранных открытых текстов и требующую выполнения  $2^{42}$  операций шифрования [148]. Фергюсон и Шнайер сделали вывод о неприменимости алгоритма Akelarre (поскольку найденную ими атаку возможно применить на практике) и предложили несколько путей усовершенствования алгоритма, в том числе полную переработку особенно слабой процедуры расширения ключа. Разработчики Akelarre предложили такую версию в том же 1997 г., однако и новая версия алгоритма оказалась достаточно слабой [148].

Еще более серьезную атаку предложили Ларс Кнудсен и Винсент Риджмен [222]. Они нашли способ раскрытия ключа алгоритма с вероятностью около 70 % на основе всего около 1000 блоков шифртекста при наличии некоторой информации о соответствующих им открытых текстах (например, достаточно знания того, что это текст на английском языке в ASCII-кодировке). Стоит сказать, что по сравнению с другими видами атак на алгоритмы шифрования, атака на основе только шифртекста наиболее легко применима на практике, поскольку для получения требуемой для атаки информации злоумышленнику достаточно простого прослушивания канала, по которому передается зашифрованная информация. Удивителен и тот факт, что обе атаки практически не зависят ни от количества раундов, ни от размера ключа алгоритма.

Статья [222] с описанием атаки Кнудсена и Риджмена называется «Two Rights Sometimes Make a Wrong», что можно перевести примерно как «Иногда два плюса дают минус». Удивительно, но алгоритм Akelarre, сочетающий в себе преобразования двух хорошо известных и криптографически стойких алгоритмов, оказался впечатляюще слаб.

### 3.5. Алгоритм Anubis

Блочный шифр Anubis разработан специально для участия в конкурсе NESSIE международным дуэтом авторов: бельгийцем Винсентом Риджменом и бразильцем Пауло Баррето (Paulo S. L. M. Barreto).

Алгоритм назван в честь древнеегипетского бога Анубиса — бога бальзамирования (embalming) и погребения (entombment); к его ведению авторы алгоритма решили отнести и криптографию [43].

Anubis шифрует данные блоками по 128 битов с использованием ключа размером от 128 до 320 битов; размер ключа должен быть кратен 32 битам.

Данный алгоритм продолжает серию алгоритмов, соавтором которых является Винсент Риджмен: Square, SHARK и Rijndael. Все эти алгоритмы объединяет их относительно редко встречающаяся (напротив, частая среди алгоритмов — участников конкурса NESSIE) структура типа «квадрат» и весьма схожий набор выполняемых преобразований.

### Структура алгоритма

Алгоритм представляет блок шифруемых данных в виде 16-байтового массива, который для удобства описания представлен в виде квадрата размером  $4 \times 4$  байта. В каждом раунде алгоритма выполняются следующие действия [43]:

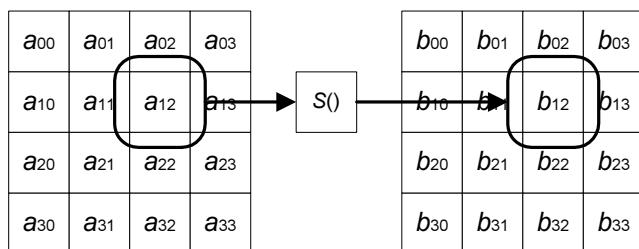


Рис. 3.17. Операция  $\gamma$  алгоритма Anubis

1. Табличная замена  $\gamma$  (рис. 3.17), выполняемая согласно таблице  $S$  (табл. 3.6, указаны шестнадцатеричные значения).

Таблица 3.6

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

## Описание алгоритмов

Значения таблицы выбраны псевдослучайным образом с учетом необходимости ее соответствия следующему соотношению:

$$S(S(x)) = x.$$

2. Операция  $\tau$  — байтовая перестановка, простейшим образом преобразующая строку обрабатываемого блока ключевой информации в столбец (рис. 3.18):

$$b_{i,j} = a_{j,i},$$

где  $a_{i,j}$  и  $b_{i,j}$  — байты массива данных до и после выполнения текущей операции соответственно.

$a_{00}$	$a_{01}$	$a_{02}$	$a_{03}$
$a_{10}$	$a_{11}$	$a_{12}$	$a_{13}$
$a_{20}$	$a_{21}$	$a_{22}$	$a_{23}$
$a_{30}$	$a_{31}$	$a_{32}$	$a_{33}$

$a_{00}$	$a_{10}$	$a_{20}$	$a_{30}$
$a_{01}$	$a_{11}$	$a_{21}$	$a_{31}$
$a_{02}$	$a_{12}$	$a_{22}$	$a_{32}$
$a_{03}$	$a_{13}$	$a_{23}$	$a_{33}$

Рис. 3.18. Операция  $\tau$  алгоритма Anubis

3. Операция  $\theta$  (рис. 3.19), представляющая собой умножение массива на фиксированную матрицу  $H$  (табл. 3.7).

Таблица 3.7

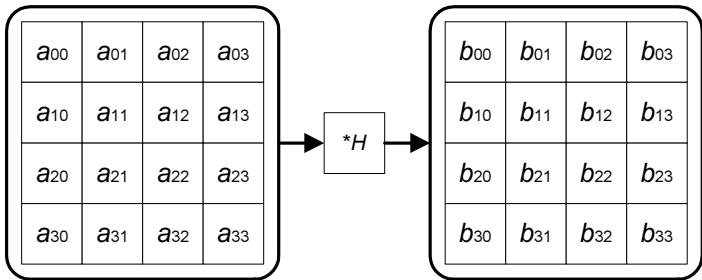
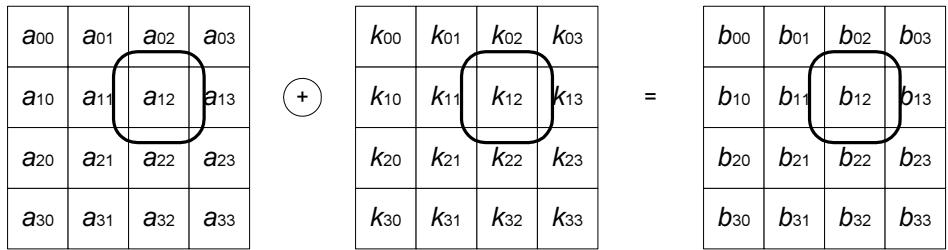
1	2	4	6
2	1	6	4
4	6	1	2
6	4	2	1

Умножение выполняется в конечном поле  $GF(2^8)$ .

4. Наложение ключа  $r$ -го раунда  $k[r]$  (процедура расширения ключа будет подробно описана далее); выполняется побитовой логической операцией «исключающее или» (XOR), применяемой к каждому биту массива данных и соответствующему биту  $k[r]$  (рис. 3.20):

$$b_{i,j} = a_{i,j} \oplus k[r]_{i,j}.$$

Эта операция обозначается как  $\sigma$ .

Рис. 3.19. Операция  $\theta$  алгоритма AnubisРис. 3.20. Операция  $\sigma$  алгоритма Anubis

Перечисленные операции выполняются в каждом раунде в указанной последовательности ( $\gamma$ ,  $\pi$ ,  $\theta$ ,  $\sigma$ ), за исключением последнего раунда алгоритма, в котором не выполняется операция  $\theta$ . Кроме того, перед первым раундом выполняется входное отбеливание данных путем наложения на шифруемый блок операцией XOR нулевого подключа.

Стоит отметить, что все перечисленные операции являются обратными самим себе, соответственно, расшифровывание выполняется с помощью тех же операций в том же порядке, что и при зашифровывании. Меняется только порядок использования подключей на обратный.

Число раундов алгоритма  $R$  зависит от размера ключа шифрования и определяется следующим образом:

$$R = 8 + N,$$

где  $N$  — размер ключа в 32-битных фрагментах.

Некоторые криптоаналитики считают, что количество раундов в алгоритме Anubis несколько завышено, видимо, с целью обеспечить более высокий запас криптостойкости [82].

## Процедура расширения ключа

Процедура расширения ключа достаточно проста и основана, практически, на той же последовательности операций, которые применяются в раундах алгоритма. Расширение ключа выполняется следующим образом: сначала исходный ключ шифрования  $K$  представляется в виде байтового массива  $4 \times 4$  (что обозначается как  $KI_0$ ), после чего в цикле выполняются следующие операции:

1. Итеративно вычисляются остальные промежуточные ключи  $KI_1 \dots KI_R$ :

$$KI_j = f(KI_{j-1}),$$

где  $f()$  — совокупность операций  $\gamma$ ,  $\pi$ ,  $\theta$  и  $\sigma$ ; в операции  $\sigma$  в качестве ключа раунда используется соответствующая из констант  $c[r]$ , которые, в свою очередь, определяются следующим образом:

$$c[r]_{0,j} = S(4 * (r - 1) + j)$$

для  $j = 0 \dots 3$  (см. описание операции  $\sigma$ ). Остальные байты  $c[r]_{i,j}$  являются нулевыми.

Операция  $\pi$  — циклический сдвиг столбцов таблицы вниз по следующему простому правилу:  $j$ -й столбец сдвигается на  $j$  позиций (рис. 3.21).

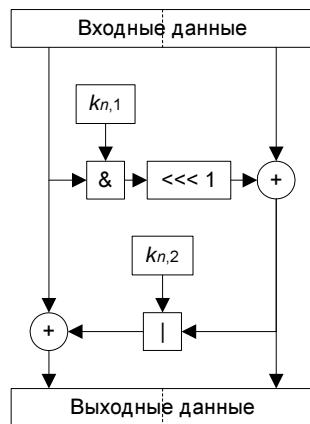


Рис. 3.21. Операция  $\pi$  алгоритма Anubis

2. На основе предварительных ключей вычисляются подключи  $k_0 \dots k_R$ :

$$k_n \dots g(KI_n),$$

где функция  $g()$  представляет собой последовательное выполнение следующих операций:  $\gamma$ ,  $\omega$ ,  $\tau$ .

Операции  $\gamma$  и  $\tau$  были описаны выше, а  $\omega$  представляет собой умножение блока на фиксированную матрицу  $V$  (аналогично операции  $\theta$  — см. рис. 3.19), приведенную в табл. 3.8.

**Таблица 3.8**

1	1	1	1
1	2	4	8
1	6	36	216
1	8	64	0

Стоит отметить, что при зашифровывании данных процедура расширения ключа позволяет вычислять ключи «на лету», т. е. по мере необходимости. Однако при расшифровывании необходимо полностью выполнить расширение ключа до начала преобразований.

## Достоинства и недостатки алгоритма

Прежде всего стоит сказать, что на конкурсе NESSIE была рассмотрена несколько модифицированная версия алгоритма Anubis, отличие которой от описанной здесь состояло лишь в операции  $\gamma$ . Модифицированная операция  $\gamma$  вместо одной табличной замены  $8 \times 8$  битов выполняла 3 уровня табличных замен  $4 \times 4$ . Данная модификация произведена для упрощения аппаратной реализации алгоритма [307].

Anubis был признан одним из наиболее быстродействующих алгоритмов шифрования (из участников конкурса) [308]. Еще одно явное достоинство алгоритма — отсутствие проблем с криптостойкостью [311]. Однако злую шутку с алгоритмом Anubis сыграло его сходство с алгоритмом Rijndael (который, как известно, является новым стандартом шифрования США под названием AES — см. разд. 3.3), который также рассматривался в рамках конкурса NESSIE. Эксперты конкурса посчитали, что при таком явном сходстве Anubis не может иметь настолько серьезных преимуществ перед алгоритмом Rijndael, которые позволили бы ему выиграть у Rijndael в финале конкурса NESSIE [308, 312]. Поэтому Anubis не был выбран во второй этап конкурса.

## 3.6. Алгоритмы Bear, Lion и Lioness

Эти три алгоритма были предложены в 1995 г. известным криптологом, профессором Кембриджского университета Россом Андерсоном в соавторстве с не менее известным криptoаналитиком Эли Бихамом.

Алгоритмы Bear, Lion и Lioness основаны на комбинировании преобразований, выполняемых над шифруемыми данными алгоритмом хэширования и алгоритмом генерации псевдослучайных чисел, определенная последовательность применения которых и представляет собой алгоритм шифрования [34].

Известно множество алгоритмов симметричного шифрования, полученных путем многократного повторения каким-либо образом других алгоритмов шифрования. Простейший пример — алгоритм Triple DES, представляющий собой трехкратное шифрование обычным алгоритмом DES и известный с 1978 г. Есть и Double DES (двойной), и Quadruple DES (четырехкратный), первый из которых, практически, не сильнее обычного DES, а второй — чрезвычайно медленный. Для того же DES придуманы и более сложные варианты комбинирования — например, алгоритм Ladder-DES, названный так из-за своей относительно сложной «лестничной» структуры, «ступеньками» которой является обычный DES (DES и его варианты описаны в разд. 3.15).

Другой вариант получения составных шифров — последовательное использование нескольких различных алгоритмов шифрования с независимыми ключами. Такой составной алгоритм шифрования при использовании действительно независимых ключей и стойких применяемых алгоритмов является весьма стойким, но имеет множество недостатков, в частности, низкую скорость шифрования, поэтому последовательное шифрование различными алгоритмами не нашло широкого применения (см., например, [194]).

Использовать в алгоритмах блочного симметричного шифрования другие типы криптографических преобразований предложили в 1988 г. Майкл Любы (Michael Luby) и Чарльз Ракофф (Charles Rackoff) [28]. Алгоритмы Bear, Lion и Lioness являются одним из воплощений этой идеи. Рассмотрим структуру данных алгоритмов.

## Алгоритм Bear

Как было сказано выше, Bear — комбинация двух криптографических функций: алгоритма хэширования  $H$  и генератора псевдослучайной последовательности  $S$ , — шифрование с помощью которых выполняется следующим образом (рис. 3.22) [34]:

$$\begin{aligned}L &= L \oplus H_{k1}(R); \\R &= R \oplus S(L); \\L &= L \oplus H_{k2}(R),\end{aligned}$$

где:

- $k1$  и  $k2$  — независимые подключи одинакового размера  $k$ ,  $k > b1$  (см. далее);

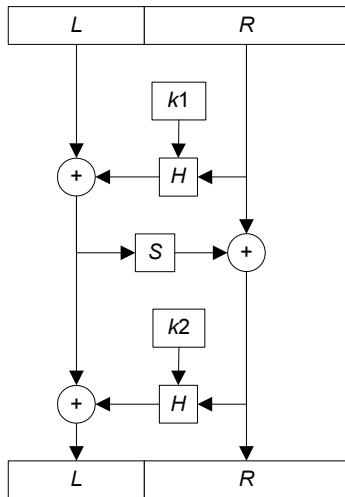


Рис. 3.22. Алгоритм Bear

□  $L$  и  $R$  — соответственно, левая и правая части (субблоки) шифруемого блока данных.

В качестве хэш-функции автор алгоритма предлагает использовать SHA-1 или MD5. Данные алгоритмы хэширования являются бесключевыми, однако автор предлагает вставлять ключ до и после хэшируемых данных, например:

$$H_{k1}(R) = MD5(k1 \bullet R \bullet k1),$$

где  $\bullet$  — операция конкатенации.

Как видно из описания, блок данных делится на субблоки различного размера: размер  $L$  (обозначим его  $b1$ ) должен быть эквивалентен размеру выходного значения используемой хэш-функции, например,  $b1 = 160$  битов для SHA-1; соответственно, размер  $R$  равен:

$$b2 = m - b1 \text{ битов},$$

где  $m$  — размер шифруемого блока в битах. Такая структура называется *небалансированной* сетью Фейстеля — в отличие от нее, в классической («сбалансированной») сети Фейстеля субблоки  $L$  и  $R$  имеют одинаковый размер.

Расшифровывание данных выполняется так:

$$L = L \oplus H_{k2}(R);$$

$$R = R \oplus S(L);$$

$$L = L \oplus H_{k1}(R),$$

т. е., фактически, представляет собой зашифровывание с использованием подключей в обратной последовательности, что характерно для многих алгоритмов, основанных на сети Фейстеля.

Алгоритм Bear не является в полной мере алгоритмом шифрования — это только шаблон для создания на его основе конкретных алгоритмов шифрования, поскольку в Bear не определены, как минимум, следующие параметры:

- размер блока  $m$  (предполагается, что алгоритм предназначен для шифрования больших блоков, предпочтительно от 1 Кбайт до 1 Мбайт);
- размер ключа шифрования;
- используемые алгоритмы  $H$  и  $S$ .

## Алгоритм Lion

Алгоритм Lion отличается от Bear лишь последовательностью использования алгоритмов  $H$  и  $S$ , а также фрагментов ключа шифрования. Зашифровывание выполняется следующим образом (рис. 3.23) [34]:

$$R = R \oplus S(L \oplus k1);$$

$$L = L \oplus H(R);$$

$$R = R \oplus S(L \oplus k2).$$

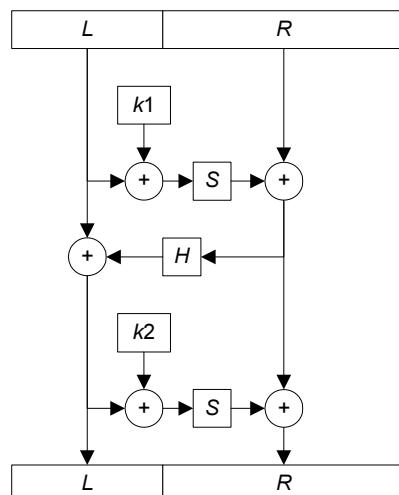


Рис. 3.23. Алгоритм Lion

Аналогично предыдущему алгоритму, расшифровывание выполняется таким же образом, но с использованием подключей «наоборот»:

$$R = R \oplus S(L \oplus k2);$$

$$L = L \oplus H(R);$$

$$R = R \oplus S(L \oplus k1).$$

Как и в алгоритме Bear, Lion делит шифруемый блок на 2 субблока  $L$  и  $R$  различного размера:  $b1$  и  $b2$ , которые выбираются аналогичным образом. Ключ шифрования также делится на два подключа  $k1$  и  $k2$ , размер каждого из которых равен  $b1$ .

## Алгоритм Lioness

Авторы алгоритма предположили, что Lion может быть подвержен сложной в реализации комбинированной атаке на основе аддитивного выбора открытого текста и шифртекста. Поэтому Андерсон и Бихам предложили более сложный (соответственно, более медленный) алгоритм шифрования Lioness, не подверженный подобной атаке [34].

В отличие от «трехраундовых» Bear и Lion, преобразования алгоритма Lioness выполняются в 4 раунда, причем в каждом раунде используется один из независимых подключей алгоритма  $k1...k4$  (рис. 3.24):

$$R = R \oplus S(L \oplus k1);$$

$$L = L \oplus H_{k2}(R);$$

$$R = R \oplus S(L \oplus k3);$$

$$L = L \oplus H_{k4}(R).$$

Расшифровывание выполняется применением обратной последовательности операций:

$$L = L \oplus H_{k4}(R);$$

$$R = R \oplus S(L \oplus k3);$$

$$L = L \oplus H_{k2}(R);$$

$$R = R \oplus S(L \oplus k1).$$

Фактически алгоритм Lioness комбинирует преобразования, используемые в алгоритмах Bear и Lion. Аналогично алгоритму Bear, подключи  $k2$  и  $k4$  имеют произвольный размер, больший значения  $b1$ , а подключи  $k1$  и  $k3$  (аналогично алгоритму Lion) имеют размер  $b1$ .

Как и алгоритм Bear, алгоритмы Lion и Lioness являются шаблонными, не конкретизирующими основные используемые параметры.

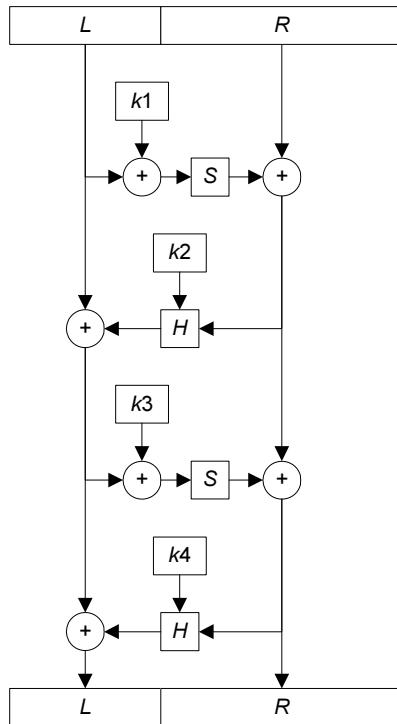


Рис. 3.24. Алгоритм Lioness

## Варианты использования

Специфика всех трех алгоритмов состоит в том, что они адаптированы для шифрования больших блоков данных. В частности, шифрование больших объемов информации может выполняться следующими способами [34]:

- информация, объем которой не превышает 1 Мбайт, может шифроваться как один блок данных;
- информация может разбиваться на блоки не более 1 Мбайт каждый; блоки шифруются на различных ключах (независимых или вычисляемых один из другого);
- каждый из блоков может шифроваться на одном и том же ключе с использованием стандартных режимов работы алгоритмов шифрования (см. разд. 1.4); однако Андерсон и Бихам рекомендуют первые два варианта как более криптографически стойкие.

## Криптоанализ алгоритмов

Помимо упоминавшейся выше атаки на алгоритм Lion, известны следующие атаки на данные алгоритмы (описаны в [271]).

- Алгоритм Bear имеет весьма интересную особенность: если атакующему известна половина ключа (подключ  $k_2$ ), он может восстановить субблок  $R$  открытого текста из шифртекста, выполнив следующие вычисления:

$$R = R' \oplus S(L' \oplus H_{k_2}(R')),$$

где  $L'$  и  $R'$  — левый и правый субблоки шифртекста.

Поскольку субблок  $R$  имеет существенно больший размер, чем  $L$ , получается, что половины ключа шифрования достаточно для восстановления почти всего открытого текста. Это явно отрицательное свойство алгоритма.

- Кроме того, к алгоритмам Lion и Bear может быть применима атака «встреча посередине», с помощью которой многократно упрощается полный перебор ключей. Впрочем, при достаточно больших ключах алгоритмов Bear и Lion полный перебор ключей не кажется практически осуществимым.

Какие-либо другие криптоаналитические работы, связанные с данными алгоритмами, не получили широкую известность, как и сами рассмотренные алгоритмы, в отличие от их авторов, которые за последние 10 лет приняли участие во многих криптографических проектах, в частности, в соавторстве с профессором Ларсом Кнудсеном изобрели алгоритм Serpent (см. разд. 3.48), вышедший в финал конкурса AES.

## 3.7. Алгоритмы BEAST и BEAST-RK

Алгоритмы BEAST (Block Encryption Algorithm with Shortcut in the Third round, алгоритм блочного шифрования с ускоренным третьим раундом) и BEAST-RK разработаны в 1999 г. криптологом Стефаном Лаксом (Stefan Lucks) из Университета г. Геттинген, Германия. Данные алгоритмы продолжают идею описанных в предыдущем разделе алгоритмов Bear и Lion. Все эти алгоритмы блочного шифрования интересны тем, что в них определенным образом комбинируются алгоритмы хэширования и потокового шифрования. В алгоритме BEAST шифрование производится следующим образом (рис. 3.25) [245]:

$$\begin{aligned}L &= L \oplus SHA_{k_1}(R); \\R &= R \oplus SEAL_{k_2}(L); \\L &= L \oplus SHA_{k_3}(R'),\end{aligned}$$

где:

- $L$  и  $R$  — текущие значения левого и правого субблоков соответственно;
- $R'$  — первые  $a$  битов текущего значения  $R$  (значение  $a$  будет пояснено далее);
- $k1...k3$  — фрагменты расширенного ключа шифрования;
- $SHA$  и  $SEAL$  — соответственно, алгоритм хэширования SHA-1 [152] и потоковый шифр SEAL [332], которые применяются в алгоритме BEAST следующим образом:

$$SHA_k(x) = SHA-1(k \oplus x),$$

$$SEAL_k(x) = SEAL(k \oplus x).$$

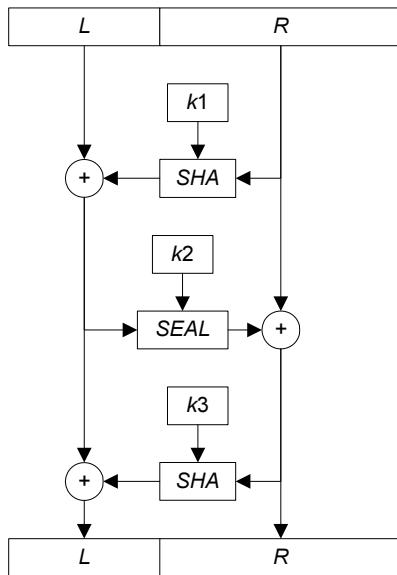


Рис. 3.25. Алгоритм BEAST

Аналогично алгоритму Bear, BEAST делит шифруемый блок размера  $m$  битов на субблоки различного размера:

- $L$ , имеющий размер  $a = 160$  битов (равен размеру выходного значения алгоритма SHA-1);
- $R$  с размером  $b = m - a$  битов.

Причем размер блока  $m$  является переменным, однако хорошее быстродействие алгоритма достигается только при больших размерах блока, т. е. при  $a \ll b$ .

В соответствии с размерами субблоков, подключ  $k1$  имеет размер  $b$  битов, а подключи  $k2$  и  $k3$  — по  $a$  битов.

Как видно из приведенных выше формул, алгоритм BEAST идентичен алгоритму Bear за исключением следующих деталей:

- автор алгоритма BEAST конкретизировал использование SHA-1 и SEAL в алгоритме BEAST, тогда как в Bear используются абстрактные алгоритм хэширования и генератор псевдослучайных последовательностей; впрочем, и в алгоритме BEAST ничто не мешает использовать другие алгоритмы вместо SHA-1 и SEAL [245];
- по-другому накладываются подключи на обрабатываемые субблоки (операцией XOR вместо конкатенации) перед их передачей в качестве параметра в алгоритмы SHA-1 и SEAL;
- второй вызов функции SHA-1 (третий раунд алгоритма) обрабатывает не весь субблок  $R$ , а лишь его 160-битный фрагмент.

Стефан Лакс в спецификации алгоритма [245] пишет, что «BEAST — это вариант Bear, который быстрее, чем Bear и Lion». Существенное увеличение скорости шифрования достигается за счет именно третьего из перечисленных выше отличий — обработка 160-битного фрагмента субблока ничтожна мала по сравнению с обработкой субблока  $R$  целиком. Естественно, при том условии, что шифруемые данные разбиваются на блоки больших размеров — порядка 1 Мбайт.

Аналогично алгоритму Bear, расшифровывание данных выполняется применением обратной последовательности операций:

$$L = L \oplus \text{SHA}_{k3}(R');$$

$$R = R \oplus \text{SEAL}_{k2}(L);$$

$$L = L \oplus \text{SHA}_{k1}(R).$$

## Криптостойкость алгоритма BEAST

Автор BEAST утверждает [245], что ускорение в третьем раунде не ухудшает криптостойкость алгоритма. Однако стоит сказать о том, что BEAST не вызвал такого интереса криptoаналитиков, как Bear и Lion, поэтому какие-либо работы, посвященные его криptoанализу, не имеют широкой известности. Возможно, причиной этого является и тот факт, что BEAST имеет настолько схожую с Bear структуру, поэтому полностью подвержен той же атаке «встреча посередине», что и алгоритмы Bear и Lion. Это подтверждает и сам Стефан Лакс в [245], который категорически рекомендует не использовать BEAST в тех случаях, когда злоумышленник может собрать необходимую ему для атаки информацию (т. е. выбранные открытые тексты и соответствующие им шифртексты, и наоборот).

## Алгоритм BEAST-RK

BEAST-RK (Remote Key, удаленный ключ) представляет собой вариант алгоритма BEAST, оптимизированный для шифрования с помощью смарт-карт с ограниченными ресурсами. Основная идея алгоритма (была высказана известным криптологом Мэттом Блейзом (Matt Blaze) в 1995 г. [94]) состоит в следующем [245]:

- операции, требующие больших ресурсов (но не требующие наличия ключа шифрования), должны выполняться на быстродействующем, но, возможно, недоверенном компьютере;
- операции с участием ключа шифрования должны быть минимизированы, тогда они могут относительно быстро выполняться на смарт-карте, доверенной (по определению), но имеющей весьма ограниченные ресурсы и низкое быстродействие.

Таким образом, ключ шифрования не покидает смарт-карту — достигается эффект полностью безопасного (отсутствует возможность перехвата ключа шифрования) и относительно высокоскоростного шифрования.

В алгоритме BEAST-RK изложенная выше идея реализуется путем разбиения субблока  $R$  на два фрагмента:

- 160-битный  $R1$ ;
- $R2$ , имеющий размер  $b - 160$  битов.

Шифрование производится выполнением следующих действий (рис. 3.26):

$$L = L \oplus \text{SHA}_{k1}(R1 \bullet \text{SHA}-1(R2));$$

$$R1 = R1 \oplus \text{SHA}_{k2}(L);$$

$$R2 = R2 \oplus \text{SEAL}(\text{SHA}_{k3}(L));$$

$$L = L \oplus \text{SHA}_{k4}(R1),$$

где знаком  $\bullet$  обозначена операция конкатенации.

Видно, что основная идея полностью достигнута:

- в каждом из четырех раундов алгоритма смарт-карта выполняет зависящую от ключа операцию (хэширование) над данными незначительного размера — 160-битным субблоком  $L$  или 160-битным фрагментом  $R1$ ;
- в раундах 1 и 3 выполняются операции над большими фрагментами данных: соответственно, бесключевое хэширование фрагмента  $R2$  и выработка псевдослучайного числа размером  $b - 160$  битов для наложения на  $R2$ ; эти длительные (но не требующие ключа шифрования) операции выполняются на быстродействующем компьютере.

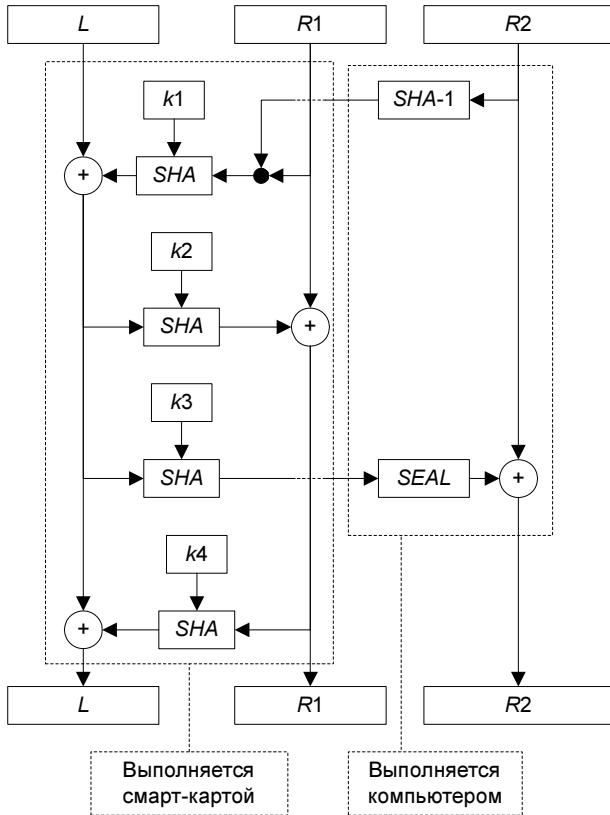


Рис. 3.26. Алгоритм BEAST-RK

### 3.8. Алгоритм Blowfish

Алгоритм Blowfish разработан Брюсом Шнайером в 1994 г. Автор алгоритма предложил Blowfish в качестве замены стандарту DES. Несомненно, в 1994 г. замена DES на новый стандарт шифрования была уже актуальна из-за короткого ключа DES, который уже тогда можно было найти путем полного перебора. Брюс Шнайер предположил, что других реальных кандидатов на замену DES нет, в частности, по следующим причинам (описаны Шнайером в спецификации алгоритма Blowfish [342]):

- многие известные и криптографически стойкие (считающиеся таковыми на момент разработки алгоритма Blowfish) алгоритмы, например, IDEA (см. разд. 3.26) или REDOC-II, являются запатентованными, что ограничивает их использование;

## Описание алгоритмов

- спецификация алгоритма ГОСТ 28147-89 (см. разд. 3.1) не содержит значений таблиц замен, т. е., по мнению Шнайера, алгоритм описан не полностью;
- алгоритм Skipjack (см. разд. 3.52) вообще являлся секретным на тот момент.

Алгоритм Blowfish оказался весьма «удачным». Он очень широко реализован в различных шифровальных средствах — на сайте Шнайера приведен список из примерно 150 продуктов [343], которые шифруют алгоритмом Blowfish. Однако заменой стандарту DES он все же не стал.

## Структура алгоритма

Поскольку Blowfish предполагался в качестве замены алгоритма DES, он, аналогично DES, шифрует данные 64-битными блоками. Размер ключа алгоритма является переменным — от 32 до 448 битов.

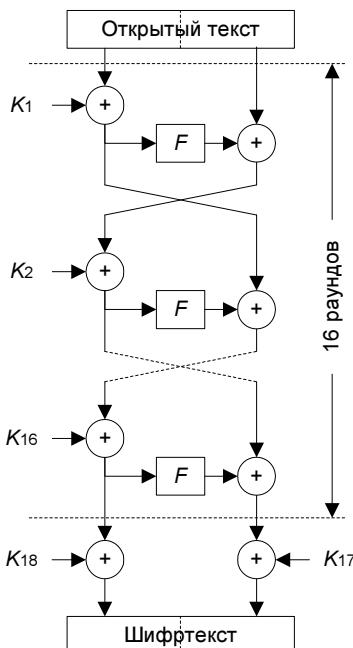


Рис. 3.27. Структура алгоритма Blowfish

Алгоритм представляет собой сеть Фейстеля, его структура приведена на рис. 3.27. Шифрование данных выполняется в 16 раундов, в каждом

из которых над левым 32-битным субблоком данных производятся следующие действия [28, 342]:

1. Значение субблока складывается с ключом  $i$ -го раунда  $K_i$  операцией XOR, результат операции становится новым значением субблока.
2. Субблок обрабатывается функцией  $F$  (описана далее), результат обработки накладывается на правый субблок операцией XOR.
3. Субблоки меняются местами во всех раундах, кроме последнего.

После 16 раундов выполняется наложение на субблоки еще двух подключей:  $K_{17}$  и  $K_{18}$  складываются операцией XOR с правым и левым субблоками соответственно.

Функция  $F$  (рис. 3.28) обрабатывает субблок следующим образом:

1. 32-битное входное значение делится на 4 фрагмента по 8 битов, каждый из которых «прогоняется» через одну из таблиц замен  $S_1 \dots S_4$  с получением четырех 32-битных выходных фрагментов. Таблицы замен содержат по 256 значений по 32 бита, они не являются фиксированными и зависят от ключа шифрования, принципы их вычисления подробно описаны далее.
2. Первые два выходных фрагмента складываются по модулю  $2^{32}$ .
3. Результат предыдущего шага складывается операцией XOR с третьим выходным фрагментом.
4. Выходное значение функции  $F$  получается путем сложения результата предыдущего шага с четвертым выходным фрагментом по модулю  $2^{32}$ .

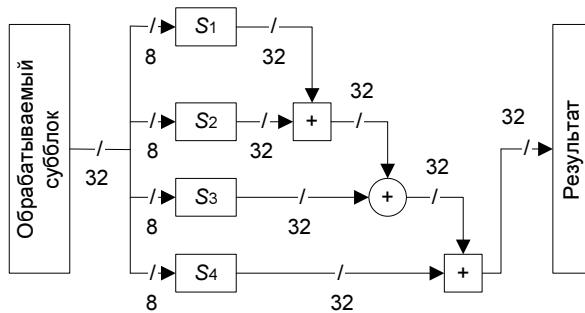


Рис. 3.28. Функция  $F$

То есть функцию  $F$  можно определить так:

$$F(x) = ((S_1(x_1) + S_2(x_2) \bmod 2^{32}) \oplus S_3(x_3)) + S_4(x_4) \bmod 2^{32},$$

где  $x_1 \dots x_4$  — 8-битные фрагменты входного значения  $x$ .

Расшифровывание выполняется аналогично зашифровыванию, но ключи  $K_1...K_{18}$  используются в обратном порядке.

## Процедура расширения ключа

Задача процедуры расширения ключа состоит в вычислении на основе ключа шифрования значений ключей раундов  $K_1...K_{18}$  и таблиц замен  $S_1...S_4$ . Для этого используется весьма сложная процедура расширения ключа, состоящая из следующих шагов:

- Исходные значения ключей раундов и таблиц замен инициализируются фиксированной псевдослучайной строкой, в качестве которой используется шестнадцатеричная запись дробной части числа  $\pi$ ; инициализированная последовательность ключей  $K_1...K_{18}$  приведена в табл. 3.9.

Таблица 3.9

243f6a88	85a308d3	13198a2e	03707344	a4093822	299f31d0
082efa98	ec4e6c89	452821e6	38d01377	be5466cf	34e90c6c
c0ac29b7	c97c50dd	3f84d5b5	b5470917	9216d5d9	8979fb1b

В *Приложении 1* приведены значения инициализированных таблиц замен, которые заполняются последующими знаками дробной части числа  $\pi$  [340].

- Операцией XOR на  $K_1$  накладываются первые 32 бита ключа шифрования, на  $K_2$  — следующие 32 бита и т. д. до  $K_{18}$ . Если используется более короткий ключ шифрования, чем необходимо для наложения на  $K_1...K_{18}$ , то ключ шифрования накладывается циклически.
- С использованием полученных ключей раундов и таблиц замен выполняется шифрование алгоритмом Blowfish блока данных, состоящего из 64 нулевых битов. Результат становится новым значением ключей  $K_1$  и  $K_2$ .
- Результат предыдущего этапа снова шифруется алгоритмом Blowfish (причем уже с измененными значениями ключей  $K_1$  и  $K_2$ ), в результате получаются новые значения ключей  $K_3$  и  $K_4$ .
- Шифрование выполняется до тех пор, пока новыми значениями не будут заполнены все ключи раундов и таблицы замен.

## Достоинства и недостатки алгоритма

В своей книге «Прикладная криптография» [28] Брюс Шнайер привел следующие ограничения алгоритма Blowfish.

- «Алгоритм Blowfish не годится для применения в случаях, где требуется частая смена ключей». Процедура расширения ключа является достаточно ресурсоемкой, поэтому одно из достоинств алгоритма Blowfish — достаточно высокая скорость шифрования — проявляется только в тех случаях, если на одном ключе шифруется достаточно большой объем информации. И наоборот, если менять ключ после каждого из шифруемых блоков, скорость алгоритма становится катастрофически низкой именно из-за необходимости каждый раз выполнять расширение ключа. Сам Шнайер рекомендует в приложениях, где критична скорость, хранить уже развернутый ключ (т. е. значения  $K_1 \dots K_{18}$  и  $S_1 \dots S_4$ ) и загружать его целиком вместо выполнения расширения исходного ключа шифрования [28].
- «Большие требования к памяти не позволяют использовать этот алгоритм в смарт-картах». Стоит сказать, что принципиальная возможность реализации алгоритма в смарт-картах была одним из важных условий при выборе нового стандарта шифрования США на конкурсе AES, т. е. данный недостаток алгоритма Blowfish можно считать серьезным [284].

Кроме того, стоит отметить и менее серьезные недостатки алгоритма:

- невозможность расширения ключа параллельно процессу шифрования;
  - небольшой по современным меркам размер блока шифруемых данных.
- Алгоритм Blowfish имеет и достаточно серьезные преимущества, в частности:
- как было сказано выше, высокая скорость шифрования на развернутом ключе (см. также [344]);
  - простота алгоритма, снижающая вероятность ошибок при его реализации;
  - отсутствие известных успешных атак на полнораундовую версию алгоритма.

Однако стоит сказать, что известный эксперт Серж Воденэ обнаружил, что методом дифференциального криптоанализа  $r$ -раундового алгоритма Blowfish с известными таблицами замен можно вычислить значения  $K_1 \dots K_{18}$  при наличии  $2^{8r+1}$  выбранных открытых текстов [381]. Это не актуально для полнораундовой версии алгоритма (и, тем более, для описанной выше полноценной версии алгоритма с вычисляемыми таблицами замен), но при использовании слабого ключа (Серж Воденэ обнаружил также у алгоритма Blowfish наличие слабых ключей, которые приводят к генерации слабых таблиц замен) выбранных открытых текстов требуется существенно меньше:  $2^{4r+1}$ .

Вероятность, что произвольный ключ окажется слабым, составляет  $2^{-14}$ . Данная атака не является критичной, поскольку для полноценной версии алгоритма слабые ключи не страшны [28, 341].

Явные достоинства и отсутствие критичных недостатков предопределили широкое использование алгоритма Blowfish.

## 3.9. Алгоритм Camellia

Алгоритм Camellia разработан компанией Mitsubishi Electric совместно с компанией Nippon Telegraph and Telephone Corporation (NTT). В его разработке приняли участие некоторые из специалистов, ранее разработавших еще один алгоритм — победитель конкурса NESSIE — MISTY1 (см. разд. 3.36): Мицуру Мацуи, Тецуя Ичикава, Тошио Токита, — а также Казумаро Аоки (Kazumaro Aoki), Масаюки Канда (Masayuki Kanda), Шихо Мориаи (Shiho Moriai) и Джунко Накаджима (Junko Nakajima).

### Структура алгоритма

Camellia шифрует данные 128-битными блоками с использованием 128-, 192- или 256-битных ключей шифрования. В зависимости от размера ключа в алгоритме предусмотрено различное количество раундов  $R$  [38]:

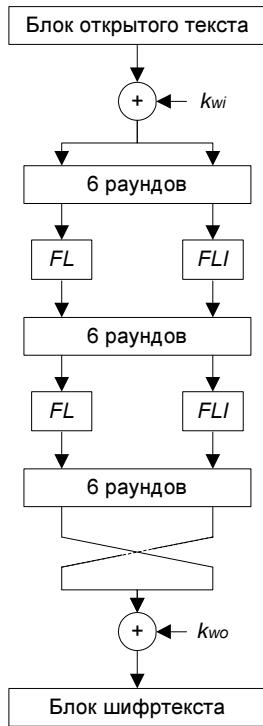
- 18 — для 128-битного ключа;
- 24 — для остальных размеров ключа.

Перед первым раундом выполняется входное отбеливание данных — на шифруемый блок операцией XOR накладывается 128-битный фрагмент расширенного ключа  $k_{wi}$ . Затем 128-битный блок данных делится на два субблока по 64 бита, после чего субблоки «прогоняются» через раунды шифрования. Между каждыми шестью раундами левый субблок обрабатывается операцией  $FL$ , а правый — операцией  $FLI$  (эти операции подробно описаны далее).

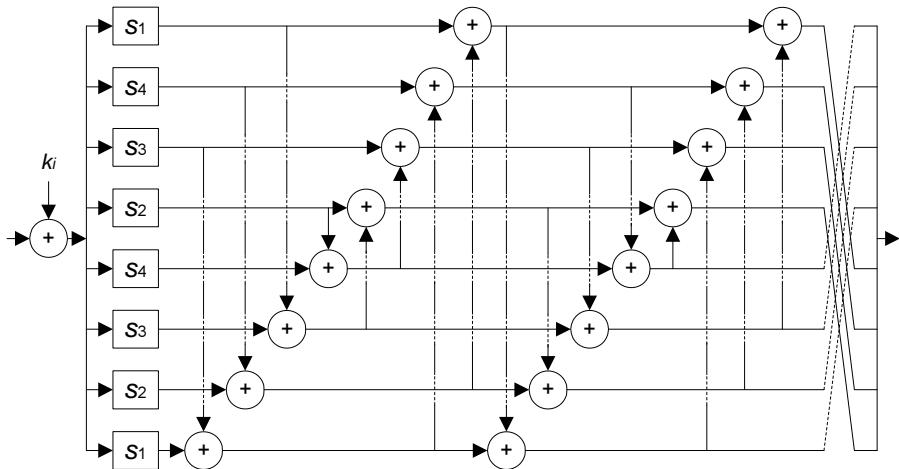
По завершении последнего раунда субблоки меняются местами. Затем выполняется выходное отбеливание данных; здесь используется еще один 128-битный фрагмент расширенного ключа —  $k_{wo}$ .

Структура алгоритма на примере варианта для 128-битного ключа приведена на рис. 3.29.

В каждом раунде левый субблок обрабатывается функцией  $F$ , которая использует 64-битный фрагмент ключа  $k_i$  ( $i$  — номер раунда), и накладывается на правый субблок операцией XOR.



**Рис. 3.29.** Основное преобразование алгоритма Camellia



**Рис. 3.30.** Функция  $F$  алгоритма Camellia

## Описание алгоритмов

Структура функции  $F$  приведена на рис. 3.30. Перечислим ее действия:

1. Прежде всего выполняется наложение на обрабатываемый субблок фрагмента ключа  $k_i$  операцией XOR.
2. Затем 64-битный результат предыдущей операции разбивается на 8 фрагментов по 8 битов, каждый из которых «прогоняется» через табличную замену. Эта операция подробно описана далее.
3. После этого выполняется наложение байтовых фрагментов друг на друга с помощью операции XOR по следующему правилу:

$$y_1 = x_1 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_7 \oplus x_8;$$

$$y_2 = x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_8;$$

$$y_3 = x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \oplus x_8;$$

$$y_4 = x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7;$$

$$y_5 = x_1 \oplus x_2 \oplus x_6 \oplus x_7 \oplus x_8;$$

$$y_6 = x_2 \oplus x_3 \oplus x_5 \oplus x_7 \oplus x_8;$$

$$y_7 = x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_8;$$

$$y_8 = x_1 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7,$$

где  $x_1 \dots x_8$  и  $y_1 \dots y_8$  — соответственно, входные и выходные значения обрабатываемых фрагментов.

4. В завершение выполняется простейшая перестановка фрагментов: байты 1...4 меняются местами с байтами 5...8 (см. рис. 3.30), после чего результат снова объединяется в 64-битный субблок.

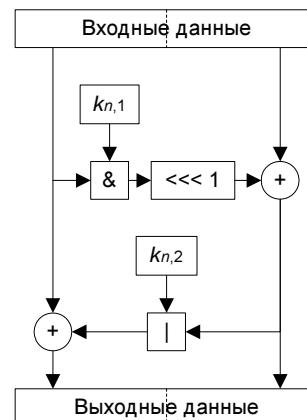


Рис. 3.31. Операция  $FL$  алгоритма Camellia

Выполняемые через каждые 6 раундов операции *FL* и *FLI* не аналогичны таковым в алгоритме MISTY1 — в Camellia используются данные операции от алгоритма KASUMI (см. разд. 3.27) с учетом двукратного увеличения размера обрабатываемого субблока. То есть операция *FL* выполняет следующие действия (рис. 3.31):

$$R' = R \oplus ((L \& k_{n,1}) \lll 1);$$

$$L' = L \oplus (R' | k_{n,2}),$$

где:

- $L$  и  $R$  — левая и правая части входного значения обрабатываемого субблока;
- $L'$  и  $R'$  — левая и правая части выходного значения;
- $k_{n,1}$  и  $k_{n,2}$  — левая и правая части фрагмента ключа для  $n$ -й операции *FL* или *FLI*;
- $\lll$  — операция циклического сдвига влево;
- $|$  и  $\&$  — побитовые логические операции «или» и «и» соответственно.

Операция *FLI* определена так (рис. 3.32):

$$L' = L \oplus (R | k_{n,2});$$

$$R' = R \oplus ((L' \& k_{n,1}) \lll 1).$$

Будем считать, что нечетные фрагменты ключа для операций *FL/FLI* (т. е. нечетные  $n$ ) используются для обработки левого субблока, четные — для обработки правого субблока.

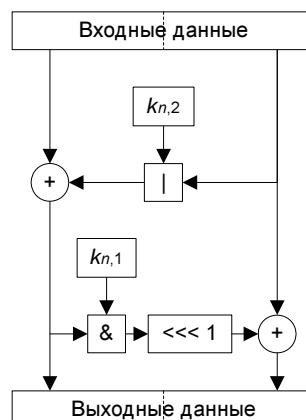


Рис. 3.32. Операция *FLI* алгоритма Camellia

## Таблицы замен

В алгоритме используются 4 таблицы замен:  $s_1 \dots s_4$ . Эти таблицы обрабатывают байтовые фрагменты  $x_1 \dots x_8$  следующим образом:

- таблица  $s_1$  обрабатывает фрагменты  $x_1$  и  $x_8$ ;
- $s_2$  обрабатывает  $x_4$  и  $x_7$ ;
- $s_3$  обрабатывает  $x_3$  и  $x_6$ ;
- $s_4$  обрабатывает  $x_2$  и  $x_5$ .

Эти таблицы приведены в *Приложении 1*.

Как и таблицы замен в алгоритме MISTY1, таблицы  $s_1 \dots s_4$  могут быть (в зависимости от требований к реализации шифратора) реализованы как непосредственно с помощью таблиц, так и с помощью следующих вычислений:

Таблица  $s_1$ :

$$y = h(g(f(C5 \oplus x))) \oplus 6E,$$

где:

- $C5$  и  $6E$  — шестнадцатеричные константы;
- $x$  и  $y$  — соответственно, входное и выходное значения; функции  $f$ ,  $g$  и  $h$  будут описаны далее.

Таблица  $s_2$ :

$$y = s_1(x) <<< 1.$$

Таблица  $s_3$ :

$$y = s_1(x) >>> 1,$$

где  $>>>$  — операция циклического сдвига вправо.

Таблица  $s_4$ :

$$y = s_1(x <<< 1).$$

Функция  $f$  преобразует байт данных так:

$$b_1 = a_6 \oplus a_2;$$

$$b_2 = a_7 \oplus a_1;$$

$$b_3 = a_8 \oplus a_5 \oplus a_3;$$

$$b_4 = a_8 \oplus a_3;$$

$$b_5 = a_7 \oplus a_4;$$

$$b_6 = a_5 \oplus a_2;$$

$$b_7 = a_8 \oplus a_1;$$

$$b_8 = a_6 \oplus a_4,$$

где  $a_1 \dots a_8$  и  $b_1 \dots b_8$  — соответственно, биты входного и выходного значения.

Функция  $g$  определяет биты выходного значения следующим образом:

$$\begin{aligned} & (b_8 + b_7\alpha + b_6\alpha^2 + b_5\alpha^3) + (b_4 + b_3\alpha + b_2\alpha^2 + b_1\alpha^3)\beta = \\ & = 1 / ((a_8 + a_7\alpha + a_6\alpha^2 + a_5\alpha^3) + (a_4 + a_3\alpha + a_2\alpha^2 + a_1\alpha^3)\beta), \end{aligned}$$

где:

- вычисления выполняются в конечном поле  $GF(2^8)$ , обратным значением от 0 является 0,  $\beta$  — элемент конечного поля  $GF(2^8)$ , удовлетворяющий условию:

$$\beta^8 + \beta^6 + \beta^5 + \beta^3 + 1 = 0;$$

- $\alpha = \beta^{238}$ .

Функция  $h$  выполняет следующие преобразования:

$$b_1 = a_5 \oplus a_6 \oplus a_2;$$

$$b_2 = a_6 \oplus a_2;$$

$$b_3 = a_7 \oplus a_4;$$

$$b_4 = a_8 \oplus a_2;$$

$$b_5 = a_7 \oplus a_3;$$

$$b_6 = a_8 \oplus a_1;$$

$$b_7 = a_5 \oplus a_1;$$

$$b_8 = a_6 \oplus a_3.$$

## Расшифровывание

Расшифровывание данных алгоритмом Camellia выполняется полностью аналогично зашифровыванию, но с использованием фрагментов расширенного ключа в обратной последовательности, т. е.:

- подключ  $k_{wo}$  используется при входном отбеливании, и наоборот, подключ  $k_{wi}$  используется при выходном отбеливании данных;
- в раундах шифрования ключи  $k_i$  применяются в обратном порядке: от  $k_r$  до  $k_1$ ;
- в операциях  $FL$  и  $FLI$  четные фрагменты расширенного ключа используются при обработке левого субблока, нечетные — правого; и те, и другие берутся в обратной последовательности относительно зашифровывания.

## Процедура расширения ключа

Задача расширения ключа состоит в формировании необходимого количества фрагментов расширенного ключа (для перечисленных выше операций) из 128-, 192- или 256-битного исходного ключа шифрования  $K$ . Данная процедура состоит из нескольких этапов.

Прежде всего выполняется инициализация переменных  $K_L$  и  $K_R$  таким образом:

- для 128-битного ключа  $K_L = K$ ,  $K_R = 0$ ;
- 192-битный ключ делится на 3 фрагмента по 64 бита, первые два из которых формируют  $K_L$ ; третий фрагмент и его побитовый комплемент формируют  $K_R$ ;
- 256-битный ключ делится на 2 части:  $K_L$  и  $K_R$ .

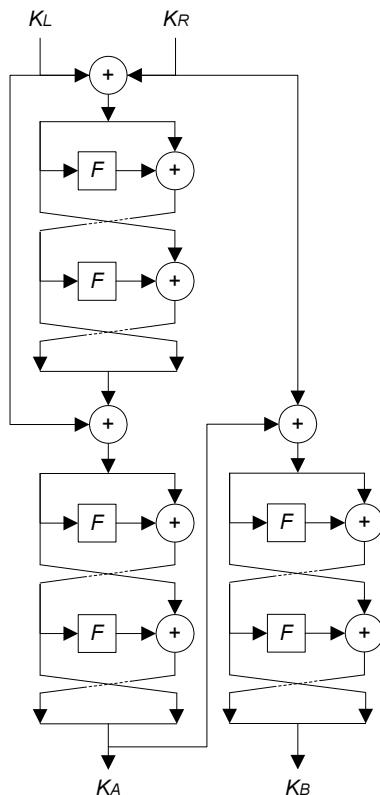


Рис. 3.33. Этап процедуры расширения ключа алгоритма Camellia

На следующем этапе вычисляются еще две ключевые переменные:  $K_A$  и  $K_B$ . Это выполняется с использованием описанной выше функции  $F$  таким образом (рис. 3.33):

1. Результат операции  $K_L \oplus K_R$  дважды обрабатывается функцией  $F$  (см. рис. 3.33), в качестве фрагментов ключа которой берутся константы  $C1$  и  $C2$  (описаны далее).
2. На результат предыдущей операции операцией XOR накладывается  $K_L$ .
3. Снова дважды применяется функция  $F$  с использованием констант  $C3$  и  $C4$  в качестве фрагментов ключа. В результате получается переменная  $K_A$ .
4. В случае, если используются 192- или 256-битные ключи, необходимо также вычислить переменную  $K_B$ . Для этого на  $K_A$  операцией XOR накладывается  $K_R$ , результат дважды обрабатывается функцией  $F$  с использованием констант  $C5$  и  $C6$ .

Константы  $C1...C6$  приведены в табл. 3.10 (указаны шестнадцатеричные значения).

**Таблица 3.10**

C1	A09E667F3BCC908B
C2	B67AE8584CAA73B2
C3	C6EF372FE94F82BE
C4	54FF53A5F1D36F1C
C5	10E527FADE682D1D
C6	B05688C2B3E6C1FD

В процессе шифрования переменные  $K_L$ ,  $K_R$ ,  $K_A$  и  $K_B$  используются в качестве фрагментов расширенного ключа согласно приведенным далее таблицам (в порядке использования фрагментов).

Для 128-битного ключа — см. табл. 3.11, для остальных размеров ключей — табл. 3.12.

**Таблица 3.11**

$k_{wi}$	$K_L$
$k_1$ и $k_2$	Левая и правая половины $K_A$
$k_3$ и $k_4$	Левая и правая половины ( $K_L << 15$ )

Таблица 3.11 (окончание)

$k_{wi}$	$K_L$
$k_5$ и $k_6$	Левая и правая половины ( $K_A <<< 15$ )
$k_{1,x}$ и $k_{2,x}$	Левая и правая половины ( $K_A <<< 30$ )
$k_7$ и $k_8$	Левая и правая половины ( $K_L <<< 45$ )
$k_9$	Левая половина ( $K_A <<< 45$ )
$k_{10}$	Правая половина ( $K_L <<< 60$ )
$k_{11}$ и $k_{12}$	Левая и правая половины ( $K_A <<< 60$ )
$k_{3,x}$ и $k_{4,x}$	Левая и правая половины ( $K_L <<< 77$ )
$k_{13}$ и $k_{14}$	Левая и правая половины ( $K_L <<< 94$ )
$k_{15}$ и $k_{16}$	Левая и правая половины ( $K_A <<< 94$ )
$k_{17}$ и $k_{18}$	Левая и правая половины ( $K_L <<< 111$ )
$k_{wo}$	$K_A <<< 111$

Таблица 3.12

$k_{wi}$	$K_L$
$k_1$ и $k_2$	Левая и правая половины $K_B$
$k_3$ и $k_4$	Левая и правая половины ( $K_R <<< 15$ )
$k_5$ и $k_6$	Левая и правая половины ( $K_A <<< 15$ )
$k_{1,x}$ и $k_{2,x}$	Левая и правая половины ( $K_R <<< 30$ )
$k_7$ и $k_8$	Левая и правая половины ( $K_B <<< 30$ )
$k_9$ и $k_{10}$	Левая и правая половины ( $K_L <<< 45$ )
$k_{11}$ и $k_{12}$	Левая и правая половины ( $K_A <<< 45$ )
$k_{3,x}$ и $k_{4,x}$	Левая и правая половины ( $K_L <<< 60$ )

Таблица 3.12 (окончание)

$k_{wi}$	$K_L$
$k_{13}$ и $k_{14}$	Левая и правая половины ( $K_R <<< 60$ )
$k_{15}$ и $k_{16}$	Левая и правая половины ( $K_B <<< 60$ )
$k_{17}$ и $k_{18}$	Левая и правая половины ( $K_L <<< 77$ )
$k_{5,x}$ и $k_{6,x}$	Левая и правая половины ( $K_A <<< 77$ )
$k_{19}$ и $k_{20}$	Левая и правая половины ( $K_R <<< 94$ )
$k_{21}$ и $k_{22}$	Левая и правая половины ( $K_A <<< 94$ )
$k_{23}$ и $k_{24}$	Левая и правая половины ( $K_L <<< 111$ )
$k_{wo}$	$K_B <<< 111$

## Криптоанализ алгоритма Camellia

Первичный анализ алгоритма Camellia был выполнен его разработчиками. В [37] показана стойкость алгоритма к линейному и дифференциальному криптоанализу, а также к использованию усеченных и невозможных дифференциалов, методу бумеранга, методу интерполяции, сдвиговым атакам и ряду других атак.

Первые отзывы известных экспертов об алгоритме Camellia также были крайне положительными. В частности, в работах [208] и [404] отмечена исключительно высокая криптостойкость Camellia. Ларс Кнудсен в [208] утверждает следующее:

- любые практически осуществимые атаки на Camellia будут возможны только после принципиальных прорывов в области криптоанализа;
- Camellia — один из наиболее стойких современных алгоритмов шифрования.

В оценке ресурсоемкости и быстродействия алгоритма Camellia эксперты были далеко не так единодушны:

- в [88] отмечается, что Camellia существенно проигрывает в скорости алгоритму Rijndael;
- в [236] показано, что Camellia предъявляет достаточно высокие требования к оперативной и энергонезависимой памяти;

## Описание алгоритмов

□ и наоборот, авторы отчета CRYPTREC (Camellia — один из рекомендованных экспертами алгоритмов для защиты в рамках проекта CRYPTREC, который посвящен созданию электронного правительства Японии) отмечали высокую скорость шифрования (особенно на серверных платформах), быструю процедуру расширения ключа и достаточно невысокие требования к энергонезависимой памяти [127].

Весьма много исследований было посвящено ослабленному варианту алгоритма Camellia — с уменьшенным количеством раундов, а также без входного/выходного отбеливания и операций *FL/FLI*. Стоит упомянуть следующие работы:

- в [65], в частности, описана атака на 9-раундовый вариант Camellia, который вскрывается при наличии  $2^{105}$  выбранных открытых текстов выполнением  $2^{64}$  операций шифрования; там же отмечается, что линейный криптоанализ неприменим для вскрытия алгоритма;
- авторами [368] найден 9-раундовый усеченный дифференциал, с помощью которого возможна атака на 11-раундовую версию Camellia;
- в работе [397] предложена атака методом поиска коллизий на 9-раундовую версию Camellia, для которой требуется  $2^{113,6}$  выбранных открытых текстов и  $2^{121}$  операций шифрования (128-битный ключ) или  $2^{13}$  выбранных открытых текстов и  $2^{175,6}$  операций (192-битный ключ); там же предложена атака на 10-раундовую версию с 256-битным ключом, для которой необходимо  $2^{14}$  выбранных открытых текстов и  $2^{239,9}$  операций;
- в совсем недавней работе [399] применен метод невозможных дифференциалов, найден 8-раундовый невозможный дифференциал, с помощью которого атакована 12-раундовая версия; для атаки требуется  $2^{120}$  выбранных открытых текстов и  $2^{181}$  операций.

Работы, анализирующие варианты, более близкие к исходной версии Camellia, встречаются существенно реже. Этим примечательна, например, работа [403], в которой применена Square-атака, вскрывающая 9-раундовый алгоритм Camellia (включая операции *FL/FLI* после шестого раунда, но без входного и выходного отбеливания) при наличии  $2^{60,5}$  выбранных открытых текстов выполнением  $2^{202,2}$  операций шифрования.

Алгоритм Camellia был исследован и с точки зрения атак, использующих утечку информации по побочным каналам. В этой связи широко известны, в частности, следующие работы:

- авторы работы [294] утверждают, что Camellia — один из лучших алгоритмов, представленных во втором раунде конкурса NESSIE, с точки зрения защищенности от атак по потребляемой мощности;

- в работе [400] предложена атака по потребляемой мощности, которая использует особенности процедуры расширения ключа алгоритма Camellia.

Camellia — один из алгоритмов — победителей конкурса NESSIE [305] (см. разд. 2.2). В рамках исследований, проведенных во время конкурса NESSIE, не было выявлено каких-либо проблем с криптостойкостью данного алгоритма [307]. В настоящее время также не найдено каких-либо серьезных уязвимостей в алгоритме Camellia, однако видно, что активный анализ этого алгоритма будет продолжаться.

### 3.10. Алгоритм CAST-128

Алгоритм CAST-128 [30] разработан канадской компанией Entrust Technologies. Этот алгоритм является предшественником хорошо известного алгоритма CAST-256 [31] (см. разд. 3.11).

CAST-128 шифрует информацию блоками по 64 бита, используя несколько фиксированных размеров ключа: от 40 до 128 битов включительно с шагом 8 битов. При этом автор алгоритма считает достаточной в реализациях алгоритма поддержку 40-, 64-, 80- и 128-битных ключей.

Данный алгоритм является сетью Фейстеля, в которой выполняется 12 или 16 раундов преобразований в зависимости от размера ключа (рис. 3.34):

- 12 раундов при ключах размером до 80 битов включительно;
- 16 раундов, если размер ключа превышает 80 битов.

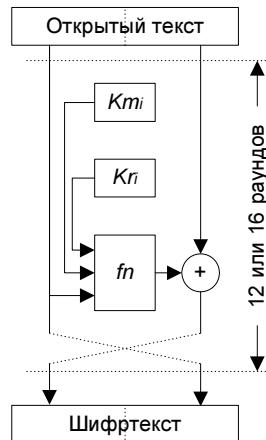


Рис. 3.34. Структура алгоритма CAST-128

В качестве функции раундов поочередно используются те же три функции  $f_1$ ,  $f_2$  и  $\beta$ , которые впоследствии были применены в алгоритме CAST-256 (см. рис. 3.37, 3.38 и 3.39).

Процедура расширения ключа предполагает формирование 32 32-битных подключей, 16 из которых становятся маскирующими подключами  $Km_i$  (см. описание алгоритма CAST-256 в разд. 3.11), а остальные 16 — подключами сдвига  $Kr_i$  (в которых используются только по 5 младших битов).

Перед выполнением расширения ключа выполняется дополнение ключа, меньшего 128 битов, нулевыми битами до достижения 128-битного размера. Собственно процедура расширения ключа является достаточно сложной, ее полное описание можно найти в спецификации алгоритма [30]. Стоит отметить тот факт, что, в отличие от алгоритма CAST-256, CAST-128 имеет 8 (а не 4) таблиц замен, причем 4 из них используются только в процедуре расширения ключа. Это сильно увеличивает ресурсоемкость алгоритма в части требований к энергонезависимой памяти. Таблицы замен алгоритма приведены в *Приложении 1*.

Расшифровывание выполняется аналогично зашифровыванию, но с обратным порядком использования ключей раундов.

Не известно каких-либо методов вскрытия алгоритма CAST-128, более быстрых, чем прямой перебор вариантов ключа. В [347] упоминается успешная атака, использующая дифференциальный криптоанализ, однако данной атаке подвержена модифицированная версия алгоритма.

## 3.11. Алгоритм CAST-256

Алгоритм шифрования CAST-256 [31] разработан специалистами канадской компании Entrust Technologies. Основой алгоритма являются преобразования широко используемого ранее и хорошо зарекомендовавшего себя алгоритма CAST-128 (см. разд. 3.10), также разработанного компанией Entrust Technologies.

### Основные характеристики и структура алгоритма

Алгоритм CAST-256 шифрует информацию 128-битными блоками и использует несколько фиксированных размеров ключа шифрования: 128, 160, 192, 224 или 256 битов.

128-битный блок данных разбивается на 4 субблока по 32 бита, каждый из которых в каждом раунде алгоритма подвергается определенному преобразованию и накладывается на один из соседних субблоков. Разработчики алгоритма классифицировали его как подстановочно-перестановочную сеть (SP-сеть, см. разд. 1.3). Однако ряд экспертов конкурса AES посчитали алгоритм CAST-256 сетью Фейстеля, в каждом раунде которой обрабатывается только один субблок, а количество «настоящих» раундов в 4 раза больше, чем заявлено в спецификации алгоритма [384].

В процессе работы алгоритма выполняется 12 раундов преобразований, в первых 6 из которых выполняется преобразование  $t1$  (называемое *прямой функцией раунда*), а в последних 6 раундах выполняется *обратная функция раунда*  $t2$ .

Преобразования  $t1$  и  $t2$  показаны на рис. 3.35 и 3.36 соответственно. Функция  $t1$  описана следующим образом:

$$\begin{aligned} C &= C \oplus f1(D, Kr_{0i}, Km_{0i}); \\ B &= B \oplus f2(C, Kr_{1i}, Km_{1i}); \\ A &= A \oplus f3(B, Kr_{2i}, Km_{2i}); \\ D &= D \oplus f1(A, Kr_{3i}, Km_{3i}), \end{aligned}$$

где  $i$  — номер текущего раунда.

Преобразование  $t2$  состоит из следующих операций:

$$\begin{aligned} D &= D \oplus f1(A, Kr_{3i}, Km_{3i}); \\ A &= A \oplus f3(B, Kr_{2i}, Km_{2i}); \\ B &= B \oplus f2(C, Kr_{1i}, Km_{1i}); \\ C &= C \oplus f1(D, Kr_{0i}, Km_{0i}). \end{aligned}$$

Функции  $f1$ ,  $f2$  и  $f3$  выполняют несколько элементарных операций над 32-битным субблоком; они приведены, соответственно, на рис. 3.37, 3.38 и 3.39. Каждая из функций принимает три параметра:

- значение обрабатываемого субблока (на рисунках обозначено как «*даные*»);
- 32-битный подключ раунда  $Km_{ni}$  (называемый *маскирующим подключом*, поскольку первой операцией каждой из функций является наложение данного ключа на обрабатываемый субблок);
- 5-битный подключ раунда  $Kr_{ni}$  (называемый *подключом сдвига*, поскольку данный ключ используется в операции циклического сдвига результата предыдущей операции на переменное число битов).

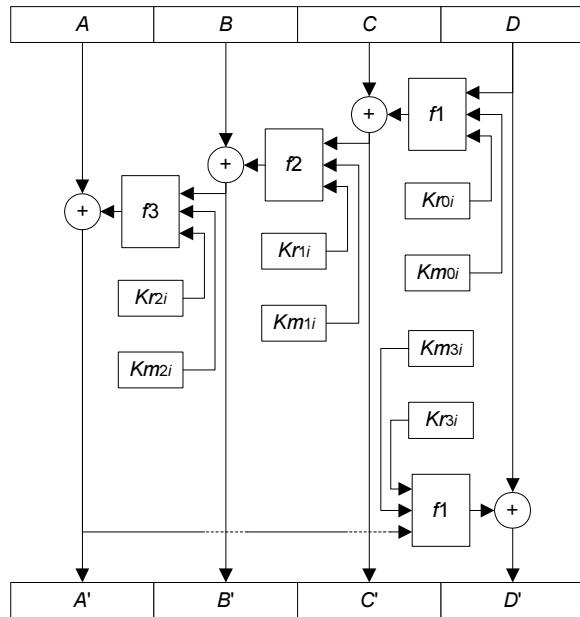
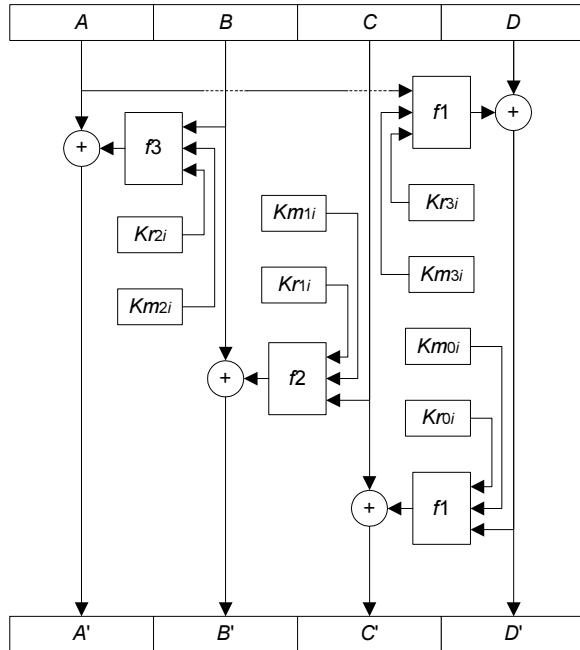
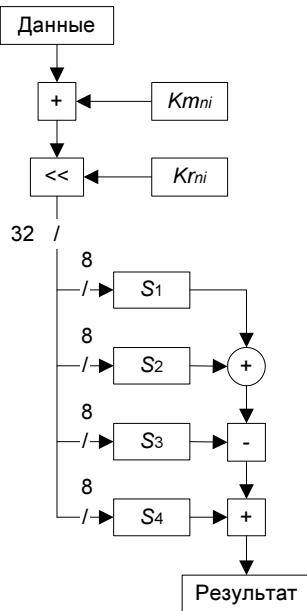
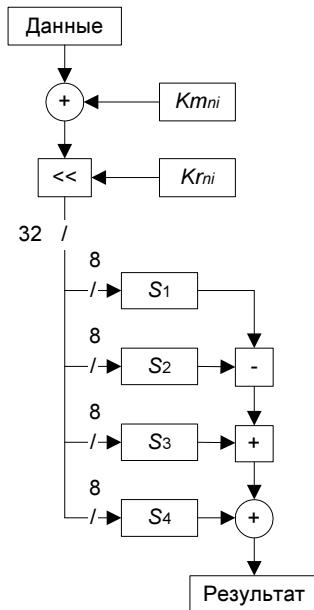
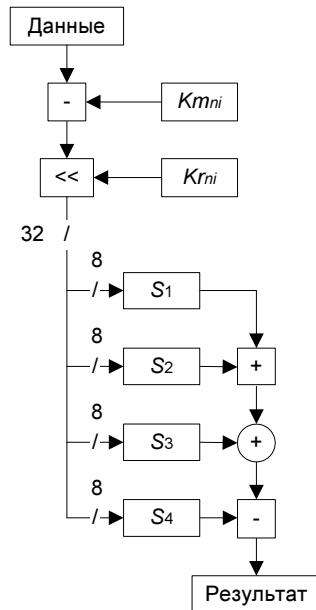


Рис. 3.35. Преобразование  $t1$



**Рис. 3.36.** Преобразование  $t2$

Рис. 3.37. Функция  $f_1$ Рис. 3.38. Функция  $f_2$ Рис. 3.39. Функция  $f_3$

Сложение и вычитание выполняются с 32-битными операндами по модулю  $2^{32}$ .

Функции  $S_1$ ,  $S_2$ ,  $S_3$  и  $S_4$  являются табличными подстановками, выполняющими замену входного 8-битного значения на 32-битное. Таблицы являются различными, каждая из них содержит 256 32-битных фиксированных значений (таблицы полностью приведены в *Приложении 1*).

Расшифровывание выполняется аналогично зашифровыванию, но подключи раундов используются в обратной последовательности, т. е. вместо подключа  $Kx_{ni}$  используется подключ  $Kx_{n(11-i)}$  (считая, что раунды нумеруются от 0 до 11).

## Процедура расширения ключа

Задача функции расширения ключа — выработать для каждого раунда 4 32-битных маскирующих подключа  $Km_{ni}$  и 4 5-битных подключа сдвига  $Kr_{ni}$ . То есть сформировать 1776 байтов ключевой информации.

Процедура расширения ключа является более сложной, чем собственно шифрование данных. Сначала 256-битный ключ шифрования делится на 8 фрагментов по 32 бита. Если используется ключ меньшего размера, то «лишние» фрагменты ключа считаются нулевыми.

После этого выполняется инициализация временных переменных, которая описана далее.

Затем выполняется 12 раундов преобразований, в каждом из которых производятся следующие действия:

1. Выполняется функция  $W_{2j}$ , где  $j$  — номер раунда процедуры расширения ключа (функция  $W_j$  приведена на рис. 3.40).
2. Выполняется функция  $W_{2j+1}$ .
3. По 5 младших битов четырех фрагментов результата операции ( $A'$ ,  $C'$ ,  $E'$ ,  $G'$ ) становятся подключами  $Kr_{0j}$ ,  $Kr_{1j}$ ,  $Kr_{2j}$  и  $Kr_{3j}$  соответственно.
4. Оставшиеся фрагменты ( $B'$ ,  $D'$ ,  $F'$ ,  $H'$ ) становятся подключами  $Km_{3j}$ ,  $Km_{2j}$ ,  $Km_{1j}$  и  $Km_{0j}$  соответственно.

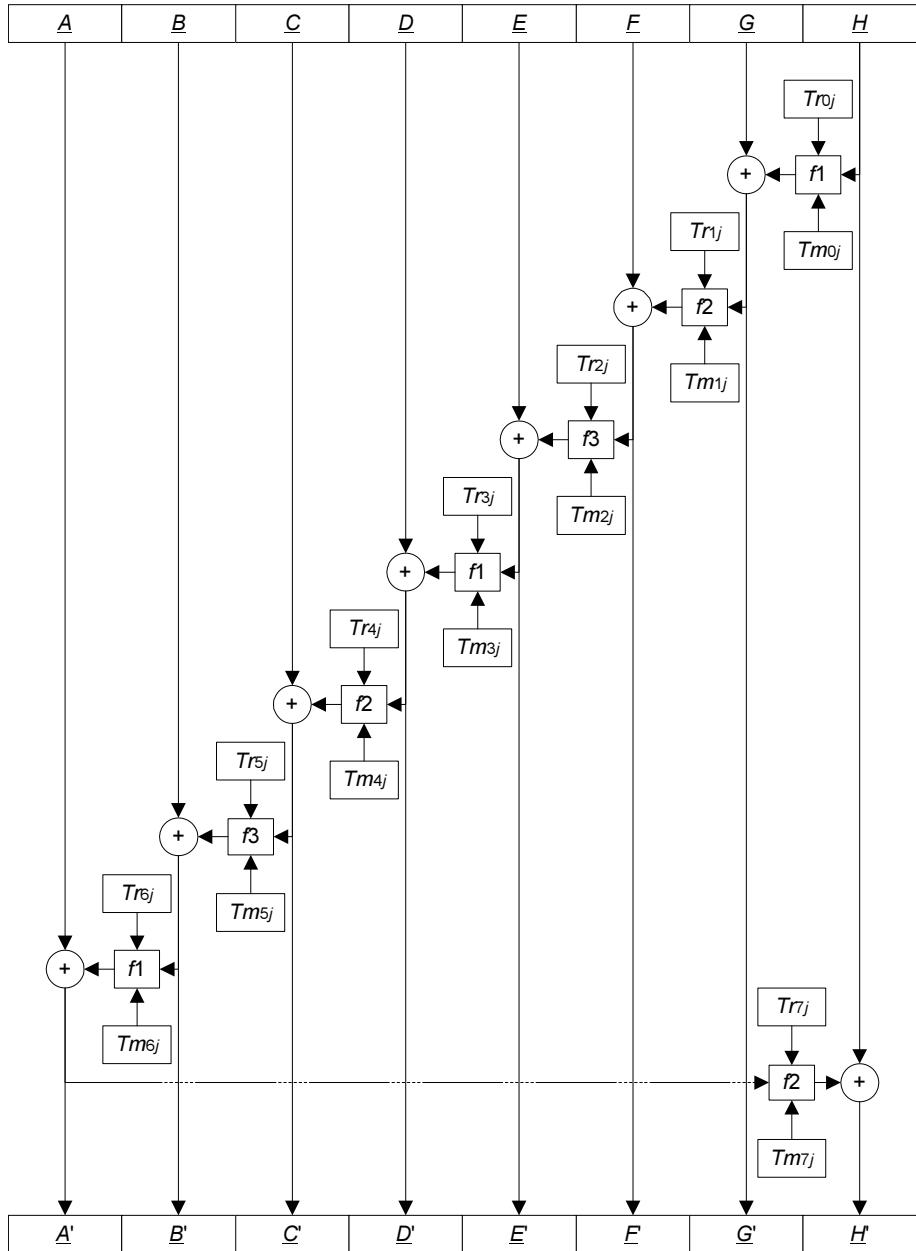
Как видно, в каждом раунде функции расширения ключа используются по 8 дополнительных переменных  $Tm_{nk}$  и  $Tr_{nk}$ , которые вычисляются следующим образом:

1. Инициализируются временные переменные (для  $Cm$  и  $Mm$  указаны шестнадцатеричные значения):

$$Cm = 2^{30} * \sqrt{2} = 5A827999;$$

$$Mm = 2^{30} * \sqrt{3} = 6ED9EBA1;$$

$$Cr = 19, \quad Mr = 17.$$

Рис. 3.40. Функция  $W_j$

2. В цикле от 0 до 23 выполняется цикл от 0 до 8, в котором производятся следующие вычисления:

$$\begin{aligned}Tm_{nk} &= Cm; \\Cm &= Cm + Mm \bmod 2^{32}; \\Tr_{nk} &= Cr; \\Cr &= Cr + Mr \bmod 2^{32},\end{aligned}$$

где:

- $n$  — счетчик внутреннего цикла;
- $k$  — счетчик внешнего цикла.

## Достоинства и недостатки алгоритма

Несомненным достоинством алгоритма CAST-256 является отсутствие доказанных уязвимостей. Кроме того, плюсом является возможность выполнения расширения ключа «на лету», т. е. в процессе операции зашифровывания (но не расшифровывания) [284]. Однако в алгоритме было найдено несколько недостатков, благодаря которым он не вышел во второй раунд конкурса [284]:

- алгоритм уступает в скорости ряду алгоритмов — участников конкурса, в том числе всем финалистам конкурса;
- достаточно высокие требования к оперативной и энергонезависимой памяти затрудняют использование данного алгоритма в смарт-картах;
- некоторые эксперты сочли алгоритм CAST-256 подверженным атакам по потребляемой мощности [80, 114].

## 3.12. Алгоритм Crypton

Алгоритм Crypton разработан в 1998 г. Че Хун Лимом (Chae Hoon Lim) из южнокорейской компании Future Systems. Изначально на конкурс AES была представлена версия алгоритма Crypton v0.5 — по словам разработчика алгоритма, ему не хватило времени на подготовку полноценной версии. Уже в процессе анализа алгоритмов в рамках первого этапа конкурса AES Crypton v0.5 был заменен на Crypton v1.0. Версия 1.0 отличалась от предыдущей измененными таблицами замен и модифицированной процедурой расширения ключа, именно она и описана в данной книге.

## Основные характеристики и структура алгоритма

Как и другие участники конкурса AES, Crypton шифрует 128-битные блоки данных. Используются ключи шифрования нескольких фиксированных размеров — от 0 до 256 битов с кратностью 8 битов.

Структура алгоритма Crypton во многом повторяет структуру алгоритма Square (см. разд. 3.54), разработанного в 1997 г. будущими авторами алгоритма Rijndael (см. разд. 3.3) — победителя конкурса AES — Винсентом Риджменом и Джоан Деймен.

Алгоритм Crypton представляет 128-битный блок шифруемых данных в виде байтового массива размером  $4 \times 4$ , над которым в процессе шифрования производится несколько раундов преобразований. В каждом раунде предполагается последовательное выполнение нижеперечисленных операций [239, 240]:

1. Табличная замена  $\gamma$  (рис. 3.41). Алгоритм Crypton использует 4 таблицы замен, каждая из которых замещает 8-битное входное значение на выходное такого же размера. Все таблицы  $S_0 \dots S_3$  являются производными от основной таблицы  $S$  (см. рис. 3.42, на котором  $<<n$  обозначает циклический сдвиг влево значения обрабатываемого байта на  $n$  битов) и приведены в *Приложении 1*.

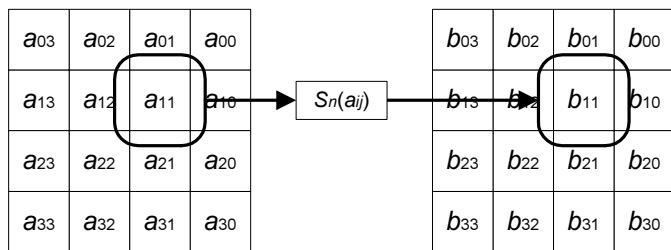


Рис. 3.41. Операция  $\gamma$

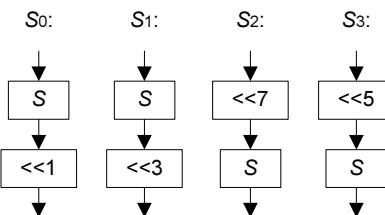


Рис. 3.42. Вычисление производных таблиц замен

В четных раундах используется операция  $\gamma_e$ , в нечетных —  $\gamma_o$ ; эти операции и таблицы замен обладают следующими свойствами (которые важны, в частности, для унификации операций зашифровывания и расшифровывания):

$$\gamma_e(\gamma_o(A)) = \gamma_o(\gamma_e(A)) = A4;$$

## Описание алгоритмов

$$S = S^{-1};$$

$$S_2 = {S_0}^{-1}4;$$

$$S_3 = {S_1}^{-1},$$

где  $A$  — текущее значение блока шифруемых данных.

Операции  $\gamma_e$  и  $\gamma_o$  можно определить следующим образом:

$$\gamma_e : b_{ij} = S_{(j+2+i \bmod 4)}(a_{ij});$$

$$\gamma_o : b_{ij} = S_{(j+i \bmod 4)}(a_{ij}),$$

где:

- $a_{ij}$  — текущее значение конкретного байта данных;
- $b_{ij}$  — значение байта данных после выполнения операции.

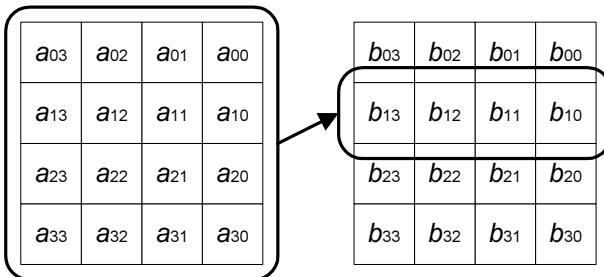


Рис. 3.43. Операция  $\pi$

2. Линейное преобразование  $\pi$  (рис. 3.43). Здесь используются 4 специальные константы (указаны шестнадцатеричные значения):

$$m_0 = \text{FC};$$

$$m_1 = \text{F3};$$

$$m_2 = \text{CF};$$

$$m_3 = \text{3F},$$

которые объединены в следующие маскирующие последовательности:

$$M_0 = m_3 \bullet m_2 \bullet m_1 \bullet m_0;$$

$$M_1 = m_0 \bullet m_3 \bullet m_2 \bullet m_1;$$

$$M_2 = m_1 \bullet m_0 \bullet m_3 \bullet m_2;$$

$$M_3 = m_2 \bullet m_1 \bullet m_0 \bullet m_3,$$

где  $\bullet$  — операция конкатенации.

Сама же операция  $\pi$  представляет собой битовую перестановку. В нечетных раундах используется операция  $\pi_o$ :

$$B[0] = (A[3] \& M_3) \oplus (A[2] \& M_2) \oplus (A[1] \& M_1) \oplus (A[0] \& M_0);$$

$$B[1] = (A[3] \& M_0) \oplus (A[2] \& M_3) \oplus (A[1] \& M_2) \oplus (A[0] \& M_1);$$

$$B[2] = (A[3] \& M_1) \oplus (A[2] \& M_0) \oplus (A[1] \& M_3) \oplus (A[0] \& M_2);$$

$$B[3] = (A[3] \& M_2) \oplus (A[2] \& M_1) \oplus (A[1] \& M_0) \oplus (A[0] \& M_3),$$

где:

- $\&$  — логическая побитовая операция «и»;
- $A[i]$  и  $B[i]$  — значение  $i$ -й строки обрабатываемых данных до и после выполнения операции соответственно.

В четных раундах используется операция  $\pi_e$ :

$$B[0] = (A[3] \& M_1) \oplus (A[2] \& M_0) \oplus (A[1] \& M_3) \oplus (A[0] \& M_2);$$

$$B[1] = (A[3] \& M_2) \oplus (A[2] \& M_1) \oplus (A[1] \& M_0) \oplus (A[0] \& M_3);$$

$$B[2] = (A[3] \& M_3) \oplus (A[2] \& M_2) \oplus (A[1] \& M_1) \oplus (A[0] \& M_0);$$

$$B[3] = (A[3] \& M_0) \oplus (A[2] \& M_3) \oplus (A[1] \& M_2) \oplus (A[0] \& M_1).$$

Фактически эта операция обеспечивает наличие в каждом результирующем байте каждого столбца двух битов каждого исходного байта того же столбца.

3. Байтовая перестановка  $\tau$ , простейшим образом преобразующая строку данных в столбец (рис. 3.44):

$$\tau : b_{ij} = a_{ji}.$$

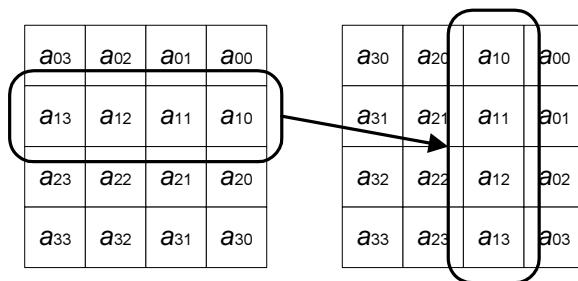


Рис. 3.44. Операция  $\tau$

4. Операция  $\sigma$ , представляющая собой побитовое сложение всего массива данных с ключом раунда (рис. 3.45):

$$\sigma : B = A \oplus K_r,$$

где:

$B$  — новое значение блока шифруемых данных;

$K_r$  — ключ текущего раунда  $r$  (процедура формирования ключей раундов описана далее).

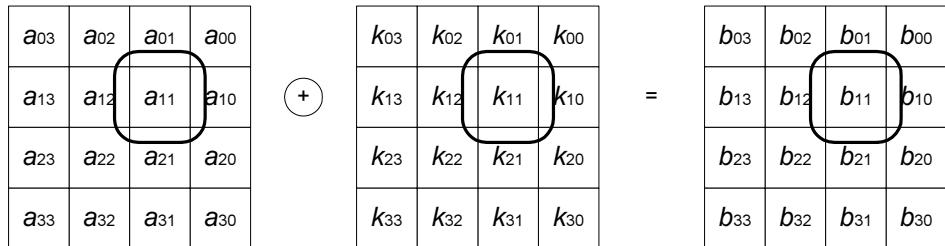


Рис. 3.45. Операция  $\sigma$

Помимо собственно 12 раундов шифрования (количество раундов строго не установлено, но автором алгоритма рекомендуется именно 12 раундов), перед первым раундом алгоритма выполняется предварительная операция  $\sigma$ , а по завершении 12 раундов выполняется выходное преобразование  $\phi_e$ , состоящее из последовательно выполняемых операций  $\tau$ ,  $\pi_e$  и  $\tau$ .

Расшифровывание выполняется применением обратных операций в обратной последовательности. Все операции, кроме  $\gamma_e$  и  $\gamma_o$ , являются обратными самим себе, а  $\gamma_e$  и  $\gamma_o$  связаны следующими соотношениями:

$$\begin{aligned}\gamma_e^{-1} &= \gamma_o; \\ \gamma_o^{-1} &= \gamma_e,\end{aligned}$$

поэтому при расшифровывании  $\gamma_o$  используется в четных раундах, а  $\gamma_e$  — в нечетных.

Стоит сказать, что в спецификации алгоритма [240] дано подробное математическое обоснование всех используемых в алгоритме операций. С особой подробностью рассмотрены принципы построения исходной таблицы замен  $S$  и ее свойства.

## Процедура расширения ключа

Алгоритм Crypton требует наличия 128-битного ключа для каждого раунда, а также 128-битного ключа для предварительной операции  $\sigma$ . Процедура выполняется в два этапа: на первом из ключа шифрования формируется 8 расширенных ключей, а на втором — из расширенных ключей вычисляются ключи раундов.

Генерация расширенных ключей производится следующим образом:

- Если ключ шифрования имеет размер, меньший 256 битов, он предваряется битовыми нулями до достижения 32-байтного исходного ключа  $K$ :

$$K = k_{31} \bullet \dots \bullet k_1 \bullet k_0.$$

- Ключ  $K$  разбивается на последовательности  $U$  и  $V$ , первая из которых содержит только четные байты ключа, вторая — только нечетные:

$$U = U_3 \bullet U_2 \bullet U_1 \bullet U_0;$$

$$V = V_3 \bullet V_2 \bullet V_1 \bullet V_0;$$

$$U_i = k_{8i+6} \bullet k_{8i+4} \bullet k_{8i+2} \bullet k_{8i};$$

$$V_i = k_{8i+7} \bullet k_{8i+5} \bullet k_{8i+3} \bullet k_{8i+1}.$$

- Над  $U$  и  $V$  выполняется один раунд шифрования (для  $U$  выполняются преобразования нечетного раунда, для  $V$  — четного) алгоритма Crypton с использованием ключа раунда, состоящего из нулевых битов; результирующие последовательности обозначаются как  $U'$  и  $V'$ .

- 8 расширенных ключей вычисляются таким образом:

$$Ke_i = U'_i \oplus T_1;$$

$$Ke_{i+4} = V'_i \oplus T_0$$

для  $i = 0, 1, 2, 3$ , где  $T_1$  и  $T_0$  — последовательности, формируемые согласно формулам:

$$T_0 = U'_0 \oplus U'_1 \oplus U'_2 \oplus U'_3;$$

$$T_1 = V'_0 \oplus V'_1 \oplus V'_2 \oplus V'_3.$$

При вычислении ключей раундов из расширенных ключей используется следующий набор констант:

$$C_0 = A54FF53A;$$

$$C_i = C_{i-1} + 3C6EF372 \bmod 2^{32};$$

$$Mc_0 = ACACACAC,$$

причем псевдослучайные константы A54FF53A и 3C6EF372 образованы значениями дробных частей от  $\sqrt{7}$  и  $\sqrt{5}$  соответственно.

Сначала вычисляются ключи для предварительной операции  $\sigma$  и первого раунда:

$$K_0[i] = Ke_i \oplus C_0 \oplus Mc_i;$$

$$K_1[i] = Ke_{i+4} \oplus C_1 \oplus Mc_i,$$

где  $K_r[i]$  —  $i$ -я строка ключа раунда  $r$  (аналогично шифруемым данным, ключ представляется в виде таблицы — см. рис. 3.45).

Константы  $Mc_i$  вычисляются путем побитового циклического сдвига каждого байта (раздельно) константы  $Mc_{i-1}$  на 1 бит влево.

Для второго и каждого последующего четного раунда ключ вычисляется таким образом:

1. Выполняется модификация первых четырех расширенных ключей:

$$\{Ke_3, Ke_2, Ke_1, Ke_0\} \leftarrow \{Ke_0 \lll 6, Ke_3 \lll 6, Ke_2 \ll 16, Ke_1 \ll 24\},$$

где знаком  $\lll$  обозначена операция побитового циклического сдвига каждого байта (раздельно) расширенного ключа на указанное число битов влево, а  $\ll$  — операция побитового циклического сдвига всего ключа на указанное число битов влево.

2. Вычисление ключа раунда с помощью модифицированных расширенных ключей:

$$K_r[i] = Ke_i \oplus C_r \oplus Mc_i.$$

Аналогичным образом вычисляются ключи для нечетных раундов:

$$\{Ke_7, Ke_6, Ke_5, Ke_4\} \leftarrow \{Ke_6 \ll 16, Ke_5 \ll 8, Ke_4 \lll 2, Ke_7 \lll 2\},$$

$$K_r[i] = Ke_{i+4} \oplus C_r \oplus Mc_i.$$

Как видно, процедура расширения ключа позволяет генерировать ключи раундов «на лету», т. е. в процессе шифрования по мере необходимости. Это является достоинством алгоритма хотя бы потому, что не требуется память для хранения ключей раундов. Спецификация алгоритма содержит также описание процедуры расширения ключа «на лету» для расшифровывания. Однако для расшифровывания возможно и выполнение прямой процедуры расширения ключа и использование полученных ключей раундов в обратном порядке.

## Достоинства и недостатки алгоритма

При анализе исходной версии алгоритма — Crypton v0.5 — сразу двое экспертов [103, 284] независимо обнаружили класс слабых ключей: таковых оказалось  $2^{32}$  256-битных ключей. Кроме того, была обнаружена атака, аналогичная известной атаке на алгоритм Square, на 6-раундовую версию алгоритма Crypton [170]. Возможно, именно эти результаты привели к появлению следующей версии алгоритма.

По правилам конкурса AES разрешалось модифицировать алгоритмы в процессе конкурса, чем воспользовались некоторые разработчики [284]. Однако изменения не должны были быть значительными, чтобы к модифицированной версии можно было применить результаты анализа исходной версии алгоритма. В отношении алгоритма Crypton эксперты посчитали изменения

значительными, поскольку они затрагивали как таблицы замен, так и процедуру расширения ключа. Поэтому версия 1.0 не была принята в качестве замены версии 0.5.

На самом деле упомянутые здесь недостатки алгоритма Crypton v0.5 не были существенными, как признали эксперты института NIST (Институт стандартов и технологий США — организатор конкурса AES). При этом список достоинств алгоритма оказался весьма внушительным [284]:

- достаточно высокая скорость на всех целевых платформах;
- небольшие требования к оперативной памяти и возможность расширения ключа «на лету» позволяют использовать алгоритм в смарт-картах с минимальными ресурсами;
- алгоритм не подвержен атакам по времени выполнения и потребляемой мощности;
- быстрое расширение ключа;
- возможность распараллеливания операций в процессе шифрования;
- алгоритм поддерживает дополнительные размеры ключей ( помимо установленных конкурсом обязательных размеров: 128, 192 и 256 битов).

Признавая отсутствие у алгоритма Crypton существенных недостатков и наличие явных достоинств, эксперты NIST все же не пропустили Crypton во второй раунд конкурса. Причиной тому было наличие двух алгоритмов со схожими характеристиками: Rijndael (будущий победитель конкурса) и Twofish, каждый из которых быстрее алгоритма Crypton на целевых платформах. Кроме того, проблем, аналогичных слабым ключам алгоритма Crypton, у Rijndael и Twofish не оказалось. Поэтому эксперты логично предположили, что при подведении итогов конкурса AES Crypton проиграет одному из этих алгоритмов, и исключили его из рассмотрения во втором раунде конкурса.

### 3.13. CS-Cipher

Авторы алгоритма шифрования CS-Cipher (или просто CS) — Жак Стерн (Jacques Stern) и Серж Воденэ из расположенного в Париже высшего учебного заведения Ecole Normale Supérieure (ENS). ENS представляет собой университет, предлагающий обучение по различным направлениям, как для студентов старших курсов, так и послевузовское. ENS ведет свою историю с 1796 г., его специалисты хорошо известны в мире своими исследованиями в области криптографии.

Название алгоритма CS происходит от французского словосочетания Chiffrement Symétrique, что весьма просто переводится как «симметричный шифр». Алгоритм шифрует данные 64-битными блоками с использованием ключа переменной длины — до 128 битов.

CS был впервые представлен авторами на конференции Fast Software Encryption в 1998 г., а в сентябре 2000 г. предложен на конкурс NESSIE его авторами при участии Пьера-Алана Фуке (Pierre-Alain Fouque) из компании CS Communication & Systemes, владеющей правами на алгоритм [311].

## Структура алгоритма

Алгоритм CS представляет собой SP-сеть, состоящую из 8 раундов преобразований. Между раундами, а также перед первым раундом и после последнего, выполняется наложение на обрабатываемый блок данных одного из фрагментов расширенного ключа  $k_0 \dots k_8$  (процедура расширения ключа будет подробно описана далее) побитовой логической операцией «исключающее или» (XOR) — рис. 3.46 [367].

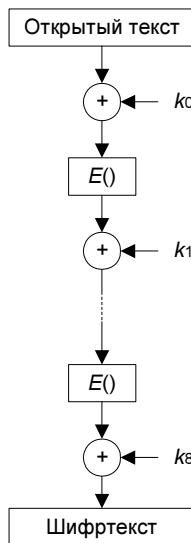


Рис. 3.46. Структура алгоритма CS

В каждом раунде (функция  $E()$  на рис. 3.46) выполняются следующие операции (рис. 3.47):

1. 64-битный блок данных делится на 4 слова по 16 битов, каждое из которых обрабатывается операцией  $M()$  (подробно описана далее).

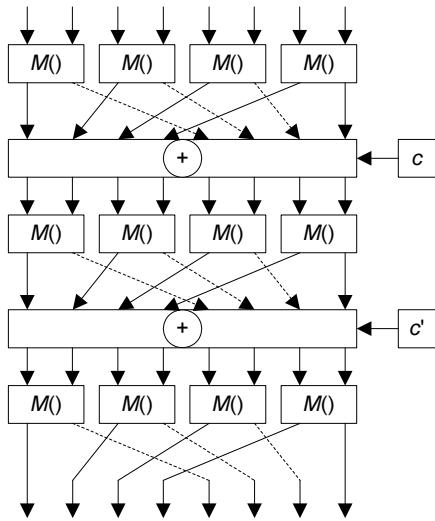


Рис. 3.47. Операция  $E$  алгоритма CS

- Выполняется байтовая перестановка результата предыдущего шага согласно таблице:

$$0, 2, 4, 6, 1, 3, 5, 7.$$

То есть байт 0 остается на своем месте, байт 2 становится байтом 1 и т. д. Байты отсчитываются справа налево.

- Результат предыдущего шага складывается операцией XOR с константой  $c$ , после чего повторяются шаги 1 и 2. 64-битные константы  $c$  и  $c'$  (используются на следующем шаге) получены из шестнадцатеричной записи первых 128 битов дробной части константы  $e$  и выглядят так:

$$c = \text{B7E151628AED2A6A};$$

$$c' = \text{BF7158809CF4F3C7}.$$

- Результат предыдущего шага обрабатывается аналогично шагу 3, но вместо константы  $c$  используется  $c'$ .

Функция  $M()$  обрабатывает 16-битные слова следующим образом (рис. 3.48):

- Входное значение  $x$  разбивается на два 8-битных фрагмента  $x_l$  и  $x_r$ .
- Вычисляются байты выходного значения:

$$y_l = P(h(x_l) \oplus x_r);$$

$$y_r = P((x_l \lll 1) \oplus x_r),$$

где:

- $<<<$  — операция циклического сдвига на указанное число битов влево;
- преобразование  $P()$  будет подробно описано далее;
- функция  $h()$  определена следующим образом:

$$h(n) = ((n <<< 1) \& 55 \oplus n).$$

Символом  $\&$  обозначена побитовая логическая операция «и», а 55 — шестнадцатеричная константа.

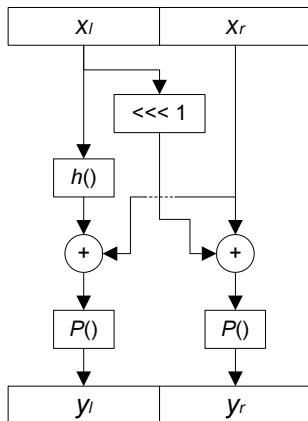


Рис. 3.48. Функция  $M$  алгоритма CS

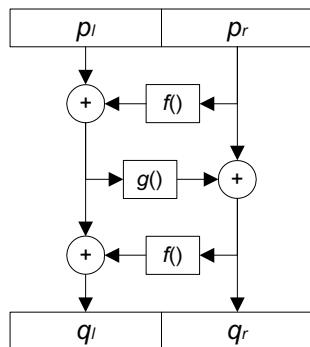


Рис. 3.49. Преобразование  $P$  алгоритма CS

3. Выходное значение  $y$  — результат конкатенации:

$$M(x) = y = y_l \bullet y_r.$$

Преобразование  $P()$  замещает 8-битное входное значение  $p = p_l \bullet p_r$  ( $p_l$  и  $p_r$  имеют размер по 4 бита) 8-битным значением  $q = q_l \bullet q_r$  путем следующих вычислений (рис. 3.49):

$$t = p_l \oplus f(p_r);$$

$$q_r = p_r \oplus g(t);$$

$$q_l = t \oplus f(q_r),$$

где:

- $t$  — временная переменная;
- функция  $g()$  будет подробно описана далее;
- функция  $f()$  определена согласно табл. 3.13.

Таблица 3.13

Входное значение	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Выходное значение	F	D	B	B	7	5	7	7	E	D	A	B	E	D	E	F

Выходное значение функции  $f()$  также может быть вычислено из входного следующим образом:

$$f(n) = \overline{n} \& (n <<< 1),$$

где  $\overline{n}$  — побитовый комплемент к  $n$ .

Функция  $g()$  является табличной заменой (табл. 3.14).

Таблица 3.14

Входное значение	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Выходное значение	A	6	0	2	B	E	1	8	D	4	5	3	F	C	7	9

Стоит сказать, что для ускорения шифрования данных преобразование  $P(p_l, p_r)$  может быть реализовано не описанными выше вычислениями, а в виде табличной замены, приведенной в табл. 3.15 (строки соответствуют значению  $p_l$ , столбцы —  $p_r$ ).

Таблица 3.15

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	29	0D	61	40	9C	EB	9E	8F	1F	85	5F	58	5B	01	39	86
1	97	2E	D7	D6	35	AE	17	16	21	B6	69	4E	A5	72	87	08
2	3C	18	E6	E7	FA	AD	B8	89	B7	00	F7	6F	73	84	11	63
3	3F	96	7F	6E	BF	14	9D	AC	A4	0E	7E	F6	20	4A	62	30
4	03	C5	4B	5A	46	A3	44	65	7D	4D	3D	42	79	49	1B	5C
5	F5	6C	B5	94	54	FF	56	57	0B	F4	43	0C	4F	70	6D	0A
6	E4	02	3E	2F	A2	47	E0	C1	D5	1A	95	A7	51	5E	33	2B
7	5D	D4	1D	2C	EE	75	EC	DD	7C	4C	A6	B4	78	48	3A	32
8	98	AF	C0	E1	2D	09	0F	1E	B9	27	8A	E9	BD	E3	9F	07
9	B1	EA	92	93	53	6A	31	10	80	F2	D8	9B	04	36	06	8E

Таблица 3.15 (окончание)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	BE	A9	64	45	38	1C	7A	6B	F3	A1	F0	CD	37	25	15	81
B	FB	90	E8	D9	7B	52	19	28	26	88	FC	D1	E2	8C	A0	34
C	82	67	DA	CB	C7	41	E5	C4	C8	EF	DB	C3	CC	AB	CE	ED
D	D0	BB	D3	D2	71	68	13	12	9A	B3	C2	CA	DE	77	DC	DF
E	66	83	BC	8D	60	C6	22	23	B2	8B	91	05	76	CF	74	C9
F	AA	F1	99	A8	59	50	3B	2A	FE	F9	24	B0	BA	FD	F8	55

## Расшифровывание

Расшифровывание выполняется применением обратных операций в обратной последовательности. Обратной к операции  $E()$  является операция  $E^{-1}()$ , приведенная на рис. 3.50.

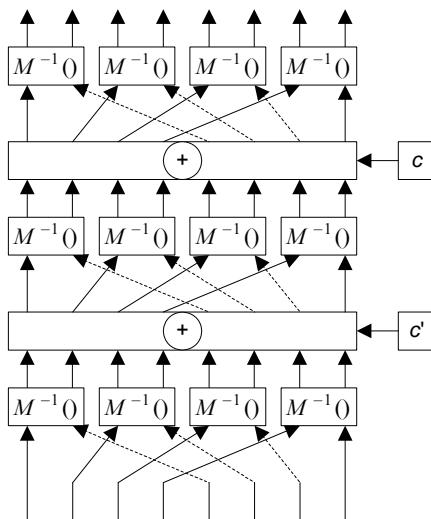


Рис. 3.50. Операция  $E^{-1}()$  алгоритма CS

Сначала в этой операции выполняется обратная (по отношению к применяемой в операции  $E()$ ) байтовая перестановка, затем применяется операция  $M^{-1}()$ , затем выполняется операция XOR обрабатываемых данных с константой  $c'$  и т. д.

Операция  $M^{-1}()$  также выполняется в несколько шагов:

1. Входное значение  $y$  разбивается на два 8-битных фрагмента  $y_l$  и  $y_r$ .
2. Вычисляются байты выходного значения:

$$x_l = h'(P(y_l) \oplus P(y_r));$$

$$x_r = (x_l \lll 1) \oplus P(y_r),$$

где функция  $h'()$  определена следующим образом:

$$h'(n) = ((n \lll 1) \& AA) \oplus n).$$

3. Выходное значение  $x$  — результат конкатенации:

$$M^{-1}(y) = x = x_l \bullet x_r.$$

## Процедура расширения ключа

Расширение ключа выполняется в несколько этапов. Сначала ключ шифрования (если его размер меньше 128 битов) дополняется справа нулями до 128 битов.

Затем дополненный 128-битный ключ  $k$  делится на две 64-битные части:

$$k = k_1 \bullet k_2,$$

на основе которых выполняется итеративное вычисление подключей  $k_0 \dots k_8$  (рис. 3.51):

$$k_i = k_{i-2} \oplus F(c_i, k_{i-1}).$$

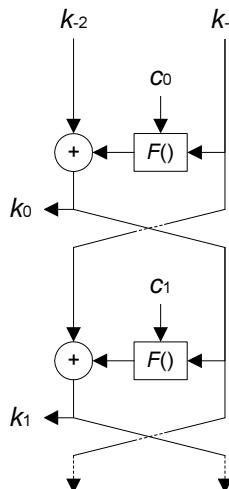


Рис. 3.51. Расширение ключа в алгоритме CS

Функция  $F()$  определена следующим образом:

$$F(n1, n2) = T(P'(n1 \oplus n2)),$$

где  $P'()$  обозначает параллельное применение функции  $P()$  к каждому байту входного 64-битного значения.

Преобразование  $P()$  было описано выше, а  $T()$  представляет собой битовую перестановку ( $x_i$  —  $i$ -й бит 64-битного значения  $x$ ):

$$T(x) = x_{63} \bullet x_{55} \bullet \dots \bullet x_7 \bullet x_{62} \bullet x_{54} \bullet \dots \bullet x_6 \bullet \dots \bullet x_{56} \bullet x_{48} \bullet \dots \bullet x_0,$$

т. е. первые 8 битов выходного значения составляются из каждого первого бита 8 байтов входного значения, затем используются 8 вторых битов и т. д.

Константы  $c_i$  получены из первых строк описанной выше таблицы преобразования  $P()$  и выглядят следующим образом:

$$c_0 = 290D61409CEB9E8F;$$

$$c_1 = 1F855F585B013986;$$

$$c_2 = 972ED7D635AE1716;$$

$$c_3 = 21B6694EA5728708;$$

$$c_4 = 3C18E6E7FAADB889;$$

$$c_5 = B700F76F73841163;$$

$$c_6 = 3F967F6EBF149DAC;$$

$$c_7 = A40E7EF6204A6230;$$

$$c_8 = 03C54B5A46A34465.$$

Как видно, при зашифровывании расширение ключа может выполняться «на лету», т. е. по мере необходимости. Однако при расшифровывании подключи применяются в обратной последовательности, поэтому расширение ключа необходимо выполнить до начала расшифровывания.

## Достоинства и недостатки алгоритма

Первичный криптоанализ алгоритма CS был проведен одним из его авторов — Сержем Воденэ [380], который доказал, что измененный вариант алгоритма с константами и подключами, замененными на случайные величины, не подвержен дифференциальному и линейному криптоанализу. Кроме того, для достижения стойкости против данных видов атак достаточно  $5\frac{1}{3}$  раундов алгоритма.

Эксперты конкурса NESSIE в отчете [313] также подтверждают, что у алгоритма CS не обнаружено слабостей и каких-либо атак на него. Однако

исследования производительности алгоритмов — участников конкурса показали, что алгоритм CS является наиболее медленным среди всех 64-битных участников [308, 311]. Поэтому CS не вышел во второй этап конкурса именно из-за низкой скорости шифрования [308].

### 3.14. Алгоритм DEAL

Алгоритм DEAL разработан в 1998 г. известным криптоаналитиком Ларсом Кнудсеном. Ларс Кнудсен является автором большого числа криптоаналитических исследований, описывающих различные виды атак на многие известные алгоритмы шифрования, в частности, на бывший тогда стандартом шифрования данных США алгоритм DES (который также называется DEA — Data Encryption Algorithm) [150].

#### Основные характеристики и структура алгоритма

Название алгоритма DEAL произошло именно от DEA; по замыслу автора алгоритма, «DEAL» — это DEA (DES) с длинным блоком. Фактически DEAL — это вариант алгоритма DES с увеличенным размером блока и увеличенной длиной ключа. По своей структуре DEAL представляет собой сеть Фейстеля, в каждом раунде которой для преобразования субблока данных используется обычный алгоритм DES (рис. 3.52) [211, 295].

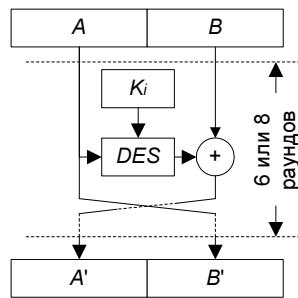


Рис. 3.52. Структура алгоритма DEAL

Алгоритм шифрует данные 128-битными блоками с использованием трех фиксированных размеров ключей: 128, 192 и 256 битов. При использовании 256-битного ключа выполняется 8 раундов шифрования, для ключей остальных

размеров — 6 раундов. В каждом раунде алгоритма выполняются следующие действия:

$$\begin{aligned} A_i &= B_{i-1} \oplus DES_{K_i}(A_{i-1}); \\ B_i &= A_{i-1}, \end{aligned}$$

где:

- $i$  — номер текущего раунда;
- $A_i$  и  $B_i$  — текущие значения левого и правого субблоков соответственно;
- $K_i$  — ключ  $i$ -го раунда.

Как было сказано выше, при использовании 128- или 192-битного ключа для шифрования двойного блока DES выполняется 6 DES-шифрований. Это означает, что скорость шифрования алгоритма DEAL полностью эквивалентна скорости алгоритма Triple DES, который шифрует 64-битный блок данных тремя последовательными DES-шифрованиями (а при 256-битном ключе DEAL шифрует еще медленнее). Стоит сказать, что Triple DES никогда не считался быстрым алгоритмом, поэтому можно утверждать, что по быстродействию DEAL заведомо проигрывает многим современным алгоритмам шифрования.

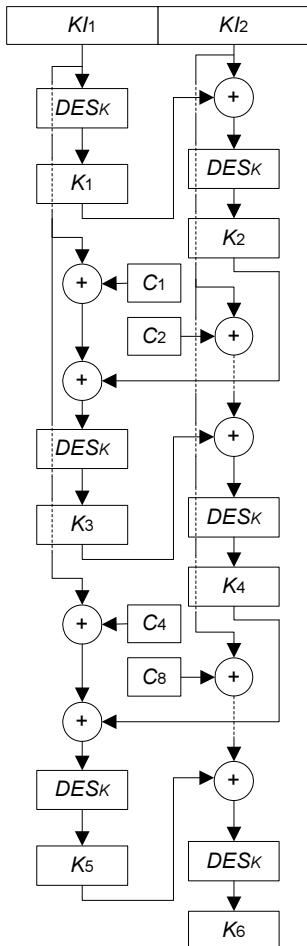
## Процедура расширения ключа

Процедура расширения ключа предназначена для генерации 6 или 8 ключей  $K_i$  из исходного ключа шифрования. Процесс генерации ключей раундов не значительно различается в зависимости от размера ключа шифрования. Для 128-битных ключей расширение выполняется следующим образом (рис. 3.53):

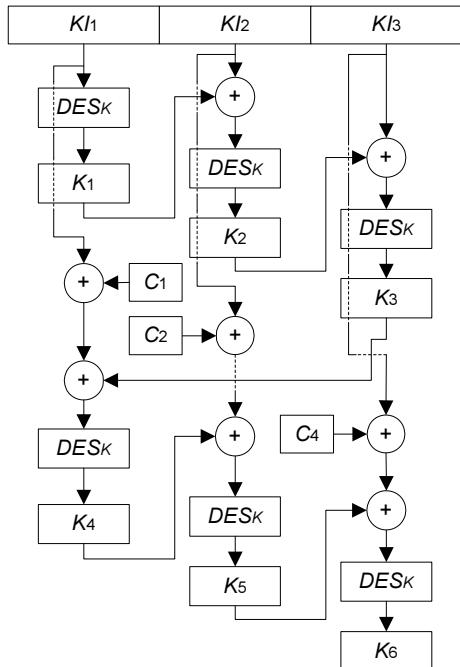
$$\begin{aligned} K_1 &= DES_K(KI_1); \\ K_2 &= DES_K(KI_2 \oplus K_1); \\ K_3 &= DES_K(KI_1 \oplus C_1 \oplus K_2); \\ K_4 &= DES_K(KI_2 \oplus C_2 \oplus K_3); \\ K_5 &= DES_K(KI_1 \oplus C_4 \oplus K_4); \\ K_6 &= DES_K(KI_2 \oplus C_8 \oplus K_5), \end{aligned}$$

где:

- $KI_n$  —  $n$ -й 64-битный фрагмент исходного ключа шифрования;
- $C_j$  — 64-битные константы, в которых бит  $j$  установлен в 1, а остальные биты обнулены;



**Рис. 3.53.** Процедура расширения 128-битного ключа



**Рис. 3.54.** Процедура расширения 192-битного ключа

- $K$  — константный ключ, определенный следующим образом (шестнадцатеричное значение):

$$K = 0123456789ABCDEF.$$

Расширение 192-битного ключа выполняется аналогично (рис. 3.54):

$$K_1 = DES_K(KI_1);$$

$$K_2 = DES_K(KI_2 \oplus K_1);$$

$$K_3 = DES_K(KI_3 \oplus K_2);$$

## Описание алгоритмов

$$K_4 = DES_K(KI_1 \oplus C_1 \oplus K_3);$$

$$K_5 = DES_K(KI_2 \oplus C_2 \oplus K_4);$$

$$K_6 = DES_K(KI_3 \oplus C_4 \oplus K_5).$$

Немного сложнее вычисляются 8 ключей раундов при использовании 256-битного ключа (рис. 3.55):

$$K_1 = DES_K(KI_1);$$

$$K_2 = DES_K(KI_2 \oplus K_1);$$

$$K_3 = DES_K(KI_3 \oplus K_2);$$

$$K_4 = DES_K(KI_4 \oplus K_3);$$

$$K_5 = DES_K(KI_1 \oplus C_1 \oplus K_4);$$

$$K_6 = DES_K(KI_2 \oplus C_2 \oplus K_5);$$

$$K_7 = DES_K(KI_3 \oplus C_4 \oplus K_6);$$

$$K_8 = DES_K(KI_4 \oplus C_8 \oplus K_7).$$

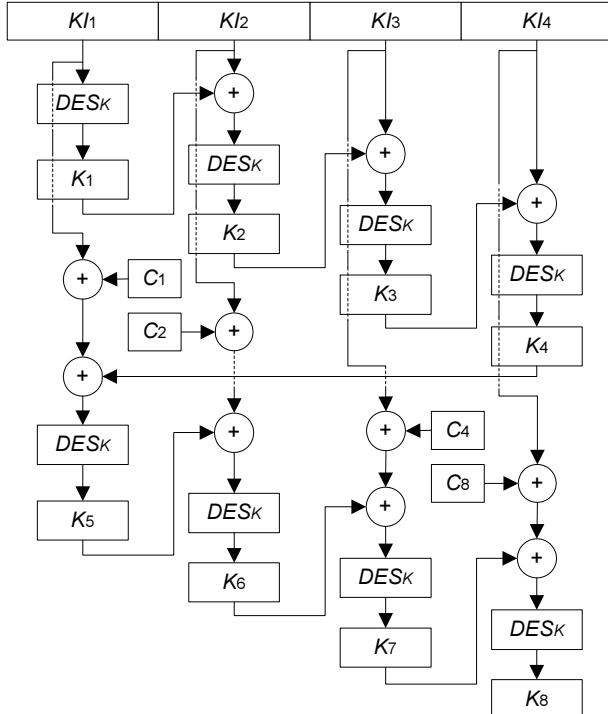


Рис. 3.55. Процедура расширения 256-битного ключа

## Достоинства и недостатки алгоритма

Алгоритм DEAL имеет единственное ярко выраженное достоинство — его можно реализовать с использованием аппаратуры шифрования, реализующей алгоритм DES [211]. Как известно, за те 20 лет, в течение которых DES был стандартом шифрования США, было произведено огромное количество аппаратных и программных средств, реализующих DES. Поэтому было разработано несколько вариантов DES (аналогично DEAL, с увеличенной длиной ключа и другими усовершенствованиями), позволяющих использовать все это множество DES-шифраторов. К сожалению, подавляющее большинство из них оказались проблемными с точки зрения криптостойкости (подробную информацию см. в разд. 3.15).

Несомненным же недостатком алгоритма является его достаточно невысокая скорость, которая обсуждалась выше. Кроме того, известны две криptoаналитические работы, в которых доказана недостаточная криптостойкость алгоритма DEAL.

- Известные криptoаналитики Джон Келси (John Kelsey) и Брюс Шнайер из американской компании Counterpane обнаружили у алгоритма DEAL наличие классов эквивалентных ключей и предложили алгоритм нахождения эквивалентных ключей. Такая проблема заметно сужает область использования алгоритма, в частности, не позволяет строить на его основе односторонние хэш-функции [196].
- Стефан Лакс из университета Манхейма, Германия, предложил атаку, позволяющую вскрыть DEAL со 192-битным ключом при наличии  $2^{56}$  выбранных шифртекстов (и соответствующих им открытых текстов) путем выполнения порядка  $2^{145}$  операций шифрования алгоритмом DES. Нельзя сказать, что эта атака является практически осуществимой, однако Стефан Лакс также доказал, что DEAL со 192-битным ключом по криптостойкости эквивалентен DEAL со 128-битным ключом [246] — эксперты конкурса AES посчитали это уже серьезным недостатком алгоритма [284].

Памятую о фактическом отсутствии заметных достоинств алгоритма DEAL, эксперты конкурса AES не выбрали данный алгоритм во второй раунд конкурса [284].

## 3.15. Алгоритм DES и его варианты

Прошло уже более 30 лет с даты принятия алгоритма DES в качестве стандарта шифрования США. DES — алгоритм шифрования с наиболее богатой и интересной историей.

## История создания алгоритма

Один из наиболее известных в мире криптологов Брюс Шнайер в своей знаменитой книге «Прикладная криптография» [28] так описал проблемы пользователей средств защиты информации в начале 70-х гг. XX века (естественно, речь идет о пользователях по ту сторону «железного занавеса»):

- не было как общепринятого стандарта шифрования данных, так и просто достаточно широко используемых алгоритмов защиты информации, поэтому о совместимости между различными программными или аппаратными средствами шифрования не могло быть и речи;
- практически любое средство шифрования представляло собой «черный ящик» с достаточно неясным содержимым: какой алгоритм шифрования используется, насколько он является криптографически стойким, грамотно ли он реализован, корректно ли создаются, хранятся, используются ключи шифрования, нет ли в средстве вставленных разработчиками недокументированных возможностей и т. д., — вся эта весьма важная информация для подавляющего большинства покупателей криптографических средств была недоступна.

Данной проблемой озабочилось Национальное Бюро Стандартов (National Bureau of Standards, NBS) США. В результате в 1973 г. был объявлен первый в истории открытый конкурс на стандарт шифрования. NBS было готово исследовать с целью выбора стандарта алгоритмы-претенденты, удовлетворяющие следующим критериям:

- алгоритм должен быть криптографически стойким;
- алгоритм должен быть быстрым;
- структура алгоритма должна быть четкой и ясной;
- стойкость шифрования должна зависеть только от ключа, сам алгоритм не должен быть секретным;
- алгоритм должен быть легко применим для различных целей;
- алгоритм должен легко реализовываться аппаратно на существующей элементной базе.

Предполагалось, что заинтересованные организации или специалисты будут присыпать в NBS подробные спецификации алгоритмов, достаточные для их реализации, т. е. не имеющие каких-либо «белых пятен». Предполагалось также, что алгоритм будет сертифицирован NBS для всеобщего использования, с него будут сняты все патентные и экспортные ограничения, в результате чего такой стандарт должен будет решить все проблемы совместимости средств шифрования. Кроме того, NBS брало на себя функции сертификации

средств шифрования — т. е. «черные ящики» должны были безвозвратно уйти в прошлое.

Фактически алгоритм-претендент оказался всего один: это был разработанный фирмой IBM алгоритм шифрования Lucifer (см. разд. 3.31). В течение двух лет проводилась доработка алгоритма:

- во-первых, NBS совместно с Агентством Национальной Безопасности (АНБ, NSA — National Security Agency) США был проведен тщательный анализ алгоритма, результатом которого явилась его достаточно существенная переработка;
- во-вторых, принимались к рассмотрению комментарии и критические замечания от всех заинтересованных организаций и частных лиц.

В результате совместной деятельности IBM, NBS и АНБ в январе 1977 г. DES был опубликован как стандарт США (последняя версия этого стандарта — в документе [150]) на алгоритм шифрования данных (кроме информации повышенной степени секретности). Алгоритм DES был запатентован фирмой IBM, однако NBS получило, фактически, бесплатную и неограниченную лицензию на использование данного алгоритма [150]. Альтернативное, но реже используемое название алгоритма — DEA (Data Encryption Algorithm).

## Основные характеристики и структура алгоритма

Алгоритм DES шифрует информацию блоками по 64 бита с помощью 64-битного ключа шифрования, в котором используется только 56 битов (процедура расширения ключа подробно описана далее).

Шифрование информации выполняется следующим образом (рис. 3.56):

1. Над 64-битным блоком данных производится начальная перестановка согласно табл. 3.16.

**Таблица 3.16**

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Таблица трактуется следующим образом: значение входного бита 58 (здесь и далее все биты нумеруются слева направо, начиная с 1-го) помещается в выходной бит 1, значение 50-го бита — в бит 2 и т. д.

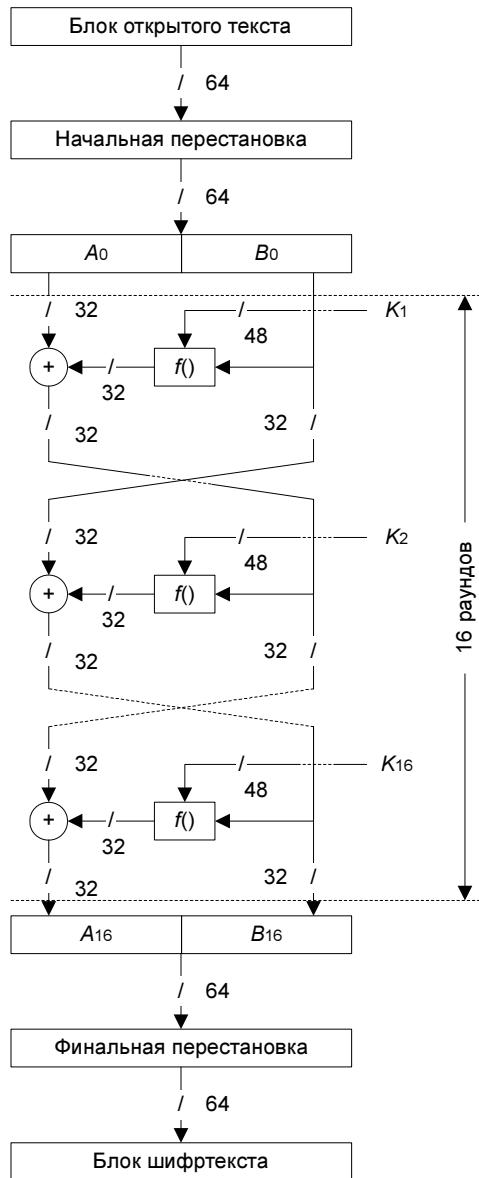


Рис. 3.56. Структура алгоритма DES

2. Результат предыдущей операции делится на 2 субблока по 32 бита (на рис. 3.56 обозначены  $A_0$  и  $B_0$ ), над которыми производятся 16 раундов следующих преобразований:

$$\begin{aligned}A_i &= B_{i-1}; \\B_i &= A_{i-1} \oplus f(B_{i-1}, K_i),\end{aligned}$$

где:

- $i$  — номер текущего раунда;
- $K_i$  — ключ раунда;
- $\oplus$  — побитовая логическая операция «исключающее или» (XOR).

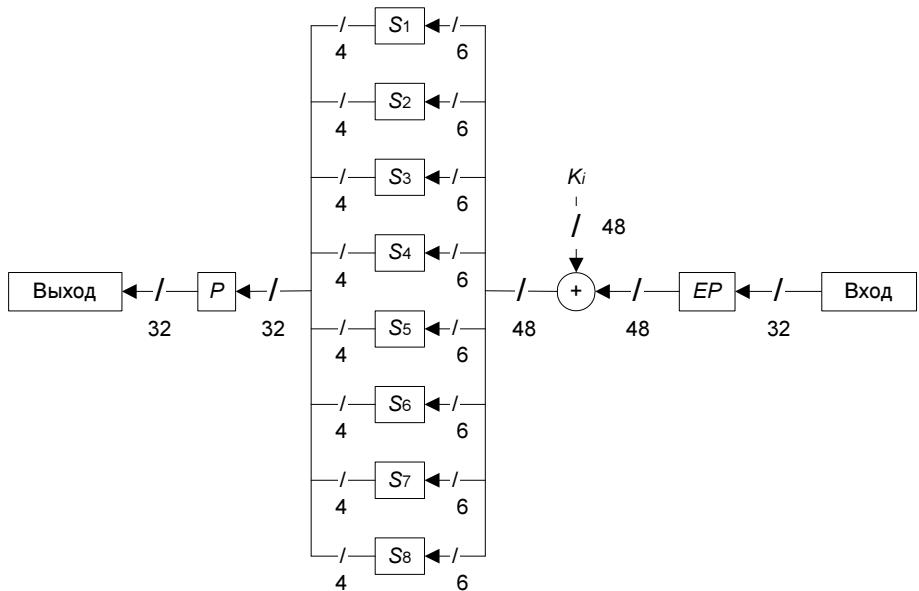
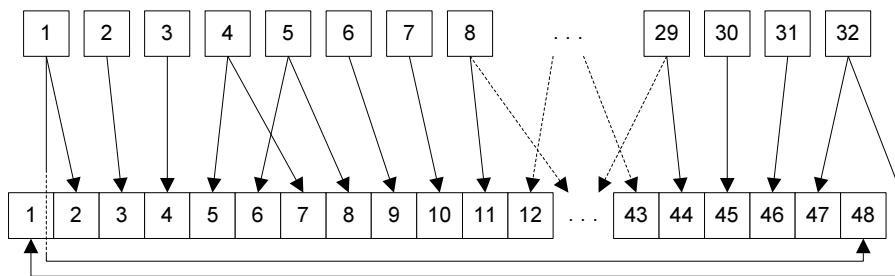
Структура функции раунда  $f()$  приведена на рис. 3.57. Эта функция выполняется в несколько шагов.

- Над 32-битным входом выполняется расширяющая перестановка  $EP$  (см. рис. 3.58). Данная операция решает две задачи: во-первых, расширяет входное значение до 48 битов для последующего сложения с ключом раунда; во-вторых, обеспечивает влияние «размножаемых» битов на две таблицы замен (описаны далее) вместо одной, что ускоряет возникновение зависимости каждого бита шифртекста от каждого входного бита [28], что называется *лавинным эффектом*.
- Результат предыдущего шага складывается с ключом раунда  $K_i$  операцией XOR.
- Результат сложения разбивается на 8 фрагментов по 6 битов, каждый из которых «прогоняется» через соответствующую таблицу замен ( $S_1 \dots S_8$ ). Таблицы замен являются фиксированными и приведены в *Приложении 1*. Каждая таблица содержит по 4 строки, содержащих по 16 значений от 0 до 15. Входное значение интерпретируется следующим образом: два крайних бита формируют номер строки (от 0 до 3), из которой выбирается число, расположеннное в столбце, номер которого соответствует значению четырех остальных битов входа. Например, при двоичном входе 101100 (десятичное число 44) выбирается значение шестой ячейки второго столбца.
- На последнем шаге 4-битные значения, полученные после выполнения замен, объединяются, после чего над ними выполняется операция  $P$ , представляющая собой простую перестановку согласно табл. 3.17.

*Таблица 3.17*

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8

## Описание алгоритмов

Рис. 3.57. Функция  $f()$ Рис. 3.58. Расширяющая перестановка  $EP$ 

Стоит отметить, что в последнем раунде алгоритма субблоки не меняются местами.

- Полученные субблоки  $A_{16}$  и  $B_{16}$  объединяются в 64-битный блок, над которым выполняется финальная перестановка данных согласно табл. 3.18.

Финальная перестановка является инверсной по отношению к начальной перестановке, выполняемой на этапе 1. Результат финальной перестановки и является блоком зашифрованных данных.

Таблица 3.18

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

Расшифровывание данных алгоритмом DES выполняется абсолютно так же, как и зашифровывание, однако с обратным порядком использования ключей раундов: в  $i$ -м раунде расшифровывания используется ключ  $K_{(17-i)}$ .

## Процедура расширения ключа

Схема процедуры расширения ключа показана на рис. 3.59. Ее задача — формирование 16 ключей раундов, которое выполняется следующим образом.

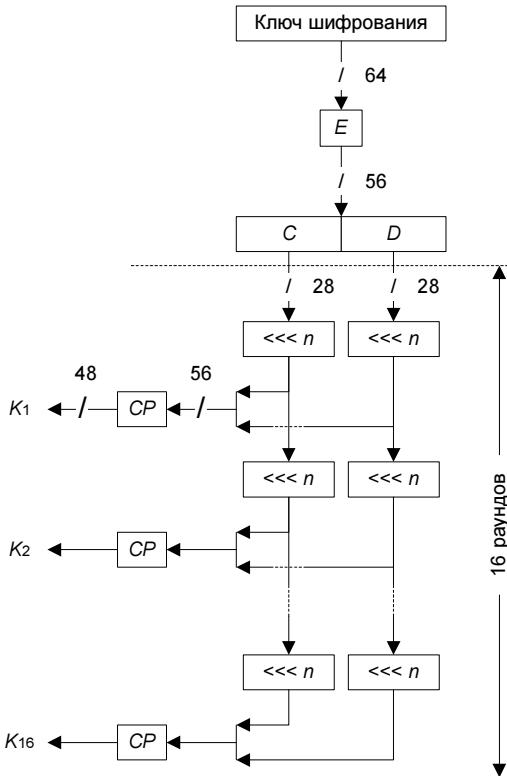


Рис. 3.59. Процедура расширения ключа

## Описание алгоритмов

Как было сказано выше, из 64-битного ключа шифрования алгоритм DES использует только 56 битов. Каждый 8-й бит отбрасывается и никак не применяется в алгоритме, причем использование оставшихся битов ключа шифрования в реализациях алгоритма DES никак не лимитировано стандартом [150]. Процедура извлечения 56 значащих битов 64-битного ключа на рис. 3.59 обозначена как  $E$ . Помимо извлечения, данная процедура выполняет еще и перестановку битов ключа согласно табл. 3.19 и 3.20.

**Таблица 3.19**

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36

**Таблица 3.20**

63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

В результате перестановки формируются два 28-битных значения  $C$  и  $D$ . Таблица 3.19 определяет выборку битов ключа для  $C$ , табл. 3.20 — для  $D$ .

Затем выполняются 16 раундов преобразований, каждый из которых дает один из ключей раундов  $K_i$ . В каждом раунде процедуры расширения ключа производятся следующие действия:

1. Текущие значения  $C$  и  $D$  циклически сдвигаются влево на переменное число битов  $n$ . Для раундов 1, 2, 9 и 16  $n=1$ , в остальных раундах выполняется циклический сдвиг на 2 бита.
2.  $C$  и  $D$  объединяются в 56-битное значение, к которому применяется сжимающая перестановка  $CP$ , результатом которой является 48-битный ключ раунда  $K_i$ . Сжимающая перестановка выполняется согласно табл. 3.21.

**Таблица 3.21**

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

Сочетание циклического сдвига и сжимающей перестановки приводит к тому, что в каждом из ключей раундов используется уникальный набор битов ключа шифрования.

При расшифровании данных можно использовать ту же процедуру расширения ключа, но применять ключи раундов в обратном порядке. Есть и другой вариант: в каждом раунде процедуры расширения ключа вместо циклического сдвига влево выполнять циклический сдвиг вправо на  $n'$  битов, где  $n'=0$  для первого раунда,  $n'=1$  для раундов 2, 9, 16 и  $n'=2$  для остальных раундов. Такая процедура расширения ключа сразу даст нужные для расшифровывания ключи раундов  $K_{(17-i)}$ .

Стоит сказать, что возможность выполнения расширения ключа «на лету» (особенно если эта возможность существует как при зашифровывании, так и при расшифровывании) считается достоинством алгоритмов шифрования, поскольку в этом случае расширение ключа можно выполнять параллельно шифрованию и не тратить память на хранение ключей других раундов, кроме текущего.

## Криптостойкость алгоритма DES

С самого начала использования алгоритма DES криptoаналитики всего мира прилагали множество усилий по его взлому. Фактически DES дал невиданный доселе толчок развитию криptoанализа. Вышли сотни трудов, посвященных различным методам криptoанализа именно в приложении к алгоритму DES, а также деталям самого алгоритма и их влиянию на криптостойкость DES. Можно утверждать, что именно благодаря DES появились целые направления криptoанализа, такие как:

- линейный криptoанализ (см. разд. 1.5);
- дифференциальный криptoанализ (см. разд. 1.5);
- криptoанализ на связанных ключах (см. разд. 1.9); стоит сказать, что в работе [56] было доказано, что DES неуязвим к данному виду атак благодаря тому, что в описанной выше процедуре расширения ключа циклический сдвиг выполняется на различное число битов в разных раундах.

«DES стойко выдержал 20 лет массового всемирного криptoанализа» [123] — десятилетия криptoанализа не привели к обнаружению серьезных уязвимостей в алгоритме. Основными результатами усилий по взлому DES можно считать следующие.

- Японский специалист Мицуру Мацуи (Mitsuru Matsui), изобретатель линейного криptoанализа, в 1993 г. доказал возможность вычисления ключа шифрования DES методом линейного криptoанализа при наличии у атакующего  $2^{47}$  пар известных открытых текстов [256].
- Криптологи из Израиля — изобретатели дифференциального криptoанализа Эли Бихам и Эди Шамир (считается, что дифференциальный кри-

тоанализ был известен АНБ США еще в 1970-х гг. [28], однако, именно Бихам и Шамир сформулировали данный вид атак и сделали его доступным всему криптологическому сообществу) — в 1991 г. представили атаку, в которой ключ шифрования вычисляется методом дифференциального криptoанализа при наличии у атакующего возможности генерации  $2^{47}$  пар выбранных открытых текстов [78].

- В 1994 г. Эли Бихам и Алекс Бирюков усилили известный с 1987 г. метод вычисления ключа DES — метод Дэвиса (Davies), основанный на специфических свойствах таблиц замен DES [59]. Усиленный метод позволяет вычислить 6 битов ключа DES (остальные 50 битов — полным перебором возможных вариантов) при наличии  $2^{50}$  пар известных открытых текстов и шифртекстов или вычислить 24 бита ключа при наличии  $2^{52}$  пар.

В дальнейшем эти атаки были несколько усилены (например, атака линейным криptoанализом при наличии  $2^{43}$  пар известных открытых текстов вместо  $2^{47}$  [263]), появлялись также новые виды атак на DES (например, атака, позволяющая вычислить ключ высокоточным облучением аппаратного шифратора и последующим анализом ошибок шифрования [81]). Однако стоит сказать, что все эти атаки требуют наличия огромного количества пар «открытый текст — шифртекст», получение которых на практике является настолько трудоемкой операцией, что наиболее простой атакой на DES считается полный перебор возможных вариантов ключа шифрования [263].

Кроме того, практически сразу после появления DES были обнаружены следующие проблемы с ключами шифрования DES [28].

- 4 ключа из возможных  $2^{56}$  ключей алгоритма являются слабыми (т. е. не обеспечивают требуемой стойкости при зашифровывании). Это ключи, в которых все биты какой-либо из половин расширяемого ключа ( $C$  или  $D$  на рис. 3.59) являются нулевыми или единичными. В этом случае все ключи раундов будут одинаковыми.
- 6 пар ключей являются эквивалентными (т. е. информация, зашифрованная одним ключом из пары, расшифровывается другим ключом той же пары), например, пара ключей E0FEE0FEF1FEF1FE и FEE0FEE0FEE1FEE1 (шестнадцатеричные значения). Процедура расширения такого ключа вместо 16 ключей раундов вырабатывает всего 2 различных ключа.
- 48 ключей являются «возможно слабыми», их полный список приведен в [28]. Возможно слабые ключи при их расширении дают только 4 различных ключа раундов, каждый из которых используется при шифровании по 4 раза.

□ Существует свойство комплементарности ключей: если

$$E_k(M) = C,$$

где:

- $E_k(M)$  — зашифровывание алгоритмом DES блока открытого текста  $M$ ;
- $C$  — полученный в результате шифртекст;

то:

$$E_{\bar{k}}(\overline{M}) = \overline{C},$$

где  $\bar{x}$  — побитовое дополнение к  $x$  (т. е. величина, полученная путем замены всех битовых нулей значения  $x$  на единицы, и наоборот). Считается, что свойство комплементарности ключей не является слабостью, однако оно приводит к тому факту, что для полного перебора ключей DES требуется  $2^{55}$ , а не  $2^{56}$  попыток.

Однако и эти недостатки не являются существенными, поскольку в программной или аппаратной реализации DES достаточно просто запретить использование проблемных ключей.

Достаточно много криptoаналитических исследований было посвящено алгоритму DES с уменьшенным количеством раундов. Считается, что количество раундов любого многораундового алгоритма шифрования можно уменьшать до определенных пределов, достигая оптимального для конкретных применений соотношения «скорость шифрования/криптостойкость». Однако алгоритм DES (как и многие другие алгоритмы шифрования) подвержен существенному снижению криптостойкости при уменьшении количества раундов — исследования DES с различным количеством раундов, меньшим 16 (например [75, 165, 249, 309]), показали: «Атаки на алгоритмы с уменьшенным количеством раундов доказывают, что не стоит уменьшать количество раундов алгоритма с целью увеличения производительности» [309].

Криptoанализ алгоритма DES, как полного, так и с уменьшенным количеством раундов, активно ведется и в XXI веке, несмотря на то, что DES уже не является стандартом шифрования (например, [122, 123, 165, 315]). Все-таки вряд ли будет преувеличением утверждение, что DES по-прежнему остается самым используемым из алгоритмов симметричного шифрования в мире (особенно с учетом распространенности описанного далее Triple DES). Поэтому криptoанализ DES и его вариантов, как сказано в [123], будет актуален еще 10–20 лет.

В своей работе [27], адресованной начинающим криptoаналитикам, Брюс Шнайер рекомендует тренироваться или оттачивать мастерство криptoанализа, в том числе на алгоритме DES и некоторых его вариантах.

## Продолжение истории алгоритма

Как было сказано выше, несмотря на массированный криптоанализ, не было изобретено достаточно практического алгоритма вскрытия DES быстрее, чем полным перебором  $2^{55}$  (с учетом свойства комплементарности) возможных вариантов ключа. Причем с развитием компьютерной техники полный перебор ключа становился все более реальным. В 1993 г. Майкл Винер (Michael Wiener) разработал принципы конструкции специализированного компьютера стоимостью порядка 1 000 000 долларов, способного перебрать ключи DES за 3,5 часа. Причем такой компьютер имел возможность наращивания — при нефантастических для крупной организации или спецслужбы затратах порядка 10 000 000 долларов полный перебор ключа DES должен был занять не более 21 минуты [394]. Абсолютно ясно, что сейчас такой компьютер стоил бы в десятки раз дешевле [315].

С другой стороны, при принятии DES в качестве стандарта планировался его пересмотр каждые 5 лет. Поэтому данный стандарт пересматривался в 1983, 1988, 1993 и 1999 гг. Последний пересмотр происходил уже после старта конкурса AES, целью которого был выбор нового стандарта шифрования США вместо DES (см. разд. 2.1). В 1999 г. уже невозможно было использовать DES для серьезной защиты из-за его короткого ключа, к этому времени уже были прецеденты вскрытия DES полным перебором ключа с использованием распределенных вычислений. Поэтому текущая версия стандарта [150] устанавливает в качестве стандарта шифрования Triple DES, а собственно DES разрешалось использовать только в целях совместимости с действующими DES-шифраторами, причем везде, где это возможно, рекомендовалось осуществить переход на Triple DES.

Стоит сказать и о том, что все время использования алгоритма DES существовал слух о том, что при его доработке АНБ США внедрило в алгоритм некую лазейку, с помощью которой АНБ могло бы вскрывать каким-либо методом зашифрованные алгоритмом DES сообщения [28, 241, 330]. Особенное подозрение у криптологического сообщества вызывали таблицы замен. Однако за все время использования DES никому не удалось полностью доказать как наличие, так и отсутствие подобных лазеек, хотя различных утверждений на данную тему было великое множество (например, в [249]). Кроме того, авторитетные эксперты высказали мнение, что, не считая уменьшения размера ключа, доработка алгоритма Lucifer серьезно усилила полученный алгоритм DES против дифференциального криптоанализа [48].

Короткий ключ алгоритма DES и описанные выше некритичные уязвимости алгоритма подтолкнули специалистов по криптологии на изобретение множества вариантов алгоритма DES, причем некоторые из них оказались достаточно успешными. «Потомки» алгоритма DES заслуживают отдельного рассмотрения.

## Алгоритм Double DES

Наиболее логичным способом противодействия полному перебору ключа DES выглядит многократное зашифровывание данных алгоритмом DES с различными ключами. Следующий алгоритм получил название Double DES (двойной DES):

$$C = E_{k_{2/2}}(E_{k_{1/2}}(M)),$$

где:

- $k_{1/2}$  и  $k_{2/2}$  — половины двойного ключа алгоритма Double DES, каждая из которых представляет собой обычный 56-битный ключ DES;
- $E$  — функция зашифровывания блока данных обычным алгоритмом DES.

Если бы при двойном шифровании DES выполнялось следующее свойство:

$$C = E_{k_{2/2}}(E_{k_{1/2}}(M)) = E_k(M)$$

для любых значений  $k_{1/2}$  и  $k_{2/2}$ , то двойное шифрование не приводило бы к усилению против полного перебора ключа — всегда нашелся бы такой ключ  $k$ , *однократное* зашифровывание которым было бы эквивалентно двукратному шифрованию на ключах  $k_{1/2}$  и  $k_{2/2}$ , а для нахождения ключа  $k$  достаточно было бы перебрать  $2^{55}$  ключей. К счастью, DES не обладает таким свойством, что доказано в [112] и [188], поэтому Double DES действительно удваивает эффективный размер ключа — до 112 битов, а при современном развитии вычислительной техники полный перебор 112-битного ключа невозможен.

Однако у Double DES есть другая проблема — атака «встреча посередине», предложенная известными криптологами Ральфом Мерклем и Мартином Хеллманом [266]. С помощью этой атаки криptoаналитик может получить  $k_{1/2}$  и  $k_{2/2}$  при наличии всего двух пар открытого текста и шифртекста ( $M_1, C_1$ ) и ( $M_2, C_2$ ) следующим образом (см. также разд. 1.6):

1. Выполняется зашифровывание  $E_{k_x}(M_1)$  на всем ключевом пространстве с записью результатов в некоторую таблицу.
2. Производится расшифровывание  $D_{k_y}(C_1)$  также на всем ключевом пространстве; результаты расшифровывания сравниваются со всеми записями в таблице, сформированной на шаге 1.
3. Если какой-либо результат, полученный на шаге 2, совпал с одним из результатов шага 1, то можно предположить, что нужный ключ найден, т. е. найдены соответствующие совпадающему результату  $k_x = k_{1/2}$  и  $k_y = k_{2/2}$ . Однако таких совпадений может быть много, их количество оценивается в [3] как  $2^{48}$ .

4. Для отсечения «ложных» ключей необходимо повторить предыдущие шаги с парой  $(M_2, C_2)$ , сузив пространство перебора до только вариантов, приводящих к совпадениям (т. е. примерно  $2^{48}$ ). Вероятность наличия более, чем одного совпадения после повторного перебора оценивается в [3] как  $2^{-16}$ .

Такая атака, выполняющая, фактически, перебор половинок двойного ключа, как со стороны открытого текста, так и со стороны шифртекста, требует примерно в 4 раза больше вычислений, чем перебор обычного ключа DES, однако требует также много памяти для хранения промежуточных результатов. Тем не менее, атака является реально осуществимой на практике, поэтому алгоритм Double DES не используется.

## Алгоритм 2-key Triple DES

Многократное шифрование по сравнению с какой-либо оптимизацией структуры алгоритма, несомненно, имеет серьезнейшее преимущество: подавляющее большинство реализаций алгоритма DES (как программных, так и аппаратных) может быть использовано для многократного шифрования данных — просто данные через шифратор нужно будет «прогонять» несколько раз.

Поскольку малая длина ключа DES вызывала беспокойство у криптологического сообщества с момента принятия DES в качестве стандарта, уже в 1978 г. была предложена схема тройного шифрования с двойным, 112-битным, ключом (алгоритм 2-key Triple DES, т. е. тройной DES с двойным ключом — см. рис. 3.60):

$$C = E_{k_{1/2}}(D_{k_{2/2}}(E_{k_{1/2}}(M))).$$

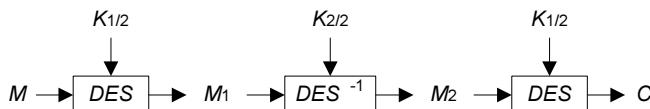


Рис. 3.60. Двухключевой Triple DES

Такая схема называется EDE (Encrypt-Decrypt-Encrypt, «зашифровывание-расшифровывание-зашифровывание»). Эта схема интересна тем, что при использовании одинаковых половинок ключа ( $k_{1/2} = k_{2/2}$ ) зашифровывание алгоритмом Triple DES эквивалентно однократному шифрованию алгоритмом DES на том же ключе [266]:

$$E_{k_{1/2}}(D_{k_{1/2}}(E_{k_{1/2}}(M))) = E_{k_{1/2}}(M).$$

Однако в той же работе [266] Меркль и Хеллман обосновали следующую атаку на двухключевой Triple DES при наличии выбранной пары «открытый текст — шифртекст» ( $M, C$ ):

- Предполагаем, что промежуточное значение  $M_1$  (см. рис. 3.60) равно нулю. Для каждого возможного значения  $k_{1/2}$  расшифровываем нулевой блок до нахождения существующего  $M$ :

$$M = D_{k_{1/2}}(M_1).$$

- $M$  зашифровывается алгоритмом Triple DES для получения  $C$ .
- Для каждого возможного значения  $k_{2/2}$  вычисляется значение  $M_2$ , результаты вычислений записываются в таблицу:

$$M_2 = D_{k_{2/2}}(M_1).$$

- Для каждого возможного значения  $k_{1/2}$   $M_2$  вычисляется со стороны  $C$ :

$$M_2 = D_{k_{1/2}}(C).$$

- Осуществляется поиск совпадений в таблице. Совпадений ожидается порядка  $2^{48}$ ; для получения верного ключа (т. е. пары  $k_{1/2}$  и  $k_{2/2}$ ) используется еще одна пара ( $M, C$ ).

Выбор пары ( $M, C$ ) выполняется именно таким образом, чтобы промежуточное значение  $M_1$  было равно нулю.

Описанная атака достаточно сложно применима на практике, однако ее достаточно для наличия сомнений в криптостойкости алгоритма 2-key Triple DES, поэтому Меркль и Хеллман рекомендуют использовать Triple DES с тремя независимыми ключами (3-key Triple DES) [266], аналогичную рекомендацию дает и Шнайер в [28].

Позже была найдена атака и по известному (а не выбранному) открытому тексту, несколько более практичная, чем описанная выше [28].

Алгоритм Triple DES достаточно часто называют TDEA (Triple Data Encryption Algorithm) или DES3 [241].

## Алгоритм 3-key Triple DES

Трехключевой Triple DES аналогичен двухключевому, но шифрование в нем выполняется с тремя независимыми подключами:

$$C = E_{k_{3/3}}(D_{k_{2/3}}(E_{k_{1/3}}(M))),$$

где  $k_{1/3}$ ,  $k_{2/3}$  и  $k_{3/3}$  — три обычных 56-битных ключа DES, в совокупности представляющие собой 168-битный ключ алгоритма Triple DES.

Известный криптолог Ларс Кнудсен предложил схему трехключевого тройного DES со 112-битным ключом шифрования [135, 209], которая называется

ТЕМК (Triple Encryption with Minimum Key, тройное шифрование с минимальным ключом).

В данной схеме 112-битный ключ шифрования ( $K_{1/2}, K_{2/2}$ ) расширяется в три подключа Triple DES следующим образом:

$$\begin{aligned}k_{1/3} &= E_{K_{1/2}}(D_{K_{2/2}}(E_{K_{1/2}}(C_1))); \\k_{2/3} &= E_{K_{1/2}}(D_{K_{2/2}}(E_{K_{1/2}}(C_2))); \\k_{3/3} &= E_{K_{1/2}}(D_{K_{2/2}}(E_{K_{1/2}}(C_3))),\end{aligned}$$

где  $C_1$ ,  $C_2$  и  $C_3$  — несекретные константы, используемые для расширения ключа.

Однако наиболее широкое распространение получила схема Triple DES с тремя независимыми подключаами. Ключ данного варианта Triple DES может быть раскрыт атакой на основе связанных ключей и выбранных открытых текстов, требующей выполнения от  $2^{56}$  до  $2^{72}$  операций шифрования алгоритмом DES [197]. Данная атака весьма сложна в применении, поэтому она не повлияла на распространенность алгоритма. Что интересно, на двухключевой Triple DES данная атака не распространяется, авторы [197] отместили, что это единственный случай, когда двухключевой Triple DES оказался сильнее трехключевого.

## Режимы работы Double DES и Triple DES

Аналогично «одинарному» алгоритму DES, Double DES и Triple DES могут быть использованы в различных режимах работы (см. разд. 1.4): ECB, CBC, CFB и OFB. Однако, например, Triple DES выполняет три раздельных шифрования каждого блока, поэтому различные связи между шифруемыми блоками данных (например, действие  $C_i = E_k(M_i \oplus C_{i-1})$  в режиме CBC) могут быть установлены после выполнения каждого элементарного шифрования обычным алгоритмом DES (это называется *внутренним* режимом), а не после завершения выполнения всех трех шифрований (для отличия от внутренних режимов классические режимы работы в применении к Double и Triple DES часто называют *внешними*) [28, 54]. Сказанное легко пояснить на примере рис. 3.61, на котором слева изображен внешний режим CBC, а справа — внутренний CBC|CBC|CBC [28]. Разница очевидна.

Вариантов комбинирования режимов работы для получения различных внутренних режимов Double и Triple DES существует великое множество, например, в [54] Эли Бихам проанализировал криптостойкость 42 «двойных» и 258 «тройных» внутренних режимов работы. Однако в этой и ряде других работ [53, 58, 175] доказано, что наибольшей криптостойкостью обладают внешние режимы работы, поэтому внутренние режимы использовать не рекомендуется.

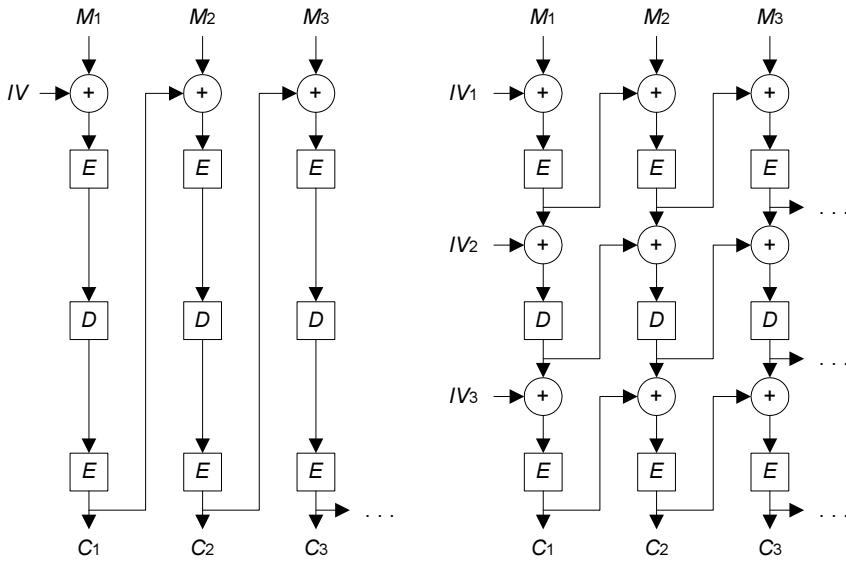


Рис. 3.61. Внешние и внутренние режимы работы

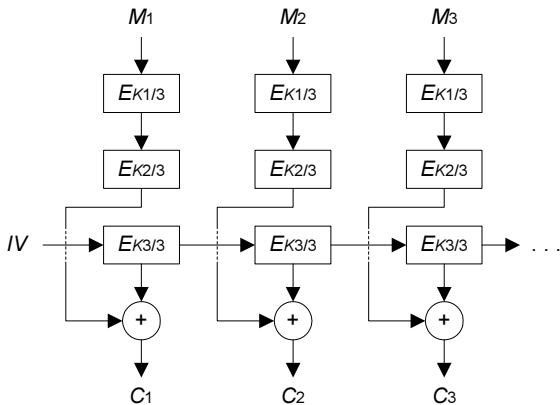


Рис. 3.62. Режим работы Triple DES ECB|ECB|OFB

В качестве примера можно привести атаку на внутренний режим 3-key Triple DES ECB|ECB|OFB (рис. 3.62). Получить 168-битный ключ атакующий может следующим образом [54]:

- Поскольку режим OFB скрывает информацию, необходимую для проведения атаки «встречка посередине» на первые два зашифровывания в ре-

жиме ECB, необходимо атаковать режим OFB. Для этого выбирается некий случайный блок  $A$ , после чего выполняется зашифровывание текста из  $2^{64}$  последовательных блоков  $A$ . Поскольку период псевдослучайной последовательности, генерируемой алгоритмом DES в режиме OFB, не может превышать  $2^{64}$ , шифрование  $2^{64}$  одинаковых блоков даст значение периода этой последовательности, поскольку оно в данном случае эквивалентно периоду шифртекста.

- Обозначим псевдослучайную последовательность, получаемую режимом OFB, как  $v[0], v[1], \dots, v[2^{64} - 1]$ , а результат двойного зашифровывания режимом ECB как  $A' = E_{k_{2/3}}(E_{k_{1/2}}(A))$ . В этом случае шифртекст представляется собой последовательность блоков, имеющих следующие значения:

$$v[0] \oplus A', v[1] \oplus A', \dots, v[2^{64} - 1] \oplus A'.$$

Атакующий рассчитывает и записывает в таблицу следующие значения:

$$v[0] \oplus v[1], v[1] \oplus v[2], \dots, v[2^{64} - 2] \oplus v[2^{64} - 1],$$

которые можно получить из шифртекста, поскольку:

$$(v[0] \oplus A') \oplus (v[1] \oplus A') = v[0] \oplus v[1].$$

- Затем криptoаналитик выбирает случайным образом одно из значений  $v[i]$ , которое обозначается как  $u_0$ . Для каждого возможного ключа  $k_{3/3}$  ( $2^{56}$  вариантов) выполняются следующие действия:

- $u_0$  дважды зашифровывается:

$$u_1 = E_{k_x}(u_0);$$

$$u_2 = E_{k_x}(u_1);$$

- рассчитываются значения  $u_0 \oplus u_1$  и  $u_1 \oplus u_2$ ;
- осуществляется поиск данных значений в таблице, сформированной на шаге 2; если найден верный ключ (т. е.  $k_{3/3} = k_x$ ), то значения  $u_0 \oplus u_1$  и  $u_1 \oplus u_2$  должны найтись в качестве последовательных значений — в этом случае криptoаналитик заканчивает поиск  $k_{3/3}$  и переходит к шагу 5.

- Если перебор ключей на шаге 3 не привел к желаемому результату, выбирается другое значение  $u_0$  и шаг 3 повторяется снова. Согласно [54], для выбора подходящего значения  $u_0$  достаточно нескольких попыток.

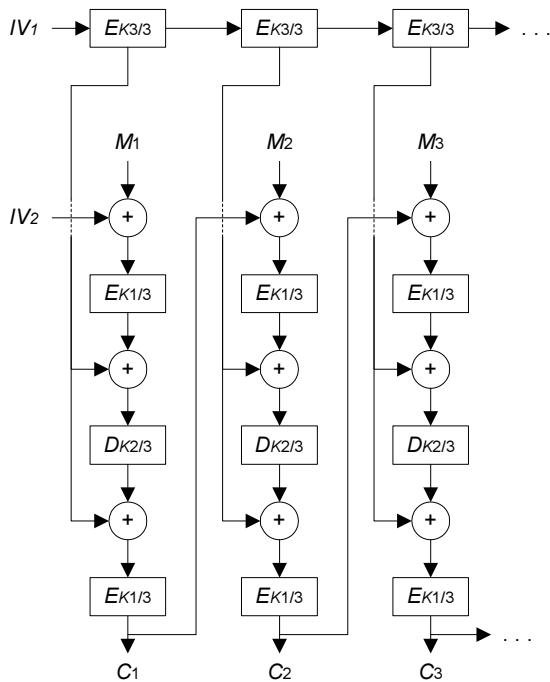
- После нахождения верного ключа  $k_{3/3}$  элементарно вычисляется  $A'$ . В результате криptoаналитик отбрасывает OFB-зашифровывание и имеет двойное шифрование в режиме ECB, а также пару  $(A, A')$ , т. е. пару «открытый текст — шифртекст» для двойного шифрования. Имея эти данные, криptoаналитик осуществляет поиск  $k_{1/3}$  и  $k_{2/3}$  описанной выше атакой

«встреча посередине» для Double DES. Для абсолютно точного вычисления  $k_{1/3}$  и  $k_{2/3}$  атакующий может сформировать еще несколько пар «открытый текст — шифртекст» путем добавления нескольких блоков с различными значениями после  $2^{64}$  блоков со значением  $A$  и их зашифровывания на шаге 1.

Как видно, данная атака требует порядка  $2^{64}$  вычислений, что несравненно меньше сложности полного перебора тройного ключа Triple DES.

## Режимы работы с маскированием

Существуют и более интересные режимы работы алгоритма Triple DES. Например, группа ученых из IBM предложила режим CBCM — внешний CBC-режим с OFB-маскированием (рис. 3.63) [120].



## Описание алгоритмов

на результаты одного из зашифровываний и расшифровывания двухключевого Triple DES в варианте EDE. Третий подключ этого режима используется для получения маски. Итак, режим CBCM можно сформулировать следующим образом:

$$\begin{aligned} Mask_n &= E_{k_{3/3}}(Mask_{n-1}); \\ C_n &= E_{k_{1/3}}(Mask_n \oplus D_{k_{2/3}}(Mask_n \oplus E_{k_{1/3}}(C_{n-1} \oplus M_n))), \end{aligned}$$

где:

- $Mask_n$  — маска, используемая для зашифровывания  $n$ -го блока данных;
- в качестве начальных значений  $Mask_0$  и  $C_0$  используются два независимых вектора инициализации:  $IV_1$  и  $IV_2$ .

Маскирующая последовательность не зависит ни от открытого текста, ни от шифртекста, поэтому маски могут быть вычислены предварительно и использоваться по мере необходимости.

В 1998 г. Эли Бихам и Ларс Кнудсен опубликовали работу [74], в которой сформулированы две атаки на Triple DES в режиме CBCM:

- атака с выбранным открытым текстом, использующая  $2^{65}$  пар блоков и требующая порядка  $2^{58}$  вычислений для нахождения ключа шифрования;
- атака, позволяющая при известном векторе инициализации и наличии известного блока открытого текста и результатов его зашифровывания на  $2^{33}$  связанных ключах вычислить  $k_{3/3}$  выполнением не более  $2^{57}$  операций зашифрования.

Там же в [74] предложено несколько вариантов усиления режима CBCM для противостояния приведенным атакам, простейший из которых — использование четвертого подключа  $k_{4/4}$  вместо повторного использования  $k_{1/3}$  в третьем зашифровывании.

Режимы работы с маскированием существуют и для однократного и двукратного алгоритма DES, например, известный алгоритм Мэта Блейза представляет собой однократный DES в режиме ECB, на который, аналогично режиму CBCM, накладывается OFB-маска [93, 120] (рис. 3.64).

Зашифровывание в алгоритме Мэта Блейза выполняется следующим образом:

$$\begin{aligned} Mask_n &= E_{k_{1/2}}(Mask_{n-1}); \\ C_n &= E_{k_{2/2}}(Mask_n \oplus M_n), \end{aligned}$$

где в качестве начального значения  $Mask_0$  используется вектор инициализации.

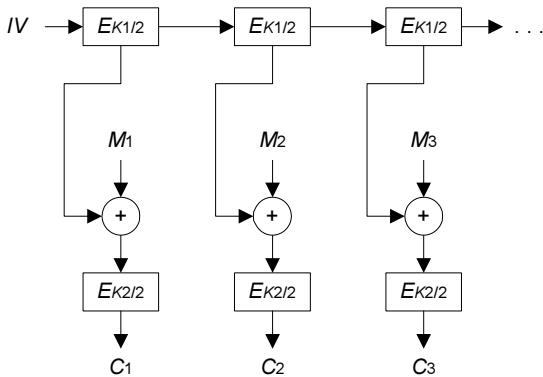


Рис. 3.64. Алгоритм Мэта Блейза

Алгоритм использует 112-битный ключ шифрования, половина которого используется для вычисления маскирующей последовательности.

Автор этого алгоритма посчитал, что, помимо решения проблемы короткого ключа DES, его алгоритм имеет ряд преимуществ, благодаря которым алгоритм идеально подходит для прозрачного шифрования файлов в операционных системах семейства UNIX. В своей работе [93] Мэт Блейз дает множество рекомендаций по построению криптографически защищенной файловой системы CFS (Cryptographic File System) на основе его алгоритма. Однако в [120] была опубликована атака на этот алгоритм, требующая наличия всего двух блоков открытого текста и соответствующих им шифртекстов, причем при их зашифровывании использовалось одно и то же значение  $IV$ . При наличии этих данных достаточно выполнения  $3 * 2^{56}$  операций для вычисления ключа шифрования.

В [120] упоминается алгоритм с аналогичным наложением масок на Double DES — алгоритм Джонса (Jones):

$$Mask_n = E_{k_{1/3}}(Mask_{n-1});$$

$$C_n = D_{k_{3/3}}(Mask_n \oplus E_{k_{2/3}}(M_n)).$$

Алгоритм Джонса в 1,5 раза медленнее, чем алгоритм Блейза, но так же легко вскрываем — 168-битный ключ этого алгоритма может быть получен выполнением  $2^{58}$  шифрований при наличии тех же исходных данных, которые требуются для вскрытия алгоритма Блейза [120].

Различные внутренние режимы работы и режимы с маскированием могут быть применены не только в вариантах многократного DES, но и в любых других алгоритмах.

## Алгоритм Quadruple DES

В ряде источников упоминается об алгоритме QDES (Quadruple DES, «четверной» DES), например, в [54] и [55]. Однако алгоритм QDES является очень медленным, поэтому неизвестны случаи широкого применения данного алгоритма.

## Алгоритм Ladder-DES

В 1994 г. Терри Риттер (Terry Ritter) предложил алгоритм Four-Rung Ladder-DES (лестничный DES с четырьмя «ступеньками») [325]. Структура алгоритма приведена на рис. 3.65. Фактически алгоритм выполняет 4 раунда преобразований, в каждом из которых выполняется однократное шифрование половины шифруемого блока данных обычным алгоритмом DES на независимых ключах раундов. Таким образом, размер блока алгоритма Ladder-DES равен 128 битам, а используемый размер ключа шифрования — 224 битам, т. е. четырехкратное увеличение относительно ключа алгоритма DES.

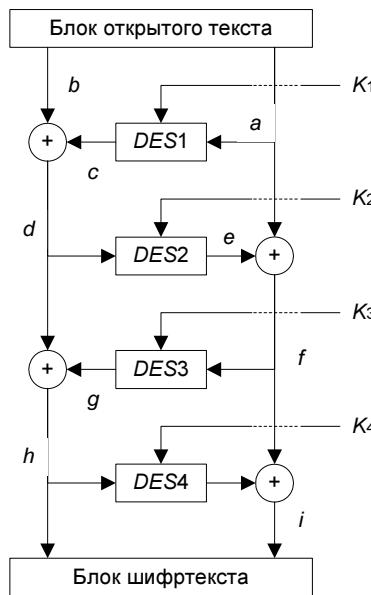


Рис. 3.65. Алгоритм Ladder-DES

Этот алгоритм продолжил рассмотренную ранее концепцию увеличения размера ключа за счет многократного шифрования. Причем Ladder-DES явно выигрывает в скорости у Triple DES, поскольку на двойной размер блока

данных относительно Triple DES приходится 4 шифрования вместо трех (т. е. скорость Ladder-DES эквивалентна скорости Double DES). При этом, аналогично Double DES и Triple DES, в Ladder-DES может быть использована аппаратура или программное обеспечение, реализующее классический DES.

Серьезная уязвимость в Ladder-DES была найдена в 1997 г. [52]. Для вычисления ключа данного алгоритма криптоаналитику необходимо сгенерировать порядка  $2^{36}$  блоков открытого текста, которые имеют следующий вид: 64-битная правая половина блока (переменная  $a$  на рис. 3.65) является константой, а левые половины ( $b$ ) для каждого из генерируемых блоков различны. Дальнейшая логика атаки такова:

- поскольку  $a$  и  $K_1$  являются фиксированными величинами, результат операции  $DES1$  также является константой для каждого из  $2^{36}$  текстов;
- значения  $b$  различны для каждого текста, а  $c$  — константа, поэтому значения  $d = b \oplus c$  также не повторяются в пределах выбранных  $2^{36}$  текстов;
- поскольку зашифровывание операцией  $DES2$  различных значений  $d$  выполняется на фиксированных ключах  $K_2$ , значения  $e$  являются различными;
- значения  $f$  являются различными по той же причине, что и значения  $b$ , а различия в значениях  $g$  объясняются аналогично различиям в значениях  $e$ ;
- поскольку  $i = DES(K_4, h) \oplus f$ , то  $f = DES(K_4, h) \oplus i$ , а пара  $(h, i)$  представляет собой блок шифртекста.

Имея в виду все вышесказанное, криптоаналитик может выполнить следующие действия для каждого возможного значения ключа  $K_4$  (цикл по  $n_1$  от 0 до  $2^{56} - 1$ ):

1. Для каждого из имеющихся в наличии шифртекстов (цикл по  $n_2$  от 1 до  $2^{36}$ ):

- вычисляется и записывается в некую временную таблицу значение:

$$f_{n1,n2} = DES(K_{n1}, h_{n2}) \oplus i_{n2};$$

- если вычисленное значение совпадает с каким-либо из предыдущих значений  $f$ , вычисленных для того же ключа  $K_{n1}$ , то данный ключ считается неверным (поскольку возникает противоречие описанной выше логике), поэтому осуществляется переход к следующему значению  $K_4$ , т. е.  $K_{n1+1}$ .

2. Если выяснилось, что для всех шифртекстов не обнаружилось совпадений  $f$ , то считается, что обнаружился правильный ключ, т. е.:

$$K_4 = K_{n1}.$$

Аналогичным образом после нахождения  $K_4$  вычисляется  $K_3$ , а  $K_2$  и  $K_1$  определяются полным перебором по  $2^{56}$  вариантов.

В среднем, для обнаружения коллизии для каждого из ключей достаточно проверки  $2^{32}$  вариантов.  $2^{36}$  пар текстов не дают абсолютной гарантии нахождения верного ключа, но вероятность ошибки пренебрежимо мала и составляет порядка  $2^{-129}$ . Таким образом, сложность вычисления ключа составляет порядка  $2^{88}$  ( $2^{56} * 2^{32}$ ) операций, что несравненно быстрее полного перебора  $2^{224}$  ключей.

В той же работе [325] был предложен «двуократный четырехступенчатый» Ladder-DES (2x Four-Rung Ladder-DES), его структура приведена на рис. 3.66. Данный алгоритм имеет двукратный размер блока по сравнению с описанным выше вариантом Ladder-DES. Двойные блоки данного алгоритма шифруются аналогично Ladder-DES на первых двух «ступеньках», затем происходит циклический сдвиг субблоков на один вправо, после чего снова выполняются две ступеньки обычного Ladder-DES.

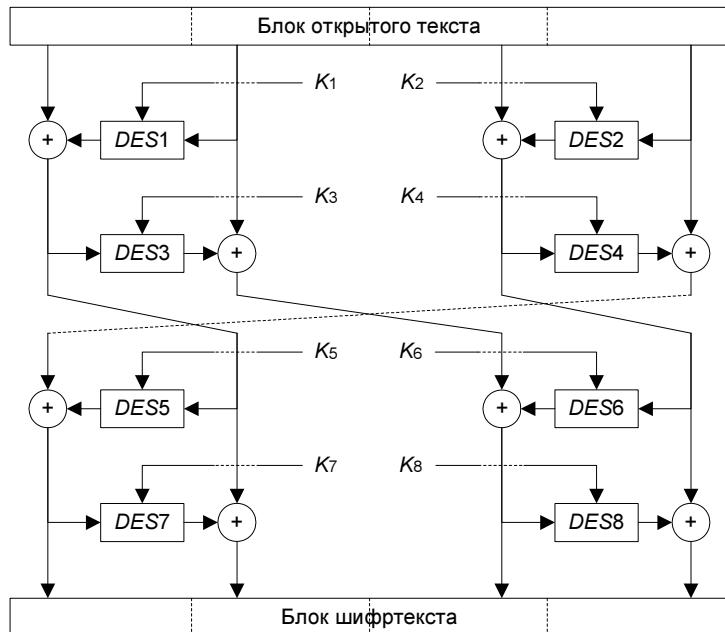


Рис. 3.66. «Двуократный» Ladder-DES

Как видно из схемы, ключ данного алгоритма имеет размер, в 8 раз превышающий размер ключа DES, однако автор алгоритма предложил следующую схему использования 224-битного ключа вместо 448-битного: нечетные ключи

раундов являются основными, четные вычисляются из них согласно следующим простым формулам:

$$K_2 = K_1 \oplus K_3;$$

$$K_4 = K_3 \oplus K_5;$$

$$K_6 = K_5 \oplus K_7;$$

$$K_8 = K_7 \oplus K_1.$$

Данный вариант имеет абсолютно те же скоростные характеристики, что и «однократный» Ladder-DES; он подвержен той же, описанной выше, атаке. И так же, как «классический» Ladder-DES, 2x Four-Rung Ladder-DES не нашел широкого применения.

## Алгоритм DESX

Алгоритм DESX предложен известным криптологом Рональдом Ривестом (Ronald Rivest) из компании RSA в 1984 г., а сформулирован только в 1996 г. Джо Килианом (Joe Kilian) и Филиппом Рогаузем (Phillip Rogaway) в работах [201] и [331]. Суть алгоритма состоит в том, что перед выполнением однократного DES и после него на данные операцией XOR накладываются различные 64-битные фрагменты ключа (рис. 3.67):

$$C = k_3 \oplus DES_{k_1}(k_2 \oplus M).$$

Операция наложения фрагмента ключа на вход и/или выход алгоритма шифрования называется *отбелыванием*, смысл которого состоит в том, чтобы помешать криptoаналитику получить для исследования алгоритма пары «открытый текст — шифртекст» [28].

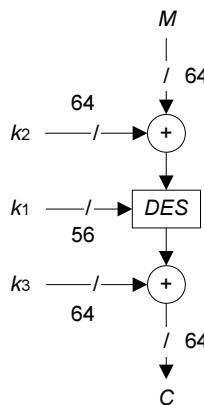


Рис. 3.67. Алгоритм DESX

DESX полностью совместим с алгоритмом DES в случае, если  $k_2 = k_3 = 0$ .

Стоит обратить внимание на нестандартный размер ключа шифрования DESX:  $k_2$  и  $k_3$  имеют размер по 64 бита, а  $k_1$  — это обычный 56-битный ключ DES, таким образом, полный размер ключа составляет 184 бита. Однако авторы алгоритма в [201] дают ряд несложных рекомендаций, позволяющих использовать переменный размер ключа шифрования, предварительно применив к нему хэширование (см. разд. 1.1) алгоритмом SHA-1. Кроме того, авторы алгоритма допускают использование 120-битного ключа шифрования ( $k_1, k_2$ ), при этом  $k_3$  устанавливается равным  $k_2$ . Существует вариант DESX, также разработанный фирмой RSA [331], со 120-битным ключом, состоящим из  $k_1$  и  $k_2$ , а значение  $k_3$  является функцией всех 16 байтов ключа шифрования. Известен и еще один вариант алгоритма DESX, в котором вместо обеих операций XOR выполняется сложение по модулю  $2^{64}$  [200].

Один из авторов DESX назвал этот алгоритм «DES, который лучше, чем DES» [331]. Однако в алгоритме DESX было найдено несколько уязвимостей, воспрепятствовавших широкому распространению этого алгоритма [200]:

- каждый ключ DESX имеет эквивалентный ключ, подключи которого комплементарны подключам исходного ключа;
- существует атака, позволяющая раскрыть ключ алгоритма; атака основана на связанных ключах и при наличии  $2^n$  выбранных открытых текстов требует выполнения  $2^{120-n}$  шифрований;
- дифференциальный криптоанализ позволяет вскрыть алгоритм при наличии  $2^{61}$  выбранных открытых текстов; линейный криптоанализ успешен при наличии  $2^{60}$  известных открытых текстов.

Описанные выше проблемы не проявляются только в варианте DESX, использующем алгоритм хэширования SHA-1 для расширения ключа алгоритма [200].

## Алгоритм DES с независимыми ключами раундов

Следующая идея по увеличению ключа шифрования алгоритма DES является наиболее логичной и понятной: DES использует 16 ключей раундов по 48 битов — почему бы не сделать их независимыми и не использовать такой 768-битный ключ ( $768 = 48 * 16$ ) вместо 56-битного ключа? Кстати, по похожим принципам ключи используют немало алгоритмов шифрования; в качестве примера можно привести отечественный стандарт шифрования ГОСТ 28147-89, в котором отсутствует процедура расширения ключа, а 256-битный ключ алгоритма просто разбивается на 32-битные фрагменты, каждый из которых участвует в нескольких раундах алгоритма (см. разд. 3.1).

Достоинства такого варианта DES очевидны: такой ключ шифрования не только не уменьшит быстродействие алгоритма по сравнению с обычным алгоритмом DES, но даже и увеличит ее за счет отсутствия процедуры расширения ключа (это важно для применений, в которых предусмотрена частая смена ключа шифрования).

Недостатки же таковы: во-первых, большой размер ключа требует несравнительно больше места для хранения и передачи (что становится с течением времени все менее существенным), во-вторых, такой алгоритм оказался уязвим к атаке «встреча посередине», поэтому для полного перебора ключа требуется не  $2^{768}$ , а  $2^{384}$  попыток, что, однако, не более реально [28].

Более серьезную уязвимость обнаружили Бихам и Шамир — существует возможность вычисления 768-битного ключа методом дифференциального криптоанализа при наличии  $2^{59}$  выбранных пар «открытый текст — шифртекст» после выполнения  $2^{61}$  операций шифрования [75]. Авторы атаки посчитали ее непрактичной из-за необходимости в наличии огромного количества выбранных открытых текстов, однако, видно, что DES с независимыми ключами раундов не намного более стоек к дифференциальному криптоанализу, чем классический DES. В [53] и [330] упоминается также атака на 768-битный DES методом линейного криптоанализа, для которой требуется порядка  $2^{60}$  известных открытых текстов и соответствующих им шифртекстов.

Кроме того, в [197] описана атака на связанных ключах, позволяющая полностью раскрыть ключ алгоритма при наличии 15 связанных ключей и 60 выбранных пар «открытый текст — шифртекст»; другой вариант данной атаки позволяет вычислить ключ шифрования, используя всего один связанный с ним ключ, но порядка  $2^{16}$  выбранных открытых текстов.

Также можно предположить и еще один недостаток у данного варианта DES — невозможность использования существующих реализаций классического алгоритма DES, поскольку подавляющее большинство программных и аппаратных средств шифрования сами выполняют расширение ключа, а не принимают уже расширенный ключ в качестве параметра. Поэтому думаю, что подавляющее большинство средств шифрования алгоритмом DES нуждалось бы в серьезной модификации для шифрования с их помощью алгоритмом DES с независимыми ключами раундов.

## Алгоритм GDES

Алгоритм GDES (Generalized DES, обобщенный DES) появился в результате ряда попыток создать более универсальный алгоритм с переменным количеством раундов и переменным размером шифруемого блока данных. GDES подробно описан в [28] и [75].

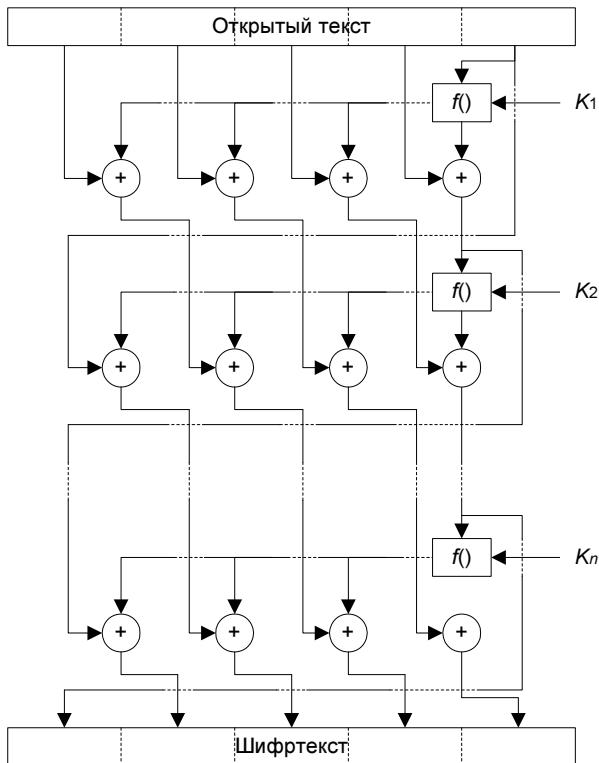


Рис. 3.68. Алгоритм GDES

GDES позволяет шифровать блоки любого размера, кратного 32 битам. На рис. 3.68 показан пример  $n$ -раундового шифрования данным алгоритмом блоков размером  $5 * 32$  битов. В каждом  $i$ -м раунде выполняются следующие действия:

1. Крайний правый 32-битный субблок обрабатывается функцией  $f()$  — функцией раунда DES с использованием ключа раунда  $K_i$ .
2. Результат предыдущего шага накладывается на остальные субблоки операцией XOR.
3. Выполняется поблочный циклический сдвиг вправо на 1 субблок.

Стоит отметить, что при размере блока 64 бита и 16 раундах GDES полностью эквивалентен классическому алгоритму DES. Автором алгоритма рекомендовано 16-раундовое шифрование блоков по  $8 * 32$  битов. В любом случае, рекомендуется шифрование блоков, состоящих из четного числа субблоков [75].

Процедура расширения ключа эквивалентна таковой в алгоритме DES. Существует также вариант GDES с независимыми ключами раундов [28].

Считается, что GDES может шифровать данные с существенно большей скоростью, чем DES, при больших размерах блоков, поскольку на блок данных сколь угодно большого размера функция  $f()$  выполняется лишь однократно. Однако для достаточной криптостойкости при большом размере блока необходимо увеличить количество раундов алгоритма. Этот вывод можно сделать благодаря существованию атак на GDES [75], в частности:

- дифференциальный криптоанализ 16-раундового алгоритма GDES с размером блока  $8 * 32$  битов позволяет вычислить ключ при наличии всего 6 выбранных открытых текстов и соответствующих им шифртекстов, а для 8 раундов при том же размере блока достаточно всего трех известных открытых текстов;
- для полного вскрытия GDES, аналогичного предыдущему, но с независимыми ключами раундов, достаточно 16 выбранных открытых текстов.

В [75] описаны различные виды атак с использованием как выбранных, так и известных открытых текстов, применяемых в зависимости от соотношений количества раундов алгоритма и количества субблоков в шифруемом блоке данных. Вывод авторов данного исследования таков: при количестве раундов, меньшем восьмикратного количества субблоков, алгоритм GDES относительно легко вскрывается, поэтому данное соотношение рекомендуется как минимально возможное (в [28] также утверждается, что GDES с 8-ю субблоками в блоке и  $n = 64$  слабее, чем DES). Таким образом, авторы утверждают, что «в общем, любой вариант GDES, более быстрый, чем DES, также и слабее, чем DES».

Еще один вариант GDES предложен специалистами Кембриджского университета в 1994 г. [391]. Однако Шнайер считает этот вариант не более надежным, чем описанный выше [28].

## Алгоритм RDES

Алгоритм RDES (Randomized DES, DES с элементами случайности) предложен группой японских специалистов в 1993 г. Смысл внесенной в DES случайности состоит в том, что перемена субблоков местами после каждого раунда алгоритма DES выполняется не всегда, а только в зависимости от значения определенных битов ключа. Данные изменения должны были усилить алгоритм против дифференциального криптоанализа [48].

## Описание алгоритмов

Однако практически сразу после этого Ишаи Бен-Ароя (Ishai Ben-Aroya) и Эли Бихам опубликовали работу [48], в которой описали множество недостатков алгоритма RDES.

- Один ключ из  $2^{15}$  ключей приводит к полному отсутствию перестановок субблоков. Последствия этого ужасны: правая половина 64-битного блока шифртекста (не считая начальной и конечной перестановок) полностью эквивалентна правой половине открытого текста, т. е. половина открытого текста появляется в шифртексте в неизменном виде! Эта ситуация показана на рис. 3.69.

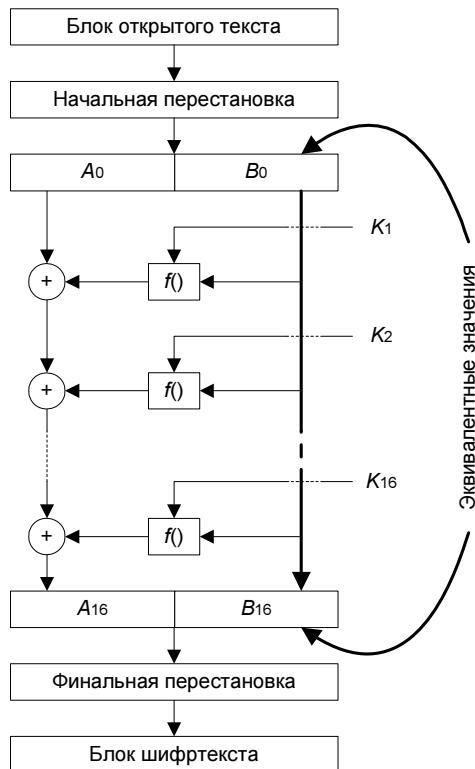


Рис. 3.69. Алгоритм RDES со слабым ключом

- Еще один ключ из  $2^{15}$  ключей приводит к тому, что перемена субблоков выполняется лишь один раз — такой алгоритм также не является проблемой для криптоаналитика.
- Подавляющее большинство ключей приводит к различного рода возможностям для вскрытия алгоритма. Бен-Ароя и Бихам оценивают количество

беспроblemных ключей всего в 2 % (!) от возможного ключевого пространства.

Вывод экспертов таков: если использовать только сильные ключи, алгоритм не является более стойким, чем DES, а остальные ключи делают алгоритм существенно более слабым.

Шнайер в [28] упоминает о последующих усилениях алгоритма RDES, новые варианты называются RDES-1...RDES-4. Алгоритм RDES-2 и его последующие варианты оцениваются в [28] как достаточно надежные.

## Алгоритмы $s^2$ DES, $s^3$ DES и $s^5$ DES

Алгоритм  $s^2$ DES был предложен корейским специалистом Кванджо Кимом (Kwangjo Kim) в 1991 г. [204]. Его целью была разработка на базе DES алгоритма, более устойчивого к дифференциальному криптоанализу, чем DES. Автор  $s^2$ DES предложил ряд дополнительных критериев создания таблиц замен и сгенерировал по этим критериям новые таблицы замен  $S_1...S_8$  (по его мнению, более оптимальные, чем в DES), в результате появился  $s^2$ DES — алгоритм, отличающийся от DES только значениями таблиц замен. Таблицы замен алгоритма  $s^2$ DES и последующих алгоритмов данного семейства приведены в *Приложении 1*.

В следующем году Ларс Кнудсен представил атаку на  $s^2$ DES методом дифференциального криптоанализа на основе порядка  $2^{42}$  выбранных открытых текстов и соответствующих им шифртекстов, которая доказывала, что классический DES более стоек к дифференциальному анализу, чем  $s^2$ DES [212]. Поэтому в 1993 г. Ким и его коллеги представили новый алгоритм —  $s^3$ DES [207], существенно более устойчивый к дифференциальному криптоанализу.  $s^3$ DES также отличался от DES и  $s^2$ DES только значениями таблиц замен, однако дифференциальный криптоанализ алгоритма  $s^3$ DES действительно оказался сложнее полного перебора ключей, поэтому авторы назвали таблицы замен алгоритма  $s^3$ DES «DA-immune», т. е. невосприимчивыми к дифференциальному анализу.

Дальнейшие исследования алгоритмов данного семейства выявили следующее соотношение устойчивости алгоритмов к линейному криптоанализу:

$$s^3\text{DES} < \text{DES} < s^2\text{DES},$$

хотя по отношению к дифференциальному криптоанализу все было ровно наоборот [205]:

$$s^2\text{DES} < \text{DES} < s^3\text{DES}.$$

Кстати, в [60] утверждается, что наиболее сильные таблицы замен для DES — это таблицы алгоритма  $s^3$ DES, но с переставленными местами  $S_1$  и  $S_2$ .

$s^3$ DES с переставленными  $S_1$  и  $S_2$  примерно в  $2^{20}$  раз более стоек к дифференциальному криптоанализу, чем классический DES.

В результате дальнейших разработок в 1995 г. был представлен алгоритм  $s^5$ DES (название  $s^4$ DES было закреплено за промежуточной версией алгоритма [206]), также отличающийся от предыдущих только измененными таблицами замен [205, 206]. По устойчивости к дифференциальному криптоанализу  $s^5$ DES был сравним с  $s^3$ DES, а к линейному криптоанализу — с  $s^2$ DES, т. е.  $s^5$ DES совместил в себе лучшие свойства предыдущих алгоритмов семейства.  $s^5$ DES был признан экспертами как весьма сильный алгоритм, например в [123] сказано, что « $s^5$ DES — очень хороший шифр по сравнению с DES».

Стоит отметить, что  $s^5$ DES не решал основную проблему DES — проблему короткого ключа, поэтому он не мог стать полноценной заменой алгоритму DES.

## Алгоритм Biham-DES

Криптолог, несомненно, добившийся наиболее значимых и известных результатов в деле криптоанализа DES и его вариантов, Эли Бихам, совместно с Алексом Бирюковым предложил свой вариант алгоритма DES, известный как Biham-DES.

В [60] Бихам и Бирюков утверждают, что большинство реализаций алгоритма DES, в том числе аппаратных, позволяют загружать значения таблиц замен. Согласно многим работам Эли Бихама и других исследователей, таблицы замен являются наиболее значащей частью преобразований, выполняемых алгоритмом DES. Поэтому эксперты продолжили анализ таблиц замен.

Для начала они рассмотрели вариант алгоритма DES, в котором таблицы замен зависят от определенного фрагмента ключа шифрования следующим образом: перед использованием таблицы выполняется наложение одного подключа на ее входное значение операцией XOR, а затем — наложение другого подключа на выходное значение (рис. 3.70). Однако у данного алгоритма оказалось множество недостатков [60]:

- каждый ключ алгоритма имеет по 3 эквивалентных ключа;
- существует много слабых и полуслабых ключей;
- фактически такая модификация алгоритма не является усилением против криптоанализа, а только увеличивает размер ключа.

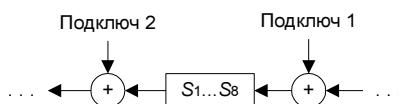


Рис. 3.70. Пример использования дополнительных подключей

В результате дальнейших усилий появился алгоритм Biham-DES. Данный алгоритм использует 119-битный ключ ( $K_a, K_b, K_c, K_d$ ), подключи которого применяются следующим образом (рис. 3.71):

- 16-битный подключ  $K_a$  определенным образом расширяется до 48 битов и в каждом раунде накладывается операцией XOR на 48-битное входное значение таблиц замен;
- 32-битный подключ  $K_b$  аналогичным образом накладывается на выходное значение таблиц замен;
- 15-битный подключ  $K_c$  определенным образом изменяет порядок таблиц замен, в качестве которых используются таблицы алгоритма s<sup>3</sup>DES с представленными таблицами  $S1$  и  $S2$ ;
- $K_d$  — 56-битный ключ, используемый аналогично обычному алгоритму DES.

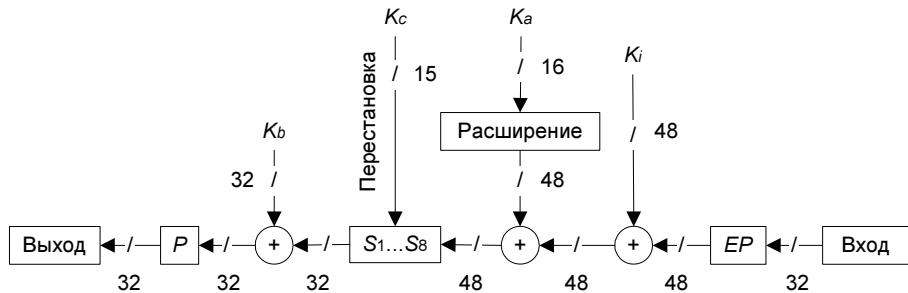


Рис. 3.71. Алгоритм Biham-DES

Существует вариант алгоритма Biham-DES с таблицами замен классического DES; в этом случае подключ  $K_c$  имеет размер 5 битов.

Также авторы предложили вариант алгоритма с таблицами s<sup>3</sup>DES со 104-битным ключом без использования ключа  $K_c$ , т. е. без перестановок таблиц.

Авторы этого алгоритма в [60] дают рекомендации по использованию существующих реализаций алгоритма DES для шифрования алгоритмом Biham-DES (имеются в виду реализации, позволяющие загружать значения таблиц замен). Поскольку в существующие реализации DES невозможно внедрить операции XOR данных с подключами  $K_a$  и  $K_b$ , эти операции эмулируются зависящими от значений подключей перестановками строк и столбцов таблиц замен, а также наложением фрагментов ключа операцией XOR на определенные элементы таблиц замен.

Из всех вариантов алгоритма Biham-DES наиболее стойким является вариант со 119-битным ключом и таблицами замен s<sup>3</sup>DES. Сложность дифференциального и линейного криптоанализа такого алгоритма составляет, соответственно,  $2^{66}$  и  $2^{64}$  операций, т. е. стойкость алгоритма существенно выше стойкости DES, но криптоанализ требует несравненно меньших затрат, чем полный перебор ключа [60]. Кроме того, все варианты алгоритма Biham-DES оказались относительно слабыми по отношению к атаке на связанных ключах — вариант с таблицами замен DES (109-битный ключ) вскрывается при наличии  $2^{31}$  выбранных открытых текстов и соответствующих им шифртекстов, зашифрованных на 32 связанных ключах [200].

## Алгоритмы xDES<sup>1</sup> и xDES<sup>2</sup>

Семейство алгоритмов xDES<sup>i</sup> было предложено группой японских специалистов и описано в [408]. Фактически это еще одно обобщение алгоритма DES — семейство алгоритмов, использующих обычный DES в качестве функции раунда. Обычный DES получил обозначение xDES<sup>0</sup>, а индекс  $i$  (при  $i > 0$ ) в названии алгоритмов xDES<sup>i</sup> определяет основные характеристики конкретного алгоритма:

- размер шифруемого блока равен  $128 * i$  битам;
- размер ключа шифрования равен  $56 * i * (2 * i + 1)$  битам;
- количество раундов алгоритма —  $2 * i + 1$ .

Авторы отметили, что вместо DES в качестве основы семейства алгоритмов может быть использован любой блочный шифр.

Таким образом, xDES<sup>1</sup> — трехраундовый алгоритм со 128-битным блоком и 168-битным ключом, в каждом раунде которого один из субблоков шифруется алгоритмом DES на независимом ключе, накладывается операцией XOR на другой субблок, после чего субблоки меняются местами. Алгоритм достаточно быстр — xDES<sup>1</sup> шифрует 128-битный блок трехкратным шифрованием алгоритмом DES, что значительно быстрее, чем шифрование алгоритмом Double DES при том же размере ключа. Однако алгоритм xDES<sup>1</sup> оказался слаб к ряду атак, в частности:

- к атаке «встреча посередине», аналогичной описанной выше для Double DES [28, 209];
- к атаке на основе всего двух пар выбранных открытых текстов и соответствующих им шифртекстов, которая позволяет вычислить ключ выполнением порядка  $2^{58}$  шифрований алгоритмом DES [209].

Соответственно, алгоритм xDES<sup>1</sup> был признан недостаточно сильным для использования вместо DES.

Следующий алгоритм семейства — xDES<sup>2</sup> — выполняет 5 раундов преобразований, в каждом из которых два 64-битных субблока шифруются алгоритмом DES и накладываются на два других субблока операцией XOR. Размер блока алгоритма в четыре раза больше размера блока DES, т. е. 256 битов, размер ключа — 560 битов. Структура алгоритма показана на рис. 3.72.

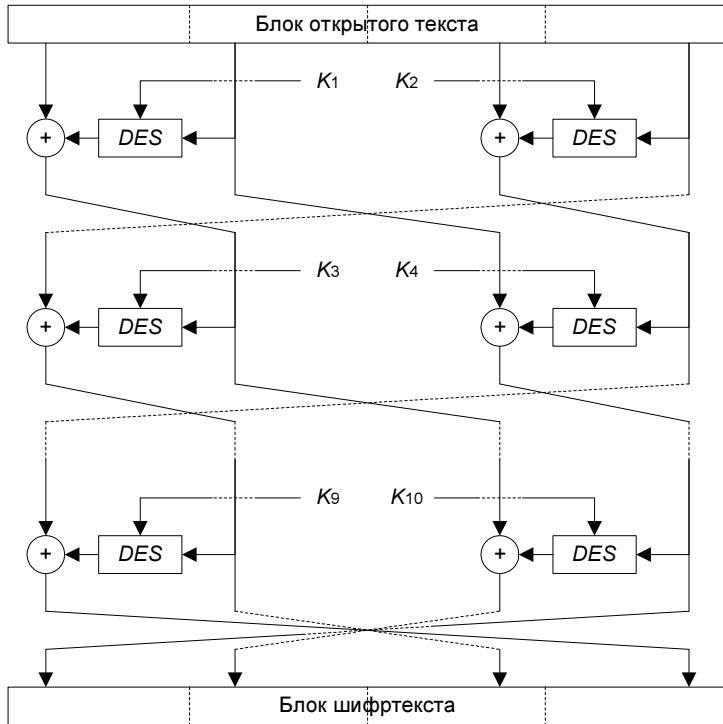


Рис. 3.72. Алгоритм xDES<sup>2</sup>

Данный алгоритм также относительно быстр — для шифрования четырехкратного блока DES xDES<sup>2</sup> выполняет 10 шифрований; xDES<sup>2</sup> также недостаточно стоек — в [209] описана атака на основе  $2^{33}$  известных открытых текстов, раскрывающая ключ алгоритма за  $2^{64}$  DES-шифрования.

xDES<sup>3</sup> и последующие алгоритмы семейства являются несравненно более громоздкими и медленными — xDES<sup>3</sup> уже медленнее, чем Triple DES. Шнайер считает, что xDES<sup>3</sup> невозможно использовать в качестве блочного шифра из-за слишком больших размеров блока и ключа [28].

## Другие варианты алгоритма DES

Существует множество других вариантов алгоритма DES. Подавляющее большинство вариантов основано на различных изменениях таблиц замен, а также на различных схемах расширения ключа, но существуют и другие. Наиболее интересны из них следующие:

- случайно сгенерированные таблицы замен; криптоанализ подавляющего большинства из них (т. е. криптоанализ алгоритмов на основе таких таблиц) требует всего  $2^{18}—2^{20}$  выбранных открытых текстов, что несравненно хуже DES [28];
- таблицы замен, формируемые на основе ключа шифрования; порядка 95 % таких таблиц вскрываются дифференциальным криптоанализом при наличии всего  $2^{26}—2^{29}$  выбранных открытых текстов [60];
- использование операции сложения по модулю  $2^4$  вместо XOR для наложения на левый субблок результата обработки правого субблока; такая модификация оказалась значительно сильнее алгоритма DES — по словам Шнайера, вскрыть ее в  $2^{17}$  раз труднее [28].

Ряд других вариантов с их криптостойкостью приведены в [28]. Все они существенно менее стойкие, чем классический алгоритм DES.

## Заключение

Алгоритм DES успешно применялся во всем мире в качестве основного алгоритма симметричного шифрования более 20 лет. Затем он был заменен на один из своих потомков — Triple DES. А сейчас уже действует новый стандарт шифрования — алгоритм AES (см. разд. 3.3). Интересно, надолго ли?

## 3.16. Алгоритм DFC

Алгоритм DFC (Decorrelated Fast Cipher, быстрый шифр без взаимосвязей) разработан в 1998 г. специально для участия в конкурсе AES несколькими французскими специалистами под руководством известного криптолога Сержа Воденэ [162]. Алгоритм создан благодаря сотрудничеству двух организаций: телекоммуникационного гиганта France Telecom и высшего учебного заведения Ecole Normale Supérieure (ENS).

## Основные характеристики и структура алгоритма

Согласно требованиям конкурса AES, алгоритм шифрует данные блоками по 128 битов. При этом алгоритм DFC использует ключи не только трех фикси-

рованных размеров, предусмотренных конкурсом (128, 192 и 256 битов), но и переменного размера от 0 до 256 битов.

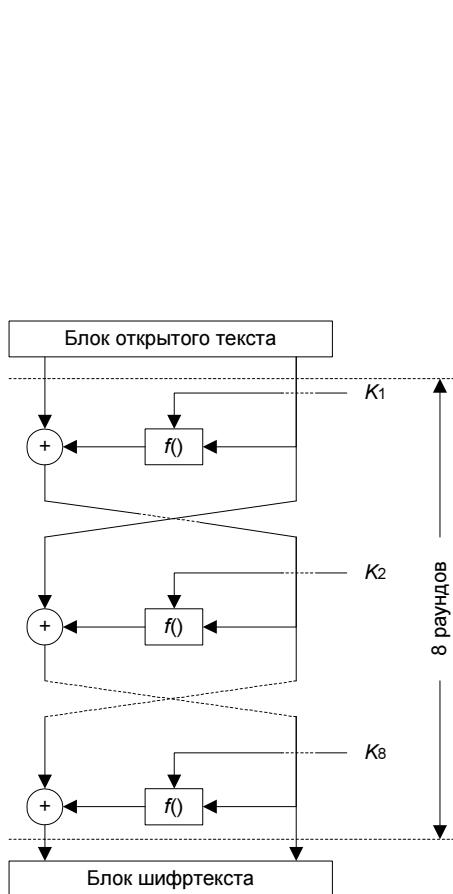


Рис. 3.73. Структура алгоритма DFC

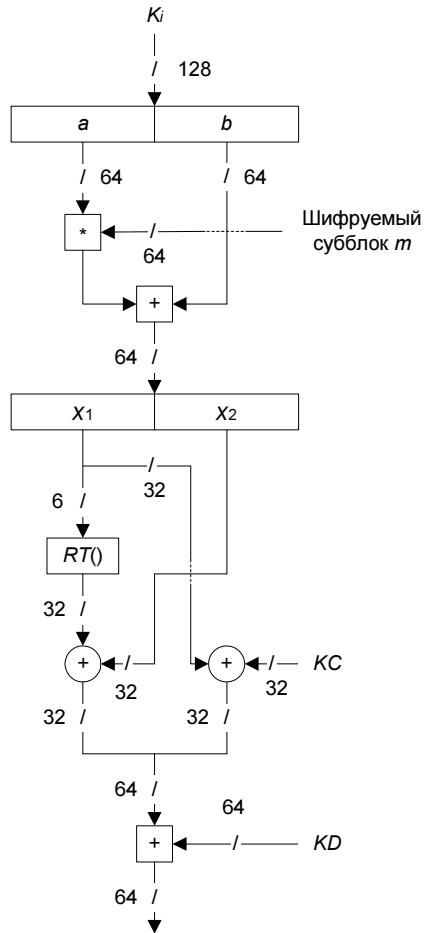


Рис. 3.74. Функция раунда алгоритма DFC

Алгоритм является сетью Фейстеля, в которой выполняется 8 раундов шифрования (рис. 3.73). В каждом раунде производятся следующие действия (см. упрощенную схему на рис. 3.74):

1. Текущее значение 128-битного ключа  $i$ -го раунда  $K_i$  (процедура расширения ключа подробно описана далее) разбивается на 2 64-битных фрагмента:  $a$  и  $b$ .

2. Вычисляется промежуточное значение  $x$  по следующей формуле:

$$x = ((\bar{a} * \bar{m} + \bar{b}) \bmod (2^{64} + 13)) \bmod 2^{64},$$

где:

- $\bar{n}$  — побитовое дополнение к  $n$ ;
- $m$  — значение 64-битного шифруемого субблока.

3. 64-битное значение  $x$  разбивается на 2 32-битных фрагмента  $x_1$  и  $x_2$ .

4. Над результатом предыдущего шага выполняется операция  $CP$  (Confusion Permutation, перемешивающая перестановка), представляющая собой следующее преобразование:

$$y = ((x_2 \oplus RT(\overline{\text{trunc}_6(x_1)})) \bullet (x_1 \oplus KC) + KD) \bmod 2^{64},$$

где:

- $\bullet$  — операция конкатенации;
- $\text{trunc}_n$  — усечение битовой строки до  $n$  левых битов;
- $KC$  и  $KD$  — фиксированные константы (указаны шестнадцатеричные значения):

$$KC = \text{EB64749A};$$

$$KD = \text{86D1BF275B9B241D};$$

- $RT$  — табличная замена, замещающая входное 6-битное значение 32-битным выходным; таблица замен полностью описана в спецификации алгоритма и представлена в табл. 3.22.

**Таблица 3.22**

B7E15162	8AED2A6A	BF715880	9CF4F3C7
62E7160F	38B4DA56	A784D904	5190CFEF
324E7738	926CFBE5	F4BF8D8D	8C31D763
DA06C80A	BB1185EB	4F7C7B57	57F59584
90CFD47D	7C19BB42	158D9554	F7B46BCE
D55C4D79	FD5F24D6	613C31C3	839A2DDF
8A9A276B	CFBFA1C8	77C56284	DAB79CD4
C2B3293D	20E9E5EA	F02AC60A	CC93ED87
4422A52E	CB238FEE	E5AB6ADD	835FD1A0
753D0A8F	78E537D2	B95BB79D	8DCAEC64

Таблица 3.22 (окончание)

2C1E9F23	B829B5C2	780BF387	37DF8BB3
00D01334	A0D0BD86	45CBFA73	A6160FFE
393C48CB	BBCA060F	0FF8EC6D	31BEB5CC
EED7F2F0	BB088017	163BC60D	F45A0ECB
1BCD289B	06CBBFEA	21AD08E1	847F3F73
78D56CED	94640D6E	F0D3D37B	E67008E1

То есть значение 0 меняется на B7E15162, значение 1 замещается значением 8AED2A6A и т. д. Всего в таблице 64 записи — по количеству возможных 6-битных значений ( $2^6$ ). Таблица является псевдослучайной, в качестве ее заполнения авторы использовали шестнадцатеричную запись дробной части математической константы  $e$ .

Расшифровывание выполняется аналогично зашифровыванию, но с обратным порядком использования ключей раундов.

## Процедура расширения ключа

Процедура расширения ключа позволяет выработать из ключа шифрования произвольного размера от 0 до 256 битов 8 128-битных ключей раундов. Данная процедура выполняется в несколько шагов:

1.  $L$ -битный ключ шифрования  $K$  дополняется  $(256 - L)$ -битной частью 256-битной константы  $KS$ , определенной в спецификации алгоритма следующим образом:

$$\begin{aligned} KS = & \text{DA06C80ABB1185EB4F7C7B5757F59584} \bullet \\ & \bullet 90CFD47D7C19BB42158D9554F7B46BCE. \end{aligned}$$

Как видно, результирующая последовательность (обозначаемая как  $PK$ ) имеет размер 256 битов.

2. Вычисляются следующие 64-битные промежуточные величины:

$$OA_1 = PK_1 \bullet PK_8;$$

$$OB_1 = PK_5 \bullet PK_4;$$

$$EA_1 = PK_2 \bullet PK_7;$$

$$EB_1 = PK_6 \bullet PK_3,$$

где  $PK_i$  — фрагмент последовательности  $PK$ :

$$PK = PK_1 \bullet PK_2 \bullet \dots \bullet PK_8.$$

Кроме того, для  $i = 2,3,4$  вычисляются следующие величины:

$$OA_i = OA_1 \oplus KA_i;$$

$$OB_i = OB_1 \oplus KB_i;$$

$$EA_i = EA_1 \oplus KA_i;$$

$$EB_i = EB_1 \oplus KB_i;$$

где  $KA_i$  и  $KB_i$  — следующие 64-битные константы:

$$KA_2 = B7E151628AED2A6A;$$

$$KA_3 = BF7158809CF4F3C7;$$

$$KA_4 = 62E7160F38B4DA56;$$

$$KB_2 = A784D9045190CFEF;$$

$$KB_3 = 324E7738926CFBE5;$$

$$KB_4 = F4BF8D8D8C31D763.$$

3. Конкатенацией определенных величин из вычисленных на предыдущем шаге получаются два временных ключа:

$$Kt_1 = OA_1 \bullet OB_1 \bullet \dots \bullet OA_4 \bullet OB_4;$$

$$Kt_2 = EA_1 \bullet EB_1 \bullet \dots \bullet EA_4 \bullet EB_4.$$

4. Вычисляются 128-битные ключи раундов:

- $K_i = Enc_4(K_{i-1}, Kt_1)$  — для нечетных значений  $i$ ;
- $K_i = Enc_4(K_{i-1}, Kt_2)$  — для четных значений  $i$ ,

где:

- $Enc_4(p, q)$  — функция, выполняющая 4-раундовое шифрование описанным выше алгоритмом блока данных  $p$  на ключе  $q$  (здесь в качестве ключей раундов поочередно используются 128-битные фрагменты временных ключей, вычисленных на предыдущем шаге);
- в качестве блока  $K_0$  берется 128 нулевых битов.

## Достоинства и недостатки алгоритма

Как и еще один участник конкурса AES — алгоритм шифрования НРС (см. разд. 3.23), алгоритм DFC оптимизирован под 64-битные платформы (за которыми уже ближайшее будущее — по мнению авторов алгоритма [47]), поэтому основным достоинством алгоритма является высокая скорость шифрования на 64-битных платформах. Кроме того, алгоритм DFC может быть использован и «на другом конце» ряда возможных платформ — в смарт-картах с минимальными ресурсами, поскольку требует менее ста байтов опе-

ративной памяти (в том числе благодаря возможности генерации ключей раундов «на лету», т. е. по мере необходимости) и менее 2 Кбайт энергонезависимой памяти [306]. Однако DFC не вышел во второй раунд конкурса, поскольку недостатки оказались более существенными.

- Низкая скорость шифрования на всех платформах, кроме 64-битных [284].
- Сами разработчики алгоритма обнаружили класс слабых ключей шифрования, при расширении которых формируются ключи раундов с нулевой  $a$ -половиной; вероятность попадания ключа в этот класс невелика и равна  $2^{-192}$ . Также авторы алгоритма предположили наличие еще одного класса возможно слабых ключей [162].
- Структура алгоритма базируется на ряде теоретических работ Сержа Воденэ (например, [382]), в которых формируется математическая модель алгоритмов шифрования с доказуемой высокой криптостойкостью против линейного и дифференциального криptoанализа. Однако соавторы более успешных алгоритмов, представленных на конкурс (вышедшего во второй раунд конкурса алгоритма Serpent и победителя конкурса — алгоритма Rijndael), Ларс Кнудсен и Винсент Риджмен в своей совместной работе, посвященной алгоритму DFC [221], доказали, что теория, лежащая в основе данного алгоритма, не гарантирует безопасности против дифференциального криptoанализа. Они же привели пример атаки, вскрывающей 6-раундовую версию алгоритма DFC при наличии  $2^{70}$  пар выбранных открытых текстов и соответствующих им шифртекстов путем выполнения порядка  $2^{129}$  вычислений описанной выше функции раунда. Данная атака вряд ли осуществима на практике; кроме того, атаке подвержена «усеченная» версия алгоритма; однако наличие этой атаки говорит о потенциальной слабости алгоритма DFC.

Впоследствии история алгоритма DFC продолжилась. Во-первых, известный криптолог Дон Копперсмит (Don Coppersmith) обнаружил еще один класс слабых ключей шифрования, приводящих к тому, что результат шифрования любого блока эквивалентен исходному, т. е. незашифрованному блоку; такой результат возникал в случае, если значение ключа второго раунда алгоритма оказывалось нулевым (что происходит с вероятностью  $2^{-128}$ ) [46]. Кроме того, была обнаружена еще одна слабость в процедуре расширения ключа, правда, не приводящая к такому плачевному результату, как предыдущая. Поэтому авторы алгоритма предложили новую версию алгоритма (DFCv2) с незначительными изменениями в процедуре расширения ключа, устранившими все описанные здесь проблемы с ключами. Кроме того, была предложена новая реализация алгоритма, еще быстрее работающая на 64-битных платформах [46, 167]. На результаты конкурса AES это уже не повлияло.

## 3.17. Алгоритм E2

Алгоритм E2 был разработан в 1998 г. группой ученых из японской корпорации NTT (Nippon Telegraph and Telephone Corporation) с участием специалистов Национального университета г. Йокогама [191, 363, 369].

### Структура алгоритма

Разработчики алгоритма E2 придали ему традиционную схему, присущую многим алгоритмам симметричного шифрования конца XX века — на основе сети Фейстеля (см. разд. 1.3). Структура алгоритма представлена на рис. 3.75.

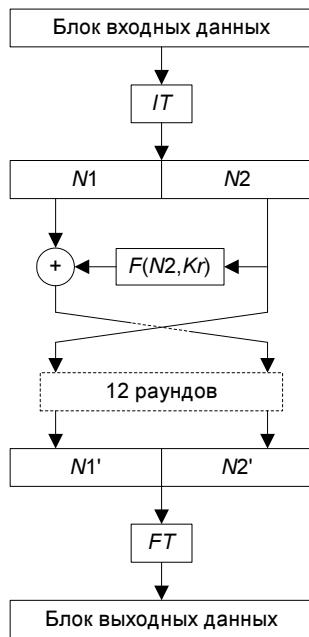


Рис. 3.75. Структура алгоритма E2

Перед выполнением описанных выше операций над субблоками выполняется начальное преобразование данных (IT, Initial Transformation), а перед завершением работы алгоритма выполняется заключительное преобразование (FT, Final Transformation). В этом алгоритм E2 также схож с многими другими.

Как видно, основная нагрузка по шифрованию данных лежит на функции  $F$ , обрабатывающей содержимое субблока  $N2$ . В качестве параметра данная функция берет также значение  $K_r$  — это ключ раунда, т. е. 128-битный

фрагмент ключа шифрования после его обработки функцией расширения ключа, с которым смешивается шифруемая информация в конкретном раунде  $r$ . Помимо ключей раундов, функция расширения ключа (выполняющая начальное преобразование ключа шифрования перед его использованием) дает еще 4 128-битных фрагмента ключа, два из которых ( $K_{13}$  и  $K_{14}$ ) используются в операции  $IT$ , а остальные два ( $K_{15}$  и  $K_{16}$ ) — в  $FT$ .

Функция  $F$  состоит из нескольких последовательно выполняемых операций, показанных на рис. 3.76. Все указанные операции выполняются над отдельными байтами субблока  $N2$ , т. е. можно считать, что 64-битный субблок  $N2$  разбивается на 8 подблоков  $n_x$ , каждый из которых имеет размер 8 битов. Рассмотрим преобразования данных внутри функции  $F$ .

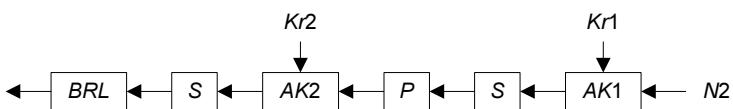


Рис. 3.76. Операции функции  $F$

- Операции  $AK1$  и  $AK2$  выполняют наложение соответственно левой ( $K_{r1}$ ) и правой ( $K_{r2}$ ) частей ключа раунда  $K_r$  на входной субблок данных. Наложение выполняется побитовым сложением по модулю 2 каждого байта субблока и соответствующего ему байта ключа раунда. То есть операции  $AK1$  и  $AK2$  выполняются так ( $n_x$  и  $n_x'$  обозначают, соответственно, текущее значение  $x$ -го байта субблока перед конкретной операцией и после нее>):

$$AK1: n_x' = K_{r1,x} \oplus n_x;$$

$$AK2: n_x' = K_{r2,x} \oplus n_x.$$

- Операция  $S$  представляет собой нелинейную замену значений байтов субблока согласно преобразованию, которое можно представить следующей формулой:

$$n_x' = a * n_x^b + c,$$

где  $a$ ,  $b$ ,  $c$  — фиксированные константы (97, 127 и 225 соответственно), а все вычисления выполняются по модулю 256.

Для ускорения вычислений может быть использована таблица замен с предварительно рассчитанными выходными значениями для каждого из 256 значений  $n_x$ , что, однако, повышает требования к энергонезависимой памяти ввиду необходимости хранения таблицы. Эти значения представлены в табл. 3.23.

Таблица 3.23

225	66	62	129	78	23	158	253	180	63	44	218	49	30	224	65
204	243	130	125	124	18	142	187	228	88	21	213	111	233	76	75
53	123	90	154	144	69	188	248	121	214	27	136	2	171	207	100
9	12	240	1	164	176	246	147	67	99	134	220	17	165	131	139
201	208	25	149	106	161	92	36	110	80	33	128	47	231	83	15
145	34	4	237	166	72	73	103	236	247	192	57	206	242	45	190
93	28	227	135	7	13	122	244	251	50	245	140	219	143	37	150
168	234	205	51	101	84	6	141	137	10	94	217	22	14	113	108
11	255	96	210	46	211	200	85	194	35	183	116	226	155	223	119
43	185	60	98	19	229	148	52	177	39	132	159	215	81	0	97
173	133	115	3	8	64	239	104	254	151	31	222	175	102	232	184
174	189	179	235	198	107	71	169	216	167	114	238	29	126	170	182
117	203	212	48	105	32	127	55	91	157	120	163	241	118	250	5
61	58	68	87	59	202	199	138	24	70	156	191	186	56	86	26
146	77	38	41	162	152	16	153	112	160	197	40	193	109	20	172
249	95	79	196	195	209	252	221	178	89	230	181	54	82	74	42

Таблица заменяет значение 0 на значение 225, 1 — на 66, 2 — на 62 и т. д.

3. Операция  $P$  обеспечивает рассеивание данных путем сложения значений  $n_x$  по модулю 2 по определенным правилам. Правила сложения представлены в табл. 3.24.

Таблица 3.24

$x$	1	2	3	4	5	6	7	8
1	—	+	+	+	+	+	+	—
2	+	—	+	+	—	+	+	+
3	+	+	—	+	+	—	+	+
4	+	+	+	—	+	+	—	+
5	+	+	—	+	+	+	—	—
6	+	+	+	—	—	+	+	—
7	—	+	+	+	—	—	+	+
8	+	—	+	+	+	—	—	+

Каждая строка таблицы определяет зависимость соответствующего выходного значения  $n_x'$ , которое формируется путем сложения по модулю 2 всех значений  $n_x$ , пересечение с которыми отмечено в таблице знаком «+». То есть, например, выходное значение 5-го байта вычисляется следующим образом:

$$n_5' = n_1 \oplus n_2 \oplus n_4 \oplus n_5 \oplus n_6.$$

Это не означает, что операция  $P$  выполняет 44 сложения (по количеству знаков «+» в таблице). Всего выполняется 8 операций сложения по модулю 2 (рис. 3.77). Например, с тем же 5-м байтом сначала складывается исходное значение байта № 1, а затем — промежуточное значение байта № 4, которое предварительно было получено сложением исходного значения  $n_4$  и промежуточного значения 6-го байта, вычисленного сложением  $n_2$  и  $n_6$ .

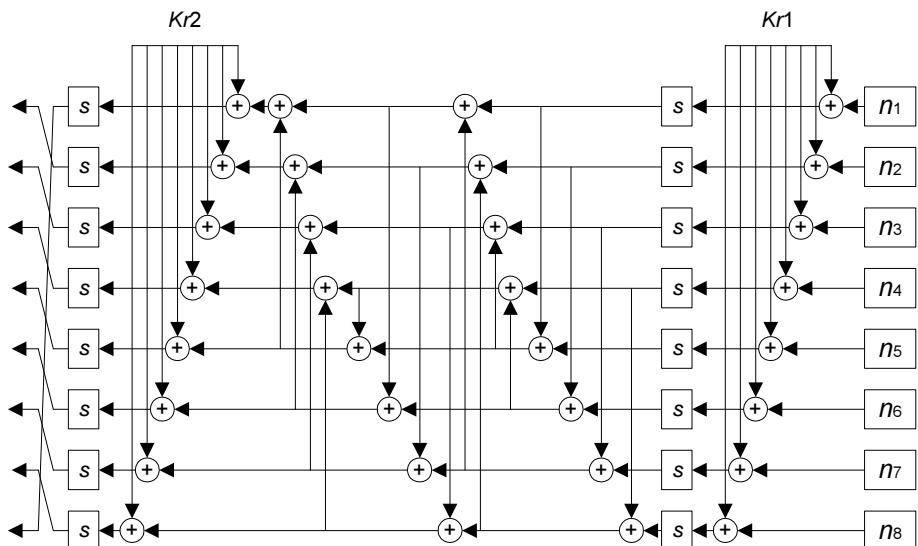


Рис. 3.77. Обработка информации функцией  $F$

Для исключения зависимости выходного значения  $n_x'$  от  $n_x$  (байты с 1-го по 4-й) значение  $n_x$  опосредованно складывается само с собой, поскольку известно свойство сложения по модулю 2:

$$A \oplus B \oplus B = A$$

для любых значений  $A$  и  $B$ . Сказанное можно проиллюстрировать на вычислении  $n_4'$ :  $n_4$  сначала складывается с  $n_6$  и  $n_2$  (операция их сложения выполнялась до этого), затем с промежуточным значением байта № 8, которое

является результатом сложения  $n_8$ ,  $n_4$ ,  $n_3$ ,  $n_5$  и  $n_1$ , т. е. значение  $n_4$  устраняется повторным сложением, а результат выглядит согласно таблице:

$$n_4' = n_6 \oplus n_2 \oplus n_8 \oplus n_3 \oplus n_5 \oplus n_1.$$

4. Операция  $BRL$  представляет собой побайтовый циклический сдвиг данных влево, т. е.:  $n_8' = n_1$ ,  $n_1' = n_2$ ,  $n_2' = n_3$  и т. д.

Начальное преобразование ( $IT$ ) выполняет над входными данными следующие операции:

1. Сложение по модулю 2 с фрагментом ключа  $K_{13}$  (как было сказано выше, функция расширения ключа дает два фрагмента ключа для  $IT$ ).
2. Умножение каждой 32-битной части входных данных на соответствующую часть фрагмента ключа  $K_{14}$ , к которой предварительно применена побитовая логическая операция «или» с 32-битным числом, имеющим значение 1. Умножение выполняется по модулю  $2^{32}$ .
3. Байтова перестановка данных согласно рис. 3.78.

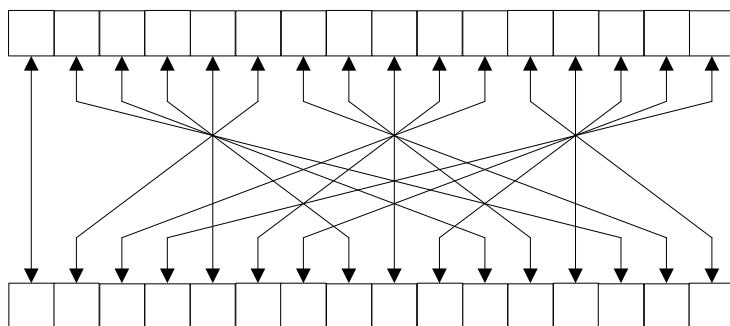


Рис. 3.78. Байтова перестановка в операциях  $IT$  и  $FT$

Операция  $FT$  выполняет обратные действия относительно операции  $IT$ : сначала выполняется обратная байтова перестановка, а затем — операция, обратная описанному выше умножению:

$$x_i' = x_i * (k_i | 1) \bmod 2^{32},$$

где:

- $x_i$  — 32-битная часть входных данных;
- $k_i$  — 32-битная часть фрагмента ключа  $K_{15}$ ;
- $|$  — побитовая логическая операция «или».

В завершение же выполняется сложение по модулю 2 со вторым из предначертанных для этой операции фрагментов ключа —  $K_{16}$ .

## Почему алгоритм E2 не вышел в финал конкурса AES

Сеть Фейстеля «по определению» дает алгоритму ряд преимуществ: алгоритмы, основанные на ней, применялись в течение десятилетий, их свойства хорошо изучены, что дает некоторую гарантию отсутствия как скрытых уязвимостей, так и незадекларированных возможностей. Известны также сильные и слабые стороны таких алгоритмов по отношению к различным методам криптоанализа. Кроме того, неоспоримым достоинством такой структуры является «симметричность» процедур зашифровывания и расшифровывания, для реализации которых можно использовать практически один и тот же программный код или аппаратный модуль.

Алгоритм E2 является очередным подтверждением надежности алгоритмов шифрования, основанных на сети Фейстеля, — в процессе анализа данного алгоритма не было выявлено каких-либо слабостей (за исключением «усеченных» версий алгоритма — с уменьшенным количеством раундов) и серьезных недостатков. Однако в соперничестве с другими алгоритмами шифрования, многие из которых также не имели серьезных недостатков, решающими факторами оказались следующие [284]:

- алгоритм E2 предъявляет очень высокие требования к энергонезависимой памяти, что делает весьма затрудненным его применение в таких устройствах, как смарт-карты;
- алгоритм E2 показывает относительно невысокую скорость шифрования;
- дополнительные ресурсы требуются для функции расширения ключа, что также не способствует реализации E2 в смарт-картах.

Таким образом, разработчики алгоритма E2 создали действительно сильный алгоритм шифрования, не подверженный каким-либо известным или прогнозируемым атакам. Однако высокая криптостойкость алгоритма достигается за счет невысокой скорости шифрования и активного использования вычислительных ресурсов и памяти, что и предопределило «неуспех» E2 в конкурсе AES.

## 3.18. Алгоритм FEAL

### История создания алгоритма

Алгоритм FEAL был разработан криптографами Акихиро Шимизу (Akihiro Shimizu) и Шоджи Миагучи (Shoji Miyaguchi) из японской компании NTT в 1987 г. и представлен на конференции EUROCRYPT'87. Название алгоритма — аббревиатура от Fast data Encipherment ALgorithm (быстрый алгоритм шифрования данных).

Стоит сказать, что компания NTT весьма известна и своими более поздними разработками в этой области, в частности:

- алгоритм E2 (*см. разд. 3.17*), разработанный компанией NTT с участием специалистов университета г. Йокогама (Япония), участвовал в конкурсе на стандарт шифрования США AES (*см. разд. 2.1*); впрочем, он не попал даже в финал конкурса [284];
- алгоритм Camellia (*см. разд. 3.9*) — совместная разработка NTT и еще одной японской компании Mitsubishi Electric — стал одним из победителей (в номинации 128-битных алгоритмов блочного симметричного шифрования) европейского конкурса криптоалгоритмов NESSIE (*см. разд. 2.2*).

Алгоритм FEAL интересен тем, что его разработчики продвигали FEAL как потенциальную замену стандарта шифрования DES — по их мнению, FEAL-4 (число в названии обозначает количество раундов алгоритма) предлагал более быстрое шифрование без потери в его качестве [28]. Кроме того, в FEAL отсутствовали табличные замены, поэтому реализация алгоритма не требовала дополнительной энергонезависимой памяти для хранения таблиц.

## Структура алгоритма

Структура  $N$ -раундового алгоритма FEAL- $N$  вместе с процедурой расширения ключа шифрования (которая будет подробно описана далее) приведена на рис. 3.79 [28, 263].

Фактически в каждом раунде алгоритма выполняется лишь обработка правого 32-битного субблока функцией  $f$ , которая в качестве второго параметра использует 16-битное значение ключа раунда. Результат обработки накладывается на левый субблок операцией XOR. Перед первым раундом выполняется операция XOR блока шифруемых данных с фрагментом расширенного ключа, а также аналогичное наложение левого субблока на правый. Те же операции в обратном порядке производятся и после финального раунда алгоритма.

Структура функции  $f$  представлена на рис. 3.80. Здесь используются лишь три вида преобразований — это операция XOR, а также функции  $S_0$  и  $S_1$ , которые можно описать следующим образом:

$$S_0(x, y) = ROL_2((x + y) \bmod 2^8);$$

$$S_1(x, y) = ROL_2((x + y + 1) \bmod 2^8),$$

где  $ROL_2$  — операция циклического сдвига влево на 2 бита.

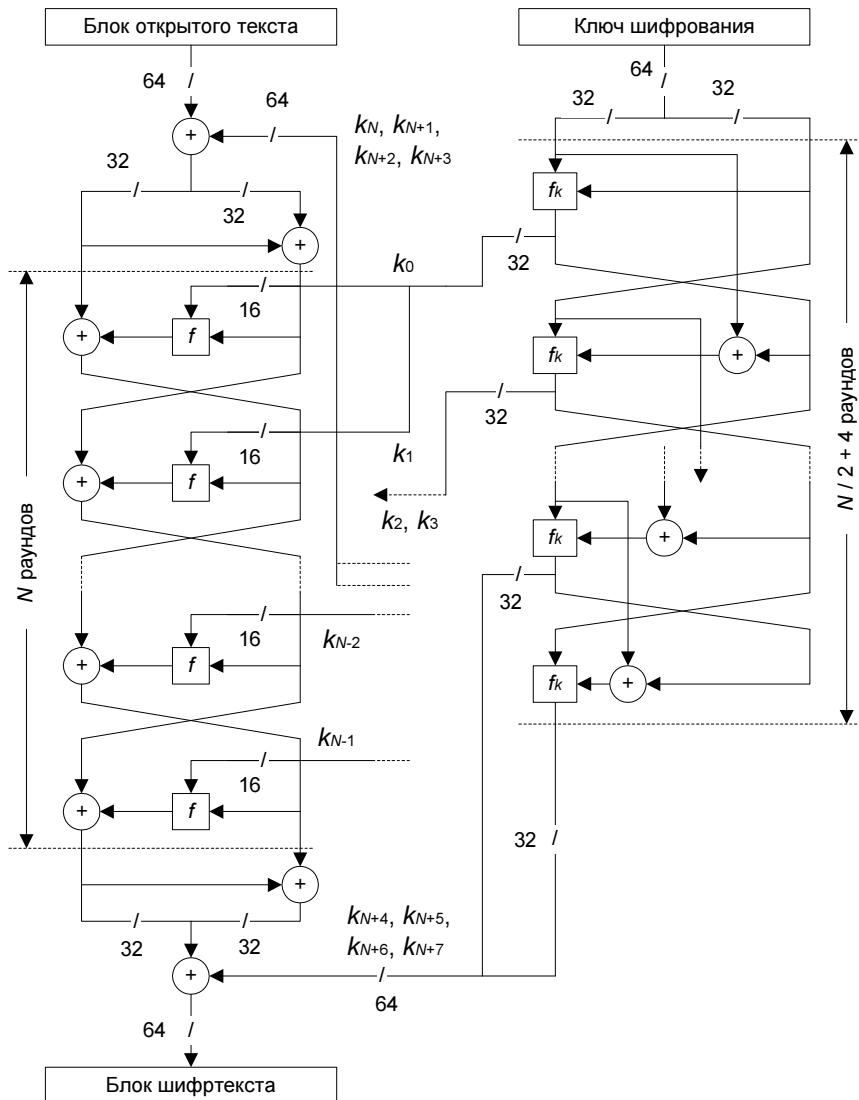


Рис. 3.79. Структура алгоритма FEAL-N

Таким образом, выполняемые функцией  $f$  преобразования формулируются так (см. рис. 3.80):

$$t_1 = m_0 \oplus m_1 \oplus kr_0;$$

$$t_2 = m_2 \oplus m_3 \oplus kr_1;$$

$$f_1 = S_1(t_1, t_2);$$

## Описание алгоритмов

$$\begin{aligned}f_0 &= S_0(m_0, f_1); \\f_2 &= S_0(t_2, f_1); \\f_3 &= S_1(f_2, m_3); \\f(m, kr) &= f_0 \bullet f_1 \bullet f_2 \bullet f_3,\end{aligned}$$

где:

- $t_1$  и  $t_2$  — временные переменные;
- — операция конкатенации.

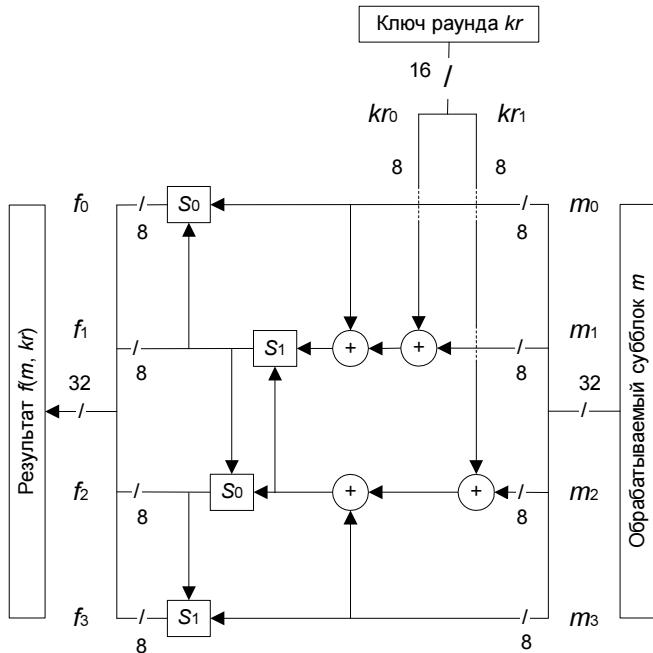


Рис. 3.80. Функция  $f$

Расшифровывание данных выполняется аналогичным образом, но с использованием ключей раундов в обратном порядке.

## Процедура расширения ключа

Функция расширения ключа должна выработать  $N$  16-битных ключей раундов, а также 2 64-битных фрагмента для операций XOR, выполняемых в начале и в конце алгоритма (см. рис. 3.79). Поскольку один раунд обработки ключа вырабатывает 32-битный фрагмент расширенного ключа, для работы функции расширения ключа требуется  $(N/2) + 4$  раундов.

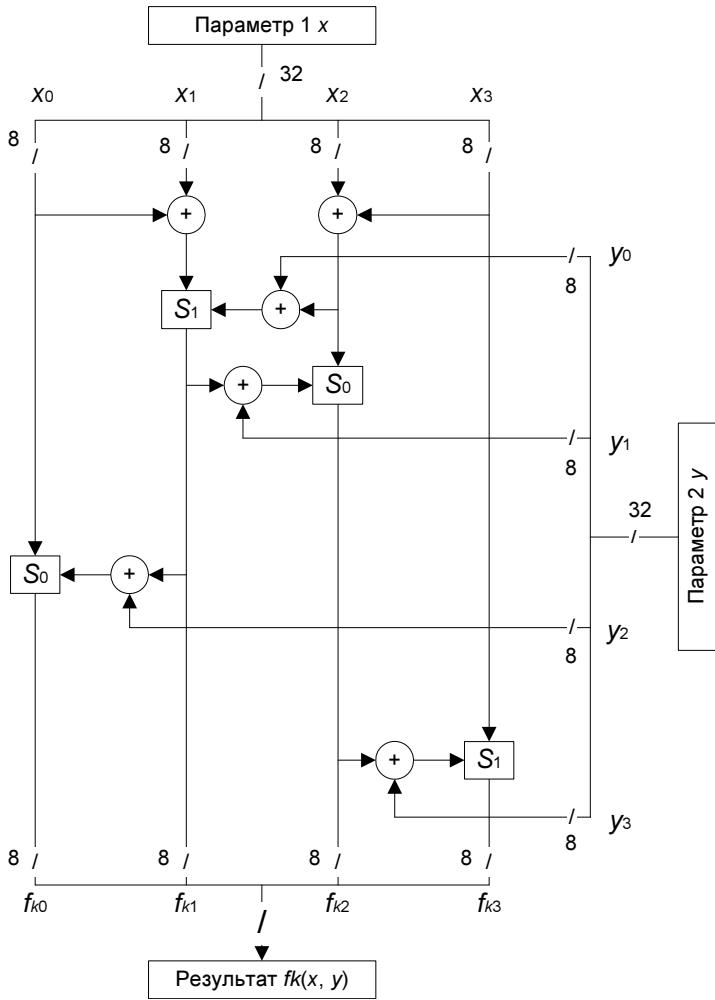


Рис. 3.81. Функция  $f_k$

Структура процедуры расширения ключа приведена на рис. 3.79. Как видно из рисунка, основой данной процедуры является функция  $f_k$ , обрабатывающая два 32-битных параметра с получением также 32-битного результата, представляющего собой, в общем случае, ключи двух раундов алгоритма. Функция  $f_k$  использует те же операции  $S_0$ ,  $S_1$  и XOR, что и функция  $f$ , но в другой комбинации, которую можно формализовать следующим образом (рис. 3.81):

$$t_1 = x_0 \oplus x_1;$$

$$t_2 = x_2 \oplus x_3;$$

$$\begin{aligned}f_{k1} &= S_1(t_1, t_2 \oplus y_0); \\f_{k2} &= S_0(t_2, f_{k1} \oplus y_1); \\f_{k0} &= S_0(x_0, f_{k1} \oplus y_2); \\f_{k3} &= S_1(x_3, f_{k2} \oplus y_3); \\f_k(x, y) &= f_{k0} \bullet f_{k1} \bullet f_{k2} \bullet f_{k3}.\end{aligned}$$

## Почему FEAL не используется

Алгоритм FEAL-4, изначально предлагавшийся в качестве замены стандарта DES, оказался весьма нестоек к различным видам криптоанализа. Та же участь постигла и алгоритм FEAL-8. Алгоритмы же FEAL-16, FEAL-32 и т. д. оказались существенно более стойкими за счет большего количества раундов (хотя для FEAL-16 также были найдены методы вскрытия), однако по той же причине скорость их работы оказалась уже ниже скорости алгоритма DES, так что никаких преимуществ перед DES у FEAL не оказалось. Соответственно, как стандарт шифрования алгоритм FEAL не состоялся.

Фактически сформировалась целая история методов вскрытия алгоритма FEAL, наиболее значительными из которых можно считать следующие.

- Через год после презентации алгоритма FEAL-4 было опубликовано исследование голландского математика Берта ден Боера (Bert den Boer) [99], в котором была доказана возможность вычисления ключа шифрования данного алгоритма на основе 10 000 выбранных открытых текстов.
- В 1990 г. английский исследователь Шон Мерфи (Sean Murphy) существенно улучшил результат предыдущего метода [274], предложив алгоритм вычисления ключа шифрования алгоритма FEAL-4 всего на основе 20 выбранных открытых текстов. Фактически этот алгоритм поставил крест на практическом применении FEAL-4, хотя по современным меркам достаточно и предыдущей атаки — например, один из участников конкурса AES — алгоритм LOKI97 — был отвергнут из-за возможности атаки на основе  $2^{56}$  выбранных или стольких же известных открытых текстов [284], что несопоставимо с числами 20 или 10 000.
- Тогда же известные израильские криптологи Эли Бихам и Эди Шамир доказали, что алгоритм FEAL-4 можно вскрыть на основе всего 8 выбранных открытых текстов, для вскрытия FEAL-8 достаточно 2000 выбранных или  $2^{37.5}$  известных открытых текстов, а для FEAL-16 —  $2^{28}$  выбранных или  $2^{46.5}$  известных открытых текстов [76]. Это исследование «похоронило» алгоритм FEAL-8 и поставило под сомнение использование алгоритма FEAL-16.

Были и другие криптоаналитические исследования FEAL, например, [189, 290, 292]. А в своей работе [27], адресованной начинающим криптоаналитикам, Брюс Шнайер утверждает, что «все современные криптоаналитические атаки работают против FEAL». Фактически, по словам Брюса Шнайера, FEAL стал хорошим объектом тренировок как для начинающих криптоаналитиков, так и для профессионалов, изобретающих новые методы криптоанализа.

Кроме того, в 90-х гг. XX века уже не выглядела фантастической возможность полного перебора  $2^{64}$  вариантов 64-битного ключа шифрования алгоритма FEAL. Поэтому его авторами был предложен также FEAL-NX со 128-битным ключом шифрования. Однако FEAL-NX отличался от FEAL-N только измененной процедурой расширения ключа, поэтому не удивительно, что в том же 1991 г. Эли Бихам и Эди Шамир доказали, что вскрытие FEAL-NX не сложнее вскрытия FEAL-N при том же значении  $N$ .

И вообще, алгоритм просто не выглядит современным. Например, на конкурсе AES принималась в расчет возможность выполнения процедуры расширения ключа параллельно с раундами шифрования (чтобы не хранить все ключи раундов в памяти, а вычислять их по мере необходимости) [284]. А на рис. 3.79 отчетливо видно, что до первой же операции XOR необходимо выполнить практически всю процедуру расширения ключа, чтобы вычислить подключи, участвующие в этой операции.

## 3.19. Алгоритм FROG

Алгоритм FROG был создан в 1998 г. тремя специалистами компании Tecnologia Apropriada (TecApro) из небольшого латиноамериканского государства Коста-Рика (неизвестного ранее своими разработками в области криптографии): Дианелосом Георгудисом (Dianelos Georgoudis), Дамианом Леру (Damian Leroux) и Билли Симоном Чавесом (Billy Simon Chaves) [160].

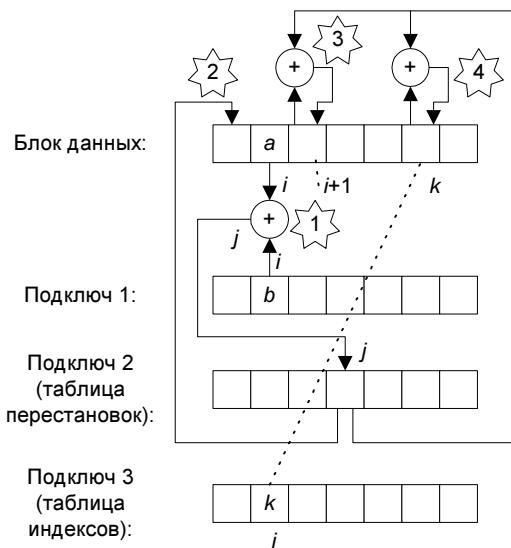
### Основные характеристики и структура алгоритма

Как известно, конкурс AES (см. разд. 2.1) устанавливал для алгоритмов — участников конкурса обязательность поддержки 128-битного размера блока шифруемых данных, а также 128-, 192- и 256-битных ключей шифрования. Однако, разработчики алгоритма FROG предложили несравненно более широкий набор значений этих параметров:

- допустимо использовать ключи размером от 5 до 125 байтов (т. е. от 40 до 1000 битов);

## Описание алгоритмов

- размер блока может варьироваться от 8 до 128 байтов, т. е. от 64 до 1024 битов (однако в авторском описании алгоритма в соответствии с требованиями AES фигурирует 16-байтный размер блока).



**Рис. 3.82.** Структура раунда алгоритма FROG

Независимо от размера ключа и блока, шифрование выполняется в 8 раундов. В каждом раунде над каждым байтом шифруемого блока (для определенности также будем считать, что блоки имеют размер 16 байтов) производятся следующие действия (рис. 3.82):

1. Значение текущего ( $i$ -го) байта складывается по модулю 2 со значением того же байта 1-й части ключа раунда (процедура расширения ключа описана далее).
2. Вторая часть ключа раунда представляет собой таблицу перестановок. Из данной таблицы выбирается байт, порядковый номер которого равен значению, вычисленному на первом шаге. Значение выбранного байта замещает значение текущего байта шифруемого блока данных.
3. Значение следующего байта ( $i + 1$ -го) складывается по модулю 2 со значением, выбранным на шаге 2. Полученный результат замещает старое значение  $i + 1$ -го байта шифруемого блока данных.
4. Третья часть ключа раунда представляет собой таблицу индексов. Значение  $i$ -го байта таблицы индексов определяет еще один модифицируемый

байт шифруемого блока данных, который изменяется полностью аналогично  $i + 1$ -му байту (т. е. складывается по модулю 2 со значением, полученным на шаге 2).

## Процедура расширения ключа

Эта процедура должна получить из ключа шифрования 8 ключей раундов — по одному на каждый раунд алгоритма. Как видно из описания алгоритма, ключ раунда состоит из трех подключей:

- 16-байтная 1-я часть;
- 256-байтная таблица перестановок;
- 16-байтная таблица индексов.

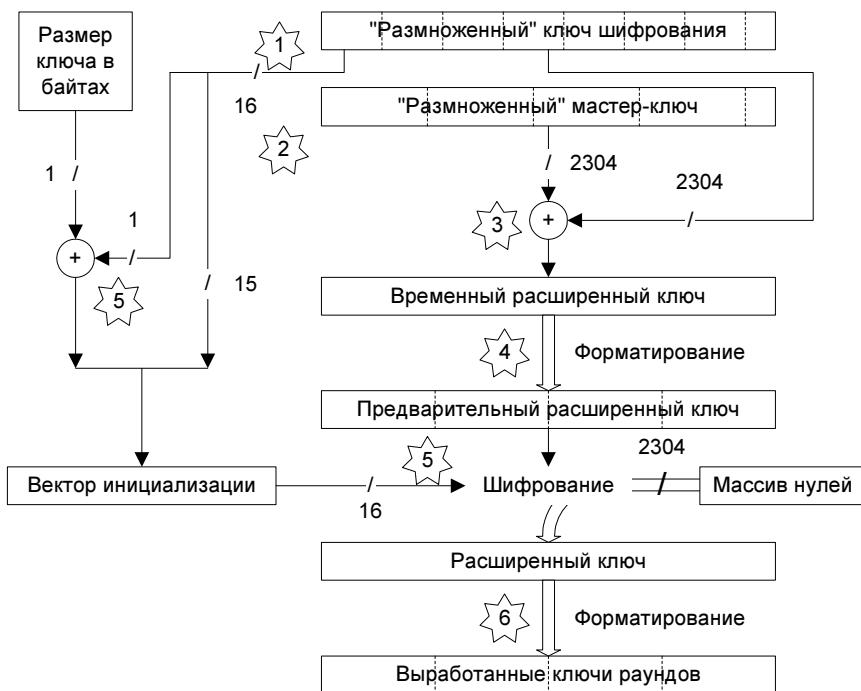


Рис. 3.83. Процедура расширения ключа

Таким образом, для работы алгоритма необходимо выработать 2304 байтос ключевой информации. Вот как это происходит (рис. 3.83):

1. Путем «размножения» ключа шифрования (т. е. значение ключа шифрования повторяется необходимое количество раз, а ключ шифрования этого

алгоритма, как было сказано выше, имеет размер от 5 до 125 байтов) вырабатывается первый 2304-байтный временный массив данных. Байты последней из «размноженных» копий ключа, выходящие за границу требуемого 2304-байтного массива (например, последний 71 байт 125-байтного ключа — ключа максимального размера), отбрасываются.

- Аналогичным «размножением» 251-байтного фрагмента мастер-ключа формируется второй 2304-байтный временный массив данных. Мастер-ключ представляет собой специально подобранный константу размером 251 байт. Побайтное значение мастер-ключа (начиная с младшего байта) приведено в табл. 3.25.

**Таблица 3.25**

113	21	232	18	113	92	63	157	124	193	166	197	126	56	229	229
156	162	54	17	230	89	189	87	169	0	81	204	8	70	203	225
160	59	167	189	100	157	84	11	7	130	29	51	32	45	135	237
139	33	17	221	24	50	89	74	21	205	191	242	84	53	3	230
231	118	15	15	107	4	21	34	3	156	57	66	93	255	191	3
85	135	205	200	185	204	52	37	35	24	68	185	201	10	224	234
7	120	201	115	216	103	57	255	93	110	42	249	68	14	29	55
128	84	37	152	221	137	39	11	252	50	144	35	178	190	43	162
103	249	109	8	235	33	158	111	252	205	169	54	10	20	221	201
178	224	89	184	182	65	201	10	60	6	191	174	79	98	26	160
252	51	63	79	6	102	123	173	49	3	110	233	90	158	228	210
209	237	30	95	28	179	204	220	72	163	77	166	192	98	165	25
145	162	91	212	41	230	110	6	107	187	127	38	82	98	30	67
225	80	208	134	60	250	153	87	148	60	66	165	72	29	165	82
211	207	0	177	206	13	6	14	92	248	60	201	132	95	35	215
118	177	121	180	27	83	131	26	39	46	12					

- Путем применения операции сложения по модулю 2 к соответствующим битам двух массивов, выработанных на шагах 1 и 2, получается временный расширенный ключ.
- Форматирование ключа, полученного на шаге 3 (т. е. его разбиение на 8 отрезков — по числу раундов — размером  $16 + 256 + 16$  байтов, а также дополнительная обработка, которая будет описана далее), дает предварительный расширенный ключ алгоритма.

5. Предварительный расширенный ключ используется для зашифровывания 2304-байтного массива, заполненного нулями. Шифрование выполняется в режиме сцепления блоков шифра (*см. разд. 1.4*), где в качестве вектора инициализации (IV) используются первые 16 байтов исходного ключа шифрования, самый первый из которых еще и складывается по модулю 2 с числом, равным размеру ключа шифрования в байтах (это необходимо для получения различных IV для ключей разного размера, но с совпадающими первыми 16 байтами). Результат операции — расширенный ключ, заполненный псевдослучайной информацией.
6. Аналогично шагу 4, выполняется форматирование расширенного ключа для получения восьми ключей раундов с необходимой структурой.

## Форматирование ключа

Шаги 4 и 6 алгоритма расширения ключа представляют собой форматирование 2304-байтной псевдослучайной последовательности с целью получения 8 ключей раундов. Если 1-я часть ключа раунда, используемая для сложения по модулю 2, может иметь абсолютно случайную структуру, то корректная таблица перестановок должна состоять из 256 неповторяющихся значений от 0 до 255, а к 16-байтной таблице индексов, кроме того, предъявляются дополнительные требования.

Таким образом, при форматировании производятся следующие действия:

1. Разбиение 2304-байтного массива на 8 фрагментов по  $16 + 256 + 16$  байтов.
2. Первые 16 байтов фрагмента становятся первой частью ключа раунда без изменений.
3. Следующие 256 байтов (обозначим этот фрагмент  $A$ ) обрабатываются специальной процедурой с целью получения корректной таблицы перестановок. Данная процедура выглядит так:
  - создается 256-байтный массив  $U$ , содержащий последовательно расположенные значения от 0 до 255;
  - в цикле от 0 до 255  $i$ -й байт таблицы перестановок  $T$  определяется по формуле:

$$T[i] = U[(index[i-1] + A[i]) \bmod L],$$

где:

- ◊  $index[i-1]$  — номер элемента массива  $U$ , использованного на предыдущем шаге (для нулевого шага принимается равным 0), а  $L$  — текущий размер массива  $U$ ;

## Описание алгоритмов

- ◊ использованный байт массива  $U$  выбрасывается, а расположенные после него элементы массива  $U$  сдвигаются на 1 байт к началу массива; таким образом, значение  $L$  в каждом проходе данного цикла уменьшается на 1;
  - если таблица перестановок создается для операции расшифровывания, то выполняется ее инвертирование.
4. Аналогично шагу 3 создается 16-байтная таблица индексов  $I$ .
5. Выполняется анализ цепочек ссылок таблицы индексов; если таблица состоит из нескольких цепочек, производится изменение таблицы с целью получения одной цепочки ссылок, длина которой будет равна размеру таблицы.
6. Таблица снова анализируется с целью поиска индексов, удовлетворяющих следующему условию:

$$I[i] = i + 1;$$

если такие элементы таблицы существуют, их значение меняется на

$$I[i] = i + 2.$$

Шаги 3–5 процедуры форматирования стоит рассмотреть на примере. Предположим, создается 6-байтная таблица индексов из следующего 6-байтного фрагмента  $A$  псевдослучайной последовательности:

$$21,247,199,108,66,239.$$

В цикле от 0 до 5 (шаг 4)  $i$ -й байт таблицы индексов  $I$  определяется по формуле:

$$I[i] = U[(index[i - 1] + A[i]) \bmod L],$$

что на примере приведенной выше последовательности выглядит так, как показано в табл. 3.26.

**Таблица 3.26**

$U$	$index[i - 1] + A[i] \bmod L$	$I$
0, 1, 2, 3, 4, 5	$0 + 21 \bmod 6 = 3$	3
0, 1, 2, 4, 5	$3 + 247 \bmod 5 = 0$	3, 0
1, 2, 4, 5	$0 + 199 \bmod 4 = 3$	3, 0, 5
1, 2, 4	$3 + 108 \bmod 3 = 0$	3, 0, 5, 1
2, 4	$0 + 66 \bmod 2 = 0$	3, 0, 5, 1, 2
4		3, 0, 5, 1, 2, 4

Результат — таблица индексов:

3,0,5,1,2,4.

Анализ таблицы (шаг 5) позволяет выяснить, что данная таблица состоит из двух цепочек ссылок:

$3 \rightarrow 1 \rightarrow 0 \rightarrow 3$  и  $5 \rightarrow 4 \rightarrow 2 \rightarrow 5$ .

Однако для достижения максимальной криптостойкости алгоритма таблица индексов должна содержать одну цепочку максимального размера. Алгоритм, выполняемый на шаге 5, позволяет объединить несколько цепочек в одну. В данном примере это достигается путем перемены местами значений 0 и 2, т. е. получается следующая таблица индексов:

3,2,5,1,0,4,

которая формирует одну цепочку:

$3 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 0 \rightarrow 3$ .

## Достоинства и недостатки алгоритма

Несомненно, FROG является одним из самых оригинальных из современных алгоритмов шифрования. Как и многие другие алгоритмы, например AES (Rijndael) или SAFER+ (см. разд. 3.3 и 3.45 соответственно), FROG является байт-ориентированным. Однако, в отличие от объяснимых и объясненных авторами Rijndael и SAFER+ преобразований, примененных в этих алгоритмах, авторы алгоритма FROG ограничились пояснением, что такая необычная структура раунда выбрана с целью обеспечить максимально быстрое рассеивание входных данных (т. е. обеспечение влияния каждого бита шифруемых данных на каждый бит шифртекста в пределах блока).

К сожалению, алгоритм FROG был признан одним из наиболее слабых участников конкурса AES; в нем было найдено множество недостатков, в частности [284, 387]:

- довольно большая часть множества возможных ключей алгоритма оказалась слабой (благодаря очень сложной процедуре расширения ключа) к различным видам атак;
- алгоритм оказался весьма медленным, причем даже по сравнению с алгоритмами, известными до конкурса AES, например, Blowfish (см. разд. 3.8) и RC5 (см. разд. 3.42);
- алгоритм FROG оказался очень требовательным к оперативной памяти — ему необходимо около 2500 байтов в случае, если ключи раундов сформированы заранее, а для работы полнофункционального алгоритма, включ-

чающего процедуру расширения ключа, необходимо около 5000 байтов; эти требования очень велики (особенно в сравнении с другими алгоритмами — участниками конкурса AES) для реализации данного алгоритма в смарт-картах.

Таким образом, алгоритм FROG не вышел в финал конкурса AES.

## 3.20. Алгоритм Grand Cru

Алгоритм Grand Cru разработан специально для участия в конкурсе NESSIE (см. разд. 2.2). Автор алгоритма — Йохан Борст (Johan Borst) из Католического Университета г. Лювен, Бельгия.

Алгоритм Grand Cru имеет структуру «квадрат». Данный алгоритм основан на алгоритме Rijndael, разработанном Джоан Деймен и Винсентом Риджменом (см. разд. 3.3). Несомненно, тот факт, что Rijndael выиграл конкурс AES и в настоящее время является принятым в США стандартом шифрования, воодушевил некоторых разработчиков алгоритмов шифрования на конструирование блочных шифров на его основе. Фактически Grand Cru является глубокой модификацией алгоритма Rijndael.

Как известно, среди четырех преобразований данных в каждом раунде алгоритма Rijndael существует только одна операция (наложение подключа операцией XOR), выполняющая зависящие от ключа преобразования [153]. По мнению автора алгоритма Grand Cru, увеличение количества ключевых преобразований в раунде алгоритма усилит криптостойкость алгоритма при том же количестве раундов или позволит уменьшить количество раундов (чем увеличит скорость шифрования) при заданном уровне криптостойкости. Соответственно, раунд Grand Cru — это, фактически, раунд Rijndael с добавлением двух ключевых операций вместо одной бесключевой [102].

Алгоритм Grand Cru шифрует данные блоками по 128 битов с использованием только 128-битного ключа шифрования.

### Структура алгоритма

Как было сказано выше, алгоритм имеет структуру «квадрат»: при шифровании данные представляются в виде таблицы  $4 \times 4$  байтов. В каждом раунде производятся следующие действия:

1. Наложение фрагмента ключа  $r$ -го раунда  $k[r,0]$  (см. далее), которое выполняется с помощью операции XOR, полностью аналогично операции AddRoundKey алгоритма Rijndael — см. рис. 3.11. Эта операция в алгоритме Grand Cru обозначается как  $\sigma$ .

Как было сказано выше, раунд алгоритма содержит 3 операции с участием фрагментов расширенного ключа. Обозначим эти фрагменты следующим образом:

- $k[r,0]$  — 128-битный фрагмент ключа раунда  $k[r]$  для наложения ключа;
- $k[r,1]$  — 40-битный фрагмент для описанной далее операции  $\pi$ ;
- $k[r,2]$  — 48-битный фрагмент для операции  $\beta$ .

2. Табличная замена  $\gamma$ , замещающая каждый байт массива согласно таблице  $S$  (табл. 3.27).

**Таблица 3.27**

99	124	119	123	242	107	111	197
48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240
173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204
52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154
7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160
82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91
106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133
69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245
188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23
196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136
70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92

Таблица 3.27 (окончание)

194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169
108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198
232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14
97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148
155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104
65	153	45	15	176	84	187	22

Эта операция также аналогична (но с другой таблицей замен) операции SubBytes алгоритма Rijndael — см. рис. 3.8.

3. Зависящая от ключа перестановка  $\pi$ . Использует 5-байтный фрагмент расширенного ключа  $k[r,1]$ , причем процедура расширения ключа гарантирует, что значение каждого байта  $k[r,1]$  лежит в диапазоне от 1 до 24 включительно. Перестановка  $\pi$  выполняется в 5 этапов.
- На первом этапе производится перестановка строк массива данных. Для этого по первому байту ключа  $k[r,1]$  (обозначим его значение как  $x$ ) из таблицы (см. табл. 3.28) выбирается константа  $t_x$  (указаны шестнадцатеричные значения).

Таблица 3.28

$x$	$t_x$	$x$	$t_x$	$x$	$t_x$
1	78	9	87	17	6C
2	36	10	63	18	D8
3	2D	11	D2	19	E1
4	C9	12	39	20	C6
5	93	13	8D	21	27
6	72	14	1E	22	B1
7	4B	15	E4	23	4E
8	9C	16	B4	24	1B

Константа  $t_x$  представляется в виде четырех 2-битных значений  $t_x[0]...t_x[3]$ ; подбор констант  $t_x$  гарантирует, что значения  $t_x[0]...t_x[3]$  представляют собой последовательность из неповторяющихся значений от 0 до 3 включительно; например, для  $x = 1$ :

$$t_x[0] = 1,$$

$$t_x[1] = 3,$$

$$t_x[2] = 2,$$

$$t_x[3] = 0.$$

Перестановка строк состоит в том, что  $i$ -я строка массива данных заменяется строкой, номер которой определяется значением  $t_x[i]$  (см. пример для  $x = 1$  на рис. 3.84).

- На этапах со второго по пятый выполняется перестановка байтов одного из столбцов массива данных: на  $i$ -м этапе переставляются байты  $i - 2$ -го столбца, в зависимости от значения  $t_x$ , выбиравшегося из приведенной выше таблицы согласно  $x$  — значению  $i$ -го байта ключа  $k[r,1]$ . Перестановка байтов выполняется абсолютно аналогично описанной выше перестановке строк — см. пример для этапа 2 и  $x = 2$  на рис. 3.85.

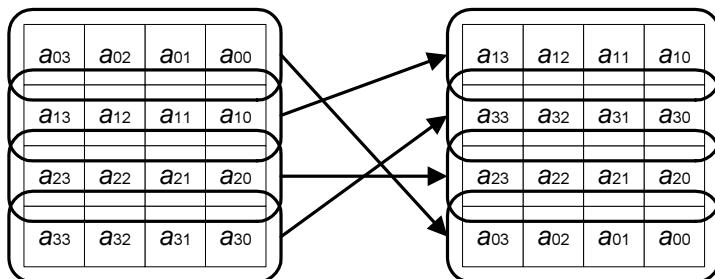


Рис. 3.84. Первый этап перестановки  $\pi$  алгоритма Grand Cru

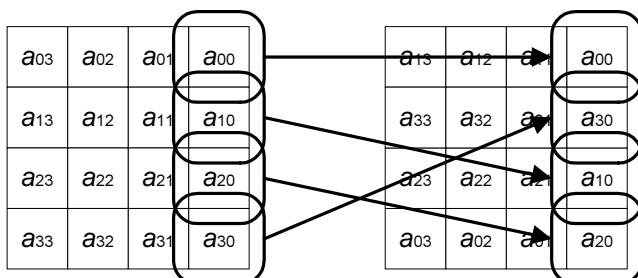


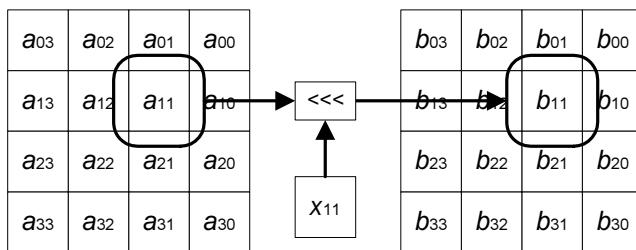
Рис. 3.85. Второй этап перестановки  $\pi$  алгоритма Grand Cru

## Описание алгоритмов

4. Операция  $\theta$  — умножение обрабатываемой таблицы на фиксированную матрицу  $H$  — аналогична операции MixColumns алгоритма Rijndael (см. рис. 3.10). Матрица  $H$  унаследована от алгоритма Rijndael, в котором она определена так, как показано в табл. 3.29 [153].

Таблица 3.29

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

Рис. 3.86. Операция  $\beta$  алгоритма Grand Cru

5. Зависящее от ключа вращение (операция  $\beta$ ) каждого байта массива на переменное число битов, определяемое значениями соответствующих битов фрагмента ключа  $r$ -го раунда  $k[r,2]$  (рис. 3.86):

$$b_{i,j} = a_{i,j} \ll\ll x_{i,j},$$

где в качестве  $x_{i,j}$  используются значения трех последовательных битов фрагмента  $k[r,2]$  в соответствии с табл. 3.30.

Таблица 3.30

i	j	Используемые биты $k[r,2]$
3	3	45, 46, 47
3	2	42, 43, 44
3	1	39, 40, 41

Таблица 3.30 (окончание)

i	j	Используемые биты k[r,2]
3	0	36, 37, 38
2	3	33, 34, 35
2	2	30, 31, 32
2	1	27, 28, 29
2	0	24, 25, 26
1	3	21, 22, 23
1	2	18, 19, 20
1	1	15, 16, 17
1	0	12, 13, 14
0	3	9, 10, 11
0	2	6, 7, 8
0	1	3, 4, 5
0	0	0, 1, 2

Таким образом, раунд алгоритма состоит из последовательно выполняемых операций:  $\sigma \rightarrow \gamma \rightarrow \pi \rightarrow \theta \rightarrow \beta$ . Кроме того, перед первым по порядку (при зашифровывании раунды нумеруются от 8 до 0) раундом выполняются операции  $\psi$  и  $\upsilon$  (составляющие предварительное преобразование), которые будут подробно описаны далее. По завершении последнего раунда выполняется последовательность операций:  $\sigma \rightarrow \gamma \rightarrow \pi \rightarrow \beta \rightarrow \sigma \rightarrow \upsilon^{-1} \rightarrow \psi$ , где  $\upsilon^{-1}$  — операция, обратная к  $\upsilon$  (также подробно описана далее). Происхождение фрагментов расширенного ключа, используемых в ключевых операциях предварительного и заключительного преобразований, будет приведено в описании процедуры расширения ключа.

Операции  $\psi$ , выполняемые перед первым и после заключительного раунда алгоритма, представляют собой входное и выходное отбеливание данных соответственно. Эти операции накладывают на обрабатываемый блок данных соответствующие фрагменты ключа (обозначим их  $kwi$  и  $kwo$ ) с помощью операции сложения по модулю 256 (рис. 3.87). Например, для входного отбеливания:

$$b_{i,j} = a_{i,j} + kwi_{i,j} \bmod 256.$$

## Описание алгоритмов

$a_{03}$	$a_{02}$	$a_{01}$	$a_{00}$
$a_{13}$	$a_{12}$	$a_{11}$	$a_{10}$
$a_{23}$	$a_{22}$	$a_{21}$	$a_{20}$
$a_{33}$	$a_{32}$	$a_{31}$	$a_{30}$

+

$k_{03}$	$k_{02}$	$k_{01}$	$k_{00}$
$k_{13}$	$k_{12}$	$k_{11}$	$k_{10}$
$k_{23}$	$k_{22}$	$k_{21}$	$k_{20}$
$k_{33}$	$k_{32}$	$k_{31}$	$k_{30}$

=

$b_{03}$	$b_{02}$	$b_{01}$	$b_{00}$
$b_{13}$	$b_{12}$	$b_{11}$	$b_{10}$
$b_{23}$	$b_{22}$	$b_{21}$	$b_{20}$
$b_{33}$	$b_{32}$	$b_{31}$	$b_{30}$

Рис. 3.87. Операция  $\psi$  алгоритма Grand Cru

Операция  $\psi$  (бесключевое рассеивание данных) выполняется в два этапа:

1. В цикле по  $n$  от 0 до 15 выполняется следующее действие:

$$b_{n+1 \bmod 16} = a_{n+1 \bmod 16} \oplus S(a_n),$$

где  $a_x$  и  $b_x$  — значения  $x$ -го байта данных до и после выполнения текущего действия соответственно (в этом случае блок данных представляется не в виде таблицы, а в виде последовательности байтов).

2. В цикле по  $n1$  от 0 до 15 выполняется цикл по  $n2$  от 0 до 15, в котором выполняется следующее действие:

$$b_{n2} = a_{n2} \oplus S(a_{n1}).$$

Это действие не выполняется при  $n2 = n1$ .

Обратная к  $\psi$  операция  $\psi^{-1}$  определена так:

1. В цикле по  $n1$  от 15 до 0 выполняется цикл по  $n2$  от 0 до 15 (кроме шага, в котором  $n2 = n1$ ), в котором выполняется следующее действие:

$$b_{n2} = a_{n2} \oplus S(a_{n1}).$$

2. В цикле по  $n$  от 15 до 0 выполняется следующее действие:

$$b_{n+1 \bmod 16} = a_{n+1 \bmod 16} \oplus S(a_n).$$

## Расшифровывание

Расшифровывание выполняется применением обратных операций в обратной последовательности:

1. Обратное заключительное преобразование:  $\psi^{-1} \rightarrow \nu \rightarrow \sigma \rightarrow \beta^{-1} \rightarrow \pi^{-1} \rightarrow \gamma^{-1} \rightarrow \sigma$ , где  $\psi^{-1}$ ,  $\beta^{-1}$ ,  $\pi^{-1}$  и  $\gamma^{-1}$  — обратные операции к  $\psi$ ,  $\beta$ ,  $\pi$  и  $\gamma$  соответственно и будут описаны далее (обратная операция к  $\sigma$  эквивалентна самой операции  $\sigma$ ).

2. 9 раундов (с 0-го по 8-й), в каждом из которых выполняется следующая последовательность действий:  $\beta^{-1} \rightarrow \theta^{-1} \rightarrow \pi^{-1} \rightarrow \gamma^{-1} \rightarrow \sigma$ , где  $\theta^{-1}$  — операция, обратная к  $\theta$ .
3. Обратное предварительное преобразование, состоящее из двух операций:  $\psi^{-1}$  и  $\nu^{-1}$ .

Рассмотрим обратные операции, используемые при расшифровывании.

Операция  $\psi^{-1}$  производит наложение фрагментов расширенного ключа  $kwo$  и  $kwi$  путем вычитания по модулю 256, например:

$$b_{i,j} = a_{i,j} - kwi_{i,j} \bmod 256.$$

Операция  $\beta^{-1}$  выполняется аналогично операции  $\beta$ , однако вращение каждого байта блока данных выполняется не влево, а вправо:

$$b_{i,j} = a_{i,j} >>> x_{i,j}.$$

Операция  $\pi^{-1}$  — обратная перестановка относительно операции  $\pi$ . Ее этапы выполняются в обратной последовательности; кроме того, на каждом этапе производится обратная перестановка.

Операция  $\gamma^{-1}$  аналогична прямой операции  $\gamma$ , но замена выполняется с помощью обратной таблицы  $S^{-1}$ .

Операция  $\theta^{-1}$  представляет собой умножение блока данных на обратную к  $H$  матрицу  $H^{-1}$ , которая приведена в табл. 3.31.

**Таблица 3.31**

E	B	D	9
9	E	B	D
D	9	E	B
B	D	9	E

## Процедура расширения ключа

Как видно из описания алгоритма шифрования, для выполняемых преобразований требуется достаточно большое количество фрагментов расширенного ключа различного размера. Поэтому процедура расширения ключа достаточно сложна. Стоит также отметить, что описание процедуры расширения ключа в документе [102] допускает неоднозначное толкование некоторых преобразований. К сожалению, не удалось найти каких-либо альтернативных источников информации об этой процедуре. Поэтому процедура расширения ключа изложена в понимании таковой автором данной книги.

Прежде всего, вычисляется последовательность промежуточных ключей  $K_0 \dots K_3$ :

1. 128-битный исходный ключ шифрования представляется в виде четырех столбцов, которые обозначаются  $K_0' \dots K_3'$ .
2. Формируется последовательность столбцов  $K_4' \dots K_{15}'$  следующим образом:  

$$K_i' = K_{i-4}' \oplus G(R(K_{i-1}') \oplus c_{i/4}), \text{ если } i \text{ кратно четырем},$$

$$\text{иначе } K_i' = K_{i-4}' \oplus K_{i-1}'.$$

Функция  $R()$  представляет собой побайтное вращение обрабатываемого столбца на 1 байт вверх.

Функция  $G()$  обрабатывает каждый байт столбца с помощью описанной выше таблицы замен  $S$ , т. е.  $x$ -й байт столбца  $a_x$  заменяется значением  $S(a_x)$ .

Модифицирующие константы  $c_x$  взяты из алгоритма Rijndael, где они определены согласно табл. 3.32.

**Таблица 3.32**

$x$	$c_x$
1	1
2	2
3	4

В принципе, в алгоритме может использоваться ключ шифрования, больший 128 битов, но не превышающий 512 битов и кратный 32 битам. В этом случае последовательность  $K_0 \dots K_{15}'$  заполняется аналогичным образом 32-битными фрагментами ключа шифрования, насколько это возможно, а недостающие столбцы рассчитываются согласно приведенным выше формулам.

3. Из столбцов, рассчитанных на предыдущем шаге, формируется последовательность промежуточных ключей  $K_0 \dots K_3$ :

$$K_i = (K_{4i+3}' \bullet K_{4i+2}' \bullet K_{4i+1}' \bullet K_{4i}'),$$

где  $\bullet$  — операция конкатенации.

Совокупность преобразований, используемых для получения промежуточного ключа  $K_i$  из исходного ключа шифрования (обозначим его  $KU$ ), обозначим следующим образом:

$$K_i = F(KU, i).$$

В дальнейшем с помощью аналогичных преобразований вычисляются необходимые подключи:

- Для операции  $\sigma$  требуются 9 128-битных подключей  $k[0,0]...k[8,0]$  — по одному для каждого раунда алгоритма, а также 2 подключа ( $k[9,0]$  и  $k[10,0]$ ), поочередно используемые в заключительном преобразовании (в обратном заключительном преобразовании эти два подключа используются в обратном порядке). Данные 11 подключей вычисляются на основе промежуточного ключа  $K_0$  следующим образом:

$$k[i,0] = F(K_0, i).$$

- Операция  $\psi$  использует два 128-битных подключа  $kwi$  и  $kwo$ , которые вычисляются так:

$$kwi = F(K_3, 0);$$

$$kwo = F(K_3, 1).$$

При расшифровывании  $kwi$  и  $kwo$  применяются в обратном порядке.

- Для операции  $\beta$  требуются 48-битные подключи  $k[0,2]...k[8,2]$  — по одному для каждого раунда, а также  $k[9,2]$  для заключительного преобразования. Эти подключи вычисляются следующим образом: для каждого  $i = 0...9$  выполняются такие действия:

- вычисляется еще один промежуточный ключ:

$$K\beta = F(K_2, i);$$

- вычисляется  $k[i,2]$ :

$$k[i,2] = LSB_3(K\beta_{15}) \bullet LSB_3(K\beta_{14}) \bullet \dots \bullet LSB_3(K\beta_0),$$

где:

- $K\beta_x$  —  $x$ -й байт промежуточного ключа  $K\beta$ ;
- $LSB_3(X)$  — три младших бита значения  $X$ .

- Для операции  $\pi$  требуется 10 40-битных подключей:  $k[0,1]...k[8,1]$  для 9 раундов алгоритма и  $k[9,1]$  для заключительного преобразования. Эти подключи для  $i = 0...9$  вычисляются так:

- вычисляется промежуточный ключ:

$$K\pi = F(K_1, i);$$

- $K\pi$  представляется в виде байтового массива из 16 элементов:  $k\pi_0...k\pi_{15}$ ;
- из данного массива набираются первые 5 байтов из тех, значения которых попадают в требуемый диапазон: от 1 до 24; если в массиве таких байтов меньше пяти, недостаточное количество набирается с начала

массива, в качестве значения байта ключа берется остаток от деления значения используемого байта массива на 16 (вместо 0 используется значение 16).

## Достоинства и недостатки алгоритма

Как видно из приведенного выше описания, алгоритм Grand Cru является весьма сложным; причем сложным и запутанным выглядит как шифрование, так и процедура расширения ключа. Из избыточной сложности алгоритма следует тот факт, что алгоритм Grand Cru оказался самым медленным (причем с огромным отрывом по быстродействию от большей части других алгоритмов) среди всех участников конкурса [311].

В процессе изучения алгоритма в рамках конкурса NESSIE в нем не было обнаружено каких-либо слабостей, и, соответственно, каких-либо атак на данный алгоритм [312]. Тем не менее, эксперты конкурса посчитали, что возможна высокая (но недоказанная) криптостойкость данного алгоритма не компенсирует ужасающее низкой скорости шифрования, поэтому Grand Cru не был выбран во второй раунд конкурса [308].

## 3.21. Алгоритм Hierocrypt-L1

Алгоритм Hierocrypt-L1 был предложен на конкурс NESSIE известной японской корпорацией Toshiba. Авторы алгоритма: Кенжи Охкума (Kenji Ohkuma), Фумихико Сано (Fumihiko Sano), Хирофуми Муратани (Hirofumi Muratani), Масахико Мотояма (Masahiko Motoyama) и Шиничи Кавамура (Shinichi Kawamura).

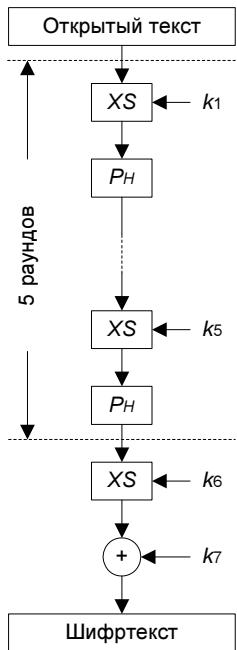
Алгоритм шифрует данные 64-битными блоками с использованием 128-битных ключей шифрования.

## Структура алгоритма

Алгоритм Hierocrypt-L1 представляет собой SP-сеть. Его структура представлена на рис. 3.88. Всего алгоритм выполняет 6 раундов преобразований (причем последний раунд отличается от предыдущих), после которых производится заключительное наложение материала [287, 288].

В каждом из первых пяти раундов над обрабатываемым 64-битным блоком данных выполняются следующие операции:

1. Операция  $XS$ , которая будет подробно описана далее.



**Рис. 3.88.** Структура алгоритма Hierocrypt-L1

2. Операция  $P_H$ , представляющая собой умножение блока на фиксированную матрицу  $M_H$ :

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \end{pmatrix} = M_H * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix},$$

где:

- $x_1 \dots x_8$  — 8-битные фрагменты входного значения;
- $y_1 \dots y_8$  — 8-битные фрагменты результата;

- умножение выполняется в поле  $GF(2^8)$ ;
- матрица  $M_H$  определена согласно табл. 3.33.

Таблица 3.33

1	0	1	0	1	1	1	0
1	1	0	1	1	1	1	1
1	1	1	0	0	1	1	1
0	1	0	1	1	1	0	1
1	1	0	1	0	1	0	1
1	1	1	0	1	0	1	0
1	1	1	1	1	1	0	1
1	0	1	0	1	0	1	1

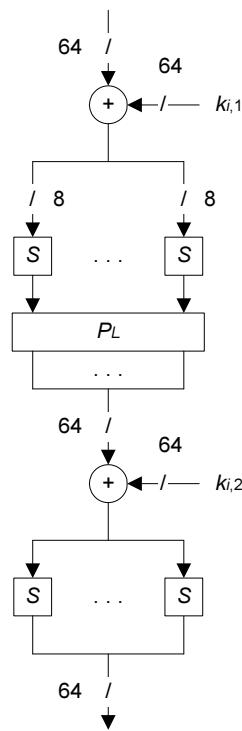


Рис. 3.89. Операция XS алгоритма Hierocrypt-L1

Операция  $XS$ , фактически, представляет собой вложенную SP-сеть. Перечислим выполняемые ею преобразования (рис. 3.89):

1. Наложение 64-битного фрагмента ключа раунда:

$$Y = X \oplus K_{i,1},$$

где:

- $X$  и  $Y$  — соответственно, входное и выходное значения текущей операции;
- $K_{i,1}$  — первый фрагмент ключа  $i$ -го раунда.

2. Табличная замена  $S$ , параллельно заменяющая значения каждого байта результата предыдущей операции согласно табл. 3.34.

**Таблица 3.34**

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

Таким образом, таблица заменяет значение 0 на 7, 1 — на FC, 2 — на 55 и т. д.

3. 64-битный блок делится на два 32-битных фрагмента, параллельно обрабатываемые процедурой  $P_L$ , которая выполняет над субблоком умножение на фиксированную матрицу  $M_L$ :

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = M_L * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix},$$

где  $x_1 \dots x_4$  и  $y_1 \dots y_4$  — 8-битные фрагменты, соответственно, входного и выходного значения, а матрица  $M_L$  приведена в табл. 3.35.

**Таблица 3.35**

C4	65	C8	8B
8B	C4	65	C8
C8	8B	C4	65
65	C8	8B	C4

Умножение выполняется в поле  $GF(2^8)$  с образующим полиномом  $x^8 + x^6 + x^5 + x + 1$ . Причем фрагменты входного значения ( $x_n$ ) и элементы матрицы ( $a$ ) преобразуются в элементы поля  $GF(2^8)$  различным образом:

$$x_n : \sum_{i=1}^8 x_{ni} 2^{8-i};$$

$$a : \sum_{i=0}^7 a_i 2^i.$$

4. Наложение второго 64-битного фрагмента ключа раунда:

$$Y = X \oplus K_{i,2}.$$

5. Повторное выполнение описанной выше табличной замены  $S$ .

Последний раунд отличается от предыдущих тем, что в нем производится только операция  $XS$ ; операция  $P_H$  в последнем раунде не выполняется.

Кроме того, как было сказано выше, после шестого раунда производится наложение на блок данных 64-битного фрагмента расширенного ключа  $K_7$ .

Расшифровывание выполняется применением обратных операций в обратной последовательности. При этом в операциях  $P_H^{-1}$  и  $P_L^{-1}$  используются матрицы, приведенные в табл. 3.36 и 3.37 соответственно.

Таблица 3.36

1	0	1	1	1	1	0	1
0	1	0	1	1	1	1	0
1	0	1	0	1	1	1	1
0	1	1	0	1	0	1	0
1	0	1	0	0	1	0	1
1	1	0	1	1	0	1	0
1	1	1	0	1	1	0	1
0	1	0	1	1	0	1	1

Таблица 3.37

82	C4	34	F6
F6	82	C4	34
34	F6	82	C4
C4	34	F6	82

## Процедура расширения ключа

Процедура расширения ключа здесь сложнее, чем во многих других алгоритмах. Она состоит из двух этапов, на которых происходят:

1. Генерация промежуточных ключей на основе ключа шифрования.
2. Вычисление расширенного ключа на основе промежуточных ключей.

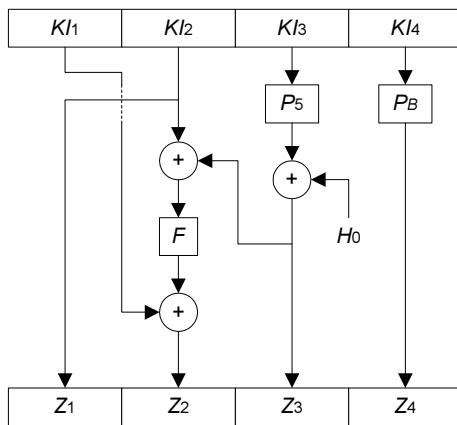


Рис. 3.90. Вычисление промежуточного ключа в алгоритме Hierocrypt-L1

Рассмотрим первый из этих этапов. Здесь выполняются следующие операции (рис. 3.90):

$$Z_3 = P_5(KI_3) \oplus H_0;$$

$$Z_4 = P_B(KI_4);$$

Описание алгоритмов

$$Z_1 = KI_2;$$

$$Z_2 = KI_1 \oplus F(KI_2 \oplus Z_3),$$

где:

- $KI_1 \dots KI_4$  — 32-битные фрагменты исходного ключа шифрования;
- $Z_1 \dots Z_4$  — 32-битные фрагменты промежуточного ключа;
- $H_0$  — одна из констант, используемых в процедуре расширения ключа (см. далее);
- $P_5$  — операция умножения в конечном поле  $GF(2^8)$  32-битного значения на фиксированную матрицу  $M_5$ :

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = M_5 * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix},$$

где  $x_1 \dots x_4$  и  $y_1 \dots y_4$  — 8-битные фрагменты, соответственно, входного и выходного значения, а матрица  $M_5$  определена согласно табл. 3.38;

- $P_B$  — аналогичная операция умножения на матрицу  $M_B$ , определенную согласно табл. 3.39.

*Таблица 3.38*

1	0	1	0
1	1	0	1
1	1	1	0
0	1	0	1

*Таблица 3.39*

0	1	0	1
1	0	1	0
1	1	0	1
1	0	1	1

Функция  $F()$  выполняется в несколько шагов:

1. Входное значение разбивается на 4 фрагмента по 8 битов.
2. Каждый из фрагментов «прогоняется» через описанную выше таблицу замен  $S$ .
3. Выходное значение функции  $F()$  — результат умножения выходного значения предыдущего шага на матрицу  $M_8$  (аналогично описанной выше операции  $P_5$ ), которая определена в табл. 3.40.

Таблица 3.40

1	0	1	0
0	1	0	1
0	1	1	1
1	0	1	1

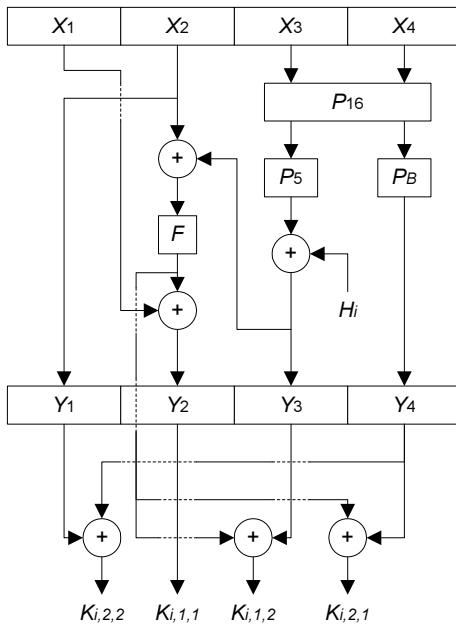


Рис. 3.91. Прямой раунд процедуры расширения ключа в алгоритме Hierocrypt-L1

Второй этап процедуры расширения ключа — вычисление необходимого количества подключей на основе промежуточного ключа. Данный процесс выполняется в 7 раундов, в первых четырех из которых выполняются следующие преобразования (рис. 3.91):

$$(T_1 \bullet T_2) = P_{16}(X_3 \bullet X_4);$$

$$Y_3 = P_5(T_1) \oplus H_i;$$

$$Y_4 = P_B(T_2);$$

$$Y_1 = X_2;$$

$$Y_2 = X_1 \oplus F(X_2 \oplus Z_3),$$

где:

- $X_1 \dots X_4$  и  $Y_1 \dots Y_4$  — соответственно, 32-битные фрагменты входного и выходного значения; в первом раунде в качестве входного значения используются  $Z_1 \dots Z_4$ , в последующих — выходные значения предыдущего раунда;
- $T_1$  и  $T_2$  — временные переменные;
- $P_{16}$  — аналогичное используемому в функции  $F()$  умножению на матрицу  $M_8$ ; однако обрабатываемое данной операцией 64-битное значение делится на 4 16-битных фрагмента (а не на 8-битные); соответственно, умножение выполняется в поле  $GF(2^{16})$ ;
- $H_1 \dots H_4$  — константы, используемые в процедуре расширения ключа (согласно с упомянутой выше  $H_0$ ):

$$H_0 = 5A827999;$$

$$H_1 = 6ED9EBA1;$$

$$H_2 = 8F1BBCDC;$$

$$H_3 = CA62C1D6;$$

$$H_4 = F7DEF58A.$$

Эти константы — шестнадцатеричная запись дробной части иррациональных чисел  $\sqrt{n}/4$ ;  $n = 2, 3, 5, 10, 15$  для  $H_0$ ,  $H_1$ ,  $H_2$ ,  $H_3$  и  $H_4$  соответственно.

Фактически каждый из первых четырех раундов идентичен первому этапу расширения ключа, за исключением присутствия операции  $P_{16}$  и других используемых констант  $H_i$ . Ключи шифрования для  $i$ -го раунда шифрования вычисляются в конце  $i$ -го раунда этапа 2 процедуры расширения ключа:

$$K_{i,1,1} = Y_2;$$

$$K_{i,1,2} = Y_2 \oplus X_1 \oplus Y_3;$$

$$K_{i,2,1} = Y_2 \oplus X_1 \oplus Y_4;$$

$$K_{i,2,2} = Y_1 \oplus Y_4,$$

где  $K_{i,j,1}$  и  $K_{i,j,2}$  — 32-битные «половинки» подключей  $i$ -го раунда  $K_{i,1}$  и  $K_{i,2}$  (их применение было описано выше).

Впоследствии выполняются 3 раунда (с 5-го по 7-й), которые являются обратными первым четырем — таким образом, для  $n=1 \dots 3$  выполняется следующее соотношение фрагментов расширенного ключа:

$$K_{n,j} = K_{8-n,j}.$$

В каждом обратном раунде выполняются следующие преобразования (рис. 3.92):

$$Y_1 = X_2 \oplus F(X_1 \oplus X_3);$$

$$Y_2 = X_1;$$

$$T_1 = P_B(X_3 \oplus H_{9-i});$$

$$T_2 = P_5(X_4);$$

$$(Y_3 \bullet Y_4) = P_{16}^{-1}(T_1 \bullet T_2).$$

Операция  $P_{16}^{-1}$  является обратной к операции  $P_{16}$ , т. е. по описанным выше правилам выполняет умножение 16-битных фрагментов входного значения на матрицу, приведенную в табл. 3.41.

Таблица 3.41

1	1	1	0
1	1	0	1
0	1	1	0
1	0	0	1

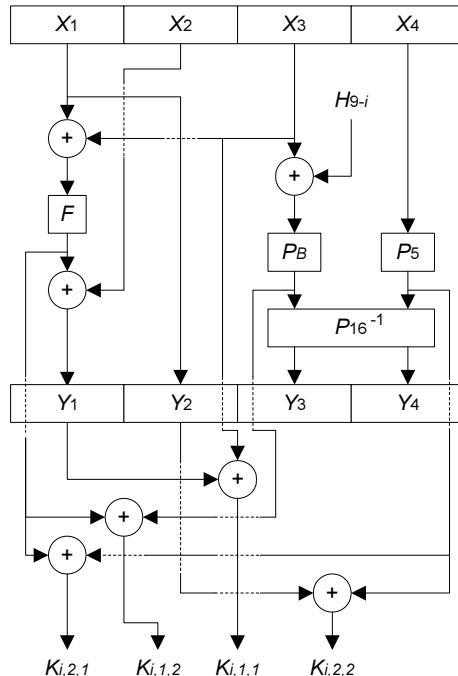


Рис. 3.92. Обратный раунд процедуры расширения ключа в алгоритме Hierocrypt-L1

Ключи раундов шифрования 5–7 вычисляются следующим образом:

$$K_{i,1,1} = Y_1 \oplus X_3;$$

$$K_{i,1,2} = Y_1 \oplus X_2 \oplus T_1;$$

$$K_{i,2,1} = Y_1 \oplus X_2 \oplus T_2;$$

$$K_{i,2,2} = Y_2 \oplus T_2.$$

Процедура расширения ключа позволяет генерировать подключи «на лету», однако только при зашифровывании — при расшифровывании придется сначала вычислить полный набор подключей.

## Достоинства и недостатки алгоритма

Эксперты конкурса NESSIE после изучения алгоритма Hierocrypt-L1, а также различных посвященных ему материалов пришли к выводу, что этот алгоритм имеет исключительно медленную процедуру расширения ключа, что существенно ограничивает возможную область применения алгоритма [308]. Кроме того, были обнаружены уязвимости в процедуре расширения ключа, а также несколько вариантов атак на версии алгоритма с усеченным количеством раундов [307]. Эти недостатки алгоритма Hierocrypt-L1 не позволили ему выйти во второй раунд конкурса NESSIE.

## 3.22. Алгоритм Hierocrypt-3

### Отличия от Hierocrypt-L1

Алгоритм Hierocrypt-3 разработан тем же коллективом разработчиков из японской корпорации Toshiba, которые создали рассмотренный в предыдущем разделе алгоритм Hierocrypt-L1. Несмотря на то, что алгоритмы Hierocrypt-L1 и Hierocrypt-3 обрабатывают блоки данных различного размера: 64 и 128 битов соответственно, — они используют практически одни и те же преобразования, поэтому различия между ними минимальны [287, 288, 289].

- Фрагменты расширенного ключа  $i$ -го раунда  $K_{i,1}$  и  $K_{i,2}$  имеют размер по 128 битов (а не по 64 бита) для их наложения на 128-битный блок данных в операции  $XS$  аналогично алгоритму Hierocrypt-L1.

□ Поскольку размер блока увеличился в 2 раза, операция  $P_H$  и лежащая в ее основе матрица  $M_H$  (см. табл. 3.42) выглядят иначе:

$$P_H : \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \\ y_{15} \\ y_{16} \end{pmatrix} = M_H^* \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \end{pmatrix}.$$

Таблица 3.42

1	0	1	0	1	0	1	0	1	1	0	1	1	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1	0	0	1	1	1	1
1	1	1	0	1	1	1	0	1	1	1	1	0	0	1	1	1
0	1	0	1	0	1	0	1	1	0	1	0	1	1	1	1	0
1	1	1	1	1	0	1	0	1	0	1	0	1	1	0	1	1
0	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	0
0	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1
1	1	1	0	0	1	0	1	0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	1	1	1	0	1	0	1	0	1	0	0
1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	1	1
1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0
1	0	1	0	1	1	1	0	0	1	0	1	0	1	0	1	1
1	0	1	0	1	1	0	1	1	1	1	1	1	0	1	0	0
1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1	1
1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	1	0
0	1	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0

- Увеличилось количество раундов алгоритма (см. далее).
- Существенно изменена процедура расширения ключа.

## Процедура расширения ключа

В отличие от Hierocrypt-L1, в котором использовались только 128-битные ключи, алгоритм Hierocrypt-3 позволяет использовать ключи нескольких размеров: 128, 192 и 256 битов. Поэтому процедура расширения ключа выглядит несколько более сложной. Она состоит из трех этапов, на которых происходят:

1. Дополнение ключа шифрования до 256-битного значения.
2. Генерация промежуточных ключей на основе дополненного ключа.
3. Вычисление расширенного ключа на основе промежуточных ключей.

Дополнение ключа выполняется следующим образом: сначала ключ шифрования  $KI$  разбивается на 64-битные фрагменты:

- $KI_1$  и  $KI_2$  для 128-битного ключа;
- $KI_1 \dots KI_3$  для 192-битного ключа;
- $KI_1 \dots KI_4$  для 256-битного ключа.

Затем формируется дополненный ключ  $KP$ , состоящий из фрагментов  $KP_1 \dots KP_4$ , которые инициализируются согласно табл. 3.43.

Таблица 3.43

Размер ключа в битах	$KP_1$	$KP_2$	$KP_3$	$KP_4$
128	$KI_1$	$KI_2$	$KI_1$	$H_3 \bullet H_2$
192	$KI_1$	$KI_2$	$KI_3$	$H_2 \bullet H_3$
256	$KI_1$	$KI_2$	$KI_3$	$KI_4$

Константы  $H_1 \dots H_4$  приведены в описании алгоритма Hierocrypt-L1 (см. разд. 3.21).

Генерация промежуточных ключей выполняется весьма похоже на Hierocrypt-L1 с учетом того факта, что фигурирующие в формулах величины  $KP_1 \dots KP_4$  и  $Z_1 \dots Z_4$  являются 64-битными:

$$\begin{aligned} Z_3 &= P_{5E}(KP_3) \oplus (H_1 \bullet H_0); \\ Z_4 &= P_{5E}(KP_4); \end{aligned}$$

$$Z_1 = KP_2;$$

$$Z_2 = KP_1 \oplus F(KP_2 \oplus Z_3).$$

Операция  $P_{5E}$  — это, фактически, операция  $P_5$  алгоритма Hierocrypt-L1, адаптированная для вычислений с 64-битными операндами:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = M_5 * \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix};$$

$$\begin{pmatrix} y_5 \\ y_6 \\ y_7 \\ y_8 \end{pmatrix} = M_{5A} * \begin{pmatrix} x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix},$$

где  $x_1 \dots x_8$  и  $y_1 \dots y_8$  — 8-битные фрагменты, соответственно, входного и выходного значения; здесь используется матрица  $M_5$  из алгоритма Hierocrypt-L1, а матрица  $M_{5A}$  приведена в табл. 3.44.

**Таблица 3.44**

1	1	1	1
0	1	1	1
0	0	1	1
1	1	1	0

Умножение выполняется в поле  $GF(2^8)$ .

Количество раундов алгоритма по сравнению с Hierocrypt-L1 осталось прежним (6 раундов) только для 128-битного ключа; шифрование со 192-битным ключом выполняется в 7 раундов, с 256-битным — в 8 раундов. Соответственно, изменилось количество раундов и в процедуре расширения ключа.

Как и в алгоритме Hierocrypt-L1, вычисление расширенного ключа на основе промежуточных ключей выполняется с помощью нескольких «прямых» раундов и нескольких «обратных» раундов процедуры расширения ключа с участием модифицирующих констант  $H_1 \dots H_4$ . Выбор типа раунда

## Описание алгоритмов

процедуры и соответствующей константы зависит от размера ключа и определяется по следующим таблицам:

- для 128-битного ключа — см. табл. 3.45;
- для 192-битного ключа — см. табл. 3.46;
- для 256-битного ключа — см. табл. 3.47.

**Таблица 3.45**

Номер раунда	Тип раунда	Константа $G$
1	прямой	$G_0$
2	прямой	$G_1$
3	прямой	$G_2$
4	прямой	$G_3$
5	обратный	$G_3$
6	обратный	$G_2$
7	обратный	$G_1$

**Таблица 3.46**

Номер раунда	Тип раунда	Константа $G$
1	прямой	$G_1$
2	прямой	$G_0$
3	прямой	$G_3$
4	прямой	$G_2$
5	обратный	$G_2$
6	обратный	$G_3$
7	обратный	$G_0$
8	обратный	$G_1$

Таблица 3.47

Номер раунда	Тип раунда	Константа $G$
1	прямой	$G_4$
2	прямой	$G_0$
3	прямой	$G_2$
4	прямой	$G_1$
5	прямой	$G_3$
6	обратный	$G_3$
7	обратный	$G_1$
8	обратный	$G_2$
9	обратный	$G_0$

Символами  $G_0 \dots G_4$  в приведенных выше таблицах обозначены 64-битные константы, получаемые путем конкатенации 32-битных констант  $H_0 \dots H_3$ :

$$G_0 = H_3 \bullet H_0;$$

$$G_1 = H_2 \bullet H_1;$$

$$G_2 = H_1 \bullet H_3;$$

$$G_3 = H_0 \bullet H_2;$$

$$G_4 = H_2 \bullet H_3.$$

Прямой и обратный раунды процедуры расширения ключа также похожи на аналогичные раунды в алгоритме Hierocrypt-L1. Прямой раунд состоит из следующей последовательности операций:

$$(T_1 \bullet T_2) = P_{32}(X_3 \bullet X_4);$$

$$Y_3 = P_{5E}(T_1) \oplus G;$$

$$Y_4 = P_{5E}(T_2);$$

$$Y_1 = X_2;$$

$$Y_2 = X_1 \oplus F(X_2 \oplus Y_3).$$

## Описание алгоритмов

После выполнения каждого прямого раунда происходит вычисление 128-битных подключей  $i$ -го раунда шифрования:  $K_{i,1}$  и  $K_{i,2}$ ; их 64-битные фрагменты  $K_{i,j,1}$  и  $K_{i,j,2}$  вычисляются следующим образом:

$$T = F(X_2 \oplus Y_3);$$

$$K_{i,1,1} = X_1 \oplus T;$$

$$K_{i,1,2} = Y_3 \oplus T;$$

$$K_{i,2,1} = Y_4 \oplus T;$$

$$K_{i,2,2} = Y_1 \oplus Y_4,$$

где  $T$  — временная переменная.

Обратный раунд определен так:

$$Y_1 = X_2 \oplus F(X_1 \oplus X_3);$$

$$Y_2 = X_1;$$

$$T_1 = P_{BE}(X_3 \oplus G);$$

$$T_2 = P_{BE}(X_4);$$

$$(Y_3 \bullet Y_4) = P_{32}^{-1}(T_1 \bullet T_2).$$

Операция  $P_{BE}$  аналогично описанной выше  $P_{5E}$  разбивает обрабатываемый 64-битный фрагмент данных на две части по 32 бита, каждая из которых умножается на одну из приведенных матриц (см. табл. 3.48 и 3.49).

**Таблица 3.48**

0	1	0	1
1	0	1	0
1	1	0	1
1	0	1	1

**Таблица 3.49**

1	1	0	0
0	1	1	0
1	0	1	1
1	0	0	1

Фрагменты ключа шифрования по истечении каждого обратного раунда вычисляются следующим образом:

$$T = F(X_1 \oplus X_3);$$

$$K_{i,1,1} = Y_1 \oplus X_3;$$

$$K_{i,1,2} = T_1 \oplus T;$$

$$K_{i,2,1} = T_2 \oplus T;$$

$$K_{i,2,2} = Y_2 \oplus T_2.$$

Используемая в процедуре расширения ключа функция  $F$  тоже несколько отличается от таковой в алгоритме Hierocrypt-L1 тем, что разбивает обрабатываемые данные на 8 (а не на 4) 8-битных фрагментов, каждый из которых обрабатывается функцией  $P_{16}$ , описанной в разд. 3.21.

И, наконец, функции  $P_{32}$  и  $P_{32}^{-1}$  аналогичны функциям  $P_{16}$  и  $P_{16}^{-1}$  соответственно; однако при обработке они делят обрабатываемые данные на 4 фрагмента по 32 бита (а не по 16 битов); их умножение выполняется в поле  $\text{GF}(2^{32})$ .

## Криptoанализ алгоритма

Недостатки алгоритмов Hierocrypt-3 и Hierocrypt-L1 практически идентичны:

- весьма медленно выполняется процедура расширения ключа, к тому же она имеет уязвимости [308];
- предложено несколько атак на варианты алгоритма Hierocrypt-3 с усеченным количеством раундов, что говорит о недостаточном запасе криптоустойчивости этого алгоритма [45, 116, 279].

В результате алгоритм Hierocrypt-3 не был выбран во второй раунд конкурса NESSIE [308].

## 3.23. Алгоритм HPC

Алгоритм HPC был разработан известным американским криптологом Ричардом Шреппелем (Richard Schroepel) из Университета штата Аризона в 1998 г. Весьма необычно авторское название алгоритма (HPC, Hasty Pudding Cipher), первые два слова которого переводятся как «мучной заварной пuding». Я не нашел где-либо объяснения такого «кулинарного» названия алгоритма; причем даже в спецификации алгоритма встречается множество кулинарных терминов, не используемых обычно в криптографии («spice» — специя, «stirring» — взбалтывание и т. д.), себя же, например, в [354] автор называет словом «puddingmeister», т. е. «изготовитель пудинга».

## Основные характеристики алгоритма

Еще одна интересная особенность алгоритма — абсолютно произвольный размер шифруемого блока и ключа шифрования — алгоритм HPC не ограничивает эти величины ничем.

На самом деле то, что автор называет алгоритмом HPC, является совокупностью пяти различных (но взаимосвязанных) субалгоритмов, каждый из кото-

рых предназначен для шифрования блоков данных определенного размера [351, 355]:

- HPC-Tiny — для шифрования блоков размером от 0 до 35 битов включительно;
- HPC-Short — от 36 до 64 битов;
- HPC-Medium — от 65 до 128 битов;
- HPC-Long — от 129 до 512 битов;
- HPC-Extended — 513 битов и более.

Рассмотрим подробно субалгоритм HPC-Medium, поскольку именно в его диапазон действия попадает 128-битный размер блока данных, актуальный для конкурса AES (*см. разд. 2.1*), в котором алгоритм HPC участвовал.

## Структура раунда

Шифрование выполняется в 8 раундов. Предварительно шифруемые данные записываются в 64-битные регистры  $S_0$  и  $S_1$ , над содержимым которых в каждом раунде выполняется множество элементарных операций, приведенных на рис. 3.93, где использованы следующие обозначения:

- $\oplus$  — операция побитового сложения по модулю 2;
- знаками + и – обозначены операции, соответственно, сложения и вычитания 64-битных операндов по модулю  $2^{64}$ ;
- $\ll n$  — циклический сдвиг влево (или вправо) на указанное фиксированное число битов;
- $\ll$  — циклический сдвиг влево (или вправо) на переменное число битов, определяемое значением 6 младших битов «бокового» параметра;
- $t()$  — функция обнуления младшего байта 64-битного операнда;
- $C_1$  — константа, участвующая в вычислениях и определяемая следующим образом:

$$C_1 = b + PI19,$$

где  $b$  — размер шифруемого блока в битах, а  $PI19$  — «псевдослучайная» константа:

$$PI19 = 3141592653589793238,$$

происхождение которой прозрачно — это запись первых 19 знаков числа  $\pi$ , игнорируя запятую;

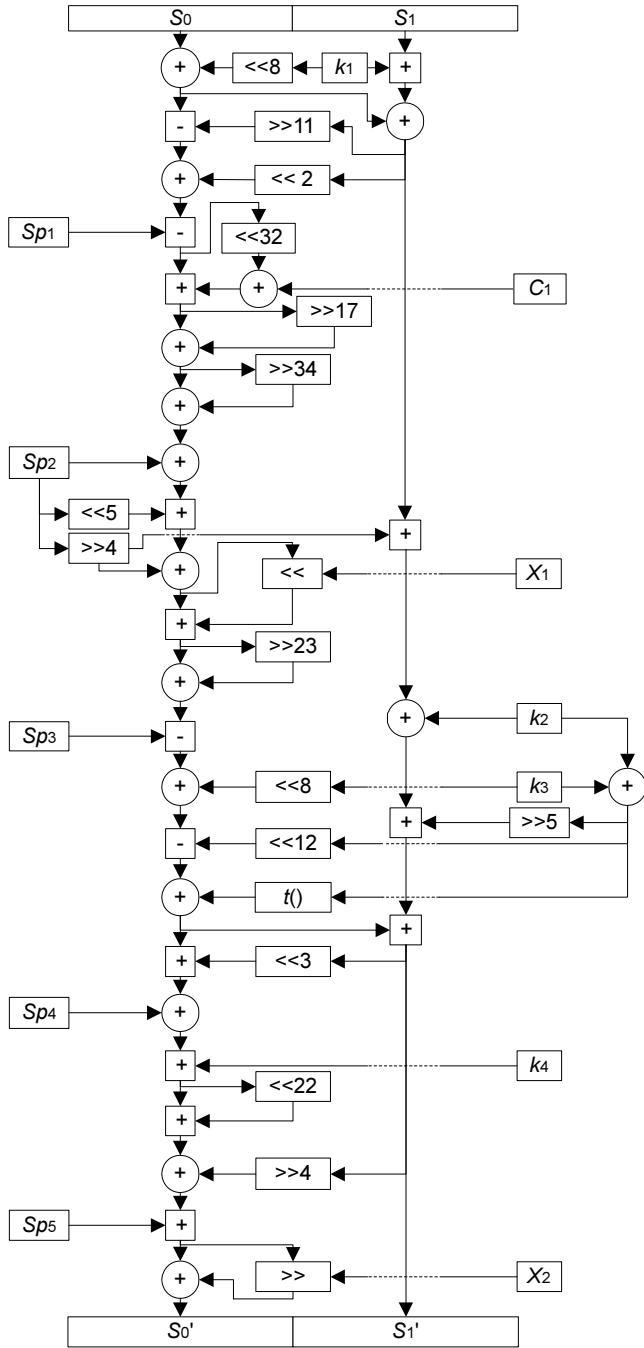


Рис. 3.93. Структура раунда алгоритма HPC

## Описание алгоритмов

- $X_n$  — значения, определяющие количество битов циклического сдвига в соответствующей операции:

$$X_1 = 22 + (< S_0 > \& 31);$$

$$X_2 = 33 + i,$$

где:

- $< S_0 >$  — текущее на момент выполнения операции значение регистра  $S_0$ ;
- $\&$  — операция побитового логического «и»;
- $i$  — номер текущего раунда (начиная с 0);

- $k_n$  — фрагмент расширенного ключа (процедура расширения ключа шифрования описана далее):

$$k_1 = K[< S_0 > \& 255];$$

$$k_2 = K[< S_0 > \& 255];^1$$

$$k_3 = K[(< S_0 > \& 255) + 3 * i + 1];$$

$$k_4 = K[b + 16 + i],$$

где  $K$  — расширенный ключ — массив, состоящий из 256 64-битных слов;

- $Sp_n$  — фрагмент дополнительного ключа, участвующего в операции шифрования (*spice*):

$$Sp_1 = Spice[i \oplus 4];$$

$$Sp_2 = Spice[i];$$

$$Sp_3 = Spice[i \oplus 7];$$

$$Sp_4 = Spice[i \oplus 2];$$

$$Sp_5 = Spice[i \oplus 1],$$

где  $Spice$  — массив дополнительного ключа, состоящий из 8 64-битных слов.

Как видно из рисунка, в каждом раунде шифрования выполняется огромное количество элементарных операций, причем большинство из них выполняется над 64-битными словами.

По завершении 8 раундов преобразований выполняются 2 дополнительные операции: значение  $S_0'$  (т. е. значение регистра  $S_0$  после выполнения приведенных на рис. 3.93 действий) складывается по модулю  $2^{64}$  с  $K[b + 8]$ , а  $S_1'$  — с  $K[b + 9]$ .

---

<sup>1</sup> Между вычислениями  $k_1$  и  $k_2$  значение  $< S_0 >$ , в общем случае, изменилось, поэтому  $k_1 \neq k_2$ .

Как написано автором в спецификации алгоритма, «его структура является сетью Фейстеля, однако измененной настолько, что не факт, что Фейстель распознал бы ее». Эксперты конкурса AES также не узнали в структуре алгоритма сеть Фейстеля [284].

Расшифровывание производится выполнением обратных операций в обратной последовательности.

## Процедура расширения ключа

Задача процедуры расширения ключа — формирование расширенного ключа, представляющего собой массив из 256 64-битных слов. Причем для каждого из субалгоритмов должен быть отдельный массив расширенного ключа (процесс расширения ключа для каждого из субалгоритмов различен). Стоит сказать, что знание одного из массивов расширенного ключа не позволяет вычислить значения других расширенных ключей, а также исходный ключ шифрования. Однако если используется фиксированный размер блока шифруемых данных, достаточно сформировать расширенный ключ только для субалгоритма, соответствующего используемому размеру блока.

Опишем процедуру расширения ключа:

1. Массив расширенного ключа инициализируется так:

$$K[0] = PI19 + Nc;$$

$$K[1] = E19 + L;$$

$$K[2] = R220 \ll Nc,$$

где:

- $Nc$  — номер субалгоритма (например, 3 для наиболее актуального субалгоритма HPC-Medium);
- $L$  — размер ключа шифрования в битах;
- $E19$  и  $R220$  — аналогичные  $PI19$  «псевдослучайные» константы:

$$E19 = 2718281828459045235;$$

$$R220 = 14142135623730950488.$$

Остальные 253 слова массива инициализируются следующим образом:

$$K[i] = K[i-1] + (K[i-2] \oplus (K[i-3] \gg 23) \oplus (k[i-3] \ll 41)) \bmod 2^{64}.$$

2. Производится побитовое сложение по модулю 2 ключа шифрования и проинициализированного массива расширенного ключа (но не более 128 слов — вспомним про переменный и неограниченный размер ключа).

3. Выполняется функция перемешивания данных расширенного ключа, которая обеспечивает влияние каждого бита ключа шифрования на значение каждого бита расширенного ключа. Далее приведена последовательность операций функции перемешивания.

- Выполняется инициализация регистров  $S_0 \dots S_7$  следующим образом:

$$S_7 = K[255], S_6 = K[254], \dots, S_0 = K[248].$$

- Для каждого слова расширенного ключа производятся вычисления, приведенные на рис. 3.94. Причем для усиления эффекта перемешивания этот этап может выполняться в несколько раундов; автор алгоритма рекомендует 3 раунда перемешивания.

На рис. 3.94 использованы следующие обозначения:

- $|$  — операция побитового логического «или»;
- $i$  — номер вычисляемого слова расширенного ключа;
- $j$  — номер раунда выполнения этапа 3;
- $kc_n$  — текущие значения слов расширенного ключа, выбираемых определенным образом:

$$kc_1 = K[i],$$

$$kc_2 = K[(i + 83) \& 255],$$

$$kc_3 = K[< S_0 > \& 255].$$

4. Если размер ключа шифрования превышает 128 64-битных слов (т. е. 8192 битов — что является более, чем достаточным размером ключа по современным меркам), то для каждого 128-словного фрагмента ключа повторяются шаги 2 и 3 — т. е. фрагмент ключа добавляется к расширенному ключу и перемешивается.

Таким образом, процедура расширения ключа является не менее сложной, чем собственно шифрование данных.

Стоит сказать несколько слов и про дополнительный ключ. Его назначение — модифицировать результат шифрования при одинаковых значениях входных данных и ключа шифрования. Таким образом, *Spice* играет роль вектора инициализации в режиме шифрования со сцеплением блоков шифра. Автор алгоритма считает возможность использования такого «внутреннего» по отношению к алгоритму вектора инициализации большим преимуществом алгоритма и рекомендует модифицировать значение *Spice* перед шифрованием каждого блока данных [351].

Дополнительный ключ может быть секретным, что увеличивает фактический объем ключевой информации, однако в алгоритме с неограниченным разме-

ром ключа такая возможность кажется излишней. Если же в дополнительном ключе нет необходимости, его можно обнулить и не выполнять ряд соответствующих операций раунда шифрования (см. рис. 3.93).

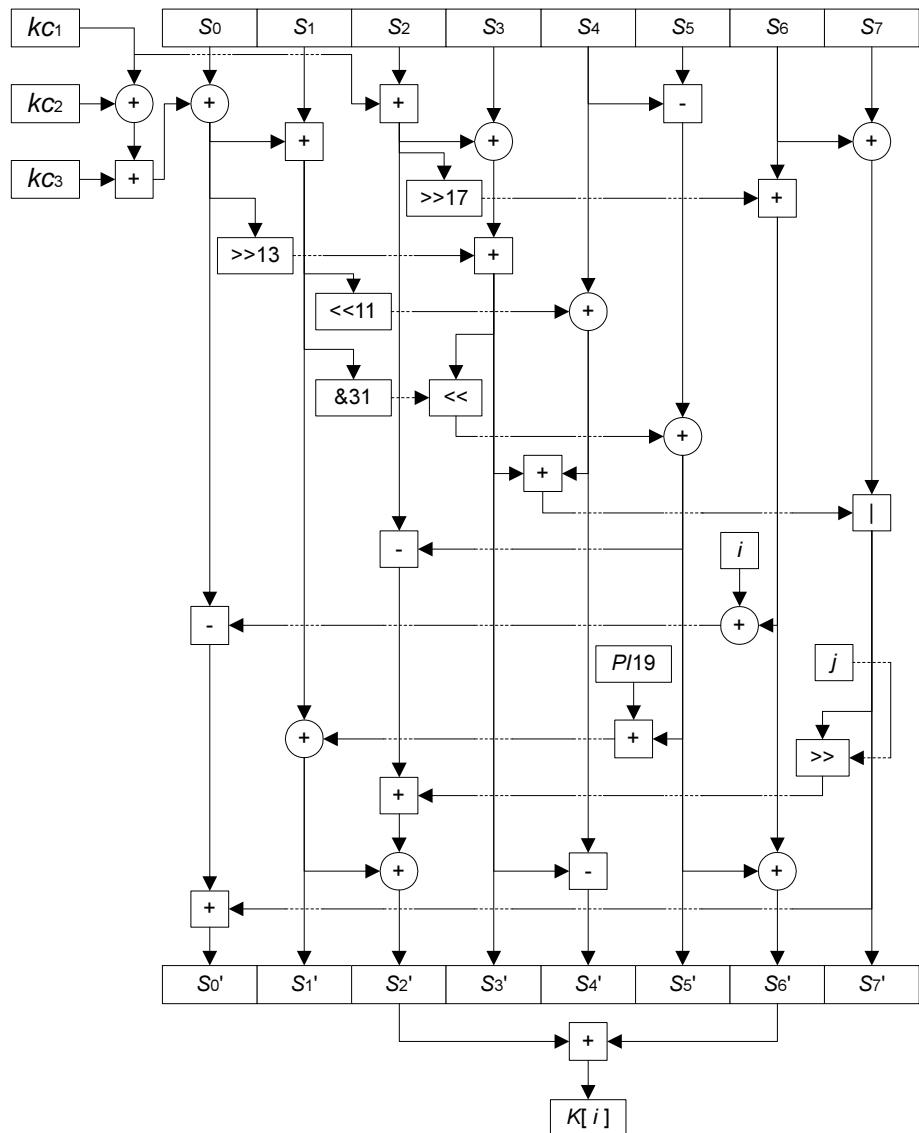


Рис. 3.94. Раунд функции  
перемешивания процедуры расширения ключа

## Достоинства и недостатки алгоритма

Раунд алгоритма HPC состоит из огромного количества операций. В сравнении, например, с раундом отечественного стандарта симметричного шифрования ГОСТ 28147-89 [4] (состоящим всего из четырех элементарных операций — см. разд. 3.1), алгоритм HPC выглядит чрезвычайно сложным. Тем не менее, поскольку подавляющее большинство действий выполняется с 64-битными операндами, на 64-битных платформах алгоритм HPC показал впечатляющую скорость шифрования [284, 352]: 128-битные блоки данных шифровались алгоритмом HPC быстрее всех алгоритмов — участников конкурса, за исключением алгоритма DFC (также ориентированного на 64-битные платформы), а 512- и 1024-битные блоки алгоритм HPC шифрует в 2 раза быстрее, чем ближайший конкурент по скорости — алгоритм DFC, и в 3 и более раза быстрее остальных алгоритмов, участвовавших в конкурсе. Однако стоит отметить, что на конкурсе AES «засчитывался» только 128-битный размер блока. Кроме того, на остальных платформах алгоритм HPC показывал заметно худшие результаты.

При проведении исследований в рамках конкурса AES были обнаружены и другие недостатки алгоритма HPC [284]:

- каждый 256-й ключ алгоритма имеет 230 эквивалентных ключей; этот недостаток был оперативно исправлен автором алгоритма еще до подведения итогов первого раунда конкурса AES (здесь рассмотрена уже исправленная версия) [351, 354];
- очень медленная процедура расширения ключа шифрования, которая не может выполняться параллельно с собственно шифрованием данных;
- алгоритм предъявляет очень большие требования к энергонезависимой и оперативной памяти, что весьма затрудняет его применение в смарт-картах;
- сложная структура алгоритма затрудняет его исследование с целью доказательства отсутствия уязвимостей.

Алгоритм HPC не попал во второй этап конкурса. В своей статье [354] автор алгоритма обрушился с критикой на экспертов конкурса, которая сводилась к следующему утверждению: на конкурсе AES неправильно расставили приоритеты — алгоритм, выбираемый, фактически, в качестве мирового стандарта, должен быть оптимизирован именно под 64-битные платформы, за которыми уже ближайшее будущее. Кроме того, по словам разработчика алгоритма HPC, нельзя разработать алгоритм, одинаково хороший для 64-битных многопроцессорных серверов и дешевых 8-битных смарт-карт. На результаты конкурса AES эта позиция не повлияла, однако лично мне подобная критика кажется весьма обоснованной.

## 3.24. Алгоритм ICE

### История создания алгоритма

Алгоритм ICE (Information Concealment Engine, механизм скрытия информации) — один из многих алгоритмов шифрования, позиционируемых своими авторами как возможная замена стандарта шифрования DES. Автор алгоритма — известный австралийский криптолог Мэттью Кван (Matthew Kwan). Алгоритм разработан в 1997 г.

Для облегчения возможной замены DES на ICE во многих приложениях автор алгоритма придал ему схожие с DES параметры:

- размер блока шифруемых данных — 64 бита;
- размер ключа шифрования — 64 бита; в отличие от 64-битного ключа DES, в котором значащими являются всего 56 битов (см. разд. 3.15), все биты ключа алгоритма ICE являются значащими.

Как известно, алгоритм ICE не стал заменой алгоритму DES и не получил широкого распространения. Судя по информации, изложенной в спецификации алгоритма [232], вопреки мнению большинства экспертов, Мэттью Кван считал основной проблемой алгоритма DES не короткий ключ, а восприимчивость к линейному и дифференциальному криптоанализу и наличие слабых ключей. 64-битный ключ алгоритма ICE всего в 256 раз длиннее ключа DES, т. е. также подвержен проблеме полного перебора ключей. Кроме того, 64-битный размер блока также стал считаться недостаточным — в том же 1997 г. начался конкурс AES по выбору нового стандарта шифрования США, который устанавливал принципиально иные характеристики нового стандарта:

- 128-битный размер блока шифруемых данных;
- три фиксированных длины ключа шифрования: 128, 192 и 256 битов.

Так что можно утверждать, что алгоритм ICE появился уже несколько устаревшим. Кроме того, в алгоритме были обнаружены уязвимости (см. далее), которые также не способствовали популярности алгоритма.

### Структура алгоритма

Алгоритм ICE представляет собой сеть Фейстеля (рис. 3.95).

В алгоритме 16 раундов, в каждом из которых над 32-битным субблоком выполняются следующие операции (рис. 3.96) [232]:

1. Функция  $E$ , выполняющая расширение субблока до четырех 10-битных величин  $E_1 \dots E_4$  таким образом:

$$E_1 = P_1, P_0, P_{31}, P_{30}, P_{29}, P_{28}, P_{27}, P_{26}, P_{25}, P_{24};$$

## Описание алгоритмов

$$E_2 = P_{25}, P_{24}, P_{23}, P_{22}, P_{21}, P_{20}, P_{19}, P_{18}, P_{17}, P_{16};$$

$$E_3 = P_{17}, P_{16}, P_{15}, P_{14}, P_{13}, P_{12}, P_{11}, P_{10}, P_9, P_8;$$

$$E_4 = P_9, P_8, P_7, P_6, P_5, P_4, P_3, P_2, P_1, P_0,$$

где  $P_n$  —  $n$ -й бит обрабатываемого субблока (считая справа налево).

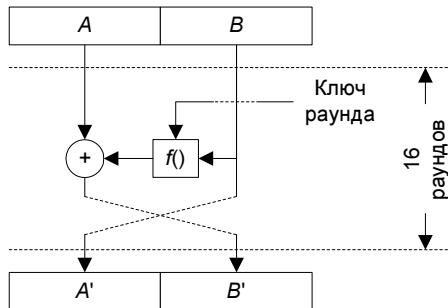


Рис. 3.95. Структура алгоритма ICE

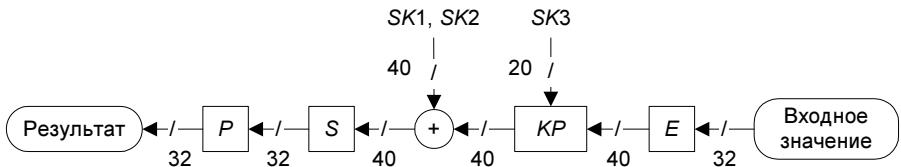


Рис. 3.96. Функция  $f$

2. Зависящая от значения первых 20 битов ключа раунда ( $K_i$ , где  $i$  — номер текущего раунда) перестановка  $KP$ , которая выполняется по следующим правилам:
  - если  $n$ -й бит ( $0 \leq n \leq 9$ ) фрагмента ключа раунда имеет значение 1, то меняются местами  $n$ -й бит  $E_2$  и  $n$ -й бит  $E_4$ ;
  - если  $(10+n)$ -й бит фрагмента ключа раунда имеет значение 1, то меняются местами  $n$ -й бит  $E_1$  и  $n$ -й бит  $E_3$ .
3. Наложение материала ключа — выполнение побитовой логической операции «исключающее или» (XOR) над каждым битом следующих 40 битов ключа раунда (ключ раунда имеет размер 60 битов) и соответствующим битом последовательности  $E_1 \dots E_4$ .
4. Табличная замена  $S$ : после наложения ключа  $E_1 \dots E_4$  «прогоняются» через таблицы замен  $S_1 \dots S_4$  соответственно. Каждая из таблиц  $S_1 \dots S_4$  заменяет

входное 10-битное значение 8-битным. Таблицы замен работают следующим образом:

- 9-й и 0-й биты входного значения формируют значение переменной  $V$ , значение остальных битов входного значения обозначается как  $Y$ ;
- выходное значение  $Z$  вычисляется следующим образом:

$$Z = (Y + O_V)^7 \bmod P_v,$$

где:

- ◊  $O_v$  — константа, определяемая согласно табл. 3.50.
- ◊  $P_v$  — неприводимый в 8-битном поле Галуа многочлен, коэффициенты которого определяются двоичной последовательностью  $m_v$ , представленной в десятичном виде в табл. 3.51.

**Таблица 3.50**

	$O_0$	$O_1$	$O_2$	$O_3$
$S_1$	83	85	9B	CD
$S_2$	CC	A7	AD	41
$S_3$	4B	2E	D4	33
$S_4$	EA	CD	2E	04

**Таблица 3.51**

	$m_0$	$m_1$	$m_2$	$m_3$
$S_1$	333	313	505	369
$S_2$	379	375	319	391
$S_3$	361	445	451	397
$S_4$	397	425	395	505

5. Функция  $P$ , выполняющая битовую перестановку данных согласно табл. 3.52.

**Таблица 3.52**

$S1_7$	$S4_7$	$S3_7$	$S2_7$	$S2_6$	$S3_6$	$S1_6$	$S4_6$
$S3_5$	$S2_5$	$S4_5$	$S1_5$	$S4_4$	$S1_4$	$S2_4$	$S3_4$
$S2_3$	$S3_3$	$S4_3$	$S1_3$	$S1_2$	$S4_2$	$S2_2$	$S3_2$
$S4_1$	$S1_1$	$S3_1$	$S2_1$	$S3_0$	$S2_0$	$S1_0$	$S4_0$

В этом случае  $Sa_b$  —  $b$ -й выходной бит таблицы замен  $Sa$ ; таблица обозначает, что бит  $S1_7$  становится битом 31, бит  $S4_7$  становится битом 30 и т. д. (рис. 3.97).

После этого обрабатываемый субблок накладывается с помощью операции XOR на необработанный.

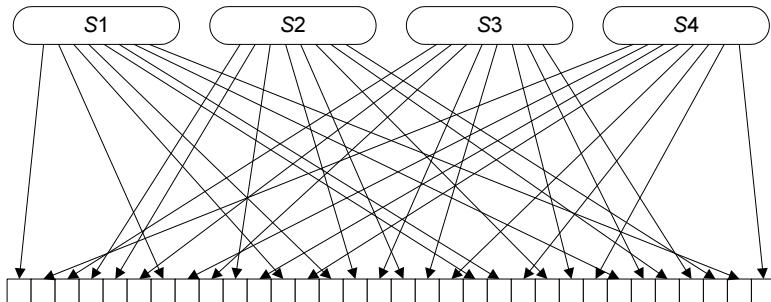


Рис. 3.97. Перестановка  $P$

Как видно, функция раунда алгоритма ICE весьма похожа на функцию раунда алгоритма DES; принципиальное отличие состоит лишь в том, что в DES отсутствует зависящая от ключа перестановка.

## Варианты алгоритма

Выше описан основной вариант алгоритма ICE. Кроме него, в спецификации алгоритма описаны следующие варианты.

- Thin-ICE — версия с уменьшенным количеством раундов (и более низкой криптостойкостью), предназначенная для высокоскоростных применений, в которых не предъявляются требования к повышенной защищенности данных. В этом варианте алгоритма выполняется 8 раундов вместо 16.
- ICE-n — версия с увеличенным (потенциально неограниченным) количеством раундов, в которой выполняется  $n * 16$  раундов шифрования. В этом варианте отличается и размер ключа алгоритма — вместо 64-битного ключа используется  $(64 * n)$ -битный ключ.

## Процедура расширения ключа

Задача процедуры расширения ключа состоит в формировании 16 ключей раундов по 60 битов. Данная процедура является достаточно сложной и выполняется в несколько шагов:

1. 64-битный ключ шифрования используется для инициализации массива временных переменных  $KB$ :

$$KB[0] = K_{63} \dots K_{48};$$

$$KB[1] = K_{47} \dots K_{32};$$

$$KB[2] = K_{31} \dots K_{16};$$

$$KB[3] = K_{15} \dots K_0,$$

где  $K_n$  —  $n$ -й бит расширяемого ключа шифрования.

2. Обнуляются регистры  $SK1 \dots SK3$ , предназначенные для хранения текущего ключа раунда. Предполагается, что назначение подключей таково:
  - $SK1$  и  $SK2$  — предназначены для наложения ключа операцией XOR;
  - $SK3$  — управляет перестановкой  $KP$  (см. рис. 3.96).
3. Для каждого раунда процедуры расширения ключа  $j$  (соответствуют раундам алгоритма) поочередно для  $SK1$ ,  $SK2$  и  $SK3$  (текущий регистр далее обозначен как  $SK$ ) пятикратно выполняется цикл по  $i$  от 0 до 3 (т. е. всего 4 уровня вложенности циклов), в котором производятся следующие действия:

$$B = KB[(i + KR[j]) \bmod 4]_0;$$

$$SK \ll 1;$$

$$SK_0 = B;$$

$$KB[(i + KR[j]) \bmod 4] \gg 1;$$

$$KB[(i + KR[j]) \bmod 4]_{19} = \sim B,$$

где:

- $B$  — временная 1-битная переменная;
- $\ll$  и  $\gg$  — операции сдвига на указанное число битов влево и вправо соответственно;
- $\sim B$  — обратное значение для  $B$ ;
- $KR[j]$  —  $j$ -й элемент массива  $KR$ , приведенного в табл. 3.53.

**Таблица 3.53**

Раунд $j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$KR[j]$	0	1	2	3	2	1	3	0	1	3	2	0	3	1	0	2

Выше описана процедура расширения ключа для основного варианта алгоритма ICE, в котором выполняется 16 раундов шифрования. Для других вариантов алгоритма расширение ключа производится следующим образом:

- для 8-раундового Thin-ICE просто выполняются первые 8 раундов описанного выше цикла шага 3 процедуры расширения ключа;

- для ICE- $n$  выполняется описанная выше процедура расширения ключа, затем массив  $KB$  инициализируется следующим 64-битным фрагментом ключа, на основе которого вычисляются еще 16 ключей раундов и т. д. в цикле всего  $n$  раз.

## Криптоанализ алгоритма

В 1998 г. несколько экспертов из Католического Университета г. Лювен, Бельгия, предложили метод вскрытия алгоритма ICE с помощью дифференциального криптоанализа [377]. Данный метод позволяет добиться следующих результатов.

- Алгоритм Thin-ICE вскрывается (т. е. вычисляется используемый ключ шифрования) при наличии  $2^{23}$  выбранных открытых текстов и соответствующих им шифртекстов с вероятностью 25 %. При увеличении количества выбранных открытых текстов до  $2^{27}$  вероятность успеха повышается до 95 %.
- Существует вероятность вскрытия и стандартного алгоритма ICE (правда, достаточно небольшая) при наличии не менее, чем  $2^{62}$  выбранных открытых текстов. Данная атака, однако, на практике реализуется с большим трудом.

## 3.25. Алгоритм ICEBERG

Алгоритм шифрования ICEBERG предложен относительно недавно — в 2004 г. — французским криптологом Жилем-Франсуа Пирэ (Gilles-Francois Piret). ICEBERG — это аббревиатура от «Involutional Cipher Efficient for Block Encryption in Reconfigurable Hardware», «инволюционный шифр для эффективного блочного шифрования на основе перестраиваемой аппаратуры». Как видно из названия, данный алгоритм оптимизирован под аппаратные реализации с помощью программируемых логических интегральных схем.

## Структура алгоритма

Алгоритм ICEBERG шифрует данные 64-битными блоками с использованием 128-битного ключа шифрования. Обрабатываемый блок данных представляется в виде 64-битного вектора, над которым в каждом раунде алгоритма последовательно выполняются следующие операции (рис. 3.98) [301]:

1. Табличная замена  $S_0$ . Вектор данных представляется в виде 16 значений по 4 бита, каждое из которых замещается согласно табл. 3.54.

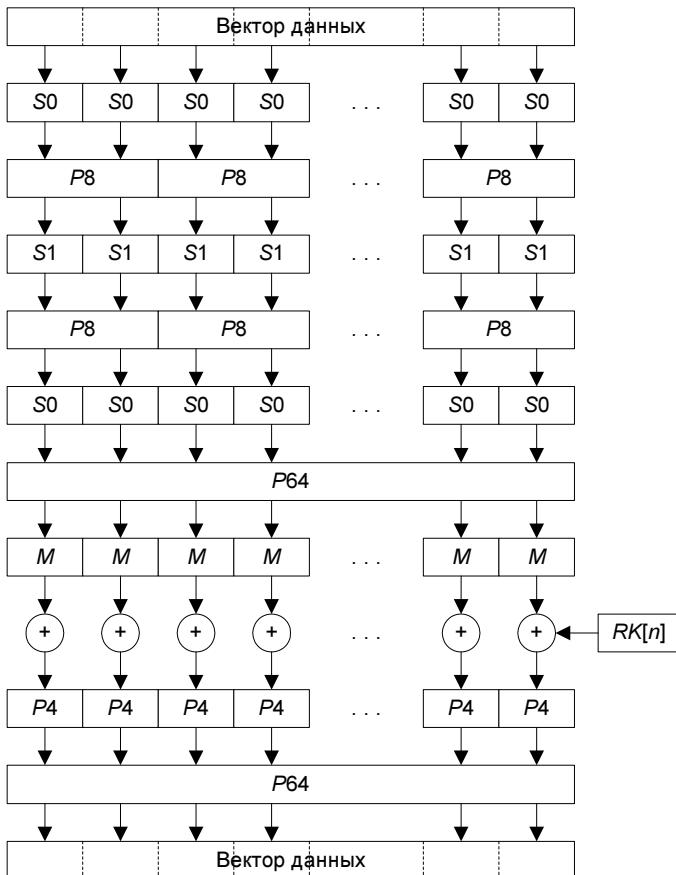


Рис. 3.98. Раунд алгоритма ICEBERG

Таблица 3.54

Входное значение	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Выходное значение	D	7	3	2	9	A	C	1	F	4	5	E	6	0	B	8

2. Битовая перестановка  $P8$ , переставляющая биты вектора данных по следующему правилу:

$$y_{8i+j} = x_{8i+p8(j)}, \quad i = 0 \dots 7, \quad j = 0 \dots 7,$$

где:

- $x_n$  и  $y_n$  — соответственно, входной и выходной биты вектора данных;
- функция  $p8()$  определена согласно табл. 3.55.

Таблица 3.55

<b>Входное значение</b>	0	1	2	3	4	5	6	7
<b>Выходное значение</b>	0	1	4	5	2	3	6	7

3. Табличная замена  $S_1$ , работающая аналогично замене  $S_0$ , но согласно другой таблице (табл. 3.56).

Таблица 3.56

<b>Входное значение</b>	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
<b>Выходное значение</b>	4	A	F	C	0	D	9	B	E	6	1	7	3	5	8	2

4. Снова выполняется перестановка  $P8$ , после которой повторно применяется табличная замена  $S_0$ .
5. Битовая перестановка  $P64$ , переставляющая биты вектора данных следующим образом:

$$y_i = x_{p64(i)}, \quad i = 0 \dots 63,$$

где функция  $p64()$  определена согласно табл. 3.57.

Таблица 3.57

0	12	23	25	38	42	53	59	22	9	26	32	1	47	51	61
24	37	18	41	55	58	8	2	16	3	10	27	33	46	48	62
11	28	60	49	36	17	4	43	50	19	5	39	56	45	29	13
30	35	40	14	57	6	54	20	44	52	21	7	34	15	31	63

Согласно таблице входным значениям 0, 1, 2, ... соответствуют выходные значения 0, 12, 23 и т. д.

6. Операция умножения на матрицу  $M$ . Применяется к каждому 4-битному фрагменту вектора данных путем его умножения на фиксированную матрицу (табл. 3.58).

Это аналогично применению следующей табличной замены (табл. 3.59).

Таблица 3.58

0	1	1	1
1	0	1	1
1	1	0	1
1	1	1	0

Таблица 3.59

Входное значение	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Выходное значение	0	E	D	3	B	5	6	8	7	9	A	4	C	2	1	F

7. Наложение материала ключа (операция  $\sigma$ ):

$$y_i = x_i \oplus RK[n]_i,$$

где  $RK[n]_i$  —  $i$ -й бит фрагмента расширенного ключа для  $n$ -го раунда (процедура расширения ключа будет описана далее).

8. Битовая перестановка  $P4$ , применяемая к каждому 4-битному фрагменту вектора данных:

$$y[i]_j = x[i]_{p4(j)}, \quad i = 0 \dots 15, \quad j = 0 \dots 3,$$

где  $y[i]$  и  $x[i]$  — соответственно, выходное и входное значения  $i$ -го 4-битного фрагмента, а функция  $p4()$  определяется согласно табл. 3.60.

Таблица 3.60

Входное значение	0	1	2	3
Выходное значение	1	0	3	2

9. Повторно применяется описанная выше операция  $P64$ .

Алгоритм состоит из 15 раундов преобразований. Перед первым раундом выполняется предварительное наложение ключа, а после заключительного раунда — финальное преобразование, состоящее из следующих операций (описанных выше):

- табличной замены  $S_0$ ;
- битовой перестановки  $P8$ ;
- табличной замены  $S_1$ ;

- битовой перестановки  $P8$ ;
- табличной замены  $S_0$ .

При расшифровывании выполняются абсолютно те же операции, но фрагменты расширенного ключа используются в обратном порядке (при этом они формируются несколько иначе, чем при зашифровывании — см. далее).

## Процедура расширения ключа

Процедура расширения ключа состоит из нескольких этапов. На первом этапе из исходного 128-битного ключа шифрования  $K$  следующим образом вычисляется последовательность 128-битных промежуточных ключей  $IK_0 \dots IK_{16}$ :

$$IK_0 = K;$$

$$IK_{i+1} = G(IK_i, C_i),$$

где  $G()$  — раунд процедуры расширения ключа, а  $C_i$  — модифицирующие константы.

В каждом раунде выполняется следующая последовательность действий (рис. 3.99):

1. Операция  $\tau$  производит циклический сдвиг входных данных в зависимости от значения  $C_i$ :

- при  $C_i = 0$  выполняется вращение на 8 битов влево;
- при  $C_i = 1$  — циклический сдвиг на 8 битов вправо.

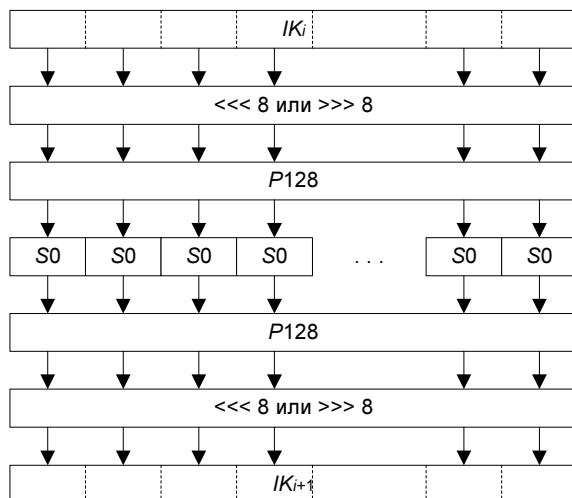


Рис. 3.99. Раунд процедуры расширения ключа

Константы  $C_i$  определены как 0 для первой половины раундов, т. е. для  $i = 0 \dots 7$ , для остальных раундов  $C_i = 1$ .

2. Битовая перестановка  $P128$  производит перестановку по следующему закону:

$$y_i = x_{p128(i)}, i = 0 \dots 127,$$

где функция  $p128()$  определена согласно табл. 3.61.

**Таблица 3.61**

76	110	83	127	67	114	92	97	98	65	121	106	78	112	91	82
71	101	89	126	72	107	81	118	90	124	73	88	64	104	100	85
109	87	75	113	120	66	103	115	122	108	95	69	74	116	80	102
84	96	125	68	93	105	119	79	123	86	70	117	111	77	99	94
28	9	37	4	51	43	58	16	20	26	44	34	0	61	12	55
46	22	15	2	48	31	57	33	27	18	24	14	6	52	63	42
49	7	8	62	30	17	47	38	29	53	11	21	41	32	1	60
13	35	5	39	45	59	23	54	36	10	40	56	25	50	19	3

Входным значениям 0, 1, 2, ... соответствуют выходные значения 76, 110, 83 и т. д.

3. К 4-битным фрагментам обрабатываемых данных применяется описанная выше табличная замена  $S_0$ .
4. Повторно применяется перестановка  $P128$ , после которой повторно выполняется операция  $\tau$ .

На следующем этапе формируются 64-битные подключи  $IK'_0 \dots IK'_{16}$ , которые просто «набираются» из значений нечетных битов соответствующих промежуточных ключей  $IK_0 \dots IK_{16}$ .

На заключительном этапе процедуры расширения ключа вычисляются раундовые ключи  $RK[n]$ . Для этого каждый 4-битный фрагмент каждого подключа  $IK'_0 \dots IK'_{16}$  параллельно обрабатывается операцией  $\phi$ , которая определена следующим образом:

$$y_0 = ((x_0 \oplus x_1 \oplus x_2) \& b) | ((x_0 \oplus x_1) \& (\sim b));$$

$$y_1 = ((x_1 \oplus x_2) \& b) | (x_1 \& (\sim b));$$

$$y_2 = ((x_2 \oplus x_3 \oplus x_0) \& b) | ((x_2 \oplus x_3) \& (\sim b));$$

$$y_3 = ((x_3 \oplus x_0) \& b) | (x_3 \& (\sim b)),$$

где:

- $y_i$  и  $x_i$  —  $i$ -е биты, соответственно, выходного и входного значения;
- $b$  — модифицирующий бит, его назначение будет пояснено далее;
- $\sim$  — логическая операция отрицания;
- $\&$  — логическая операция «и»;
- $|$  — логическая операция «или».

Модифицирующий бит  $b$  управляет формированием различных значений подключей для зашифровывания и расшифровывания:

- при зашифровывании для операции предварительного наложения ключа и 15 раундов преобразований соответствующие подключи ( $RK[0]$  и  $RK[1] \dots RK[15]$ ) формируются со значением  $b = 1$ ; подключ  $RK[16]$ , используемый в финальном преобразовании, формируется со значением  $b = 0$ ;
- при расшифровывании для предварительного наложения ключа и 15 раундов преобразований используются, соответственно, фрагменты  $RK[16]$  и  $RK[15] \dots RK[1]$ , которые формируются с модифицирующим битом  $b = 0$ ; подключ  $RK[0]$ , используемый в финальном преобразовании, формируется со значением  $b = 1$ .

## Криптоанализ алгоритма

Алгоритм ICEBERG появился относительно недавно и, видимо, не вызвал широкого интереса со стороны криптологов — какие-либо работы, посвященные криптоанализу данного алгоритма, не получили широкой известности.

## 3.26. Алгоритмы IDEA, PES, IPES

Первоначальный вариант алгоритма IDEA появился в 1990 г. [234]. Разработчики алгоритма: Ксуеджа Лай (Xuejia Lai) и Джеймс Мэсси (James L. Massey) из швейцарского института ETH Zurich, — дали алгоритму название PES (Proposed Encryption Standard, предлагаемый стандарт шифрования), поскольку данный алгоритм был предложен на замену стандарта DES. Стоит сказать, что институт ETH Zurich и профессор Джеймс Мэсси известны также благодаря разработанным ими алгоритмам семейства SAFER (см. разд. 3.44–3.46).

Через год алгоритм был модифицирован с целью усиления криптостойкости против дифференциального криптоанализа, новая версия получила название IPES (Improved PES, улучшенный PES) [235], впоследствии алгоритм сменил название на IDEA (International Data Encryption Algorithm, международный алгоритм шифрования данных).

## Основные характеристики и структура

Алгоритм шифрует данные блоками по 64 бита, а ключ шифрования алгоритма имеет размер 128 битов.

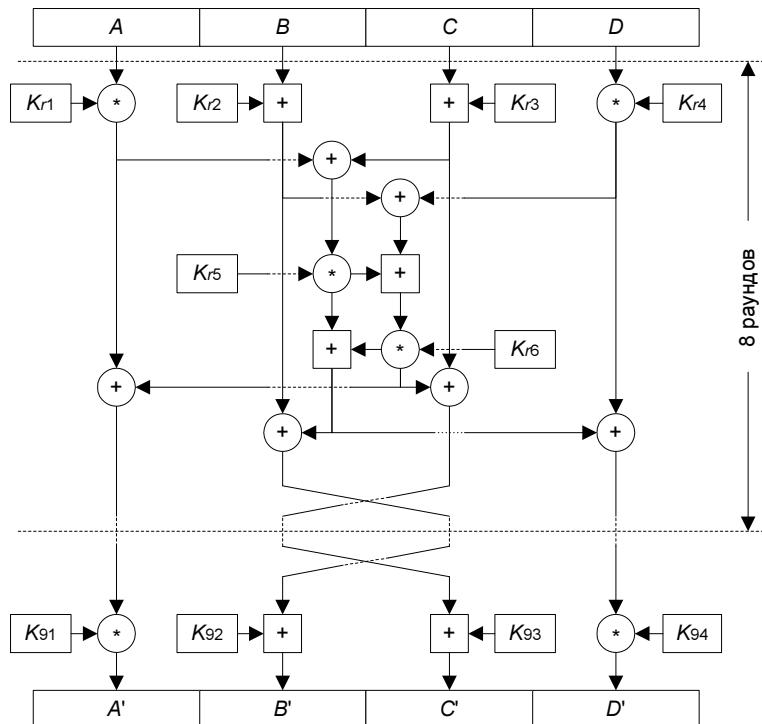


Рис. 3.100. Структура алгоритма IDEA

Блок шифруемых данных разбивается на 4 16-битных субблока  $A$ ,  $B$ ,  $C$  и  $D$  (рис. 3.100), над которыми выполняется 8 раундов преобразований [21, 28, 235]:

$$A = A \otimes K_{r1};$$

$$B = B + K_{r2};$$

$$C = C + K_{r3};$$

$$D = D \otimes K_{r4};$$

$$T1 = A \oplus C;$$

$$T2 = B \oplus D;$$

$$T1 = T1 \otimes K_{r5};$$

$$\begin{aligned}
 T2 &= T1 + T2; \\
 T2 &= T2 \otimes K_{r6}; \\
 T1 &= T1 + T2; \\
 A &= A \oplus T2; \\
 B &= B \oplus T1; \\
 C &= C \oplus T2; \\
 D &= D \oplus T1,
 \end{aligned}$$

где:

- $K_r$  — подключ  $n$  раунда  $r$ ;
- $+$  — операция сложения 16-битных операндов по модулю  $2^{16}$ ;
- $\otimes$  — умножение 16-битных операндов по модулю  $(2^{16} + 1)$ , причем в качестве значения субблока, состоящего из одних нулей, берется значение  $2^{16}$ .

После выполнения описанных выше действий два внутренних субблока ( $B$  и  $C$ ) меняются местами — во всех раундах, кроме последнего. По завершении 8 раундов выполняются дополнительные преобразования (иногда называемые девятым раундом алгоритма — например, в [263]):

$$\begin{aligned}
 A' &= A \otimes K_{91}; \\
 B' &= B + K_{92}; \\
 C' &= C + K_{93}; \\
 D' &= D \otimes K_{94}.
 \end{aligned}$$

Шифртекст представляет собой результат конкатенации полученных значений  $A'$ ,  $B'$ ,  $C'$  и  $D'$ .

Операция расшифровывания аналогична зашифровыванию, однако при расшифровывании используются модифицированные подключи и в другой последовательности:

$$\begin{aligned}
 K'_{r1} &= (K_{(10-r)1})^{-1}; \\
 K'_{r2} &= -K_{(10-r)3}; \\
 K'_{r3} &= -K_{(10-r)2}; \\
 K'_{r4} &= (K_{(10-r)4})^{-1}; \\
 K'_{r5} &= K_{(9-r)5}; \\
 K'_{r6} &= K_{(9-r)6},
 \end{aligned}$$

за исключением раундов 1 и 9, в которых подключи  $K'_{r2}$  и  $K'_{r3}$  меняются местами.

В формулах использованы следующие обозначения:

- $K'_{rn}$  — подключ  $n$  раунда расшифровывания  $r$ ;
- $-x$  и  $x^{-1}$  — обратные значения  $x$  относительно описанных выше операций сложения по модулю  $2^{16}$  и умножения по модулю  $(2^{16} + 1)$  соответственно. При этом  $0^{-1} = 0$ .

## Процедура расширения ключа

Задача процедуры расширения ключа — формирование 52 16-битных подключей, используемых в раундах шифрования и дополнительных преобразованиях, т. е. всего 832 битов ключевой информации. Данная процедура весьма проста; выполняется она следующим образом:

1. 128-битный ключ шифрования делится на 8 подключей по 16 битов; они становятся первыми восемью подключами алгоритма (т. е.  $K_{11}, K_{12}, K_{13}, K_{14}, K_{15}, K_{16}, K_{21}, K_{22}$ ).
2. Ключ шифрования циклически сдвигается влево на 25 битов.
3. Результат делится на 8 следующих подключей.
4. Ключ шифрования циклически сдвигается влево на 25 битов и т. д. до выработки необходимого количества подключей.

## Криптостойкость алгоритма

Уже в следующем году после появления алгоритма PES его авторы опубликовали работу [235], в которой была доказана слабость алгоритма PES против дифференциального криптоанализа: для определения ключа шифрования достаточно выполнения  $2^{64}$  операций шифрования, тогда как полный перебор значений 128-битного ключа потребовал бы в среднем выполнения  $2^{127}$  операций.

Алгоритм IDEA появился в результате достаточно незначительных модификаций алгоритма PES (рис. 3.101). Если сравнить схемы алгоритмов IDEA и PES, видно лишь несколько отличий:

- операция умножения субблока  $B$  со вторым подключом раунда заменена операцией сложения;
- операция сложения субблока  $D$  с четвертым подключом раунда заменена операцией умножения;
- по-другому выполняется сдвиг субблоков в конце раунда.

Один из наиболее известных в мире специалистов-криптологов Брюс Шнайер в своей книге «Прикладная криптография» [28] сказал, что «удивительно, как такие незначительные изменения могут привести к столь большим различиям».

## Описание алгоритмов

Алгоритм IDEA оказался, фактически, неуязвим к дифференциальному криптоанализу, а в той же книге (вышедшей в 1996 г.) Брюс Шнайер написал следующее: «Мне кажется, это самый лучший и надежный блочный алгоритм, опубликованный до настоящего времени».

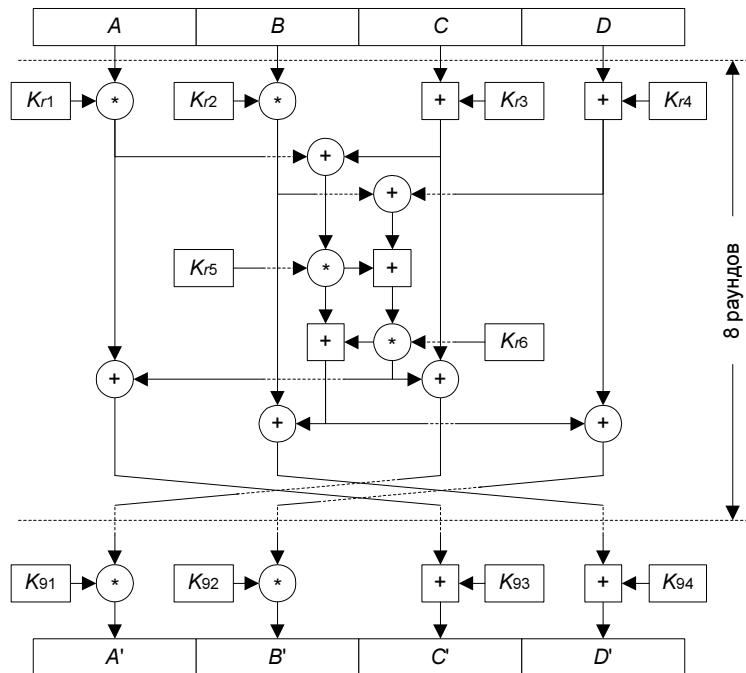


Рис. 3.101. Структура алгоритма PES

Однако в том же 1996 г. известный криптоаналитик Пол Кохер изобрел достаточно сложно реализуемую атаку [228], позволяющую путем многократных высокоточных замеров времени выполнения шифрования  $2^{20}$  случайно выбранных открытых текстов на ключах, связанных определенным соотношением с искомым ключом, и последующего анализа результатов вычислить искомый ключ шифрования (подробную информацию см. в разд. 1.7).

А ранее, в 1993 г., несколько криптологов из Бельгии обнаружили в алгоритме IDEA несколько классов слабых ключей (всего  $2^{23} + 2^{35} + 2^{51}$  ключей различной степени слабости) [129], часть из которых, например, можно вычислить криптоаналитической атакой с выбранным открытым текстом. Однако сами авторы данной атаки предложили «противоядие» — слабые ключи исключаются путем наложения операцией XOR специальной шестнадцатиричной константы 0DAE на каждый подключ перед его использованием.

Таким образом, несмотря на обнаруженные недостатки, алгоритм IDEA считается алгоритмом с высокой криптостойкостью. Неоспоримым же достоинством данного алгоритма является высокая скорость зашифровывания — не менее, чем в 2 раза быстрее алгоритма DES в зависимости от платформы, на которой выполняется шифрование [28]. А возможность выполнения операции расширения ключа «на лету» (т. е. параллельно с выполнением раундов шифрования) считается весьма желаемой и в более современных алгоритмах шифрования. Однако стоит сказать, что скорость расшифровывания несколько ниже из-за наличия ресурсоемких операций вычисления мультиплексивных обратных величин по модулю ( $2^{16} + 1$ ).

Алгоритм IDEA не стал международным стандартом шифрования, как того желали его авторы. Однако этот алгоритм можно считать одним из наиболее распространенных в мире алгоритмов шифрования. IDEA используется до сих пор во множестве различных приложений, в том числе в широко известной и используемой программе защиты данных PGP.

### **3.27 Алгоритм KASUMI**

Этот разработанный относительно недавно алгоритм уже имеет достаточно интересную историю и, видимо, будет широко применяться в будущем.

В основе алгоритма KASUMI лежит алгоритм MISTY1 (см. разд. 3.36) — один из алгоритмов семейства MISTY, разработанного в 1996–1997 гг. японской корпорацией Mitsubishi Electric [258]. Впоследствии алгоритм MISTY1 стал одним из победителей европейского конкурса криптоалгоритмов NESSIE. Стоит отметить, что еще одним из победителей конкурса NESSIE стал алгоритм Camellia (см. разд. 3.9), также разработанный компанией Mitsubishi Electric в сотрудничестве с еще одной японской корпорацией NTT (Nippon Telegraph and Telephone).

В декабре 1998 г. был образован консорциум 3GPP (3<sup>rd</sup> Generation Partnership Project), целью которого являлась разработка стандартов мобильной связи третьего поколения [29]. Одно из направлений стандартизации — разработка протоколов и алгоритмов аутентификации и защиты данных в сетях сотовой связи.

Рабочая группа консорциума 3GPP — SAGE (Security Algorithms Group of Experts, группа экспертов по алгоритмам безопасности) — проанализировала ряд известных на тот момент алгоритмов шифрования, и в качестве кандидата будущего стандарта шифрования данных в сетях сотовой связи был выбран алгоритм блочного симметричного шифрования MISTY1, шифрующий 64-битные блоки данных 128-битным ключом. Поскольку к тому времени

## Описание алгоритмов

были известны некоторые теоретические атаки на MISTY1, алгоритм был доработан с целью его адаптации для аппаратной реализации (в том числе в условиях ограниченных вычислительных ресурсов), а также усиления против некоторых видов криптоаналитических атак [159, 260, 286], в том числе:

- атак «встреча посередине»;
- дифференциального криптоанализа и ряда его вариантов;
- атак, использующих наличие слабых ключей или классов ключей;
- линейного криптоанализа.

В результате доработки MISTY1 был получен алгоритм KASUMI, незначительно отличающийся по своей структуре от MISTY1. Подробная спецификация KASUMI [365] появилась в конце декабря 1999 г. Слово «kasumi» — это перевод на японский английского слова «misty» («туманный») [260].

Следует учесть, что для использования алгоритма KASUMI необходимо получение лицензии у корпорации Mitsubishi Electric [364].

## Структура алгоритма

Алгоритм KASUMI имеет весьма необычную структуру — он основан на «вложенных» сетях Фейстеля [365]. Сначала 64-битный шифруемый блок данных разбивается на два 32-битных субблока, после чего выполняется 8 раундов следующих преобразований (рис. 3.102):

1. Над обрабатываемым субблоком производятся зависящие от ключа операции *FL* и *FO* (в указанной последовательности — в нечетных раундах, в четных — наоборот).
2. Результат этих операций накладывается побитовой логической операцией XOR на необработанный субблок.
3. Субблоки меняются местами.

Операция *FL* является достаточно простой. Обрабатываемый ей субблок разбивается на два 16-битных фрагмента, над которыми выполняются следующие действия (рис. 3.103):

$$R' = R \oplus ((L \& KL_{i,1}) \lll 1);$$

$$L' = L \oplus ((R' | KL_{i,2}) \lll 1),$$

где:

- L* и *R* — входное значение левого и правого фрагмента соответственно;
- L'* и *R'* — выходные значения;

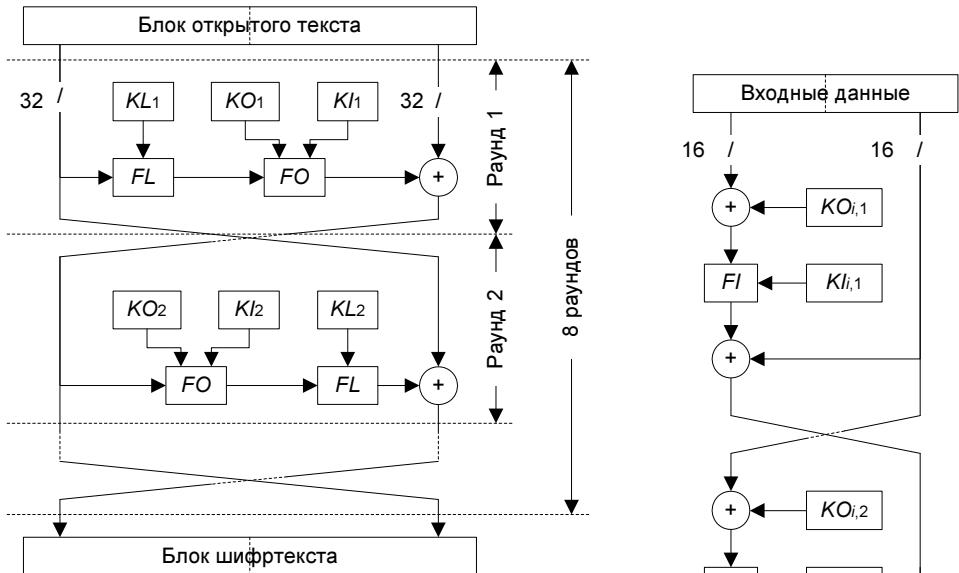
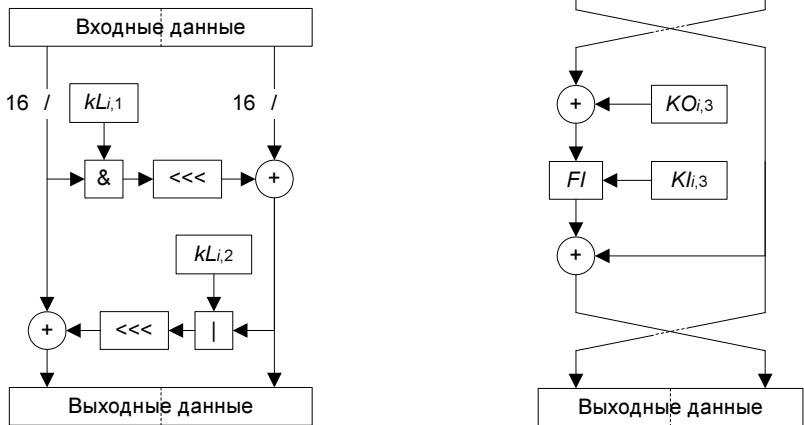


Рис. 3.102. Структура алгоритма KASUMI

Рис. 3.103. Операция  $FL$ Рис. 3.104. Операция  $FO$ 

- $KL_{i,1}$  и  $KL_{i,2}$  — фрагменты ключа  $i$ -го раунда для операции  $FL$  (процедура расширения ключа подробно описана далее);
- $\&$  и  $|$  — побитовые логические операции «и» и «или» соответственно;
- $<<<1$  — побитовый циклический сдвиг операнда на 1 бит влево.

## Описание алгоритмов

Операция  $FO$  более интересна — именно она является вложенной сетью Фейстеля (рис. 3.104). Как и ранее, входное значение разбивается на два 16-битных фрагмента, после чего выполняются 3 раунда следующих действий:

1. На левый фрагмент операцией XOR накладывается фрагмент ключа раунда  $KO_{i,j}$  ( $j$  — номер раунда операции  $FO$ ).
2. Выполняется зависящая от ключа операция  $FI$ .
3. На левый фрагмент накладывается операцией XOR значение правого фрагмента.
4. Фрагменты меняются местами.

Операция  $FI$  также представляет собой сеть Фейстеля, т. е. это уже третий уровень вложенности. В отличие от сетей Фейстеля на двух верхних уровнях, данная сеть является несбалансированной: обрабатываемый 16-битный фрагмент делится на две части: 9-битную левую и 7-битную правую. Затем выполняются 4 раунда, состоящие из следующих действий (рис. 3.105):

1. Левая часть «прогоняется» через таблицу замен. 9-битная часть (в нечетных раундах) обрабатывается таблицей  $S_9$ , а 7-битная (в четных раундах) — таблицей  $S_7$ . Эти таблицы полностью приведены в *Приложении 1*. Кроме того, учитывая тот факт, что алгоритм KASUMI предназначен преимущественно для аппаратной реализации, в том числе в устройствах, имеющих ограниченные ресурсы, биты выходного значения таблиц замен могут вычисляться из битов входного значения. В этих вычислениях используются только побитовые логические операции «и» и XOR; формулы вычислений также приведены в *Приложении 1*.
2. На левую часть операцией XOR накладывается текущее значение правой части. При этом, если справа 7-битная часть, она дополняется нулями слева, а у 9-битной части удаляются слева два бита.
3. Во втором раунде на левую часть операцией XOR накладывается фрагмент ключа раунда  $KI_{i,j,1}$ , а на правую — фрагмент  $KI_{i,j,2}$ . В остальных раундах эти действия не выполняются.
4. Левая и правая части меняются местами.

Для наглядности на рис. 3.105 жирными линиями выделен 9-битный поток данных.

Как видно, в алгоритме выполняется множество простых операций, скомбинированных весьма оригинальным образом.

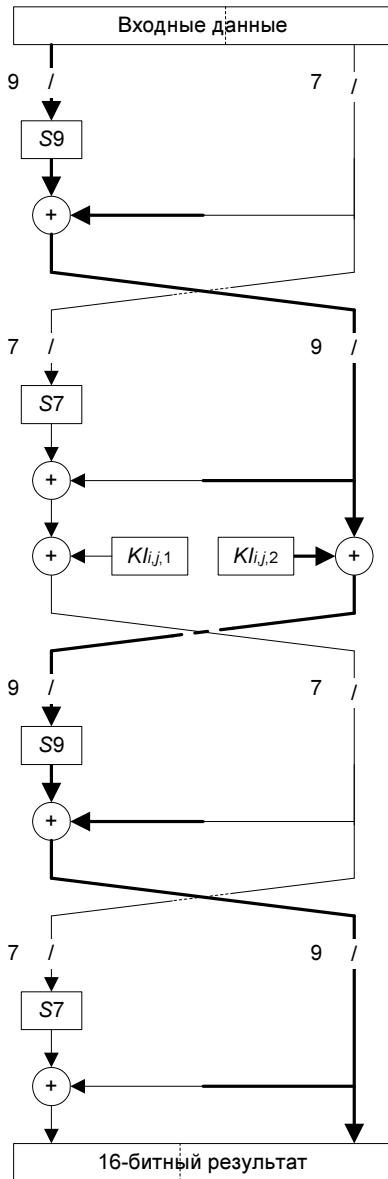


Рис. 3.105. Операция  $FI$

## Процедура расширения ключа

Задача процедуры расширения ключа состоит в формировании следующего набора используемых фрагментов ключа [365]:

- 16 фрагментов ключа  $KL_{i,k}$  ( $1 \leq i \leq 8$ ,  $1 \leq k \leq 2$ ), каждый из которых имеет размер по 16 битов;
- 24 16-битных фрагмента  $KO_{i,j}$  ( $1 \leq j \leq 3$ );
- 24 7-битных фрагмента  $KI_{i,j,1}$ ;
- 24 9-битных фрагмента  $KI_{i,j,2}$ .

Таким образом, процедура расширения ключа вычисляет 1024 бита ключевой информации из 128-битного ключа шифрования алгоритма KASUMI. Выполняется данное вычисление следующим образом:

1. 128-битный ключ делится на 8 фрагментов  $K1...K8$  по 16 битов каждый.
2. Формируются значения  $K1'...K8'$  согласно следующей формуле:

$$Kn' = Kn \oplus Cn,$$

где  $Cn$  — набор 16-битных констант, которые приведены в табл. 3.62 (указаны шестнадцатеричные значения).

Таблица 3.62

$C1$	0123
$C2$	4567
$C3$	89AB
$C4$	CDEF
$C5$	FEDC
$C6$	BA98
$C7$	7654
$C8$	3210

3. Необходимые фрагменты расширенного ключа «набираются» по мере выполнения преобразований из массивов  $K1...K8$  и  $K1'...K8'$  согласно табл. 3.63 и 3.64.

16-битный фрагмент  $KI_{i,j}$  делится на 7-битный фрагмент  $KI_{i,j,1}$  и 9-битный  $KI_{i,j,2}$ .

Таблица 3.63

Раунд	1	2	3	4
$KL_{i,1}$	$K1 <<< 1$	$K2 <<< 1$	$K3 <<< 1$	$K4 <<< 1$
$KL_{i,2}$	$K3'$	$K4'$	$K5'$	$K6'$
$KO_{i,1}$	$K2 <<< 5$	$K3 <<< 5$	$K4 <<< 5$	$K5 <<< 5$
$KO_{i,2}$	$K6 <<< 8$	$K7 <<< 8$	$K8 <<< 8$	$K1 <<< 8$
$KO_{i,3}$	$K7 <<< 13$	$K8 <<< 13$	$K1 <<< 13$	$K2 <<< 13$
$KI_{i,1}$	$K5'$	$K6'$	$K7'$	$K8'$
$KI_{i,2}$	$K4'$	$K5'$	$K6'$	$K7'$
$KI_{i,3}$	$K8'$	$K1'$	$K2'$	$K3'$

Таблица 3.64

Раунд	5	6	7	8
$KL_{i,1}$	$K5 <<< 1$	$K6 <<< 1$	$K7 <<< 1$	$K8 <<< 1$
$KL_{i,2}$	$K7'$	$K8'$	$K1'$	$K2'$
$KO_{i,1}$	$K6 <<< 5$	$K7 <<< 5$	$K8 <<< 5$	$K1 <<< 5$
$KO_{i,2}$	$K2 <<< 8$	$K3 <<< 8$	$K4 <<< 8$	$K5 <<< 8$
$KO_{i,3}$	$K3 <<< 13$	$K4 <<< 13$	$K5 <<< 13$	$K6 <<< 13$
$KI_{i,1}$	$K1'$	$K2'$	$K3'$	$K4'$
$KI_{i,2}$	$K8'$	$K1'$	$K2'$	$K3'$
$KI_{i,3}$	$K4'$	$K5'$	$K6'$	$K7'$

Расширение ключа может производиться «на лету», т. е. в процессе выполнения шифрования, что является несомненным достоинством алгоритма, особенно в условиях ограниченных ресурсов оперативной памяти.

## Использование алгоритма

Алгоритм KASUMI предназначен для обеспечения защиты данных и аутентификации в сетях сотовой связи. Для этих целей консорциум 3GPP разработал еще два алгоритма —  $f8$  и  $f9$ , каждый из которых использует в качестве ядра алгоритм KASUMI [159]. Алгоритм  $f8$  предназначен для шифрования передаваемых данных, а  $f9$  — для их аутентификации, т. е. в данном случае для обеспечения целостности данных. Алгоритмы  $f8$  и  $f9$  подробно описаны в спецификации [364]. Рассмотрим их вкратце.

Алгоритм  $f8$  представляет собой алгоритм потокового шифрования, в котором блочный шифр KASUMI используется для генерации псевдослучайной последовательности (*гаммы шифра*). Полученная последовательность  $S$  накладывается на шифруемые данные операцией XOR:

$$C = M \oplus S,$$

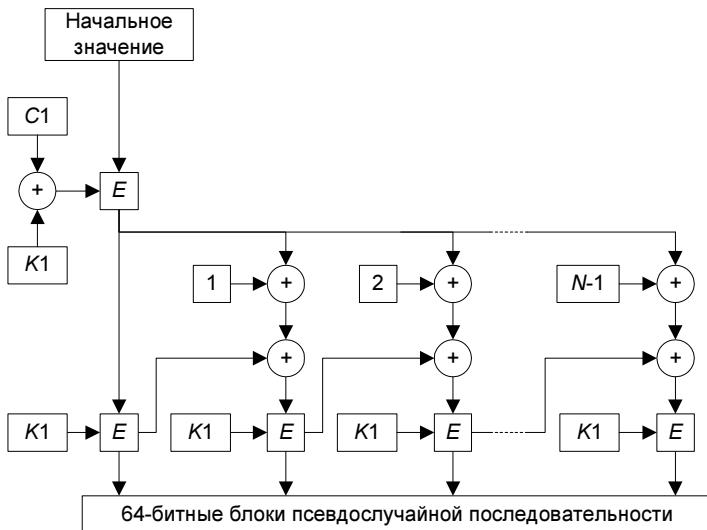
где  $M$  — шифруемое сообщение, а  $C$  — полученный в результате зашифровывания шифртекст.

Получатель шифртекста генерирует такую же псевдослучайную последовательность, которую накладывает повторно для получения исходной информации:

$$M = C \oplus S.$$

Структура алгоритма  $f8$  приведена на рис. 3.106. Алгоритм  $f8$  является достаточно простым:

1. Сначала формируется 64-битное начальное значение алгоритма, состоящее из следующих данных:
  - 32-битный номер генерируемой последовательности;
  - 5-битное значение, специфичное для конкретного пользователя;
  - бит, определяющий направление (входящие или исходящие) передачи шифруемых данных;
  - дополнение битовыми нулями до 64 битов.
2. Затем выполняется зашифровывание алгоритмом KASUMI (на рис. 3.106 обозначено как  $E$ ) начального значения на ключе, являющемся результатом применения операции XOR к ключу шифрования  $K1$  (уникальному для каждого пользователя) и константе  $C1$ , используемой для модификации ключа.
3. После этого для шифрования каждого 64-битного блока шифруемых данных (полного или неполного) следующим образом генерируется гамма шифра:
  - на результат шага 2 операцией XOR накладывается 64-битное значение, соответствующее номеру текущего блока (от 0 до  $N - 1$ , где  $N$  — требуемое количество блоков);

Рис. 3.106. Алгоритм  $f_8$ 

- к результату предыдущего действия применяется операция XOR с предыдущим блоком гаммы шифра; при генерации первого блока гаммы это действие не выполняется;
- выполняется зашифровывание результата предыдущей операции на ключе  $K1$ ; выходное значение и является вычисляемым блоком гаммы шифра.

Алгоритм  $f9$  вычисляет с помощью KASUMI код аутентификации сообщения (MAC), который используется для проверки целостности сообщений. MAC передается вместе с сообщением, вычисляется повторно на принимающей стороне, после чего результат вычисления сравнивается с присланным MAC.

Алгоритм  $f9$  состоит из следующих операций (рис. 3.107):

1. Защищаемое сообщение дополняется таким образом:

- перед сообщением вставляется 32-битный номер защищаемого сообщения, а также 32-битное случайное число;
- после сообщения добавляется бит направления (см. выше);
- результирующая последовательность при необходимости дополняется до размера, кратного 64 битам; дополнение производится единичным битом и необходимым количеством нулевых битов.

Последовательность разбивается на 64-битные блоки, каждый из которых шифруется «с зацеплением» (аналогично алгоритму  $f8$ ) на ключе  $K2$ , уникальном для каждого пользователя и предназначенном для аутентификации данных.

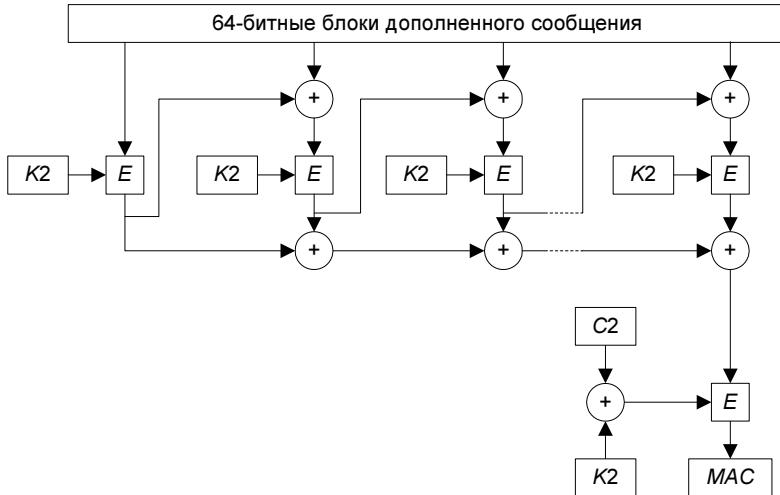


Рис. 3.107. Алгоритм  $f9$

2. Результаты зашифровывания блоков последовательности накладываются операциями XOR друг на друга.
  3. Результат предыдущей операции зашифровывается алгоритмом KASUMI на ключе, являющемся результатом применения операции XOR к ключу  $K_2$  и константе  $C_2$ , используемой для модификации ключа. MAC — левые 32 бита полученного 64-битного результата зашифровывания.

## Криптоанализ алгоритма

Считается, что криптоанализ алгоритма MISTY1 применим и к KASUMI (см., например, [184]). Есть как примеры атак, применимых к обоим алгоритмам (например, [230]), так и атак, применимых к MISTY, но неприменимых к KASUMI (например, [231]), и наоборот (например, [98]). Стоит сказать, что весьма мало различающиеся между собой алгоритмы IDEA и PES (см. разд. 3.26) имеют принципиально различные уровни криптостойкости. Поэтому рассмотрим здесь только те криптоаналитические результаты, которые имеют отношение именно к KASUMI.

Еще в процессе разработки алгоритм KASUMI был подвергнут первичному анализу командой разработчиков [159]. А после того, как спецификация алгоритма была определена, KASUMI был предоставлен для исследования трем независимым группам специалистов-криптологов. Причем криптоанализ проводился именно с учетом специфики использования алгоритма KASUMI — в качестве блочного шифра, на котором построены функции потокового

шифрования  $f8$  и аутентификации  $f9$ . В результате анализа все группы специалистов сделали вывод о том, что, во-первых, не обнаружено каких-либо атак на алгоритм KASUMI, осуществимых на практике; во-вторых, алгоритм полностью соответствует его предполагаемым применением. Однако, считая недостаточным время, отведенное на данные исследования, рабочей группой консорциума 3GPP было принято решение о повторном исследовании алгоритма и пересмотре данных выводов каждые 5 лет [159].

Когда криptoаналитики не могут обнаружить недостатки полнофункциональной версии исследуемого алгоритма, выполняется анализ урезанных версий, прежде всего вариантов алгоритма с уменьшенным количеством раундов. О криптостойкости алгоритма можно сделать косвенный вывод при наличии достаточного количества криptoаналитических работ, посвященных его урезанным версиям. Известны следующие криptoаналитические работы, посвященные модифицированным вариантам алгоритма KASUMI.

- Исследователи из Великобритании Марк Блэнден (Mark Blunden) и Эдриан Эскотт (Adrian Escott) предложили атаку на связанных ключах на 5-раундовую версию KASUMI. Для успешного осуществления данной атаки требуется наличие примерно  $2^{19}$  выбранных открытых текстов и около  $2^{33}$  тестовых операций шифрования. Данная атака стала возможной благодаря тому, что процедура расширения ключа алгоритма KASUMI существенно упрощена по сравнению с MISTY1 с целью упростить аппаратную реализацию алгоритма и ускорить процесс развертки ключа шифрования. Алгоритм MISTY1 с оригинальной процедурой расширения ключа подобной атаке не подвержен. Аналогичная атака возможна и на 6-раундовую версию KASUMI; для нее необходимо наличие такого же количества выбранных открытых текстов и около  $2^{112}$  попыток шифрования [98].
- Группа японских специалистов: Хидема Танака (Hidema Tanaka), Нобуюки Сугио (Nobuyuki Sugio) и Тошинобу Канеко (Toshinobu Kaneko), — успешно атаковала методом дифференциального криptoанализа 5-раундовую версию алгоритма, которая вскрывается при наличии  $2^{22}$  выбранных открытых текстов выполнением  $2^{63}$  операций шифрования [371].
- Несколько раньше Танака, Канеко и Чикаши Ишии (Chikashi Ishii) исследовали 4-раундовую модификацию алгоритма KASUMI, в которой отсутствовали функции  $FL$ . Не удивительно, что такой вариант оказался еще более слабым и для его вскрытия требуется всего 1416 выбранных открытых текстов и  $2^{22}$  тестовых операций шифрования [371].
- Ульрих Кюн (Ulrich Kuhn) из исследовательского подразделения Дрезденского банка предложил атаку на 6-раундовую версию алгоритма KASUMI

с использованием невозможных дифференциалов; для атаки необходимо наличие  $2^{55}$  выбранных открытых текстов и выполнение  $2^{100}$  операций шифрования [230].

Стоит отметить, что и эти атаки на усеченные версии алгоритма (за исключением атаки на алгоритм без функции *FL*) достаточно сложно осуществимы на практике.

Из всего вышесказанного можно было бы сделать однозначный вывод об отсутствии серьезных недостатков у алгоритма KASUMI и что данный алгоритм будет достаточно широко применяться уже в ближайшем будущем. Однако в последнее время были обнаружены новые методы атак на алгоритм KASUMI, существенно более эффективные, чем предыдущие; кроме того, возникли сомнения в эффективности самого алгоритма KASUMI.

- В 2005 г. Эли Бихам, Орр Данкельман (Orr Dunkelman) и Натан Келлер атаковали полнораундовую версию алгоритма KASUMI методом дифференциального криптоанализа на связанных ключах. Для успешной атаки требуется  $2^{54.6}$  выбранных открытых текстов и  $2^{76.1}$  операций зашифровывания (есть и вариант атаки с адаптивным выбором открытых текстов и шифртекстов, которой требуется меньше данных, но более сложная их обработка; оба варианта атаки являются теоретическими и маловероятно реализуемыми на практике). Кроме того, вышеперечисленные криптоаналитики существенно усилили упомянутую ранее атаку Кюна на 6-раундовую версию: новая атака требует наличия всего  $2^{13}$  открытых текстов и шифртекстов с адаптивным выбором и  $2^{13}$  операций шифрования [66].
- В конце 2005 г. специалист из Малайзии Йи Вей Лоу (Yee Wei Low) доказал, что далеко не на всех аппаратных платформах алгоритм KASUMI более эффективен, чем MISTY1. В частности, процедура расширения ключа KASUMI, несмотря на то, что она более проста, чем у MISTY1, выполняется почти в 2 раза дольше, чем в MISTY1 [236].
- А совсем недавно, в начале 2006 г., Эли Бихам, Натан Келлер и Элад Баркан (Elad Barkan) опубликовали работу, в которой рассмотрены практически реализуемые атаки на протоколы защиты в сотовых сетях второго поколения (GSM), в том числе на протокол, использующий в своей основе алгоритм KASUMI (еще не введенный в действие, но планируемый в ближайшее время) [42].

Алгоритм KASUMI все еще считается достаточно сильным. Однако исследования продолжаются.

## 3.28. Алгоритм Khazad

Алгоритм Khazad разработан специально для участия в конкурсе NESSIE (см. разд. 2.2) международным дуэтом специалистов: Пауло Баррето из Бразилии и Винсентом Риджменом из Бельгии. Название алгоритма произошло от названия Казад-Дум (Khazad-dum) — Подгорного Царства гномов из трилогии Дж. Р. Р. Толкиена «Властелин Колец» [395].

### Структура алгоритма

Предшественником алгоритма Khazad считается разработанный в 1995 г. Винсентом Риджменом и Джоан Деймен алгоритм SHARK (см. разд. 3.50). Авторы Khazad утверждают, что в основе алгоритма лежит стратегия разработки криптографически стойких алгоритмов шифрования (Wide-Tail Strategy), предложенная Джоан Деймен в работе [128].

Алгоритм Khazad шифрует 64-битные блоки данных с использованием 128-битного ключа шифрования. Этот алгоритм представляет блок данных в виде строки из 8 байтов. В каждом раунде алгоритма для обработки строки применяется следующая последовательность операций [44]:

1. Табличная замена  $\gamma$ : значение каждого байта строки заменяется с помощью таблицы  $S(x)$  (см. табл. 3.65, указаны шестнадцатеричные значения).

*Таблица 3.65*

A7	D3	E6	71	D0	AC	4D	79
3A	C9	91	FC	1E	47	54	BD
8C	A5	7A	FB	63	B8	DD	D4
E5	B3	C5	BE	A9	88	0C	A2
39	DF	29	DA	2B	A8	CB	4C
4B	22	AA	24	41	70	A6	F9
5A	E2	B0	36	7D	E4	33	FF
60	20	08	8B	5E	AB	7F	78
7C	2C	57	D2	DC	6D	7E	0D
53	94	C3	28	27	06	5F	AD
67	5C	55	48	0E	52	EA	42
5B	5D	30	58	51	59	3C	4E

Таблица 3.65 (окончание)

38	8A	72	14	E7	C6	DE	50
8E	92	D1	77	93	45	9A	CE
2D	03	62	B6	B9	BF	96	6B
3F	07	12	AE	40	34	46	3E
DB	CF	EC	CC	C1	A1	C0	D6
1D	F4	61	3B	10	D8	68	A0
B1	0A	69	6C	49	FA	76	C4
9E	9B	6E	99	C2	B7	98	BC
8F	85	1F	B4	F8	11	2E	00
25	1C	2A	3D	05	4F	7B	B2
32	90	AF	19	A3	F7	73	9D
15	74	EE	CA	9F	0F	1B	75
86	84	9C	4A	97	1A	65	F6
ED	09	BB	26	83	EB	6F	81
04	6A	43	01	17	E1	87	F5
8D	E3	23	80	44	16	66	21
FE	D5	31	D9	35	18	02	64
F2	F1	56	CD	82	C8	BA	F0
EF	E9	E8	FD	89	D7	C7	B5
A4	2F	95	13	0B	F3	E0	37

Эта таблица заменяет значение 0 на A7, 1 — на D3 и т. д.

Стоит сказать, что таблица полностью эквивалентна используемой в алгоритме Anubis (это еще один алгоритм шифрования, представленный на конкурс NESSIE теми же авторами, его подробное описание можно найти в разд. 3.5). Значения таблицы соответствуют следующему соотношению:

$$S(S(x)) = x.$$

2. Преобразование  $\theta$ : умножение байтовой строки данных на фиксированную матрицу  $H$  (табл. 3.66).

Таблица 3.66

1	3	4	5	6	8	B	7
3	1	5	4	8	6	7	B
4	5	1	3	B	7	6	8
5	4	3	1	7	B	8	6
6	8	B	7	1	3	4	5
8	6	7	B	3	1	5	4
B	7	6	8	4	5	1	3
7	B	8	6	5	4	3	1

3. Наложение материала ключа  $\sigma(K_i)$ : на блок данных операцией XOR накладывается 8-байтный фрагмент расширенного ключа  $K_i$  для текущего раунда  $i$ . Процедура расширения ключа подробно описана далее.

В алгоритме предусмотрено выполнение восьми раундов, перед первым из которых выполняется входное отбеливание — операция  $\sigma$  с фрагментом ключа  $K_0$  (раунды нумеруются, начиная с 1). В последнем раунде алгоритма не выполняется операция  $\theta$ , т. е. последовательность операций при зашифровании выглядит следующим образом:

$$\sigma(K_0) \rightarrow (\gamma \rightarrow \theta \rightarrow \sigma(K_1)) \rightarrow \dots \rightarrow (\gamma \rightarrow \theta \rightarrow \sigma(K_7)) \rightarrow \gamma \rightarrow \sigma(K_8).$$

Расшифровывание выполняется аналогично зашифрованию, но с использованием модифицированных фрагментов расширенного ключа, а именно:

$$\sigma(K_8) \rightarrow (\gamma \rightarrow \theta \rightarrow \sigma(\theta(K_1))) \rightarrow \dots \rightarrow (\gamma \rightarrow \theta \rightarrow \sigma(\theta(K_7))) \rightarrow \gamma \rightarrow \sigma(K_0).$$

Тот факт, что такая последовательность действий приведет к корректному расшифровыванию, доказан в спецификации алгоритма [44].

## Процедура расширения ключа

Процедура расширения ключа также весьма проста. Она выполняется в два этапа, на первом из которых инициализируются 64-битные переменные  $K_{-1}$  и  $K_{-2}$ : в качестве  $K_{-2}$  берутся первые 8 байтов исходного ключа шифрования, в качестве  $K_{-1}$  — оставшиеся 8 байтов.

Затем вычисляется последовательность подключей  $K_0 \dots K_8$  следующим образом:

$$K_i = f(C_i, K_{i-1}) \oplus K_{i-2},$$

где:

- $f(x, y)$  — функция раунда алгоритма, т. е. последовательная обработка 64-битной переменной  $x$  операциями  $\gamma$ ,  $\theta$  и  $\sigma(y)$ ;
- $C_i$  — набор 64-битных констант; каждый  $j$ -й байт константы  $C_i$  вычисляется так:

$$C_{i,j} = S(8i + j).$$

## Модификация алгоритма

Пользуясь возможностью незначительного изменения алгоритма в течение первого раунда конкурса, авторы Khazad также внесли изменения в свой алгоритм. В новом варианте спецификации алгоритма [44] исходный алгоритм Khazad назван Khazad-0, а название Khazad присвоено модифицированному алгоритму.

Изменения коснулись только таблицы замен: теперь для упрощения аппаратной реализации алгоритма таблица замен является суперпозицией двух других таблиц замен (замещающих 4-битные значения)  $P$  и  $Q$  (строку таблицы замен приведена на рис. 3.108). Таблицы  $P$  и  $Q$  приведены, соответственно, в табл. 3.67 и 3.68.

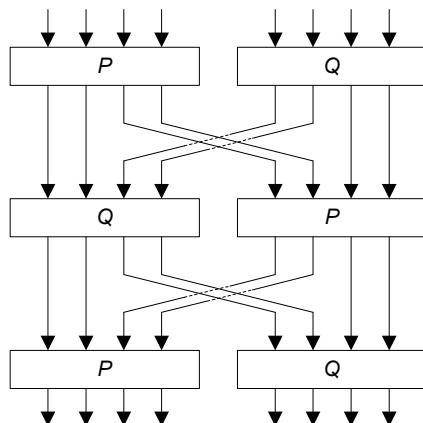


Рис. 3.108. Структура таблицы замен модифицированного алгоритма Khazad

Таблица 3.67

3	F	E	0	5	4	B	C	D	A	9	6	7	8	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Таблица 3.68

9	E	5	6	A	2	3	C	F	0	4	D	7	B	1	8
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

В зависимости от ресурсов шифратора таблица  $S$  может быть реализована как с помощью приведенных таблиц  $P$  и  $Q$ , так и напрямую. Сама модифицированная таблица  $S$  выглядит следующим образом (табл. 3.69):

**Таблица 3.69**

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

## Криптоанализ алгоритма Khazad

Первичный криптоанализ алгоритма был выполнен его авторами: в [44] утверждается, что Khazad обладает высокой криптостойкостью против линейного и дифференциального криптоанализа, использования усеченных дифференциалов, атак на связанных ключах и ряда других.

Алгоритм Khazad не привлек такого пристального внимания криптоаналитиков, как, например, победители конкурса NESSIE: алгоритмы MISTY1, Camellia и SHACAL, — известно существенно меньше работ, посвященных его криптоанализу, среди которых можно отметить следующие.

- Некоторые недочеты первичного криптоанализа Khazad (незначительно снижающие заявленную авторами алгоритма стойкость к линейному криптоанализу) были указаны авторами работы [65].
- В работе [272] предлагается атака на 5-раундовый Khazad методом интегрального криптоанализа (расширение дифференциального криптоанализа, исследует характеристики наборов текстов, шифруемых анализируемым алгоритмом), для которой требуется  $2^{64}$  открытых текстов (т. е., фактически, результаты зашифровывания всех возможных значений блока открытого текста).
- В работе [294] показано, что реализации Khazad легко защитить от атак по потребляемой мощности.
- Авторы [302] применили против алгоритма Khazad дифференциальный анализ на основе сбоев. Полнораундовый Khazad может быть вскрыт всего при наличии трех шифртекстов с внесенной ошибкой, однако модель атаки не является реалистичной.
- В работе [82] найден класс из  $2^{64}$  слабых ключей, при использовании которых 5-раундовый Khazad вскрывается при наличии  $2^{34}$  выбранных открытых текстов или выбранных шифртекстов.

Как видно, никаких сколько-нибудь серьезных проблем с криптостойкостью алгоритма Khazad обнаружено не было, что отмечено и экспертами конкурса NESSIE [305]. Кроме того, экспертами отмечена весьма высокая скорость шифрования данного алгоритма [311]. Khazad был признан перспективным алгоритмом, весьма интересным для дальнейшего изучения, но не стал одним из победителей конкурса из-за подозрений экспертов, что структура алгоритма может содержать скрытые уязвимости, которые могут быть обнаружены в будущем [305].

## 3.29. Алгоритмы Khufu и Khafre

Автор данных алгоритмов — Ральф Меркль (Ralph C. Merkle) — широко известен как один из изобретателей криптосистем с открытым ключом, т. е. асимметричного шифрования и электронной цифровой подписи. Существенно меньшую известность получили алгоритмы симметричного шифрования, разработанные Ральфом Мерклем: Khufu и Khafre.

### История создания алгоритмов

Ральфу Мерклю показалось несправедливым, что подавляющее большинство существовавших в то время (конец 1980-х гг.) алгоритмов симметричного шифрования было адаптировано под аппаратные реализации, несмотря на то, что аппаратная реализация алгоритма шифрования является, прежде всего, более дорогостоящей, чем программная, т. е. менее доступной для подавляющего большинства потенциальных пользователей. В результате Ральф Меркль (тогда являвшийся сотрудником исследовательского центра корпорации Xerox) создал семейство алгоритмов симметричного шифрования, взяв за основу два следующих постулата (описанных в спецификации алгоритмов Khufu и Khafre [264]).

- Программные реализации алгоритмов, предназначенные для работы на персональных компьютерах (а не на устройствах с ограниченными ресурсами типа смарт-карт), имеют в своем распоряжении практически неограниченный (по сравнению с аппаратными шифраторами) объем как оперативной, так и энергонезависимой памяти. Это означает, что алгоритм шифрования может использовать существенно большие объемы памяти для хранения таблиц замен, предварительно вычисленных ключей раундов и т. д.
- Алгоритм шифрования должен использовать именно те элементарные операции, которые оптимизированы для использования в программных реализациях. Меркль приводит в пример таблицы замен алгоритма DES, в котором, как известно, 32-битный шифруемый субблок после его расширения до 48 битов разделяется на 8 фрагментов по 6 битов, каждый из которых «прогоняется» через отдельную таблицу замен (заменяющую 6-битное входное значение 4-битным), в результате чего получается 32-битный субблок, обрабатываемый в дальнейшем. Тогда как аппаратный шифратор может выполнять все 8 замен параллельно, программная реализация алгоритма DES, работающая на однопроцессорном компьютере, выполняет замены последовательно, что занимает несравненно большее время. Меркль считает, что алгоритм, адаптированный для программных реализаций, должен выполнять прямую замену данных без разбивки на фрагменты. Соответственно, в каждом раунде алгоритмов Khufu и Khafre

таблица замен используется только один раз для замещения 8-битного входного значения 32-битным.

Алгоритмы Khufu и Khafre были разработаны в 1990 г. Необычные названия алгоритмов произошли от имен фараонов Древнего Египта — Хуфу (Хеопса) и Хафра, причем, Хафр являлся сыном Хуфу [395].

В следующем году корпорация Xerox получила патент на алгоритмы Khufu и Khafre [265], их коммерческое использование было запрещено держателем патента.

Алгоритмы тесно связаны между собой, хотя и адаптированы для различных применений. Рассмотрим поочередно их структуру.

## Структура алгоритма Khufu

Алгоритм Khufu шифрует данные блоками по 64 бита с использованием ключа шифрования переменного размера — от 64 до 512 битов; размер ключа должен быть кратен 64 битам [264].

Алгоритм выполняет  $R$  раундов преобразований, где  $R$  — переменное число, кратное 8. В каждом раунде производятся следующие действия (рис. 3.109):

1. Младший байт левой половины шифруемого блока (64-битный блок делится на два субблока по 32 бита) «прогоняется» через таблицу замен с получением 32-битного значения. В каждом октете (*октетом* в применении к данному алгоритму называется 8 раундов преобразований) используется своя таблица замен (таблицы замен будут подробно описаны далее).
2. Результат предыдущего шага накладывается операцией «исключающее или» (XOR) на правый субблок.
3. Над левым субблоком выполняется циклический сдвиг влево на различное число битов:
  - 8 — в раундах № 3 и 4 каждого октета;
  - 24 — в раундах № 7 и 8 каждого октета;
  - 16 — во всех остальных раундах.
4. Субблоки меняются местами.

Перед первым раундом выполняется наложение операцией XOR 32-битных подключей  $K_0$  и  $K_1$  на левый и правый субблоки соответственно. Аналогичные действия с подключами  $K_2$  и  $K_3$  выполняются после заключительного раунда алгоритма. Процедура расширения ключа (т. е. вычисления подключей на основе ключа шифрования) будет описана далее.

Количество раундов алгоритма  $R$  не является фиксированным, допускаются значения  $R$  от 8 до 64 включительно, кратные восьми; т. е. выполняется

от 1 до 8 октетов преобразований. Понятно, что 64-раундовое шифрование является максимально стойким, но, однако, неэффективно с точки зрения быстродействия. Автор алгоритма считает оптимальным выполнение 16, 24 или 32 раундов [264].

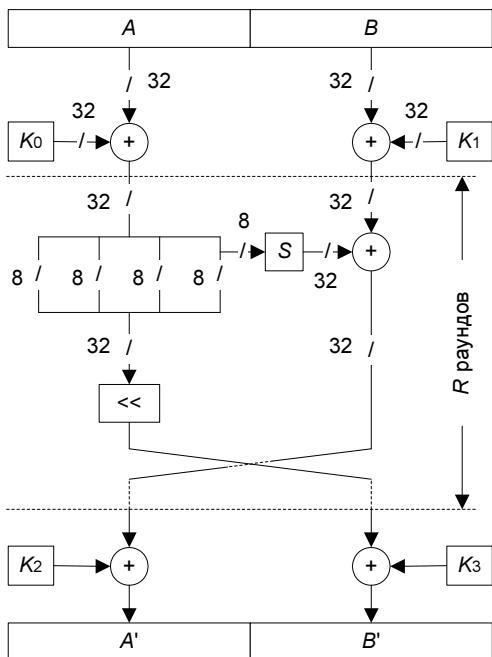


Рис. 3.109. Структура алгоритма Khufu

## Процедура расширения ключа и таблицы замен

Интересной особенностью алгоритма Khufu является зависимость таблиц замен от ключа шифрования. Таким образом, задача процедуры расширения ключа состоит не только в выработке подключей  $K_0 \dots K_3$ , но и в получении таблиц замен, соответствующих значению ключа.

Опишем процедуру расширения ключа:

- Выполняется инициализация 64-байтного блока данных, в который записывается ключ шифрования, а неиспользуемые биты обнуляются.

2. Этот блок данных шифруется алгоритмом Khufu в режиме сцепления блоков шифра (см. разд. 1.4) с использованием фиксированной стандартной таблицы замен и нулевых значений подключей  $K_0 \dots K_3$ . В результате получается 64-байтная псевдослучайная последовательность, зависящая от ключа шифрования. При необходимости этот шаг повторяется до получения последовательности требуемого размера.
3. Из последовательности, сгенерированной на шаге 2, «набираются» значения подключей  $K_0 \dots K_3$ .
4. На основе той же последовательности вычисляются значения таблиц замен для каждого раунда:
  - вычисляемая таблица замен инициализируется фиксированной исходной таблицей замен (см. далее);

Строки:	Столбцы:			
	0	1	2	3
0	64	F9	00	1B
1	FE	DD	CD	F6
2	7C	8F	F1	E2
3	11	D7	15	14
4	8B	8C	18	D3
5	DD	DF	88	1E
...	...	...	...	...
FE	33	7C	D1	29
FF	82	11	CE	FD
Входное значение	Выходное значение			

Рис. 3.110. Пример таблицы замен алгоритма Khufu

- в цикле по столбцам вычисляемой таблицы замен от 0-го до 3-го включительно (поскольку таблица замен замещает 8-битное значение 32-битным, ее можно представить в виде байтовой таблицы из 4 столбцов и 256 строк; фрагмент примера таблицы, приведенного в патенте [265], см. на рис. 3.110) выполняется цикл по строкам, в котором каждый

текущий байт (т. е. байт на пересечении текущей строки и текущего столбца внешнего цикла) меняется местами с одним из следующих байтов того же столбца, определяемым случайным образом (т. е. определяемым значением текущего байта псевдослучайной последовательности); результат — исходная таблица замен с «хаотично» переставленными байтами каждого столбца.

Упомянутая выше стандартная таблица замен формируется следующим образом:

1. Формируется исходная таблица замен, значение каждого байта которой эквивалентно номеру строки, в которой находится данный байт (т. е., например, все 4 байта 0-й строки имеют значение 0).
2. Выполняется перестановка байтов каждого столбца исходной таблицы аналогично шагу 4 процедуры расширения ключа, где в качестве псевдослучайной последовательности используется известный набор из миллиона случайных чисел фирмы RAND Corporation, опубликованный в 1955 г. [319].

В упомянутом выше патенте [265] приведен исходный код алгоритмов Khufu и Khafre на языке Си.

## Алгоритм Khafre

Алгоритм Khafre очень похож на Khufu. Между этими алгоритмами есть два принципиальных различия [264]:

- таблица замен алгоритма Khafre не зависит от ключа шифрования и является фиксированной — используется описанная выше стандартная таблица замен;
- добавлена дополнительная операция наложения ключа в каждом октете алгоритма (рис. 3.111).

В отличие от Khufu, алгоритм Khafre не ограничивает ни максимальное количество раундов алгоритма, ни максимальный размер ключа. Однако, как и для Khufu, размер ключа должен быть кратен 64 битам, а количество раундов — кратным 8. Процедура расширения ключа аналогична алгоритму Khufu. Однако, поскольку нет зависимости таблицы замен от ключа, из зависящей от ключа псевдослучайной последовательности только «набираются» значения используемых подключей  $K_0 \dots K_{n-1}$ , где  $n$  — количество подключей:

$$n = ((R/8) + 1) * 2.$$

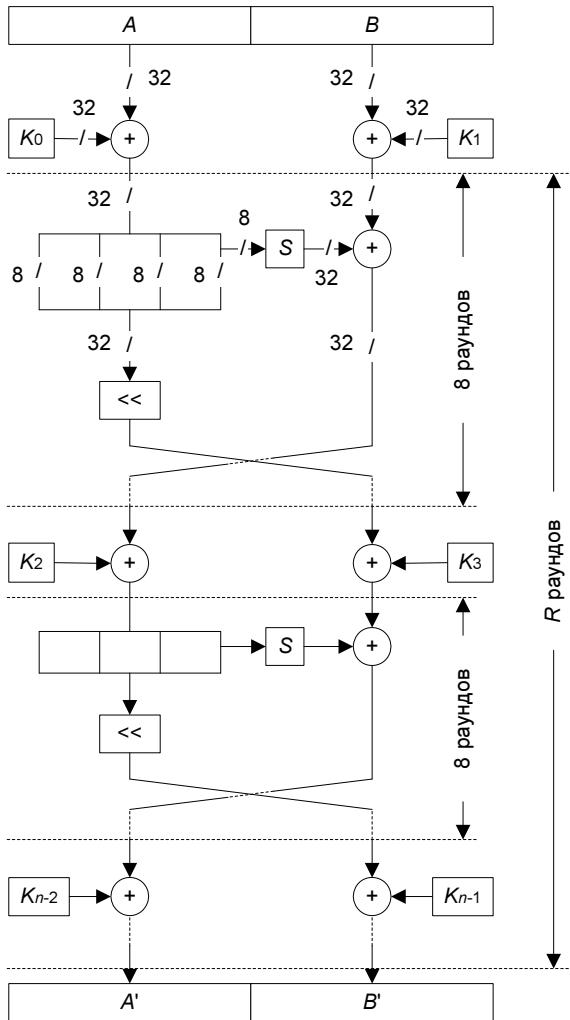


Рис. 3.111. Структура алгоритма Khafre

## Сравнение алгоритмов

Как видно из описаний алгоритмов, алгоритм Khufu выполняет достаточно длительную процедуру расширения ключа шифрования с целью не только вычисления подключей, но и модификации таблиц замен. А у алгоритма Khafre процедура расширения ключа является существенно более простой и быстрой. Однако собственно шифрование данных алгоритмом Khafre выполняется существенно медленнее по причине того, что отсутствие зависимости

таблиц замен от ключа в этом случае требует существенного увеличения количества раундов алгоритма для достижения того же уровня криптостойкости, что и у алгоритма Khufu с заданным количеством раундов. Зависимость таблиц замен алгоритма Khufu от ключа шифрования делает таблицы замен секретными для криptoаналитика и, соответственно, существенно усиливает стойкость алгоритма против дифференциального криptoанализа [28].

Отсюда следует и специфика использования этих алгоритмов [28, 264]:

- алгоритм Khufu должен использоваться там, где необходимо скоростное шифрование больших объемов данных с редкой сменой ключа;
- алгоритм Khafre, наоборот, идеален в тех случаях, когда требуется частая смена ключа в процессе шифрования данных.

## Алгоритм Snelru

Данное семейство алгоритмов представлено еще одним криптографическим алгоритмом — Snelru (также названным по имени фараона Древнего Египта), который представляет собой хэш-функцию, вычисляющую 128- или 256-битные хэш-значения [395].

В 1990 г. известные криптологи Эли Бихам и Эди Шамир нашли практически реализуемый метод атаки на алгоритм Snelru, использующий дифференциальный криptoанализ для нахождения коллизий (т. е. различных сообщений с совпадающим хэш-значением, см. [77]).

Алгоритм Snelru был усилен, однако усиленная версия также оказалась подверженной атаке с помощью дифференциального криptoанализа, хотя и существенно более сложно реализуемой на практике. В результате хэш-функция Snelru не получила широкого распространения.

## Криptoанализ алгоритмов

Как и многие другие алгоритмы, претендующие на замену стандарта DES, алгоритмы Khufu и Khafre подверглись многим исследованиям с целью проверки их криптостойкости. Начало положил сам изобретатель алгоритмов Khufu и Khafre, описавший в спецификации алгоритмов [264] несколько вариантов атак на 8-раундовый алгоритм Khufu. Из более поздних исследований наиболее значительными явились следующие результаты.

- В упомянутой выше работе [77] Бихам и Шамир показали возможность вскрытия методом дифференциального криptoанализа 16-раундового алгоритма Khafre при наличии  $3 * 2^{16}$  выбранных открытых текстов и соот-

вествующих им шифртекстов выполнением всего около 1500 операций шифрования. При наличии такого же количества известных открытых текстов и соответствующих им шифртекстов атака потребует около  $2^{38}$  шифрований. Подобным же образом вскрывается и 24-раундовый алгоритм Khafre — при наличии  $2^{60}$  шифртекстов выполнением около  $2^{53}$  операций шифрования.

- Эксперты из французской компании France Telecom предложили атаку на 16-раундовую версию алгоритма Khufu, позволяющую вычислить секретный ключ на основе  $2^{43}$  выбранных открытых текстов выполнением  $2^{43}$  операций шифрования [161].
- В 1999 г. была опубликована работа [64], в которой была описана более простая в реализации, чем предыдущие, атака на Khufu и Khafre на основе невозможных дифференциалов.
- В том же году профессор Калифорнийского Университета Дэвид Вагнер изобрел новый вид атак на алгоритмы шифрования, в которых анализируется эволюция зависимостей между двумя парами выбранных открытых текстов (атака методом бумеранга — см. разд. 1.6). Для выполнения этой атаки на 16-раундовый алгоритм Khufu требуется адаптивный выбор порядка  $2^{18}$  открытых текстов и порядка  $2^{18}$  операций шифрования. На данный момент эта атака является наиболее практически реализуемым методом вскрытия алгоритма Khufu.

### 3.30. Алгоритм LOKI97

#### История создания алгоритма

Алгоритм LOKI был разработан в 1989 г. профессорами Лоуренсом Брауном (Lawrence Brown), Йозефом Пьепржиком (Josef Pieprzyk) и Дженнифер Себерри (Jennifer Seberry) из Академии Вооруженных Сил Австралии [110]. Интересно происхождение названия алгоритма — от имени скандинавского бога Локи, которого издание [20] описывает как «Злобный бог, отец чудо-вищ. Приносил богам много вреда...». Впоследствии первоначальный алгоритм LOKI получил название LOKI89 для отличия от последующих версий алгоритма.

Алгоритм LOKI был замечен мировым криптологическим сообществом, в частности, благодаря его участию в австралийском конкурсе криptoалгоритмов Auscrypt90, проходившем в Сиднее в январе 1990 г. В 1991 г. появился алгоритм LOKI91 [107, 108], являющийся усилением предыдущего (разработан вышеперечисленными авторами при участии Мэтью Квана). Однако

в алгоритме было найдено несколько недостатков, не способствующих широкому распространению данного алгоритма, в частности [28, 182]:

- 64-битный секретный ключ алгоритма не мог противостоять атакам, выполняемым методом «грубой силы»;
- возможность атаки, базирующейся на связанных ключах и происходящей из-за слабости схемы расширения ключа, которая устанавливала повышенные требования к генераторам ключей шифрования, используемых в данном алгоритме;
- была обнаружена эффективность линейного криптоанализа против алгоритма LOKI91 с усеченным количеством раундов (до 12 раундов вместо 16 у основного варианта алгоритма).

В результате дальнейших исследований авторов алгоритма LOKI появился алгоритм LOKI97, которому и посвящен данный раздел.

## Основные характеристики и структура алгоритма

Алгоритм LOKI97 шифрует данные блоками по 128 битов и использует ключи размером 128, 192 или 256 битов, что соответствует основным требованиям конкурса AES (см. разд. 2.1).

Структура алгоритма представлена на рис. 3.112 [109]. Как видно из рисунка, алгоритм базируется на сети Фейстеля: данные шифруемого блока разбиваются на два субблока по 64 бита, над одним из которых выполняются следующие операции:

1. Сложение с первым фрагментом ключа раунда по модулю  $2^{64}$ . Процедура расширения ключа шифрования относительно сложна и подробно описана далее.
2. Наложение функции  $f()$ , выполняющей нелинейное преобразование данных и использующей в качестве аргумента второй фрагмент ключа раунда.
3. Снова сложение по модулю  $2^{64}$  с третьим фрагментом ключа раунда.

После выполнения перечисленных действий обработанный субблок накладывается на другой применением операции XOR, после чего субблоки меняются местами (за исключением последнего раунда).

Функция  $f()$  является достаточно сложной, она предусматривает следующие преобразования данных (рис. 3.113):

1. *KP* (keyed permutation) — простая перестановка входных данных на основе младших 32 битов используемого функцией  $f()$  фрагмента ключа раунда  $K_{3i-1}$ , где  $i$  — номер раунда. Перестановка выполняется так:
  - входные данные разбиваются на 2 субблока по 32 бита;

## Описание алгоритмов

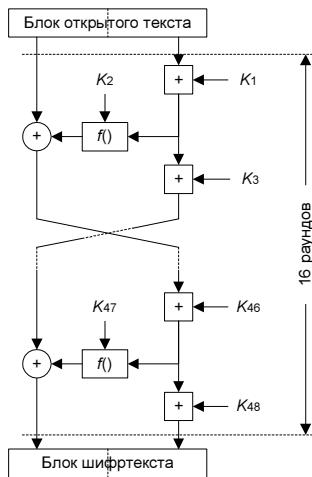
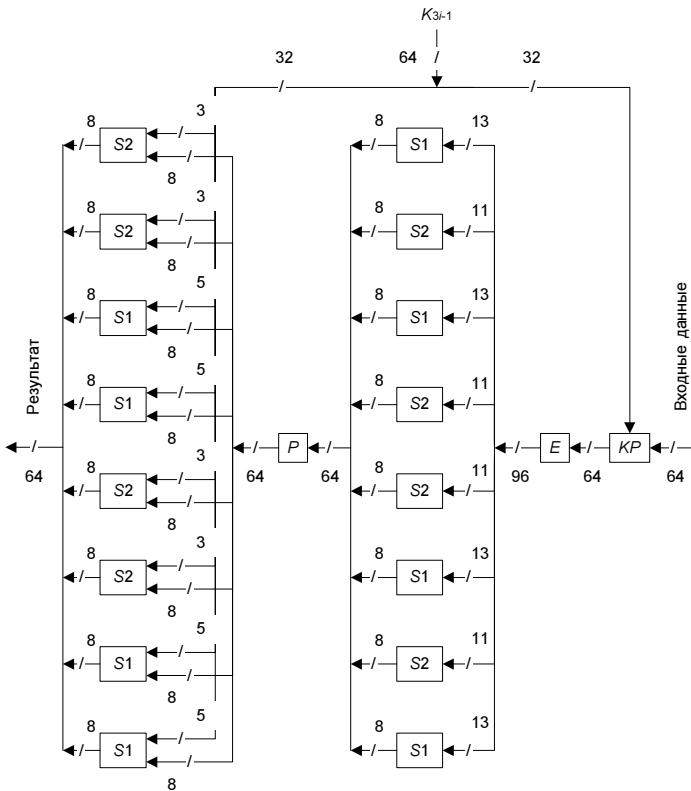


Рис. 3.112. Структура алгоритма LOKI97

Рис. 3.113. Операции, выполняемые функцией  $f()$

- для каждого единичного бита фрагмента ключа выполняется перестановка соответствующих ему битов субблоков между собой (например, нулевой бит ключа имеет значение 1; в этом случае нулевой бит первого субблока входных данных принимает значение нулевого бита второго субблока, и наоборот).
2.  $E$  (expansion) — генерация 96-битной последовательности на основе 64-битного результата операции  $KP$ . Последовательность формируется на основе следующих битов входа (здесь и далее 0-й бит — самый младший бит входной последовательности):
- $$[4 - 0,63 - 56,58 - 48,52 - 40,42 - 32,34 - 24,28 - 16,18 - 8,12 - 0].$$
3.  $S_1, S_2$  (substitution boxes) — таблицы замен;  $S_1$  обрабатывает 13-битные фрагменты 96-битной последовательности, полученной операцией  $E$ ,  $S_2$  — 11-битные фрагменты. Замена, фактически, выполняется путем возведения в куб инвертированного входного значения  $x$  в поле  $GF(2^{13})$  ( $S_1$ ) или  $GF(2^{11})$  ( $S_2$ ) согласно следующим формулам (в формулах указаны шестнадцатеричные константы):
- $$S_1(x) = ((x \oplus 1FFF)^3 \bmod 2911) \& FF;$$
- $$S_2(x) = ((x \oplus 7FF)^3 \bmod AA7) \& FF.$$
- В зависимости от наличия ресурсов в конкретных применениях алгоритма,  $S_1$  и  $S_2$  могут быть реализованы как в виде приведенных выше вычислений, так и напрямую в виде таблиц замен.
4.  $P$  (permutation) — побитовая перестановка входной последовательности согласно табл. 3.70.

Таблица 3.70

56	48	40	32	24	16	08	00	57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02	59	51	43	35	27	19	11	03
60	52	44	36	28	20	12	04	61	53	45	37	29	21	13	05
62	54	46	38	30	22	14	06	63	55	47	39	31	23	15	07

То есть 63-й бит входной последовательности становится 56-м битом результата, 62-й бит становится 48-м и т. д.

5. 64-битный результат перестановки  $P$  и 32-битная старшая часть фрагмента ключа раунда повторно обрабатываются таблицей замен. При этом входная 64-битная последовательность поровну (по 8 битов) распределяется между таблицами, а из фрагмента ключа берутся недостающие биты

до необходимых 13 битов входа для таблицы  $S_1$  или 11 битов входа для таблицы  $S_2$ . То есть входные последовательности для таблиц формируются так:

- [биты 63–61  $K_{3i-1}$ , биты 63–56 выхода операции  $P$ ] — 11 битов для  $S_2$ ;
- [биты 60–58  $K_{3i-1}$ , биты 55–48 выхода операции  $P$ ] — 11 битов для  $S_2$ ;
- [биты 57–53  $K_{3i-1}$ , биты 47–40 выхода операции  $P$ ] — 13 битов для  $S_1$  и т. д.

## Процедура расширения ключа

Процедура расширения ключа на основе ключа шифрования обеспечивает выработку 48 фрагментов ключей раундов  $K_1 \dots K_{48}$ , по три из которых используются в каждом раунде алгоритма, как описано выше.

В начале этой процедуры выполняется инициализация исходной 256-битной ключевой последовательности, которая, в зависимости от длины ключа шифрования, выполняется одним из следующих способов:

- 256-битный ключ используется в неизменном виде;
- 192-битный ключ дополняется результатом применения описанной выше функции  $f()$ , которая в качестве обрабатываемых данных (первый аргумент) использует первые 64 бита ключа шифрования, а в качестве подключа (второй аргумент) — следующие 64 бита ключа шифрования;
- 128-битный ключ дополняется результатом применения функции  $f()$  с обратным относительно 192-битного ключа порядком аргументов, а затем дополняется аналогичным 192-битному ключу образом.

Последующая обработка исходной ключевой последовательности основана на сети Фейстеля (рис. 3.114), после каждого раунда которой появляется фрагмент ключа раунда алгоритма. Таким образом, выполняется 48 раундов преобразований.

В каждом раунде три 64-битных фрагмента текущего состояния ключевой последовательности обрабатываются функцией  $g()$  и накладываются операцией XOR на оставшийся 64-битный фрагмент. Функция  $g()$  описывается следующим образом:

$$g(x, y, z) = f(x + y + \Delta * i, z),$$

где:

- $i$  — номер текущего раунда процедуры расширения ключа;
- $\Delta$  — константа, определяемая так:

$$\Delta = \lfloor (\sqrt{5} - 1) * 2^{63} \rfloor = 9E3779B97F4A7C15.$$

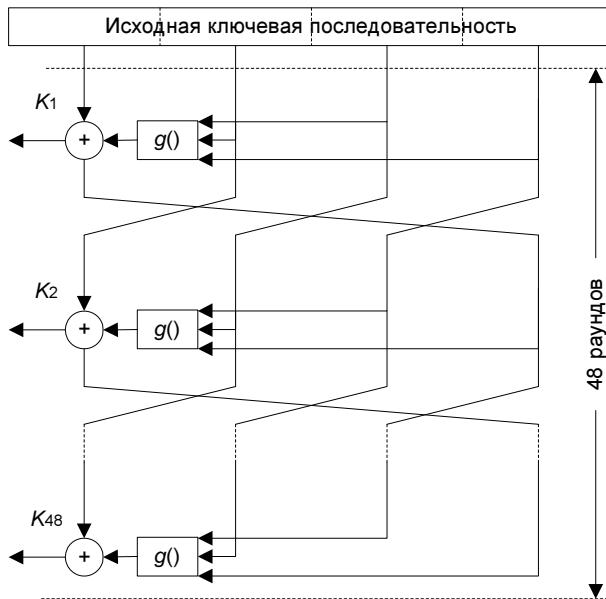


Рис. 3.114. Процедура расширения ключа

## Почему LOKI97 не вышел в финал конкурса AES

При детальном рассмотрении алгоритма в рамках конкурса AES экспертами были сделаны выводы о том, что данный алгоритм не имеет каких-либо преимуществ в криптостойкости по сравнению с другими алгоритмами, участвовавшими в конкурсе. При этом в алгоритме были найдены следующие недостатки [223, 284]:

- очень низкая (по сравнению с другими участниками конкурса) скорость шифрования;
- очень высокие требования к ресурсам, особенно к энергонезависимой памяти, что делает практически невозможной реализацию алгоритма в смарт-картах или аналогичных устройствах, имеющих ограниченные ресурсы (а возможность применения алгоритма в смарт-картах была одним из важных требований конкурса);
- возможность криптоатаки в случае наличия у злоумышленника порядка  $2^{56}$  пар «открытый текст — зашифрованный текст».

Любого из этих недостатков было достаточно для отсутствия алгоритма LOKI97 во втором раунде конкурса AES.

### 3.31. Алгоритм Lucifer

Принятый в США стандарт шифрования DES был основан на алгоритме Lucifer, разработанном в начале 1970-х гг. Алгоритм Lucifer и его история являются достаточно интересными и заслуживают отдельного рассмотрения.

Lucifer часто называют «первым алгоритмом шифрования для гражданских применений» [338, 395]. На самом деле Lucifer представляет собой не один алгоритм, а целое семейство связанных между собой (разработанных в рамках одноименной исследовательской программы по компьютерной криптографии фирмы IBM [28]), но созданных в разное время, алгоритмов блочного симметричного шифрования данных.

По словам Брюса Шнайера [28], «существует, по меньшей мере, два различных алгоритма с таким именем, что ... привело к заметной путанице». А посвященная алгоритму Lucifer статья в интернет-энциклопедии «Wikipedia» [395] упоминает о четырех вариантах этого алгоритма.

Рассмотрим поочередно различные варианты алгоритма Lucifer.

#### Вариант № 1

Исходный вариант алгоритма Lucifer был разработан коллективом специалистов из компании IBM под руководством Хорста Фейстеля (Horst Feistel). Этот вариант алгоритма был запатентован компанией IBM в 1971 г. (патент выдан в 1974 г.), его подробное описание можно найти в патенте США № 3798359 [144].

Этот вариант алгоритма шифрует данные блоками по 48 битов, используя при этом 48-битный ключ шифрования.

Алгоритм является подстановочно-перестановочной сетью. В процессе шифрования выполняется 16 раундов преобразований (это рекомендованное автором алгоритма количество раундов), в каждом из которых над обрабатываемым блоком данных производятся следующие действия (рис. 3.115):

1. Обеспечение обратной связи. Выполняется логическая операция «исключающее или» (XOR) между каждым битом обрабатываемого блока и предыдущим значением того же бита в случае, если аналогичный бит ключа раунда имеет единичное значение; в противном случае операция не выполняется. Предыдущие значения каждого обрабатываемого бита сохраняются в регистре блока обратной связи. В первом раунде алгоритма блок обратной связи операцию XOR не выполняет, а только запоминает биты обрабатываемого блока для следующего раунда.

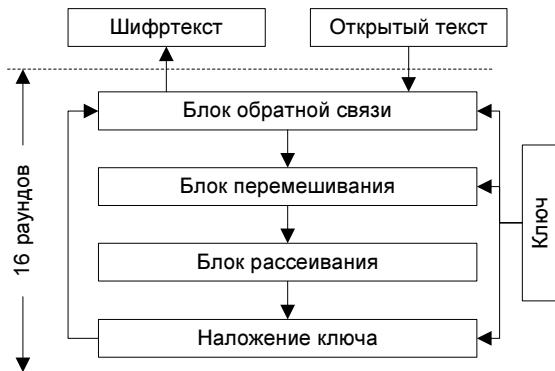


Рис. 3.115. Обобщенная схема варианта № 1

2. Перемешивающее преобразование. Выполняется нелинейное преобразование данных, полученных в результате предыдущей операции, путем табличной замены, зависящей от значения конкретного бита ключа раунда. Причем такая зависимость является достаточно простой: если значение управляющего бита равно 1, выполняется табличная замена  $S_0$ , в противном случае используется таблица  $S_1$ .

Каждый nibл данных заменяется независимо от других данных; таким образом, таблицы заменяют 4-битное входное значение также на 4-битное выходное.

Стоит сказать, что в патенте [144] не приведены конкретные значения таблиц замен; в качестве примера приведена следующая таблица:

$$2, 5, 4, 0, 7, 6, 10, 1, 9, 14, 3, 12, 8, 15, 13, 11.$$

Это означает, что входное значение 0 заменяется значением 2, значение 1 заменяется на 5 и т. д. до значения 15, которое заменяется на 11.

3. Рассеивающее преобразование, состоящее в простом перенаправлении входных битов таким образом, что значения всех входных битов меняются местами по определенному закону. Как и значения таблиц замен, сам закон, по которому выполняется рассеивание данных, не описан в патенте [144].

Эта операция выполняется абсолютно независимо от значения ключа шифрования.

4. Наложение ключа. Выполняется путем побитового применения операции XOR к результату предыдущей операции и соответствующим битам ключа раунда.

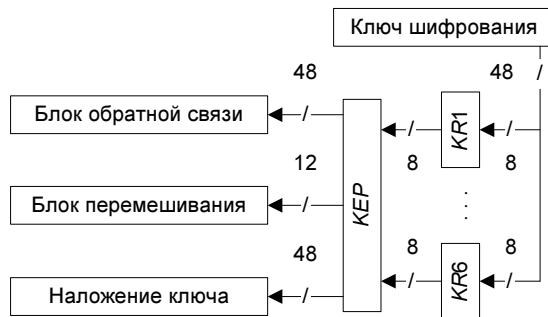
## Описание алгоритмов

Как видно из приведенного описания, в каждом раунде шифрования требуется 108 битов ключевой информации:

- 48 битов для блока обратной связи;
  - 12 битов для блока перемешивания;
  - 48 битов для блока наложения ключа.

Для получения 16 108-битных раундовых ключей из исходного 48-битного ключа шифрования выполняется процедура расширения ключа (рис. 3.116):

1. 48-битный ключ загружается в 8-битные регистры  $KR1...KR6$ .
  2. Из этих регистров «набирается» необходимое количество битов ключевой информации с помощью расширяющей перестановки  $KEP$ . Операция  $KEP$  не выполняет каких-либо вычислений, получение 108 битов ключевой информации из 48 происходит путем неоднократного использования определенных битов ключевых регистров  $KR1...KR6$ . Ключевая перестановка  $KEP$  не описана подробно в спецификации алгоритма.
  3. Для получения ключевой информации для следующего раунда выполняется побитовый циклический сдвиг на 1 бит содержимого каждого из ключевых регистров  $KR1...KR6$ . Затем снова производится перестановка  $KEP$ . Этот шаг повторяется в цикле необходимое число раз до завершения работы алгоритма.



**Рис. 3.116.** Процедура расширения ключа в варианте № 1

Стоит сказать о том, что данный алгоритм строго нацелен на аппаратное шифрование — в патенте [144] приведено несколько подробных схем, описывающих возможные аппаратные реализации алгоритма. Программному же шифрованию, практически, не удалено внимание в данном патенте.

Очевиден и тот факт, что в описании алгоритма [144] существует множество «белых птиц». Например, не приведены таблицы замен, нет подробного

описания используемого линейного преобразования, отсутствует описание перестановки *KEP* и т. д. Фактически патент устанавливает некий шаблон для разработки на его основе конкретных алгоритмов шифрования с «уточненными» процедурами.

Одновременно с патентом [144] 30 июня 1971 г. Хорстом Фейстелем были поданы еще две заявки на патенты; обе предлагали специфические применения описанного выше алгоритма:

- защиту данных, передаваемых и обрабатываемых в многотерминальных системах, с использованием алгоритма Lucifer; запатентованная схема подразумевает также обеспечение целостности данных, а также наличие двух режимов аутентификации пользователей: парольного и «запрос-ответ» [145];
- многоступенчатое шифрование данных, обеспечивающее различные уровни защиты информации при ее передаче в многотерминальных системах; эта схема также основана на применении алгоритма Lucifer [146].

Указанные патенты также были получены фирмой IBM в 1974 г.

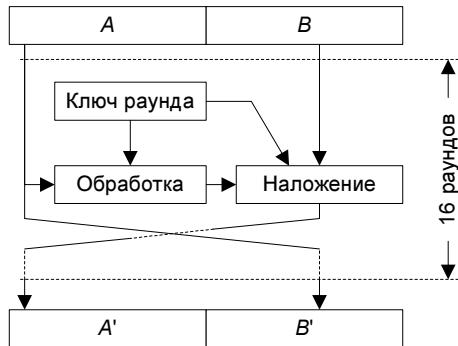
## Вариант № 2

По современным меркам 48-битный размер ключа шифрования является абсолютно недостаточным; видимо, подобные мысли были и у разработчиков алгоритма Lucifer, поэтому уже в том же 1971 г. появился второй вариант алгоритма, шифрующий данные 64-битным ключом. Однако второй вариант шифровал блоками всего по 32 бита, что также явно недостаточно (поскольку при  $b$ -битном блоке после зашифровывания порядка  $2^{b/2}$  блоков открытого текста на одном и том же ключе начинает происходить утечка информации о шифруемых данных — см., например, [120]).

Этот вариант алгоритма, как и предыдущий, запатентован фирмой IBM и описан в патенте [360].

Между вариантами № 2 и № 1 существенно больше различий, чем сходств. Второй вариант алгоритма разбивает шифруемый 32-битный блок данных на два субблока по 16 битов и выполняет 16 раундов преобразований, в каждом из которых выполняется следующая последовательность операций (общая структура алгоритма приведена на рис. 3.117):

1. Шифруемый блок данных разбивается на два субблока: *A* и *B*.
2. Субблок *A* разбивается на 4 фрагмента по 4 бита, каждый из которых обрабатывается раздельно (см. рис. 3.118), т. е. остальные операции повторяются для каждого фрагмента.



**Рис. 3.117.** Структура варианта № 2

3. Каждый из фрагментов складывается по модулю 16 с  $KA_{i,n}$  — 4-битным фрагментом ключа раунда для операции сложения (процедура расширения ключа шифрования будет описана далее), где:
    - $n$  — номер обрабатываемого фрагмента;
    - $i$  — номер текущего раунда.
  4. Над каждым из фрагментов выполняется зависящая от ключа табличная замена: обрабатываемый фрагмент заменяется 4-битным значением  $T0..T3$  согласно табл. 3.71.

### **Таблица 3.71**

Входное значение																Резуль-тат
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
$\sim k$	$\sim k$	$k$	1	$\sim k$	1	$\sim k$	0	$k$	$k$	0	$k$	$\sim k$	$k$	$k$	$\sim k$	T0
1	$\sim k$	1	0	$\sim k$	$\sim k$	0	$k$	$k$	1	0	0	$k$	0	$\sim k$	$k$	T1
$k$	1	1	$\sim k$	1	$k$	$k$	0	$\sim k$	$\sim k$	$\sim k$	$k$	$k$	$k$	0	0	T2
$k$	$\sim k$	$\sim k$	$k$	$k$	1	$\sim k$	0	0	$k$	1	$\sim k$	$\sim k$	$k$	$\sim k$	$k$	T3

где:

- $k = KS_i[n]$ , представляет собой  $n$ -й бит 4-битного фрагмента ключа раунда, управляющего заменами ( $KS_i$ ),
  - $\sim k$  — обратное значение для  $k$ .

Согласно входному значению, из таблицы выбираются битовые значения  $T0...T3$ , например, для входного значения 14 и  $k=1$  будут получены следующие значения:

$$T0 = 1;$$

$$T1 = 0;$$

$$T2 = 0;$$

$$T3 = 0.$$

5. Выполняется наложение операцией XOR последовательности  $T0...T3$ , представляющей собой результат обработки фрагмента субблока  $A$ , на определенные биты субблока  $B$ . Биты субблока  $B$ , участвующие в данной операции, выбираются с участием определенного 4-битного фрагмента ключа раунда  $KX_{i,n}$  согласно табл. 3.72.

**Таблица 3.72**

Значение управляющего бита фрагмента ключа $KX_{i,n}$		
	1	0
$T0$	$B_{0,(n+1 \bmod 4)}$	$B_{0,n}$
$T1$	$B_{1,(n+2 \bmod 4)}$	$B_{1,(n+3 \bmod 4)}$
$T2$	$B_{2,(n+3 \bmod 4)}$	$B_{2,(n+2 \bmod 4)}$
$T3$	$B_{3,n}$	$B_{3,(n+1 \bmod 4)}$

Управляющим битом для  $TY$  ( $Y = 0...3$ ) является бит №  $Y$  фрагмента  $KX_{i,n}$ .

Например, при обработке фрагмента № 0 и всех единичных битах  $KX_{i,n}$  выполняются следующие действия (см. рис. 3.118 — все данные, вовлеченные в обработку 0-го фрагмента субблока при всех единичных битах  $KX_{i,n}$ , выделены жирными линиями):

$$B_{0,1} = B_{0,1} + T0;$$

$$B_{1,2} = B_{1,2} + T1;$$

$$B_{2,3} = B_{2,3} + T2;$$

$$B_{3,0} = B_{3,0} + T3.$$

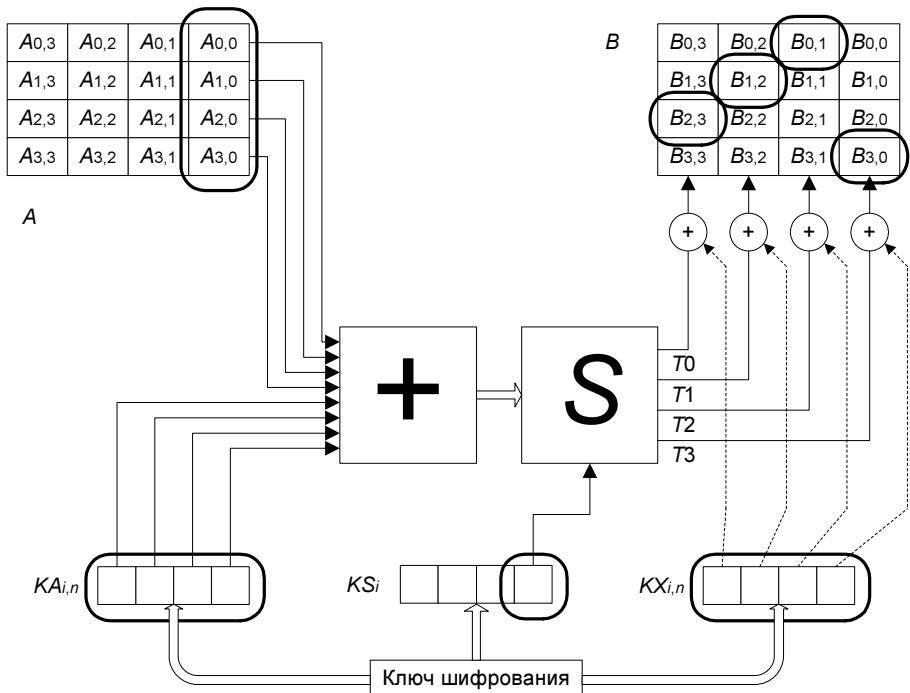


Рис. 3.118. Обработка фрагмента субблока A (вариант № 2)

После каждого раунда, кроме последнего, субблоки меняются местами.

Расширение ключа, т. е. получение необходимого количества фрагментов ключей раундов:

- KA<sub>i,0</sub>...KA<sub>i,3</sub> — для сложения по модулю 16;
- KS<sub>i</sub> — для участия в табличной замене;
- KX<sub>i,0</sub>...KX<sub>i,3</sub> — для контроля наложения результата на субблок B, выполняется весьма простым способом:

1. 64-битный ключ шифрования делится на 16 частей по 4 бита, которые нумеруются от 0 до 15.
2. При необходимости выбирается определенный фрагмент ключа согласно табл. 3.73.

Этот вариант алгоритма описан гораздо более подробно, чем предыдущий: в патенте [360] «белые пятна» отсутствуют, поэтому данный алгоритм может быть вполне реализован только с использованием содержащейся в патенте [360] информации.

Таблица 3.73

Раунд	$KS_i$	$KA_{i,0}$	$KX_{i,0}$	$KA_{i,1}$	$KX_{i,1}$	$KA_{i,2}$	$KX_{i,2}$	$KA_{i,3}$	$KX_{i,3}$
1	0	1	2	3	4	5	6	7	8
2	9	10	11	12	13	14	15	0	1
3	2	3	4	5	6	7	8	9	10
4	11	12	13	14	15	0	1	2	3
5	4	5	6	7	8	9	10	11	12
6	13	14	15	0	1	2	3	4	5
7	6	7	8	9	10	11	12	13	14
8	15	0	1	2	3	4	5	6	7
9	8	9	10	11	12	13	14	15	0
10	1	2	3	4	5	6	7	8	9
11	10	11	12	13	14	15	0	1	2
12	3	4	5	6	7	8	9	10	11
13	12	13	14	15	0	1	2	3	4
14	5	6	7	8	9	10	11	12	13
15	14	15	0	1	2	3	4	5	6
16	7	8	9	10	11	12	13	14	15

Как и в предыдущем случае, в патенте [360] приведена подробнейшая информация для аппаратной реализации алгоритма, особенности программной реализации в патенте не рассматриваются.

### Вариант № 3

Это наименее подробно описанный вариант алгоритма Lucifer, его описание приведено в статье [25]. Судя по описанию, алгоритм представляет собой последовательность следующих операций (рис. 3.119):

- управляемые контрольными битами секретного ключа табличные замены — аналогично варианту № 1, в зависимости от значения контрольного бита ключа выполняется замена  $S_0$  или  $S_1$ ;
- фиксированные битовые перестановки  $P$ .

## Описание алгоритмов

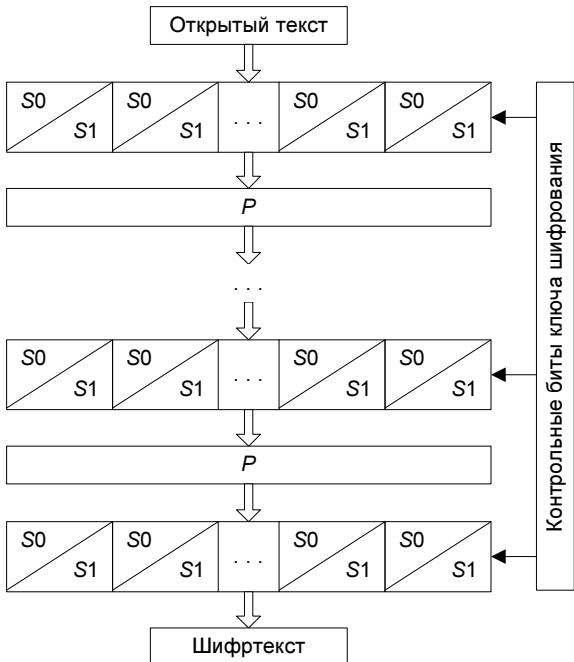


Рис. 3.119. Структура варианта № 3

Многоократное повторение этих операций и представляет собой алгоритм шифрования.

В статье [25] не указывается подавляющее большинство параметров алгоритма: нет ни точного количества раундов, ни значений таблиц замен и перестановок, не указаны размер блока и ключа (128-битные блоки и ключи указываются лишь в качестве возможных значений). Таким образом, данный вариант алгоритма Lucifer является еще более «шаблонным», чем вариант № 1.

Стоит отметить тот факт, что даже всемирно известные эксперты в области криптографии делают различные предположения о том, какой из вариантов алгоритма Lucifer был предшественником алгоритма DES. Например, в [263] «прямым предшественником» DES назван алгоритм, описанный в патенте [360] (т. е. вариант № 2), тогда как в [28] сказано, что DES разработан на базе 128-битного алгоритма Lucifer, т. е. варианта № 3 (вариант № 4 появился существенно позже, чем DES, поэтому предшественником быть не мог). Видимо, решающим фактором в пользу алгоритма № 3 стоит считать тот факт, что описывающий DES патент [143] ссылается именно на статью [25] в качестве первоисточника.

## Вариант № 4

Можно утверждать, что вариант № 4 похож одновременно на три предыдущих. Как и в варианте № 2, здесь выполняется разбиение 128-битного шифруемого блока на два субблока, один из которых обрабатывается следующим образом (рис. 3.120) [338]:

1. Выполняется наложение ключа раунда путем применения операции XOR к битам обрабатываемого субблока и 64-битного фрагмента ключа раунда  $KX_i$  ( $i$  — номер текущего раунда).

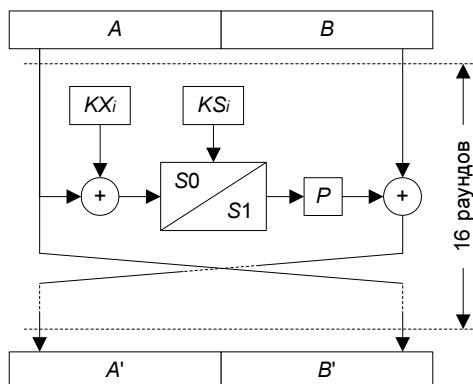


Рис. 3.120. Структура варианта № 4

2. Управляемая ключом табличная замена выполняется весьма похоже на варианты № 1 и № 3:

- если значение  $j$ -го бита ( $j = 0 \dots 7$ ) 8-битного управляющего фрагмента ключа раунда  $KS_i$  равно 1, nibлы  $j$ -го байта обрабатываемого субблока меняются местами;
- старший nibл каждого байта «прогоняется» через таблицу замен  $S_0$ , младший — через  $S_1$ . Таблицы определены следующим образом (табл. 3.74).

Таблица 3.74

	Входное значение															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_0$	12	15	7	10	14	13	11	0	2	6	3	1	9	4	5	8
$S_1$	7	2	14	9	3	11	0	4	12	13	1	10	6	15	8	5

3. Выполняется фиксированная перестановка ( $P$ ) битов субблока согласно табл. 3.75.

Таблица 3.75

10	21	52	56	27	1	47	38	18	29	60	0	35	9	55	46
26	37	4	8	43	17	63	54	34	45	12	16	51	25	7	62
42	53	20	24	59	33	15	6	50	61	28	32	3	41	23	14
58	5	36	40	11	49	31	22	2	13	44	48	19	57	39	30

Значение входного бита 10 (здесь биты нумеруются слева направо, начиная с 0-го) помещается в выходной бит 0, значение 21-го бита — в бит 1 и т. д.

4. Результат предыдущей операции накладывается на необработанный субблок операцией XOR.

5. Субблоки меняются местами (во всех раундах, кроме последнего).

Всего выполняется 16 раундов описанных выше преобразований.

Процедура расширения ключа решает задачу получения 16 ключей раундов по 72 бита каждый (64 бита  $KX_i$  и 8 битов  $KS_i$ ) из исходного 128-битного ключа шифрования. Расширение выполняется весьма простым способом:

1. В качестве фрагмента  $KS_i$  используется первый байт ключа шифрования (считая байты ключа слева направо, начиная с 1).
2. В качестве фрагмента  $KX_i$  используются первые 8 байтов ключа шифрования (т. е. первый байт ключа используется в обоих фрагментах).
3. Ключ шифрования циклически сдвигается влево на 7 байтов, после чего аналогично «набираются» фрагменты ключа для следующего раунда.

В отличие от вариантов № 1 и № 3, вариант № 4 описан достаточно подробно для его реализации, как программной, так и аппаратной.

## Криптоанализ вариантов алгоритма

Какие-либо криптоаналитические работы, посвященные вариантам № 1 и № 2 алгоритма Lucifer, не получили широкую известность. Двум последним вариантам «повезло» больше; известны следующие результаты их криптоанализа.

- В 1991 г. Эли Бихам и Эди Шамир исследовали вариант № 3 [77]. Для определенности они использовали перестановку  $P$  из алгоритма DES, а в качестве таблиц  $S_0$  и  $S_1$  были взяты строки 3 и 4 таблицы замен  $S_1$  алгоритма DES (табл. 3.76).

Таблица 3.76

	Входное значение															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_0$	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
$S_1$	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Согласно [77], 8-раундовый вариант № 3 с 32-битным блоком вскрывается при наличии всего 24 выбранных открытых текстов и соответствующих им шифртекстов путем выполнения порядка  $2^{21}$  тестовых операций шифрования.

В той же работе Бихам и Шамир описали возможную атаку на аналогичный алгоритм со 128-битным блоком, для успешного осуществления которой требуется 60–70 выбранных открытых текстов и порядка  $2^{53}$  операций шифрования.

Кроме того, Бихам и Шамир утверждали, что вариант № 4 является еще более слабым.

- Последнее утверждение было доказано в работе [48], которую опубликовали Ишаи Бен-Аройя и Эли Бихам в 1993 г. В ней описана атака на вариант № 4 алгоритма Lucifer, в котором обнаружилось подмножество ключей (весыма большое — около 55 % всех возможных значений ключа), слабых к дифференциальному криптоанализу. При использовании ключа шифрования из данного подмножества алгоритм вскрывается при наличии  $2^{36}$  выбранных открытых текстов.

Данные криптоаналитические исследования развеяли широко распространенный миф о том, что «превращение» алгоритма Lucifer в DES ослабило алгоритм. Статьи [48] и [77] доказывают обратное — Lucifer существенно менее стоеч, чем DES.

### 3.32. Алгоритм MacGuffin

Алгоритм MacGuffin (или McGuffin) был разработан в 1994 г. известными криптологами Брюсом Шнайером и Мэттом Блейзом.

В том же 1994 г. Брюс Шнайер и Джон Келси представили работу [346], описывающую теоретические основы несбалансированных сетей Фейстеля (см. разд. 1.3). По словам авторов алгоритма MacGuffin, при разработке данного алгоритма не ставилась цель разработать криптографически стойкий

и эффективный алгоритм шифрования (хотя это было бы немаловажным «побочным эффектом»); цель MacGuffin — привлечение внимания криптологического сообщества к изучению и криптоанализу обобщенных несбалансированных сетей Фейстеля. Авторы частично добились цели — алгоритм MacGuffin привлек внимание криптоаналитиков и был вскрыт в том же 1994 г. (см. далее), однако впоследствии подобная структура использовалась в алгоритмах шифрования крайне редко.

## Структура алгоритма

Алгоритм MacGuffin шифрует данные 64-битными блоками с использованием 128-битного ключа шифрования. 64-битный блок данных разбивается на два субблока: 48-битный и 16-битный. В каждом раунде алгоритма больший из субблоков определенным образом обрабатывается и накладывается операцией XOR на меньший субблок (это можно представить и как обработку трех 16-битных фрагментов блока и их наложение на четвертый — см. рис. 3.121). Обработка включает в себя следующие операции [96]:

1. Обрабатываемые 16-битные фрагменты складываются операцией XOR с фрагментами расширенного ключа  $K_i$ , где  $i$  — номер текущего раунда (процедура расширения ключа будет описана далее).
2. Выполняется перестановка  $P$ , переставляющая биты входных данных перед их табличной заменой; перестановка выполняется согласно табл. 3.77.

Таблица 3.77

Таблицы замен	Биты входных данных					
	0	1	2	3	4	5
$S_1$	$a_2$	$a_5$	$b_6$	$b_9$	$c_{11}$	$c_{13}$
$S_2$	$a_1$	$a_4$	$b_7$	$b_{10}$	$c_8$	$c_{14}$
$S_3$	$a_3$	$a_6$	$b_8$	$b_{13}$	$c_0$	$c_{15}$
$S_4$	$a_{12}$	$a_{14}$	$b_1$	$b_2$	$c_4$	$c_{10}$
$S_5$	$a_0$	$a_{10}$	$b_3$	$b_{14}$	$c_6$	$c_{12}$
$S_6$	$a_7$	$a_8$	$b_{12}$	$b_{15}$	$c_1$	$c_5$
$S_7$	$a_9$	$a_{15}$	$b_5$	$b_{11}$	$c_2$	$c_7$
$S_8$	$a_{11}$	$a_{13}$	$b_0$	$b_4$	$c_3$	$c_9$

где  $a_x$ ,  $b_x$  и  $c_x$  — биты № $x$  входных фрагментов  $A$ ,  $B$  и  $C$  соответственно — см. рис. 3.121.

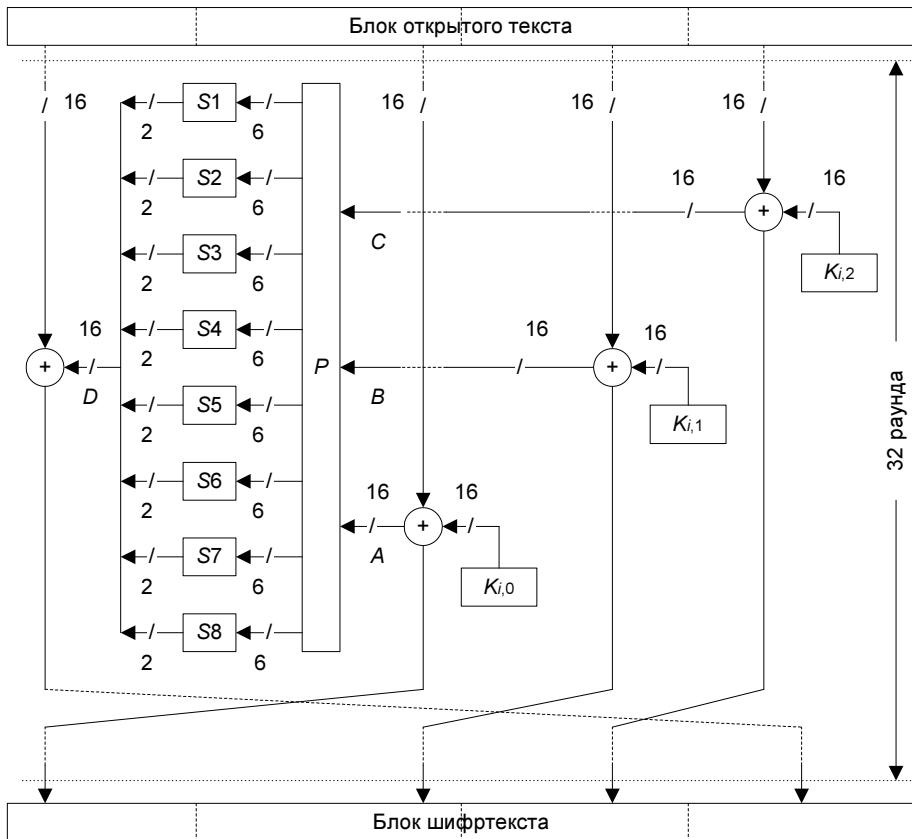


Рис. 3.121. Алгоритм MacGuffin

3. Табличные замены  $S_1 \dots S_8$  замещают 6-битное входное значение 2-битным. Таблицы замен  $S_1 \dots S_8$  определены следующим образом:

Таблица  $S_1$ :

2, 0, 0, 3, 3, 1, 1, 0, 0, 2, 3, 0, 3, 3, 2, 1, 1, 2, 2, 0, 0, 2, 2, 3, 1, 3, 3, 1, 0, 1, 1, 2, 0, 3, 1, 2, 2, 2, 2, 0, 3, 0, 0, 3, 0, 1, 3, 1, 3, 1, 2, 3, 3, 1, 1, 2, 1, 2, 2, 0, 1, 0, 0, 3.

Таблица  $S_2$ :

3, 1, 1, 3, 2, 0, 2, 1, 0, 3, 3, 0, 1, 2, 0, 2, 3, 2, 1, 0, 0, 1, 3, 2, 2, 0, 0, 3, 1, 3, 2, 1, 0, 3, 2, 2, 1, 2, 3, 1, 2, 1, 0, 3, 3, 0, 1, 0, 1, 3, 2, 0, 2, 1, 0, 2, 3, 0, 1, 1, 0, 2, 3, 3.

Таблица  $S_3$ :

2, 3, 0, 1, 3, 0, 2, 3, 0, 1, 1, 0, 3, 0, 1, 2, 1, 0, 3, 2, 2, 1, 1, 2, 3, 2, 0, 3, 0, 3, 2, 1, 1, 3, 1, 0, 2, 0, 3, 3, 0, 2, 0, 3, 3, 1, 2, 0, 1, 3, 0, 1, 3, 0, 2, 2, 1, 1, 3, 2, 1, 2, 0, 1, 2.

Таблица  $S_4$ :

1, 3, 3, 2, 2, 3, 1, 1, 0, 0, 0, 3, 3, 0, 2, 1, 1, 0, 0, 1, 2, 0, 1, 2, 3, 1, 2, 2, 0, 2, 3, 3, 2, 1, 0, 3, 3, 0, 0, 0, 2, 2, 3, 1, 1, 3, 3, 2, 3, 3, 1, 0, 1, 1, 2, 3, 1, 2, 0, 1, 2, 0, 0, 2.

Таблица  $S_5$ :

0, 2, 2, 3, 0, 0, 1, 2, 1, 0, 2, 1, 3, 3, 0, 1, 2, 1, 1, 0, 1, 3, 3, 2, 3, 1, 0, 3, 2, 2, 3, 0, 0, 3, 0, 2, 1, 2, 3, 1, 2, 1, 3, 2, 1, 0, 2, 3, 3, 0, 3, 3, 2, 0, 1, 3, 0, 2, 1, 0, 0, 1, 2, 1.

Таблица  $S_6$ :

2, 2, 1, 3, 2, 0, 3, 0, 3, 1, 0, 2, 0, 3, 2, 1, 0, 0, 3, 1, 1, 3, 0, 2, 2, 0, 1, 3, 1, 1, 3, 2, 3, 0, 2, 1, 3, 0, 1, 2, 0, 3, 2, 1, 2, 3, 1, 2, 1, 3, 0, 2, 0, 1, 2, 1, 1, 0, 3, 0, 3, 2, 0, 3.

Таблица  $S_7$ :

0, 3, 3, 0, 0, 3, 2, 1, 3, 0, 0, 3, 2, 1, 3, 2, 1, 2, 2, 1, 3, 1, 1, 2, 1, 0, 2, 3, 0, 2, 1, 0, 1, 0, 0, 3, 3, 3, 3, 2, 2, 1, 1, 0, 1, 2, 2, 1, 2, 3, 3, 1, 0, 0, 2, 3, 0, 2, 1, 0, 3, 1, 0, 2.

Таблица  $S_8$ :

3, 1, 0, 3, 2, 3, 0, 2, 0, 2, 3, 1, 3, 1, 1, 0, 2, 2, 3, 1, 1, 0, 2, 3, 1, 0, 0, 2, 2, 3, 1, 0, 1, 0, 3, 1, 0, 2, 1, 1, 3, 0, 2, 2, 2, 0, 3, 0, 3, 0, 2, 2, 3, 3, 0, 3, 1, 1, 1, 0, 2, 3.

Это означает, например, что входное значение 0 таблицы  $S_1$  заменяет на 2, значение 1 — на 0 и т. д. В алгоритме MacGuffin используются таблицы замен алгоритма DES. Однако, поскольку таблицы замен DES заменяют 6-битное входное значение 4-битным, здесь используются по два «внешних» бита выходных значений таблиц замен DES; «внутренние» биты не используются.

По окончании каждого раунда выполняется циклический сдвиг 16-битных фрагментов шифруемого блока влево (на один фрагмент).

Исходный текст алгоритма MacGuffin на языке Си приведен в спецификации алгоритма [96].

## Процедура расширения ключа

Расширение ключа (т. е. получение  $32 * 48$  битов ключевой информации из 128-битного ключа шифрования) выполняется с помощью самого алгоритма MacGuffin следующим образом [96]:

- Первые 64 бита ключа шифрования используются в качестве входного значения для 32 раундов преобразований алгоритма MacGuffin на нулевом

ключа. Каждый раунд  $i$  дает временное значение фрагментов расширенного ключа:

$$K_{i,0} = D;$$

$$K_{i,1} = A;$$

$$K_{i,2} = B.$$

2. Аналогичным образом обрабатывается вторая 64-битная половина ключа шифрования; в каждом раунде завершается формирование ключа раунда алгоритма:

$$K_{i,0} = K_{i,0} \oplus D;$$

$$K_{i,1} = K_{i,1} \oplus A;$$

$$K_{i,2} = K_{i,2} \oplus B.$$

## Криптоанализ алгоритма

Как было сказано выше, алгоритм MacGuffin был вскрыт в том же 1994 г.: специалисты из Бельгии Винсент Риджмен и Барт Пренел (Bart Preneel) доказали, что полнораундовый MacGuffin существенно менее стоек к дифференциальному криптоанализу, чем DES [324]. Они же проверили различные варианты выбора выходных битов таблиц замен DES (например, два младших бита вместо «внешних») в качестве таблиц замен MacGuffin и нашли более сильные варианты. Однако после их работы [324] алгоритм MacGuffin стал представлять лишь историческую ценность; какие-либо дальнейшие исследования криптостойкости данного алгоритма не получили известность.

### 3.33. Алгоритм MAGENTA

Алгоритм шифрования MAGENTA был разработан в 1998 г. специалистами немецкого телекоммуникационного гиганта Deutsche Telekom. Название алгоритма является аббревиатурой от Multifunctional Algorithm for General-purpose Encryption and Network Telecommunication — многофункциональный алгоритм для шифрования общего назначения и сетевых коммуникаций. Игра слов состоит в том, что ярко-красный (magenta) цвет является одним из корпоративных цветов компании Deutsche Telekom.

Основы данного алгоритма разрабатывались в корпорации Deutsche Telekom с 1990 г. [186], однако представлен был алгоритм MAGENTA только на конкурс AES (см. разд. 2.1) в 1998 г.

## Структура алгоритма

Аналогично другим алгоритмам — участникам конкурса, MAGENTA шифрует данные блоками по 128 битов с использованием ключей трех фиксированных размеров: 128, 192 и 256 битов.

Алгоритм представляет собой сеть Фейстеля, в которой выполняется 6 раундов преобразований при использовании 128- и 192-битных ключей или 8 раундов для 256-битных ключей шифрования [186].

При этом процедура расширения ключа, фактически, отсутствует: в раундах алгоритма используются в определенной последовательности 64-битные фрагменты ключа шифрования (см. табл. 3.78).

Таблица 3.78

Номер раунда	Размер ключа в битах		
	128	192	256
1	$K_1$	$K_1$	$K_1$
2	$K_1$	$K_2$	$K_2$
3	$K_2$	$K_3$	$K_3$
4	$K_2$	$K_3$	$K_4$
5	$K_1$	$K_2$	$K_4$
6	$K_1$	$K_1$	$K_3$
7	—	—	$K_2$
8	—	—	$K_1$

В табл. 3.78  $K_1 \dots K_4$  — 64-битные фрагменты ключа шифрования; например, 128-битный ключ состоит из двух фрагментов  $K_1$  и  $K_2$ , а 256-битный — из всех четырех:  $K_1 \dots K_4$  (рис. 3.122).

В каждом раунде шифрования блок данных делится на два субблока по 64 бита, один из которых обрабатывается функцией раунда  $F()$ ; результат обработки накладывается на необработанный субблок побитовой логической операцией «исключающее или» (XOR), после чего субблоки меняются местами (рис. 3.123).

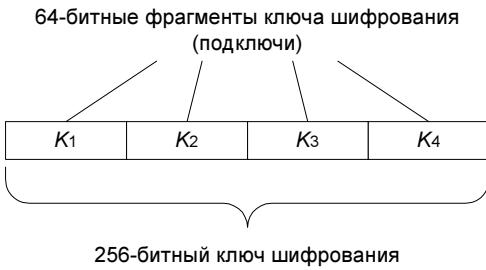


Рис. 3.122. Подключи 256-битного ключа

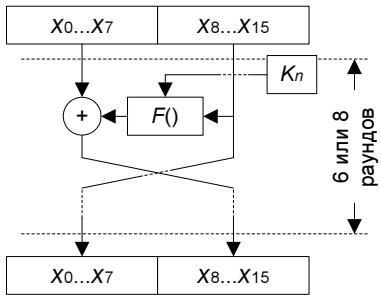


Рис. 3.123. Структура алгоритма MAGENTA

Основой алгоритма является функция  $f()$ , обрабатывающая 8-битные значения, которая определена следующим образом:

$$f(x) = 0 \text{ при } x = 255;$$

$$f(x) = \alpha^x \text{ при } x \neq 255,$$

где  $\alpha$  — образующий элемент поля  $GF(2^8)$  с образующим полиномом  $x^8 + x^6 + x^5 + x^2 + 1$ .

На основе данной функции определено преобразование  $P()$ :

$$P(x, y) = (f(x \oplus f(y)), f(y \oplus f(x))).$$

Входные значения функции  $P()$ :  $x$  и  $y$  — имеют размерность по 8 битов.

Еще одно преобразование —  $Q()$  — определено так:

$$Q(x_0, x_1, \dots, x_{15}) = (P(x_0, x_8), P(x_1, x_9), \dots, P(x_7, x_{15})),$$

где  $x_0, x_1, \dots, x_{15}$  — байты входного значения (128 битов).

На основе  $Q()$  определено также преобразование  $T()$ :

$$T(x_0, x_1, \dots, x_{15}) = Q(Q(Q(Q(x_0, x_1, \dots, x_{15})))).$$

И, наконец, преобразование  $C()$  определено так:

$$C^{(j+1)}(x_0, x_1, \dots, x_{15}) = T((x_0, x_1, \dots, x_7) \oplus Ce^{(j)}, (x_8, x_9, \dots, x_{15}) \oplus Co^{(j)}),$$

где  $Ce^{(j)}$  — четные байты  $C^{(j)}$ , а  $Co^{(j)}$  — нечетные байты  $C^{(j)}$ .

Начальное значение  $C^{(1)}(x_0, x_1, \dots, x_{15}) = T(x_0, x_1, \dots, x_{15})$ .

Функция раунда  $F()$  (см. рис. 3.123) представляет собой  $Ce^{(3)}(x_8, x_9, \dots, x_{15}, k_0, k_1, \dots, k_7)$ , где  $(k_0, k_1, \dots, k_7)$  — 64-битный ключ раунда (см. выше), а  $(x_8, x_9, \dots, x_{15})$  — 64-битный субблок шифруемых данных.

Расшифровывание выполняется аналогично зашифровыванию, но после выполнения описанных выше операций 64-битные субблоки меняются местами.

## Достоинства и недостатки алгоритма

В 1999 г. несколько известных криптологов в совместном докладе [62] на одной из конференций, посвященных конкурсу AES, описали несколько атак на алгоритм MAGENTA, в частности, вариант со 128-битным ключом подвержен следующим атакам:

- алгоритм вскрывается при наличии  $2^{64}$  выбранных открытых текстов и соответствующих им шифртекстов выполнением  $2^{64}$  тестовых операций шифрования;
- алгоритм вскрывается при наличии  $2^{33}$  известных открытых текстов и соответствующих им шифртекстов выполнением  $2^{97}$  операций шифрования.

С учетом того факта, что алгоритм MAGENTA не имеет явных преимуществ, а также медленно выполняется на всех платформах, на которых производилось тестирование производительности в рамках конкурса AES [284], эксперты не выбрали этот алгоритм в финал конкурса AES.

## 3.34. Алгоритм MARS

Алгоритм MARS был разработан коллективом криптологов из корпорации IBM. Именно IBM в свое время разработала семейство алгоритмов Lucifer, которое легло в основу предыдущего стандарта шифрования США — DES. Из авторов Lucifer в разработке алгоритма MARS принял участие Дон Конперсмит, известный также и другими работами в области криптологии.

По правилам конкурса AES разрешалось вносить незначительные изменения в участвовавшие в конкурсе алгоритмы в течение первого раунда конкурса. Пользуясь этим правилом, авторы алгоритма MARS изменили процедуру расширения ключа, в результате чего существенно снизились требования,

предъявляемые алгоритмом к энергонезависимой и оперативной памяти [284]. Рассмотрим здесь именно модифицированную версию алгоритма.

## Структура алгоритма

При разработке алгоритма его авторы исходили из двух следующих предложений [111]:

- Многие известные криptoаналитические методы отличают первый и последний раунды алгоритма (или несколько первых и/или несколько последних раундов) от остальных и применяют к ним другие приемы, не жели к «центральным» раундам алгоритма. Таким образом, различные раунды алгоритма шифрования играют различное значение в обеспечиваемой алгоритмом криптостойкости.
- Скорее всего, алгоритм с гетерогенной структурой будет лучше противостоять криptoаналитическим методам будущего, чем алгоритм, все раунды которого идентичны.

В результате разработчики алгоритма MARS придали ему сильно гетерогенную структуру — раунды алгоритма весьма различаются между собой. Алгоритм MARS можно описать следующим образом (рис. 3.124):

1. Выполняется предварительное наложение ключа: на 32-битные субблоки  $A, B, C, D$  накладываются 4 фрагмента расширенного ключа  $k_0 \dots k_3$  операцией сложения по модулю  $2^{32}$ .
2. Выполняются 8 раундов прямого перемешивания (без участия ключа шифрования).
3. Выполняются 8 раундов прямого криптопреобразования.
4. Выполняются 8 раундов обратного криптопреобразования. Этапы 3 и 4 называются «криптографическим ядром» алгоритма MARS.
5. Выполняются 8 раундов обратного перемешивания, также без участия ключа шифрования.
6. Выполняется финальное наложение фрагментов расширенного ключа  $k_{36} \dots k_{39}$  операцией вычитания по модулю  $2^{32}$ .
7. Алгоритм представляет собой расширенную сеть Фейстеля. В каждом раунде выполняется обработка одного из субблоков и наложение результатов обработки на остальные субблоки, после чего субблоки меняются местами. Конкретные преобразования зависят от типа раунда и будут рассмотрены далее. Кроме того, между раундами могут выполняться различные дополнительные действия, которые также будут описаны далее.

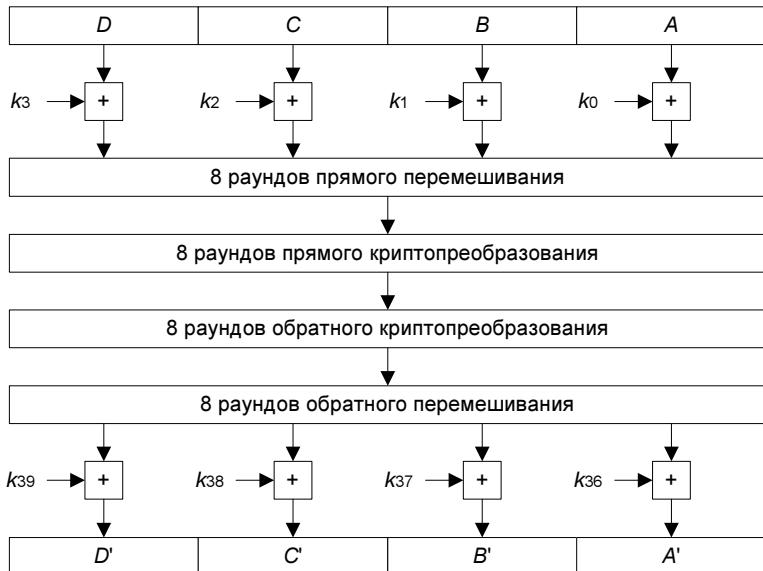


Рис. 3.124. Структура алгоритма MARS

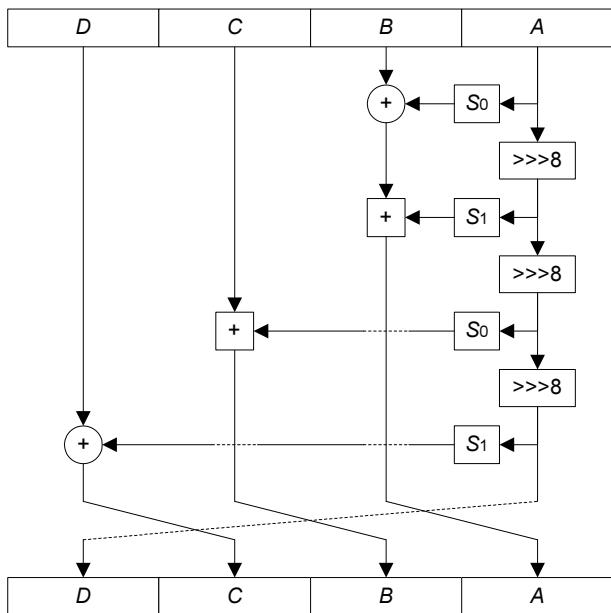


Рис. 3.125. Раунд прямого перемешивания алгоритма MARS

Раунд прямого перемешивания показан на рис. 3.125. Как видно из рисунка, в раунде выполняются следующие действия:

1. Значение субблока  $A$  «прогоняется» через таблицу замен  $S_0$  и накладывается на субблок  $B$  операцией XOR.
2. Исходное значение субблока  $A$  вращается на 8 битов вправо.
3. Результат предыдущего шага обрабатывается таблицей замен  $S_1$  и снова накладывается на субблок  $B$  операцией сложения по модулю  $2^{32}$ .
4. Результат шага 2 вращается на 8 битов вправо.
5. Результат предыдущего шага обрабатывается таблицей замен  $S_0$  и накладывается на субблок  $C$  операцией сложения по модулю  $2^{32}$ .
6. Результат шага 4 вращается на 8 битов вправо.
7. Результат предыдущего шага обрабатывается таблицей замен  $S_1$  и накладывается на субблок  $D$  операцией XOR.
8. Субблоки меняются местами, как показано на рис. 3.125.

Кроме того, в некоторых раундах прямого перемешивания выполняются следующие дополнительные операции (не приведены на рис. 3.125):

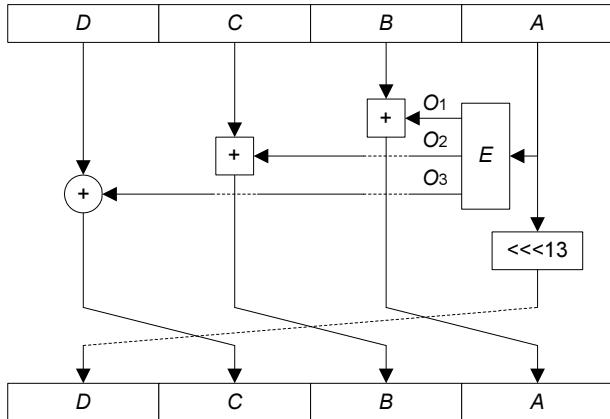
- в раундах №№ 0 и 4 после шага 7 выполняется наложение значения субблока  $D$  на субблок  $A$  операцией сложения по модулю  $2^{32}$ ;
- в раундах №№ 1 и 5 субблок  $B$  аналогичным образом накладывается на субблок  $A$ .

По словам авторов алгоритма, эти операции существенно усиливают алгоритм MARS против дифференциального криптоанализа [111].

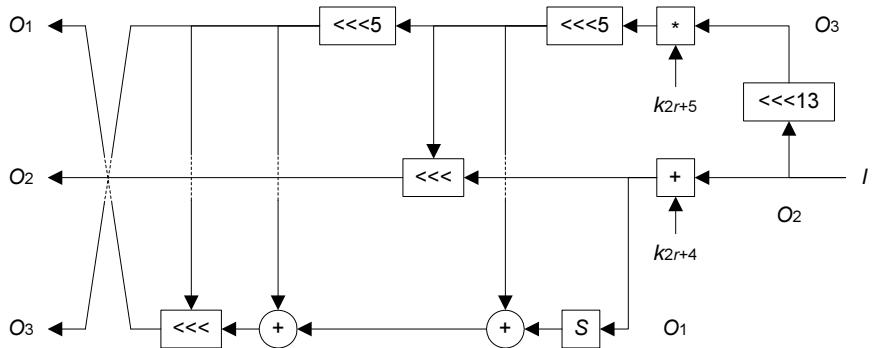
Таблицы замен  $S_0$  и  $S_1$  приведены в *Приложении*. В качестве входного значения  $S_0$  (аналогично и  $S_1$ ) принимает 8 младших битов входного слова. Таблица  $S_0$  меняет значение 0 на 09D0C479, значение 1 — на 28C8FFE0 и т. д.

Структура раунда прямого криптопреобразования приведена на рис. 3.126. Основой раунда является расширяющее криптопреобразование  $E$ , преобразующее 32-битное входное слово  $A$  в три выходных 32-битных значения, каждое из которых определенным образом накладывается на остальные субблоки (см. рис. 3.126). После этого субблок  $A$  вращается влево на 13 битов, затем субблоки меняются местами аналогично раунду прямого перемешивания.

## Описание алгоритмов



**Рис. 3.126.** Раунд прямого криптопреобразования алгоритма MARS



**Рис. 3.127.** Операция  $E$  алгоритма MARS

Преобразование  $E$  показано на рис. 3.127. Из входного значения формируются три потока  $O_1 \dots O_3$ , над которыми производятся следующие действия:

$$O_2 = I;$$

$$O_3 = O_2 <<< 13;$$

$$O_2 = O_2 + k_{2r+4} \bmod 2^{32};$$

$$O_3 = O_3 * k_{2r+5} \bmod 2^{32};$$

$$O_3 = O_3 <<< 5;$$

$$O_1 = S(O_2);$$

$$O_1 = O_1 \oplus O_3;$$

$$O_2 = O_2 <<< O_3';$$

$$O_3 = O_3 \lll 5;$$

$$O_1 = O_1 \oplus O_3;$$

$$O_1 = O_1 \lll O_3',$$

где:

- $I$  — входное значение;
- $r$  — номер текущего раунда, считая с 0-го (при нумерации раундов в данном случае учитываются только раунды криптоядра алгоритма);
- $S$  — табличная замена для операции  $E$ , представляет собой объединение описанных выше таблиц  $S_0$  и  $S_1$ ; объединенная таблица содержит 512 значений, выходное значение выбирается по значению 9 младших битов входного слова.

Стоит обратить внимание на то, что в преобразовании  $E$  используются операции вращения на переменное число битов; в этом случае запись  $O_3'$  обозначает, что число битов вращения определяется значением младших пяти битов текущего значения  $O_3$ .

Структура обратного криптораунда показана на рис. 3.128. От прямого криптораунда его отличает лишь измененный порядок наложения выходных значений преобразования  $E$   $O_1...O_3$  на слова  $B$ ,  $C$  и  $D$ .

Раунд обратного перемешивания (рис. 3.129) более существенно отличается от прямого. Фактически обратное перемешивание выполняет обратные операции в обратной последовательности (см. рис. 3.125 и 3.129):

1. Значение субблока  $A$  «прогоняется» через таблицу замен  $S_1$  и накладывается на субблок  $B$  операцией XOR.

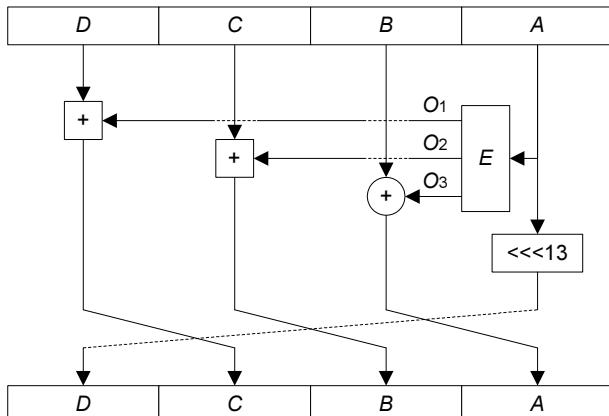
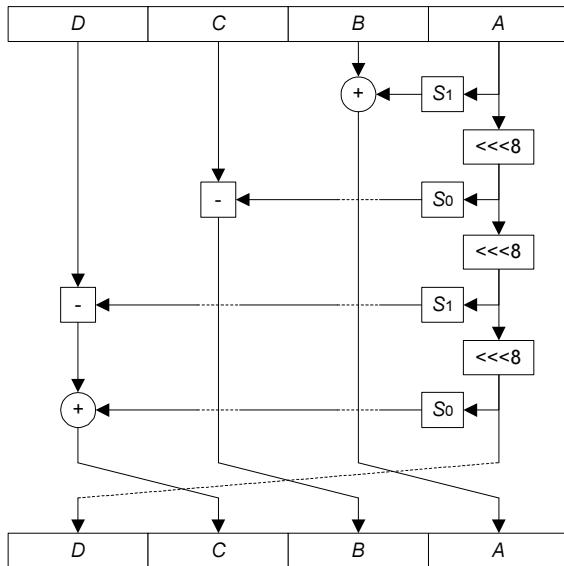


Рис. 3.128. Раунд обратного криптопреобразования алгоритма MARS



**Рис. 3.129.** Раунд обратного перемешивания алгоритма MARS

- Исходное значение субблока  $A$  вращается на 8 битов влево.
  - Результат предыдущего шага обрабатывается таблицей замен  $S_0$  и накладывается на субблок  $C$  операцией вычитания по модулю  $2^{32}$ .
  - Результат шага 2 вращается на 8 битов влево.
  - Результат предыдущего шага обрабатывается таблицей замен  $S_1$  и накладывается на субблок  $D$  операцией вычитания по модулю  $2^{32}$ .
  - Результат шага 4 вращается на 8 битов влево.
  - Результат предыдущего шага обрабатывается таблицей замен  $S_0$  и накладывается на субблок  $D$  операцией XOR.
  - Субблоки меняются местами, как показано на рис. 3.129.

Аналогично прямому перемешиванию, в некоторых раундах выполняются следующие дополнительные операции, не показанные на рис. 3.129:

- в раундах №№ 1 и 5 после шага 7 выполняется наложение значения субблока  $A$  на субблок  $B$  операцией вычитания по модулю  $2^{32}$ ;
  - в раундах №№ 2 и 6 субблок  $C$  аналогичным образом накладывается на субблок  $B$ .

Расшифровывание выполняется применением обратных операций в обратной последовательности; подробно расшифровывание описано в спецификации алгоритма [111].

## Процедура расширения ключа

Ключ шифрования алгоритма MARS может иметь любой размер в диапазоне от 128 до 448 битов включительно, кратный 32 битам. Расширение ключа представляет собой формирование на основе ключа шифрования 40 подключей по 32 бита каждый (причем подключи должны обладать определенными характеристиками, о которых будет сказано далее), для чего выполняются следующие операции:

1. Формируется временный массив  $T_0 \dots T_{14}$ :

$$\begin{aligned} T_0 &= KI_0; \\ T_1 &= KI_1; \\ &\dots \\ T_{n-1} &= KI_{n-1}; \\ T_n &= n; \\ T_{n+1} &= T_{n+2} = \dots = T_{14} = 0, \end{aligned}$$

где:

- $KI_0 \dots KI_{n-1}$  — исходный ключ шифрования;
- $n$  — его размер в 32-битных словах — от 4 до 14 включительно.

2. Данный шаг повторяется 4 раза (каждая итерация дает 10 вычисленных фрагментов расширенного ключа) и содержит следующие операции:

- линейное преобразование:

$$T_i = T_i \oplus ((T_{i-7 \bmod 15} \oplus T_{i-2 \bmod 15}) \lll 3) \oplus (4i + j),$$

где:

◊  $j$  — номер итерации, начиная с 0;

◊  $i = 0 \dots 14$ ;

- перемешивание массива  $T$ :

$$T_i = (T_i \oplus S(T_{i-1 \bmod 15})) \lll 9,$$

где  $S$  — табличная замена, выполняемая по тем же правилам, что и в операции  $E$ ;

- заключительная перестановка:

$$k_{10j+n} = T_{4n \bmod 15},$$

где  $n = 0 \dots 9$ .

3. На вычисленные подключи накладываются дополнительные требования, состоящие в следующем:

- каждый подключ, используемый для умножения в операции  $E$  (т. е. подключи с нечетными индексами от  $k_5$  до  $k_{35}$  включительно), должен

иметь нечетное значение; мало того, единичными должны быть два младших бита подключа, а не один;

- те же подключи не должны содержать 10 нулевых или 10 единичных битов подряд.

Модификация подключей в соответствии с данными требованиями выполняется следующим образом.

- 2 младших бита обрабатываемого подключа устанавливаются в единичные значения; старое значение запоминается в переменной  $j$  (оно будет использовано впоследствии). Результирующее значение подключа обозначается как  $W$ .
- Вычисляется 32-битная маска  $M$ , которая будет использована для модификации подключа — для обеспечения отсутствия в нем 10 подряд нулевых или единичных битов. Маска вычисляется следующим образом:
  - ◊ устанавливаются в 1 биты  $M$ , соответствующие тем битам обрабатываемого подключа, которые входят в последовательности из 10 нулевых или 10 единичных битов; остальные биты устанавливаются в 0;
  - ◊ обнуляются те единичные биты маски  $M$ , которые соответствуют любому из условий:

$$i < 2;$$

$$i = 31;$$

$$W_i \neq W_{i-1};$$

$$W_i \neq W_{i+1},$$

где  $i$  — номер бита, начиная с 0, а  $W_i$  — значение  $i$ -го бита.

- Используется таблица корректирующих значений  $B$ , определенная следующим образом:

$$B_0 = \text{a4a8d57b};$$

$$B_1 = \text{5b5d193b};$$

$$B_2 = \text{c8a8309b};$$

$$B_3 = \text{73f9a978}.$$

Элементы таблицы  $B$  эквивалентны элементам с 265-го по 268-й включительно объединенной таблицы  $S$ ; именно эти элементы используются для коррекции подключей благодаря их специфическим свойствам.

С помощью данной таблицы следующим образом вычисляется финальное значение подключа  $K_i$ :

$$K_i = W \oplus ((B_j << K_{i-1}) \& M),$$

где  $\&$  — логическая побитовая операция «и», а вращение  $B_j$  определяется пятью младшими битами предыдущего подключа  $K_{i-1}$ .

Описанная процедура гарантирует требуемые свойства у подключей [111].

## Достоинства и недостатки алгоритма

Сравнительный анализ достоинств и недостатков алгоритмов — финалистов конкурса AES приведен в разд. 2.1.

### 3.35. Алгоритм Mercy

Алгоритм Mercy разработан в 2000 г. Полом Кроули (Paul Crowley) из английской компании DataCash [125].

#### Структура алгоритма

Алгоритм Mercy шифрует данные блоками переменного размера, но рекомендуемый размер блока составляет 4096 битов (т. е. соответствует размеру сектора диска). Помимо 128-битного ключа шифрования, существует еще один параметр алгоритма — 128-битная величина, которая может быть случайной, а может и обозначать, например, номер шифруемого сектора. Главное — эта величина должна быть известна при расшифровании данных, но не обязательно должна быть секретной (далее будет обозначена как  $Sp$ ).

Несмотря на то, что простота дизайна алгоритма, по словам его автора, была одним из основных факторов при разработке алгоритма, структура алгоритма (по сравнению с многими другими алгоритмами блочного симметричного шифрования) оказалась достаточно сложной. Рассмотрим структуру «снизу вверх», т. е. от простейших операций — к комплексным.

Основным преобразованием алгоритма Mercy является функция  $T()$ , выполняющая преобразование 8-битных слов  $x$  в 32-битные по следующему закону (рис. 3.130):

$$T(x) = \sum_{i=0}^3 2^{8i} d_{2i+1}[N(d_{2i}[x])],$$

где:

$d_0 \dots d_7$  — зависящие от ключа замены, будут подробно описаны далее;

## Описание алгоритмов

- $N(x)$  — мультипликативная обратная величина от  $x$  в поле  $GF(2^8)$  с образующим полиномом  $x^8 + x^4 + x^3 + x + 1$ .

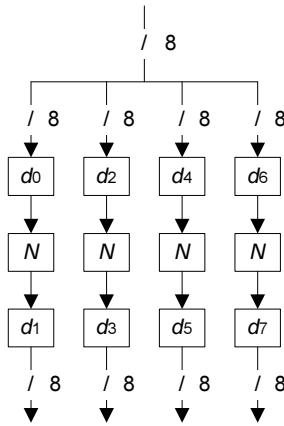


Рис. 3.130. Операция  $T$   
алгоритма Mercy

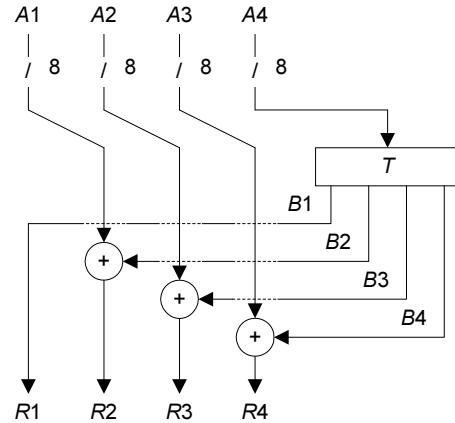


Рис. 3.131. Операция  $M$   
алгоритма Mercy

Уровнем выше находится функция  $M()$ , обрабатывающая 32-битные слова с использованием операции  $T()$  (рис. 3.131). Данная функция выполняется следующим образом:

1. Обрабатываемое слово  $A$  делится на 8-битные фрагменты  $A1...A4$ , правый из которых ( $A4$ ) «прогоняется» через операцию  $T()$ .
2. Левый байт результата (обозначим результат как  $B1...B4$ ) операции  $T(A4)$  становится левым байтом выходного значения ( $R1...R4$ ), т. е.:

$$R1 = B1.$$

3. Остальные байты результата операции  $T(A4)$  накладываются на входные байты побитовой логической операцией «исключающее или» (XOR) следующим образом:

$$R2 = A1 \oplus B2;$$

$$R3 = A2 \oplus B3;$$

$$R4 = A3 \oplus B4.$$

Функция  $M()$  используется операцией  $Q()$ , которая преобразует одно входное 32-битное слово в одно выходное с использованием 4 слов состояния  $S1...S4$  (рис. 3.132). Начальное значение состояния будет пояснено далее; здесь рассмотрим действия, выполняемые операцией  $Q()$ :

1. Входное значение  $I$  складывается с  $S1$  по модулю  $2^{32}$ .

2. Результат предыдущего шага обрабатывается операцией  $M()$ .
3. Результат предыдущего шага складывается с  $S_3$  по модулю  $2^{32}$  — так формируется выходное значение (обозначим его  $O$ ).
4. Новое значение слова состояния  $S_1$  формируется сложением выходного значения с  $S_4$  при помощи операции XOR.
5. Новые значения остальных слов состояния  $S_2$ ,  $S_3$  и  $S_4$  равны старым значениям слов  $S_1$ ,  $S_2$  и  $S_3$  соответственно.

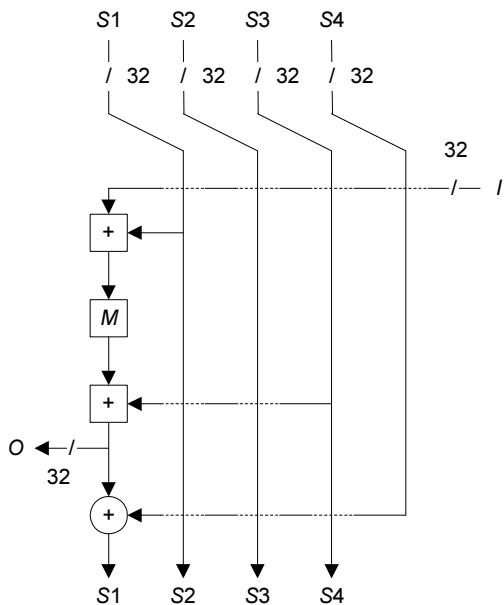


Рис. 3.132. Операция  $Q$  алгоритма Mercy

Все действия, выполняемые операцией  $Q()$ , можно представить в виде следующих формул:

$$O = M(I + S_1) + S_3;$$

$$T = S_4;$$

$$S_4 = S_3;$$

$$S_3 = S_2;$$

$$S_2 = S_1;$$

$$S_1 = O \oplus T,$$

где  $T$  — временная переменная, а знаком  $+$  обозначено сложение по модулю  $2^{32}$ .

Еще один уровень выше — функция  $F()$ , активно использующая операцию  $Q()$  следующим образом (рис. 3.133):

- Последние 4 слова входного значения  $Q_{n-3}...Q_n$  (обозначим входное значение операции  $F()$  как  $Q_1...Q_n$ , где  $n$  — размер половины блока шифруемых данных в 32-битных словах) становятся исходным значением состояния операции  $Q()$ , т. е.  $S1...S4$ .
- Выполняются 4 операции  $Q()$ , в которых в качестве входного значения используются слова 128-битного параметра  $Sp$  ( $Sp_1...Sp_4$ ). Выходные значения этих операций не используются, однако видно, что эти операции изменяют значения слов состояния  $S1...S4$ .

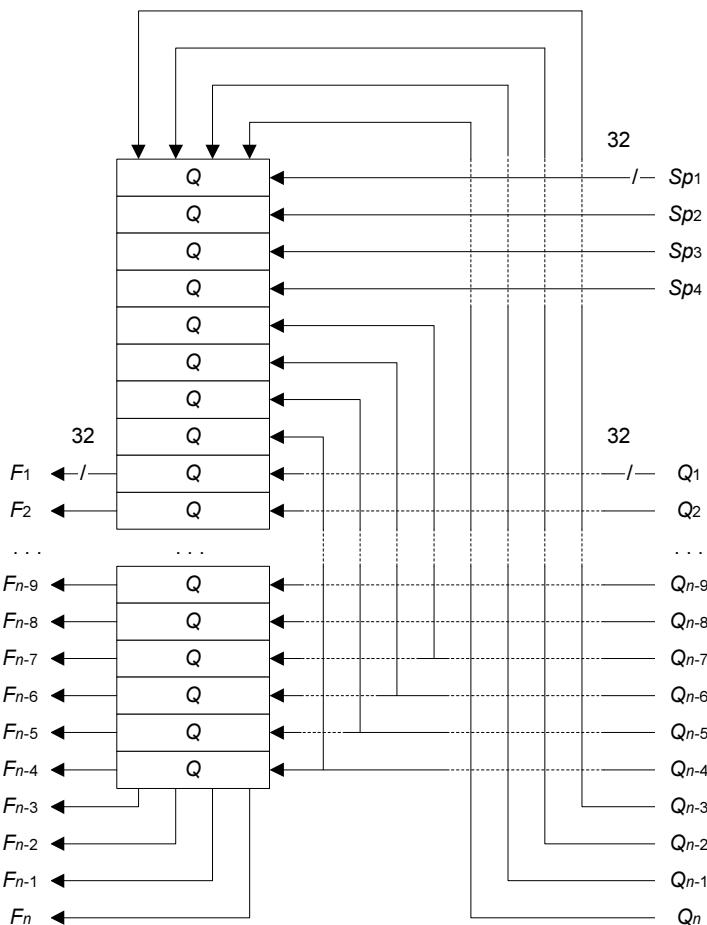


Рис. 3.133. Операция  $F$  алгоритма Mercy

3. Выполняются еще 4 операции  $Q()$ , входными значениями которых являются 4 слова входного блока  $Q_{n-7}...Q_{n-4}$ . Выходные значения этих операций также не используются.
4. Наконец, выполняются  $n - 4$  операций  $Q()$ , в качестве входных значений которых используются слова  $Q_1...Q_{n-4}$ . Выходные слова операций  $Q()$  формируют слова выходного значения функции  $F()$ , обозначим их как  $F_1...F_{n-4}$ . Последние 4 выходных слова  $F_{n-3}...F_n$  равны словам состояния  $S1...S4$  после выполнения всех перечисленных операций  $Q()$ .

Стоит отметить, что 4-словный фрагмент  $Q_{n-7}...Q_{n-4}$  в качестве входного значения операций  $Q()$  используется дважды.

Теперь рассмотрим полностью процедуру шифрования в алгоритме Mercy. По своей структуре алгоритм представляет собой сеть Фейстеля с частичным предварительным и заключительным отбеливанием, т. е. наложением на обрабатываемые субблоки операций XOR псевдослучайной последовательности, полученной из ключа шифрования в процессе выполнения процедуры расширения ключа.

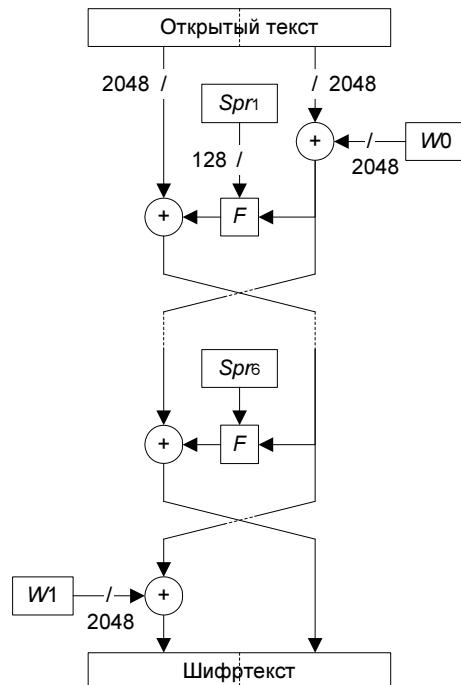


Рис. 3.134. Структура алгоритма Mercy

Структура алгоритма Mercy приведена на рис. 3.134. Как видно из рисунка, алгоритм выполняет 6 раундов преобразований, в которых в качестве функции раунда используется описанная выше функция  $F()$ . Обозначенные на рис. 3.134 128-битные элементы  $Spr_1 \dots Spr_6$  представляют собой результат расширения параметра  $Sp$ ; эта процедура будет описана далее.

Перед первым раундом на правый субблок открытого текста накладывается зависящее от ключа шифрования 2048-битное значение  $W0$  (предварительное отбеливание), а после шестого раунда таким же образом на левый субблок накладывается также зависящее от ключа значение  $W1$  (пост-отбеливание).

Расшифровывание данных выполняется аналогично их зашифровыванию, за исключением того, что пре- и пост-отбеливание меняются местами, т. е. наложение  $W1$  выполняется до первого раунда, а  $W0$  используется по завершении шести раундов.

## Процедура расширения ключа

Процедура расширения ключа алгоритма Mercy подразумевает использование любого генератора псевдослучайных чисел, в качестве которого автором алгоритма рекомендуется использовать потоковый шифр RC4, разработанный известнейшим криптологом Рональдом Ривестом. Генератор псевдослучайных чисел инициализируется ключом шифрования, а первые 256 байтов псевдослучайных чисел отбрасываются. Далее расширение ключа выполняется таким образом:

1. В цикле по  $i$  от 0 до 7 выполняются следующие действия:

- текущее случайное число становится значением нулевого байта  $d_i$ ;
- в цикле по  $j$  от 0 до 7:
  - ◊ выбирается случайное число  $r$ ; если  $r$  уже записано в текущей таблице (от байта 0 до байта  $2^j - 1$  таблицы  $d_i$ ), запрос случайного числа повторяется;
  - ◊ в цикле по  $k$  от 0 до  $2^j - 1$  выполняется следующее действие:

$$d_i[k + 2^j] = d_i[k] \oplus r \oplus d_i[0].$$

2. Вычисляется  $W0$  (после чего таким же образом вычисляется  $W1$ ), для чего в цикле по  $j$  от 0 до 64 выполняется следующее:

- $j$ -й байт  $W0$  обнуляется;
- в цикле по  $k$  от 0 до 3 к текущему значению  $j$ -го байта  $W0$  прибавляется новое случайное число, умноженное на значение  $2^{8k}$ .

Как было сказано выше, перед зашифровыванием также выполняется расширение 128-битного значения  $Sp$ , в результате которого вычисляются 6 128-битных величин  $Spr_1 \dots Spr_6$ . Для этого используется функция  $F()$ , дающая 24 выходных 32-битных значения, которые последовательно используются в качестве соответствующих фрагментов  $Spr_1 \dots Spr_6$ . При этом исходные значения  $Sp_1 \dots Sp_4$  используются именно так, как показано на рис. 3.133. В качестве же входного значения для функции  $F()$  используется последовательность 32-битных значений  $H_0 \dots H_{23}$ , вычисляемая достаточно просто:

$$H_i = \sum_{j=0}^3 2^{8j} (4i + j).$$

## Криптостойкость алгоритма

Относительно широкую известность получила лишь одна работа, посвященная криптоанализу алгоритма Mercy, автор которой — Скотт Фларер (Scott R. Fluhrer) из компании Cisco Systems [155]. Однако какие-либо практические осуществимые атаки на алгоритм Mercy в этой работе не были предложены.

Несмотря на то, что в алгоритме не были найдены значительные уязвимости, весьма маловероятно, что Mercy найдет достаточно широкое применение из-за весьма узкого назначения.

## 3.36. Алгоритмы MISTY1 и MISTY2

Алгоритм MISTY1 разработан в 1995–1996 гг. командой специалистов под руководством известного криптолога Мицуру Мацуи из компании Mitsubishi Electric (Япония). В разработке алгоритма приняли участие Тецуя Ичикава (Tetsuya Ichikawa), Джун Соримачи (Jun Sorimachi), Тошио Токита (Toshio Tokita) и Ацуhiro Ямагиши (Atsuhiro Yamagishi) [395]. Известны также две модификации алгоритма MISTY1: MISTY2 и KASUMI, которые будут кратко описаны далее.

Начнем с подробного описания алгоритма MISTY1.

### Структура алгоритма MISTY1

Алгоритм MISTY1 имеет весьма необычную структуру — он основан на «вложенных» сетях Фейстеля. Сначала 64-битный шифруемый блок данных разбивается на два 32-битных субблока, после чего выполняется  $r$  раундов следующих преобразований (рис. 3.135) [258, 291]:

1. Каждый субблок обрабатывается операцией  $FL$  (операции описаны далее). Этот шаг выполняется только в нечетных раундах.
2. Над обрабатываемым субблоком выполняется операция  $FO$ .

## Описание алгоритмов

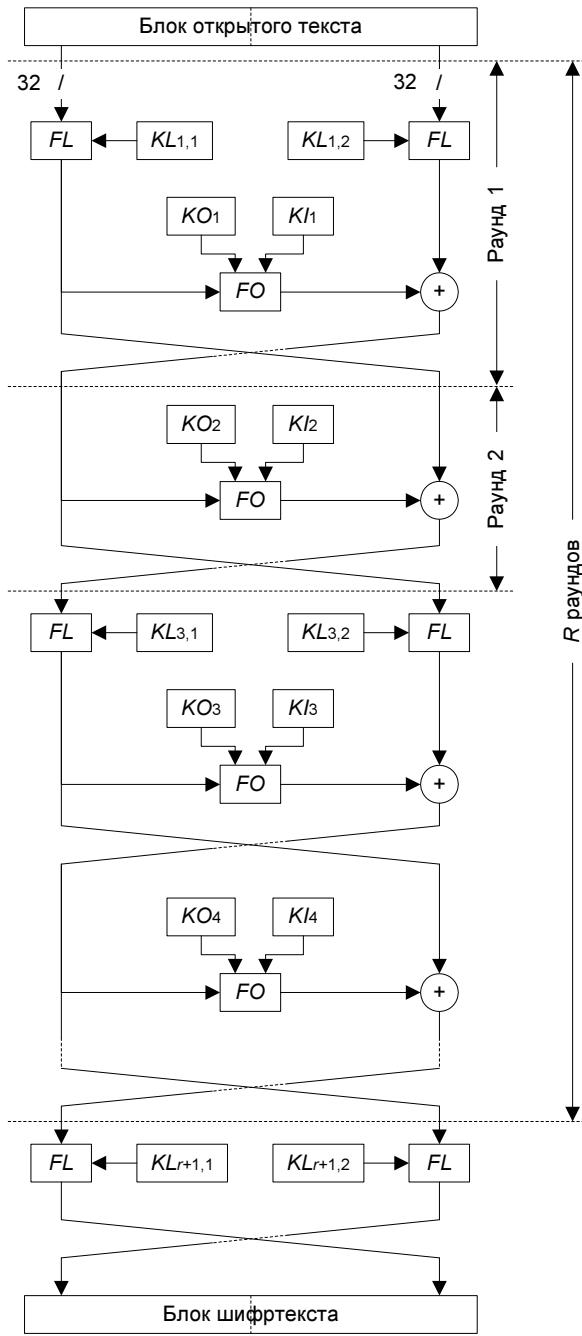


Рис. 3.135. Структура алгоритма MISTY1

3. Результат этих операций накладывается побитовой логической операцией «исключающее или» (XOR) на необработанный субблок.
4. Субблоки меняются местами.

После заключительного раунда оба субблока еще раз обрабатываются операцией  $FL$ .

Рекомендуемым количеством раундов алгоритма является 8, но количество раундов алгоритма может быть также любым, превышающим 8 и кратным четырем.

Операция  $FL$  является достаточно простой. Обрабатываемый ей субблок разбивается на два 16-битных фрагмента, над которыми выполняются следующие действия (рис. 3.136):

$$\begin{aligned} R' &= R \oplus (L \& KL_{i,j,1}); \\ L' &= L \oplus (R' | KL_{i,j,2}), \end{aligned}$$

где:

- $L$  и  $R$  — входные значения левого и правого фрагментов соответственно;
- $L'$  и  $R'$  — выходные значения;
- $KL_{i,j,1}$  и  $KL_{i,j,2}$  — фрагменты  $j$ -го подключа  $i$ -го раунда для операции  $FL$  (процедура расширения ключа подробно описана далее);
- $\&$  и  $|$  — побитовые логические операции «и» и «или» соответственно.

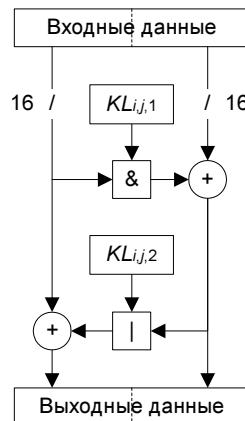


Рис. 3.136. Операция  $FL$  алгоритма MISTY1

Операция  $FO$  более интересна — именно она является вложенной сетью Фейстеля (рис. 3.137). Как и ранее, входное значение разбивается на два

16-битных фрагмента, после чего выполняются 3 раунда следующих действий:

1. На левый фрагмент операцией XOR накладывается фрагмент ключа раунда  $KO_{i,k}$  ( $k$  — номер раунда операции  $FO$ ).
2. Левый фрагмент обрабатывается операцией  $FI$ .
3. На левый фрагмент накладывается операцией XOR значение правого фрагмента.
4. Фрагменты меняются местами.

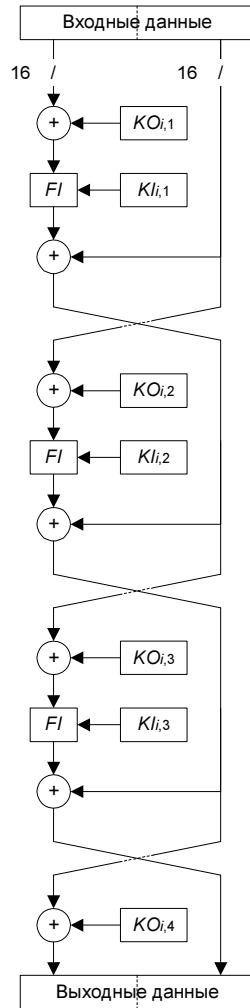
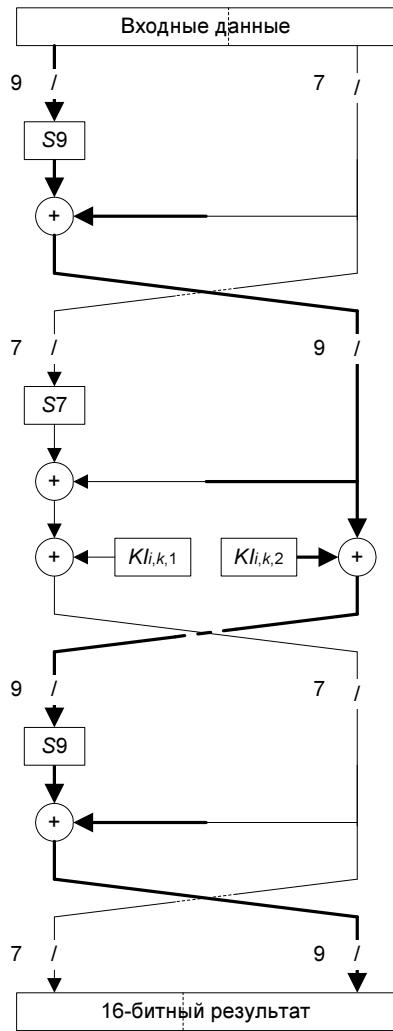


Рис. 3.137. Операция  $FO$  алгоритма MISTY1



**Рис. 3.138.** Операция *FI*

После третьего раунда операции *FO* на левый фрагмент накладывается операцией XOR дополнительный фрагмент ключа  $KO_{i,4}$ .

Операция *FI* также представляет собой сеть Фейстеля, т. е. это уже третий уровень вложенности. В отличие от сетей Фейстеля на двух верхних уровнях, данная сеть является несбалансированной: обрабатываемый 16-битный фрагмент делится на две части: 9-битную левую и 7-битную правую.

## Описание алгоритмов

Затем выполняются 3 раунда, которые состоят из следующих действий (рис. 3.138):

1. Левая часть «прогоняется» через таблицу замен. 9-битная часть (в раундах №№ 1 и 3) обрабатывается таблицей  $S_9$ , а 7-битная (в раунде № 2) — таблицей  $S_7$ . Эти таблицы приведены далее.
2. На левую часть операцией XOR накладывается текущее значение правой части. При этом, если справа 7-битная часть, она дополняется нулями слева, а у 9-битной части удаляются слева два бита.
3. Во втором раунде на левую часть операцией XOR накладывается фрагмент ключа раунда  $KI_{i,k,1}$ , а на правую — фрагмент  $KI_{i,k,2}$ . В остальных раундах эти действия не выполняются.
4. Левая и правая части меняются местами.

Для наглядности на рис. 3.138 жирными линиями выделен 9-битный поток данных.

Таблицы  $S_7$  и  $S_9$  алгоритма MISTY1 могут быть реализованы как с помощью вычислений, так и непосредственно таблицами, хранимыми в энергонезависимой памяти шифрующего устройства [258, 291]. При реализации алгоритма должен выбираться вариант использования таблиц в зависимости от ресурсов шифрующего устройства.

Таблица  $S_7$  приведена в табл. 3.79 (указаны шестнадцатеричные значения).

**Таблица 3.79**

1b	32	33	5a	3b	10	17	54	5b	1a	72	73	6b	2c	66	49
1f	24	13	6c	37	2e	3f	4a	5d	0f	40	56	25	51	1c	04
0b	46	20	0d	7b	35	44	42	2b	1e	41	14	4b	79	15	6f
0e	55	09	36	74	0c	67	53	28	0a	7e	38	02	07	60	29
19	12	65	2f	30	39	08	68	5f	78	2a	4c	64	45	75	3d
59	48	03	57	7c	4f	62	3c	1d	21	5e	27	6a	70	4d	3a
01	6d	6e	63	18	77	23	05	26	76	00	31	2d	7a	7f	61
50	22	11	06	47	16	52	4e	71	3e	69	43	34	5c	58	7d

Таблица  $S_7$  может быть также реализована следующим образом:

$$\begin{aligned}y_7 = & x_7 \oplus x_6x_4 \oplus x_7x_4x_3 \oplus x_6x_2 \oplus x_7x_5x_2 \oplus \\& \oplus x_3x_2 \oplus x_7x_6x_1 \oplus x_5x_1 \oplus x_7x_2x_1 \oplus x_4x_2x_1 \oplus 1;\end{aligned}$$

$$\begin{aligned}
 y_6 &= x_7x_5 \oplus x_7x_3 \oplus x_4x_3 \oplus x_6x_2 \oplus x_5x_3x_2 \oplus \\
 &\oplus x_1 \oplus x_7x_1 \oplus x_4x_1 \oplus x_5x_4x_1 \oplus x_6x_3x_1 \oplus x_7x_2x_1 \oplus 1; \\
 y_5 &= x_6x_5 \oplus x_7x_5x_4 \oplus x_3 \oplus x_6x_3 \oplus x_7x_6x_3 \oplus \\
 &\oplus x_7x_2 \oplus x_7x_3x_2 \oplus x_4x_3x_2 \oplus x_6x_1 \oplus x_4x_1 \oplus x_7x_4x_1 \oplus x_3x_1 \oplus x_5x_3x_1; \\
 y_4 &= x_7 \oplus x_6 \oplus x_7x_6x_5 \oplus x_7x_4 \oplus x_5x_3 \oplus x_6x_3x_2 \oplus \\
 &\oplus x_5x_1 \oplus x_6x_4x_1 \oplus x_7x_3x_1 \oplus x_2x_1 \oplus 1; \\
 y_3 &= x_5x_4 \oplus x_7x_3 \oplus x_6x_4x_3 \oplus x_2 \oplus x_5x_2 \oplus x_6x_5x_2 \oplus \\
 &\oplus x_7x_4x_2 \oplus x_6x_1 \oplus x_6x_2x_1 \oplus x_3x_2x_1 \oplus 1; \\
 y_2 &= x_7 \oplus x_6 \oplus x_5 \oplus x_7x_6x_5 \oplus x_7x_4 \oplus x_6x_5x_4 \oplus x_6x_3 \oplus \\
 &\oplus x_7x_5x_3 \oplus x_7x_2 \oplus x_7x_6x_2 \oplus x_4x_2 \oplus x_7x_1 \oplus x_5x_2x_1; \\
 y_1 &= x_7x_6 \oplus x_4 \oplus x_7x_4 \oplus x_5x_4x_3 \oplus x_7x_2 \oplus x_5x_2 \oplus \\
 &\oplus x_4x_2 \oplus x_6x_4x_2 \oplus x_6x_1 \oplus x_6x_5x_1 \oplus x_7x_4x_1 \oplus x_3x_1 \oplus x_5x_2x_1.
 \end{aligned}$$

В данных формулах применены следующие обозначения:

- $x_n$  — значение  $n$ -го бита входного значения;
- $y_n$  —  $n$ -й бит выходного значения;
- $x_1x_2$  — операция логического «и» между значениями  $x_1$  и  $x_2$ .

Таблица  $S_9$  приведена в табл. 3.80.

Таблица 3.80

1c3	0cb	153	19f	1e3	0e9	0fb	035	181	0b9	117	1eb	133	009	02d	0d3
0c7	14a	037	07e	0eb	164	193	1d8	0a3	11e	055	02c	01d	1a2	163	118
14b	152	1d2	00f	02b	030	13a	0e5	111	138	18e	063	0e3	0c8	1f4	01b
001	09d	0f8	1a0	16d	1f3	01c	146	07d	0d1	082	1ea	183	12d	0f4	19e
1d3	0dd	1e2	128	1e0	0ec	059	091	011	12f	026	0dc	0b0	18c	10f	1f7
0e7	16c	0b6	0f9	0d8	151	101	14c	103	0b8	154	12b	1ae	017	071	00c
047	058	07f	1a4	134	129	084	15d	19d	1b2	1a3	048	07c	051	1ca	023
13d	1a7	165	03b	042	0da	192	0ce	0c1	06b	09f	1f1	12c	184	0fa	196
1e1	169	17d	031	180	10a	094	1da	186	13e	11c	060	175	1cf	067	119
065	068	099	150	008	007	17c	0b7	024	019	0de	127	0db	0e4	1a9	052
109	090	19c	1c1	028	1b3	135	16a	176	0df	1e5	188	0c5	16e	1de	1b1

Таблица 3.80 (окончание)

0c3	1df	036	0ee	1ee	0f0	093	049	09a	1b6	069	081	125	00b	05e	0b4
149	1c7	174	03e	13b	1b7	08e	1c6	0ae	010	095	1ef	04e	0f2	1fd	085
0fd	0f6	0a0	16f	083	08a	156	09b	13c	107	167	098	1d0	1e9	003	1fe
0bd	122	089	0d2	18f	012	033	06a	142	0ed	170	11b	0e2	14f	158	131
147	05d	113	1cd	079	161	1a5	179	09e	1b4	0cc	022	132	01a	0e8	004
187	1ed	197	039	1bf	1d7	027	18b	0c6	09c	0d0	14e	06c	034	1f2	06e
0ca	025	0ba	191	0fe	013	106	02f	1ad	172	1db	0c0	10b	1d6	0f5	1ec
10d	076	114	1ab	075	10c	1e4	159	054	11f	04b	0c4	1be	0f7	029	0a4
00e	1f0	077	04d	17a	086	08b	0b3	171	0bf	10e	104	097	15b	160	168
0d7	0bb	066	1ce	0fc	092	1c5	06f	016	04a	0a1	139	0af	0f1	190	00a
1aa	143	17b	056	18d	166	0d4	1fb	14d	194	19a	087	1f8	123	0a7	1b8
141	03c	1f9	140	02a	155	11a	1a1	198	0d5	126	1af	061	12e	157	1dc
072	18a	0aa	096	115	0ef	045	07b	08d	145	053	05f	178	0b2	02e	020
1d5	03f	1c9	1e7	1ac	044	038	014	0b1	16b	0ab	0b5	05a	182	1c8	1d4
018	177	064	0cf	06d	100	199	130	15a	005	120	1bb	1bd	0e0	04f	0d6
13f	1c4	12a	015	006	0ff	19b	0a6	043	088	050	15f	1e8	121	073	17e
0bc	0c2	0c9	173	189	1f5	074	1cc	1e6	1a8	195	01f	041	00d	1ba	032
03d	1d1	080	0a8	057	1b9	162	148	0d9	105	062	07a	021	1ff	112	108
1c0	0a9	11d	1b0	1a6	0cd	0f3	05c	102	05b	1d9	144	1f6	0ad	0a5	03a
1cb	136	17f	046	0e1	01e	1dd	0e6	137	1fa	185	08c	08f	040	1b5	0be
078	000	0ac	110	15e	124	002	1bc	0a2	0ea	070	1fc	116	15c	04c	1c2

Аналогично  $S_7$ ,  $S_9$  также может быть реализована с помощью следующих операций:

$$\begin{aligned}
 y_9 &= x_9x_5 \oplus x_9x_4 \oplus x_8x_4 \oplus x_8x_3 \oplus x_7x_3 \oplus x_7x_2 \oplus \\
 &\quad \oplus x_6x_2 \oplus x_6x_1 \oplus x_5x_1 \oplus 1; \\
 y_8 &= x_9x_7 \oplus x_6 \oplus x_8x_6 \oplus x_7x_6 \oplus x_6x_5 \oplus x_5x_4 \oplus \\
 &\quad \oplus x_9x_3 \oplus x_7x_3 \oplus x_2 \oplus x_9x_1 \oplus x_6x_1 \oplus x_4x_1 \oplus 1;
 \end{aligned}$$

$$\begin{aligned}
 y_7 &= x_9x_8 \oplus x_8x_6 \oplus x_5 \oplus x_9x_5 \oplus x_7x_5 \oplus x_6x_5 \oplus \\
 &\quad \oplus x_5x_4 \oplus x_9x_3 \oplus x_4x_3 \oplus x_8x_2 \oplus x_6x_2 \oplus x_1; \\
 y_6 &= x_9 \oplus x_8x_7 \oplus x_7x_5 \oplus x_4 \oplus x_8x_4 \oplus x_6x_4 \oplus \\
 &\quad \oplus x_5x_4 \oplus x_4x_3 \oplus x_8x_2 \oplus x_3x_2 \oplus x_7x_1 \oplus x_5x_1; \\
 y_5 &= x_8 \oplus x_9x_6 \oplus x_7x_6 \oplus x_9x_4 \oplus x_6x_4 \oplus x_3 \oplus x_7x_3 \oplus \\
 &\quad \oplus x_5x_3 \oplus x_4x_3 \oplus x_3x_2 \oplus x_7x_1 \oplus x_2x_1; \\
 y_4 &= x_7 \oplus x_9x_6 \oplus x_8x_5 \oplus x_6x_5 \oplus x_8x_3 \oplus x_5x_3 \oplus \\
 &\quad \oplus x_2 \oplus x_6x_2 \oplus x_4x_2 \oplus x_3x_2 \oplus x_9x_1 \oplus x_2x_1; \\
 y_3 &= x_9x_8 \oplus x_6 \oplus x_8x_5 \oplus x_7x_4 \oplus x_5x_4 \oplus x_7x_2 \oplus \\
 &\quad \oplus x_4x_2 \oplus x_1 \oplus x_9x_1 \oplus x_5x_1 \oplus x_3x_1 \oplus x_2x_1 \oplus 1; \\
 y_2 &= x_8 \oplus x_9x_8 \oplus x_8x_7 \oplus x_7x_6 \oplus x_9x_5 \oplus x_4 \oplus x_8x_3 \oplus \\
 &\quad \oplus x_6x_3 \oplus x_9x_2 \oplus x_5x_2 \oplus x_3x_2 \oplus x_8x_1 \oplus 1; \\
 y_1 &= x_9 \oplus x_9x_8 \oplus x_8x_7 \oplus x_5 \oplus x_9x_4 \oplus x_7x_4 \oplus x_6x_3 \oplus \\
 &\quad \oplus x_4x_3 \oplus x_9x_2 \oplus x_9x_1 \oplus x_6x_1 \oplus x_3x_1 \oplus 1.
 \end{aligned}$$

## Расшифровывание

Расшифровывание производится выполнением тех же операций, что и зашифровывание, но со следующими изменениями:

- фрагменты расширенного ключа используются в обратной последовательности;
- вместо операции  $FL$  используется обратная ей операция (обозначим ее  $FLI$ ).

Схема процедуры расшифровывания приведена на рис. 3.139.

Операция  $FLI$  определена следующим образом (рис. 3.140):

$$L' = L \oplus (R | KL_{i,j,2});$$

$$R' = R \oplus (L' \& KL_{i,j,1}).$$

## Описание алгоритмов

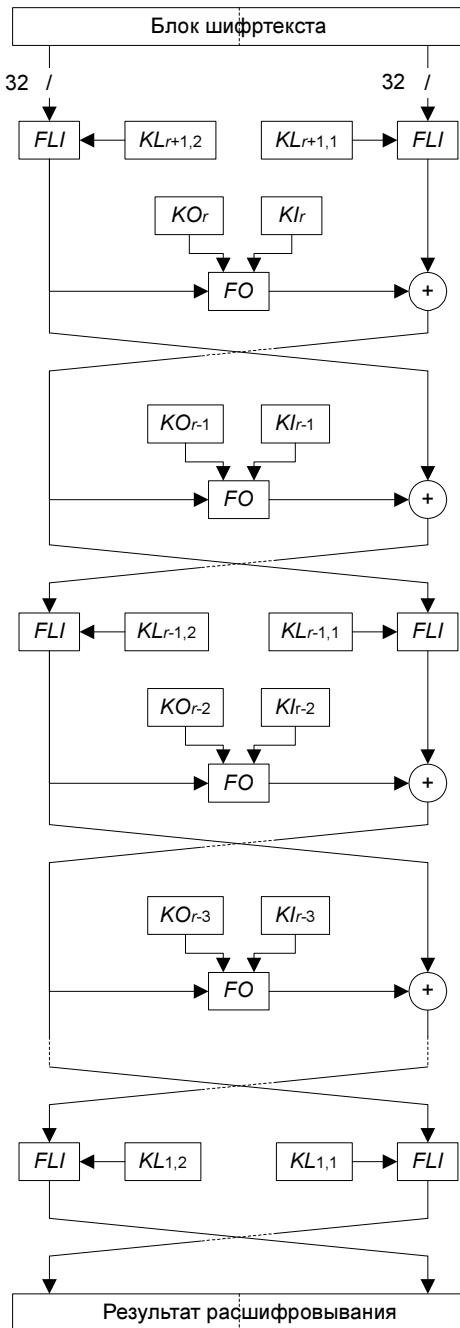
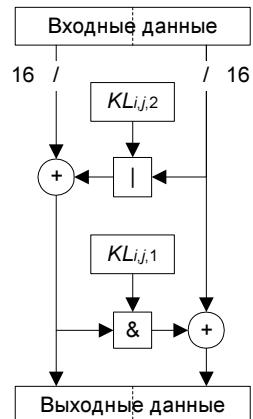


Рис. 3.139. Расшифровывание алгоритмом MISTY1

Рис. 3.140. Операция *FLI* алгоритма MISTY1

## Процедура расширения ключа

Задача процедуры расширения ключа состоит в формировании следующего набора используемых фрагментов ключа (для 8 раундов алгоритма):

- 20 фрагментов ключа  $KL_{i,j,m}$  ( $i = 1, 3, 5, 7, 9$ ;  $1 \leq j \leq 2$ ;  $1 \leq m \leq 2$ ), каждый из которых имеет размер по 16 битов;
- 32 16-битных фрагмента  $KO_{i,k}$  ( $1 \leq i \leq 8$ ,  $1 \leq k \leq 4$ );
- 24 7-битных фрагмента  $KI_{i,k,1}$  (при  $k = 4$ , т. е. в 4-м раунде операции  $FO$ , операция  $FI$  не выполняется);
- 24 9-битных фрагмента  $KI_{i,k,2}$ .

Таким образом, процедура расширения ключа вычисляет 1216 битов ключевой информации из 128-битного ключа шифрования алгоритма MISTY1. Выполняется это вычисление следующим образом:

1. 128-битный ключ делится на 8 фрагментов  $K_1 \dots K_8$  по 16 битов каждый.
2. Формируются значения  $K'_1 \dots K'_8$  (рис. 3.141): в качестве  $K'_n$  используется результат обработки значения  $K_n$  функцией  $FI$ , которая в качестве ключа (т. е. совокупности требуемых 7- и 9-битного фрагментов) использует значение  $K_{n+1}$  (здесь и далее, если индекс  $n$  фрагмента ключа превышает 8, то вместо него используется индекс  $n - 8$ ).

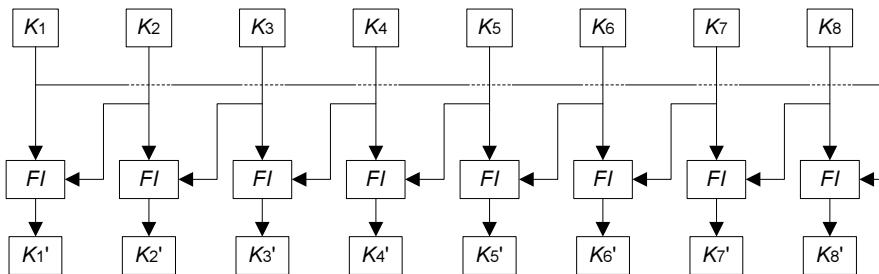


Рис. 3.141. Формирование промежуточных ключей в алгоритме MISTY1

3. Необходимые фрагменты расширенного ключа «набираются» по мере выполнения преобразований из массивов  $K_1 \dots K_8$  и  $K'_1 \dots K'_8$  согласно табл. 3.81 и 3.82.

Таблица 3.81

Назначение	$KO_{i,1}$	$KO_{i,2}$	$KO_{i,3}$	$KO_{i,4}$	$KI_{i,1}$	$KI_{i,2}$	$KI_{i,3}$
Фрагмент	$K_i$	$K_{i+2}$	$K_{i+7}$	$K_{i+4}$	$K_{i+5}'$	$K_{i+1}'$	$K_{i+3}'$

Таблица 3.82

Назначение	$KL_{i,1,1}$	$KL_{i,2,1}$	$KL_{i,1,2}$	$KL_{i,2,2}$
Фрагмент	$K_{(i+1)/2}$	$K_{(i+1)/2+2}'$	$K_{(i+1)/2+6}'$	$K_{(i+1)/2+4}$

4. 16-битный фрагмент  $KL_{i,k}$  делится на 7-битный фрагмент  $KL_{i,k,1}$  и 9-битный  $KL_{i,k,2}$ .

## Алгоритм MISTY2

В отличие от MISTY1, алгоритм MISTY2 не участвовал в конкурсе NESSIE (см. разд. 2.2) и не снискал такой широкой известности. MISTY2 незначительно отличается от MISTY1: он использует те же функции, но в несколько другой последовательности.

Структура MISTY2 приведена на рис. 3.142. В данном алгоритме существует не 2, а 4 различных типа раундов, которые выполняются поочередно, после чего повторяются требуемое количество раз [255, 257].

Структура раунда 1 (после приведенных далее операций в каждом раунде субблоки меняются местами):

1. Оба субблока обрабатываются операцией  $FL$ .
2. Левый субблок обрабатывается операцией  $FO$ .
3. Значение правого субблока обрабатывается операцией  $FL$  и накладывается на левый субблок операцией XOR.

Структура раунда 2:

1. Левый субблок обрабатывается операцией  $FO$ .
2. Значение правого субблока накладывается на левый.

Структура раунда 3:

1. Левый субблок обрабатывается операцией  $FO$ .
2. Значение правого субблока обрабатывается операцией  $FL$  и накладывается на левый.

Структура раунда 4 аналогична раунду 2.

Процедура расширения ключа, а также структура операций  $FO$  и  $FL$  аналогичны описанным выше для алгоритма MISTY1. Рекомендуемым количеством раундов для MISTY2 является 12, а не 8.

Алгоритм MISTY2 является более быстродействующим, чем MISTY1 при возможности параллельной обработки данных, поскольку может выполнять по 2 раунда параллельно [257, 310].

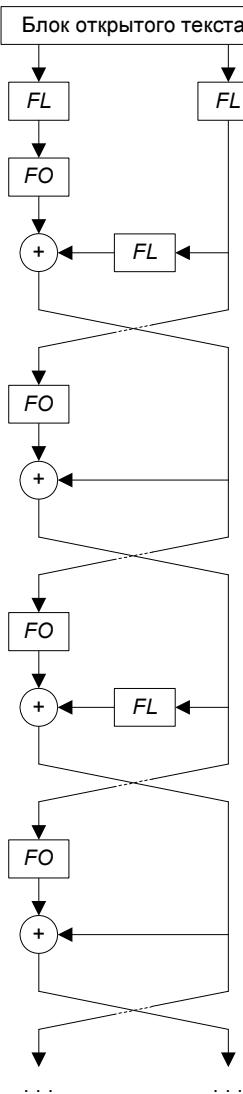


Рис. 3.142. Структура алгоритма MISTY2

## Алгоритм KASUMI

Алгоритм KASUMI несколько больше отличается от MISTY1, чем MISTY2. Основные отличия таковы [138]:

- операция  $FL$  обрабатывает только один субблок, но в каждом раунде;

- в данной операции присутствует дополнительное действие — вращение одного из обрабатываемых фрагментов на 1 бит;
- в операции *FO* используется 3 фрагмента ключа, а не 4;
- в операции *FI* используется 4 «прогона» данных через таблицы замен, а не 3, причем таблицы замен также изменены;
- принципиально изменена процедура расширения ключа.

Алгоритм KASUMI является стандартом шифрования данных в сетях сотовой связи третьего поколения. Существует мнение, что алгоритм KASUMI менее стоек, чем MISTY1, поскольку существуют атаки на его полнорундовую версию. По данным, приведенным в [236], KASUMI проигрывает MISTY1 по всем основным параметрам. В частности, несмотря на существенно более простую процедуру расширения ключа, расширение ключа в KASUMI выполняется в 2 раза медленнее, чем в MISTY1. Хотя считается, что изменения, внесенные в MISTY1 с целью получения алгоритма KASUMI, должны были, наоборот, усилить алгоритм [83].

Подробное описание KASUMI и результатов его криптоанализа можно найти в разд. 3.27.

Помимо алгоритмов MISTY2 и KASUMI, часть преобразований алгоритма MISTY1 используется и в алгоритме Camellia, который подробно описан в разд. 3.9.

## Криптоанализ алгоритмов MISTY1 и MISTY2

Алгоритм MISTY1 вызвал огромный интерес у криптологов как благодаря своему участию в конкурсе NESSIE, так и из-за своего родства с не менее известным алгоритмом KASUMI.

Первичный криптоанализ алгоритма выполнен его разработчиками, его результаты опубликованы в [259]. Авторы алгоритма доказывают криптостойкость алгоритма к линейному и дифференциальному криптоанализу, а также утверждают, что MISTY1 устойчив к сдвиговым атакам, использованию невозможных дифференциалов и ряду других атак.

В 2001 г. невозможные дифференциалы для атаки на варианты MISTY с уменьшенным количеством раундов применил Ульрих Кюн: 4-раундовый MISTY1 вскрывается при наличии  $2^{38}$  выбранных открытых текстов с помощью  $2^{62}$  тестовых операций шифрования; для атаки на 5-раундовый MISTY2 требуется  $2^{28}$  выбранных открытых текстов и  $2^{76}$  операций [230]. Он же в 2002 г. предложил принципиально новую атаку (названную «*slicing attack*»), действующую против MISTY1, которой для вскрытия 4-раундового

MISTY1 требуется существенно меньше ресурсов:  $2^{22,25}$  выбранных открытых текстов и  $2^{45}$  операций шифрования [231].

В том же 2001 г. известные криptoаналитики Ларс Кнудсен и Дэвид Вагнер предложили атаку на MISTY1 и MISTY2 методом интегрального криptoанализа. Данная атака позволяет вскрыть 5-раундовый MISTY1 при наличии  $2^{34}$  выбранных открытых текстов выполнением  $2^{48}$  операций шифрования или 6-раундовый MISTY2 при наличии  $2^{34}$  выбранных открытых текстов выполнением  $2^{71}$  операций [226]. Эти результаты являются лучшими из опубликованных для алгоритмов MISTY на текущий момент.

Проводились исследования и модифицированных вариантов алгоритма. В частности, в [370] исследован вариант без функций *FL*. Вскрывается 6 раундов алгоритма MISTY1 без *FL*-функций при наличии  $2^{12}$  выбранных открытых текстов выполнением  $2^{97}$  операций.

В результате анализа алгоритма MISTY1, проведенного в рамках конкурса NESSIE и до него, эксперты сделали вывод, что никаких серьезных уязвимостей данный алгоритм не имеет (они особо отметили, что именно структура алгоритма с вложенными сетями Фейстеля существенно затрудняет криptoанализ [307]). На этот вывод повлияли и результаты исследований алгоритма KASUMI, который, как было сказано выше, использует те же преобразования, что и MISTY1. В результате MISTY1 был выбран победителем конкурса NESSIE среди 64-битных алгоритмов блочного симметричного шифрования [305].

Аналогичные исследования были проведены и в рамках проекта CRYPTREC по выбору криptoалгоритмов для электронного правительства Японии. Эксперты проекта весьма положительно оценили алгоритм MISTY1, сделав вывод, что у него высокий запас криптостойкости, алгоритм имеет высокую скорость шифрования и весьма эффективен для аппаратной реализации [126].

Анализ алгоритмов семейства MISTY активно продолжается и в настоящее время. В качестве примера можно привести совсем недавнюю работу [238], в которой выполняется проверка стойкости преобразования *FO* против дифференциального криptoанализа. Стоит сказать и о том, что некоторые эксперты (например, в [310]) предрекают, что в ближайшем будущем будут появляться новые алгоритмы с MISTY-подобной структурой, поскольку данная структура весьма успешно зарекомендовала себя с точки зрения криптостойкости и быстродействия.

### 3.37. Алгоритм **Nimbus**

#### Структура алгоритма

Алгоритм шифрования Nimbus разработан Алексисом Уорнером Мачадо (Alexis Warner Machado) из компании Gauss Informatica, Бразилия. Алгоритм

исключительно прост в описании, структуру его раунда можно, фактически, представить одной формулой (рис. 3.143) [247]:

$$y = k_i * g(k_{5+i} \oplus x) \bmod 2^{64},$$

где:

- $x$  и  $y$  — соответственно, входное и выходное значения текущего ( $i$ -го) раунда преобразований,  $i = 0 \dots 4$ ;
- $k_0 \dots k_9$  — фрагменты расширенного ключа (см. далее);
- функция  $g()$  представляет собой битовую перестановку, она просто меняет местами биты №№  $n$  и  $63 - n$  для  $n = 0 \dots 31$ .

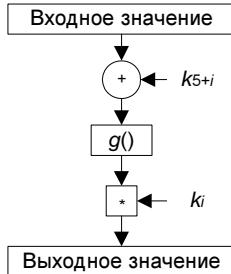


Рис. 3.143. Раунд алгоритма Nimbus

Расшифровывание выглядит не сложнее — так же, как и при зашифровывании, выполняется 5 раундов преобразований:

$$x = g(k'_{4-i} * y \bmod 2^{64}) \oplus k_{9-i},$$

где:

- $x$  и  $y$  — соответственно, выходное и входное значения  $i$ -го раунда расшифровывания;
- $k'_n$  — мультипликативная обратная величина к  $k_n$  по модулю  $2^{64}$ .

Стоит сказать, что автор алгоритма не ограничивает размер блока 64 битами: можно шифровать блоками по 128 и 256 битов и более. При этом используются вычисления по модулю  $2^{128}$ ,  $2^{256}$  и т. д. Поскольку с увеличением размера блока сложность подобных вычислений нелинейно возрастает, дальнейшее увеличение размера блока не выглядит оправданным.

## Процедура расширения ключа

Алгоритм Nimbus использует ключи шифрования переменного размера, от 64 до 576 битов; размер ключа шифрования должен быть кратен 64 битам. Исходный ключ представляется в виде последовательности 64-битных

блоков  $K_0 \dots K_m$ . На их основе генерируется последовательность подключей  $k_0 \dots k_9$ :

1. Выполняется инициализация подключей, состоящая в обнулении последовательности  $k_0 \dots k_9$ .
  2. В цикле по  $i$  от 0 до  $m$  ( $m$  — размер ключа в 64-битных блоках) выполняются следующие действия.
- Вычисляется промежуточное значение  $x$ :

$$x = K_i \oplus E(K_i),$$

где  $E()$  представляет собой зашифровывание входного значения алгоритмом Nimbus на наборе фиксированных подключей, который выглядит согласно табл. 3.83.

*Таблица 3.83*

$k_0$	243F6A8885A308D3
$k_1$	13198A2E03707345
$k_2$	A4093822299F31D1
$k_3$	082EFA98EC4E6C89
$k_4$	452821E638D01377
$k_5$	BE5466CF34E90C6C
$k_6$	C0AC29B7C97C50DD
$k_7$	3F84D5B5B5470917
$k_8$	9216D5D98979FB1B
$k_9$	D1310BA698DFB5AC

Эти константы представляют собой шестнадцатеричную запись дробной части числа  $\pi$ .

- В цикле по  $j$  от 0 до 9 выполняются следующие операции:

$$x = E(x),$$

$$k_j = x \oplus E(x + k_j \bmod 2^{64}).$$

3. Обеспечивается нечетность значений подключей  $k_0 \dots k_4$ , которые используются в операциях умножения при шифровании — путем установки младших битов  $k_0 \dots k_4$  в единичное значение.

Стоит обратить внимание на следующие свойства алгоритма и процедуры расширения ключа:

- при необходимости достаточно легко можно увеличить количество раундов алгоритма (в абсолютном большинстве алгоритмов это приводит к увеличению криптостойкости за счет снижения скорости шифрования); для этого достаточно лишь увеличить количество подключей (и соответствующие циклы в описанных выше шагах 1–3 процедуры расширения ключа) и, собственно, количество раундов алгоритма;
- расширение ключа никоим образом невозможно выполнять «на лету», что можно считать недостатком алгоритма.

## Достоинства и недостатки алгоритма

В спецификации алгоритма [247] его автор утверждал, что Nimbus обладает стойкостью против всех известных атак, в случае, если количество раундов алгоритма превышает 4 (а в спецификации установлено 5 раундов алгоритма). Однако алгоритм не был выбран во второй раунд конкурса NESSIE из-за найденной специалистами атаки, которую весьма реально осуществить на практике [307].

Данная атака была изобретена известными криптологами из института Technion, Израиль: Эли Бихамом и Владимиром Фурманом (Vladimir Furman) [71]. Атака позволяет вычислить ключ шифрования полнораундового алгоритма Nimbus на основе всего 256 выбранных открытых текстов и соответствующих им шифртекстов путем выполнения не более  $2^{10}$  тестовых операций шифрования. При наличии у злоумышленника возможности выбора шифртекстов для атаки требуется еще меньше материала — достаточно 136 выбранных шифртекстов (и соответствующих им открытых текстов) при той же вычислительной сложности. Впоследствии специалистам из Университета Беркли (США) удалось распространить принципы атаки, предложенной Бихамом и Фурманом, на ряд других алгоритмов шифрования [101].

Таким образом, алгоритм Nimbus оказался наиболее слабым из всех алгоритмов блочного шифрования, участвовавших в конкурсе NESSIE.

## 3.38. Алгоритм Noekeon

Алгоритм Noekeon разработан в 2000 г. коллективом разработчиков из Бельгии; авторы алгоритма:

- Джоан Деймен, Михаэль Петерс (Michael Peeters) и Жиль Ван Аске (Gilles Van Assche) из компании Proton World;
- Винсент Риджмен из Католического Университета г. Лювен.

И снова вспомним алгоритм Rijndael, поскольку среди авторов алгоритма Noekeon присутствуют оба автора Rijndael — Джоан Деймен и Винсент Риджмен. Однако Noekeon несравненно меньше похож на Rijndael, чем, например, другие участники конкурса NESSIE: Anubis (см. разд. 3.5) и Grand Cru (см. разд. 3.20).

Noekeon шифрует данные 128-битными блоками с использованием 128-битного ключа шифрования. Алгоритм существует в двух режимах: прямом (direct mode) и косвенном (indirect mode), разница между которыми существует лишь в процедуре расширения ключа; оба режима будут рассмотрены далее.

## Структура алгоритма

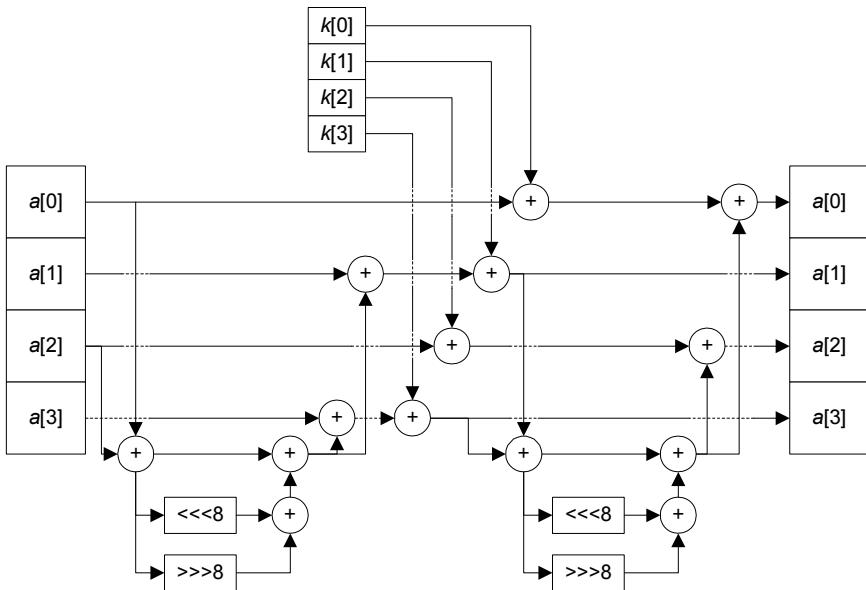
Алгоритм Noekeon выполняет 16 раундов преобразований. Блок данных представляется в виде массива из четырех 32-битных строк  $a[0]...a[3]$ . В каждом раунде алгоритма выполняются следующие действия [131]:

1. Верхняя строка массива данных складывается с помощью операции XOR с константой  $C1[r]$ , где  $r$  — номер текущего раунда (начиная с 0). Константы  $C1[0]...C1[15]$  будут описаны далее.
2. Линейное преобразование  $\theta$  выполняет следующие операции над строками массива данных с участием 128-битного «рабочего ключа» (working key), который также представляется в виде 32-битных строк  $k[0]...k[3]$  (рис. 3.144):

$$\begin{aligned}
 t &= a[0] \oplus a[2]; \\
 t &= t \oplus (t \ggg 8) \oplus (t \lll 8); \\
 a[1] &= a[1] \oplus t; \\
 a[3] &= a[3] \oplus t; \\
 a[0] &= a[0] \oplus k[0]; \\
 a[1] &= a[1] \oplus k[1]; \\
 a[2] &= a[2] \oplus k[2]; \\
 a[3] &= a[3] \oplus k[3]; \\
 t &= a[1] \oplus a[3]; \\
 t &= t \oplus (t \ggg 8) \oplus (t \lll 8); \\
 a[0] &= a[0] \oplus t; \\
 a[2] &= a[2] \oplus t,
 \end{aligned}$$

где  $t$  — временная переменная.

## Описание алгоритмов

Рис. 3.144. Операция  $\Theta$  алгоритма Noekeon

3. Верхняя строка массива данных снова складывается операцией XOR с константой  $C2[r]$ . В каждом раунде только одна из констант  $C1[r]$  и  $C2[r]$  не является нулевой и равна константе раунда  $C[r]$ . При зашифровывании нулю равны константы  $C2[r]$  (т. е., фактически, в каждом раунде зашифровывания не выполняется действие № 3). При расшифровывании — наоборот, нулевыми являются константы  $C1[r]$  и, соответственно, не выполняется действие № 1.

Каждая из констант  $C[r]$  содержит 3 нулевых байта и один (младший) байт с отличным от 0 значением, которое обозначим как  $C'[r]$ . Таким образом, в каждом раунде алгоритма совокупность действий № 1 и № 3 сводится к модификации только одного байта массива данных (рис. 3.145):

$$b_{0,3} = a_{0,3} \oplus C'[r].$$

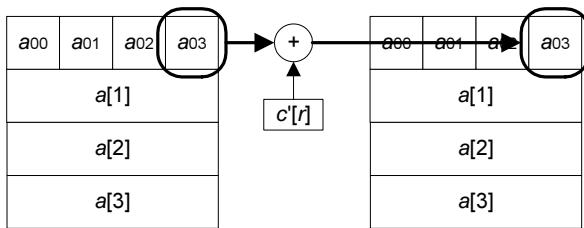
Константы  $C'[r]$  итеративно вычисляются по следующим правилам:

- инициализируется  $C'[0]$  (здесь и далее указаны шестнадцатеричные значения):

$$C'[0] = 80;$$

- вычисляются последующие константы:

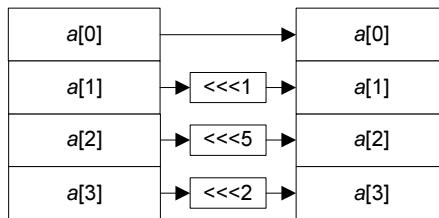
$C'[i] = (C'[i-1] \lll 1) \oplus 1B$ , если  $C'[i-1] \& 80 > 0$  (где  $\&$  — побитовая логическая операция «и»), иначе  $C'[i] = C'[i-1] \lll 1$ .



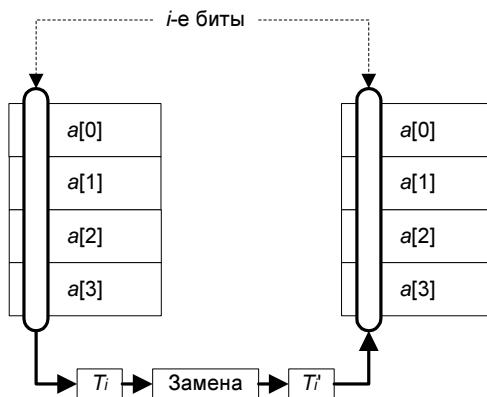
**Рис. 3.145.** Наложение раундовой константы в алгоритме Noekeon

4. Вращение данных  $\pi_1$  выполняет побитовый циклический сдвиг влево трех строк массива данных (рис. 3.146):

- строка  $a[1]$  сдвигается на 1 бит;
- $a[2]$  сдвигается на 5 битов;
- $a[3]$  — на 2 бита.



**Рис. 3.146.** Операция  $\pi_1$  алгоритма Noekeon



**Рис. 3.147.** Операция  $\gamma$  алгоритма Noekeon

## Описание алгоритмов

5. Операция  $\gamma$ , представляющая собой табличную замену. Замена выполняется следующим образом (рис. 3.147):

- массив данных представляется в виде 32 4-битных субблоков,  $i$ -й субблок  $T_i$  формируется значениями  $i$ -х битов каждой строки массива, т. е.:

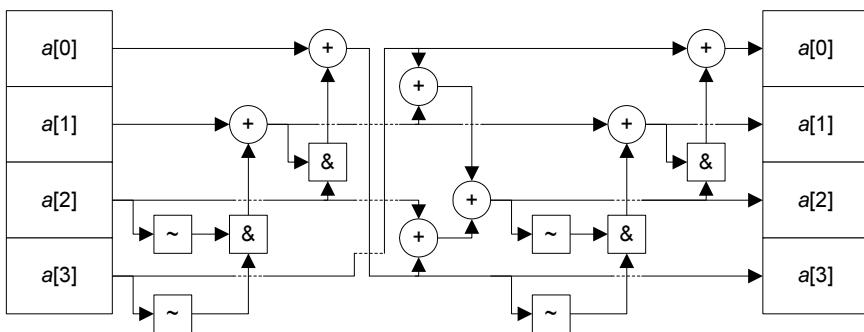
$$T_i = a[3]_i * 8 + a[2]_i * 4 + a[1]_i * 2 + a[0]_i,$$

где  $a[n]_i$  —  $i$ -й бит  $n$ -й строки массива данных;

- значение каждого субблока заменяется согласно таблице замен (см. табл. 3.84).

Таблица 3.84

Входное значение	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Выходное значение	7	A	2	C	4	8	F	0	5	9	1	E	3	D	B	6

Рис. 3.148. Операция  $\gamma$  в виде последовательности операций

Эту табличную замену можно представить и в виде следующей последовательности операций над строками массива данных (рис. 3.148):

$$a[1] = a[1] \oplus (\sim a[3] \& \sim a[2]);$$

$$a[0] = a[0] \oplus (a[2] \& a[1]);$$

$$t = a[3];$$

$$a[3] = a[0];$$

$$a[0] = t;$$

$$a[2] = a[0] \oplus a[1] \oplus a[2] \oplus a[3];$$

$$a[1] = a[1] \oplus (\sim a[3] \& \sim a[2]);$$

$$a[0] = a[0] \oplus (a[2] \& a[1]),$$

где  $\sim x$  — побитовый комплемент к  $x$ .

6. Вращение данных  $\pi_2$  является обратным к  $\pi_1$ , т. е. выполняет вращение тех же трех строк массива данных на указанное в описании операции  $\pi_1$  число битов вправо.

Помимо описанных выше операций, после 16 раундов выполняется заключительное преобразование, состоящее из двух операций:

- наложения на  $a[0]$  константы  $C[16]$  операцией XOR;
- дополнительной операции  $\theta$  над массивом данных.

## Расшифровывание

Процедура расшифровывания практически идентична зашифровыванию, за исключением следующих отличий.

- Как было сказано выше, при расшифровывании константы  $C1[r]$  и  $C2[r]$  меняются местами.
- Вместо прямой операции  $\theta$  выполняется обратная — в ней используется модифицированный рабочий ключ  $K'$ :

$$K' = \theta(Null, K),$$

где:

- $K$  — рабочий ключ;
- $Null$  — 32-битный блок, состоящий из нулевых битов.

Модификация рабочего ключа выполняется однократно до начала расшифровывания, после чего  $K'$  используется во всех операциях, где должен использоваться ключ  $K$ .

- Раунды расшифровывания нумеруются в обратном порядке — соответственно, в обратном порядке используются константы  $C[16]...C[1]$ .
- Иначе выполняется заключительное преобразование (после выполнения всех раундов расшифровывания): сначала выполняется дополнительная операция  $\theta$  (с участием  $K'$ ), после чего выполняется наложение на  $a[1]$  константы  $C[0]$ .

Следует учесть, что для обеспечения вычисления констант  $C[16]...C[0]$  «на лету» в процессе расшифровывания данные константы могут вычисляться и в обратном порядке.

## Процедура расширения ключа

Расширение ключа алгоритма Noekeon выполняется исключительно просто, как в прямом, так и в косвенном режиме алгоритма.

- В прямом режиме расширение ключа полностью отсутствует: 128-битный ключ шифрования используется в качестве 128-битного рабочего ключа.
- В косвенном режиме рабочий ключ получается из ключа шифрования зашифровыванием алгоритмом Noekeon значения *Null* на ключе шифрования (который в данном случае используется в качестве рабочего ключа).

Авторы алгоритма рекомендуют использовать прямой режим шифрования только в том случае, когда невозможно выполнение потенциальным злоумышленником атак на связанных ключах.

## Криptoанализ алгоритма

На рассмотрение в рамках конкурса NESSIE (см. разд. 2.2) были приняты оба режима алгоритма Noekeon. И оба режима оказались подвержены атаке на основе связанных ключей (в том числе косвенный режим), которую предложили криптологи Ларс Кнудсен и Хавард Раддум (Havard Raddum) в своей работе [220]. Кроме того, ими же было доказано, что критерии создания таблиц замен (используемых в операции  $\gamma$ ) не способствуют высокой криптостойкости алгоритма: при генерации согласно данным критериям таблицы замен результирующий алгоритм с вероятностью 85 % окажется подвержен линейному и/или дифференциальному криptoанализу [220]. Этих причин оказалось достаточно для невыхода алгоритма Noekeon во второй раунд конкурса [308].

## 3.39. Алгоритм NUSH

Алгоритм NUSH предложен на конкурс NESSIE российской компанией «ЛАН Крипто», которая представляет собой одну из отечественных компаний — разработчиков средств криптографической защиты информации (в том числе и на основе криptoалгоритмов собственной разработки). Авторы алгоритма: Президент компании «ЛАН Крипто» Лебедев А. Н. и Президент Российской Криптологической Ассоциации «РусКрипто» Волчков А. А.

Алгоритм NUSH имеет несколько фиксированных размеров шифруемого блока данных: 64, 128 и 256 битов. Рассмотрим 64-битный вариант алгоритма.

## Структура алгоритма

Структура алгоритма показана на рис. 3.149 [237]. Шифруемый блок данных разбивается на 4 субблока по 16 битов (обозначаемые как  $a, b, c, d$ ).

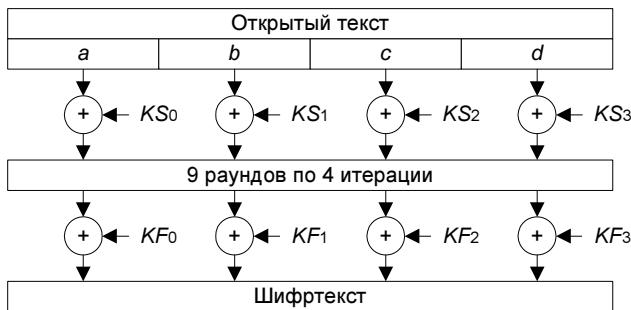


Рис. 3.149. Структура алгоритма NUSH

Прежде всего выполняется начальное преобразование, состоящее в том, что на каждый субблок операцией XOR накладывается фрагмент расширенного ключа (процедура расширения ключа будет описана далее) для начального преобразования  $KS_0...KS_3$ :

$$\begin{aligned} A &= a \oplus KS_0; \\ B &= b \oplus KS_1; \\ C &= c \oplus KS_2; \\ D &= d \oplus KS_3, \end{aligned}$$

где  $A$ ,  $B$ ,  $C$ ,  $D$  — значения соответствующих субблоков после выполнения текущей операции.

Затем выполняются 9 раундов преобразований; структура раунда будет описана далее.

По завершении 9 раундов выполняется финальное преобразование, в котором операцией XOR на субблоки накладываются подключи для финального преобразования  $KF_0...KF_3$ :

$$\begin{aligned} A &= a \oplus KF_0; \\ B &= b \oplus KF_1; \\ C &= c \oplus KF_2; \\ D &= d \oplus KF_3. \end{aligned}$$

Основой каждого раунда алгоритма является преобразование ("итерация" согласно спецификации алгоритма)  $R(X1, X2, X3, X4, k, s)$ ; в одной итерации выполняются следующие действия (рис. 3.150):

$$\begin{aligned} Y3 &= (X2 + (X3 \oplus k) \bmod 2^{16}) \ggg s; \\ Y1 &= X1 + (Y3 \# X4) \bmod 2^{16}; \end{aligned}$$

## Описание алгоритмов

$$Y2 = X2;$$

$$Y4 = X4,$$

где:

- $X1 \dots X4$  и  $Y1 \dots Y4$  — соответственно, входные и выходные значения обрабатываемых субблоков;
- $k$  — модифицированный фрагмент расширенного ключа для текущей итерации;
- $>>>$  — побитовый циклический сдвиг операнда вправо на переменное число битов;
- $s$  — число битов сдвига, различно для каждой итерации и выбирается из таблицы  $S$  (табл. 3.85).

**Таблица 3.85**

Раунд	Итерация	$s$	Раунд	Итерация	$s$
0	0	4	4	2	5
0	1	7	4	3	1
0	2	11	5	0	2
0	3	8	5	1	4
1	0	7	5	2	12
1	1	14	5	3	3
1	2	5	6	0	9
1	3	4	6	1	2
2	0	8	6	2	11
2	1	2	6	3	13
2	2	9	7	0	12
2	3	4	7	1	3
3	0	13	7	2	6
3	1	1	7	3	11
3	2	14	8	0	7
3	3	6	8	1	15
4	0	7	8	2	4
4	1	12	8	3	14

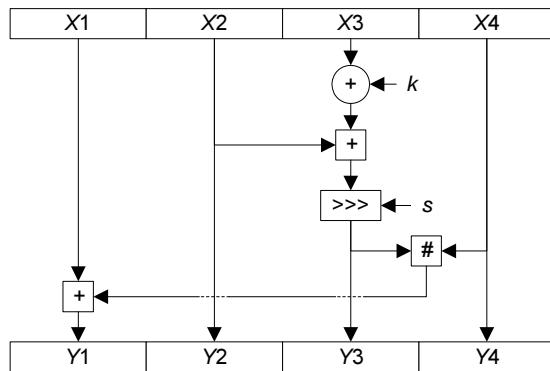


Рис. 3.150. Итерация алгоритма NUSH

Символом # обозначена побитовая логическая операция "и" (&) или побитовая логическая операция "или" (), конкретная из которых также выбирается согласно табл. 3.86.

Таблица 3.86

Раунд	Итерация	#	Раунд	Итерация	#
0	0	&	4	2	&
0	1		4	3	&
0	2	&	5	0	&
0	3		5	1	&
1	0		5	2	&
1	1		5	3	
1	2		6	0	&
1	3		6	1	
2	0	&	6	2	
2	1		6	3	
2	2		7	0	&
2	3	&	7	1	
3	0		7	2	&
3	1	&	7	3	&
3	2		8	0	
3	3		8	1	
4	0		8	2	&
4	1		8	3	

Раунд состоит из четырех итераций (в таблицах выше пронумерованы от 0 до 3), выполняемых по следующему закону:

$$\begin{aligned} & R(a, b, c, d, KRC_{4i}, S[4i]); \\ & R(b, c, d, a, KRC_{4i+1}, S[4i+1]); \\ & R(c, d, a, b, KRC_{4i+2}, S[4i+2]); \\ & R(d, a, b, c, KRC_{4i+1+3}, S[4i+3]), \end{aligned}$$

где:

- $i$  — номер текущего раунда, начиная с 0;
- $KRC_n$  — модифицированный фрагмент расширенного ключа для текущей итерации ( $KR_n$ ):

$$KRC_n = KR_n + c \bmod 2^{16},$$

где  $c$  — модифицирующая константа согласно табл. 3.87.

Таблица 3.87

Раунд	Итерация	$c$	Раунд	Итерация	$c$
0	0	AC25	4	2	96DA
0	1	8A93	4	3	905F
0	2	243D	5	0	D631
0	3	262E	5	1	AA62
1	0	F887	5	2	4D15
1	1	C4F2	5	3	70CB
1	2	8E36	6	0	7533
1	3	9FA1	6	1	45FC
2	0	7DC0	6	2	5337
2	1	6A29	6	3	D25E
2	2	6D84	7	0	A926
2	3	34BD	7	1	1C7B
3	0	A267	7	2	5F12
3	1	CC15	7	3	4ECC
3	2	04FE	8	0	3C86
3	3	B94A	8	1	28DB
4	0	DF24	8	2	FC01
4	1	40EF	8	3	7CB1

## Расшифровывание

Расшифровывание выполняется применением обратных операций в обратной последовательности. Таким образом, при расшифровывании сначала выполняется обратное финальное преобразование, затем 9 раундов по 4 обратных итерации, после чего — обратное начальное преобразование.

Обратные финальное и начальное преобразования полностью аналогичны прямым — на субблоки  $a, b, c, d$  операцией XOR накладываются, соответственно, фрагменты расширенного ключа  $KF_0 \dots KF_3$  и  $KS_0 \dots KS_3$ .

Итерации каждого раунда также выполняются в обратной последовательности ( $i = 9 \dots 1$ ):

$$R^{-1}(d, a, b, c, KRC_{4i-1}, S[4i-1]);$$

$$R^{-1}(c, d, a, b, KRC_{4i-2}, S[4i-2]);$$

$$R^{-1}(b, c, d, a, KRC_{4i-3}, S[4i-3]);$$

$$R^{-1}(a, b, c, d, KRC_{4i-4}, S[4i-4]).$$

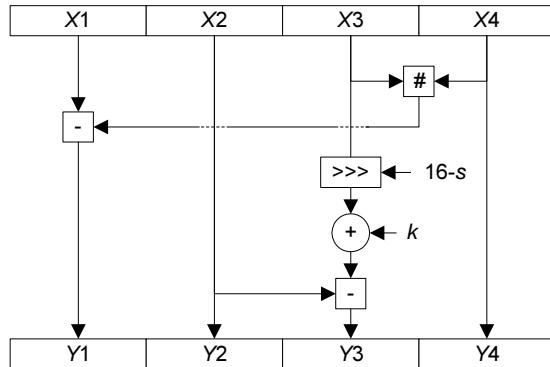


Рис. 3.151. Обратная итерация алгоритма NUSH

Обратная итерация  $R^{-1}(X1, X2, X3, X4, k, s)$  выглядит следующим образом (рис. 3.151):

$$Y4 = X4;$$

$$Y2 = X2;$$

$$Y1 = X1 - (X3 \# X4) \bmod 2^{16};$$

$$Y3 = ((X3 >>> (16 - s)) \oplus k) - X2 \bmod 2^{16}.$$

## Процедура расширения ключа

Процедура расширения ключа весьма проста и напоминает таковую у отечественного стандарта шифрования ГОСТ 28147-89 (*см. разд. 3.1*) — исходный ключ шифрования делится на фрагменты  $K_0 \dots K_{n-1}$  ( $n$  — размер ключа шифрования в 16-битных словах), т. е.:

- $K_0 \dots K_7$  для 128-битного ключа;
- $K_0 \dots K_{11}$  для 192-битного ключа;
- $K_0 \dots K_{15}$  для 256-битного ключа.

Затем фрагменты ключа используются в итерациях, а также в начальном и финальном преобразованиях согласно табл. 3.88.

Таблица 3.88

Применение	Используемый фрагмент ключа (в зависимости от его размера)		
	128 битов	192 бита	256 битов
$KS_0$	$K_4$	$K_4$	$K_{12}$
$KS_1$	$K_5$	$K_5$	$K_{13}$
$KS_2$	$K_6$	$K_6$	$K_{14}$
$KS_3$	$K_7$	$K_7$	$K_{15}$
$KF_0$	$K_3$	$K_{11}$	$K_{13}$
$KF_1$	$K_2$	$K_{10}$	$K_{12}$
$KF_2$	$K_1$	$K_9$	$K_{15}$
$KF_3$	$K_0$	$K_8$	$K_{14}$
$KR_i$	$K_{i \bmod n}$		

Символом  $i$  в табл. 3.88 обозначен сквозной номер итерации (от 0 до 35), т. е. в итерациях фрагменты ключа используются поочередно в прямом порядке.

## Криптостойкость алгоритма

Алгоритм NUSH не выбран во второй раунд конкурса [312] благодаря атаке, предложенной китайскими учеными Вен-Лингом Ву (Wen-Ling Wu) и Дэнг-Гуо Фенгом (Deng-Guo Feng) [398]. Найденная ими атака позволяет методом

линейного криптоанализа вычислить 128-битный ключ шифрования алгоритма NUSH при наличии  $2^{58}$  известных открытых текстов (и соответствующих им шифртекстов) выполнением  $2^{124}$  тестовых операций шифрования. При этом, если имеется  $2^{62}$  известных открытых текстов, вычислительная сложность снижается до  $2^{55}$  тестовых операций шифрования. Аналогичные атаки существуют для 192- и 256-битных ключей.

Следует отметить, что данные атаки весьма сложно реализуемы на практике, однако их оказалось достаточно в качестве доказательства недостаточного запаса криптостойкости (security margin) [312] алгоритма NUSH.

## 128-битный вариант

Алгоритм NUSH (аналогично, например, известному алгоритму RC5 — см. разд. 3.42) весьма легко преобразуется под другие размеры блока. На конкурс NESSIE, помимо 64-битного варианта, были представлены еще и варианты алгоритма, обрабатывающие 128-битные и 256-битные блоки данных [237].

Отличия 128-битного алгоритма NUSH от 64-битного состоят в следующем:

- выполняется 17 раундов преобразований вместо 9;
- блок данных делится на те же 4 субблока, но они имеют размер по 32 бита (а не по 16 битов). Соответственно, операции сложения и вычитания выполняются по модулю  $2^{32}$ ;
- используются другие константы  $c$  (константы для модификации ключей, используемых в итерациях алгоритма) и  $s$  (число битов сдвига), а также другая последовательность логических операций #; их значения сведены в табл. 3.89 (для  $c$  указаны шестнадцатеричные значения);

Таблица 3.89

Раунд	Итерация	$s$	$c$	#
0	0	7	9B28A37B	&
0	1	5	9DE5B521	
0	2	15	0B8EE0D7	&
0	3	14	672AA715	
1	0	3	0E356C9F	
1	1	30	BF54692A	

Таблица 3.89 (продолжение)

Раунд	Итерация	s	c	#
1	2	4	DC9E15C8	
1	3	23	06D736E8	
2	0	13	9263E8CF	&
2	1	12	1FCD682D	
2	2	26	7368B074	
2	3	16	2654F15A	&
3	0	9	00EB3E4D	
3	1	28	18D62F6D	&
3	2	8	632A557A	
3	3	18	1D953D21	
4	0	23	CD4B2ACD	
4	1	8	49A0D3F4	
4	2	26	C443FC66	&
4	3	4	E84C5BCB	&
5	0	29	F750A732	&
5	1	16	2CDE9942	&
5	2	2	370C437A	&
5	3	22	DA8B5654	
6	0	23	99A76750	&
6	1	11	A1559437	
6	2	26	9EA46718	
6	3	13	83E984F8	
7	0	20	AB5692E4	&
7	1	5	A6C5C46A	
7	2	28	25FB110E	&
7	3	17	55955B2E	&
8	0	19	FA639063	
8	1	22	027E4DC6	

**Таблица 3.89** (продолжение)

<b>Раунд</b>	<b>Итерация</b>	<i>s</i>	<i>c</i>	#
8	2	6	919E96B2	&
8	3	25	62E96D0C	
9	0	12	AA7DE138	
9	1	24	A674A66C	&
9	2	27	B3F54983	
9	3	10	AE29D0DB	&
10	0	16	599470CB	
10	1	24	3B2E3FA0	&
10	2	9	A354CC6F	&
10	3	13	516AF8C4	
11	0	5	ADE11D33	
11	1	10	860D95F2	&
11	2	26	BC2731A4	&
11	3	30	CCD12BAA	&
12	0	9	BA518E95	&
12	1	16	22F7583A	&
12	2	28	6C0A5FE8	&
12	3	24	8FAC2D74	&
13	0	27	D129E934	&
13	1	6	11DCE4C9	&
13	2	7	362F2F4A	
13	3	15	6CCB630D	&
14	0	1	97919D88	
14	1	13	823F95AC	
14	2	15	67C99A98	
14	3	1	8E91D0CB	&
15	0	23	AB796817	&

Таблица 3.89 (окончание)

Раунд	Итерация	$s$	$c$	#
15	1	28	356459A7	&
15	2	12	668D9FA8	
15	3	2	0D4DBF40	
16	0	28	1ACCE5D8	&
16	1	14	F53B24C1	
16	2	15	6DB89876	&
16	3	12	5C965DA5	

- изменена процедура расширения ключа. В 128-битном алгоритме NUSH фрагменты исходного ключа  $K_0 \dots K_{n-1}$  ( $n$  — размер ключа шифрования в 32-битных словах) используются в итерациях ( $i$  — сквозной номер итерации, начиная с 0), а также в начальном и финальном преобразованиях следующим образом (см. табл. 3.90).

Таблица 3.90

Применение	Используемый фрагмент ключа (в зависимости от его размера)		
	128 битов	192 бита	256 битов
$KS_0$	$K_3$	$K_2$	$K_4$
$KS_1$	$K_2$	$K_3$	$K_5$
$KS_2$	$K_1$	$K_4$	$K_6$
$KS_3$	$K_0$	$K_5$	$K_7$
$KF_0$	$K_1$	$K_5$	$K_5$
$KF_1$	$K_0$	$K_4$	$K_4$
$KF_2$	$K_3$	$K_3$	$K_7$
$KF_3$	$K_2$	$K_2$	$K_6$
$KR_i$	$K_{i \bmod n}$		

## 256-битный вариант

256-битный вариант алгоритма, по сути, имеет те же отличия от 64-битного варианта:

- количество раундов алгоритма увеличено до 33;
- субблоки имеют размер по 64 бита, а операции сложения и вычитания выполняются по модулю  $2^{64}$ ;
- используются другие константы  $c$  и  $s$  согласно табл. 3.91;

Таблица 3.91

Раунд	Итерация	$s$	$c$
0	0	12	1A028E3B458FE65F
0	1	45	10CB1C5CAC3C7A75
0	2	7	0AA54C8D55CC6F5E
0	3	48	EE4AC8B12E2FC8D5
1	0	14	F787D15C240344D7
1	1	43	CACCAF60F2998693
1	2	8	4EA93E4DF9558E82
1	3	54	B57CDA0316BC1C92
2	0	49	623C7496C0D6FB68
2	1	47	BD7B065E84D852A9
2	2	37	A6CD2E5C6B1A30E7
2	3	55	788D9EFC078281B5
3	0	58	D0CF11A8FF9943E4
3	1	32	D04F01C7F3EA8E96
3	2	16	5313F574E5D1D2C8
3	3	36	DC8AB4437AAD50CF
4	0	13	66ED63D790921A4D
4	1	35	FA351C5183EBDA0B
4	2	50	DA694B14554D17C9
4	3	58	0A392FA5DE785CD1

Таблица 3.91 (продолжение)

Раунд	Итерация	<i>s</i>	<i>c</i>
5	0	21	75B1D5DE6561D08C
5	1	56	BC128DB2F22C591E
5	2	4	D19F06A961BC6E36
5	3	52	F3F2D208215DDA85
6	0	32	D9A5D482F930B1AF
6	1	19	FD98B3A189AD9851
6	2	28	B671A790FB204AE3
6	3	10	4E3B9DB2A290EC98
7	0	63	2CA2AFB114DF74A2
7	1	53	705CE63837B3616D
7	2	50	679D058EAL89A2EE
7	3	27	8398BAB59E3A506C
8	0	18	181F8AEFD8499AD4
8	1	40	17C41D9833728FE9
8	2	13	7E692E4DB9D09471
8	3	14	C900CDE6CB8AA557
9	0	8	EB2B8576C0419FE3
9	1	21	927C3FE32C9A2365
9	2	6	427410EB1EACBE4F
9	3	59	18A6FE2878B4D78D
10	0	17	436EB84357C5342F
10	1	5	1B94C23F94C24B3E
10	2	23	D3D831585E585A9C
10	3	10	F37E22A1587B9670
11	0	32	96A27FA6164197CD
11	1	20	C21BC4EAF449AC7E
11	2	53	BCCE8974A35A69D4
11	3	3	7FA98C9B495C2782

**Таблица 3.91** (продолжение)

<b>Раунд</b>	<b>Итерация</b>	<i>s</i>	<i>c</i>
12	0	20	3B64D65041406FFB
12	1	42	AF82F6418C48F7DC
12	2	1	13B7D80A170E6AB6
12	3	58	09DFC1BBF5A51842
13	0	12	45B2F2934E2BECC4
13	1	30	F456D827335C90D3
13	2	38	7A2C6EE4672634D8
13	3	6	3AA0D9523BBBD398
14	0	23	D578F2AEA135F841
14	1	61	9A6635DA5227B8E9
14	2	7	F40F12A5B07BC3D8
14	3	12	BBD16B68649B4271
15	0	33	042753CE1B63F27B
15	1	41	A471D892D743F58D
15	2	17	B6CACF5958204C67
15	3	35	FB7786E2234AA30A
16	0	30	97EB25E4C9F33038
16	1	3	CD5D27E1802E58F4
16	2	60	0289FBE8CE5BD06A
16	3	55	26DBAA50CBC1E8B9
17	0	37	4116B2B8D89AFF86
17	1	50	1D658D6EEF814E49
17	2	12	A4B511D2427E3F73
17	3	41	E2A77BD9898E1326
18	0	7	65DEA88074B941FD
18	1	40	8E55B0DC3CEE4398
18	2	35	C14E2ADD6601EBDC
18	3	45	A24F31D25E456E34

Таблица 3.91 (продолжение)

Раунд	Итерация	s	c
19	0	2	AD83615AC0E7AEAE
19	1	44	81FCC39F84A54A8B
19	2	4	D15C7E21FE235136
19	3	49	5F5AC08E5A961B43
20	0	29	0CEC9543F2A66676
20	1	12	7C034EBA929A8B8E
20	2	56	C0F4CE12EC988EBB
20	3	18	5D358844AE5699F9
21	0	59	42E8D74DB4919B52
21	1	21	8250D178F5557F8A
21	2	45	532394E648E4F3FC
21	3	60	3E2BF92B03691AD8
22	0	12	FA9268E710647D5B
22	1	62	BBD56F8408E2E651
22	2	59	793C3027EB0C5B8C
22	3	51	7643D2BB11326B87
23	0	20	4B9FF22BB56211E4
23	1	42	AA39E9382F34B664
23	2	6	E212D331BFE06A72
23	3	27	1755736EA478F948
24	0	1	59CA19F718A53EAA
24	1	17	F44B30FA21C0A6ED
24	2	24	71F47E295DA0855C
24	3	51	5036E2EE9C4166B9
25	0	32	6D32721CF1269E70
25	1	4	C51E826355EC445F
25	2	26	0E8E66931EF37C41
25	3	46	9A94B3039660D3DE

**Таблица 3.91 (окончание)**

<b>Раунд</b>	<b>Итерация</b>	<i>s</i>	<i>c</i>
26	0	2	1ED158ECD9D68529
26	1	1	0ECE52DC8F1C3952
26	2	38	86A20A1FFFC847E5
26	3	12	FF1DADC90C09A612
27	0	7	B896156E08C55F6D
27	1	41	644DEA351C86F456
27	2	45	29B4B572556F360D
27	3	37	875399911A5A79D1
28	0	24	32EC6F05BC921BA5
28	1	10	CE0FB52C15C61A97
28	2	4	7F4E15212953F03D
28	3	2	873CA0565BBEC3E8
29	0	6	CDF1B94C29B3812F1
29	1	18	AAAEA6E308E92F68
29	2	9	703CCA8345EC51FC
29	3	52	4618BB1B1B33EF0C
30	0	8	F039732AAD11FE46
30	1	57	86D89114CE8DE23F
30	2	1	330AEDC7E44B8AF0
30	3	31	96D7869EDD33E500
31	0	35	F59CC3B1E9354045
31	1	33	AD3DB4F4A1AA8433
31	2	11	724ECE1C833975EA
31	3	16	98516AB5C5303E6E
32	0	6	ACF4FD043B90CCB6
32	1	13	8D8A1DA51BE5CEC1
32	2	15	11D0127B77B9427B
32	3	45	67C2DE1924CAA5ED

- что касается логических операций  $\#$ , то они выполняются в той же последовательности, что и для 128-битного алгоритма; в раундах, которые отсутствуют в 128-битном варианте, используются операции согласно следующему правилу:

$$\#_i = \#_{i \bmod 64},$$

где  $i$  — сквозной номер итерации;

- фрагменты исходного ключа используются в преобразованиях алгоритма согласно табл. 3.92 (в данном случае ключ делится на 64-битные слова).

**Таблица 3.92**

Применение	Используемый фрагмент ключа (в зависимости от его размера)		
	128 битов	192 бита	256 битов
$KS_0$	$K_1$	$K_2$	$K_3$
$KS_1$	$K_0$	$K_1$	$K_2$
$KS_2$	$K_1$	$K_0$	$K_1$
$KS_3$	$K_0$	$K_2$	$K_0$
$KF_0$	$K_0$	$K_1$	$K_2$
$KF_1$	$K_1$	$K_2$	$K_3$
$KF_2$	$K_0$	$K_2$	$K_0$
$KF_3$	$K_1$	$K_0$	$K_1$
$KR_i$	$K_{i \bmod n}$		

## Криптоанализ 128- и 256-битного вариантов алгоритма

Против 128- и 256-битных вариантов алгоритма NUSH применима та же атака, что и против 64-битного варианта [398]. Атака, предложенная китайскими учеными Вен-Лингом Ву и Денг-Гуо Фенгом, позволяет методом линейного криптоанализа вычислить, например, 256-битный ключ 128-битного варианта алгоритма при наличии  $2^{126}$  известных открытых текстов выполнением  $2^{64}$  операций шифрования. Аналогичные атаки существуют и для 256-битного варианта, и для других размеров ключей.

Поскольку данные атаки говорят о недостаточном запасе криптостойкости алгоритма NUSH, 192- и 256-битный варианты алгоритма также не вышли во второй раунд конкурса NESSIE [312].

## 3.40. Алгоритм Q

Так же, как и ряд других алгоритмов — участников конкурса NESSIE (см. разд. 2.2), например, Anubis (см. разд. 3.5) и Grand Cru (см. разд. 3.20), алгоритм Q наследует структуру и часть преобразований от алгоритма Rijndael. Соответственно, алгоритм представляет 128-битный блок данных в виде байтового массива  $4 \times 4$ , над которым и выполняются преобразования.

Алгоритм, теоретически, не ограничивает размер используемых ключей шифрования ни сверху, ни снизу. Однако в рамках конкурса NESSIE рассматривалось только три фиксированных размера ключа: 128, 192 и 256 битов.

Автор алгоритма — Лесли МакБрайд (Leslie McBride) из компании Mack One Software (США). На конкурс NESSIE была представлена версия 2.0 алгоритма Q, которая и описана далее.

Стоит отметить, что спецификация алгоритма Q [261], по сравнению с подробнейшими описаниями других участников конкурса NESSIE, например, алгоритмов NUSH (см. разд. 3.39) и SC2000 (см. разд. 3.47), недостаточно детально описывает этот алгоритм. Кроме того, некоторые эксперты (см., например, [193]) отмечают наличие противоречий в данной спецификации. Поэтому далее приведено лишь краткое описание алгоритма Q.

### Структура алгоритма

В каждом раунде алгоритма выполняются следующие операции [261]:

1. Первое наложение ключа (операция  $KA$ ). Выполняется побитовой логической операцией исключающее «или» (XOR), которая применяется к каждому биту обрабатываемого блока и соответствующему биту фрагмента расширенного ключа для данной операции.
2. Первая табличная замена  $BS1$ . Используемая здесь таблица взята в неизменном виде из алгоритма Rijndael, как и таблица замен алгоритма Grand Cru. Эта таблица приведена в разд. 3.3.
3. Второе наложение ключа (операция  $KB$ ). Аналогично  $KA$ , наложение выполняется с помощью операции XOR.
4. Вторая табличная замена  $BS2$ . Эта операция унаследована от алгоритма Serpent — см. разд. 3.48.

## Описание алгоритмов

5. Третье наложение ключа (операция  $KN$ ). Отличие этой операции от  $KA$  и  $KB$  состоит в том, что они используют один и тот же фрагмент расширенного ключа во всех раундах, тогда как подключ, используемый в операции  $KN$ , уникален для каждого раунда алгоритма.
6. Операция  $CS$  — побайтный циклический сдвиг каждого (кроме первого) столбца массива данных.  $j$ -й столбец массива сдвигается на  $j$  битов вверх.
7. Третья табличная замена  $BS3$ . Также унаследована от алгоритма Serpent.

Количество раундов  $R$  алгоритма определено в его описании [261] как 8 для обычных применений или 9 в случаях, когда требуется усиленная защита.

Общая структура алгоритма такова:

1. Сначала выполняется предварительное отбеливание, т. е. наложение на данные фрагмента ключа для предварительного отбеливания, которое выполняется абсолютно аналогично операции  $KA$ .
2. Затем выполняется  $R$  описанных выше раундов преобразований.
3. Наконец, выполняется заключительное преобразование, состоящее из последовательно выполняемых операций  $KA$ ,  $BS1$  и  $KB$ , после чего выполняется финальное отбеливание. Финальное отбеливание аналогично предварительному, за исключением того, что в нем используется другой фрагмент ключа.

Расшифровывание выполняется практически аналогично зашифровыванию, за исключением следующего:

- при расшифровывании меняются местами операции  $KN$  и  $CS$ ;
- используются инверсные таблицы замен;
- раундовые ключи используются в обратной последовательности.

## Криптоанализ алгоритма

Результаты исследований алгоритма Q в рамках конкурса NESSIE оказались весьма неутешительными: алгоритм подвержен как дифференциальному [65, 72], так и линейному криптоанализу [193]. Поэтому алгоритм Q не был выбран во второй раунд конкурса [308].

## 3.41. Алгоритм RC2

Алгоритм шифрования RC2 был разработан в конце 1980-х гг. (в различных источниках указаны 1987 [395] и 1989 [224] гг.) Рональдом Ривестом, который, в частности, разработал алгоритм RC5, рассмотренный в разд. 3.42.

Алгоритм RC2 является собственностью компании RSA Data Security [390]; при этом известен тот факт, что разработку данного алгоритма инициировала и частично спонсировала фирма Lotus, которой требовался сильный (но не являющийся широко распространенным) алгоритм шифрования для последующего использования в программной системе Lotus Notes. Причем криптостойкость алгоритма должна была быть проверена Агентством Национальной Безопасности (АНБ) США. АНБ также внесло свой вклад в разработку алгоритма, предложив некоторые детали реализации, внедренные в алгоритм Ривестом [395].

Еще одна интересная особенность состоит в том, что RC2 был разрешен к экспорту за пределы США [395]. Есть источники, в которых утверждается, что взамен на такое благоволение со стороны контролирующих органов Ривест внес в процедуру расширения ключа некие ослабляющие алгоритм действия. Автору данной книги не удалось, однако, найти об этом подробную информацию, поэтому алгоритм RC2 далее будет рассмотрен в варианте без подобных ослаблений.

Не менее интересен и тот факт, что долгое время после разработки RC2 структура данного алгоритма оставалась в секрете, но в начале 1996 г. анонимный автор опубликовал в Usenet-конференции sci.crypt (см. [320]) исходные тексты алгоритма. Причем не ясно, была ли это утечка информации из RSA Data Security или результат реверс-инжиниринга какой-либо программной реализации алгоритма RC2 [395].

Не известно, насколько долго алгоритм оставался бы в секрете, не будь этой анонимной публикации, но в марте 1998 г. Ривест полностью опубликовал алгоритм в виде RFC 2268 [327].

Алгоритм RC2 получил широкое распространение, как минимум, благодаря тому факту, что он является основным и обязательным для реализации алгоритмом шифрования согласно стандарту защиты сообщений электронной почты S/MIME (см. [142]).

## Структура алгоритма

Алгоритм RC2 шифрует данные блоками по 64 бита с использованием ключей переменного размера: от 8 до 1024 битов включительно; рекомендуемым размером ключа является 64 бита [327].

Алгоритм является сетью Фейстеля, в нем выполняются 18 раундов преобразований. Причем раунды алгоритма делятся на 2 типа: смешивающие (mix) раунды и объединяющие (mesh) раунды. Общая структура алгоритма такова:

1. Выполняются 5 смешивающих раундов.
2. Выполняется 1 объединяющий раунд.

## Описание алгоритмов

3. Выполняются 6 смещающих раундов.
4. Выполняется 1 объединяющий раунд.
5. Выполняются 5 смещающих раундов.

Структура смещающего раунда приведена на рис. 3.152. Предполагается, что шифруемый блок данных разделен на 4 16-битных слова  $R_0 \dots R_3$ , над которыми смещающий раунд в цикле по  $i$  от 0 до 3 выполняет следующие операции (составляющие показанную на рис. 3.152 функцию  $f()$ ):

$$R_i = R_i + K_j + (R_{i-1 \bmod 4} \& R_{i-2 \bmod 4}) + (\sim R_{i-1 \bmod 4} \& R_{i-3 \bmod 4}) \bmod 2^{16};$$

$$j = j + 1;$$

$$R_i = R_i \lll S_i,$$

где:

- $K_j$  — фрагмент расширенного ключа, определяемый глобальной переменной  $j$ ; данная переменная изначально равна нулю и увеличивается на 1 (как показано выше) в каждом смещающем раунде; процедура расширения ключа подробно описана далее;
- $\&$  — побитовая логическая операция «и»;
- $\sim x$  — побитовый комплемент к  $x$ ;

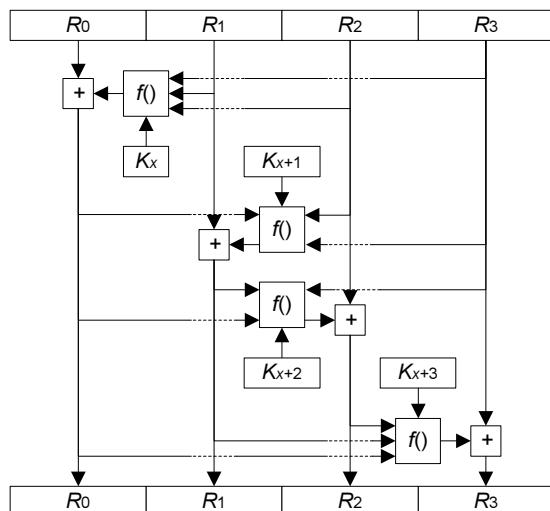


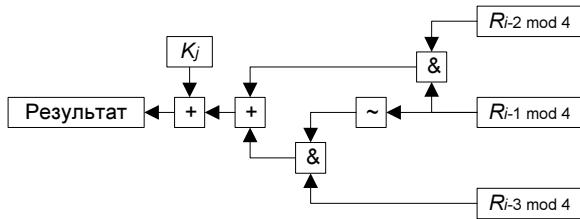
Рис. 3.152. Смещающий раунд алгоритма RC2

- <<< — циклический сдвиг влево на число битов, определяемое значением  $S_i$  (табл. 3.93).

Таблица 3.93

$i$	0	1	2	3
$S_i$	1	2	3	5

Таким образом, в каждой  $i$ -й итерации смещающего раунда выполняется описанное выше преобразование  $f()$  (рис. 3.153), которое модифицирует  $R_i$  на основе текущих значений трех остальных слов шифруемого блока и фрагмента расширенного ключа.

Рис. 3.153. Функция  $f()$  алгоритма RC2

Аналогично смещающему раунду, в объединяющем раунде выполняется цикл по  $i$  от 0 до 3; в каждой итерации цикла выполняется следующая операция:

$$R_i = R_{i-1} + K_n \bmod 2^{16},$$

где  $n = R_{i-1} \bmod 4 \& 63$ .

Таким образом, итерация объединяющего раунда представляет собой наложение операцией сложения по модулю  $2^{16}$  фрагмента расширенного ключа, индекс которого определяется 6 младшими битами текущего значения слова  $R_{i-1 \bmod 4}$ .

## Процедура расширения ключа

Как было сказано выше, алгоритм RC2 использует ключи шифрования размером от 8 до 1024 битов, т. е. от 1 до 128 байтов.

Расширение ключа подразумевает получение из ключа шифрования 16-битных фрагментов расширенного ключа  $K_0 \dots K_{63}$ , используемых в смеши-

## Описание алгоритмов

вающих раундах — по одному в каждой из 4 итераций каждого из 16 смешивающих раундов. Данная процедура выполняется в несколько шагов:

- Инициализируется байтовый массив  $L_0 \dots L_{127}$ , используемый при расширении ключа:

$$L_0 = KI_0;$$

...

$$L_{T-1} = KI_{T-1},$$

где  $K_0 \dots K_{T-1}$  — исходный ключ шифрования, имеющий размер  $T$  байтов.

Остальные байты массива  $L$  обнуляются.

- Инициализируются другие переменные, участвующие в расширении ключа:

$$T8 = (T1 + 7) / 8;$$

$$TM = 255 \bmod 2^{(8+T1-8*T8)},$$

где:

- $T1$  — размер ключа в битах;
- $T8$  — эффективный размер ключа в байтах;
- $TM$  — битовая маска, учитывающая остаточные биты ключа, если его размер в битах не кратен 8.

- В цикле по  $i$  от  $T$  до 127 выполняется следующая операция:

$$L_i = P(L_{i-1} + L_{i-T} \bmod 256),$$

где  $P$  — табличная замена, приведенная в табл. 3.94.

Таблица 3.94

d9	78	f9	c4	19	dd	b5	ed	28	e9	fd	79	4a	a0	d8	9d
c6	7e	37	83	2b	76	53	8e	62	4c	64	88	44	8b	fb	a2
17	9a	59	f5	87	b3	4f	13	61	45	6d	8d	09	81	7d	32
bd	8f	40	eb	86	b7	7b	0b	f0	95	21	22	5c	6b	4e	82
54	d6	65	93	ce	60	b2	1c	73	56	c0	14	a7	8c	f1	dc
12	75	ca	1f	3b	be	e4	d1	42	3d	d4	30	a3	3c	b6	26
6f	bf	0e	da	46	69	07	57	27	f2	1d	9b	bc	94	43	03
f8	11	c7	f6	90	ef	3e	e7	06	c3	d5	2f	c8	66	1e	d7
08	e8	ea	de	80	52	ee	f7	84	aa	72	ac	35	4d	6a	2a
96	1a	d2	71	5a	15	49	74	4b	9f	d0	5e	04	18	a4	ec

Таблица 3.94 (окончание)

c2	e0	41	6e	0f	51	cb	cc	24	91	af	50	a1	f4	70	39
99	7c	3a	85	23	b8	b4	7a	fc	02	36	5b	25	55	97	31
2d	5d	fa	98	e3	8a	92	ae	05	df	29	10	67	6c	ba	c9
d3	00	e6	cf	e1	9e	a8	2c	63	16	01	3f	58	e2	89	a9
0d	38	34	1b	ab	33	ff	b0	bb	48	0c	5f	b9	b1	cd	2e
c5	f3	db	47	e5	a5	9c	77	0a	a6	20	68	fe	7f	c1	ad

Таким образом, значение 0 заменяется на D9, 1 — на 78 и т. д.

Псевдослучайная таблица  $P$  сформирована на основе шестнадцатеричной записи дробной части числа  $\pi$ .

4. Вычисляется  $L_{128-T8}$ :

$$L_{128-T8} = P(L_{128-T8} \& TM).$$

5. В цикле по  $i$  от  $127 - T8$  до 0 выполняется следующее действие:

$$L_i = P(L_{i+1} \oplus L_{i+T8}).$$

6. Для использования в шифрующих преобразованиях 128-байтная последовательность  $L_0 \dots L_{127}$  представляется в виде 16-битных слов  $K_0 \dots K_{63}$ :

$$K_i = L_{2i} + 256 * L_{2i+1}.$$

## Расшифровывание

Расшифровывание выполняется по той же общей схеме, что и зашифровывание. Однако при расшифровывании используются другие операции, выполняемые в смешивающем и объединяющем раундах.

Смешивающий раунд расшифровывания в цикле по  $i$  от 3 до 0 выполняет следующие операции:

$$R_i = R_i >>> S_i;$$

$$R_i = R_i - K_j - (R_{i-1 \bmod 4} \& R_{i-2 \bmod 4}) - (\sim R_{i-1 \bmod 4} \& R_{i-3 \bmod 4}) \bmod 2^{16}, \\ j = j - 1,$$

где:

- $>>>$  — циклический сдвиг вправо на число битов, определяемое значением  $S_i$  (см. табл. 3.93);
- начальное значение  $j$  устанавливается не в 0, а в 63.

Аналогичным образом изменен и объединяющий раунд при расшифровывании по сравнению с зашифровыванием. В нем в цикле по  $i$  от 3 до 0 выполняется следующая операция:

$$R_i = R_i - K_n \bmod 2^{16};$$

где  $n = R_{i-1 \bmod 4} \& 63$ .

## Криптостойкость алгоритма

Почти сразу после опубликования RC2 вышла работа [224] ряда известных криптологов: Рональда Ривеста, Ларса Кнудсена, Винсента Риджмена и Мэта Робшоу, — в которой исследовалось воздействие дифференциального и линейного криptoанализа на алгоритм RC2. Результаты оказались таковы:

- алгоритм не подвержен атаке методом линейного криptoанализа;
- алгоритм может быть теоретически вскрыт методом дифференциального криptoанализа, для чего необходимо наличие не менее  $2^{59}$  пар выбранных открытых текстов и соответствующих им шифртекстов — такая атака вряд ли реализуема на практике.

Не более практической является атака на связанных ключах, для успешного проведения которой требуется наличие  $2^{34}$  выбранных открытых текстов и соответствующих им шифртекстов, причем зашифровывание должно выполняться на ключе, связанном с искомым определенным простым соотношением. Данная атака изобретена не менее известными криптологами: Джоном Келси, Брюсом Шнайером и Дэвидом Вагнером [200].

Других методов вскрытия алгоритма RC2 на настоящий момент не известно.

## 3.42. Алгоритм RC5

Алгоритм RC5 интересен по многим причинам.

- Алгоритм разработан известнейшим криптологом Рональдом Ривестом — одним из разработчиков асимметричной системы RSA и одним из основателей одноименной фирмы (RSA Data Security), которая, несомненно, является одним из мировых лидеров рынка средств криптографической защиты информации. Аббревиатура RC обозначает, по разным источникам, либо Rivest Cipher, либо Ron's Code, т. е., в совокупности, «шифр Рона Ривеста» [28, 395].
- Аналогично предыдущим алгоритмам шифрования Рона Ривеста RC2 (см. разд. 3.41) и RC4 (является потоковым шифром, поэтому в данной книге не описан), алгоритм RC5 получил весьма широкое распространение;

по количеству пользователей в мире он стоит в одном ряду с такими известными алгоритмами, как IDEA (см. разд. 3.26) и Blowfish (см. разд. 3.8).

- На преобразованиях, используемых в RC5, основана последующая разработка компании RSA — алгоритм RC6 (см. разд. 3.43), который стал финалистом конкурса AES по выбору нового стандарта шифрования США; алгоритм RC6 не победил в конкурсе, но, видимо, превзойдет своего предка по широте использования.

## Структура алгоритма

Аналогично, например, алгоритму SHARK (см. разд. 3.50), часть основных параметров алгоритма RC5 являются переменными. Как пишет автор алгоритма «RC5 — это несколько различных алгоритмов», поскольку, помимо секретного ключа, параметрами алгоритма являются следующие [328]:

- размер слова  $w$  (в битах); RC5 шифрует блоками по два слова; допустимыми значениями  $w$  являются 16, 32 или 64, причем 32 является рекомендуемым;
- количество раундов алгоритма  $R$  — в качестве значения допустимо любое целое число от 0 до 255 включительно;
- размер секретного ключа в байтах  $b$  — любое целое значение от 0 до 255 включительно.

Наиболее часто для уточнения параметров алгоритма, используемых в его конкретной реализации, применяется обозначение  $RC5-w/R/b$ ; например,  $RC5-32/12/16$  обозначает алгоритм RC5 с 64-битным блоком, 12 раундами и 128-битным (16-байтным) ключом. Такую комбинацию параметров Ривест рекомендует в качестве основного варианта алгоритма.

По мнению автора алгоритма, переменные параметры расширяют сферу использования алгоритма, а также позволяют сильно сократить издержки при необходимости перехода на более сильный вариант алгоритма — в отличие от DES (основная проблема которого — короткий 56-битный ключ), в программной или аппаратной реализации RC5, поддерживающей переменные параметры, легко было бы заменить ключ более длинным, таким образом устранив проблему. Вот что пишет об этом Рон Ривест: «Фиксированные параметры могут быть не менее опасны [переменных], поскольку они не могут быть улучшены при необходимости. Рассмотрим проблему DES: его ключ слишком короток и нет простого способа увеличить его» [328].

Автор предусмотрел и проблему совместимости реализаций RC5 с различными параметрами — каждое зашифрованное сообщение рекомендуется

предварять заголовком, содержащим список значений основных параметров алгоритма — предполагается, что в этом случае для расшифровывания сообщения следует установить параметры из заголовка, после чего (при наличии корректного ключа) сообщение легко будет расшифровано.

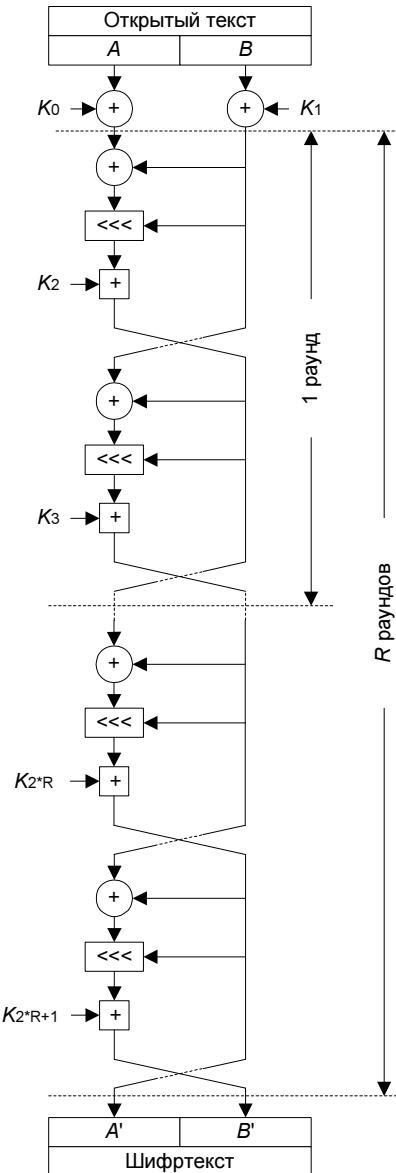


Рис. 3.154. Структура алгоритма RC5

Структура алгоритма представлена на рис. 3.154. Алгоритм представляет собой сеть Фейстеля, в каждом раунде которой выполняются следующие операции [40, 328]:

$$\begin{aligned} A &= ((A \oplus B) \lll B) + K_{2^*r} \bmod 2^w; \\ B &= ((A \oplus B) \lll A) + K_{2^*r+1} \bmod 2^w, \end{aligned}$$

где:

- $r$  — номер текущего раунда, начиная с 1;
- $K_n$  — фрагмент расширенного ключа;
- $\lll n$  — операция циклического сдвига на  $x$  битов влево, где  $x$  — значение младших  $\log_2 w$  битов  $n$ .

Перед первым раундом выполняются операции наложения двух первых фрагментов расширенного ключа на шифруемые данные:

$$\begin{aligned} A &= A + K_0 \bmod 2^w; \\ B &= B + K_1 \bmod 2^w. \end{aligned}$$

Стоит отметить, что под словом «раунд» в описании алгоритма Ривест понимает преобразования, соответствующие двум раундам обычных алгоритмов, структура которых является сетью Фейстеля (см. рис. 3.154). То есть раунд алгоритма RC5 обрабатывает блок целиком, тогда как типичный раунд сети Фейстеля обрабатывает только один субблок — обычно половину блока.

Алгоритм поразительно прост — в нем используются только операции сложения по модулю 2 и по модулю  $2^w$ , а также сдвиги на переменное число битов. Последняя из операций представляется автором алгоритма как революционное решение, не использованное в более ранних алгоритмах шифрования (до алгоритма RC5 такие использовались только в алгоритме Madryga [28], не получившем широкого распространения), — сдвиг на переменное число битов является весьма просто реализуемой операцией, которая, однако, существенно усложняет дифференциальный и линейный криптоанализ алгоритма. Простота алгоритма может рассматриваться как его важное достоинство — простой алгоритм легче реализовать и легче анализировать на предмет возможных уязвимостей.

Расшифровывание выполняется применением обратных операций в обратной последовательности, т. е. в каждом раунде  $r$  (с обратной последовательностью раундов) выполняются следующие операции:

$$\begin{aligned} B &= ((B - K_{2^*r+1} \bmod 2^w) \ggg A) \oplus A; \\ A &= ((A - K_{2^*r} \bmod 2^w) \ggg B) \oplus B, \end{aligned}$$

где  $\ggg n$  — аналогичная описанной выше ( $\lll n$ ) операция побитового циклического сдвига вправо.

Соответственно, после  $R$  раундов выполняются следующие операции:

$$B = B - K_1 \bmod 2^w;$$

$$A = A - K_0 \bmod 2^w.$$

Алгоритм RC5 и некоторые его варианты являются запатентованными. Патенты принадлежат фирме RSA Data Security [40].

## Процедура расширения ключа

Процедура расширения ключа незначительно сложнее собственно шифрования. Опишем эту процедуру:

1. Производится выравнивание ключа шифрования, в рамках которого ключ шифрования, если его размер в байтах  $b$  не кратен  $w/8$  (т. е. размеру слова в байтах), дополняется нулевыми байтами до ближайшего большего размера  $c$ , кратного  $w/8$ .
2. Инициализация массива расширенных ключей  $K_0 \dots K_{2*R+1}$  производится следующим образом:

$$K_0 = P_w;$$

$$K_{i+1} = K_i + Q_w,$$

где  $P_w$  и  $Q_w$  — псевдослучайные константы, образованные путем умножения на  $2^w$  дробной части и последующего округления до ближайшего нечетного целого двух математических констант (е и ф соответственно). В спецификации алгоритма приведены вычисленные константы для возможных значений  $w$  (указаны шестнадцатеричные значения):

$$P_{16} = \text{B7E1};$$

$$Q_{16} = \text{9E37};$$

$$P_{32} = \text{B7E15163};$$

$$Q_{32} = \text{9E3779B9};$$

$$P_{64} = \text{B7E151628AED2A6B};$$

$$Q_{64} = \text{9E3779B97F4A7C15}.$$

3. Циклически выполняются следующие действия:

$$A = K_i = (K_i + A + B) \lll 3;$$

$$B = KC_j = (KC_j + A + B) \lll (A + B);$$

$$i = i + 1 \bmod (2 * R + 1);$$

$$j = j + 1 \bmod c,$$

где:

- $i, j, A$  и  $B$  — временные переменные, их начальные значения равны нулю;
- $KC$  — выровненный на этапе 1 ключ шифрования.

Количество итераций цикла  $N$  определяется как  $N = 3 * m$ , где  $m$  — максимальное из двух значений:  $c$  или  $(2 * R + 1)$ .

## Криптоанализ алгоритма

Считается, что именно революционные сдвиги на переменное число битов привлекли внимание криптоаналитиков к алгоритму RC5 — RC5 стал одним из наиболее изученных алгоритмов на предмет возможных уязвимостей.

Начало криптоанализу алгоритма RC5 было положено сотрудниками RSA Laboratories — научного подразделения фирмы RSA Data Security — Бертоном Калиски (Burton S. Kaliski Jr.) и Икван Лайзой Ин (Yiqun Lisa Yin). В период с 1995 по 1998 гг. они опубликовали ряд отчетов (например, [190]), в которых подробно проанализировали криптостойкость алгоритма RC5. Выводы таковы.

- Алгоритм RC5 почти невозможно вскрыть методом линейного криптоанализа. Во многом это свойство алгоритма предопределило наличие операции циклического сдвига на переменное число битов. Однако дальнейшие исследования показали, что существует класс ключей, при использовании которых алгоритм может быть вскрыт линейным криптоанализом [179].
- Дифференциальный криптоанализ существенно более эффективен при атаках на алгоритм RC5. Калиски и Ин предложили атаку на алгоритм RC5-32/12/16, для которой требовалось наличие  $2^{63}$  пар выбранных открытых текстов и соответствующих им шифртекстов. Этот результат улучшили Ларс Кнудсен и Уилли Мейер (Willi Meier), для атаки которых требовалось  $2^{54}$  выбранных открытых текстов [218]. Они же нашли несколько классов слабых ключей, использование которых упрощает дифференциальный криптоанализ. А наилучшим результатом является криптоаналитический метод, предложенный криптологами Алексом Бирюковым и Эйялом Кушилевичем (Eyal Kushilevitz), которым необходимо  $2^{44}$  выбранных открытых текстов для успешной атаки [90]. Тем не менее, все описанные выше атаки являются непрактичными — для их выполнения требуется наличие огромного количества выбранных открытых текстов. Бирюков и Кушилевич считают, что для обеспечения полной невскрываемости алгоритма дифференциальным криптоанализом достаточно выполнения 18–20 раундов вместо 12.

□ На основании того факта, что на ряде платформ операция циклического сдвига на переменное число битов выполняется за различное число тактов процессора, изобретатель метода вскрытия алгоритмов шифрования по времени исполнения Пол Кохер высказал предположение о возможности атаки по времени исполнения на алгоритм RC5 на таких plataформах [228]. Два варианта подобной атаки были сформулированы криптоаналистами Говардом Хейзом (Howard M. Heys) и Хеленой Хандшух (Helena Handschuh), которые показали, что секретный ключ можно вычислить путем выполнения около  $2^{20}$  операций шифрования с высокоточными замерами времени исполнения и последующим выполнением от  $2^{28}$  до  $2^{40}$  тестовых операций шифрования [171, 179]. Однако Калиски и Ин предложили весьма простое «противоядие» против этой атаки — принудительно выполнять все сдвиги за одинаковое число тактов (т. е. как наиболее медленный из возможных сдвигов — это, несомненно, несколько снижает среднюю скорость шифрования) [190]. Аналогичную методику противодействия атакам по времени исполнения советует и сам Кохер [228].

Таким образом, наиболее реальным методом взлома алгоритма RC5 (не считая варианты с небольшим количеством раундов и с коротким ключом) является полный перебор возможных вариантов ключа шифрования. Что означает, что у алгоритма RC5 практически отсутствуют недостатки с точки зрения его стойкости. Это косвенно подтверждается тем, что достаточно много исследований стойкости алгоритма было направлено против вариантов с усеченным количеством раундов (например, [89, 90, 218]): такие варианты обычно исследуются в случае отсутствия серьезных уязвимостей у полноценных вариантов исследуемого алгоритма.

Было немало и других исследований данного алгоритма. Причем их подавляющее большинство применялось к 64-битной версии алгоритма (т. е. для  $w = 32$ ). Это не означает, что RC5 со 128-битным блоком шифруемых данных является существенно менее изученным — результаты исследований показывают, что 128-битный вариант RC5 с достаточным количеством раундов вскрыть существенно сложнее 64-битного. Например, Бирюков и Кушилевиц предложили атаку на алгоритм RC5-64/16/16 на основе  $2^{63}$  выбранных открытых текстов, что достаточно нереально для практического применения [90].

## Варианты RC5

Структура алгоритма RC5, несмотря на свою простоту, представлялась многим криптологам как поле для возможных усовершенствований. Соответственно, появилось множество известных вариантов алгоритма RC5,

в которых преобразования в «пол-раундах» классического RC5 несколько изменены [190]:

- Алгоритм RC5XOR, в котором сложение с ключом раунда по модулю  $2^w$  заменено операцией XOR (рис. 3.155)<sup>1</sup>:

$$A = ((A \oplus B) \lll B) \oplus K_{2^*r}.$$

Данный алгоритм оказался менее стоеч, чем RC5, как к линейному, так и к дифференциальному криптоанализу. В частности, Бирюков и Кушлевиц предложили атаку методом дифференциального криптоанализа, вскрывающую алгоритм RC5XOR-32/12/16 на основе  $2^{28}$  выбранных открытых текстов [90].

- Алгоритм RC5P, в котором сложение левого и правого обрабатываемых субблоков операцией XOR заменено сложением по модулю  $2^w$  (рис. 3.156):

$$A = ((A + B \bmod 2^w) \lll B) + K_{2^*r} \bmod 2^w.$$

Алгоритм оказался так же стоеч, как и RC5, против линейного криптоанализа, но значительно слабее против дифференциального [190, 199].

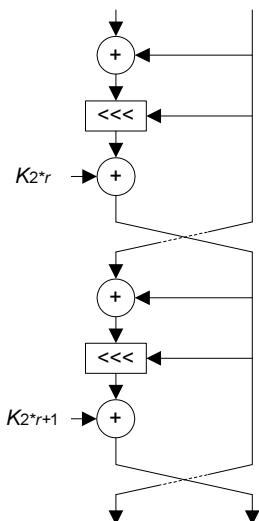


Рис. 3.155. Раунд алгоритма RC5XOR

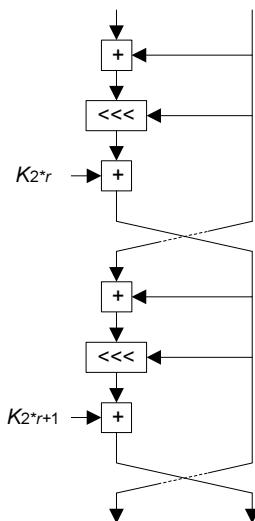


Рис. 3.156. Раунд алгоритма RC5P

<sup>1</sup> Здесь и далее в качестве примера приведено только преобразование для вычисления левого субблока; правый вычисляется в следующей половине раунда аналогичным образом (см. рисунки).

- Алгоритм RC5PFR, отличающийся от RC5 циклическим сдвигом на фиксированное, а не переменное число битов (рис. 3.157):

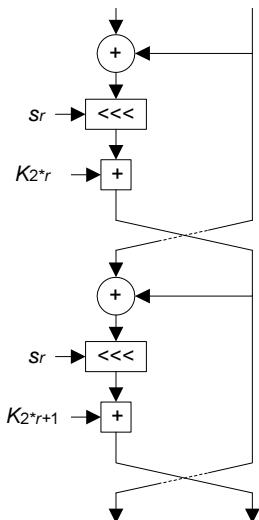
$$A = ((A \oplus B) \ll s_r) + K_{2^*r} \bmod 2^w,$$

где  $s_r$  — число битов циклического сдвига, которое может быть различным в различных раундах алгоритма; в этом случае последовательность  $s_1 \dots s_R$  является дополнительным параметром алгоритма.

Данный вариант алгоритма RC5 не является хорошо изученным, однако эксперты предполагают, что алгоритм RC5PFR нестойек против дифференциального криптоанализа [190].

- Алгоритм RC5KFR, в котором число битов сдвига является функцией ключа шифрования  $KC$ , т. е. для каждого ключа шифрования число битов сдвига является фиксированным (может быть различным для разных раундов алгоритма) (рис. 3.158):

$$A = ((A \oplus B) \ll s_r(KC)) + K_{2^*r} \bmod 2^w.$$



- Алгоритм RC5RA, в котором выполняется циклический сдвиг на переменное число битов, определяемое не значением младших  $\log_2 w$  битов другого субблока, а некоей функцией  $f()$ , обрабатывающей в качестве входного значения все биты другого субблока (рис. 3.159):

$$A = ((A \oplus B) \lll f(B)) + K_{2^r} \bmod 2^w.$$

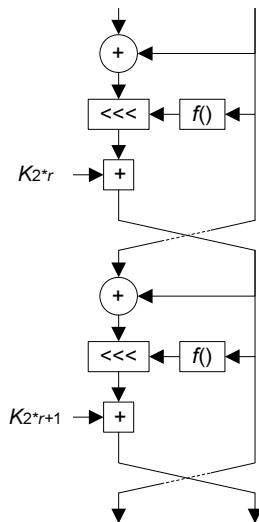


Рис. 3.159. Раунд алгоритма RC5RA

И этот вариант алгоритма еще недостаточно изучен (и, видимо, уже не будет достаточно изучен, поскольку можно утверждать, что RC5 и его варианты сейчас представляют лишь исторический интерес), но существует мнение, что алгоритм RC5RA может быть еще сильнее, чем RC5, против известных методов криптоанализа [190].

## Продолжение истории алгоритма

Как было сказано выше, на основе алгоритма RC5 в 1998 г. был разработан алгоритм RC6, который также основан на циклических сдвигах на переменное число битов. Подавляющее большинство криптоаналитических исследований алгоритма RC5 могут быть в различной мере применены и к RC6. У RC6 уже своя богатая история, которая заслуживает отдельного описания.

## 3.43. Алгоритм RC6

Алгоритм RC6 был разработан в 1998 г. рядом специалистов научного подразделения известнейшей фирмы RSA Data Security — RSA Laboratories: Рональдом Ривестом (основатель RSA Data Security), Мэттом Робшоу, Рэем Сидни (Ray Sidney) и Икван Лайзой Ин специально для участия в конкурсе AES [389]. Алгоритм во многом унаследовал черты предыдущего алгоритма Рональда Ривеста — 64-битного блочного шифра RC5 (*см. разд. 3.42*), разработанного в 1997 г. [328]. Фактически алгоритм претерпел два принципиальных изменения:

- в отличие от RC5, в алгоритме используется умножение только по модулю  $2^{32}$ ;
- для сохранения 32-битных вычислений вместо разбиения шифруемого блока данных (128 битов согласно принципиальному требованию конкурса AES) на два 64-битных субблока выполняется его разбиение на 4 32-битных субблока и их обработка по несколько измененной схеме [389].

### Структура алгоритма

Как и RC5, алгоритм RC6 имеет гибкую структуру: помимо секретного ключа, параметрами являются:

- размер слова  $w$ ; алгоритм RC6 шифрует блоками по 4 слова;
- количество раундов  $R$ ;
- размер секретного ключа в байтах  $b$ .

Аналогично RC5, для уточнения параметров алгоритма, используемых в его конкретной реализации, применяется обозначение  $RC6-w/R/b$ . Поскольку в конкурсе AES 128-битный блок являлся обязательным, значение  $w$  фиксировано и равно 32. В спецификации алгоритма [329] зафиксировано также количество раундов:  $R = 20$ . Поскольку конкурс AES допускал использование ключей трех фиксированных размеров, рассмотрим три следующих варианта алгоритма:  $RC6-32/20/16$ ,  $RC6-32/20/24$  и  $RC6-32/20/32$ , совокупность которых мы и будем далее иметь в виду, говоря об «RC6».

Структура алгоритма представлена на рис. 3.160. Как было сказано выше, в алгоритме используется 20 раундов преобразований, перед которыми выполняется частичное входное отбеливание:

$$\begin{aligned} B &= B + K_0 \bmod 2^{32}; \\ D &= D + K_1 \bmod 2^{32}, \end{aligned}$$

где:

- $B$  и  $D$  — текущие значения двух из четырех ( $A$ ,  $B$ ,  $C$ ,  $D$ ) обрабатываемых 32-битных субблоков;

- $K_0$  и  $K_1$  — первые два из используемых в алгоритме фрагментов расширенного ключа  $K_0 \dots K_{43}$ .

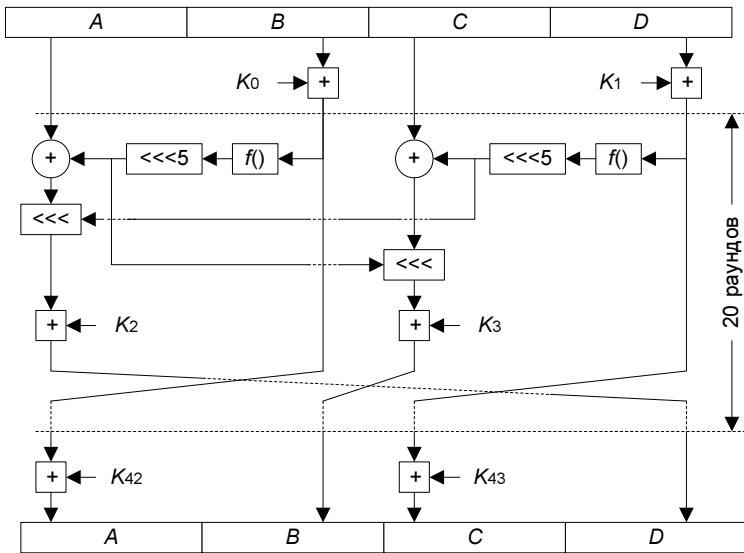


Рис. 3.160. Структура алгоритма RC6

Аналогичным образом выполняется частичное выходное отбеливание:

$$A = A + K_{42} \bmod 2^{32};$$

$$C = C + K_{43} \bmod 2^{32}.$$

В каждом  $i$ -м раунде алгоритма выполняются следующие действия:

$$t_1 = f(B) \lll 5;$$

$$t_2 = f(D) \lll 5;$$

$$A = ((A \oplus t_1) \lll t_2) + K_{2i} \bmod 2^{32};$$

$$C = ((C \oplus t_2) \lll t_1) + K_{2i+1} \bmod 2^{32},$$

где:

- $t_1$  и  $t_2$  — временные переменные;
- количество битов вращения на переменное число битов определяется значением 5 младших битов параметра ( $t_1$  или  $t_2$ );
- функция  $f()$  выполняет следующее квадратичное преобразование:

$$f(x) = x * (2x + 1) \bmod 2^{32}.$$

В конце каждого раунда выполняется сдвиг субблоков — см. рис. 3.160.

При расшифровывании подключи используются в обратном порядке, наложение подключей вместо сложения по модулю  $2^{32}$  выполняется вычитанием, а также сдвиг субблоков выполняется в начале раунда и в обратную сторону. Преобразование  $f()$  не претерпело изменений (рис. 3.161).

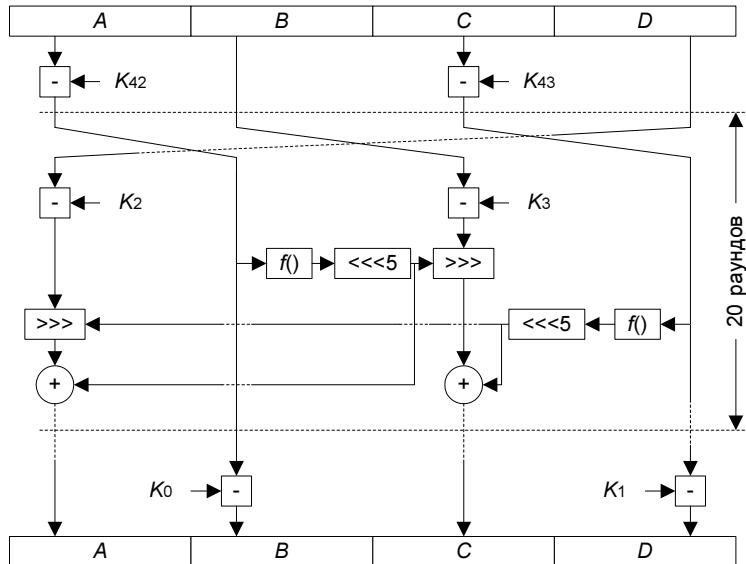


Рис. 3.161. Расшифровывание алгоритмом RC6

## Процедура расширения ключа

Процедура расширения ключа алгоритма RC6 аналогична RC5, за исключением того, что алгоритму RC6 требуется несколько больше генерированных подключей, а именно  $2R + 4$ , т. е.  $K_0 \dots K_{43}$  для 20 раундов. Рассмотрим данную процедуру для алгоритма RC6 в варианте для конкурса AES, т. е. с указанными выше фиксированными параметрами.

Расширение ключа выполняется в два этапа:

- Инициализация массива расширенных ключей  $K_0 \dots K_{43}$  производится следующим образом:

$$K_0 = P_{32};$$

$$K_{i+1} = K_i + Q_{32},$$

где  $P_{32}$  и  $Q_{32}$  — псевдослучайные константы, образованные путем умножения на  $2^{32}$  дробной части и последующего округления до ближайшего нечетного целого двух математических констант ( $e$  и  $\phi$  соответственно):

$$P_{32} = \text{B7E15163};$$

$$Q_{32} = \text{9E3779B9}.$$

Авторы алгоритма в его спецификации [329] утверждают, что выбор данных значений не является важным. Соответственно, аналогично, например, алгоритму Twofish-FK (см. разд. 3.56), при необходимости создания реализации алгоритма RC6, не совместимой со стандартной, следует изменить значения  $P_{32}$  и  $Q_{32}$ .

2. Циклически выполняются следующие действия:

$$A = K_i = (K_i + A + B) \lll 3;$$

$$B = KI_j = (KI_j + A + B) \lll (A + B);$$

$$i = i + 1 \bmod 44;$$

$$j = j + 1 \bmod c,$$

где:

- $i, j, A$  и  $B$  — временные переменные, их начальные значения равны нулю;
- $KI$  — исходный ключ шифрования, представленный в виде  $c$  32-битных слов.

Выполняются  $3c$  итераций цикла.

## Достоинства и недостатки алгоритма

Сравнительный анализ достоинств и недостатков алгоритмов — финалистов конкурса AES приведен в разд. 2.1.

### 3.44. Алгоритмы SAFER K и SAFER SK

Алгоритм блочного симметричного шифрования SAFER K-64 был разработан в 1993 г. известным криптологом Джеймсом Мэсси из Технологического Института г. Цюрих, Швейцария, для американской корпорации Cylink. Название алгоритма — аббревиатура от Secure and Fast Encryption Routine, т. е. «сильный и быстрый алгоритм шифрования»; «K-64» в названии алгоритма обозначает использование 64-битного ключа шифрования [251].

## Структура алгоритма SAFER K-64

SAFER K-64 шифрует данные 64-битными блоками. Блок данных представляется в виде 8-байтового массива (байты нумеруются слева направо от 1 до 8). Алгоритм представляет собой подстановочно-перестановочную сеть (см. разд. 1.3), в каждом раунде которой выполняются следующие преобразования (рис. 3.162) [251]:

- Первое наложение ключа: на данные накладывается 8-байтный фрагмент расширенного ключа  $K_{2i-1}$ , где  $i$  — номер текущего раунда (раунды нумеруются, начиная с 1; процедура расширения ключа будет описана далее). Байты ключа №№ 1, 4, 5 и 8 накладываются на аналогичные байты данных с помощью побитовой логической операции «исключающее или» (XOR); для наложения остальных байтов ключа используется операция сложения по модулю 256.

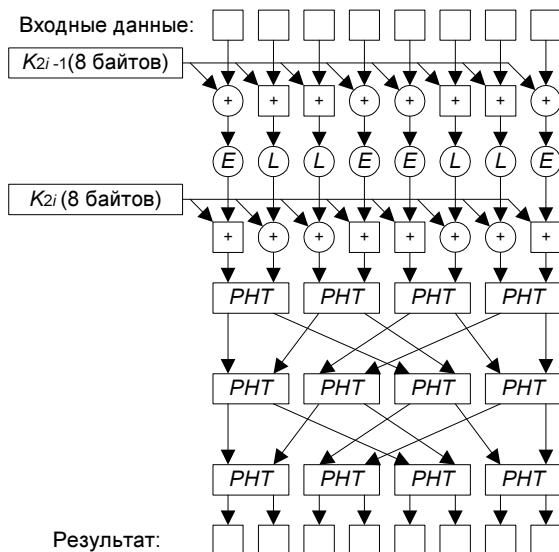


Рис. 3.162. Раунд алгоритма SAFER K-64

- Нелинейное преобразование:

- байты данных №№ 1, 4, 5 и 8 обрабатываются следующей операцией (на рис. 3.162 обозначена как  $E$ ):

$$y = 45^x \bmod 257,$$

где  $x$  и  $y$  — соответственно, входное и выходное значения этой операции; причем если  $y = 256$  (что происходит при  $x = 128$ ), то  $y$  обнуляется;

- остальные байты обрабатываются обратной операцией  $L$ :

$$y = \log_{45} x \bmod 257,$$

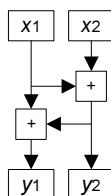
причем для  $x = 0$  значение  $y$  устанавливается в 128.

3. Второе наложение ключа выполняется аналогично первому (однако с использованием другого фрагмента расширенного ключа —  $K_{2i}$ ), но с инверсным применением операций: байты, которые в первом наложении ключа обрабатывались операцией XOR, здесь обрабатываются сложением по модулю 256, и наоборот.
4. Три уровня преобразований *PHT* (Pseudo Hadamard Transform, псевдоадамарово преобразование); операция *PHT* выполняет несложное преобразование двух входных байтов данных ( $x_1$  и  $x_2$ ), давая в результате два выходных байта  $y_1$  и  $y_2$ , и определяется следующим образом:

$$y_1 = 2x_1 + x_2 \bmod 256;$$

$$y_2 = x_1 + x_2 \bmod 256,$$

что схематически представлено на рис. 3.163.



**Рис. 3.163.** Преобразование *PHT*

Между уровнями преобразований *PHT* выполняется перестановка байтов данных по следующему закону (табл. 3.95).

**Таблица 3.95**

1	3	5	7	2	4	6	8
---	---	---	---	---	---	---	---

Это означает, что байт № 1 остается на своем месте, входное значение байта № 3 становится выходным значением байта № 2 и т. д.

Автор алгоритма рекомендует выполнять 6 раундов описанных выше преобразований (количество раундов алгоритма обозначим как  $R$ ), но допускает увеличение количества раундов до десяти.

После выполнения  $R$  раундов алгоритма применяется выходное преобразование, полностью аналогичное описанной выше операции первого наложения

ключа; в данном случае применяется последний из фрагментов расширенного ключа  $K_{2R+1}$ .

Следует учесть, что в зависимости от требований к реализации алгоритма описанные выше операции  $E$  и  $L$  могут быть реализованы напрямую или в виде таблиц замен. Таблица замен для  $E$  приведена в табл. 3.96 [339].

Таблица 3.96

1	45	226	147	190	69	21	174	120	3	135	164	184	56	207	63
8	103	9	148	235	38	168	107	189	24	52	27	187	191	114	247
64	53	72	156	81	47	59	85	227	192	159	216	211	243	141	177
255	167	62	220	134	119	215	166	17	251	244	186	146	145	100	131
241	51	239	218	44	181	178	43	136	209	153	203	140	132	29	20
129	151	113	202	95	163	139	87	60	130	196	82	92	28	232	160
4	180	133	74	246	19	84	182	223	12	26	142	222	224	57	252
32	155	36	78	169	152	158	171	242	96	208	108	234	250	199	217
0	212	31	110	67	188	236	83	137	254	122	93	73	201	50	194
249	154	248	109	22	219	89	150	68	233	205	230	70	66	143	10
193	204	185	101	176	210	198	172	30	65	98	41	46	14	116	80
2	90	195	37	123	138	42	91	240	6	13	71	111	112	157	126
16	206	18	39	213	76	79	214	121	48	104	54	117	125	228	237
128	106	144	55	162	94	118	170	197	127	61	175	165	229	25	97
253	77	124	183	11	238	173	75	34	245	231	115	35	33	200	5
225	102	221	179	88	105	99	86	15	161	49	149	23	7	58	40

Согласно таблице значение 0 заменяется значением 1, значение 1 — значениям 45 и т. д.

Таблица замен для  $L$  представлена в табл. 3.97 [339].

*Таблица 3.97*

128	0	176	9	96	239	185	253	16	18	159	228	105	186	173	248
192	56	194	101	79	6	148	252	25	222	106	27	93	78	168	130
112	237	232	236	114	179	21	195	255	171	182	71	68	1	172	37
201	250	142	65	26	33	203	211	13	110	254	38	88	218	50	15
32	169	157	132	152	5	156	187	34	140	99	231	197	225	115	198
175	36	91	135	102	39	247	87	244	150	177	183	92	139	213	84
121	223	170	246	62	163	241	17	202	245	209	23	123	147	131	188
189	82	30	235	174	204	214	53	8	200	138	180	226	205	191	217
208	80	89	63	77	98	52	10	72	136	181	86	76	46	107	158
210	61	60	3	19	251	151	81	117	74	145	113	35	190	118	42
95	249	212	85	11	220	55	49	22	116	215	119	167	230	7	219
164	47	70	243	97	69	103	227	12	162	59	28	133	24	4	29
41	160	143	178	90	216	166	126	238	141	83	75	161	154	193	14
122	73	165	44	129	196	199	54	43	127	67	149	51	242	108	104
109	240	2	40	206	221	155	234	94	153	124	20	134	207	229	66
184	64	120	45	58	233	100	31	146	144	125	57	111	224	137	48

Преобразования РНТ в совокупности также могут быть заменены умножением байтового массива данных на матрицу  $M$  (в конечном поле GF(256)) [273, 281] (табл. 3.98).

**Таблица 3.98**

## Расшифровывание

При расшифровывании шифртекст, прежде всего, подвергается входному преобразованию с участием фрагмента расширенного ключа  $K_{2R+1}$ , которое инверсно выходному преобразованию при зашифровывании:

- байты №№ 1, 4, 5 и 8 обрабатываются операцией XOR;
- остальные байты обрабатываются операцией вычитания:

$$y = x - k \bmod 256,$$

где  $k$  — соответствующий байт накладываемого фрагмента ключа.

После этого выполняется  $R$  раундов расшифровывания, в каждом из которых производятся следующие действия (рис. 3.164):

1. Три уровня обратного псевдоадамарового преобразования  $IPHT$  (Inverse PHT), которое определяется таким образом:

$$y_1 = x_1 - x_2 \bmod 256;$$

$$y_2 = -x_1 + 2x_2 \bmod 256.$$

Между уровнями преобразований  $IPHT$  выполняется перестановка байтов, обратная перестановке, которая использовалась при зашифровывании (табл. 3.99).

Таблица 3.99

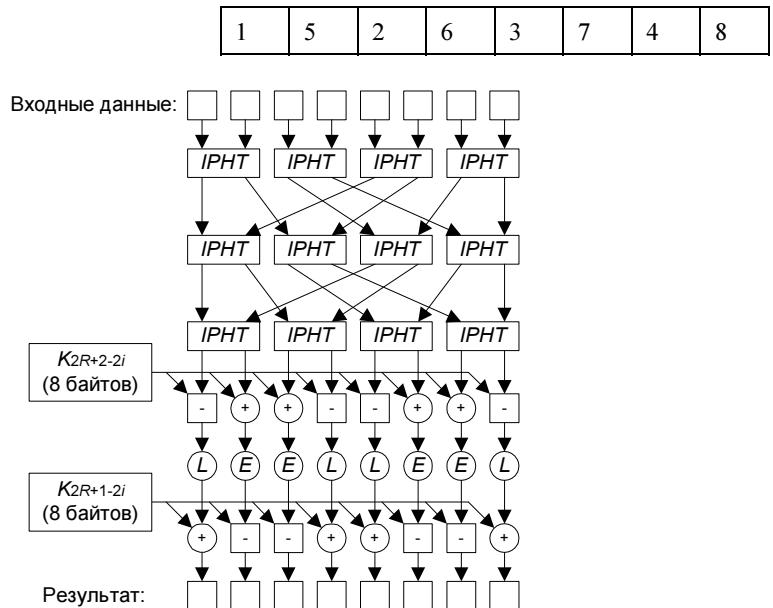


Рис. 3.164. Раунд расшифровывания алгоритма SAFER K-64

2. Обратное второе наложение ключа: на данные накладывается 8-байтный фрагмент расширенного ключа  $K_{2R+2-2i}$ . Байты ключа №№ 2, 3, 6 и 7 накладываются на аналогичные байты данных с помощью операции XOR; для наложения остальных байтов ключа используется описанная выше операция вычитания.
3. Обратное нелинейное преобразование:
  - байты данных №№ 1, 4, 5 и 8 обрабатываются операцией  $L$ ;
  - остальные байты обрабатываются операцией  $E$ .
4. Обратное первое наложение ключа, полностью аналогичное описанному выше входному преобразованию; здесь используется фрагмент расширенного ключа  $K_{2R+1-2i}$ .

Как видно, при расшифровывании происходит выполнение обратных операций в обратном порядке.

Преобразование *IPHT* также может быть заменено умножением на матрицу  $M^{-1}$  (аналогично описанному выше для *PHT*) [281] (табл. 3.100).

**Таблица 3.100**

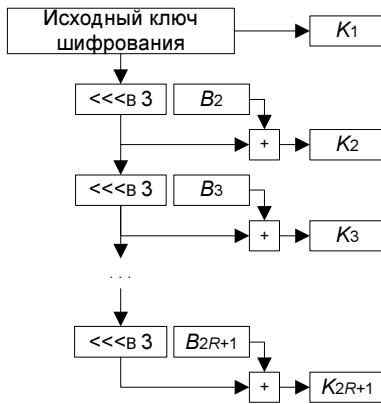
1	-1	-1	1	-1	1	1	-1
-1	1	1	-1	2	-2	-2	2
-1	2	1	-2	1	-2	-1	2
1	-2	-1	2	-2	4	2	-4
-1	1	2	-2	1	-1	-2	2
1	-1	-2	2	-2	2	4	-4
1	-2	-2	4	-1	2	2	-4
-1	2	2	-4	2	-4	-4	8

## Процедура расширения ключа

Процедура расширения ключа алгоритма SAFER K-64 достаточно проста. Она состоит из  $2R+1$  раундов, в каждом из которых вычисляется один из 8-байтных фрагментов расширенного ключа. Раунд процедуры расширения ключа состоит из следующих операций (рис. 3.165):

1. Каждый байт результата шага 1 предыдущего раунда вращается влево на 3 бита (на рис. 3.165 эта операция обозначена как  $<<<_B$ ). В первом раунде процедуры расширения ключа вместо результата шага 1 предыдущего раунда используется исходный 8-байтный ключ шифрования.

## Описание алгоритмов



**Рис. 3.165.** Процедура расширения ключа алгоритма SAFER K-64

2. Результат предыдущего шага складывается побайтно по модулю 256 с константой  $B_i$  (где  $i$  — номер раунда процедуры расширения ключа,  $i = 2\dots 2R + 1$ ). Константы  $B_i$  определяются с помощью описанной выше операции  $E$ :

$$B_i[j] = E(E(9i + j)),$$

где  $B_i[j]$  —  $j$ -й байт константы  $B_i$ . Стоит отметить, что константы  $B_i$  могут как вычисляться в процессе расширения ключа, так и быть заданы в виде набора констант (для 10 раундов шифрования, начиная с  $B_2$ ) [339] (табл. 3.101).

**Таблица 3.101**

16733B1E8E70BD86	FDFB1740E6511D41
477E2456F1778846	8F29DD0480DEE731
B1BAA3B7100AC537	7F01A2F739DA6F23
C95A28AC64A5ECAB	FE3AD01CD1303E12
C66795580DF89AF6	CD0FE0A8AF82592C
66DC053DD38AC3D8	7DADB2EFC287CE75
6AE9364943BFEBD4	1302904F2E723385
9B68A0655D57921F	8DCFA981E2C4272F
715CBB22C1BE7BBC	7A9F52E115382BFC
63945F2A61B83432	42C708E409555E8C

Фрагмент расширенного ключа  $K_1$  эквивалентен исходному ключу шифрования без каких-либо изменений.

Несомненное достоинство процедуры расширения ключа состоит в том, что фрагменты расширенного ключа могут вычисляться «на лету», т. е. в процессе зашифровывания по мере необходимости.

## Криптоанализ алгоритма SAFER K-64

Криптоанализ алгоритма SAFER K-64 был начат Ларсом Кнудсеном в 1995 г. Он предложил атаку на связанных ключах [213]: для почти каждого ключа  $K$  существует ключ  $K'$  (отличающийся от  $K$  значением одного байта), такой, что для достаточно большого множества открытых текстов (до  $2^{28}$  из возможных  $2^{64}$ ) после 6 раундов шифрования результаты шифрования каждого из текстов на ключах  $K$  и  $K'$  абсолютно эквивалентны. Финальное преобразование делает результаты шифрования различными, но только в одном байте шифртекста. Данная особенность позволяет вычислить 8 битов 64-битного ключа шифрования алгоритма SAFER K-64 на основе от  $2^{44}$  до  $2^{47}$  выбранных открытых текстов. Найденная Кнудсеном атака не снижает практической криптостойкости алгоритма. Однако, согласно [213], в случае, если SAFER K-64 используется в режиме хэширования, то для нахождения коллизии (двух текстов с одинаковым хэш-значением) требуется  $2^{23}$  операций шифрования вместо  $2^{32}$  (последнее следует из парадокса дней рождения), что делает весьма сомнительной возможность использования данного алгоритма в качестве основы хэш-функций. Вывод автора [213] состоит в том, что для достижения достаточной криптостойкости алгоритму SAFER K-64 недостаточно раундов: необходимо, как минимум, 10 вместо 6.

Данная атака была усиlena Джоном Келси, Брюсом Шнайером и Дэвидом Вагнером [197]: их вариант атаки для нахождения с высокой вероятностью 28 битов ключа требует от  $2^{24}$  до  $2^{29}$  выбранных открытых текстов (и их зашифровывание на 256 связанных ключах). Как пишут авторы [197], алгоритм SAFER K-64 подвержен атакам на связанных ключах из-за его достаточно простой и «однообразной» (генерация подключей происходит с минимальными отличиями) процедуры расширения ключа.

Еще одна слабость процедуры расширения ключа была описана в работе [273]. Описанный далее алгоритм SAFER SK-64 противостоит приведенным выше атакам.

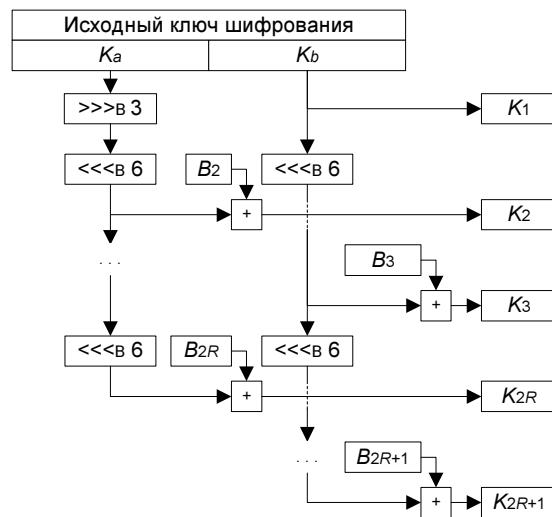
В работах [216] и [217] описана атака на 5-раундовый (из 6 раундов) SAFER K-64, для которой требуется  $2^{45}$  выбранных открытых текстов и  $2^{46}$  операций шифрования или  $2^{46}$  выбранных открытых текстов и  $2^{35}$  операций.

## Алгоритм SAFER K-128

В 1995 г. автор алгоритма SAFER K-64 в [252] писал (устный доклад на эту тему был сделан автором на конференции в декабре 1994 г.), что данный алгоритм был весьма положительно принят в мире и реализован в множестве различных средств. Поэтому почти сразу после презентации алгоритма SAFER K-64 Джеймс Мэсси начал разработку алгоритма SAFER K-128, который шифрует с использованием уже 128-битного ключа шифрования.

SAFER K-128 отличается от SAFER K-64 только процедурой расширения ключа, которая была предложена автору алгоритма специалистами МВД Сингапура [252]. Процедура расширения ключа алгоритма SAFER K-128 приведена на рис. 3.166 [356]. Ее отличие от исходной легко видно в том, что фрагменты расширенного ключа формируются параллельно из двух 64-битных половин 128-битного исходного ключа шифрования:

- правая половина ключа ( $K_b$ ) является первым фрагментом расширенного ключа ( $K_1$ ) и формирует все остальные фрагменты с нечетными индексами;
- левая половина ключа ( $K_a$ ) формирует все фрагменты расширенного ключа с четными индексами.



в SAFER K-64: 10 раундов рекомендует автор алгоритма в [252], максимальное количество раундов — 12.

Там же, в [252], Джеймс Мэсси приводит ряд соображений касательно описанной выше атаки Ларса Кнудсена [213]:

- в части нахождения коллизий за  $2^{23}$  вместо  $2^{32}$  операций атака применима и к алгоритму SAFER K-128;
- автор алгоритма считает, что применение алгоритма блочного шифрования в качестве основы хэш-функции происходит достаточно редко, но в этом случае достаточно увеличить количество раундов алгоритма SAFER K-64 с 6 до, как минимум, 10; каких-либо более серьезных модификаций алгоритма не требуется.

## Алгоритмы SAFER SK-64, SAFER SK-128 и SAFER SK-40

Усиление процедуры расширения ключа алгоритма SAFER K-64 против изобретенной им же атаки предложил в [213] Ларс Кнудсен. В сообщении на конференции sci.crypt в 1995 г. [250] Мэсси предложил два усовершенствованных варианта алгоритма SAFER: SAFER SK-64 и SAFER SK-128.

По сравнению с алгоритмом SAFER K-64 в них изменилась только процедура расширения ключа, которая была усилена именно в соответствии с предложениями Ларса Кнудсена.

Процедура расширения ключа алгоритма SAFER SK-64 приведена на рис. 3.167. Ее отличия от исходной состоят в следующем:

- исходный 64-битный ключ шифрования разбивается на 8 байтов, к которым добавляется еще один, полученный в результате применения побитовой операции XOR к соответствующим битам 8 байтов ключа;
- в каждом раунде процедуры расширения ключа выполняется сдвиг этих 9 байтов на 1 байт влево;
- в результате в формировании фрагментов расширенного ключа в каждом раунде процедуры участвуют различные байты ключа (включая дополнительный 9-й байт):  $K_1$  формируется полностью аналогично алгоритму SAFER K-64 из байтов исходного ключа,  $K_2$  — из байтов со 2-го по 9-й,  $K_3$  — из байтов с 3-го по 9-й и т. д.

В свою очередь, процедура расширения ключа алгоритма SAFER SK-128 является некоторой комбинацией процедур расширения ключа алгоритмов SAFER K-128 и SAFER SK-64, а именно:

- исходный 128-битный ключ шифрования разбивается на 2 64-битных фрагмента  $K_a$  и  $K_b$ ;

## Описание алгоритмов

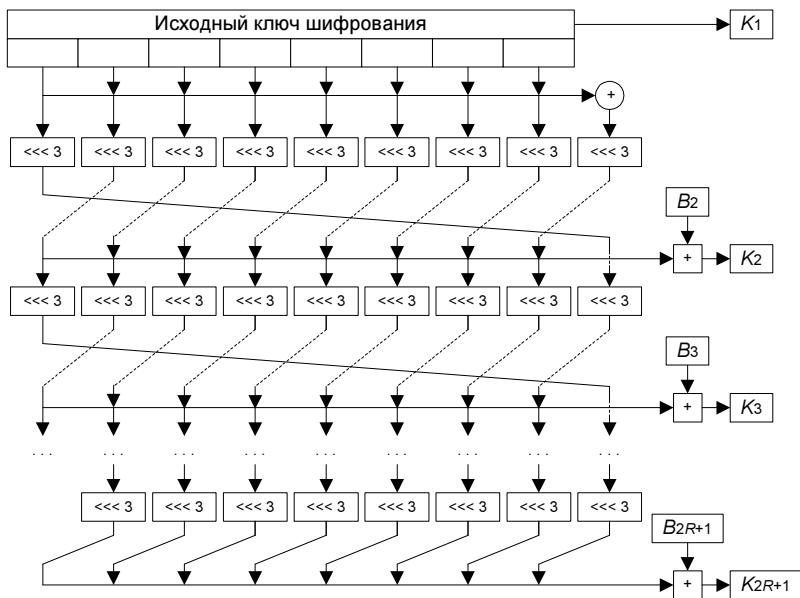
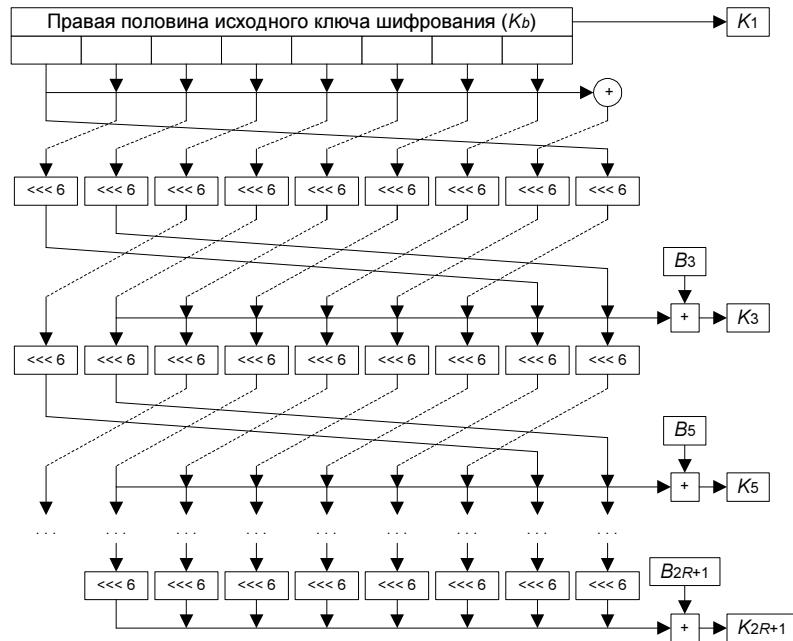


Рис. 3.167. Процедура расширения ключа алгоритма SAFER SK-64

Рис. 3.168. Процедура расширения ключа алгоритма SAFER SK-128  
(нечетные фрагменты)

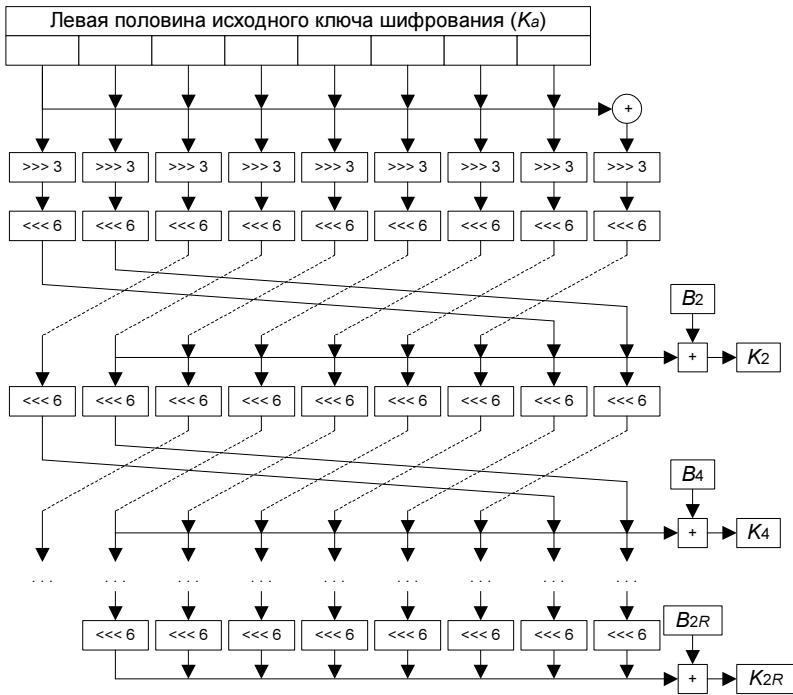


Рис. 3.169. Процедура расширения ключа алгоритма SAFER SK-128 (четные фрагменты)

- правая половина ключа ( $K_b$ ) дополняется 9-м байтом (аналогично процедуре SAFER SK-64) и участвует в формировании фрагментов расширенного ключа с нечетными индексами (рис. 3.168);
- левая половина ключа ( $K_a$ ) также дополняется 9-м байтом и участвует в формировании фрагментов с четными индексами (рис. 3.169).

У алгоритмов SAFER SK-64 и SAFER SK-128 сохраняется то свойство, что при  $K_a = K_b$  алгоритм SAFER SK-128 является полностью совместимым (при одинаковом количестве раундов  $R$ ) с алгоритмом SAFER SK-64, в котором используется ключ  $K = K_a = K_b$ . Минимальное и рекомендованное количество раундов алгоритма SAFER SK-64 увеличено по сравнению с SAFER K-64 и составляет 8 вместо 6. У алгоритма SAFER SK-128 количество раундов осталось прежним (как у SAFER K-128).

«SK» в названии алгоритма обозначает Strengthened Key Schedule, т. е. «усиленное расширение ключа». Однако известна шутка [395], что SK так же хорошо может обозначать и «Stop Knudsen», поскольку новая процедура расширения ключа делает атаку Кнудсена невозможной.

## Описание алгоритмов

В ряде источников, например, [253] и [339], упоминается алгоритм SAFER SK-40, использующий 40-битный ключ шифрования. Такой короткий ключ шифрования позволял алгоритму SAFER SK-40 обойти существовавшие на тот момент в США экспортные ограничения. Алгоритм (включая процедуру расширения ключа) мало чем отличался от SAFER SK-64. Как было сказано выше, в процедуре расширения ключа алгоритма SAFER SK-64 к 8 байтам исходного ключа определенным образом добавляется 9-й байт (см. рис. 3.167). У SAFER SK-40 эти 9 байтов (обозначим их  $KI_1 \dots KI_9$ ) формируются иначе (рис. 3.170) [339]:

- байты с 1-го по 5-й формируются непосредственно из 40-битного ключа;
- остальные байты (с 6-го по 9-й) формируются, соответственно, так:

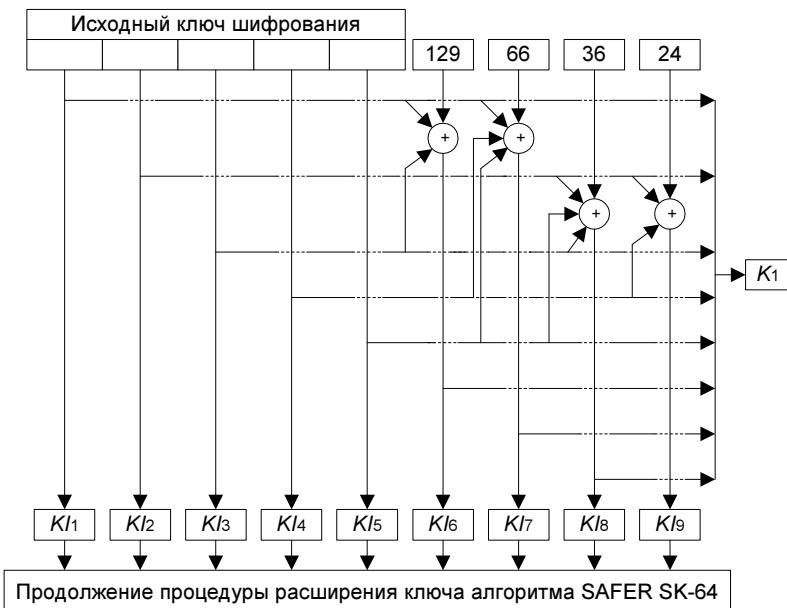
$$KI_6 = KI_1 \oplus KI_3 \oplus 129;$$

$$KI_7 = KI_1 \oplus KI_4 \oplus KI_5 \oplus 66;$$

$$KI_8 = KI_2 \oplus KI_3 \oplus KI_5 \oplus 36;$$

$$KI_9 = KI_2 \oplus KI_4 \oplus 24.$$

Далее идет обычная для SAFER SK-64 процедура расширения ключа.



**Рис. 3.170.** Начальный этап процедуры расширения ключа алгоритма SAFER SK-40

## Криптоанализ алгоритмов SAFER SK-64 и SAFER SK-128

Какие-либо работы, вскрывающие полнораундовые версии алгоритмов SAFER SK-64 и SAFER SK-128, не получили широкой известности. Было предложено несколько атак на версии алгоритмов с уменьшенным количеством раундов, среди которых можно отметить следующие:

- в работе [281] описана атака на 3,75-раундовый SAFER SK-64 (по мнению ее авторов, «0,75 раунда» — это раунд алгоритма без преобразований *PHT*) методом невозможных дифференциалов, для которой требуется  $2^{32}$  выбранных открытых текстов и  $2^{62}$  тестовых операций шифрования;
- там же описана Square-атака на 3,25-раундовые SAFER SK-64 и SAFER SK-128 (0,25 раунда включает в себя только первое наложение ключа), использующая  $2^{10,3}$  выбранных открытых текстов и  $2^{38}$  тестовых операций;
- в [279] описана атака методом линейного криптоанализа на 4,75-раундовый SAFER SK-128, для которой требуется  $2^{64}$  известных открытых текстов (т. е. все возможные открытые тексты и соответствующие им шифртексты) и  $2^{90}$  операций шифрования.

Как видно, даже атаки на усеченные версии данных алгоритмов не являются достаточно практическими.

## 3.45. Алгоритм SAFER+

Алгоритм SAFER+ был разработан в 1998 г. по заказу той же корпорации Cylink, что и рассмотренные в разд. 3.44 алгоритмы семейства SAFER, специально для участия в конкурсе AES (*см. разд. 2.1*). SAFER+ является усовершенствованием ранее разработанных алгоритмов семейства SAFER. В его разработке, помимо Джеймса Мэсси, принимали участие специалисты Академии Наук Армении Гурген Хачатрян (Gurgen H. Khachatrian) и Мельсик Курегян (Melsik K. Kuregian).

### Структура алгоритма

Согласно требованиям конкурса AES, в отличие от алгоритмов SAFER K и SAFER SK, алгоритм SAFER+ шифрует данные 128-битными блоками. Таким образом, обрабатываемый блок данных имеет размер 16 байтов, которые нумеруются слева направо от 1 до 16. Для удобства дальнейшего описания байты данных можно разделить на 2 группы:

- группа № 1: байты №№ 1, 4, 5, 8, 9, 12, 13 и 16;
- группа № 2: остальные байты.

## Описание алгоритмов

Количество раундов  $R$  определяется размером ключа (согласно требованиям конкурса, алгоритм должен поддерживать 128-, 192- и 256-битные ключи):

- $R = 8$  для 128-битного ключа;
- $R = 12$  для 192-битного ключа;
- $R = 16$  для 256-битного ключа.

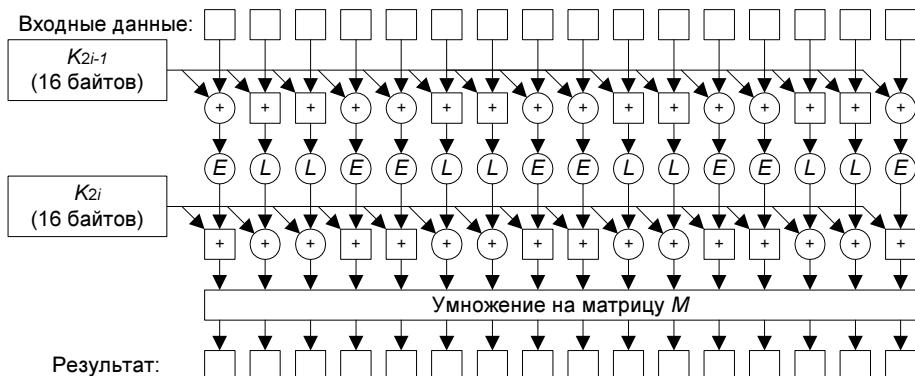


Рис. 3.171. Раунд алгоритма SAFER+

По своей структуре алгоритм SAFER+ весьма похож на SAFER K-64 (с учетом измененного размера блока; алгоритм SAFER K-64 подробно описан в разд. 3.44). В каждом раунде алгоритма выполняются следующие действия (рис. 3.171) [253]:

1. Первое наложение ключа. На данные накладывается 16-байтный фрагмент расширенного ключа  $K_{2i-1}$ , где  $i$  — номер текущего раунда (раунды нумеруются, начиная с 1; процедура расширения ключа будет описана далее). Группа № 1 байтов ключа накладывается на аналогичные байты данных с помощью операции XOR; для наложения остальных байтов ключа используется операция сложения по модулю 256.
2. Нелинейное преобразование. Выполняется с помощью тех же описанных в разд. 3.44 операций  $L$  и  $E$ : операция  $E$  применяется к байтам 1-й группы, а к остальным байтам применяется операция  $L$ .
3. Второе наложение ключа с использованием фрагмента  $K_{2i}$ . Для обработки 1-й группы байтов используется операция сложения по модулю 256, для остальных — операция XOR.
4. Умножение 16-байтного блока данных на матрицу  $M$  (по модулю 256), приведенную в табл. 3.102.

Таблица 3.102

2	2	1	1	16	8	2	1	4	2	4	2	1	1	4	4
1	1	1	1	8	4	2	1	2	1	4	2	1	1	2	2
1	1	4	4	2	1	4	2	4	2	16	8	2	2	1	1
1	1	2	2	2	1	2	1	4	2	8	4	1	1	1	1
4	4	2	1	4	2	4	2	16	8	1	1	1	1	2	2
2	2	2	1	2	1	4	2	8	4	1	1	1	1	1	1
1	1	4	2	4	2	16	8	2	1	2	2	4	4	1	1
1	1	2	1	4	2	8	4	2	1	1	1	2	2	1	1
2	1	16	8	1	1	2	2	1	1	4	4	4	2	4	2
2	1	8	4	1	1	1	1	1	2	2	4	2	2	1	
4	2	4	2	4	4	1	1	2	2	1	1	16	8	2	1
2	1	4	2	2	2	1	1	1	1	1	1	8	4	2	1
4	2	2	2	1	1	4	4	1	1	4	2	2	1	16	8
4	2	1	1	1	1	2	2	1	1	2	1	2	1	8	4
16	8	1	1	2	2	1	1	4	4	2	1	4	2	4	2
8	4	1	1	1	1	1	1	2	2	2	1	2	1	4	2

В зависимости от требований к реализации алгоритма, умножение на матрицу  $M$  может быть заменено 4-мя уровнями преобразований РНТ (РНТ было описано в разд. 3.44), между которыми выполняются три байтовые перестановки (по одной между каждыми двумя уровнями — см. рис. 3.172), представленные в табл. 3.103, 3.104 и 3.105 [336]:

Таблица 3.103

9	2	11	4	5	16	7	14	3	12	1	10	15	8	13	6
---	---	----	---	---	----	---	----	---	----	---	----	----	---	----	---

Таблица 3.104

5	2	3	8	1	6	7	10	9	14	15	12	13	14	11	16
---	---	---	---	---	---	---	----	---	----	----	----	----	----	----	----

Таблица 3.105

1	4	3	6	5	8	7	2	9	12	11	10	13	16	15	14
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

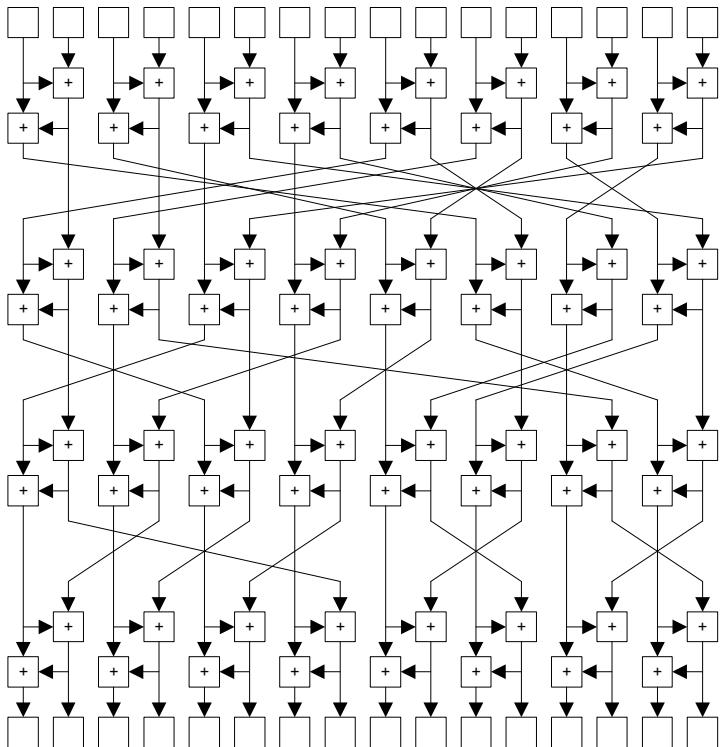


Рис. 3.172. Альтернативное представление умножения на матрицу  $M$

Это означает (на примере последней из данных перестановок), что байт № 1 остается на своем месте, входное значение байта № 4 становится выходным значением байта № 2 и т. д.

После выполнения  $R$  раундов алгоритма производится выходное преобразование, которое полностью аналогично описанной выше операции первого наложения ключа; здесь используется последний из фрагментов расширенного ключа  $K_{2R+1}$ .

Таким образом, различия между SAFER+ и SAFER K-64 (не считая процедуры расширения ключа) сводятся к следующим:

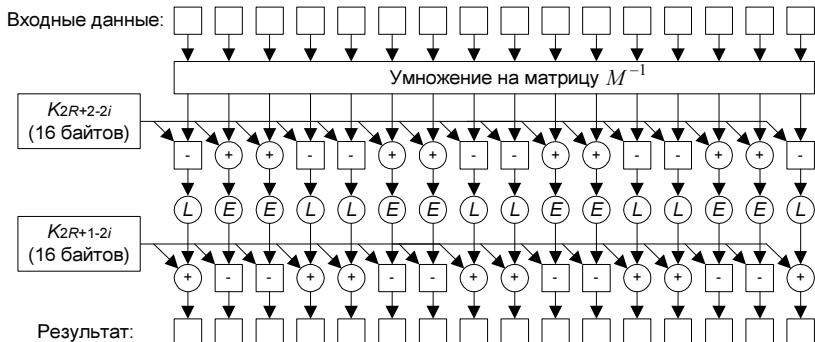
- размер шифруемого блока данных удвоен, причем первые три операции раунда, фактически, не изменились, а просто «расширились» на 128-битный блок;
- более существенно изменилось применение преобразований  $PHT$  — теперь 4 уровня вместо трех плюс «несимметричность» перестановок между уровнями.

## Расшифровывание

Как и в алгоритме SAFER K-64, расшифровывание в SAFER+ выполняется применением обратных операций в обратной последовательности (см., в частности, рис. 3.173). Поэтому не привожу здесь подробное описание расшифровывания, ограничившись только матрицей  $M^{-1}$ , заменяющей 4 уровня обратных преобразований  $IPHT$  — см. табл. 3.106.

**Таблица 3.106**

2	-2	1	-2	1	-1	4	-8	2	-4	1	-1	1	-2	1	-1
-4	4	-2	4	-2	2	-8	16	-2	4	-1	1	-1	2	-1	1
1	-2	1	-1	2	-4	1	-1	1	-1	1	-2	2	-2	4	-8
-2	4	-2	2	-2	4	-1	1	-1	1	-1	2	-4	4	-8	16
1	-1	2	-4	1	-1	1	-2	1	-2	1	-1	4	-8	2	-2
-1	1	-2	4	-1	1	-1	2	-2	4	-2	2	-8	16	-4	4
2	-4	1	-1	1	-2	1	-1	2	-2	4	-8	1	-1	1	-2
-2	4	-1	1	-1	2	-1	1	-4	4	-8	16	-2	2	-2	4
1	-1	1	-2	1	-1	2	-4	4	-8	2	-2	1	-2	1	-1
-1	1	-1	2	-1	1	-2	4	-8	16	-4	4	-2	4	-2	2
1	-2	1	-1	4	-8	2	-2	1	-1	1	-2	1	-1	2	-4
-1	2	-1	1	-8	16	-4	4	-2	2	-2	4	-1	1	-2	4
4	-8	2	-2	1	-2	1	-1	1	-2	1	-1	2	-4	1	-1
-8	16	-4	4	-2	4	-2	2	-1	2	-1	1	-2	4	-1	1
1	-1	4	-8	2	-2	1	-2	1	-1	2	-4	1	-1	1	-2
-2	2	-8	16	-4	4	-2	4	-1	1	-2	4	-1	1	-1	2



**Рис. 3.173.** Раунд расшифровывания алгоритма SAFER+

## Процедура расширения ключа

Расширение ключа алгоритма SAFER+ похоже на такое у алгоритма SAFER SK-64 с учетом увеличения размера блока (и, соответственно, удвоения размера каждого фрагмента расширенного ключа). Данная процедура выполняется следующим образом (рис. 3.174):

1. Вычисляется дополнительный байт (как и в SAFER SK-64, операцией XOR, применяемой ко всем байтам исходного ключа шифрования). То есть, например, для 256-битного ключа получается временный ключ размером 33 байта.
2. Значение каждого байта временного ключа циклически сдвигается влево на 3 бита.
3. После выполнения сдвига выбираются 16 байтов, начиная с байта, номер которого соответствует номеру раунда расширения ключа  $i$  (например, для раунда № 18 расширения 256-битного ключа выбираются байты с 18 по 33-й, для последующих раундов необходимое количество байтов «додирается», начиная с 1-го байта).
4. Выбранные байты побайтно складываются по модулю 256 с соответствующими байтами 16-байтной константы  $B_i$ . Константы  $B_i$  описаны далее. Результатом данного шага является очередной фрагмент ключа  $K_i$ .
5. Шаги 2–4 представляют собой раунд процедуры расширения ключа. Раунды нумеруются, начиная с 2. Фрагментом  $K_1$  являются первые 16 байтов исходного ключа шифрования без каких-либо изменений.

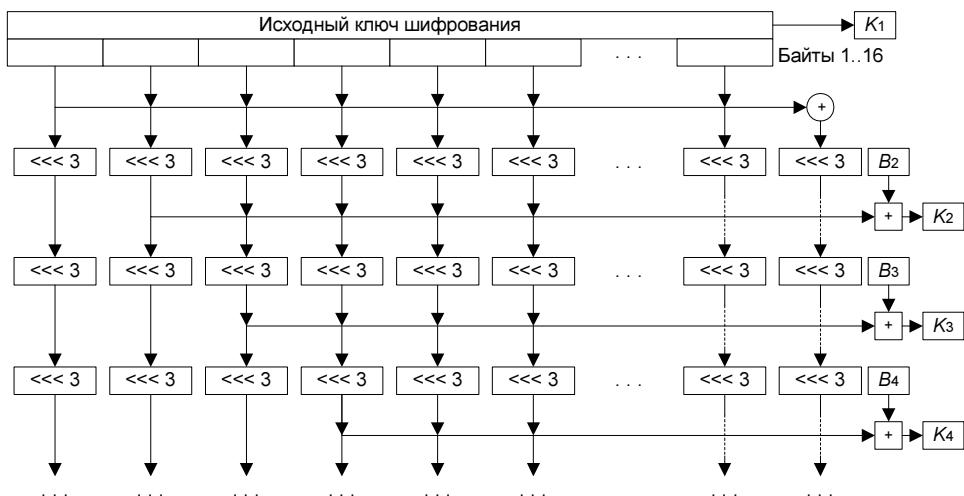


Рис. 3.174. Расширение ключа в алгоритме SAFER+

Константы  $B_i$  вычисляются с помощью следующих формул:

$$B_i[j] = E(E(17i + j)) \text{ для } i = 2 \dots 17;$$

$$B_i[j] = E(17i + j) \text{ для } i \geq 18,$$

где  $B_i[j]$  —  $j$ -й байт константы  $B_i$ .

Константы  $B_i$  могут быть также заданы таблицей (табл. 3.107),  $i$ -я строка которой содержит байты константы  $B_{i+1}$ .

**Таблица 3.107**

70	151	177	186	163	183	16	10	197	55	179	201	90	40	172	100
236	171	170	198	103	149	88	13	248	154	246	110	102	220	5	61
138	195	216	137	106	233	54	73	67	191	235	212	150	155	104	160
93	87	146	31	213	113	92	187	34	193	190	123	188	153	99	148
42	97	184	52	50	25	253	251	23	64	230	81	29	65	68	143
221	4	128	222	231	49	214	127	1	162	247	57	218	111	35	202
58	208	28	209	48	62	18	161	205	15	224	168	175	130	89	44
125	173	178	239	194	135	206	117	6	19	2	144	79	46	114	51
192	141	207	169	129	226	196	39	47	108	122	159	82	225	21	56
252	32	66	199	8	228	9	85	94	140	20	118	96	255	223	215
250	11	33	0	26	249	166	185	232	158	98	76	217	145	80	210
24	180	7	132	234	91	164	200	14	203	72	105	75	78	156	53
69	77	84	229	37	60	12	74	139	63	204	167	219	107	174	244
45	243	124	109	157	181	38	116	242	147	83	176	240	17	237	131
182	3	22	115	59	30	142	112	189	134	27	71	126	36	86	241
136	70	151	177	186	163	183	16	10	197	55	179	201	90	40	172
220	134	119	215	166	17	251	244	186	146	145	100	131	241	51	239
44	181	178	43	136	209	153	203	140	132	29	20	129	151	113	202
163	139	87	60	130	196	82	92	28	232	160	4	180	133	74	246
84	182	223	12	26	142	222	224	57	252	32	155	36	78	169	152
171	242	96	208	108	234	250	199	217	0	212	31	110	67	188	236
137	254	122	93	73	201	50	194	249	154	248	109	22	219	89	150
233	205	230	70	66	143	10	193	204	185	101	176	210	198	172	30
98	41	46	14	116	80	2	90	195	37	123	138	42	91	240	6

Таблица 3.107

71	111	112	157	126	16	206	18	39	213	76	79	214	121	48	104
117	125	228	237	128	106	144	55	162	94	118	170	197	127	61	175
229	25	97	253	77	124	183	11	238	173	75	34	245	231	115	35
200	5	225	102	221	179	88	105	99	86	15	161	49	149	23	7
40	1	45	226	147	190	69	21	174	120	3	135	164	184	56	207
8	103	9	148	235	38	168	107	189	24	52	27	187	191	114	247
53	72	156	81	47	59	85	227	192	159	216	211	243	141	177	255
62	220	134	119	215	166	17	251	244	186	146	145	100	131	241	51

## Криптоанализ алгоритма SAFER+

Поскольку SAFER+ участвовал в конкурсе AES, к нему было приковано достаточно пристальное внимание криптологов как в течение конкурса, так и по его завершении. Среди криптоаналитических исследований данного алгоритма стоит отметить следующие:

- в работе [198] предлагается атака на алгоритм SAFER+/256 (т. е. SAFER+ с 256-битным ключом шифрования), для проведения которой требуется не более трех известных открытых текстов (и соответствующих им шифртекстов) и около  $2^{240}$  тестовых операций шифрования; там же описана и атака на связанных ключах (на тот же SAFER+/256), для которой требуется  $3 * 2^{32}$  выбранных открытых текстов, зашифрованных на двух ключах, и около  $2^{200}$  операций; обе атаки абсолютно непрактичны, но доказывают недостаточный запас криптостойкости SAFER+ с 256-битным ключом; авторы [198] предложили меры по усилению процедуры расширения ключа, которые сделали бы обе атаки невозможными;
- в работе [114] утверждается, что SAFER+ подвержен дифференциальному криптоанализу по потребляемой мощности; аналогичные выводы сделаны в [80];
- ряд атак на версии алгоритма с усеченным количеством раундов предложены в [278] и [279].

Кроме того, в итоговом документе первого раунда конкурса AES [284] дается следующая информация о SAFER+.

- Достоинства алгоритма:
  - алгоритм имеет достаточный запас криптостойкости (с учетом модификации процедуры расширения ключа, которая будет описана далее);

- алгоритм достаточно нетребователен к ресурсам, поэтому может применяться в устройствах с ограниченными ресурсами (например, в смарт-картах).

Недостатки:

- алгоритм обладает весьма низкой скоростью шифрования и достаточно медленной процедурой расширения ключа;
- алгоритм подвержен атакам по потребляемой мощности (что, впрочем, справедливо даже для алгоритма Rijndael — победителя конкурса).

В результате алгоритм SAFER+ не вышел во второй раунд конкурса AES: во-первых, из-за недостаточного быстродействия, а во-вторых, «благодаря» алгоритму Serpent, обладающему схожими характеристиками, но более быстрому и с большим запасом криптостойкости — эксперты решили, что в финале конкурса из этих двух алгоритмов предпочтение однозначно будет отдано алгоритму Serpent [284].

## Единое расширение ключа SAFER+

В процессе проведения конкурса AES авторы алгоритма SAFER+ модифицировали процедуру расширения ключа (такие модификации допускались условиями конкурса), чтобы обеспечить противостояние атакам, описанным в [198]. Результирующая процедура расширения ключа напоминает таковую для SAFER SK-128; ее схема для 256-битного ключа приведена на рис. 3.175 и 3.176 [396]. Эта же процедура (названная «SAFER+ Unified Key Schedule» — «единое расширение ключа SAFER+») используется для расширения 128- и 192-битных ключей, но со следующими изменениями:

- в качестве «правой половины ключа» (рис. 3.175) при расширении 128-битного ключа используется сам исходный ключ шифрования;
- байты №№ 17...24 192-битного ключа формируют первые 8 байтов «правой половины ключа»; остальные 8 байтов формируются применением операции XOR к следующим группам байтов исходного ключа:
  - №№ 1, 9, 17;
  - №№ 2, 10, 18;
  - №№ 3, 11, 19 и т. д.

Данная модификация алгоритма не способствовала его выходу во второй раунд конкурса AES, поскольку не повлияла принципиально на его характеристики.

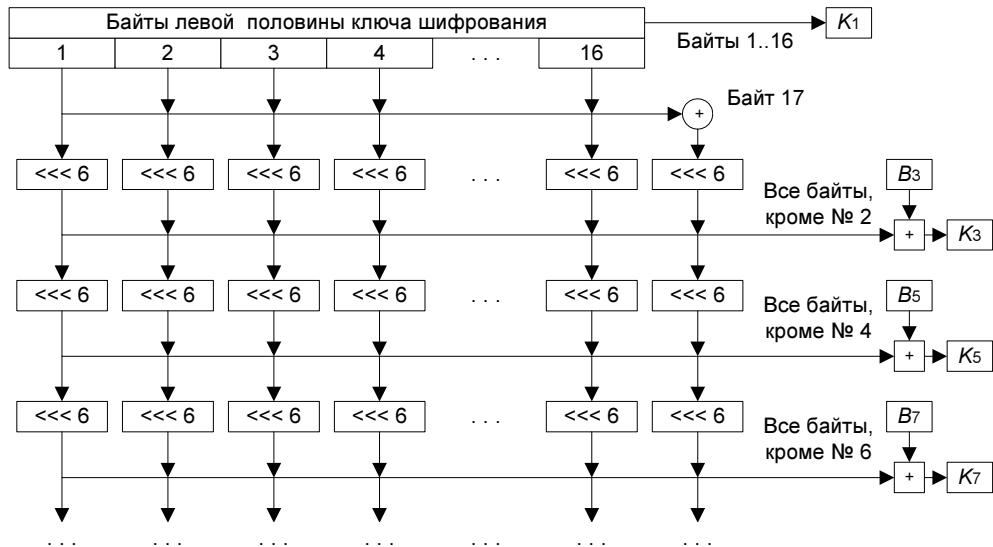


Рис. 3.175. Процедура SAFER+ Unified Key Schedule (нечетные фрагменты)

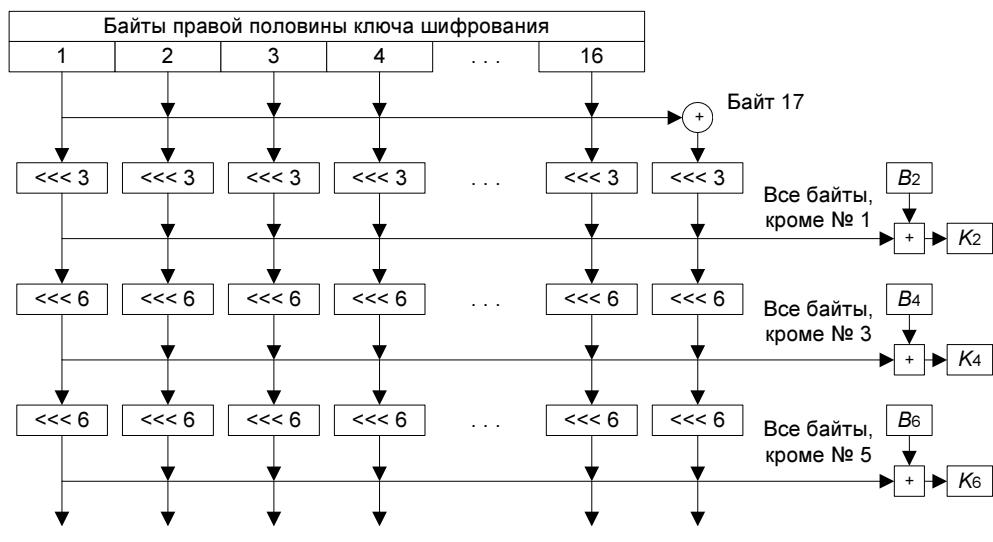


Рис. 3.176. Процедура SAFER+ Unified Key Schedule (четные фрагменты)

### 3.46. Алгоритм SAFER++

Алгоритм SAFER++ был разработан авторами алгоритма SAFER+ (см. разд. 3.45) в 2000 г. для участия в конкурсе NESSIE (см. разд. 2.2), в котором проводился выбор криптографических стандартов Евросоюза.

#### Структура алгоритма

Алгоритм SAFER++ существует в двух вариантах:

- основной вариант со 128-битным размером блока; использует 128- и 256-битные ключи шифрования (поддержка 192-битного ключа исключена);
- вариант с 64-битным размером блока и 128-битным ключом шифрования.

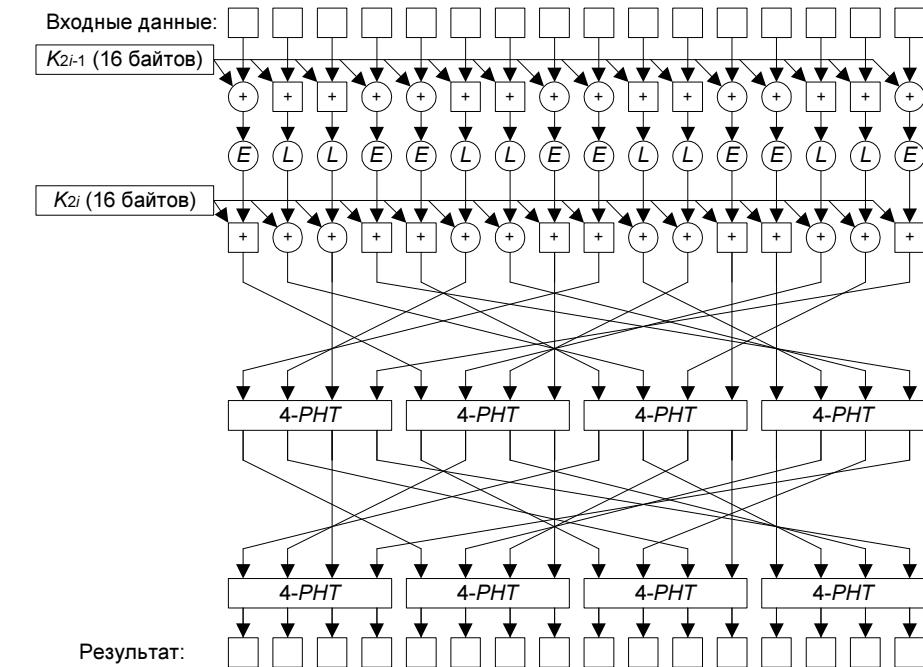


Рис. 3.177. Раунд алгоритма SAFER++

Рассмотрим сначала вариант № 1 алгоритма SAFER++. Его раунд (рис. 3.177) весьма похож на раунд алгоритма SAFER+: первые три действия идентичны описанным в разд. 3.45 для алгоритма SAFER+, вместо же действия 4 выполняется следующая последовательность операций [254]:

1. Байтовая перестановка согласно табл. 3.108.

Таблица 3.108

9	6	3	16	1	14	11	8	5	2	15	12	13	10	7	4
---	---	---	----	---	----	----	---	---	---	----	----	----	----	---	---

2. Операция 4-PHT, которую можно представить следующим образом:

$$[A, B, C, D] = [a, b, c, d]^* H4,$$

где:

- $a \dots d$  и  $A \dots D$  — это значения входных и выходных байтов соответственно;
- умножение выполняется по модулю 256;
- матрица  $H4$  приведена в табл. 3.109.

Таблица 3.109

2	1	1	1
1	2	1	1
1	1	2	1
1	1	1	1

В [254] приведен пример реализации данного преобразования — см. рис. 3.178.

3. Две предыдущие операции этой последовательности выполняются повторно.

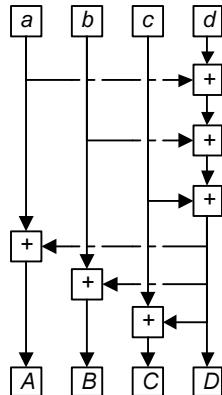


Рис. 3.178. Операция 4-PHT

Кроме того, уменьшено количество раундов алгоритма, теперь:

- $R = 7$  для 128-битного ключа;
- $R = 10$  для 256-битного ключа.

Выходное преобразование, а также входное преобразование при расшифровывании не претерпели каких-либо изменений по сравнению с SAFER+.

Входные данные:

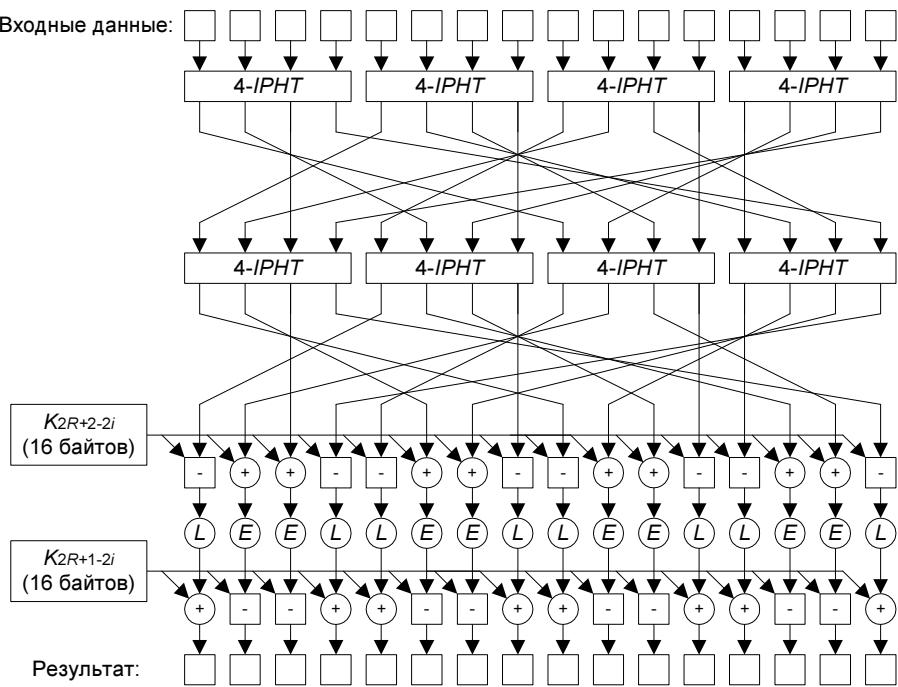


Рис. 3.179. Раунд расшифровывания алгоритма SAFER++

Раунд расшифровывания алгоритма SAFER++ изменился в соответствии с изменениями в раунде зашифровывания (рис. 3.179). Здесь используется обратная перестановка, приведенная в табл. 3.110.

Таблица 3.110

5	10	3	16	9	2	15	8	1	14	7	12	13	6	11	4
---	----	---	----	---	---	----	---	---	----	---	----	----	---	----	---

Кроме того, вместо 4-PHT используется обратное преобразование 4-IPHT, представляющее собой умножение на обратную матрицу (табл. 3.111).

Таблица 3.111

1	0	0	-1
0	1	0	-1
0	0	1	-1
-1	-1	-1	4

В качестве процедуры расширения ключа используется описанная выше процедура единого расширения ключа SAFER+ (см. рис. 3.175 и 3.176), но с некоторым изменением в вычислении констант  $B_i$ :

- $B_i[j] = E(E(17i + j))$  для  $i = 2 \dots 15$ ;
- $B_i[j] = E(17i + j)$  для  $i = 16 \dots 21$ ,

что приводит к изменению по сравнению с SAFER+ только значений  $B_{16}$  и  $B_{17}$ , которые приведены в табл. 3.112.

Таблица 3.112

103	9	148	235	38	168	107	189	24	52	27	187	191	114	247	64
72	156	81	47	59	85	227	192	159	216	211	243	141	177	255	167

## 64-битный вариант SAFER++

Второй вариант алгоритма SAFER++ представляет собой некоторое «усечение» первого варианта до 64-битного размера блока. В данном алгоритме выполняются 8 раундов преобразований, в каждом из которых производятся следующие действия (рис. 3.180) [254]:

1. Первое наложение ключа. Выполняется аналогично алгоритму SAFER K-64; в качестве ключевого материала здесь используются первые 8 байтов фрагмента расширенного ключа  $K_i$  (где  $i$  — номер раунда).
2. Нелинейное преобразование, полностью аналогичное алгоритму SAFER K-64.
3. Второе наложение ключа, также аналогичное алгоритму SAFER K-64; здесь используются байты 9...16 фрагмента расширенного ключа  $K_i$ .
4. Расширение 8-байтного блока данных до 16 байтов. Каждый nibl входных байтов формирует старшую половину одного из выходных байтов, младшая половина обнуляется.
5. Два уровня байтовых перестановок и операций 4-PHT. Выполняются аналогично основному варианту алгоритма SAFER++.

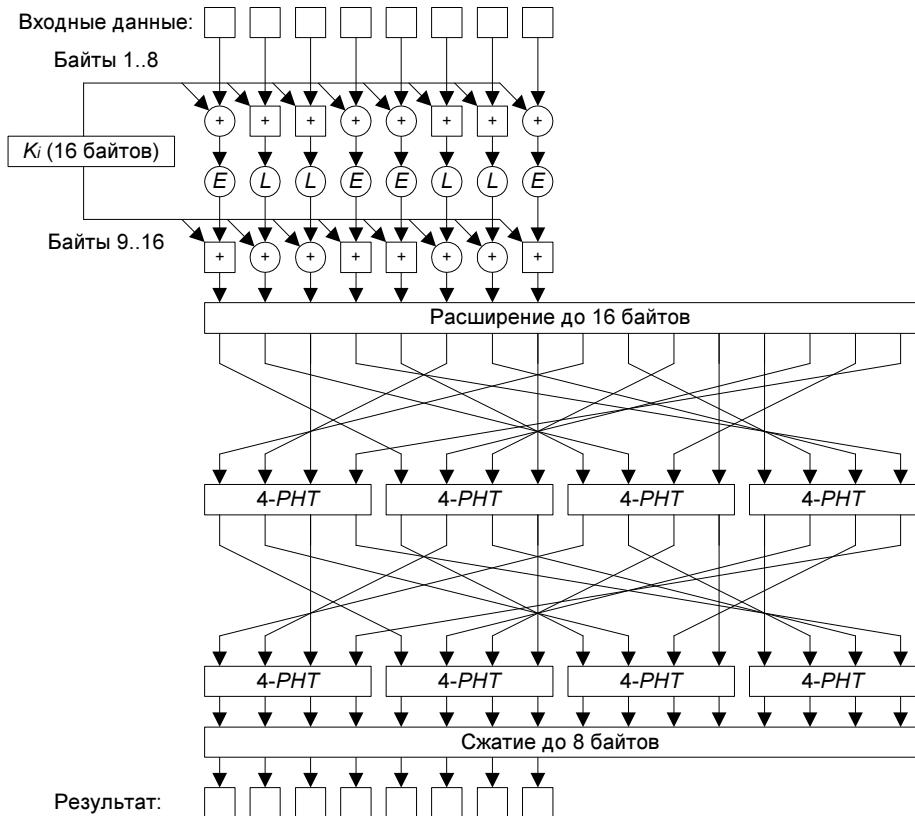


Рис. 3.180. Раунд 64-битного варианта алгоритма SAFER++

6. Сжатие полученного 16-байтного блока данных до 8 байтов. Каждый из выходных байтов данной операции формируется из старших nibлов двух входных байтов (младшие nibлы остаются равными нулю, исходя из свойств операции 4-PHT).

По завершении 8 раундов выполняется выходное преобразование, аналогичное первому наложению ключа; здесь используются первые 8 байтов  $K_9$ .

Процедура расширения ключа полностью аналогична таковой для основного варианта алгоритма SAFER++, но с учетом того, что в данном варианте используется только 9 фрагментов  $K_1 \dots K_9$ .

Как и во всех рассмотренных ранее алгоритмах семейства SAFER, расшифровывание выполняется применением обратных операций в обратной последовательности. Структура раунда расшифровывания 64-битного варианта алгоритма SAFER++ приведена на рис. 3.181.

## Описание алгоритмов

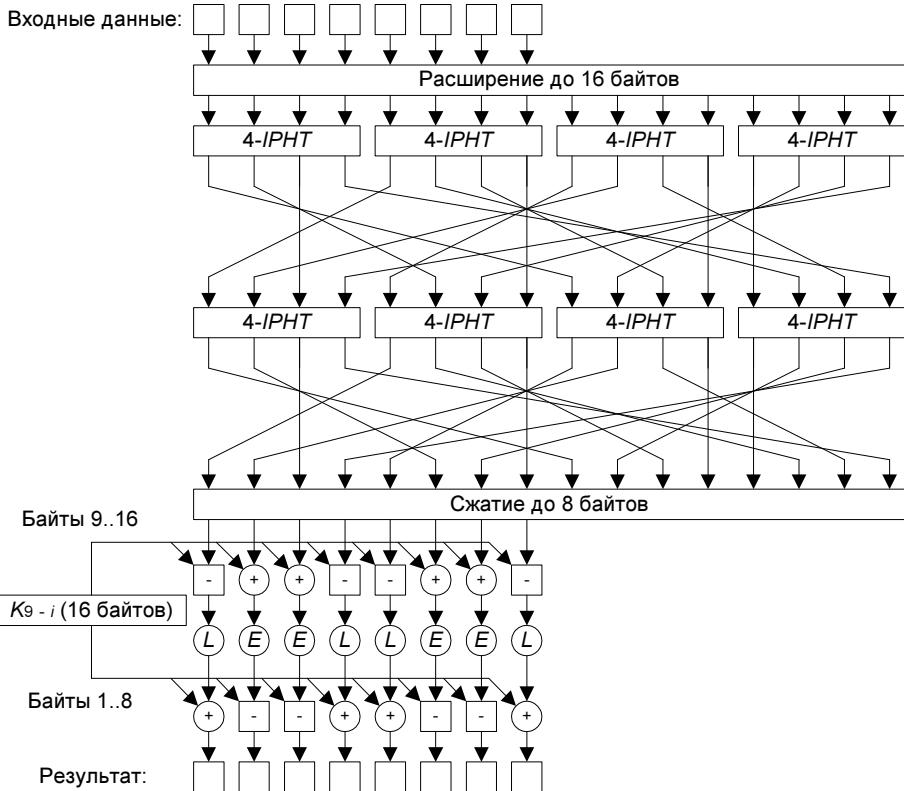


Рис. 3.181. Раунд расшифровывания 64-битного варианта алгоритма SAFER++

**Криптоанализ алгоритма SAFER++**

В рамках конкурса NESSIE алгоритм SAFER++ был достаточно глубоко исследован. Как отмечают эксперты конкурса [307], SAFER++ не унаследовал каких-либо уязвимостей от более ранних алгоритмов семейства SAFER, а каких-либо новых уязвимостей не было найдено.

Криптоаналитики предложили множество атак на различные варианты алгоритма SAFER++ с усеченным количеством раундов (см., например, [278, 281, 303]), причем предложенная в [87] атака методом бумеранга позволяет вскрыть до 5,5 (из 7) раундов алгоритма (для атаки на 5,5 раундов необходимо  $2^{108}$  выбранных открытых текстов и шифртекстов с адаптивным выбором и столько же операций шифрования); несмотря на абсолютную непрактичность этой атаки, она позволила экспертам конкурса говорить о недостаточном запасе криптостойкости алгоритма SAFER++.

В результате из-за недостаточного запаса криптостойкости, относительно низкой скорости шифрования (за исключением реализаций для 8-битных платформ) и предположений ряда экспертов о том, что структура SAFER++ может содержать скрытые уязвимости, алгоритм SAFER++ не стал одним из победителей конкурса NESSIE [305].

## Заключение

Несмотря на то, что алгоритмы семейства SAFER не получили искомого статуса стандартов США или Евросоюза, стоит сказать о том, что данные алгоритмы нашли достаточно широкое применение. В частности, в [254] отмечается, что SAFER+ используется в качестве основы протокола аутентификации в Bluetooth. В любом случае, видно, что от версии к версии алгоритмов семейства SAFER происходит повышение их криптостойкости в ответ на появление новых атак на данные алгоритмы. Есть уверенность, что история семейства SAFER на алгоритме SAFER++ не закончится.

## 3.47. Алгоритм SC2000

Алгоритм SC2000 разработан в 2000 г. группой японских специалистов из компании Fujitsu Laboratories и Университета г. Токио.

Алгоритм SC2000 имеет три фиксированных размера ключа шифрования: 128, 192 и 256 битов.

### Структура алгоритма

Алгоритм представляет собой SP-сеть; в нем выполняются 7 раундов преобразований при использовании 128-битного ключа или 8 раундов при использовании ключей больших размеров. Раунд алгоритма имеет достаточно сложную структуру; первый раунд показан на рис. 3.182. В нем выполняются следующие операции [357]:

1. Входное 128-битное значение делится на 4 субблока по 32 бита, на каждый из которых операцией XOR накладывается соответствующий 32-битный фрагмент расширенного ключа (процедура расширения ключа будет описана далее), в этом случае — фрагменты  $k_0 \dots k_3$ .
2. Затем выполняется операция  $T()$ , которая переразбивает блок данных на 32 субблока по 4 бита каждый. Переразбиение выполняется так (рис. 3.183):  $i$ -й 4-битный субблок (обозначим его  $o_i$ ) формируется из четырех  $i$ -х битов субблоков  $a, b, c$  и  $d$ .

## Описание алгоритмов

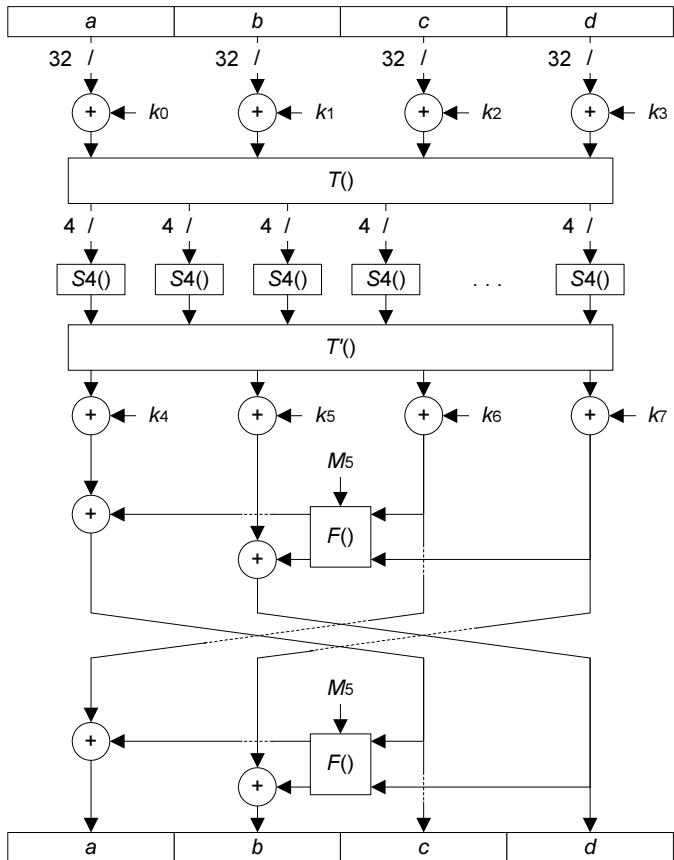


Рис. 3.182. Первый раунд алгоритма SC2000

3. Каждый 4-битный субблок «прогоняется» через таблицу замен  $S4$ : входное 4-битное значение заменяется выходным того же размера согласно табл. 3.113.

Таблица 3.113

2	5	10	12	7	15	1	11	13	6	0	9	4	8	3	14
---	---	----	----	---	----	---	----	----	---	---	---	---	---	---	----

То есть значение 0 заменяется значением 2, значение 1 меняется на 5 и т. д.

4. Обрабатываемый блок данных снова переразбивается на 32-битные субблоки с помощью операции  $T'()$ , которая, фактически, является обратной к операции  $T()$  — см. рис. 3.184 (на рис. 4-битные субблоки обозначены как  $t0...t31$ ).

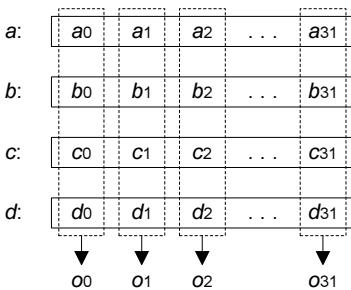


Рис. 3.183. Операция  $T()$   
алгоритма SC2000

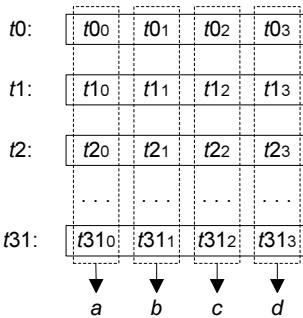


Рис. 3.184. Операция  $T'()$   
алгоритма SC2000

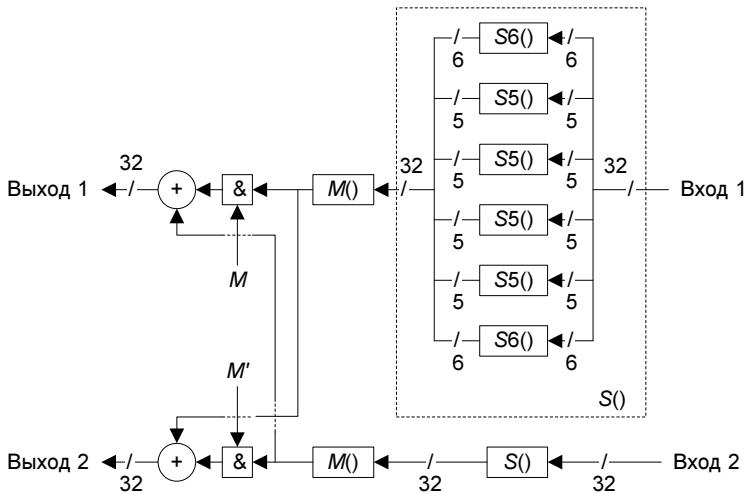


Рис. 3.185. Функция  $F()$  алгоритма SC2000

5. С помощью операции XOR выполняется наложение еще четырех фрагментов расширенного ключа; для первого раунда это фрагменты  $k_4 \dots k_7$ .
6. Значения субблоков  $C$  и  $D$  (а также маскирующая константа  $M_5 = 55555555$ , указано шестнадцатеричное значение) подаются на вход функции  $F()$ , которая будет подробно описана далее. В результате выполнения этой функции получаются два 32-битных значения, которые накладываются операцией XOR на субблоки  $A$  и  $B$ .
7. Субблоки  $A$  и  $B$  меняются местами с субблоками  $C$  и  $D$  соответственно, после чего повторно выполняется шаг 6.

## Описание алгоритмов

Упомянутая выше функция  $F()$  обрабатывает два входных 32-битных субблока следующим образом (рис. 3.185):

- Каждое из входных 32-битных значений разбивается на 6 фрагментов, два из которых являются 6-битными, а остальные имеют размер по 5 битов.
- 6-битные фрагменты «прогоняются» через таблицу замен  $S6$  (см. табл. 3.114), а 5-битные обрабатываются таблицей  $S5$  (см. табл. 3.115).

**Таблица 3.114**

47	59	25	42	15	23	28	39	25	38	36	19	60	24	29	56
37	63	20	61	55	2	30	44	9	10	6	22	53	48	51	11
62	52	35	18	14	46	0	54	17	40	27	4	31	8	5	12
3	16	41	34	33	7	45	49	50	58	1	21	43	57	32	13

**Таблица 3.115**

20	26	7	31	19	12	10	15	22	30	13	14	4	24	9	18
27	11	1	21	6	16	2	28	23	5	8	3	0	17	29	25

Принцип действия этих таблиц аналогичен таблице S4.

- Полученные после замены значения снова объединяются в 32-битные субблоки, каждый из которых обрабатывается операцией  $M()$ ; данная операция выполняет умножение 32-битного входного значения (представленного в виде вектора) на матрицу размером  $32 \times 32$  битов, которая представлена в табл. 3.116.

**Таблица 3.116**

D0C19225	A5A2240A	1B84D250	B728A4A1
6A704902	85DDDBE6	766FF4A4	ECDFE128
AFD13E94	DF837D09	BB27FA52	695059AC
52A1BB58	CC322F1D	1844565B	B4A8ACF6
34235438	6847A851	E48C0CBB	CD181136
9A112A0C	43EC6D0E	87D8D27D	487DC995
90FB9B4B	A1F63697	FC513ED9	78A37D93
8D16C5DF	9E0C8BBE	3C381F7C	E9FB0779

4. На первое из 32-битных значений, полученных после обработки субблоков операцией  $M()$ , с помощью логической побитовой операции «и» ( $\&$ ) накладывается маскирующая константа (на рис. 3.185 обозначенная  $M$ ). На второе значение аналогичным образом накладывается константа  $M'$  — побитовый комплемент к  $M$ .
5. Два выходных значения (обозначим их  $Out1$  и  $Out2$ ) получаются в результате выполнения следующих действий:

$$Out1 = (V1 \& M) \oplus V2;$$

$$Out2 = (V2 \& M') \oplus V1,$$

где  $V1$  и  $V2$  — значения, полученные после обработки 32-битных субблоков операцией  $M()$ .

Как было сказано выше, в алгоритме выполняются 7 раундов преобразований при использовании 128-битных ключей или 8 раундов при использовании ключей размером 192 или 256 битов. В каждом раунде используется по 8 различных 32-битных фрагментов расширенного ключа. Раунды незначительно различаются между собой в следующем:

- в каждом четном раунде (считая, что раунды нумеруются с 1-го) вместо описанной выше маскирующей константы  $M_5$  используется константа  $M_3 = 33333333$  (шестнадцатеричное значение);
- в последнем раунде не выполняются шаги 6 и 7 описанной выше последовательности действий для первого раунда.

## Расшифровывание

Расшифровывание данных алгоритмом SC2000 весьма похоже на зашифровывание, за исключением следующих моментов:

- фрагменты расширенного ключа применяются в обратной последовательности, т. е. в первом раунде (при использовании 128-битного ключа) операцией XOR накладываются сначала подключи  $k_{52}...k_{55}$ , а затем —  $k_{48}...k_{51}$  и т. д. в следующих раундах расшифровывания;
- маскирующие константы используются в обратной последовательности, т. е.  $M_3$  в нечетных раундах и  $M_5$  — в четных;
- вместо таблицы  $S4$  используется обратная ей таблица, которая выглядит следующим образом (табл. 3.117).

Таблица 3.117

10	6	0	14	12	1	9	4	13	11	2	7	3	8	15	5
----	---	---	----	----	---	---	---	----	----	---	---	---	---	----	---

## Процедура расширения ключа

Расширение ключа выполняется в два этапа: сначала на основе ключа шифрования вычисляется промежуточный ключ, после чего из промежуточного ключа вычисляется требуемое количество фрагментов расширенного ключа.

Прежде всего ключ шифрования используется для инициализации начального ключевого массива  $uk_0...uk_7$  следующим образом:

- 256-битный ключ просто разбивается на 32-битные фрагменты  $uk_0...uk_7$ ;
- 192-битный ключ делится на фрагменты  $uk_0...uk_5$ , два остальных фрагмента инициализируются так:

$$uk_6 = uk_0;$$

$$uk_7 = uk_1.$$

- 128-битный ключ разбивается на фрагменты  $uk_0...uk_3$ , фрагменты  $uk_4...uk_7$  эквивалентны фрагментам  $uk_0...uk_3$ .

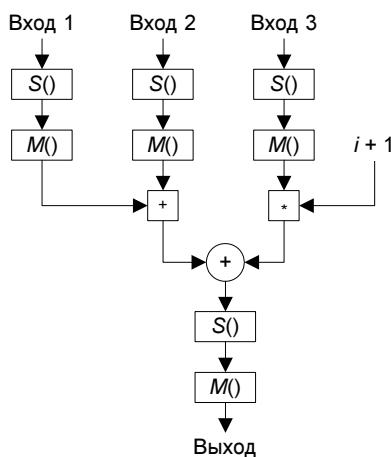


Рис. 3.186. Процедура генерации промежуточных ключей алгоритма SC2000

Затем с помощью процедуры генерации промежуточных ключей, показанной на рис. 3.186, вычисляется набор промежуточных ключей:  $a_0...a_2$ ,  $b_0...b_2$ ,  $c_0...c_2$  и  $d_0...d_2$ . Процедура повторяется 12 раз по количеству вычисляемых промежуточных ключей. Входные и выходные параметры этой процедуры определяются согласно табл. 3.118.

Таблица 3.118

Вход 1	Вход 2	Вход 3	Выход
$4 * i$	$uk_0$	$uk_1$	$a_i, i = 0...2$
$4 * i + 1$	$uk_2$	$uk_3$	$b_i, i = 0...2$
$4 * i + 2$	$uk_4$	$uk_5$	$c_i, i = 0...2$
$4 * i + 3$	$uk_6$	$uk_7$	$d_i, i = 0...2$

Для каждой группы промежуточных ключей в цикле по  $i = 0...2$  процедура выполняет следующие действия:

- Каждый из трех входов обрабатывается функцией  $S()$ , которая осуществляет «прогон» 32-битного фрагмента данных через описанные выше таблицы замен  $S_5$  и  $S_6$  — см. рис. 3.185.
- Результаты предыдущего шага обрабатываются описанной выше функцией  $M()$ .
- Первый из фрагментов, полученных на предыдущем шаге (т. е. результат обработки функциями  $S()$  и  $M()$  первого входа), складывается со вторым по модулю  $2^{32}$ .
- Третий из фрагментов, полученных на шаге 2, умножается на значение  $i + 1$  по модулю  $2^{32}$ .
- Результаты шагов 3 и 4 объединяются операцией XOR.
- Результат предыдущего шага обрабатывается функцией  $S()$ , а затем — функцией  $M()$ , в результате чего получается выходное значение.

После этого из промежуточного ключа вычисляются фрагменты расширенного ключа выполнением в цикле по  $n$  от 0 до 55 (для 192- и 256-битных ключей шифрования — в цикле от 0 до 63) процедуры, показанной на рис. 3.187. В качестве входных значений процедура берет 4 различных фрагмента промежуточного ключа (будет описано далее), над которыми выполняются следующие действия:

- Первое входное значение циклически сдвигается влево на 1 бит.
- Аналогично сдвигается третье входное значение.
- Результат шага 1 складывается со вторым входным значением по модулю  $2^{32}$ .
- Из результата шага 2 вычитается по модулю  $2^{32}$  четвертое входное значение.

## Описание алгоритмов

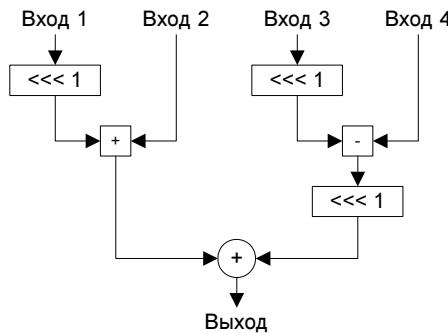


Рис. 3.187. Генерация фрагментов расширенного ключа алгоритма SC2000

5. Результат предыдущего шага циклически сдвигается влево на 1 бит.
6. Результаты шагов 3 и 5 объединяются операцией XOR, что и дает выходное значение.

Выходным значением данной процедуры является  $n$ -й 32-битный фрагмент расширенного ключа. Входные значения определяются следующими таблицами, первая из которых (табл. 3.119) определяет группу промежуточных ключей (т. е.  $a_i$ ,  $b_i$ ,  $c_i$  или  $d_i$ ), из которой берется конкретное входное значение.

Таблица 3.119

$t$	Вход 1	Вход 2	Вход 3	Вход 4
0	$a_i$	$b_i$	$c_i$	$d_i$
1	$b_i$	$a_i$	$d_i$	$c_i$
2	$c_i$	$d_i$	$a_i$	$b_i$
3	$d_i$	$c_i$	$b_i$	$a_i$
4	$a_i$	$c_i$	$d_i$	$b_i$
5	$b_i$	$d_i$	$c_i$	$a_i$
6	$c_i$	$a_i$	$b_i$	$d_i$
7	$d_i$	$b_i$	$a_i$	$c_i$
8	$a_i$	$d_i$	$b_i$	$c_i$
9	$b_i$	$c_i$	$a_i$	$d_i$
10	$c_i$	$b_i$	$d_i$	$a_i$
11	$d_i$	$a_i$	$c_i$	$b_i$

Параметр  $t$  определяется из  $n$  следующим образом:

$$t = n + \lfloor n/36 \rfloor \bmod 12.$$

Вторая таблица (табл. 3.120) определяет индекс  $i$  фрагмента промежуточного ключа для конкретного входа.

**Таблица 3.120**

$n \bmod 9$	Вход 1	Вход 2	Вход 3	Вход 4
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	0	1	0	1
4	1	2	1	2
5	2	0	2	0
6	0	2	0	2
7	1	0	1	0
8	2	1	2	1

## Криптоанализ алгоритма

Алгоритм SC2000 не вышел во второй раунд конкурса NESSIE (см. разд. 2.2) — с одной стороны, у алгоритма достаточная криптостойкость (известны лишь методы вскрытия вариантов алгоритма SC2000 с уменьшенным количеством раундов — например, описанные в [139] и [318]) и неплохое быстродействие. С другой стороны, алгоритм имеет весьма сложную структуру, преимущества которой недостаточно убедительно обоснованы в спецификации алгоритма. Опасаясь скрытых слабостей и возможных недокументированных особенностей алгоритма, эксперты не выбрали этот алгоритм во второй раунд [308].

## 3.48. Алгоритм SERPENT

Алгоритм SERPENT разработан тремя известнейшими криптологами: Россом Андерсоном, Эли Бихамом и Ларсом Кнудсеном. Каждый из них знаменит своими криптоаналитическими работами, а также разработанными ранее алгоритмами шифрования (например, широкую известность получили алгоритмы

Bear и Lion, разработанные Андерсоном и Бихамом (см. разд. 3.6); не менее известен алгоритм Square, разработанный авторами алгоритма Rijndael Джоан Деймен и Винсентом Риджменом при участии Ларса Кнудсена [130]). Именно Эли Бихама без преувеличения можно назвать величайшим криптоаналитиком современности — его авторству (часто в соавторстве с другими специалистами) принадлежит множество работ, посвященных методам вскрытия различных известных алгоритмов шифрования.

Еще до конкурса AES (см. разд. 2.1) появился алгоритм SERPENT-0, отличающийся от присланного на конкурс алгоритма SERPENT-1 (или просто Serpent) только тем, что в нем были использованы таблицы замен алгоритма DES (в незначительно модифицированном виде). В SERPENT используются уже оригинальные таблицы замен, которые, по словам его авторов [35], вкупе с незначительным изменением процедуры расширения ключа усилили алгоритм против дифференциального и линейного криптоанализа.

Рассмотрим здесь именно тот алгоритм SERPENT, который был прислан на конкурс AES и стал одним из его финалистов.

## Структура алгоритма

Алгоритм SERPENT представляет собой SP-сеть, в которой блок данных в процессе шифрования разбивается на 4 субблока по 32 бита (рис. 3.188) [35].

Алгоритм состоит из 32 раундов; перед первым раундом выполняется начальная перестановка *IP*, после заключительного раунда — финальная перестановка *FP*. Начальная перестановка определена согласно табл. 3.121.

Таблица 3.121

0	32	64	96	1	33	65	97	2	34	66	98	3	35	67	99
4	36	68	100	5	37	69	101	6	38	70	102	7	39	71	103
8	40	72	104	9	41	73	105	10	42	74	106	11	43	75	107
12	44	76	108	13	45	77	109	14	46	78	110	15	47	79	111
16	48	80	112	17	49	81	113	18	50	82	114	19	51	83	115
20	52	84	116	21	53	85	117	22	54	86	118	23	55	87	119
24	56	88	120	25	57	89	121	26	58	90	122	27	59	91	123
28	60	92	124	29	61	93	125	30	62	94	126	31	63	95	127

Это означает, что бит 0 остается на своем месте, бит 32 входного значения становится битом 1, бит 64 — битом 2 и т. д.

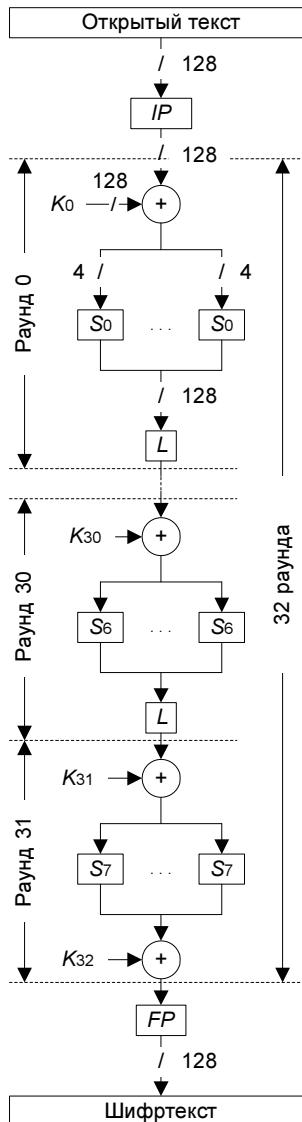


Рис. 3.188. Структура алгоритма SERPENT

Финальная перестановка является инверсной начальной перестановке и представлена в табл. 3.122.

Таблица 3.122

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
1	5	9	13	17	21	25	29	33	37	41	45	49	53	57	61
65	69	73	77	81	85	89	93	97	101	105	109	113	117	121	125
2	6	10	14	18	22	26	30	34	38	42	46	50	54	58	62
66	70	74	78	82	86	90	94	98	102	106	110	114	118	122	126
3	7	11	15	19	23	27	31	35	39	43	47	51	55	59	63
67	71	75	79	83	87	91	95	99	103	107	111	115	119	123	127

Функция раунда весьма проста. Перечислим ее действия:

1. Наложение 128-битного ключа раунда побитовой логической операцией исключающее «или» (XOR).
2. Табличная замена. Обрабатываемый 128-битный блок данных разбивается на 32 фрагмента по 4 бита, над каждым из которых выполняется табличная замена  $4 \times 4$  битов. Полученный результат объединяется обратно в 128-битный блок.

В каждом раунде используется одна и та же таблица; всего же в алгоритме определены 8 таблиц замен  $S_0 \dots S_7$ , которые циклически используются в 32 раундах шифрования (табл. 3.123).

Таблица 3.123

$S_0$	3	8	15	1	10	6	5	11	14	13	4	2	7	0	9	12
$S_1$	15	12	2	7	9	0	5	10	1	11	14	8	6	13	3	4
$S_2$	8	6	7	9	3	12	10	15	13	1	14	4	0	11	5	2
$S_3$	0	15	11	8	12	9	6	3	13	1	2	4	10	7	5	14
$S_4$	1	15	8	3	12	0	11	6	2	5	4	10	9	14	7	13
$S_5$	15	5	2	11	4	10	9	12	0	3	14	8	13	6	7	1
$S_6$	7	2	12	5	8	4	6	11	14	9	1	15	13	3	10	0
$S_7$	1	13	15	0	14	8	2	11	7	4	12	10	9	3	5	6

Ячейки таблицы содержат выходные значения; входное значение определяет требуемый номер столбца. Например, таблица  $S_0$  меняет 0 на 3, 1 на 8 и т. д.

Таблицы замен алгоритма SERPENT определенным образом сгенерированы из таблиц алгоритма DES (см. разд. 3.15). В спецификации алгоритма [35] приведен псевдокод, использованный для генерации таблиц  $S_0 \dots S_7$ .

- Линейное преобразование блока данных, которое представлено в табл. 3.124.

**Таблица 3.124**

16, 52, 56, 70, 83, 94, 105	72, 114, 125	2, 9, 15, 30, 76, 84, 126	36, 90, 103
20, 56, 60, 74, 87, 98, 109	1, 76, 118	2, 6, 13, 19, 34, 80, 88	40, 94, 107
24, 60, 64, 78, 91, 102, 113	5, 80, 122	6, 10, 17, 23, 38, 84, 92	44, 98, 111
28, 64, 68, 82, 95, 106, 117	9, 84, 126	10, 14, 21, 27, 42, 88, 96	48, 102, 115
32, 68, 72, 86, 99, 110, 121	2, 13, 88	14, 18, 25, 31, 46, 92, 100	52, 106, 119
36, 72, 76, 90, 103, 114, 125	6, 17, 92	18, 22, 29, 35, 50, 96, 104	56, 110, 123
1, 40, 76, 80, 94, 107, 118	10, 21, 96	22, 26, 33, 39, 54, 100, 108	60, 114, 127
5, 44, 80, 84, 98, 111, 122	14, 25, 100	26, 30, 37, 43, 58, 104, 112	3, 118
9, 48, 84, 88, 102, 115, 126	18, 29, 104	30, 34, 41, 47, 62, 108, 116	7, 122
2, 13, 52, 88, 92, 106, 119	22, 33, 108	34, 38, 45, 51, 66, 112, 120	11, 126
6, 17, 56, 92, 96, 110, 123	26, 37, 112	38, 42, 49, 55, 70, 116, 124	2, 15, 76
10, 21, 60, 96, 100, 114, 127	30, 41, 116	0, 42, 46, 53, 59, 74, 120	6, 19, 80
3, 14, 25, 100, 104, 118	34, 45, 120	4, 46, 50, 57, 63, 78, 124	10, 23, 84
7, 18, 29, 104, 108, 122	38, 49, 124	0, 8, 50, 54, 61, 67, 82	14, 27, 88
11, 22, 33, 108, 112, 126	0, 42, 53	4, 12, 54, 58, 65, 71, 86	18, 31, 92
2, 15, 26, 37, 76, 112, 116	4, 46, 57	8, 16, 58, 62, 69, 75, 90	22, 35, 96
6, 19, 30, 41, 80, 116, 120	8, 50, 61	12, 20, 62, 66, 73, 79, 94	26, 39, 100
10, 23, 34, 45, 84, 120, 124	12, 54, 65	16, 24, 66, 70, 77, 83, 98	30, 43, 104
0, 14, 27, 38, 49, 88, 124	16, 58, 69	20, 28, 70, 74, 81, 87, 102	34, 47, 108
0, 4, 18, 31, 42, 53, 92	20, 62, 73	24, 32, 74, 78, 85, 91, 106	38, 51, 112
4, 8, 22, 35, 46, 57, 96	24, 66, 77	28, 36, 78, 82, 89, 95, 110	42, 55, 116
8, 12, 26, 39, 50, 61, 100	28, 70, 81	32, 40, 82, 86, 93, 99, 114	46, 59, 120

Таблица 3.124 (окончание)

12, 16, 30, 43, 54, 65, 104	32, 74, 85	36, 90, 103, 118	50, 63, 124
16, 20, 34, 47, 58, 69, 108	36, 78, 89	40, 94, 107, 122	0, 54, 67
20, 24, 38, 51, 62, 73, 112	40, 82, 93	44, 98, 111, 126	4, 58, 71
24, 28, 42, 55, 66, 77, 116	44, 86, 97	2, 48, 102, 115	8, 62, 75
28, 32, 46, 59, 70, 81, 120	48, 90, 101	6, 52, 106, 119	12, 66, 79
32, 36, 50, 63, 74, 85, 124	52, 94, 105	10, 56, 110, 123	16, 70, 83
0, 36, 40, 54, 67, 78, 89	56, 98, 109	14, 60, 114, 127	20, 74, 87
4, 40, 44, 58, 71, 82, 93	60, 102, 113	3, 18, 72, 114, 118, 125	24, 78, 91
8, 44, 48, 62, 75, 86, 97	64, 106, 117	1, 7, 22, 76, 118, 122	28, 82, 95
12, 48, 52, 66, 79, 90, 101	68, 110, 121	5, 11, 26, 80, 122, 126	32, 86, 99

Каждая ячейка таблицы соответствует биту выходного значения (от 0 до 127); в ячейке перечислены входные биты, применение к которым операции XOR дает выходное значение. Например:

$$Y_0 = X_{16} \oplus X_{52} \oplus X_{56} \oplus X_{70} \oplus X_{83} \oplus X_{94} \oplus X_{105};$$

$$Y_1 = X_{72} \oplus X_{114} \oplus X_{125} \text{ и т. д.,}$$

где  $X_n$  и  $Y_n$  — соответственно,  $n$ -е биты входного и выходного значения.

Линейное преобразование может быть реализовано как в виде приведенной выше таблицы, так и с помощью следующих вычислений (рис. 3.189):

$$W_0 = W_0 <<< 13;$$

$$W_2 = W_2 <<< 3;$$

$$W_1 = W_1 \oplus W_0 \oplus W_2;$$

$$W_3 = W_3 \oplus W_2 \oplus (W_0 << 3);$$

$$W_1 = W_1 <<< 1;$$

$$W_3 = W_3 <<< 7;$$

$$W_0 = W_0 \oplus W_1 \oplus W_3;$$

$$W_2 = W_2 \oplus W_3 \oplus (W_1 << 7);$$

$$W_0 = W_0 <<< 5;$$

$$W_2 = W_2 <<< 22,$$

где:

- $W_n$  —  $n$ -е 32-битное слово обрабатываемого блока;
- $<<$  и  $<<<$  — соответственно, операции сдвига и циклического сдвига влево на указанное число битов.

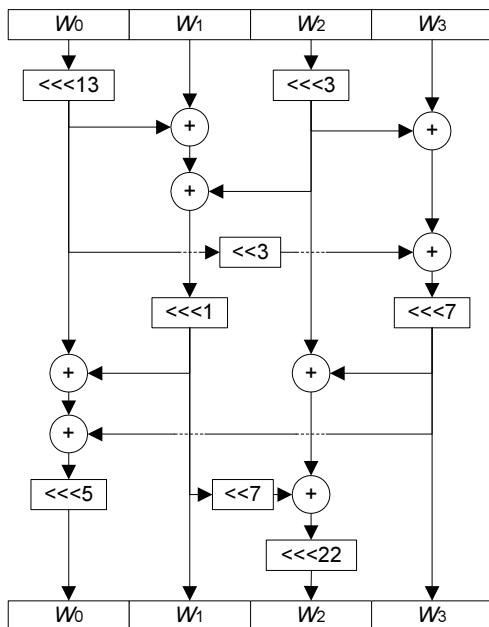


Рис. 3.189. Линейное преобразование в алгоритме SERPENT

Основным критерием при разработке линейного преобразования было максимальное ускорение влияния каждого бита входного значения и ключа шифрования на каждый бит шифртекста. Как пишут авторы алгоритма [35], такое влияние достигается уже после трех раундов алгоритма SERPENT.

Таким образом, преобразования, выполняемые в каждом раунде алгоритма, можно представить следующей формулой:

$$Y = L(S'_{(i \bmod 8)}(X \oplus K_i)),$$

где:

- $X$  и  $Y$  — соответственно, входное и выходное значения обрабатываемого блока;
- $i$  — номер раунда (от 0 до 31);
- $K_i$  — ключ  $i$ -го раунда;

- $L$  — линейное преобразование;
- символом  $S'$  обозначено параллельное использование 32 соответствующих таблиц замен.

Последний раунд отличается от предыдущих тем, что вместо линейного преобразования в нем выполняется дополнительное наложение ключа; здесь используется 33-й фрагмент расширенного ключа  $K_{32}$ :

$$Y = S'_7(X \oplus K_{31}) \oplus K_{32}.$$

## Расшифровывание

Расшифровывание производится выполнением обратных операций в обратной последовательности. В частности, вместо таблиц замен  $S_0...S_7$  применяются в обратной последовательности инверсные таблицы замен  $InvS_0...InvS_7$  (табл. 3.125).

Таблица 3.125

$InvS_0$	13	3	11	0	10	6	5	12	1	14	4	7	15	9	8	2
$InvS_1$	5	8	2	14	15	6	12	3	11	4	7	9	1	13	10	0
$InvS_2$	12	9	15	4	11	14	1	2	0	3	6	13	5	8	10	7
$InvS_3$	0	9	10	7	11	14	6	13	3	5	12	2	4	8	15	1
$InvS_4$	5	0	8	3	10	9	7	14	2	12	11	6	4	15	13	1
$InvS_5$	8	15	2	9	4	1	13	14	11	6	5	3	7	12	10	0
$InvS_6$	15	10	1	13	5	3	6	0	4	9	14	7	2	12	8	11
$InvS_7$	3	0	6	13	9	14	15	8	5	12	11	7	10	1	4	2

Аналогичным образом вместо «прямого» линейного преобразования используется обратное, которое приведено в табл. 3.126 (принцип действия обратного преобразования аналогичен прямому и описан выше).

Таблица 3.126

53, 55, 72	1, 5, 20, 90	15, 102	3, 31, 90
57, 59, 76	5, 9, 24, 94	19, 106	7, 35, 94
61, 63, 80	9, 13, 28, 98	23, 110	11, 39, 98

**Таблица 3.126** (окончание)

65, 67, 84	13, 17, 32, 102	27, 114	1, 3, 15, 20, 43, 102
69, 71, 88	17, 21, 36, 106	1, 31, 118	5, 7, 19, 24, 47, 106
73, 75, 92	21, 25, 40, 110	5, 35, 122	9, 11, 23, 28, 51, 110
77, 79, 96	25, 29, 44, 114	9, 39, 126	13, 15, 27, 32, 55, 114
81, 83, 100	1, 29, 33, 48, 118	2, 13, 43	1, 17, 19, 31, 36, 59, 118
85, 87, 104	5, 33, 37, 52, 122	6, 17, 47	5, 21, 23, 35, 40, 63, 122
89, 91, 108	9, 37, 41, 56, 126	10, 21, 51	9, 25, 27, 39, 44, 67, 126
93, 95, 112	2, 13, 41, 45, 60	14, 25, 55	2, 13, 29, 31, 43, 48, 71
97, 99, 116	6, 17, 45, 49, 64	18, 29, 59	6, 17, 33, 35, 47, 52, 75
101, 103, 120	10, 21, 49, 53, 68	22, 33, 63	10, 21, 37, 39, 51, 56, 79
105, 107, 124	14, 25, 53, 57, 72	26, 37, 67	14, 25, 41, 43, 55, 60, 83
0, 109, 111	18, 29, 57, 61, 76	30, 41, 71	18, 29, 45, 47, 59, 64, 87
4, 113, 115	22, 33, 61, 65, 80	34, 45, 75	22, 33, 49, 51, 63, 68, 91
8, 117, 119	26, 37, 65, 69, 84	38, 49, 79	26, 37, 53, 55, 67, 72, 95
12, 121, 123	30, 41, 69, 73, 88	42, 53, 83	30, 41, 57, 59, 71, 76, 99
16, 125, 127	34, 45, 73, 77, 92	46, 57, 87	34, 45, 61, 63, 75, 80, 103
1, 3, 20	38, 49, 77, 81, 96	50, 61, 91	38, 49, 65, 67, 79, 84, 107
5, 7, 24	42, 53, 81, 85, 100	54, 65, 95	42, 53, 69, 71, 83, 88, 111
9, 11, 28	46, 57, 85, 89, 104	58, 69, 99	46, 57, 73, 75, 87, 92, 115
13, 15, 32	50, 61, 89, 93, 108	62, 73, 103	50, 61, 77, 79, 91, 96, 119
17, 19, 36	54, 65, 93, 97, 112	66, 77, 107	54, 65, 81, 83, 95, 100, 123
21, 23, 40	58, 69, 97, 101, 116	70, 81, 111	58, 69, 85, 87, 99, 104, 127
25, 27, 44	62, 73, 101, 105, 120	74, 85, 115	3, 62, 73, 89, 91, 103, 108
29, 31, 48	66, 77, 105, 109, 124	78, 89, 119	7, 66, 77, 93, 95, 107, 112
33, 35, 52	0, 70, 81, 109, 113	82, 93, 123	11, 70, 81, 97, 99, 111, 116
37, 39, 56	4, 74, 85, 113, 117	86, 97, 127	15, 74, 85, 101, 103, 115, 120
41, 43, 60	8, 78, 89, 117, 121	3, 90	19, 78, 89, 105, 107, 119, 124
45, 47, 64	12, 82, 93, 121, 125	7, 94	0, 23, 82, 93, 109, 111, 123
49, 51, 68	1, 16, 86, 97, 125	11, 98	4, 27, 86, 97, 113, 115, 127

## Процедура расширения ключа

Согласно требованиям конкурса AES, алгоритм SERPENT поддерживает три размера ключа шифрования: 128, 192 и 256 битов. Однако фактически ключ алгоритма SERPENT может иметь произвольный размер, меньший 256. Такой «неполный» ключ перед выполнением его расширения дополняется по следующему правилу:

1. Ключ дополняется одним единичным битом справа.
2. Затем ключ дополняется справа нулевыми битами до достижения 256-битного размера.

Только после этого производится процедура расширения ключа, состоящая из следующих шагов:

1. 256-битный дополненный (при необходимости) ключ шифрования представляется в виде 8 32-битных слов, обозначаемых  $w_{-8}...w_{-1}$ .
2. Эти слова используются в качестве исходных значений для вычисления промежуточного ключа — последовательности  $w_0...w_{131}$ :

$$w_i = (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) \lll 11,$$

где  $\phi$  — округленная дробная часть золотого сечения  $(\sqrt{5}+1)/2$ , т. е. 9E3779B9 в шестнадцатеричной записи.

3. Вычисляются ключи раундов с использованием таблиц замен  $S_0...S_7$  (аналогично описанному выше их использованию в алгоритме SERPENT):

$$K_0 = S_3(\{w_0, w_1, w_2, w_3\});$$

$$K_1 = S_2(\{w_4, w_5, w_6, w_7\});$$

$$K_2 = S_1(\{w_8, w_9, w_{10}, w_{11}\});$$

$$K_3 = S_0(\{w_{12}, w_{13}, w_{14}, w_{15}\});$$

$$K_4 = S_7(\{w_{16}, w_{17}, w_{18}, w_{19}\});$$

$$K_5 = S_6(\{w_{20}, w_{21}, w_{22}, w_{23}\});$$

$$K_6 = S_5(\{w_{24}, w_{25}, w_{26}, w_{27}\});$$

$$K_7 = S_4(\{w_{28}, w_{29}, w_{30}, w_{31}\});$$

...

$$K_{31} = S_4(\{w_{124}, w_{125}, w_{126}, w_{127}\});$$

$$K_{32} = S_3(\{w_{128}, w_{129}, w_{130}, w_{131}\}).$$

## Криптостойкость алгоритма

В процессе исследований эксперты не обнаружили каких-либо криптоаналитических атак на полнорундовую версию алгоритма SERPENT. Это спрашивливо и в отношении остальных алгоритмов — финалистов конкурса AES. Однако данные алгоритмы значительно отличаются по своим характеристикам — см. разд. 2.1.

### 3.49. Алгоритм SHACAL

Алгоритмы семейства SHACAL имеют весьма интересную структуру — они основаны на преобразованиях, используемых функциями хэширования семейства SHA (Secure Hash Algorithm). Алгоритмы SHA являются основой американского стандарта хэширования SHS (Secure Hash Standard) [152], в связи с чем эти алгоритмы получили широчайшее распространение.

Подобное «родство» алгоритма симметричного шифрования и алгоритма хэширования в данном случае интересно, в частности, по следующим соображениям.

- Алгоритмы SHACAL основаны на очень тщательно исследованных криптоаналитиками алгоритмах SHA. Действительно, за годы, в течение которых SHA являются стандартом хэширования (с 1993 г.), данные алгоритмы были весьма тщательно исследованы. И можно утверждать, что семейство SHACAL «унаследовало» результаты криптоанализа алгоритмов SHA.
- Достаточно часто алгоритмы хэширования используются вместе с алгоритмами шифрования. Алгоритм семейства SHACAL и соответствующий ему алгоритм SHA могут быть весьма просто реализованы «в паре», поскольку они разделяют один и тот же набор функций.

Семейство SHACAL разработано двумя специалистами французской корпорации Gemplus, которая известна как один из крупнейших в мире производителей смарт-карт и оборудования для работы с ними. Авторы алгоритма: Хелена Хандшух и Давид Насаш (David Naccache).

### Алгоритм SHACAL-1

Алгоритм SHACAL-1 (или просто SHACAL) был изначально предложен на конкурс NESSIE (см. разд. 2.2) [173]. Он основан на преобразованиях алгоритма SHA-1. Эта книга не содержит описания всех функций SHA (подробное описание можно найти в [152]) — здесь описаны только те преобразования, которые используются непосредственно в алгоритме шифрования.

Итак, SHACAL-1 шифрует 160-битный блок данных с использованием 512-битного ключа шифрования. Допускается использование более коротких ключей шифрования (не короче 128 битов), которые перед выполнением расширения ключа (процедура расширения ключа также унаследована от SHA-1 и будет описана далее) должны быть дополнены нулевыми битами для достижения 512-битного размера.

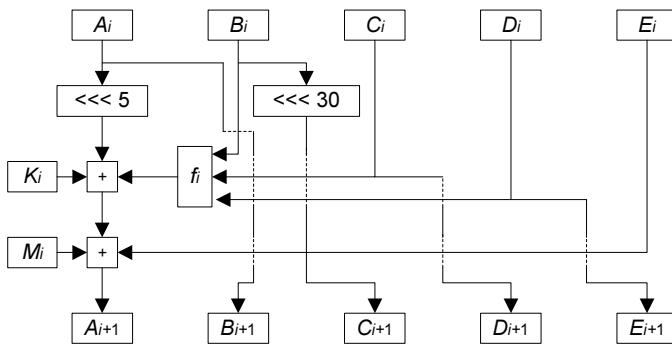


Рис. 3.190. Раунд алгоритма SHACAL-1

В алгоритме SHACAL-1 предусмотрено 80 раундов шифрования. Шифруемое сообщение представляется в виде пяти 32-битных субблоков  $A, B, C, D$  и  $E$ , над которыми в каждом раунде выполняются следующие действия (рис. 3.190):

$$A_{i+1} = K_i + (A_i \lll 5) + f_i(B_i, C_i, D_i) + E_i + M_i \bmod 2^{32};$$

$$B_{i+1} = A_i;$$

$$C_{i+1} = B_i \lll 30;$$

$$D_{i+1} = C_i;$$

$$E_{i+1} = D_i,$$

где:

- $i$  — номер раунда ( $i = 0 \dots 79$ );
- $K_i$  — фрагмент расширенного ключа для  $i$ -го раунда;
- $f_i$  — функция для  $i$ -го раунда (см. далее);
- $\lll$  — операция побитового циклического сдвига влево;
- $M_i$  — модифицирующие константы, приведенные в табл. 3.127 (указаны шестнадцатеричные значения).

Таблица 3.127

Раунды	Значение константы
0...19	5A827999
20...39	6ED9EBA1
40...59	8F1BBCDC
60...79	CA62C1D6

Используемые в раундах функции  $f_i$  определены согласно табл. 3.128.

Таблица 3.128

Раунды	Функция
0...19	$f(x, y, z) = (x \& y)   (x' \& z)$
20...39, 60...79	$f(x, y, z) = x \oplus y \oplus z$
40...59	$f(x, y, z) = (x \oplus y)   (x \oplus z)   (y \oplus z)$

В таблице символами  $\&$ ,  $|$  и  $\oplus$  обозначены, соответственно, побитовые логические операции «и», «или» и исключающее «или» (XOR);  $x'$  обозначает побитовый комплемент к  $x$ .

Шифртекстом является конкатенация содержимого переменных  $A_{80}$ ,  $B_{80}$ ,  $C_{80}$ ,  $D_{80}$  и  $E_{80}$ .

Процедура расширения ключа в алгоритме SHACAL-1 также весьма проста, она выполняется в два этапа:

1. 512-битный исходный ключ шифрования делится на 16 фрагментов по 32 бита  $K_0 \dots K_{15}$ .
2. Остальные фрагменты расширенного ключа  $K_{16} \dots K_{79}$  вычисляются из первых 16 фрагментов следующим образом:

$$K_i = (K_{i-3} \oplus K_{i-8} \oplus K_{i-14} \oplus K_{i-16}) \lll 1.$$

Ни в [173], ни в [174] авторы алгоритма не описали процесс расшифровывания алгоритмом SHACAL. Такое описание можно найти, например, в работе [202]. Согласно этому описанию, раунды расшифровывания выполняются в обратной последовательности; в каждом из них производятся следующие операции (рис. 3.191):

$$A_i = B_{i+1};$$

$$B_i = C_{i+1} \lll 2;$$

$$\begin{aligned}
 C_i &= D_{i+1}; \\
 D_i &= E_{i+1}; \\
 E_i &= K'_i + (B_{i+1} \lll 5) + f'_i(C_{i+1} \lll 2, D_{i+1}, E_{i+1}) + A_{i+1} + M'_i + 4 \bmod 2^{32}.
 \end{aligned}$$

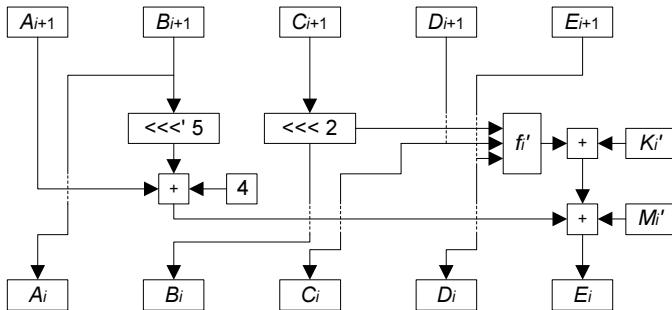


Рис. 3.191. Раунд расшифровывания алгоритма SHACAL-1

Здесь и далее запись  $f'(x)$  обозначает побитовый комплемент результата выполнения операции  $f(x)$ .

Поражает простота структуры алгоритма SHACAL: достаточно сравнить его с другими победителями конкурса NESSIE — алгоритмами MISTY1 (см. разд. 3.36) и Camellia (см. разд. 3.9). Из этого вытекают следующие достоинства данного алгоритма:

- простота реализации;
- легкая доказуемость отсутствия каких-либо внедренных авторами недокументированных возможностей.

Кроме того, стоит отметить, что SHACAL предъявляет весьма низкие требования к ресурсам и обладает высоким быстродействием. Высокая скорость шифрования данным алгоритмом на всех тестируемых платформах была отмечена, например, в [311]. А авторы [262] описали архитектуру шифратора, выполняющего шифрование по алгоритму SHACAL-1 со скоростью 17 Гбит в секунду, что вряд ли достижимо (в разумных ценовых пределах) для подавляющего большинства других алгоритмов шифрования.

## Алгоритм SHACAL-0

Стоит отметить, что в некоторых работах (например, в [400]) наряду с алгоритмом SHACAL-1 упоминается некий алгоритм SHACAL-0. Видимо, имеется в виду возможный (не предложенный авторами семейства SHACAL) вариант,

основанный на функции SHA-0. Такой вариант отличался бы от SHACAL-1 только процедурой расширения ключа, в которой в описанной выше формуле вычисления  $K_i$  отсутствовал бы циклический сдвиг влево на 1 бит [173].

## Алгоритм SHACAL-2

В 2001 г. авторы алгоритма SHACAL-1 воспользовались тем, что в ходе конкурса NESSIE можно было вносить некоторые незначительные изменения в алгоритм, и представили новый вариант алгоритма SHACAL — SHACAL-2 [174].

Стоит сказать, что в данном случае различия между алгоритмами SHACAL-1 и SHACAL-2 сложно считать несущественными. SHACAL-2 основан на преобразованиях более нового алгоритма хэширования — SHA-256 [152], который принципиально отличается от SHA-1.

Алгоритм SHACAL-2 шифрует данные 256-битными блоками с использованием 512-битного ключа. Аналогично алгоритму SHACAL-1, допускается использование ключей меньших размеров (не менее 128 битов), которые дополняются битовыми нулями до 512 битов.

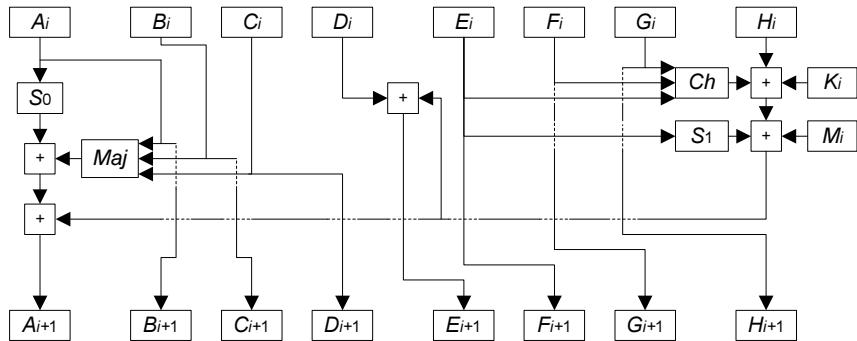


Рис. 3.192. Раунд алгоритма SHACAL-2

Шифруемый блок данных делится на 8 фрагментов по 32 бита (которые обозначены буквами  $A \dots H$ ). Алгоритм выполняет 64 раунда преобразований, в каждом из которых эти фрагменты обрабатываются следующим образом (рис. 3.192):

$$T = H_i + S_1(E_i) + Ch(E_i, F_i, G_i) + M_i + K_i \bmod 2^{32};$$

$$H_{i+1} = G_i;$$

$$G_{i+1} = F_i;$$

## Описание алгоритмов

$$\begin{aligned}
 F_{i+1} &= E_i; \\
 E_{i+1} &= D_i + T; \\
 D_{i+1} &= C_i; \\
 C_{i+1} &= B_i; \\
 B_{i+1} &= A_i; \\
 A_{i+1} &= T + S_0(A_i) + \text{Maj}(A_i, B_i, C_i) \bmod 2^{32},
 \end{aligned}$$

где  $T$  — временная переменная.

Используемые функции определены следующим образом:

$$\begin{aligned}
 S_0(x) &= (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22); \\
 S_1(x) &= (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25); \\
 Ch(x, y, z) &= (x \& y) \oplus (x' \& z); \\
 \text{Maj}(x, y, z) &= (x \& y) \oplus (x \& z) \oplus (y \& z),
 \end{aligned}$$

где  $\ggg$  — операция побитового циклического сдвига вправо.

Модифицирующие константы  $M_i$  ( $i = 0 \dots 63$ ) приведены в табл. 3.129 (по порядку от  $M_0$  до  $M_{63}$ ).

**Таблица 3.129**

428a2f98	71374491	b5c0fbcf	e9b5dba5
3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3
72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc
2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7
c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13
650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3
d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5
391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208
90beffff	a4506ceb	bef9a3f7	c67178f2

Фрагменты расширенного ключа  $K_0 \dots K_{63}$  вычисляются в процессе процедуры расширения ключа, которая выполняется следующим образом:

1. Аналогично алгоритму SHACAL-1, 512-битный исходный ключ шифрования делится на 16 фрагментов по 32 бита  $K_0 \dots K_{15}$ .
2. Остальные фрагменты расширенного ключа  $K_{16} \dots K_{63}$  вычисляются из первых 16 фрагментов следующим образом:

$$K_i = O_1(K_{i-2}) + K_{i-7} + O_0(K_{i-15}) + K_{i-16},$$

где функции  $O_0$  и  $O_1$  определены так:

$$O_0(x) = (x >> 7) \oplus (x >> 18) \oplus (x > 3);$$

$$O_1(x) = (x >> 17) \oplus (x >> 19) \oplus (x > 10),$$

где  $>>$  — операция побитового сдвига (не циклического) вправо.

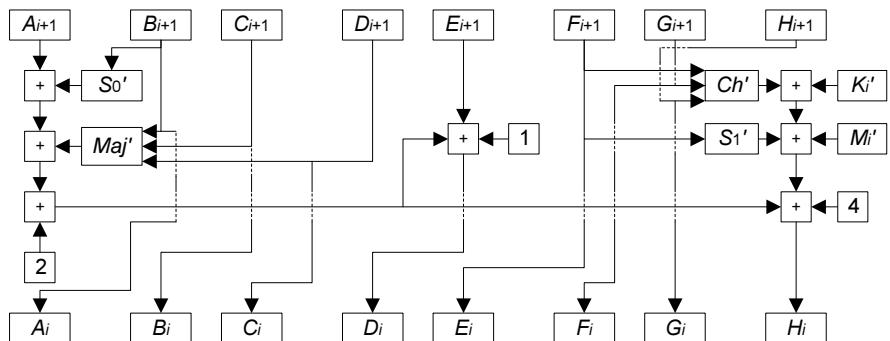


Рис. 3.193. Раунд расшифровывания алгоритма SHACAL-2

Аналогично алгоритму SHACAL-1, авторы SHACAL-2 в [174] не описали процесс расшифровывания. Такое описание можно найти в работе [202]. Раунды расшифровывания алгоритма выполняются в обратной последовательности; в каждом из них производятся следующие действия (рис. 3.193):

$$T = A_{i+1} + S_0'(B_{i+1}) + \text{Maj}'(B_{i+1}, C_{i+1}, D_{i+1}) + 2 \bmod 2^{32};$$

$$H_i = T + S_1'(F_{i+1}) + \text{Ch}'(F_{i+1}, G_{i+1}, H_{i+1}) + M_i' + K_i' + 4 \bmod 2^{32};$$

$$G_i = H_{i+1}, \quad F_i = G_{i+1}, \quad E_i = F_{i+1};$$

$$D_i = E_{i+1} + T + 1 \bmod 2^{32};$$

$$C_i = D_{i+1}; \quad B_i = C_{i+1}; \quad A_i = B_{i+1}.$$

## Криптоанализ алгоритма SHACAL-1

Первичный криптоанализ алгоритма SHACAL-1 был выполнен авторами алгоритма в [172] и [173], где была отмечена стойкость алгоритма к линейному и дифференциальному криптоанализу.

В дальнейшем, как отмечено авторами [67], SHACAL подвергся массированному криптоанализу со стороны ряда экспертов. Из криптоаналитических исследований данного алгоритма стоит отметить следующие.

- В работе [335] описаны слабости процедуры расширения ключа алгоритма SHACAL-1, с помощью которых можно атаковать данный алгоритм с использованием связанных ключей.
- Ряд экспертов из университета г. Сеул, Корея, предложили атаку усовершенствованным методом бумеранга на SHACAL-1 с различными размерами ключей шифрования. Основные результаты приведены в табл. 3.130 [203].

Таблица 3.130

Размер ключа в битах	Количество раундов	Требуемое количество выбранных открытых текстов	Количество тестовых операций шифрования
160	37	$2^{158,8}$	$2^{87,8}$
256	39	$2^{158,5}$	$2^{250,8}$
512	47	$2^{158,5}$	$2^{508,4}$

- В работе [69] были скорректированы результаты предыдущей работы: предложенная атака на вариант с 512-битным ключом распространена на 49 раундов, для ее осуществления требуется  $2^{151,9}$  выбранных открытых текстов или выбранных шифртекстов и  $2^{508,5}$  операций.
- Уже после конкурса NESSIE, в 2006 г. вышла работа [242], в которой предложена лучшая на текущий момент атака на алгоритм SHACAL-1 из атак, не использующих связанные ключи. Атака на 55 раундов SHACAL-1 с 512-битным ключом использует дифференциальный криптоанализ, она успешна при наличии  $2^{154}$  выбранных шифртекстов и при выполнении  $2^{507,26}$  операций шифрования. В данной работе предложены и некоторые другие варианты атак на SHACAL-1.
- В том же году авторы работы [140] сделали вывод, что, в связи с «наследственностью» SHACAL-1 от SHA-1, новые результаты в поиске коллизий в SHA-1 будут выливаться в новые атаки на алгоритм SHACAL-1 на свя-

занных ключах, которые действуют, как минимум, на класс слабых ключей. В данной работе была предложена первая атака на полнораундовый SHACAL-1 с 512-битным ключом с использованием 4-х связанных ключей. Для этой атаки требуется  $2^{159,8}$  выбранных открытых текстов и  $2^{423}$  операций шифрования или  $2^{153,8}$  выбранных открытых текстов и  $2^{504,2}$  операций. Авторы отмечают, что атака является абсолютно теоретической, поскольку требуемые для атаки ресурсы безусловно превышают существующие вычислительные возможности.

- А в работе [67], появившейся совсем недавно, предыдущая атака была усилена: авторами работы предложено несколько вариантов атак, одна из которых требует наличия  $2^{101,3}$  выбранных открытых текстов, зашифрованных на 8 связанных ключах, и выполнения  $2^{101,3}$  операций. По сравнению с предыдущей атакой предложенная в [67] атака выглядит существенно более практической, что позволило ее авторам сделать вывод о структуре алгоритма SHACAL-1: «Использование линейной процедуры расширения ключа совместно с раундами схожей структуры — не лучший способ создания стойких блочных шифров».

Кроме того, в работе [294] утверждается, что алгоритм SHACAL относительно сложно защищать от атак по потребляемой мощности.

По результатам первого этапа конкурса NESSIE алгоритм SHACAL-1 был выбран в финал конкурса. Эксперты отметили отсутствие уязвимостей, а также высокое быстродействие алгоритма. Кроме того, интересной показалась возможность интеграции алгоритмов SHACAL-1 и SHA-1, о которой было сказано выше [308].

Однако в финале конкурса SHACAL-1 не оказался в числе алгоритмов-победителей из-за сомнений экспертов в стойкости его процедуры расширения ключа [305].

## Криptoанализ алгоритма SHACAL-2

В рамках конкурса NESSIE не было найдено каких-либо уязвимостей в алгоритме SHACAL-2. Мало того, во время конкурса не были известны какие-либо атаки даже на усеченные версии алгоритма [307]. Алгоритм SHACAL-2 стал одним из победителей конкурса NESSIE [305], причем, помимо криптоустойчивости, эксперты отметили и высочайшую скорость шифрования этим алгоритмом [308].

Уже после окончания конкурса начали появляться различные работы, посвященные атакам на варианты алгоритма SHACAL-2 с уменьшенным количеством раундов. Например, в работе [243] предложена атака на 42-раундовый

вариант SHACAL-2 с 512-битным ключом, для которой требуется наличие  $2^{243,38}$  выбранных открытых текстов на двух связанных ключах и  $2^{488,37}$  операций.

В работе [202] описаны другие варианты атак на SHACAL-2, однако они еще менее эффективны, чем упомянутая выше атака на 42 раунда алгоритма.

Алгоритмы симметричного шифрования, основанные на хэш-функциях, были предложены различными криптологами задолго до появления алгоритма SHACAL (например, алгоритмы Bear, Lion и Lioness — см. разд. 3.6). Однако все они имели те или иные проблемы с криптостойкостью и/или быстродействием. Таким образом, на сегодняшний день алгоритм SHACAL-2 является единственным широко известным алгоритмом (из основанных на хэш-функциях), не имеющим проблем с криптостойкостью и обладающим высокой скоростью шифрования.

### 3.50. Алгоритмы SHARK и SHARK\*

Как и Square (см. разд. 3.54), алгоритм SHARK разработан Винсентом Риджменом и Джоан Деймен — будущими авторами стандарта AES (алгоритма Rijndael — см. разд. 3.3), правда, в соавторстве с еще тремя специалистами, представляющими Католический Университет г. Лювен, Бельгия. Это немногим более ранняя разработка, чем алгоритм Square, — SHARK разработан в 1995 г.

Сходство между SHARK и Square наблюдается, как минимум, в следующем:

- оба алгоритма обрабатывают за один раунд блок целиком, а не половину, как сети Фейстеля;
- в раунде алгоритма SHARK используются преобразования, весьма похожие и на Square, и на Rijndael — операция XOR с ключом, табличная замена и умножение на фиксированную матрицу.

Спецификация алгоритма [323] не фиксирует его основные параметры. В частности, блок шифруемых данных имеет переменный размер  $m * n$  битов (значение параметров  $m$  и  $n$  будет ясно из приведенного далее описания раунда алгоритма), а обработка данных выполняется с переменным количеством раундов  $R$ .

В каждом раунде алгоритма  $r$  выполняются следующие операции (рис. 3.194):

1. Наложение ключа раунда  $K_r$  на обрабатываемый блок операцией XOR. Ключ раунда также имеет размер  $m * n$  битов; подробнее о вычислении ключей раундов будет сказано далее.

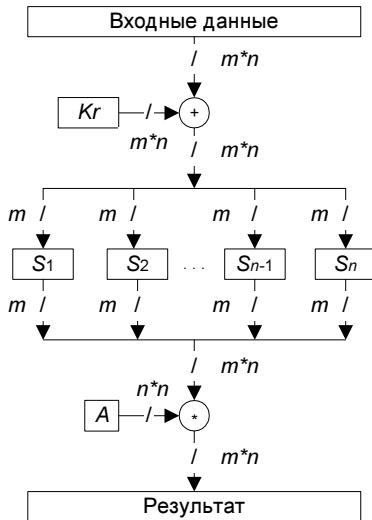


Рис. 3.194. Раунд алгоритма SHARK

2. Табличная замена, выполняемая следующим образом:

- ( $m * n$ ) -битный блок данных разбивается на  $n$  субблоков по  $m$  битов;
- каждый из них «прогоняется» через одну из таблиц замен  $S_1 \dots S_n$ ;
- результаты замен объединяются в ( $m * n$ ) -битный блок.

3. Умножение данных на фиксированную матрицу  $A$  размером  $n * n$  битов.

Обрабатываемые данные можно представить в виде двумерного байтового массива размером  $(m/8)*n$  байтов (аналогично алгоритму Square) или в виде одномерного массива из  $n$   $m$ -битных элементов. В последнем случае выполняемые в одном раунде операции можно представить таким образом:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \dots \\ Y_n \end{bmatrix} = A * \begin{bmatrix} S_1[X_1 \oplus Kr_1] \\ S_2[X_2 \oplus Kr_2] \\ \dots \\ S_n[X_n \oplus Kr_n] \end{bmatrix},$$

где  $X_1 \dots X_n$ ,  $Y_1 \dots Y_n$ ,  $Kr_1 \dots Kr_n$  —  $m$ -битные элементы входного значения, результата и ключа раунда соответственно.

После выполнения  $r$  раундов алгоритма выполняется также финальное преобразование, состоящее из наложения ключа дополнительного раунда  $K_{R+1}$  и последующего умножения на обратную матрицу.

Расшифровывание производится выполнением обратных операций в обратной последовательности.

Фактически SHARK представляет собой не алгоритм шифрования, а некий шаблон для построения на его основе различных алгоритмов шифрования с описанной выше структурой. Такой вывод можно сделать из-за обилия переменных величин в структуре алгоритма; переменными являются следующие параметры:

- описанные выше параметры  $n$ ,  $m$  и  $r$ ;
- значения таблиц замен; в описании алгоритма приводится пример таблицы замен, согласно которой входное значение  $x$  заменяется обратной величиной в поле  $GF(2^m)$ :

$$S(x) = x^{-1} \bmod 2^m;$$

однако авторы алгоритма предположили, что при использовании такой таблицы замен возможны потенциальные уязвимости, поэтому порекомендовали использовать таблицы с более сложным соотношением входного и выходного значений;

- значения элементов матрицы  $A$ ; при этом авторы алгоритма описали ряд критериев, которым должны соответствовать данные элементы для достижения высокой криптостойкости алгоритма;
- размер ключа шифрования, который может достигать  $2 * (R + 1) * m * n$  битов (сумма размеров всех ключей раундов), однако авторы алгоритма посоветовали ограничиться не более, чем 128-битным ключом;
- даже для процедуры расширения ключа авторы алгоритма предложили несколько возможных вариантов.

Исходные тексты одного из вариантов алгоритма SHARK, разработанные Винсентом Риджменом [322], можно скачать по FTP с сервера [ftp.esat.kuleuven.ac.be](ftp://esat.kuleuven.ac.be).

Кроме того, помимо алгоритма SHARK, его авторами был предложен алгоритм SHARK\*, отличающийся от SHARK лишь зависимостью таблиц замен от ключа шифрования и также не снабженный исчерпывающим описанием [323].

Видимо, в силу своей неопределенности алгоритмы SHARK и SHARK\* не вызвали большого интереса у криptoаналитиков, поэтому какие-либо работы, связанные с анализом криптостойкости или атаками на эти алгоритмы, не получили широкую известность.

## 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].

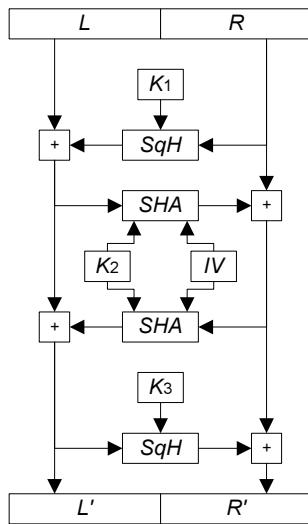


Рис. 3.195. Структура алгоритма Sha-zam

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, больших  $2^{160}$ .

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:

$$s_i = \text{SHA}(s', C'_i),$$

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  $c_i$  ( $c_0 \dots c_m$  — 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  $h(s_1) \dots h(s_m)$ , 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  $s_1 \dots s_m$ .

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 соб-

ственno 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].

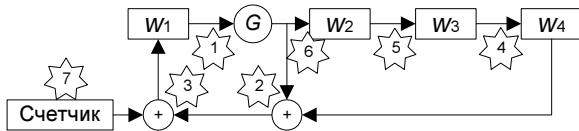
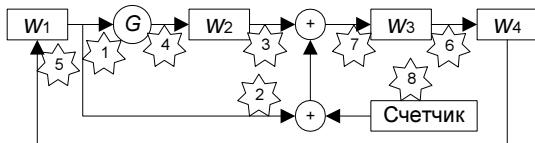


Рис. 3.196. Функция  $A$

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.:

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

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.

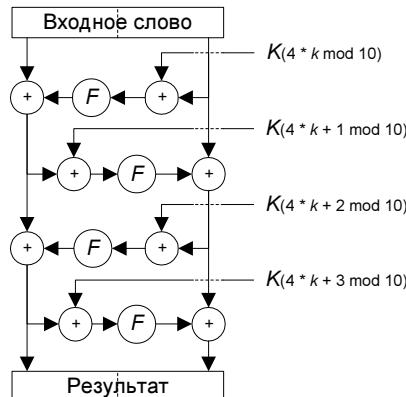


Рис. 3.198. Операция  $G$

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:

$$i = 4 * k + r \text{ mod } 10,$$

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 nibl входного значения определяет номер строки

(начиная с нуля), а младший nibл — 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	I2	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 криptoаналитическими методами, включая метод дифференциального криptoанализа;
- 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 криptoанализ 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;

## Описание алгоритмов

- 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  $P1 \dots P4$  (рис. 3.199). В каждой фазе выполняется  $R/4$  раундов преобразований. Структура раунда приведена на рис. 3.200.

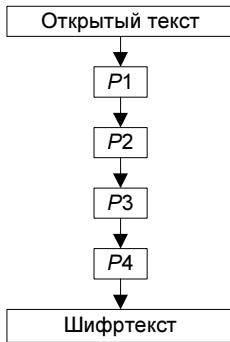


Рис. 3.199. Структура алгоритма

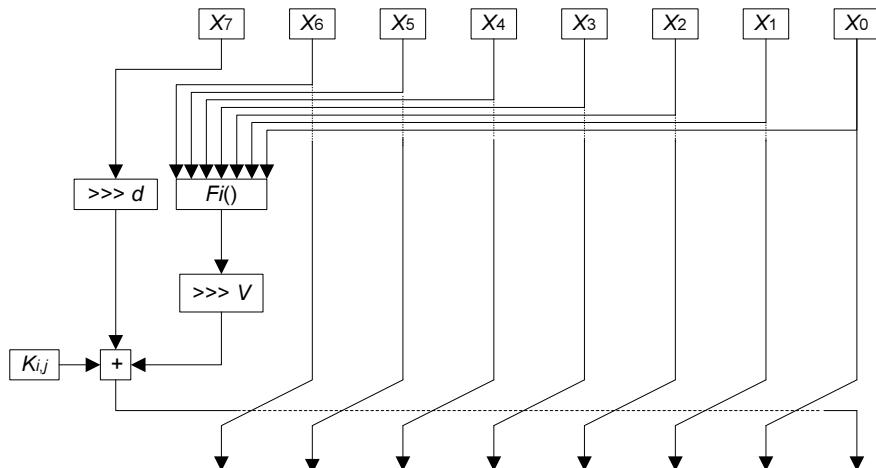


Рис. 3.200. Раунд алгоритма

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

- Фрагменты  $X_0 \dots X_6$  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:

$$F1(X_0 \dots X_6) = X_6X_3 \oplus X_5X_1 \oplus X_4X_2 \oplus X_1X_0 \oplus X_0;$$

$$F2(X_0 \dots X_6) = X_6X_4X_0 \oplus X_4X_3X_0 \oplus X_5X_2 \oplus X_4X_3 \oplus X_4X_1 \oplus X_3X_0 \oplus X_1;$$

$$F3(X_0 \dots X_6) = X_5X_4X_0 \oplus X_6X_4 \oplus X_5X_2 \oplus X_3X_0 \oplus X_1X_0 \oplus X_3;$$

$$F4(X_0 \dots X_6) = X_6X_4X_2X_0 \oplus X_6X_5 \oplus X_4X_3 \oplus X_3X_2 \oplus X_1X_0 \oplus X_2,$$

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

- 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.

- Значение фрагмента  $X_7$  циклически сдвигается вправо на фиксированное число битов  $d$ , которое определено так:

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

- 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.

- Фрагменты  $X_0 \dots X_6$  сдвигаются на 1 фрагмент влево, т. е.:

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

- 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 ( $X_0 \dots X_6$ ) 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.

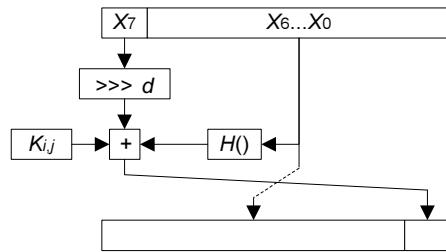


Рис. 3.201. Альтернативное представление раунда алгоритма

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
$L$	$\geq 64$	$\geq 64$	$\geq 64$
$R$	$\geq 64$	$\geq 48$	$\geq 48$

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  $F_i()$   $i = 4 \dots 1$ . 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  $X_0 \dots X_6$  обрабатываются функцией  $F_i()$ .
3. Результат предыдущего шага вращается вправо на  $v$  битов.

4. Побитовый комплемент (на рис. 3.202 операция его получения обозначена как  $\sim$ ) 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  $X_7$ .

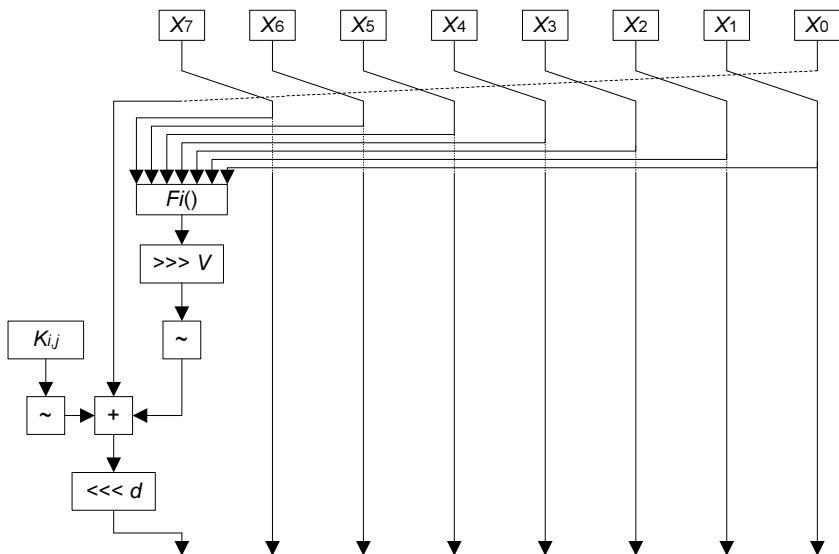


Рис. 3.202. Раунд расшифровывания

## 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 ( $Z$ ) непрямую 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  $S_0 \dots S_2$ , 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  $Q_0 \dots Q_2$  — 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  $Q_0 \dots Q_2$  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  $Q_0 \dots Q_2$  берутся 3 последние константы массива.

Затем на основе проинициализированных элементов массива ( $kb_0 \dots kb_{L/16-1}$ ) формируются остальные его элементы ( $kb_{L/16} \dots kb_{Z-1}$ ). 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):

- Переменные состояния  $S_0 \dots S_2$  обрабатываются функцией  $G()$ , которая определена таким образом:

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

- 16-битный результат предыдущей операции вращается влево на 5 битов.
- Результат предыдущего шага складывается по модулю  $2^{16}$  со значением  $S_2$  и проинициализированным ранее элементом массива  $kb_0 \dots kb_{Z-1}$ , 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  $S_0$ .

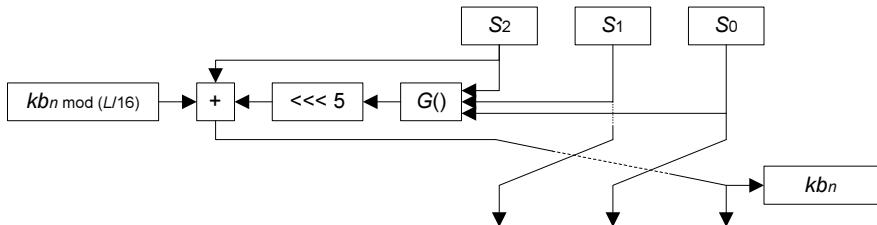


Рис. 3.203. Раунд процедуры расширения ключа

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  $F_i()$ , т. е. от самой 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 известным 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 спрашивливо 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 двухмерного байтового массива (таблицы) размером  $4 \times 4$  — 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  $\theta$ , 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):

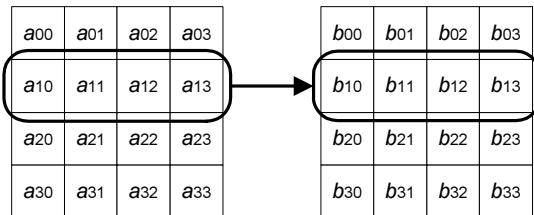
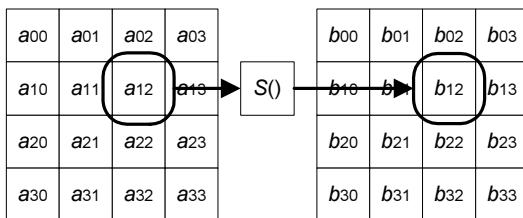
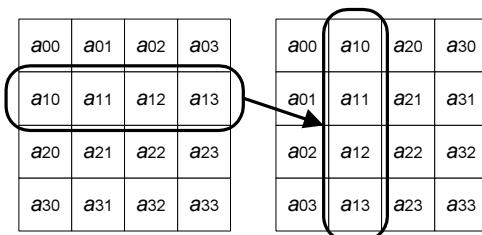
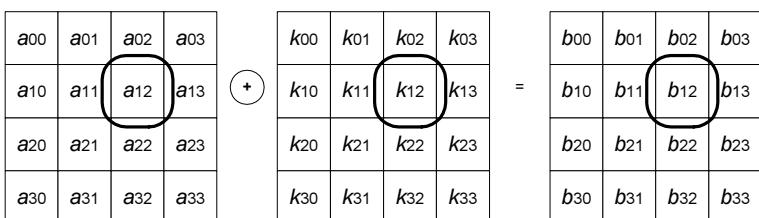
$$\gamma : b_{i,j} = S(a_{i,j}).$$

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

То есть значение 0 заменяется на B1, 1 — на CЕ 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  $\theta$ Рис. 3.205. Операция  $\gamma$ Рис. 3.206. Операция  $\pi$ Рис. 3.207. Операция  $\sigma$

3. Байтовая перестановка  $\pi$ , простейшим образом преобразующая строку состояния в столбец (рис. 3.206):

$$\pi : b_{i,j} = a_{j,i}.$$

4. Операция  $\sigma$ , представляющая собой побитовое сложение состояния с ключом раунда (рис. 3.207):

$$\sigma : b = a \oplus K_t,$$

где:

- $a$  и  $b$  — значение всего массива состояния до и после преобразования соответственно;
- $K_t$  — ключ текущего раунда  $t$  (процедура формирования ключей раундов описана далее).

Помимо 8 раундов описанных преобразований, перед первым раундом выполняется «нулевой» раунд, состоящий из обратного линейного преобразования  $\theta^{-1}$  и наложения ключа нулевого раунда  $K_0$  операцией  $\sigma$ .

Расшифровывание данных выполняется аналогично зашифровыванию, но с использованием обратных операций  $\gamma^{-1}$  и  $\theta^{-1}$  вместо  $\gamma$  и  $\theta$  соответственно. И наоборот, в нулевом раунде используется прямое преобразование  $\theta$  вместо обратного  $\theta^{-1}$ . Операция  $\gamma^{-1}$  представляет собой обратную замену, а  $\theta^{-1}$  предполагает использование вместо констант  $c_n$  набора констант  $d_n$ , приводящих к обратному результату и также определенных в спецификации алгоритма.

## Процедура расширения ключа

Задача процедуры расширения ключа состоит в получении 8 128-битных ключей раундов и ключа  $K_0$  из 128-битного ключа шифрования алгоритма. Расширение ключа выполняется следующим простым преобразованием:

$$K_t = \psi(K_{t-1}).$$

Операция  $\psi$  представляет собой набор функций, с помощью которых вычисляются ключи раундов (рис. 3.208):

$$\begin{aligned} k_{0,t+1} &= k_{0,t} \oplus \text{rotl}(k_{3,t}) \oplus C_t; \\ k_{1,t+1} &= k_{1,t} \oplus k_{0,t+1}; \\ k_{2,t+1} &= k_{2,t} \oplus k_{1,t+1}; \\ k_{3,t+1} &= k_{3,t} \oplus k_{2,t+1}, \end{aligned}$$

где:

- $k_{n,t}$  —  $n$ -я строка (аналогично состоянию, ключ раунда представляется в виде байтовой таблицы  $4 \times 4$ ) ключа  $t$ -го раунда  $K_t$ ;

## Описание алгоритмов

- $C_t$  — набор итеративно вычисляемых констант;
- $rotl()$  — операция циклического сдвига байтовой строки на один байт влево.

В качестве начального значения  $K_0$  используется исходный ключ шифрования алгоритма.

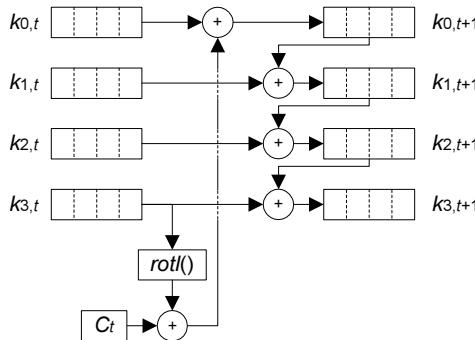


Рис. 3.208. Операция  $\psi$

## Криптостойкость алгоритма

В спецификации алгоритма Square [130] авторы алгоритма привели также строгое математическое обоснование операций алгоритма, которые выбирались, исходя из требований высокой криптостойкости против линейного и дифференциального криптоанализа. Кроме того, там же приведены и возможные атаки на данный алгоритм, наиболее интересная из которых позволяет вскрыть 6-раундовый вариант алгоритма выполнением  $2^{72}$  операций шифрования при наличии  $2^{32}$  блоков открытого текста и соответствующих им блоков шифртекста.

Все приведенные атаки позволяют вскрыть только «урезанные» (с уменьшенным количеством раундов) варианты алгоритма Square. Авторы алгоритма не обнаружили каких-либо атак на полнораундовый алгоритм. Однако они же предостерегают потенциальных пользователей Square от использования алгоритма, не прошедшего полного изучения специалистами на предмет отсутствия уязвимостей.

Наиболее подробное описание алгоритма Square, а также исходные тексты реализующих его программ можно найти на странице Винсента Риджмена, которая находится на сайте <http://www.esat.kuleuven.ac.be> [374].

## 3.55. Алгоритмы TEA, XTEA и их варианты

Одно из наиболее интересных семейств алгоритмов блочного симметричного шифрования — это алгоритмы семейства TEA (Tiny Encryption Algorithm, «крошечный алгоритм шифрования»). Они разработаны в 1990-х гг. Дэвидом Уилером (David J. Wheeler) и Роджером Нидэмом (Roger M. Needham) из компьютерной лаборатории Кембриджского Университета.

Все алгоритмы семейства TEA отличает простота реализации и относительно высокая криптостойкость. Начнем рассмотрение этих алгоритмов с описания алгоритма TEA.

### Алгоритм TEA

Алгоритм TEA шифрует данные 64-битными блоками с использованием 128-битного ключа шифрования. Алгоритм выполняет 64 раунда преобразований (такое количество раундов является не обязательным, а рекомендованным авторами), в каждом из которых предусмотрены следующие операции (рис. 3.209) [393]:

1. В начале каждого четного раунда (раунды нумеруются, начиная с 0) модифицируется временная переменная  $T$  (здесь и далее операции сложения и вычитания производятся по модулю  $2^{32}$ ):

$$T = T + C,$$

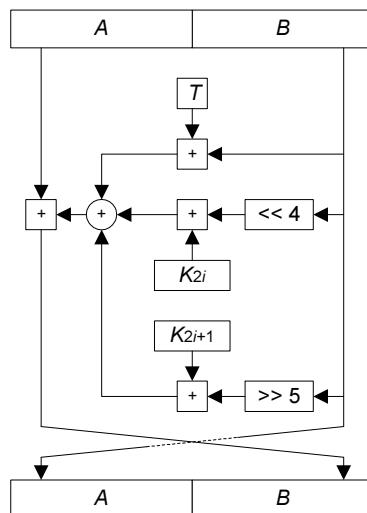


Рис. 3.209. Раунд алгоритма TEA

где:

- $C$  — константа  $(\sqrt{5} - 1) * 2^{31} = 9E3779B9$ ;
- начальное значение переменной  $T$  равно нулю.

2. Обрабатывается правый 32-битный субблок данных  $B$  и накладывается на левый субблок  $A$ :

$$A = A + ((B + T) \oplus ((B \ll 4) + K_{2i}) \oplus ((B \gg 5) + K_{2i+1})),$$

где:

- $\ll$  и  $\gg$  — операции сдвига влево и вправо соответственно на указанное число битов;
- $K_x$  —  $x$ -й фрагмент расширенного ключа (процедура расширения ключа будет описана далее);
- $i$  — номер раунда алгоритма;
- операция сложения выполняется по модулю  $2^{32}$  (это относится и к модификации переменной  $T$ ).

3. Субблоки меняются местами.

Каждые два раунда авторы алгоритма называют «циклом». За один цикл модифицируются оба субблока и временная переменная; алгоритм выполняет 32 цикла.

Процедура расширения ключа у алгоритма TEA фактически отсутствует: в качестве фрагментов расширенного ключа в раундах алгоритма поочередно используются 4 32-битных фрагмента исходного ключа шифрования  $K[0]...K[3]$ , т. е.  $K_x = K[x \bmod 4]$

или

$$K_0 = K[0], K_1 = K[1], K_2 = K[2], K_3 = K[3], K_4 = K[0], K_5 = K[1] \text{ и т. д.}$$

Раунд расшифровывания выглядит следующим образом (рис. 3.210):

1. Обрабатывается значение левого субблока данных и накладывается на правый субблок:

$$B = B - ((A + T) \oplus ((A \ll 4) + K_{2i+2}) \oplus ((A \gg 5) + K_{2i+3})),$$

где операция вычитания выполняется по модулю  $2^{32}$ .

2. В конце каждого нечетного раунда выполняется модификация временной переменной  $T$ :

$$T = T - C.$$

Начальное значение переменной  $T$  равно  $N * C$ , где  $N$  — количество циклов.

3. Субблоки меняются местами.

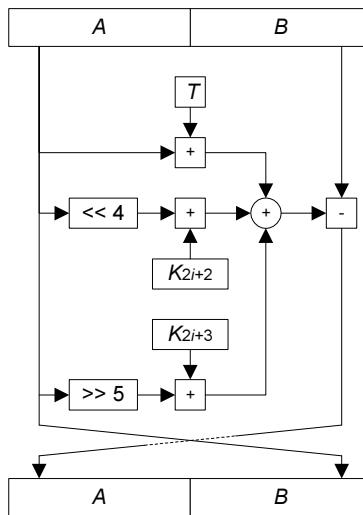


Рис. 3.210. Раунд расшифровывания алгоритма ТЕА

Как видно из описания, алгоритм действительно максимально прост в реализации; он основан только на операциях XOR, циклического сдвига на фиксированное число битов, а также модульного сложения и вычитания. Все эти операции могут быть легко реализованы на любой платформе. Авторы алгоритма признают, что есть и более простые, и более скоростные алгоритмы шифрования, однако, по их мнению, алгоритм ТЕА является «лучшим компромиссом между стойкостью, простотой реализации ... и производительностью» [393].

## Криптоанализ алгоритма ТЕА

Алгоритм ТЕА вызвал достаточно большой интерес у криптологического сообщества. До сих пор периодически появляются различные работы, посвященные его криптоанализу.

Одна из первых таких работ — известная работа Джона Келси, Брюса Шнайера и Дэвида Вагнера [200], посвященная криптоанализу ряда алгоритмов с помощью связанных ключей. Ее авторы предложили 3 атаки на полнораундовый алгоритм ТЕА методом дифференциального криптоанализа на связанных ключах. Наиболее эффективная из них требует наличия  $2^{23}$  выбранных открытых текстов и соответствующих им шифртекстов (зашифрованных на искомом ключе и одном связанным с ним ключе). Алгоритм вскрывается выполнением около  $2^{32}$  операций шифрования.

## Описание алгоритмов

Кроме того, для алгоритма TEA обнаружены классы эквивалентных ключей: каждый ключ имеет по 3 эквивалентных ключа, полученных инверсией старших битов  $K[0]$  и/или  $K[1]$ . Таким образом, вместо  $2^{128}$  различных ключей TEA имеет  $2^{126}$  классов эквивалентных ключей [197]. Благодаря этому алгоритм не может быть использован для построения функций хэширования. Данное свойство алгоритма было успешно использовано хакерами при взломе игровой консоли Microsoft Xbox, в которой TEA использовался именно в качестве основы для хэш-функции [106].

Ряд исследований статистических свойств алгоритма выполнила группа специалистов из Университета г. Мадрид; в частности, работа [178] посвящена методике, позволяющей отличить от случайной последовательности шифртекст, полученный с помощью алгоритма TEA.

Группа корейских специалистов из Университета г. Сеул предложила несколько атак на алгоритм TEA с уменьшенным количеством раундов:

- 11-раундовый TEA вскрывается методом невозможных дифференциалов при наличии  $2^{52.5}$  выбранных открытых текстов путем применения  $2^{85}$  операций шифрования [270];
- несколько позже атака была распространена и на 12-раундовый TEA [181].

Были исследованы и некоторые модификации алгоритма TEA.

- Фаузан Мирза (Fauzan Mirza) из Университета Royal Holloway (Лондон) предложил в работе [267] крайне упрощенный вариант алгоритма TEA — Simplified TEA (STEA), функция раунда которого выполняет лишь операцию XOR над обрабатываемым субблоком и используемым фрагментом ключа; алгоритм STEA оказался подвержен множеству атак, что никак не распространяется на алгоритм TEA, поскольку его функция раунда не сравнимо сложнее.
- Роджер Флеминг (Roger Fleming) исследовал вариант алгоритма TEA, в котором отсутствовала модификация переменной  $T$ , т. е.  $T$  была константой (конкретное значение не важно). Данный вариант вскрывается с помощью сдвиговой атаки (впоследствии предложенной в работе [92] для ряда алгоритмов), для которой требуется  $2^{32}$  известных открытых текстов и  $2^{98}$  операций шифрования [154]. Данная атака также не применима к оригинальному алгоритму TEA.

Интересный обзор алгоритмов семейства TEA и атак на них, а также исследование статистических свойств этих алгоритмов выполнил студент Университета штата Алабама Викрам Редди Андем (Vikram Reddy Andem) в работе [321]. Не менее интересный обзор можно найти на Web-странице Мэттью Рассела (Matthew Russell) [333].

Выводы исследователей таковы: за исключением атак на связанных ключах, TEA не имеет серьезных проблем с криптостойкостью. Тем не менее, авторы алгоритма предложили алгоритм XTEA (Extended TEA), в котором усиlena процедура расширения ключа с целью противодействия подобным атакам.

## Алгоритм XTEA

Помимо усиления процедуры расширения ключа, в алгоритме XTEA изменена также функция раунда. Как и в TEA, авторами алгоритма рекомендуется выполнять 64 раунда преобразований, в каждом из которых производятся следующие действия (рис. 3.211) [285]:

1. Обрабатывается правый 32-битный субблок данных  $B$  и накладывается на левый субблок  $A$ :

$$A = A + ((B \ll 4 \oplus B \gg 5) + B \oplus T + K_i).$$

2. В конце каждого четного раунда аналогично алгоритму TEA модифицируется временная переменная  $T$ .
3. Субблоки меняются местами.

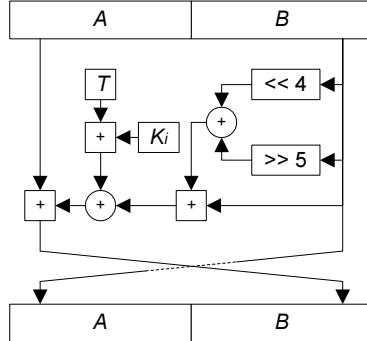


Рис. 3.211. Раунд алгоритма XTEA

Процедура расширения ключа, как и у алгоритма TEA, здесь практически отсутствует. Исходный 128-битный ключ так же представляется в виде 4 32-битных фрагментов  $K[0]...K[3]$ , но в раундах шифрования они используются в более сложном порядке:

- в четных раундах в качестве  $K_i$  используется фрагмент  $K[T \& 3]$ , где  $\&$  — побитовая логическая операция «и»;
- в нечетных — фрагмент  $K[(T >> 11) \& 3]$ .

Функция раунда выше приведена в нотации, используемой авторами алгоритма. В [333] отмечается, что нотация оставляет различные сомнения в приоритете выполнения операций. Основываясь на приоритете выполнения операций в языке Си, Мэтью Рассел делает вывод, что правильная последовательность выполнения такова (именно она показана на рис. 3.211):

$$A = A + (((B \ll 4 \oplus B \gg 5) + B) \oplus (T + K_i)).$$

Однако ему известны ошибочные реализации алгоритма XTEA, в которых применяется одна из следующих последовательностей:

$$A = A + ((B \ll 4 \oplus B \gg 5) + (B \oplus T) + K_i);$$

$$A = A + (((((B \ll 4 \oplus B \gg 5) + B) \oplus T) + K_i).$$

Раунд расшифровывания алгоритма XTEA выглядит так (рис. 3.212):

1. Обрабатывается значение левого субблока данных и накладывается на правый субблок:

$$B = B - ((A \ll 4 \oplus A \gg 5) + A \oplus T + K_i).$$

В качестве  $K_i$  в четных раундах расшифровывания используется  $K[(T >> 11) \& 3]$ , в нечетных —  $K[T \& 3]$ .

2. В конце каждого четного раунда выполняется модификация временной переменной  $T$ :

$$T = T - C.$$

Начальное значение переменной  $T$ , как и в алгоритме TEA, равно  $N * C$ .

3. Субблоки меняются местами.

Как видно из описания, XTEA не является заметно более сложным, чем TEA.

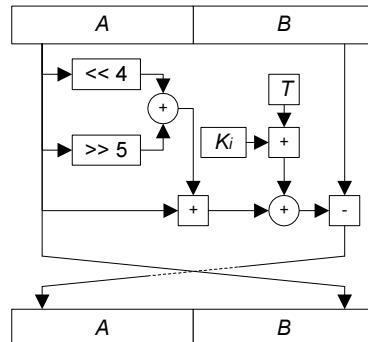


Рис. 3.212. Раунд расшифровывания алгоритма XTEA

## Криптоанализ алгоритма XTEA

Наибольшего прогресса в криптоанализе алгоритма XTEA достигли те же упомянутые выше корейские специалисты:

- сначала ими была найдена атака методом невозможных дифференциалов на 14-раундовый алгоритм XTEA, который вскрывается при наличии  $2^{62,5}$  выбранных открытых текстов использованием  $2^{85}$  операций шифрования [270];
- затем атака была распространена на 23-раундовый XTEA [181];
- кроме того, была предложена атака методом усеченных дифференциалов на связанных ключах на 27-раундовый XTEA, для которой требуется  $2^{20,5}$  выбранных открытых текстов и  $2^{115,15}$  операций шифрования (вероятность успеха атаки — 96,9 %) [227].

Авторы XTEA явно достигли успеха в противодействии атакам Келси, Шнайера и Вагнера, использующим слабость процедуры расширения ключа. Однако, судя по достижениям корейских специалистов, функция раунда XTEA явно слабее таковой у TEA. Тем не менее, полнораундовый алгоритм XTEA обладает весьма высоким запасом криптостойкости.

## Алгоритм Block TEA

Алгоритмы TEA и XTEA могут быть использованы для шифрования больших объемов данных в различных стандартных режимах работы [151] (см. разд. 1.4). Кроме того, для шифрования больших блоков данных (неограниченного размера, кратного 32 битам) авторы предложили алгоритм Block TEA [285]. В этом алгоритме используется то же преобразование, что и в каждом раунде алгоритма XTEA (т. е. описанный выше шаг 1 раунда XTEA, на рис. 3.213 обозначен как  $f()$ ).

Количество раундов алгоритма  $R$  определяется так:

$$R = 6 + 52/n,$$

где  $n$  — размер обрабатываемого блока данных в 32-битных словах, но не менее двух.

В каждом раунде производятся следующие действия:

1. Аналогично TEA и XTEA выполняется модификация переменной  $T$ .
2. В цикле по  $j$  от 0 до  $n - 1$  выполняется функция  $f()$  над соответствующим 32-битным субблоком. В качестве используемого фрагмента ключа в данном случае берется фрагмент  $K[(j \& 3) \oplus ((T >> 2) \& 3)]$ .

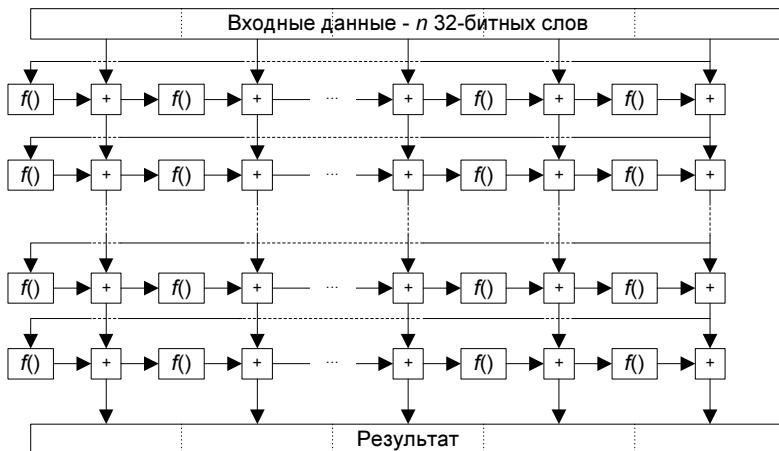


Рис. 3.213. Структура алгоритма Block TEA

Структура алгоритма Block TEA приведена на рис. 3.213.

Расшифровывание производится с использованием преобразования, выполняемого при расшифровывании в алгоритме XTEA:

$$B = B - ((A \ll 4 \oplus A \gg 5) + A \oplus T + K_i).$$

Субблоки данных обрабатываются в обратной последовательности, а модификация переменной  $T$  выполняется вычитанием из нее константы  $C$ . Начальное значение  $T$  равно  $C * R$ .

Считается, что алгоритм Block TEA при шифровании им сообщений достаточно большого размера является более эффективным, чем алгоритм XTEA, примененный к таким сообщениям в стандартных режимах работы CBC, OFB или CFB [285, 333].

Известный финский криптоаналитик Марку-Юхани Сааринен (Markku-Juhani Saarinen) нашел уязвимость в алгоритме Block TEA; для вычисления ключа с ее помощью необходимо наличие  $2^{34}$  выбранных шифртекстов и результатов их расшифровывания (см. [333]). Кроме того, упомянутый выше Викрам Редди Андем предположил наличие одного эквивалентного ключа для каждого ключа алгоритма Block TEA.

## Алгоритм XXTEA

С целью противодействия атаке, предложенной Саариненом, Уилер и Нидэм предложили новый вариант алгоритма Block TEA, названный XXTEA (он же Corrected Block TEA [395] или Block TEA 2 [333]).

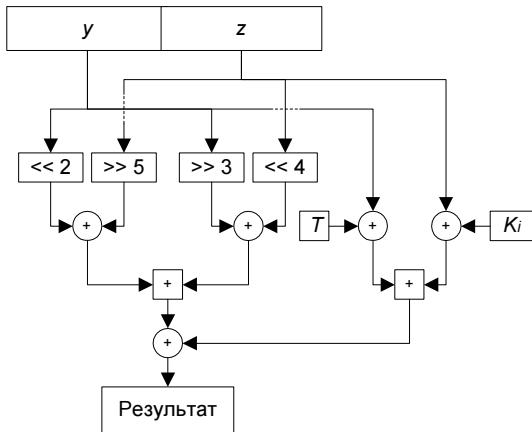


Рис. 3.214. Функция  $MX$  алгоритма XXTEA

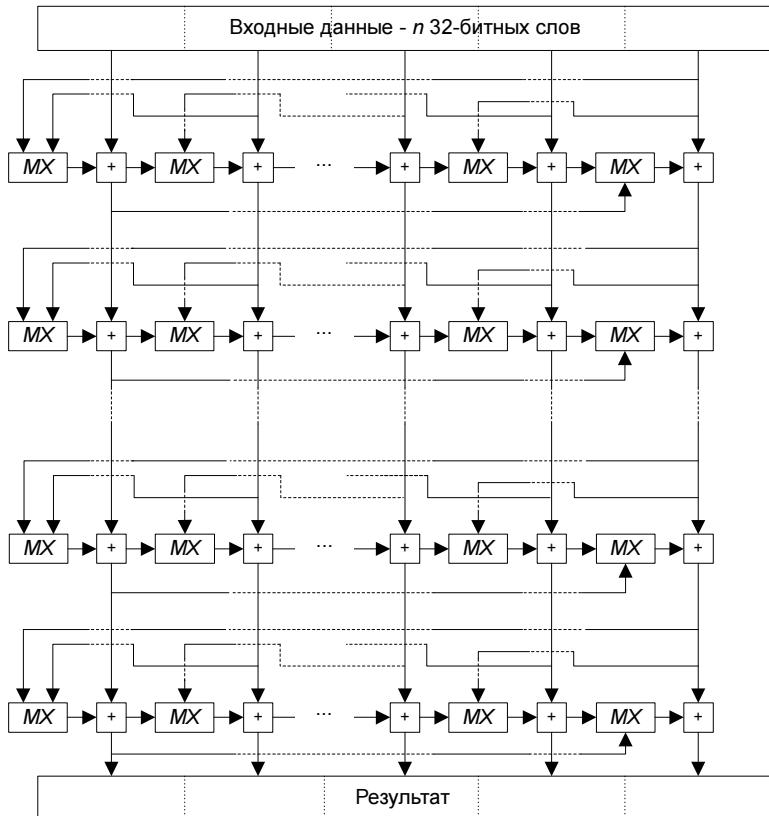


Рис. 3.215. Структура алгоритма XXTEA

Алгоритм XXTEA весьма похож на Block TEA, различия между ними перечислены далее [392].

- Вместо преобразования  $f()$  используется принципиально другое преобразование  $MX()$ , которое обрабатывает два 32-битных входа вместо одного:

$$MX(y, z) = (z \gg 5 \oplus y \ll 2) + (y \gg 3 \oplus z \ll 4) \oplus (T \oplus y) + (K_i \oplus z),$$

где  $y$  и  $z$  — входные значения.

Схема функции  $MX$  приведена на рис. 3.214.

- В качестве значения  $y$  используется ( $j - 1$ )-й субблок данных (или ( $n - 1$ )-й субблок для  $j = 0$ ), в качестве  $z$  берется ( $j + 1$ )-й субблок или 0-й субблок при  $j = n - 1$  (рис. 3.215).

Алгоритм XXTEA действительно оказался сильнее, чем Block TEA. Единственное широко известное исследование данного алгоритма принадлежит все тому же Сааринену, однако его результаты позволяют лишь отличить шифртекст, полученный с помощью 6-раундового XXTEA, от случайной последовательности (см. [333]).

## 3.56. Алгоритмы Twofish и Twofish-FK

Алгоритм Twofish разработан коллективом известных криптологов под руководством Брюса Шнайера — автора множества работ в области криптологии (в т. ч. известнейшей книги [28]), разработчика известного алгоритма шифрования Blowfish (см. разд. 3.8) и основателя американской компании Counterpane Systems (<http://www.counterpane.com>), являющейся одним из мировых лидеров в области разработки средств криптографической защиты информации. Кроме него, в разработке алгоритма принимали участие Джон Келси, Крис Холл и Нильс Фергюсон из той же компании Counterpane Systems, а также Дуг Уайтинг (Doug Whiting) из Hifn Incorporated (разработка аппаратных средств защиты интернет-коммуникаций, <http://www.hifn.com>) и Дэвид Вагнер из Университета Беркли (<http://www.berkeley.edu>).

### Структура алгоритма

Алгоритм Twofish разбивает шифруемые данные на четыре 32-битных субблока (обозначим их  $A, B, C, D$ ), над которыми производится 16 раундов преобразований, в каждом из которых выполняются следующие операции (рис. 3.216) [347]:

1. Содержимое субблока  $B$  циклически сдвигается влево на 8 битов.
2. Субблок  $A$  обрабатывается операцией  $g()$ , которая будет подробно описана далее.

3. Субблок  $B$  также обрабатывается операцией  $g()$ .
4. Субблок  $B$  накладывается на  $A$  с помощью сложения по модулю  $2^{32}$ , после чего аналогичным образом выполняется наложение субблока  $A$  на субблок  $B$ .
5. Фрагмент расширенного ключа  $K_{2r+8}$  (где  $r$  — номер текущего раунда, начиная с 0) складывается с субблоком  $A$  по модулю  $2^{32}$ .
6. Аналогично предыдущему шагу,  $K_{2r+9}$  накладывается на субблок  $B$ .
7. Субблок  $A$  накладывается на  $C$  операцией XOR.
8. Содержимое субблока  $D$  циклически сдвигается влево на 1 бит.
9. Субблок  $B$  накладывается на  $D$  операцией XOR.
10. Содержимое субблока  $C$  циклически сдвигается вправо на 1 бит.

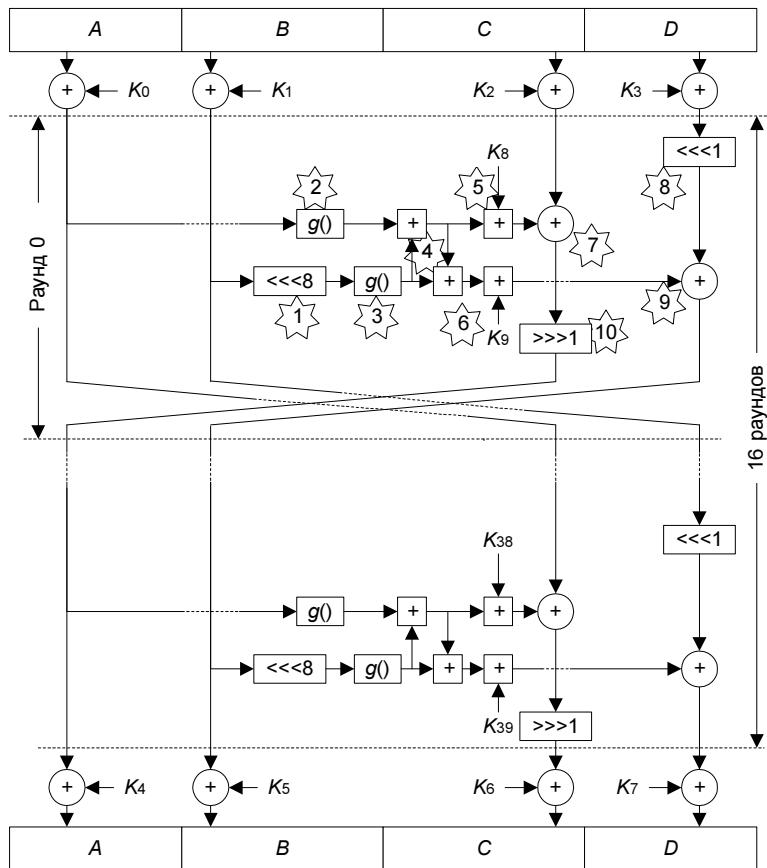


Рис. 3.216. Структура алгоритма Twofish

Описанные выше действия можно представить следующим образом:

$$B = B \lll 8;$$

$$A = g(A);$$

$$B = g(B);$$

$$A = A + B \bmod 2^{32};$$

$$B = A + B \bmod 2^{32};$$

$$A = A + K_{2r+8} \bmod 2^{32};$$

$$B = B + K_{2r+9} \bmod 2^{32};$$

$$C = C \oplus A;$$

$$D = D \lll 1;$$

$$D = D \oplus B;$$

$$C = C \ggg 1.$$

Операция  $g()$  обрабатывает 32-битный входной субблок выполнением следующих шагов (рис. 3.217):

1. Субблок делится на 4 фрагмента по 8 битов каждый.
2. Фрагменты «прогоняются» через таблицы замен  $8 \times 8$  битов  $S_0 \dots S_3$ . Таблицы замен вычисляются динамически и зависят от ключа шифрования; алгоритм построения таблиц замен будет подробно описан далее.
3. Результат замен (обозначим его  $s_0 \dots s_3$ ) умножается на фиксированную матрицу  $M_1$ :

$$\begin{pmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{pmatrix} = M_1 * \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix},$$

где  $g_0 \dots g_3$  — байты выходного значения функции  $g()$ .

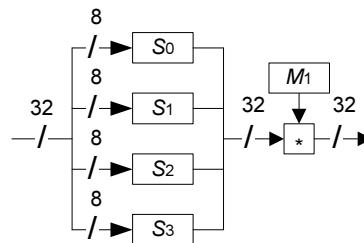


Рис. 3.217. Функция  $g$  алгоритма Twofish

Умножение выполняется в конечном поле  $GF(2^8)$  с образующим полиномом  $x^8 + x^6 + x^5 + x^3 + x + 1$ .

Матрица  $M_1$  приведена в табл. 3.136 (здесь и далее в таблицах указаны шестнадцатеричные значения).

**Таблица 3.136**

01	EF	5B	5B
5B	EF	EF	01
EF	5B	01	EF
EF	01	EF	5B

В конце каждого раунда, за исключением последнего, субблоки  $A$  (значение до описанной выше обработки) и  $C$  меняются местами; субблоки  $B$  (значение до обработки) и  $D$  также меняются местами.

Перед первым раундом выполняется входное отбеливание — наложение операцией XOR на обрабатываемые субблоки четырех фрагментов расширенного ключа  $K_0 \dots K_3$ . Аналогично выполняется выходное отбеливание — после заключительного раунда с использованием подключей  $K_4 \dots K_7$ .

## Процедура расширения ключа

Процедура расширения ключа формирует 40 32-битных подключей для использования в 16 раундах алгоритма и для выполнения операций отбеливания. Кроме того, на основе ключа шифрования вычисляются таблицы замен  $S_0 \dots S_3$ .

Как и Serpent (см. разд. 3.48), алгоритм Twofish использует ключи шифрования любого размера до 256 битов включительно. Однако дополнение ключа выполняется несколько иначе: исходный ключ, при необходимости, дополняется нулевыми битами до ближайшего стандартного размера, т. е. до 128, 192 или 256 битов. Процедура расширения ключа обрабатывает дополненный таким образом ключ.

Сначала производится предварительная обработка ключа, включающая в себя следующие шаги:

1. Выполняется инициализация переменных, участвующих в дальнейших расчетах:

$$k = N/64,$$

где  $N$  — размер дополненного ключа шифрования в битах, т. е.  $k$  принимает значение 2, 3 или 4. Ключ шифрования представляется в виде

## Описание алгоритмов

$8k$  байтов  $m_0 \dots m_{8k-1}$  или в виде  $2k$  32-битных слов, обозначаемых как  $M_0 \dots M_{2k-1}$ .

2. Формируются 3 массива, каждый из которых состоит из  $k$  32-битных слов:

$$M_e = (M_0, M_2, \dots, M_{2k-2});$$

$$M_o = (M_1, M_3, \dots, M_{2k-1});$$

$$V = (V_{k-1}, V_{k-2}, \dots, V_0),$$

где:

$$V_i = \sum_{j=0}^3 v_{i,j} * 2^{8j};$$

$$\begin{pmatrix} v_{i,0} \\ v_{i,1} \\ v_{i,2} \\ v_{i,3} \end{pmatrix} = M_2 * \begin{pmatrix} m_{8i} \\ m_{8i+1} \\ m_{8i+2} \\ m_{8i+3} \\ m_{8i+4} \\ m_{8i+5} \\ m_{8i+6} \\ m_{8i+7} \end{pmatrix}.$$

Матрица  $M_2$  приведена в табл. 3.137.

Таблица 3.137

01	A4	55	87	5A	58	DB	9E
A4	56	82	F3	1E	C6	68	E5
02	A1	FC	C1	47	AE	3D	19
A4	55	87	5A	58	DB	9E	03

Генерация подключей  $k_0 \dots k_{39}$  производится на основе вычисленных на предварительном этапе массивов  $M_e$  и  $M_o$  следующим образом:

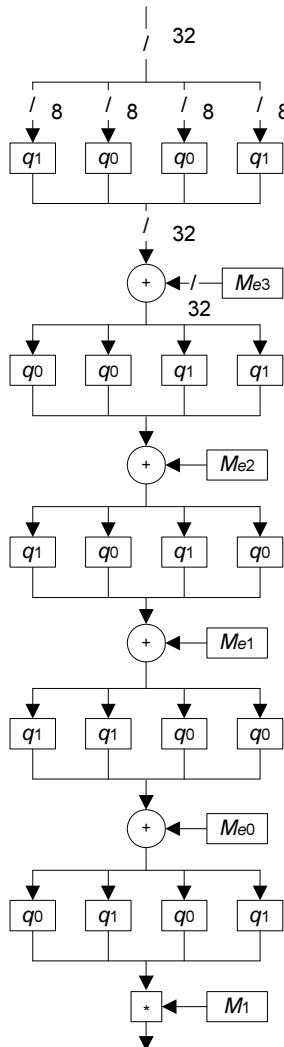
$$k_{2i} = A_i + B_i \bmod 2^{32};$$

$$k_{2i+1} = (A_i + 2B_i \bmod 2^{32}) \lll 9,$$

где  $i = 0 \dots 19$ , а  $A_i$  и  $B_i$  — промежуточные величины, вычисляемые так:

$$A_i = h(2i\rho, M_e);$$

$$B_i = h((2i+1)\rho, M_o) \lll 8.$$



**Рис. 3.218.** Функция  $h$  алгоритма Twofish

Константа  $\rho$  определена следующим образом:

$$\rho = 2^{24} + 2^{16} + 2^8 + 1,$$

а функция  $h()$  заслуживает подробного описания.

Эта функция схематично представлена на рис. 3.218. Она выполняется в несколько шагов, состав которых зависит от размера дополненного ключа в 64-битных фрагментах, т. е. от описанного выше значения  $k$ . В качестве пара-

метров функция  $h()$  принимает 32-битное слово и массив 32-битных слов размерностью  $k$ . Последовательность выполнения такова:

**Шаг 1.** Входное слово делится на 4 8-битных фрагмента, которые «прогоняются» через специфические операции замены  $q_0$  и  $q_1$  (как показано на рис. 3.218). Результат замены объединяется в 32-битное слово, которое складывается с третьим словом входного массива (на рис. 3.218 в качестве входного массива показан массив  $M_e$ ). Данный шаг не выполняется, если  $k < 4$  (т. е.  $k = 2$  или  $k = 3$ ). На рис. 3.218 показаны все возможные шаги функции  $h()$ .

**Шаг 2.** Результат предыдущего шага или входное слово обрабатывается аналогичным образом (с другой последовательностью применения  $q_0$  и  $q_1$ , которая различна для каждого шага — см. рис. 3.218), но складывается со вторым словом входного массива. Шаг 2 не выполняется, если  $k = 2$ .

**Шаги 3 и 4.** Выполняются всегда и включают в себя обработку результата предыдущего шага (или входного слова — для шага 3 и  $k = 2$ ), аналогичную предыдущим шагам с использованием, соответственно, первого и нулевого слов входного массива.

**Шаг 5.** Как и на предыдущих шагах, применяются замены  $q_0$  и  $q_1$ , после чего выполняется следующее преобразование:

$$\begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix} = M_1 * \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix};$$

$$H = \sum_{i=0}^3 h_i * 2^{8i},$$

где:

- $y_0 \dots y_3$  — байты результата выполнения замен на шаге 5;
- матрица  $M_1$  была описана выше;
- $H$  — выходное значение функции  $h()$ .

Операции  $q_0$  и  $q_1$  представляют собой не табличные замены  $8 \times 8$  битов, а вычисляют выходные значения с использованием нескольких таблиц замен  $4 \times 4$  следующим образом:

$$a_0 = \lfloor x/16 \rfloor;$$

$$b_0 = x \bmod 16;$$

$$\begin{aligned}
 a_1 &= a_0 \oplus b_0; \\
 b_1 &= a_0 \oplus (b_0 \ggg_4 1) \oplus 8a_0 \bmod 16; \\
 a_2 &= t_0(a_1); \\
 b_2 &= t_1(b_1); \\
 a_3 &= a_2 \oplus b_2; \\
 b_3 &= a_2 \oplus (b_2 \ggg_4 1) \oplus 8a_2 \bmod 16; \\
 a_4 &= t_2(a_3); \\
 b_4 &= t_3(b_3); \\
 y &= 16b_4 + a_4,
 \end{aligned}$$

где:

- $x$  — входное значение;
- $y$  — выходное значение;
- $\ggg_4$  — операция циклического вращения вправо 4-битных величин, т. е. раздельное вращение каждого nibla обрабатываемого байта;
- $t_i$  — табличные замены, различные для  $q_0$  и  $q_1$ ;  $t_i$  для  $q_0$  представлена в табл. 3.138.

Таблица 3.138

$t_0$	8	1	7	D	6	F	3	2	0	B	5	9	E	C	A	4
$t_1$	E	C	B	8	1	2	3	5	F	4	A	6	7	0	9	D
$t_2$	B	A	5	E	6	D	9	0	C	8	F	3	2	4	7	1
$t_3$	D	7	F	4	1	2	6	E	9	B	3	0	8	5	C	A

Выходное значение таблицы берется из позиции, соответствующей входному значению; например,  $t_1$  заменяет 0 на E, 1 на C и т. д.

Таблицы для  $q_1$  приведены в табл. 3.139.

Таблица 3.139

$t_0$	2	8	B	D	F	7	6	E	3	1	9	4	0	A	C	5
$t_1$	1	E	2	B	4	C	3	7	6	D	A	5	F	9	0	8
$t_2$	4	C	7	5	1	6	9	A	0	E	D	8	2	B	3	F
$t_3$	B	9	5	1	C	3	D	E	6	4	7	F	2	0	8	A

Осталось описать зависимость от ключа таблиц замен  $S_0...S_3$ . Фактически в алгоритме шифрования вместо описанной выше функции  $g()$  применяется функция  $h()$ , использующая в качестве входного значения 32-битный субблок  $A$  или  $B$ , а в качестве входного массива — описанный ранее массив  $V$ . Таким образом, замена выполняется на основе тех же таблиц  $t_0...t_3$ , которые используются в процедуре расширения ключа.

Видно, что процедура расширения ключа является сложной и занимает несравненно больше времени, чем шифрование одного блока данных. Авторы алгоритма в его спецификации [347] описали несколько возможных вариантов оптимизации процедуры расширения ключа, один из которых можно выбрать в конкретной реализации алгоритма, учитывая следующие параметры:

- доступный объем энергонезависимой и оперативной памяти;
- предполагаемую частоту смены ключа при шифровании данных;
- требования к скорости шифрования блоков данных.

Общий принцип оптимизации состоит в поисках требуемого компромисса между скоростью шифрования и временем выполнения процедуры расширения ключа: чем быстрее нужно шифровать, тем дольше будет выполняться расширение ключа, и наоборот.

## Алгоритм Twofish-FK

Помимо описанного выше «стандартного» варианта алгоритма Twofish, авторы предложили еще и Twofish Family Key (или Twofish-FK) — шаблон для формирования на основе Twofish вариантов алгоритма, несовместимых между собой [347] и предназначенных, таким образом, для ограниченных применений, например, в рамках конкретных организаций для защиты внутреннего документооборота.

Несовместимость вариантов достигается путем применения дополнительного ключа (*FK*), одинакового для всех субъектов, использующих конкретный вариант алгоритма Twofish-FK. То есть конкретное значение используемого *FK* формирует некий «контуру совместимости» криптосредств, реализующих Twofish-FK.

Дополнительный ключ *FK* «подмешивается» на этапе расширения ключа следующим образом:

1. Инициализируются переменные, используемые в дальнейших операциях (этот шаг выполняется однократно для конкретного *FK*):

$$T_0 = FK;$$

$$T_1 = (E_{FK}(0) \bullet E_{FK}(1)) \oplus FK;$$

$$\begin{aligned}T_2 &= E_{FK}(2); \\ T_3 &= E_{FK}(3); \\ T_4 &= E_{FK}(4) \bmod 2^{64},\end{aligned}$$

где:

- $E_{FK}(N)$  — результат зашифровывания алгоритмом Twofish на ключе  $FK$  128-битного блока, первый байт которого содержит значение  $N$ , а остальные байты обнулены;
- • — операция конкатенации.

Стоит отметить, что переменные  $T_0$  и  $T_1$  имеют размерность 256 битов,  $T_2$  и  $T_3$  — по 128 битов, а  $T_4$  — 64 бита.

2. Наложение ключа  $FK$  выполняется однократно для каждого ключа шифрования, используемого совместно с данным  $FK$ .

- Перед выполнением процедуры расширения ключа шифрования производится наложение  $T_0$  на ключ шифрования операцией XOR. Если ключ шифрования имеет размер, меньший 256 битов, то используется необходимое количество битов  $T_0$ .
- После выполнения процедуры расширения ключа операциями XOR  $T_2$  накладывается на подключи  $K_0...K_3$ ,  $T_3$  — на подключи  $K_4...K_7$ , а  $T_4$  — на каждую пару подключей, используемых в раундах шифрования.
- Перед использованием табличных замен выполняется наложение операцией XOR требуемого количества битов  $T_1$  на ключ шифрования.

Криптостойкость алгоритмов, построенных на основе Twofish-FK, по словам авторов алгоритма, полностью эквивалентна криптостойкости стандартного Twofish, поскольку подобное наложение дополнительного ключа никак не влияет на основные параметры алгоритма [347].

## Достоинства и недостатки алгоритма

Сравнительный анализ достоинств и недостатков алгоритмов — финалистов конкурса AES приведен в разд. 2.1.



## Приложение

# Таблицы замен

В этом приложении представлены таблицы замен некоторых из описанных в главе 3 алгоритмов шифрования.

## П1. Алгоритм Blowfish

В этом разделе приведены начальные значения таблиц замен алгоритма Blowfish, которые подвергаются модификации в процессе расширения ключа шифрования — см. описание алгоритма в разд. 3.8.

Начальные значения таблиц  $S_1$ ,  $S_2$ ,  $S_3$  и  $S_4$  представлены, соответственно, в табл. П1–П4.

Таблица П1

d1310ba6	98dfb5ac	2ffd72db	d01adfb7	b8e1afed	6a267e96	ba7c9045	f12c7f99
24a19947	b3916cf7	0801f2e2	858efc16	636920d8	71574e69	a458fea3	f4933d7e
0d95748f	728eb658	718bcd58	82154aee	7b54a41d	c25a59b5	9c30d539	2af26013
c5d1b023	286085f0	ca417918	b8db38ef	8e79dcb0	603a180e	6c9e0e8b	b01e8a3e
d71577c1	bd314b27	78af2fda	55605c60	e65525f3	aa55ab94	57489862	63e81440
55ca396a	2aab10b6	b4cc5c34	1141e8ce	a15486af	7c72e993	b3ee1411	636fbc2a
2ba9c55d	741831f6	ce5c3e16	9b87931e	af6ba33	6c24cf5c	7a325381	28958677
3b8f4898	6b4bb9af	c4bfe81b	66282193	61d809cc	fb21a991	487cac60	5dec8032
ef845d5d	e98575b1	dc262302	eb651b88	23893e81	d396acc5	0f6d6ff3	83f44239
2e0b4482	a4842004	69c8f04a	9e1f9b5e	21c66842	f6e96c9a	670c9c61	abd388f0
6a51a0d2	d8542f68	960fa728	ab5133a3	6eef0b6c	137a3be4	ba3bf050	7efb2a98

Таблица П1 (окончание)

a1f1651d	39af0176	66ca593e	82430e88	8cee8619	456f9fb4	7d84a5c3	3b8b5ebe
e06f75d8	85c12073	401a449f	56c16aa6	4ed3aa62	363f7706	1bfedf72	429b023d
37d0d724	d00a1248	db0fead3	49f1c09b	075372c9	80991b7b	25d479d8	f6e8def7
e3fe501a	b6794c3b	976ce0bd	04c006ba	c1a94fb6	409f60c4	5e5c9ec2	196a2463
68fb6faf	3e6c53b5	1339b2eb	3b52ec6f	6dfc511f	9b30952c	cc814544	af5ebd09
bee3d004	de334afd	660f2807	192e4bb3	c0cba857	45c8740f	d20b5f39	b9d3fbdb
5579c0bd	1a60320a	d6a100c6	402c7279	679f25fe	fb1fa3cc	8ea5e9f8	db3222f8
3c7516df	fd616b15	2f501ec8	ad0552ab	323db5fa	fd238760	53317b48	3e00df82
9e5c57bb	ca6f8ca0	1a87562e	df1769db	d542a8f6	287effc3	ac6732c6	8c4f5573
695b27b0	bbca58c8	e1ffa35d	b8f011a0	10fa3d98	fd2183b8	4afcb56c	2dd1d35b
9a53e479	b6f84565	d28e49bc	4fb9790	e1ddf2da	a4cb7e33	62fb1341	cee4c6e8
ef20cada	36774c01	d07e9efe	2bf11fb4	95dbda4d	ae909198	eaad8e71	6b93d5a0
d08ed1d0	afc725e0	8e3c5b2f	8e7594b7	8ff6e2fb	f2122b64	8888b812	900df01c
4fad5ea0	688fc31c	d1cff191	b3a8c1ad	2f2f2218	be0e1777	ea752dfe	8b021fa1
e5a0cc0f	b56f74e8	18acf3d6	ce89e299	b4a84fe0	fd13e0b7	7cc43b81	d2ada8d9
165fa266	80957705	93cc7314	211a1477	e6ad2065	77b5fa86	c75442f5	fb9d35cf
ebcdaf0c	7b3e89a0	d6411bd3	ae1e7e49	00250e2d	2071b35e	226800bb	57b8e0af
2464369b	f009b91e	5563911d	59dfa6aa	78c14389	d95a537f	207d5ba2	02e5b9c5
83260376	6295cfa9	11c81968	4e734a41	b3472dca	7b14a94a	1b510052	9a532915
d60f573f	bc9bc6e4	2b60a476	81e67400	08ba6fb5	571be91f	f296ec6b	2a0dd915
b6636521	e7b9f9b6	ff34052e	c5855664	53b02d5d	a99f8fa1	08ba4799	6e85076

Таблица П2

4b7a70e9	b5b32944	db75092e	c4192623	ad6ea6b0	49a7df7d	9cee60b8	8fedb266
ecaa8c71	699a17ff	5664526c	c2b19ee1	193602a5	75094c29	a0591340	e4183a3e
3f54989a	5b429d65	6b8fe4d6	99f73fd6	a1d29c07	efe830f5	4d2d38e6	f0255dc1
4cdd2086	8470eb26	6382e9c6	021ecc5e	09686b3f	3ebaefc9	3c971814	6b6a70a1
687f3584	52a0e286	b79c5305	aa500737	3e07841c	7fdeae5c	8e7d44ec	5716f2b8
b03ada37	f0500c0d	f01c1f04	0200b3ff	ae0cf51a	3cb574b2	25837a58	dc0921bd

## Таблицы замен

Таблица П2 (окончание)

d19113f9	7ca92ff6	94324773	22f54701	3ae5e581	37c2dadc	c8b57634	9af3ddda7
a9446146	0fd0030e	ecc8c73e	a4751e41	e238cd99	3bea0e2f	3280bba1	183eb331
4e548b38	4f6db908	6f420d03	f60a04bf	2cb81290	24977c79	5679b072	bcaf89af
de9a771f	d9930810	b38bae12	dccf3f2e	5512721f	2e6b7124	501adde6	9f84cd87
7a584718	7408da17	bc9f9abc	e94b7d8c	ec7aec3a	db851dfa	63094366	c464c3d2
ef1c1847	3215d908	dd433b37	24c2ba16	12a14d43	2a65c451	50940002	133ae4dd
71dff89e	10314e55	81ac77d6	5f11199b	043556f1	d7a3c76b	3c11183b	5924a509
f28fe6ed	97f1fbfa	9ebabf2c	1e153c6e	86e34570	eae96fb1	860e5e0a	5a3e2ab3
771fe71c	4e3d06fa	2965dcb9	99e71d0f	803e89d6	5266c825	2e4cc978	9c10b36a
c6150eba	94e2ea78	a5fc3c53	1e0a2df4	f2f74ea7	361d2b3d	1939260f	19c27960
5223a708	f71312b6	ebadfe6e	eac31f66	e3bc4595	a67bc883	b17f37d1	018cff28
c332ddef	be6c5aa5	65582185	68ab9802	eecea50f	db2f953b	2aef7dad	5b6e2f84
1521b628	29076170	ecdd4775	619f1510	13cca830	eb61bd96	0334fe1e	aa0363cf
b5735c90	4c70a239	d59e9e0b	cbaade14	eeccc86bc	60622ca7	9cab5cab	b2f3846e
648b1leaf	19bdf0ca	a02369b9	655abb50	40685a32	3c2ab4b3	319ee9d5	c021b8f7
9b540b19	875fa099	95f7997e	623d7da8	f837889a	97e32d77	11ed935f	16681281
0e358829	c7e61fd6	96dedfa1	7858ba99	57f584a5	1b227263	9b83c3ff	1ac24696
cdb30aeb	532e3054	8fd948e4	6dbc3128	58ebf2ef	34c6ffea	fe28ed61	ee7c3c73
5d4a14d9	e864b7e3	42105d14	203e13e0	45eee2b6	a3aaabea	db6c4f15	facb4fd0
c742f442	ef6abbb5	654f3b1d	41cd2105	d81e799e	86854dc7	e44b476a	3d816250
cf62a1f2	5b8d2646	fc8883a0	c1c7b6a3	7f1524c3	69cb7492	47848a0b	5692b285
095bbf00	ad19489d	1462b174	23820e00	58428d2a	0c55f5ea	1dadf43e	233f7061
3372f092	8d937e41	d65fecf1	6c223bdb	7cde3759	cbef7460	4085f2a7	ce77326e
a6078084	19f8509e	e8efd855	61d99735	a969a7aa	c50c06c2	5a04abfc	800bcadc
9e447a2e	c3453484	fdd56705	0e1e9ec9	db73bdb3	105588cd	675fda79	e3674340
c5c43465	713e38d8	3d28f89e	f16dff20	153e21e7	8fb03d4a	e6e39f2b	db83adf

Таблица П3

e93d5a68	948140f7	f64c261c	94692934	411520f7	7602d4f7	bcf46b2e	d4a20068
d4082471	3320f46a	43b7d4b7	500061af	1e39f62e	97244546	14214f74	bf8b8840
4d95fc1d	96b591af	70f4ddd3	66a02f45	bfb09ec	03bd9785	7fac6dd0	31cb8504
96eb27b3	55fd3941	da2547e6	abca0a9a	28507825	530429f4	0a2c86da	e9b66dfb
68dc1462	d7486900	680ec0a4	27a18dee	4f3ffea2	e887ad8c	b58ce006	7af4d6b6
aace1e7c	d3375fec	ce78a399	406b2a42	20fe9e35	d9f385b9	ee39d7ab	3b124e8b
1dc9faf7	4b6d1856	26a36631	eae397b2	3a6efa74	dd5b4332	6841e7f7	ca7820fb
fb0af54e	d8feb397	454056ac	ba489527	55533a3a	20838d87	fe6ba9b7	d096954b
55a867bc	a1159a58	cca92963	99e1db33	a62a4a56	3f3125f9	5ef47e1c	9029317c
fdf8e802	04272f70	80bb155c	05282ce3	95c11548	e4c66d22	48c1133f	c70f86dc
07f9c9ee	41041f0f	404779a4	5d886e17	325f51eb	d59bc0d1	f2bcc18f	41113564
257b7834	602a9c60	dff8e8a3	1f636c1b	0e12b4c2	02e1329e	af664fd1	cad18115
6b2395e0	333e92e1	3b240b62	eebeb922	85b2a20e	e6ba0d99	de720c8c	2da2f728
d0127845	95b794fd	647d0862	e7ccf5f0	5449a36f	877d48fa	c39dfd27	f33e8d1e
0a476341	992eff74	3a6f6eab	f4f8fd37	a812dc60	a1ebddf8	991be14c	db6e6b0d
c67b5510	6d672c37	2765d43b	dcd0e804	f1290dc7	cc00ffa3	b5390f92	690fed0b
667b9ffb	cedb7d9c	a091cf0b	d9155ea3	bb132f88	515bad24	7b9479bf	763bd6eb
37392eb3	cc115979	8026e297	f42e312d	6842ada7	c66a2b3b	12754ccc	782ef11c
6a124237	b79251e7	06a1bbe6	4fbfb6350	1a6b1018	11caedfa	3d25bdd8	e2e1c3c9
44421659	0a121386	d90cec6e	d5abea2a	64af674e	da86a85f	bebfe988	64e4c3fe
9dbc8057	f0f7c086	60787bf8	6003604d	d1fd8346	f6381fb0	7745ae04	d736fcc
83426b33	f01eab71	b0804187	3c005e5f	77a057be	bde8ae24	55464299	bf582e61
4e58f48f	f2ddfd42	f474ef38	8789bdc2	5366f9c3	c8b38e74	b475f255	46fc9b9
7aeb2661	8b1ddf84	846a0e79	915f95e2	466e598e	20b45770	8cd55591	c902de4c
b90bace1	bb8205d0	11a86248	7574a99e	b77f19b6	e0a9dc09	662d09a1	c4324633
e85a1f02	09f0be8c	4a99a025	1d6efe10	1ab93d1d	0ba5a4df	a186f20f	2868f169
dc77da83	573906fe	a1e2ce9b	4fc7f52	50115e01	a70683fa	a002b5c4	0de6d027
9af88c27	773f8641	c3604c06	61a806b5	f0177a28	c0f586e0	006058aa	30dc7d62
11e69ed7	2338ea63	53c2dd94	c2c21634	bbcbee56	90bcb6de	ebfc7da1	ce591d76

## Таблицы замен

Таблица П3 (окончание)

6f05e409	4b7c0188	39720a3d	7c927c24	86e3725f	724d9db9	1ac15bb4	d39eb8fc
ed545578	08fcab5b5	d83d7cd3	4dad0fc4	1e50ef5e	b161e6f8	a28514d9	6c51133c
6fd5c7e7	56e14ec4	362abfce	ddc6c837	d79a3234	92638212	670efa8e	406000e

Таблица П4

3a39ce37	d3faf5cf	abc27737	5ac52d1b	5cb0679e	4fa33742	d3822740	99bc9bbe
d5118e9d	bf0f7315	d62d1c7e	c700c47b	b78c1b6b	21a19045	b26eb1be	6a366eb4
5748ab2f	bc946e79	c6a376d2	6549c2c8	530ff8ee	468dde7d	d5730a1d	4cd04dc6
2939bbdb	a9ba4650	ac9526e8	be5ee304	a1fad5f0	6a2d519a	63ef8ce2	9a86ee22
c089c2b8	43242ef6	a51e03aa	9cf2d0a4	83c061ba	9be96a4d	8fe51550	ba645bd6
2826a2f9	a73a3ae1	4ba99586	ef5562e9	c72fefd3	f752f7da	3f046f69	77fa0a59
80e4a915	87b08601	9b09e6ad	3b3ee593	e990fd5a	9e34d797	2cf0b7d9	022b8b51
96d5ac3a	017da67d	d1cf3ed6	7c7d2d28	1f9f25cf	adf2b89b	5ad6b472	5a88f54c
e029ac71	e019a5e6	47b0acf0	ed93fa9b	e8d3c48d	283b57cc	f8d56629	79132e28
785f0191	ed756055	f7960e44	e3d35e8c	15056dd4	88f46dba	03a16125	0564f0bd
c3eb9e15	3c9057a2	97271aec	a93a072a	1b3f6d9b	1e6321f5	f59c66fb	26dcf319
7533d928	b155fdf5	03563482	8aba3cbb	28517711	c20ad9f8	abcc5167	ccad925f
4de81751	3830dc8e	379d5862	9320f991	ea7a90c2	fb3e7bce	5121ce64	774fbe32
a8b6e37e	c3293d46	48de5369	6413e680	a2ae0810	dd6db224	69852dfd	09072166
b39a460a	6445c0dd	586cdecf	1c20c8ae	5bbef7dd	1b588d40	ccd2017f	6bb4e3bb
dda26a7e	3a59ff45	3e350a44	bcb4cdd5	72eacea8	fa6484bb	8d6612ae	bf3c6f47
d29be463	542f5d9e	aec2771b	f64e6370	740e0d8d	e75b1357	f8721671	af537d5d
4040cb08	4eb4e2cc	34d2466a	0115af84	e1b00428	95983a1d	06b89fb4	ce6ea048
6f3f3b82	3520ab82	011a1d4b	277227f8	611560b1	e7933fdc	bb3a792b	344525bd
a08839e1	51ce794b	2f32c9b7	a01fbac9	e01cc87e	bcc7d1f6	cf0111c3	a1e8aac7
1a908749	d44fb9a	d0dadecb	d50ada38	0339c32a	c6913667	8df9317c	e0b12b4f
f79e59b7	43f5bb3a	f2d519ff	27d9459c	bf97222c	15e6fc2a	0f91fc71	9b941525
fae59361	ceb69ceb	c2a86459	12baa8d1	b6c1075e	e3056a0c	10d25065	cb03a442
e0ec6e0e	1698db3b	4c98a0be	3278e964	9f1f9532	e0d392df	d3a0342b	8971f21e

Таблица П4 (окончание)

1b0a7441	4ba3348c	c5be7120	c37632d8	df359f8d	9b992f2e	e60b6f47	0fe3f11d
e54cd454	1edad891	ce6279cf	cd3e7e6f	1618b166	fd2c1d05	848fd2c5	f6fb2299
f523f357	a6327623	93a83531	56cccd02	acf08162	5a75ebb5	6e163697	88d273cc
de966292	81b949d0	4c50901b	71e65614	e6c6c7bd	327a140a	45e1d006	c3f27b9a
c9aa53fd	62a80f00	bb25bfe2	35bdd2f6	71126905	b2040222	b6cbcf7c	cd769c2b
53113ec0	1640e3d3	38abbd60	2547adf0	ba38209c	f746ce76	77afa1c5	20756060
85cbfe4e	8ae88dd8	7aaaaf9b0	4cf9aa7e	1948c25c	02fb8a8c	01c36ae4	d6ebe1f9
90d4f869	a65cddea0	3f09252d	c208e69f	b74e6132	ce77e25b	578fdf3	3ac372e

## П2. Алгоритм Camellia

Таблицы  $S_1 \dots S_4$  алгоритма Camellia (см. разд. 3.9) приведены, соответственно, в табл. П5–П8.

Таблица П5

112	130	44	236	179	39	192	229	228	133	87	53	234	12	174	65
35	239	107	147	69	25	165	33	237	14	79	78	29	101	146	189
134	184	175	143	124	235	31	206	62	48	220	95	94	197	11	26
166	225	57	202	213	71	93	61	217	1	90	214	81	86	108	77
139	13	154	102	251	204	176	45	116	18	43	32	240	177	132	153
223	76	203	194	52	126	118	5	109	183	169	49	209	23	4	215
20	88	58	97	222	27	17	28	50	15	156	22	83	24	242	34
254	68	207	178	195	181	122	145	36	8	232	168	96	252	105	80
170	208	160	125	161	137	98	151	84	91	30	149	224	255	100	210
16	196	0	72	163	247	117	219	138	3	230	218	9	63	221	148
135	92	131	2	205	74	144	51	115	103	246	243	157	127	191	226
82	155	216	38	200	55	198	59	129	150	111	75	19	190	99	46
233	121	167	140	159	110	188	142	41	245	249	182	47	253	180	89
120	152	6	106	231	70	113	186	212	37	171	66	136	162	141	250
114	7	185	85	248	238	172	10	54	73	42	104	60	56	241	164
64	40	211	123	187	201	67	193	21	227	173	244	119	199	128	158

**Таблицы замен**

Таблица трактуется следующим образом: входное значение 0 заменяется на 112, 1 — на 130 и т. д. до входного значения 255, которое заменяется на 158.

**Таблица П6**

224	5	88	217	103	78	129	203	201	11	174	106	213	24	93	130
70	223	214	39	138	50	75	66	219	28	158	156	58	202	37	123
13	113	95	31	248	215	62	157	124	96	185	190	188	139	22	52
77	195	114	149	171	142	186	122	179	2	180	173	162	172	216	154
23	26	53	204	247	153	97	90	232	36	86	64	225	99	9	51
191	152	151	133	104	252	236	10	218	111	83	98	163	46	8	175
40	176	116	194	189	54	34	56	100	30	57	44	166	48	229	68
253	136	159	101	135	107	244	35	72	16	209	81	192	249	210	160
85	161	65	250	67	19	196	47	168	182	60	43	193	255	200	165
32	137	0	144	71	239	234	183	21	6	205	181	18	126	187	41
15	184	7	4	155	148	33	102	230	206	237	231	59	254	127	197
164	55	177	76	145	110	141	118	3	45	222	150	38	125	198	92
211	242	79	25	63	220	121	29	82	235	243	109	94	251	105	178
240	49	12	212	207	140	226	117	169	74	87	132	17	69	27	245
228	14	115	170	241	221	89	20	108	146	84	208	120	112	227	73
128	80	167	246	119	147	134	131	42	199	91	233	238	143	1	61

**Таблица П7**

56	65	22	118	217	147	96	242	114	194	171	154	117	6	87	160
145	247	181	201	162	140	210	144	246	7	167	39	142	178	73	222
67	92	215	199	62	245	143	103	31	24	110	175	47	226	133	13
83	240	156	101	234	163	174	158	236	128	45	107	168	43	54	166
197	134	77	51	253	102	88	150	58	9	149	16	120	216	66	204
239	38	229	97	26	63	59	130	182	219	212	152	232	139	2	235
10	44	29	176	111	141	136	14	25	135	78	11	169	12	121	17
127	34	231	89	225	218	61	200	18	4	116	84	48	126	180	40
85	104	80	190	208	196	49	203	42	173	15	202	112	255	50	105

**Таблица П7 (окончание)**

8	98	0	36	209	251	186	237	69	129	115	109	132	159	238	74
195	46	193	1	230	37	72	153	185	179	123	249	206	191	223	113
41	205	108	19	100	155	99	157	192	75	183	165	137	95	177	23
244	188	211	70	207	55	94	71	148	250	252	91	151	254	90	172
60	76	3	53	243	35	184	93	106	146	213	33	68	81	198	125
57	131	220	170	124	119	86	5	27	164	21	52	30	28	248	82
32	20	233	189	221	228	161	224	138	241	214	122	187	227	64	79

**Таблица П8**

112	44	179	192	228	87	234	174	35	107	69	165	237	79	29	146
134	175	124	31	62	220	94	11	166	57	213	93	217	90	81	108
139	154	251	176	116	43	240	132	223	203	52	118	109	169	209	4
20	58	222	17	50	156	83	242	254	207	195	122	36	232	96	105
170	160	161	98	84	30	224	100	16	0	163	117	138	230	9	221
135	131	205	144	115	246	157	191	82	216	200	198	129	111	19	99
233	167	159	188	41	249	47	180	120	6	231	113	212	171	136	141
114	185	248	172	54	42	60	241	64	211	187	67	21	173	119	128
130	236	39	229	133	53	12	65	239	147	25	33	14	78	101	189
184	143	235	206	48	95	197	26	225	202	71	61	1	214	86	77
13	102	204	45	18	32	177	153	76	194	126	5	183	49	23	215
88	97	27	28	15	22	24	34	68	178	181	145	8	168	252	80
208	125	137	151	91	149	255	210	196	72	247	219	3	218	63	148
92	2	74	51	103	243	127	226	155	38	55	59	150	75	190	46
121	140	110	142	245	182	253	89	152	106	70	186	37	66	162	250
7	85	238	10	73	104	56	164	40	123	201	193	227	244	199	158

### П3. Алгоритм CAST-128

Таблицы замен  $S_1 \dots S_8$  алгоритма CAST-128 (см. разд. 3.10) приведены, соответственно, в табл. П9–П16.

Таблица П9

30fb40d4	9fa0ff0b	6becccd2f	3f258c7a	1e213f2f	9c004dd3	6003e540	cf9fc949
bfd4af27	88bbbbdb5	e2034090	98d09675	6e63a0e0	15c361d2	c2e7661d	22d4ff8e
28683b6f	c07fd059	ff2379c8	775f50e2	43c340d3	df2f8656	887ca41a	a2d2bd2d
a1c9e0d6	346c4819	61b76d87	22540f2f	2abe32e1	aa54166b	22568e3a	a2d341d0
66db40c8	a784392f	004dff2f	2db9d2de	97943fac	4a97c1d8	527644b7	b5f437a7
b82cbaef	d751d159	6ff7f0ed	5a097a1f	827b68d0	90ecf52e	22b0c054	bc8e5935
4b6d2f7f	50bb64a2	d2664910	bee5812d	b7332290	e93b159f	b48ee411	4bff345d
fd45c240	ad31973f	c4f6d02e	55fc8165	d5b1caad	a1ac2dae	a2d4b76d	c19b0c50
882240f2	0c6e4f38	a4e4bfd7	4f5ba272	564c1d2f	c59c5319	b949e354	b04669fe
b1b6ab8a	c71358dd	6385c545	110f935d	57538ad5	6a390493	e63d37e0	2a54f6b3
3a787d5f	6276a0b5	19a6fcdf	7a42206a	29f9d4d5	f61b1891	bb72275e	aa508167
38901091	c6b505eb	84c7cb8c	2ad75a0f	874a1427	a2d1936b	2ad286af	aa56d291
d7894360	425c750d	93b39e26	187184c9	6c00b32d	73e2bb14	a0bebcb3c	54623779
64459eab	3f328b82	7718cf82	59a2cea6	04ee002e	89fe78e6	3fab0950	325ff6c2
81383f05	6963c5c8	76cb5ad6	d49974c9	ca180dcf	380782d5	c7fa5cf6	8ac31511
35e79e13	47da91d0	f40f9086	a7e2419e	31366241	051ef495	aa573b04	4a805d8d
548300d0	00322a3c	bf64cddf	ba57a68e	75c6372b	50afd341	a7c13275	915a0bf5
6b54bfab	2b0b1426	ab4cc9d7	449ccd82	f7fbf265	ab85c5f3	1b55db94	aad4e324
cfa4bd3f	2deaa3e2	9e204d02	c8bd25ac	eadf55b3	d5bd9e98	e31231b2	2ad5ad6c
954329de	adbe4528	d8710f69	aa51c90f	aa786bf6	22513f1e	aa51a79b	2ad344cc
7b5a41f0	d37cfbad	1b069505	41ece491	b4c332e6	032268d4	c9600acc	ce387e6d
bf6bb16c	6a70fb78	0d03d9c9	d4df39de	e01063da	4736f464	5ad328d8	b347cc96
75bb0fc3	98511bfb	4ffbcc35	b58bcf6a	e11f0abc	bfc5fe4a	a70aec10	ac39570a
3f04442f	6188b153	e0397a2e	5727cb79	9ceb418f	1cacd68d	2ad37c96	0175cb9d
c69dff09	c75b65f0	d9db40d8	ec0e7779	4744ead4	b11c3274	dd24cb9e	7e1c54bd
f01144f9	d2240eb1	9675b3fd	a3ac3755	d47c27af	51c85f4d	56907596	a5bb15e6
580304f0	ca042cf1	011a37ea	8dbfaadb	35ba3e4a	3526ffa0	c37b4d09	bc306ed9
98a52666	5648f725	ff5e569d	0ced63d0	7c63b2cf	700b45e1	d5ea50f1	85a92872
af1fbda7	d4234870	a7870bf3	2d3b4d79	42e04198	0cd0ede7	26470db8	f881814c

**Таблица П9 (окончание)**

474d6ad7	7c0c5e5c	d1231959	381b7298	f5d2f4db	ab838653	6e2f1e23	83719c9e
bd91e046	9a56456e	dc39200c	20c8c571	962bda1c	e1e696ff	b141ab08	7cca89b9
1a69e783	02cc4843	a2f7c579	429ef47d	427b169c	5ac9f049	dd8f0f00	5c8165bf

**Таблица П10**

1f201094	ef0ba75b	69e3cf7e	393f4380	fe61cf7a	eec5207a	55889c94	72fc0651
ada7ef79	4e1d7235	d55a63ce	de0436ba	99c430ef	5f0c0794	18dcdb7d	a1d6eff3
a0b52f7b	59e83605	ee15b094	e9ffd909	dc440086	ef944459	ba83ccb3	e0c3cdfb
d1da4181	3b092ab1	f997f1c1	a5e6cf7b	01420ddb	e4e7ef5b	25a1ff41	e180f806
1fc41080	179bee7a	d37ac6a9	fe5830a4	98de8b7f	77e83f4e	79929269	24fa9f7b
e113c85b	acc40083	d7503525	f7ea615f	62143154	0d554b63	5d681121	c866c359
3d63cf73	cee234c0	d4d87e87	5c672b21	071f6181	39f7627f	361e3084	e4eb573b
602f64a4	d63acd9c	1bbc4635	9e81032d	2701f50c	99847ab4	a0e3df79	ba6cf38c
10843094	2537a95e	f46f6ffe	a1ff3b1f	208cfb6a	8f458c74	d9e0a227	4ec73a34
fc884f69	3e4de8df	ef0e0088	3559648d	8a45388c	1d804366	721d9bfd	a58684bb
e8256333	844e8212	128d8098	fed33fb4	ce280ae1	27e19ba5	d5a6c252	e49754bd
c5d655dd	eb667064	77840b4d	a1b6a801	84db26a9	e0b56714	21f043b7	e5d05860
54f03084	066ff472	a31aa153	dadc4755	b5625dbf	68561be6	83ca6b94	2d6ed23b
eccf01db	a6d3d0ba	b6803d5c	af77a709	33b4a34c	397bc8d6	5ee22b95	5f0e5304
81ed6f61	20e74364	b45e1378	de18639b	881ca122	b96726d1	8049a7e8	22b7da7b
5e552d25	5272d237	79d2951c	c60d894c	488cb402	1ba4fe5b	a4b09f6b	1ca815cf
a20c3005	8871df63	b9de2fcb	0cc6c9e9	0beeff53	e3214517	b4542835	9f63293c
ee41e729	6e1d2d7c	50045286	1e6685f3	f33401c6	30a22c95	31a70850	60930f13
73f98417	a1269859	ec645c44	52c877a9	cdff33a6	a02b1741	7cbad9a2	2180036f
50d99c08	cb3f4861	c26bd765	64a3f6ab	80342676	25a75e7b	e4e6d1fc	20c710e6
cdff0b680	17844d3b	31eef84d	7e0824e4	2ccb49eb	846a3bae	8ff77888	ee5d60f6
7af75673	2fdd5cdb	a11631c1	30f66f43	b3faec54	157fd7fa	ef8579cc	d152de58
db2ffd5e	8f32ce19	306af97a	02f03ef8	99319ad5	c242fa0f	a7e3ebb0	c68e4906
b8da230c	80823028	dcdef3c8	d35fb171	088a1bc8	bec0c560	61a3c9e8	bca8f54d

## Таблицы замен

Таблица П10 (окончание)

c72feffa	22822e99	82c570b4	d8d94e89	8b1c34bc	301e16e6	273be979	b0ffea6
61d9b8c6	00b24869	b7ffce3f	08dc283b	43daf65a	f7e19798	7619b72f	8f1c9ba4
dc8637a0	16a7d3b1	9fc393b7	a7136eeb	c6bcc63e	1a513742	ef6828bc	520365d6
2d6a77ab	3527ed4b	821fd216	095c6e2e	db92f2fb	5eea29cb	145892f5	91584f7f
5483697b	2667a8cc	85196048	8c4bacea	833860d4	0d23e0f9	6c387e8a	0ae6d249
b284600c	d835731d	dcb1c647	ac4c56ea	3ebd81b3	230eabb0	6438bc87	f0b5b1fa
8f5ea2b3	fc184642	0a036b7a	4fb089bd	649da589	a345415e	5c038323	3e5d3bb9
43d79572	7e6dd07c	06fdf1e	6c6cc4ef	7160a539	73bfbe70	83877605	4523ecf1

Таблица П11

8defc240	25fa5d9f	eb903dbf	e810c907	47607fff	369fe44b	8c1fc644	aececa90
beb1f9bf	eefbcaea	e8cf1950	51df07ae	920e8806	f0ad0548	e13c8d83	927010d5
11107d9f	07647db9	b2e3e4d4	3d4f285e	b9afa820	fade82e0	a067268b	8272792e
553fb2c0	489ae22b	d4ef9794	125e3fbc	21fffcee	825b1bfd	9255c5ed	1257a240
4e1a8302	bae07fff	528246e7	8e57140e	3373f7bf	8c9f8188	a6fc4ee8	c982b5a5
a8c01db7	579fc264	67094f31	f2bd3f5f	40fff7c1	1fb78dfc	8e6bd2c1	437be59b
99b03dbf	b5dbc64b	638dc0e6	55819d99	a197c81c	4a012d6e	c5884a28	ccc36f71
b843c213	6c0743f1	8309893c	0feddd5f	2f7fe850	d7c07f7e	02507fbf	5afb9a04
a747d2d0	1651192e	af70bf3e	58c31380	5f98302e	727cc3c4	0a0fb402	0f7fef82
8c96fdad	5d2c2aae	8ee99a49	50da88b8	8427f4a0	1eac5790	796fb449	8252dc15
efbd7d9b	a672597d	ada840d8	45f54504	fa5d7403	e83ec305	4f91751a	925669c2
23efe941	a903f12e	60270df2	0276e4b6	94fd6574	927985b2	8276dbcb	02778176
f8af918d	4e48f79e	8f616ddf	e29d840e	842f7d83	340ce5c8	96bbb682	93b4b148
ef303cab	984faf28	779faf9b	92dc560d	224d1e20	8437aa88	7d29dc96	2756d3dc
8b907cee	b51fd240	e7c07ce3	e566b4a1	c3e9615e	3cf8209d	6094d1e3	cd9ca341
5c76460e	00ea983b	d4d67881	fd47572c	f76cedd9	bda8229c	127dadaa	438a074e
1f97c090	081bdb8a	93a07ebe	b938ca15	97b03cff	3dc2c0f8	8d1ab2ec	64380e51
68cc7fbf	d90f2788	12490181	5de5ffd4	dd7ef86a	76a2e214	b9a40368	925d958f
4b39ffff	ba39aee9	a4ffd30b	faf7933b	6d498623	193cbcfa	27627545	825cf47a

Таблица П11 (окончание)

61bd8ba0	d11e42d1	cead04f4	127ea392	10428db7	8272a972	9270c4a8	127de50b
285ba1c8	3c62f44f	35c0eaa5	e805d231	428929fb	b4fcdf82	4fb66a53	0e7dc15b
1f081fab	108618ae	fcfd086d	f9ff2889	694bcc11	236a5cae	12deca4d	2c3f8cc5
d2d02dfe	f8ef5896	e4cf52da	95155b67	494a488c	b9b6a80c	5c8f82bc	89d36b45
3a609437	ec00c9a9	44715253	0a874b49	d773bc40	7c34671c	02717ef6	4feb5536
a2d02fff	d2bf60c4	d43f03c0	50b4ef6d	07478cd1	006e1888	a2e53f55	b9e6d4bc
a2048016	97573833	d7207d67	de0f8f3d	72f87b33	abcc4f33	7688c55d	7b00a6b0
947b0001	570075d2	f9bb88f8	8942019e	4264a5ff	856302e0	72dbd92b	ee971b69
6ea22fde	5f08ae2b	af7a616d	e5c98767	cfl1feb2	61efc8c2	f1ac2571	cc8239c2
67214cb8	b1e583d1	b7dc3e62	7f10bdce	f90a5c38	0ff0443d	606e6dc6	60543a49
5727c148	2be98a1d	8ab41738	20e1be24	af96da0f	68458425	99833be5	600d457d
282f9350	8334b362	d91d1120	2b6d8da0	642b1e31	9c305a00	52bce688	1b03588a
f7baefd5	4142ed9c	a4315c11	83323ec5	dfe4636	a133c501	e9d3531c	ee353783

Таблица П12

9db30420	1fb6e9de	a7be7bef	d273a298	4a4f7bdb	64ad8c57	85510443	fa020ed1
7e287aff	e60fb663	095f35a1	79ebf120	fd059d43	6497b7b1	f3641f63	241e4adf
28147f5f	4fa2b8cd	c9430040	0cc32220	fd30b30	c0a5374f	1d2d00d9	24147b15
ee4d111a	0fcfa5167	71ff904c	2d195ffe	1a05645f	0c13fefc	081b08ca	05170121
80530100	e83e5efe	ac9af4f8	7fe72701	d2b8ee5f	06df4261	bb9e9b8a	7293ea25
ce84ffd9	f5718801	3dd64b04	a26f263b	7ed48400	547eebe6	446d4ca0	6cf3d6f5
2649abdf	aea0c7f5	36338cc1	503f7e93	d3772061	11b638e1	72500e03	f80eb2bb
abe0502e	ec8d77de	57971e81	e14f6746	c9335400	6920318f	081dbb99	ffc304a5
4d351805	7f3d5ce3	a6c866c6	5d5bcc9	daec6fea	9f926f91	9f46222f	3991467d
a5bf6d8e	1143c44f	43958302	d0214eef	022083b8	3fb6180c	18f8931e	281658e6
26486e3e	8bd78a70	7477e4c1	b506e07c	f32d0a25	79098b02	e4eabb81	28123b23
69dead38	1574ca16	df871b62	211c40b7	a51a9ef9	0014377b	041e8ac8	09114003
bd59e4d2	e3d156d5	4fe876d5	2f91a340	557be8de	00eae4a7	0ce5c2ec	4db4bba6
e756bdff	dd3369ac	ec17b035	06572327	99afc8b0	56c8c391	6b65811c	5e146119

## Таблицы замен

Таблица П12 (окончание)

6e85cb75	be07c002	c2325577	893ff4ec	5bbfc92d	d0ec3b25	b7801ab7	8d6d3b24
20c763ef	c366a5fc	9c382880	0ace3205	aac9548a	eca1d7c7	041afa32	1d16625a
6701902c	9b757a54	31d477f7	9126b031	36cc6fdb	c70b8b46	d9e66a48	56e55a79
026a4ceb	52437eff	2f8f76b4	0df980a5	8674cde3	edda04eb	17a9be04	2c18f4df
b7747f9d	ab2af7b4	efc34d20	2e096b7c	1741a254	e5b6a035	213d42f6	2c1c7c26
61c2f50f	6552daf9	d2c231f8	25130f69	d8167fa2	0418f2c8	001a96a6	0d1526ab
63315c21	5e0a72ec	49bafefd	187908d9	8d0dbd86	311170a7	3e9b640c	cc3e10d7
d5cad3b6	0caec388	f73001e1	6c728aff	71eae2a1	1f9af36e	cfcbd12f	c1de8417
ac07be6b	cb44a1d8	8b9b0f56	013988c3	b1c52fea	b4be31cd	d8782806	12a3a4e2
6f7de532	58fd7eb6	d01ee900	24adffc2	f4990fc5	9711aac5	001d7b95	82e5e7d2
109873f6	00613096	c32d9521	ada121ff	29908415	7fb977f	af9eb3db	29c9ed2a
5ce2a465	a730f32c	d0aa3fe8	8a5cc091	d49e2ce7	0ce454a9	d60acd86	015f1919
77079103	dea03af6	78a8565e	dee356df	21f05cbe	8b75e387	b3c50651	b8a5c3ef
d8eeb6d2	e523be77	c2154529	2f69efdf	afe67afb	f470c4b2	f3e0eb5b	d6cc9876
39e4460c	1fda8538	1987832f	ca007367	a99144f8	296b299e	492fc295	9266beab
b5676e69	9bd3ddda	df7e052f	db25701c	1b5e51ee	f65324e6	6acf36c	0316cc04
8644213e	b7dc59d0	7965291f	ccd6fd43	41823979	932bcd6	b657c34d	4edfd282
7ae5290c	3cb9536b	851e20fe	9833557e	13ecf0b0	d3ffb372	3f85c5c1	0aef7ed2

Таблица П13

7ec90c04	2c6e74b9	9b0e66df	a6337911	b86a7fff	1dd358f5	44dd9d44	1731167f
08fbf1fa	e7f511cc	d2051b00	735aba00	2ab722d8	386381cb	acf6243a	69befd7a
e6a2e77f	f0c720cd	c4494816	ccf5c180	38851640	15b0a848	e68b18cb	4caadef
5f480a01	0412b2aa	259814fc	41d0efe2	4e40b48d	248eb6fb	8dba1cfe	41a99b02
1a550a04	ba8f65cb	7251f4e7	95a51725	c106ecd7	97a5980a	c539b9aa	4d79fe6a
f2f3f763	68af8040	ed0c9e56	11b4958b	e1eb5a88	8709e6b0	d7e07156	4e29fea7
6366e52d	02d1c000	c4ac8e05	9377f571	0c05372a	578535f2	2261be02	d642a0c9
df13a280	74b55bd2	682199c0	d421e5ec	53fb3ce8	c8adedb3	28a87fc9	3d959981
5c1ff900	fe38d399	0c4eff0b	062407ea	aa2f4fb1	4fb96976	90c79505	b0a8a774

Таблица П13 (окончание)

ef55a1ff	e59ca2c2	a6b62d27	e66a4263	df65001f	0ec50966	dfdd55bc	29de0655
911e739a	17af8975	32c7911c	89f89468	0d01e980	524755f4	03b63cc9	0cc844b2
bcf3f0aa	87ac36e9	e53a7426	01b3d82b	1a9e7449	64ee2d7e	cddb1da	01c94910
b868bf80	0d26f3fd	9342ede7	04a5c284	636737b6	50f5b616	f24766e3	8eca36c1
136e05db	fef18391	fb887a37	d6e7f7d4	c7fb7dc9	3063fcdf	b6f589de	ec2941da
26e46695	b7566419	f654efc5	d08d58b7	48925401	c1bacb7f	e5ff550f	b6083049
5bb5d0e8	87d72e5a	ab6a6ee1	223a66ce	c62bf3cd	9e0885f9	68cb3e47	086c010f
a21de820	d18b69de	f3f65777	fa02c3f6	407edac3	ccb3d550	1793084d	b0d70eba
0ab378d5	d951fb0c	ded7da56	4124bbe4	94ca0b56	0f5755d1	e0e1e56e	6184b5be
580a249f	94f74bc0	e327888e	9f7b5561	c3dc0280	05687715	646c6bd7	44904db3
66b4f0a3	c0f1648a	697ed5af	49e92ff6	309e374f	2cb6356a	85808573	4991f840
76f0ae02	083be84d	28421c9a	44489406	736e4cb8	c1092910	8bc95fc6	7d869cf4
134f616f	2e77118d	b31b2be1	aa90b472	3ca5d717	7d161bba	9cad9010	af462ba2
9fe459d2	45d34559	d9f2da13	dbc65487	f3e4f94e	176d486f	097c13ea	631da5c7
445f7382	175683f4	cdc66a97	70be0288	b3cdcf72	6e5dd2f3	20936079	459b80a5
be60e2db	a9c23101	eba5315c	224e42f2	1c5c1572	f6721b2c	1ad2fff3	8c25404e
324ed72f	4067b7fd	0523138e	5ca3bc78	dc0fd66e	75922283	784d6b17	58ebb16e
44094f85	3f481d87	fcfeae7b	77b5ff76	8c2302bf	aaf47556	5f46b02a	2b092801
3d38f5f7	0ca81f36	52af4a8a	66d5e7c0	df3b0874	95055110	1b5ad7a8	f61ed5ad
6cf6e479	20758184	d0cefa65	88f7be58	4a046826	0ff6f8f3	a09c7f70	5346aba0
5ce96c28	e176eda3	6bac307f	376829d2	85360fa9	17e3fe2a	24b79767	f5a96b20
d6cd2595	68ff1ebf	7555442c	f19f06be	f9e0659a	eeb9491d	34010718	bb30cab8
e822fe15	88570983	750e6249	da627e55	5e76ffa8	b1534546	6d47de08	efe9e7d4

Таблица П14

f6fa8f9d	2cac6ce1	4ca34867	e2337f7c	95db08e7	016843b4	eced5cbc	325553ac
bf9f0960	dfa1e2ed	83f0579d	63ed86b9	1ab6a6b8	de5ebe39	f38ff732	8989b138
33f14961	c01937bd	f506c6da	e4625e7e	a308ea99	4e23e33c	79cbd7cc	48a14367
a3149619	fec94bd5	a114174a	eaa01866	a084db2d	09a8486f	a888614a	2900af98

## Таблицы замен

Таблица П14 (окончание)

01665991	e1992863	c8f30c60	2e78ef3c	d0d51932	cf0fec14	f7ca07d2	d0a82072
fd41197e	9305a6b0	e86be3da	74bed3cd	372da53c	4c7f4448	dab5d440	6dba0ec3
083919a7	9fbaeed9	49dbcfb0	4e670c53	5c3d9c01	64bdb941	2c0e636a	ba7dd9cd
ea6f7388	e70bc762	35f29adb	5c4cdd8d	f0d48d8c	b88153e2	08a19866	1ae2eac8
284caf89	aa928223	9334be53	3b3a21bf	16434be3	9aea3906	efe8c36e	f890cdd9
80226dae	c340a4a3	df7e9c09	a694a807	5b7c5ecc	221db3a6	9a69a02f	68818a54
ceb2296f	53c0843a	fe893655	25bfe68a	b4628abc	cf222ebf	25ac6f48	a9a99387
53bddb65	e76ffbe7	e967fd78	0ba93563	8e342bc1	e8a11be9	4980740d	c8087dfc
8de4bf99	a11101a0	7fd37975	da5a26c0	e81f994f	9528cd89	fd339fed	b87834bf
5f04456d	22258698	c9c4c83b	2dc156be	4f628daa	57f55ec5	e2220abe	d2916ebf
4ec75b95	24f2c3c0	42d15d99	cd0d7fa0	7b6e27ff	a8dc8af0	7345c106	f41e232f
35162386	e6ea8926	3333b094	157ec6f2	372b74af	692573e4	e9a9d848	f3160289
3a62ef1d	a787e238	f3a5f676	74364853	20951063	4576698d	b6fad407	592af950
36f73523	4cfb6e87	7da4cec0	6c152daa	cb0396a8	c50dfe5d	fcd707ab	0921c42f
89dff0bb	5fe2be78	448f4f33	754613c9	2b05d08d	48b9d585	dc049441	c8098f9b
7dede786	c39a3373	42410005	6a091751	0ef3c8a6	890072d6	28207682	a9a9f7be
bf32679d	d45b5b75	b353fd00	cbb0e358	830f220a	1f8fb214	d372cf08	cc3c4a13
8cf63166	061c87be	88c98f88	6062e397	47cf8e7a	b6c85283	3cc2acfb	3fc06976
4e8f0252	64d8314d	da3870e3	1e665459	c10908f0	513021a5	6c5b68b7	822f8aa0
3007cd3e	74719eef	dc872681	073340d4	7e432fd9	0c5ec241	8809286c	f592d891
08a930f6	957ef305	b7fbffbd	c266e96f	6fe4ac98	b173ecc0	bc60b42a	953498da
fba1ae12	2d4bd736	0f25faab	a4f3fce8	e2969123	257f0c3d	9348af49	361400bc
e8816f4a	3814f200	a3f94043	9c7a54c2	bc704f57	da41e7f9	c25ad33a	54f4a084
b17f5505	59357cbe	edbd15c8	7f97c5ab	ba5ac7b5	b6f6deaf	3a479c3a	5302da25
653d7e6a	54268d49	51a477ea	5017d55b	d7d25d88	44136c76	0404a8c8	b8e5a121
b81a928a	60ed5869	97c55b96	eaec991b	29935913	01fdb7f1	088e8dfa	9ab6f6f5
3b4cbf9f	4a5de3ab	e6051d35	a0e1d855	d36b4cf1	f544edeb	b0e93524	beb8fdbd
a2d762cf	49c92f54	38b5f331	7128a454	48392905	a65b1db8	851c97bd	d675cf2f

Таблица П15

85e04019	332bf567	662dbfff	cfc65693	2a8d7f6f	ab9bc912	de6008a1	2028da1f
0227bce7	4d642916	18fac300	50f18b82	2cb2cb11	b232e75c	4b3695f2	b28707de
a05fbcf6	cd4181e9	e150210c	e24ef1bd	b168c381	fde4e789	5c79b0d8	1e8bfd43
4d495001	38be4341	913cee1d	92a79c3f	089766be	baeeadf4	1286becf	b6eacb19
2660c200	7565bde4	64241f7a	8248dca9	c3b3ad66	28136086	0bd8dfa8	356d1cf2
107789be	b3b2e9ce	0502aa8f	0bc0351e	166bf52a	eb12ff82	e3486911	d34d7516
4e7b3aff	5f43671b	9cf6e037	4981ac83	334266ce	8c9341b7	d0d854c0	cb3a6c88
47bc2829	4725ba37	a66ad22b	7ad61f1e	0c5cbafa	4437f107	b6e79962	42d2d816
0a961288	e1a5c06e	13749e67	72fc081a	b1d139f7	f9583745	cf19df58	bec3f756
c06eba30	07211b24	45c28829	c95e317f	bc8ec511	38bc46e9	c6e6fa14	bae8584a
ad4ebc46	468f508b	7829435f	f124183b	821dba9f	aff60ff4	ea2c4e6d	16e39264
92544a8b	009b4fc3	aba68ced	9ac96f78	06a5b79a	b2856e6e	1aec3ca9	be838688
0e0804e9	55f1be56	e7e5363b	b3a1f25d	f7debb85	61fe033c	16746233	3c034c28
da6d0c74	79aac56c	3ce4e1ad	51f0c802	98f8f35a	1626a49f	eed82b29	1d382fe3
0c4fb99a	bb325778	3ec6d97b	6e77a6a9	cb658b5c	d45230c7	2bd1408b	60c03eb7
b9068d78	a33754f4	f430c87d	c8a71302	b96d8c32	ebd4e7be	be8b9d2d	7979fb06
e7225308	8b75cf77	11ef8da4	e083c858	8d6b786f	5a6317a6	fa5cf7a0	5dda0033
f28ebfb0	f5b9c310	a0eac280	08b9767a	a3d9d2b0	79d34217	021a718d	9ac6336a
2711fd60	438050e3	069908a8	3d7fedc4	826d2bef	4eeb8476	488dcf25	36c9d566
28e74e41	c2610aca	3d49a9cf	bae3b9df	b65f8de6	92aeaf64	3ac7d5e6	9ea80509
f22b017d	a4173f70	dd1e16c3	15e0d7f9	50b1b887	2b9f4fd5	625aba82	6a017962
2ec01b9c	15488aa9	d716e740	40055a2c	93d29a22	e32dbf9a	058745b9	3453dc1e
d699296e	496cff6f	1c9f4986	dfe2ed07	b87242d1	19de7eae	053e561a	15ad6f8c
66626c1c	7154c24c	ea082b2a	93eb2939	17dcb0f0	58d4f2ae	9ea294fb	52cf564c
9883fe66	2ec40581	763953c3	01d6692e	d3a0c108	a1e7160e	e4f2dfa6	693ed285
74904698	4c2b0edd	4f757656	5d393378	a132234f	3d321c5d	c3f5e194	4b269301
c79f022f	3c997e7e	5e4f9504	3ffafbbd	76f7ad0e	296693f4	3d1fce6f	c61e45be
d3b5ab34	f72bf9b7	1b0434c0	4e72b567	5592a33d	b5229301	cf2a87f	60aeb767
1814386b	30bcc33d	38a0c07d	fd1606f2	c363519b	589dd390	5479f8e6	1cb8d647

## Таблицы замен

Таблица П15 (окончание)

97fd61a9	ea7759f4	2d57539d	569a58cf	e84e63ad	462e1b78	6580f87e	f3817914
91da55f4	40a230f3	d1988f35	b6e318d2	3ffa50bc	3d40f021	c3c0bdae	4958c24c
518f36b2	84b1d370	0fedce83	878ddada	f2a279c7	94e01be8	90716f4b	954b8aa3

Таблица П16

e216300d	bbddffffc	a7ebdabd	35648095	7789f8b7	e6c1121b	0e241600	052ce8b5
11a9cfb0	e5952f11	ece7990a	9386d174	2a42931c	76e38111	b12def3a	37ddddfc
de9adeb1	0a0cc32c	be197029	84a00940	bb243a0f	b4d137cf	b44e79f0	049eedfd
0b15a15d	480d3168	8bbbde5a	669ded42	c7ece831	3f8f95e7	72df191b	7580330d
94074251	5c7dcdfa	abbe6d63	aa402164	b301d40a	02e7d1ca	53571dae	7a3182a2
12a8ddec	fdcaa335d	176f43e8	71fb46d4	38129022	ce949ad4	b84769ad	965bd862
82f3d055	66fb9767	15b80b4e	1d5b47a0	4cfde06f	c28ec4b8	57e8726e	647a78fc
99865d44	608bd593	6c200e03	39dc5ff6	5d0b00a3	ae63aff2	7e8bd632	70108c0c
bbd35049	2998df04	980cf42a	9b6df491	9e7edd53	06918548	58cb7e07	3b74ef2e
522ffffb1	d24708cc	1c7e27cd	a4eb215b	3cf1d2e2	19b47a38	424f7618	35856039
9d17dee7	27eb35e6	c9aff67b	36baf5b8	09c467cd	c18910b1	e11dbf7b	06cd1af8
7170c608	2d5e3354	d4de495a	64c6d006	bcc0c62c	3dd00db3	708f8f34	77d51b42
264f620f	24b8d2bf	15c1b79e	46a52564	f8d7e54e	3e378160	7895cda5	859c15a5
e6459788	c37bc75f	db07ba0c	0676a3ab	7f229b1e	31842e7b	24259fd7	f8bef472
835ffcb8	6df4c1f2	96f5b195	fd0af0fc	b0fe134c	e2506d3d	4f9b12ea	f215f225
a223736f	9fb4c428	25d04979	34c713f8	c4618187	ea7a6e98	7cd16efc	1436876c
f1544107	bedeeee14	56e9af27	a04aa441	3cf7c899	92ecbae6	dd67016d	151682eb
a842eedf	fdb460b4	f1907b75	20e3030f	24d8c29e	e139673b	efa63fb8	71873054
b6f2cf3b	9f326442	cb15a4cc	b01a4504	f1e47d8d	844a1be5	bae7fdfc	42cbda70
cd7dae0a	57e85b7a	d53f5af6	20cf4d8c	cea4d428	79d130a4	3486ebfb	33d3cddc
77853b53	37effcb5	c5068778	e580b3e6	4e68b8f4	c5c8b37e	0d809ea2	398feb7c
132a4f94	43b7950e	2fee7d1c	223613bd	dd06caa2	37df932b	c4248289	acf3ebc3
5715f6b7	ef3478dd	f267616f	c148cbe4	9052815e	5e410fab	b48a2465	2eda7fa4
e87b40e4	e98ea084	5889e9e1	efd390fc	dd07d35b	db485694	38d7e5b2	57720101

**Таблица П16 (окончание)**

730edebc	5b643113	94917e4f	503c2fba	646f1282	7523d24a	e0779695	f9c17a8f
7a5b2121	d187b896	29263a4d	ba510cdf	81f47c9f	ad1163ed	ea7b5965	1a00726e
11403092	00da6d77	4a0cdd61	ad1f4603	605bdfb0	9eedc364	22ebe6a8	cee7d28a
a0e736a0	5564a6b9	10853209	c7eb8f37	2de705ca	8951570f	df09822b	bd691a6c
aa12e4f2	87451c0f	e0f6a27a	3ada4819	4cf1764f	0d771c2b	67cdb156	350d8384
5938fa0f	42399ef3	36997b07	0e84093d	4aa93e61	8360d87b	1fa98b0c	1149382c
e97625a5	0614d1b7	0e25244b	0c768347	589e8d82	0d2059d1	a466bb1e	f8da0a82
04f19130	ba6e4ec0	99265164	1ee7230d	50b2ad80	eaee6801	8db2a283	ea8bf59e

## П4. Алгоритм CAST-256

Таблицы замен  $S_1 \dots S_4$  алгоритма CAST-256 (см. разд. 3.11) приведены, соответственно, в табл. П17–П20.

**Таблица П17**

30fb40d4	9fa0ff0b	6beccd2f	3f258c7a	1e213f2f	9c004dd3	6003e540	cf9fc949
bfd4af27	88bbbdb5	e2034090	98d09675	6e63a0e0	15c361d2	c2e7661d	22d4ff8e
28683b6f	c07fd059	ff2379c8	775f50e2	43c340d3	df2f8656	887ca41a	a2d2bd2d
a1c9e0d6	346c4819	61b76d87	22540f2f	2abe32e1	aa54166b	22568e3a	a2d341d0
66db40c8	a784392f	004dff2f	2db9d2de	97943fac	4a97c1d8	527644b7	b5f437a7
b82cbaef	d751d159	6ff7f0ed	5a097a1f	827b68d0	90ecf52e	22b0c054	bc8e5935
4b6d2f7f	50bb64a2	d2664910	bee5812d	b7332290	e93b159f	b48ee411	4bff345d
fd45c240	ad31973f	c4f6d02e	55fc8165	d5b1caad	a1ac2dae	a2d4b76d	c19b0c50
882240f2	0c6e4f38	a4e4bfd7	4f5ba272	564c1d2f	c59c5319	b949e354	b04669fe
b1b6ab8a	c71358dd	6385c545	110f935d	57538ad5	6a390493	e63d37e0	2a54f6b3
3a787d5f	6276a0b5	19a6fcdf	7a42206a	29f9d4d5	f61b1891	bb72275e	aa508167
38901091	c6b505eb	84c7cb8c	2ad75a0f	874a1427	a2d1936b	2ad286af	aa56d291
d7894360	425c750d	93b39e26	187184c9	6c00b32d	73e2bb14	a0bebcb3c	54623779
64459eab	3f328b82	7718cf82	59a2cea6	04ee002e	89fe78e6	3fab0950	325ff6c2
81383f05	6963c5c8	76cb5ad6	d49974c9	ca180dcf	380782d5	c7fa5cf6	8ac31511
35e79e13	47da91d0	f40f9086	a7e2419e	31366241	051ef495	aa573b04	4a805d8d

## Таблицы замен

Таблица П17 (окончание)

548300d0	00322a3c	bf64cddf	ba57a68e	75c6372b	50afd341	a7c13275	915a0bf5
6b54bfab	2b0b1426	ab4cc9d7	449ccd82	f7fbf265	ab85c5f3	1b55db94	aad4e324
cfa4bd3f	2deaa3e2	9e204d02	c8bd25ac	eadf55b3	d5bd9e98	e31231b2	2ad5ad6c
954329de	adbe4528	d8710f69	aa51c90f	aa786bf6	22513f1e	aa51a79b	2ad344cc
7b5a41f0	d37cfbad	1b069505	41ece491	b4c332e6	032268d4	c9600acc	ce387e6d
bf6bb16c	6a70fb78	0d03d9c9	d4df39de	e01063da	4736f464	5ad328d8	b347cc96
75bb0fc3	98511bfb	4ffbcc35	b58bcf6a	e11f0abc	bfc5fe4a	a70aec10	ac39570a
3f04442f	6188b153	e0397a2e	5727cb79	9ceb418f	1cacd68d	2ad37c96	0175cb9d
c69dff09	c75b65f0	d9db40d8	ec0e7779	4744ead4	b11c3274	dd24cb9e	7e1c54bd
f01144f9	d2240eb1	9675b3fd	a3ac3755	d47c27af	51c85f4d	56907596	a5bb15e6
580304f0	ca042cf1	011a37ea	8dbfaadb	35ba3e4a	3526ffa0	c37b4d09	bc306ed9
98a52666	5648f725	ff5e569d	0ced63d0	7c63b2cf	700b45e1	d5ea50f1	85a92872
af1fbda7	d4234870	a7870bf3	2d3b4d79	42e04198	0cd0ede7	26470db8	f881814c
474d6ad7	7c0c5e5c	d1231959	381b7298	f5d2f4db	ab838653	6e2f1e23	83719c9e
bd91e046	9a56456e	dc39200c	20c8c571	962bda1c	e1e696ff	b141ab08	7cca89b9
1a69e783	02cc4843	a2f7c579	429ef47d	427b169c	5ac9f049	dd8f0f00	5c8165bf

Таблица П18

1f201094	ef0ba75b	69e3cf7e	393f4380	fe61cf7a	eec5207a	55889c94	72fc0651
ada7ef79	4e1d7235	d55a63ce	de0436ba	99c430ef	5f0c0794	18dcdb7d	a1d6eff3
a0b52f7b	59e83605	ee15b094	e9ffd909	dc440086	ef944459	ba83ccb3	e0c3cdfb
d1da4181	3b092ab1	f997f1c1	a5e6cf7b	01420ddb	e4e7ef5b	25a1ff41	e180f806
1fc41080	179bee7a	d37ac6a9	fe5830a4	98de8b7f	77e83f4e	79929269	24fa9f7b
e113c85b	acc40083	d7503525	f7ea615f	62143154	0d554b63	5d681121	c866c359
3d63cf73	cee234c0	d4d87e87	5c672b21	071f6181	39f7627f	361e3084	e4eb573b
602f64a4	d63acd9c	1bbc4635	9e81032d	2701f50c	99847ab4	a0e3df79	ba6cf38c
10843094	2537a95e	f46f6ffe	a1ff3b1f	208cfb6a	8f458c74	d9e0a227	4ec73a34
fc884f69	3e4de8df	ef0e0088	3559648d	8a45388c	1d804366	721d9bfd	a58684bb
e8256333	844e8212	128d8098	fed33fb4	ce280ae1	27e19ba5	d5a6c252	e49754bd

Таблица П18 (окончание)

c5d655dd	eb667064	77840b4d	a1b6a801	84db26a9	e0b56714	21f043b7	e5d05860
54f03084	066ff472	a31aa153	dadc4755	b5625dbf	68561be6	83ca6b94	2d6ed23b
eccf01db	a6d3d0ba	b6803d5c	af77a709	33b4a34c	397bc8d6	5ee22b95	5f0e5304
81ed6f61	20e74364	b45e1378	de18639b	881ca122	b96726d1	8049a7e8	22b7da7b
5e552d25	5272d237	79d2951c	c60d894c	488cb402	1ba4fe5b	a4b09f6b	1ca815cf
a20c3005	8871df63	b9de2fcb	0cc6c9e9	0beeff53	e3214517	b4542835	9f63293c
ee41e729	6e1d2d7c	50045286	1e6685f3	f33401c6	30a22c95	31a70850	60930f13
73f98417	a1269859	ec645c44	52c877a9	cdff33a6	a02b1741	7cbad9a2	2180036f
50d99c08	cb3f4861	c26bd765	64a3f6ab	80342676	25a75e7b	e4e6d1fc	20c710e6
cdf0b680	17844d3b	31eef84d	7e0824e4	2ccb49eb	846a3bae	8ff77888	ee5d60f6
7af75673	2fdd5cdb	a11631c1	30f66f43	b3faec54	157fd7fa	ef8579cc	d152de58
db2ffd5e	8f32ce19	306af97a	02f03ef8	99319ad5	c242fa0f	a7e3ebb0	c68e4906
b8da230c	80823028	dcdef3c8	d35fb171	088a1bc8	bec0c560	61a3c9e8	bca8f54d
c72feffa	22822e99	82c570b4	d8d94e89	8b1c34bc	301e16e6	273be979	b0ffeaa6
61d9b8c6	00b24869	b7ffce3f	08dc283b	43daf65a	f7e19798	7619b72f	8f1c9ba4
dc8637a0	16a7d3b1	9fc393b7	a7136eeb	c6bcc63e	1a513742	ef6828bc	520365d6
2d6a77ab	3527ed4b	821fd216	095c6e2e	db92f2fb	5eea29cb	145892f5	91584f7f
5483697b	2667a8cc	85196048	8c4bacea	833860d4	0d23e0f9	6c387e8a	0ae6d249
b284600c	d835731d	dcb1c647	ac4c56ea	3ebd81b3	230eabb0	6438bc87	f0b5b1fa
8f5ea2b3	fc184642	0a036b7a	4fb089bd	649da589	a345415e	5c038323	3e5d3bb9
43d79572	7e6dd07c	06dfdf1e	6c6cc4ef	7160a539	73bfbe70	83877605	4523ecf1

Таблица П19

8defc240	25fa5d9f	eb903dbf	e810c907	47607fff	369fe44b	8c1fc644	aececa90
beb1f9bf	eefbcaea	e8cf1950	51df07ae	920e8806	f0ad0548	e13c8d83	927010d5
11107d9f	07647db9	b2e3e4d4	3d4f285e	b9afa820	fade82e0	a067268b	8272792e
553fb2c0	489ae22b	d4ef9794	125e3fbc	21fffcee	825b1bfd	9255c5ed	1257a240
4e1a8302	bae07fff	528246e7	8e57140e	3373f7bf	8c9f8188	a6fc4ee8	c982b5a5

## Таблицы замен

Таблица П19 (окончание)

a8c01db7	579fc264	67094f31	f2bd3f5f	40fff7c1	1fb78dfc	8e6bd2c1	437be59b
99b03dbf	b5dbc64b	638dc0e6	55819d99	a197c81c	4a012d6e	c5884a28	ccc36f71
b843c213	6c0743f1	8309893c	0feddd5f	2f7fe850	d7c07f7e	02507fbf	5afb9a04
a747d2d0	1651192e	af70bf3e	58c31380	5f98302e	727cc3c4	0a0fb402	0f7fef82
8c96fdad	5d2c2aae	8ee99a49	50da88b8	8427f4a0	1eac5790	796fb449	8252dc15
efbd7d9b	a672597d	ada840d8	45f54504	fa5d7403	e83ec305	4f91751a	925669c2
23efe941	a903f12e	60270df2	0276e4b6	94fd6574	927985b2	8276dbcb	02778176
f8af918d	4e48f79e	8f616ddf	e29d840e	842f7d83	340ce5c8	96bbb682	93b4b148
ef303cab	984faf28	779faf9b	92dc560d	224d1e20	8437aa88	7d29dc96	2756d3dc
8b907cee	b51fd240	e7c07ce3	e566b4a1	c3e9615e	3cf8209d	6094d1e3	cd9ca341
5c76460e	00ea983b	d4d67881	fd47572c	f76cedd9	bda8229c	127dadaa	438a074e
1f97c090	081bdb8a	93a07ebe	b938ca15	97b03cff	3dc2c0f8	8d1ab2ec	64380e51
68cc7fbf	d90f2788	12490181	5de5ffd4	dd7ef86a	76a2e214	b9a40368	925d958f
4b39ffff	ba39aee9	a4ffd30b	faf7933b	6d498623	193cbcfa	27627545	825cf47a
61bd8ba0	d11e42d1	cead04f4	127ea392	10428db7	8272a972	9270c4a8	127de50b
285ba1c8	3c62f44f	35c0eaa5	e805d231	428929fb	b4fcdf82	4fb66a53	0e7dc15b
1f081fab	108618ae	fcfd086d	f9ff2889	694bcc11	236a5cae	12deca4d	2c3f8cc5
d2d02dfe	f8ef5896	e4cf52da	95155b67	494a488c	b9b6a80c	5c8f82bc	89d36b45
3a609437	ec00c9a9	44715253	0a874b49	d773bc40	7c34671c	02717ef6	4feb5536
a2d02fff	d2bf60c4	d43f03c0	50b4ef6d	07478cd1	006e1888	a2e53f55	b9e6d4bc
a2048016	97573833	d7207d67	de0f8f3d	72f87b33	abcc4f33	7688c55d	7b00a6b0
947b0001	570075d2	f9bb88f8	8942019e	4264a5ff	856302e0	72dbd92b	ee971b69
6ea22fde	5f08ae2b	af7a616d	e5c98767	cflfebd2	61efc8c2	f1ac2571	cc8239c2
67214cb8	b1e583d1	b7dc3e62	7f10bdce	f90a5c38	0ff0443d	606e6dc6	60543a49
5727c148	2be98a1d	8ab41738	20e1be24	af96da0f	68458425	99833be5	600d457d
282f9350	8334b362	d91d1120	2b6d8da0	642b1e31	9c305a00	52bce688	1b03588a
f7baefd5	4142ed9c	a4315c11	83323ec5	dfef4636	a133c501	e9d3531c	ee353783

Таблица П20

9db30420	1fb6e9de	a7be7bef	d273a298	4a4f7bdb	64ad8c57	85510443	fa020ed1
7e287aff	e60fb663	095f35a1	79ebf120	fd059d43	6497b7b1	f3641f63	241e4adf
28147f5f	4fa2b8cd	c9430040	0cc32220	fd30b30	c0a5374f	1d2d00d9	24147b15
ee4d111a	0fca5167	71ff904c	2d195ffe	1a05645f	0c13fefc	081b08ca	05170121
80530100	e83e5efe	ac9af4f8	7fe72701	d2b8ee5f	06df4261	bb9e9b8a	7293ea25
ce84ffd9	f5718801	3dd64b04	a26f263b	7ed48400	547eebe6	446d4ca0	6cf3d6f5
2649abdf	aea0c7f5	36338cc1	503f7e93	d3772061	11b638e1	72500e03	f80eb2bb
abe0502e	ec8d77de	57971e81	e14f6746	c9335400	6920318f	081dbb99	ffc304a5
4d351805	7f3d5ce3	a6c866c6	5d5bcc9	daec6fea	9f926f91	9f46222f	3991467d
a5bf6d8e	1143c44f	43958302	d0214eeb	022083b8	3fb6180c	18f8931e	281658e6
26486e3e	8bd78a70	7477e4c1	b506e07c	f32d0a25	79098b02	e4eabb81	28123b23
69dead38	1574ca16	df871b62	211c40b7	a51a9ef9	0014377b	041e8ac8	09114003
bd59e4d2	e3d156d5	4fe876d5	2f91a340	557be8de	00eae4a7	0ce5c2ec	4db4bba6
e756bdff	dd3369ac	ec17b035	06572327	99afc8b0	56c8c391	6b65811c	5e146119
6e85cb75	be07c002	c2325577	893ff4ec	5bbfc92d	d0ec3b25	b7801ab7	8d6d3b24
20c763ef	c366a5fc	9c382880	0ace3205	aac9548a	eca1d7c7	041afa32	1d16625a
6701902c	9b757a54	31d477f7	9126b031	36cc6fdb	c70b8b46	d9e66a48	56e55a79
026a4ceb	52437eff	2f8f76b4	0df980a5	8674cde3	edda04eb	17a9be04	2c18f4df
b7747f9d	ab2af7b4	efc34d20	2e096b7c	1741a254	e5b6a035	213d42f6	2c1c7c26
61c2f50f	6552daf9	d2c231f8	25130f69	d8167fa2	0418f2c8	001a96a6	0d1526ab
63315c21	5e0a72ec	49bafefd	187908d9	8d0dbd86	311170a7	3e9b640c	cc3e10d7
d5cad3b6	0caec388	f73001e1	6c728aff	71eae2a1	1f9af36e	cfcbd12f	c1de8417
ac07be6b	cb44a1d8	8b9b0f56	013988c3	b1c52fca	b4be31cd	d8782806	12a3a4e2
6f7de532	58fd7eb6	d01ee900	24adffcc	f4990fc5	9711aac5	001d7b95	82e5e7d2
109873f6	00613096	c32d9521	ada121ff	29908415	7fbb977f	af9eb3db	29c9ed2a
5ce2a465	a730f32c	d0aa3fe8	8a5cc091	d49e2ce7	0ce454a9	d60acd86	015f1919
77079103	dea03af6	78a8565e	dee356df	21f05cbe	8b75e387	b3c50651	b8a5c3ef
d8eeb6d2	e523be77	c2154529	2f69efdf	afe67afb	f470c4b2	f3e0eb5b	d6cc9876
39e4460c	1fda8538	1987832f	ca007367	a99144f8	296b299e	492fc295	9266beab

## Таблицы замен

Таблица П20 (окончание)

b5676e69	9bd3ddda	df7e052f	db25701c	1b5e51ee	f65324e6	6afce36c	0316cc04
8644213e	b7dc59d0	7965291f	ccd6fd43	41823979	932bcd6	b657c34d	4edfd282
7ae5290c	3cb9536b	851e20fe	9833557e	13ecf0b0	d3ffb372	3f85c5c1	0aef7ed2

## П5. Алгоритм Crypton 0

Таблицы  $S_0$ ,  $S_1$ ,  $S_2$  и  $S_3$  алгоритма Crypton 0 (см. разд. 3.12) приведены в табл. П21–П24 соответственно.

Таблица П21

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

Таблица П22

8d	b3	65	aa	6f	3a	99	03	dc	f0	50	ff	4c	11	a6	46
ec	e1	36	bf	0b	a8	c3	5f	85	7a	96	f2	21	54	48	1d
b7	09	68	cc	e0	23	5c	42	9a	57	75	95	a9	fb	3e	86

Таблица П22 (окончание)

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

Таблица П23

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

## Таблицы замен

Таблица П24

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

## П6. Алгоритм DES

Таблицы замен  $S_1 \dots S_8$  алгоритма DES (см. разд. 3.15) приведены, соответственно, в табл. П25–П32.

Таблица П25

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Таблица П26

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Таблица П27

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Таблица П28

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Таблица П29

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Таблица П30

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

## Таблицы замен

Таблица П31

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Таблица П32

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

## П7. Алгоритм KASUMI

Таблицы  $S_7$  и  $S_9$  алгоритма KASUMI (см. разд. 3.27) могут быть реализованы как с помощью вычислений, так и непосредственно таблицами, хранимыми в энергонезависимой памяти шифрующего устройства [365]. При реализации алгоритма должен выбираться вариант использования таблиц в зависимости от ресурсов шифрующего устройства. Далее приведены оба варианта таблиц.

В описании формул замен применены следующие обозначения:

- $x_N$  — значение  $N$ -го бита входного значения;
- $y_N$  —  $N$ -й бит выходного значения;
- $x_1x_2$  — операция логического «и» между значениями  $x_1$  и  $x_2$ ;
- $\oplus$  — операция XOR.

Таблица  $S_7$  (вычисления):

$$\begin{aligned}y_0 = & x_1x_3 \oplus x_4 \oplus x_0x_1x_4 \oplus x_5 \oplus x_2x_5 \oplus x_3x_4x_5 \oplus x_6 \oplus x_0x_6 \oplus \\& \oplus x_1x_6 \oplus x_3x_6 \oplus x_2x_4x_6 \oplus x_1x_5x_6 \oplus x_4x_5x_6;\end{aligned}$$

$$\begin{aligned}y_1 = & x_0x_1 \oplus x_0x_4 \oplus x_2x_4 \oplus x_5 \oplus x_1x_2x_5 \oplus x_0x_3x_5 \oplus x_6 \oplus x_0x_2x_6 \oplus \\& \oplus x_3x_6 \oplus x_4x_5x_6 \oplus 1;\end{aligned}$$

$$\begin{aligned}y_2 = & x_0 \oplus x_0x_3 \oplus x_2x_3 \oplus x_1x_2x_4 \oplus x_0x_3x_4 \oplus x_1x_5 \oplus x_0x_2x_5 \oplus \\& \oplus x_0x_6 \oplus x_0x_1x_6 \oplus x_2x_6 \oplus x_4x_6 \oplus 1;\end{aligned}$$

$$y_3 = x_1 \oplus x_0x_1x_2 \oplus x_1x_4 \oplus x_3x_4 \oplus x_0x_5 \oplus x_0x_1x_5 \oplus x_2x_3x_5 \oplus \\ \oplus x_1x_4x_5 \oplus x_2x_6 \oplus x_1x_3x_6;$$

$$y_4 = x_0x_2 \oplus x_3 \oplus x_1x_3 \oplus x_1x_4 \oplus x_0x_1x_4 \oplus x_2x_3x_4 \oplus x_0x_5 \oplus \\ \oplus x_1x_3x_5 \oplus x_0x_4x_5 \oplus x_1x_6 \oplus x_3x_6 \oplus x_0x_3x_6 \oplus x_5x_6 \oplus 1;$$

$$y_5 = x_2 \oplus x_0x_2 \oplus x_0x_3 \oplus x_1x_2x_3 \oplus x_0x_2x_4 \oplus x_0x_5 \oplus x_2x_5 \oplus \\ \oplus x_4x_5 \oplus x_1x_6 \oplus x_1x_2x_6 \oplus x_0x_3x_6 \oplus x_3x_4x_6 \oplus x_2x_5x_6 \oplus 1;$$

$$y_6 = x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_4 \oplus x_1x_5 \oplus x_3x_5 \oplus x_6 \oplus x_0x_1x_6 \oplus \\ \oplus x_2x_3x_6 \oplus x_1x_4x_6 \oplus x_0x_5x_6.$$

Таблица  $S_9$  (вычисления):

$$y_0 = x_0x_2 \oplus x_3 \oplus x_2x_5 \oplus x_5x_6 \oplus x_0x_7 \oplus x_1x_7 \oplus x_2x_7 \oplus x_4x_8 \oplus \\ \oplus x_5x_8 \oplus x_7x_8 \oplus 1;$$

$$y_1 = x_1 \oplus x_0x_1 \oplus x_2x_3 \oplus x_0x_4 \oplus x_1x_4 \oplus x_0x_5 \oplus x_3x_5 \oplus x_6 \oplus x_1x_7 \oplus \\ \oplus x_2x_7 \oplus x_5x_8 \oplus 1;$$

$$y_2 = x_1 \oplus x_0x_3 \oplus x_3x_4 \oplus x_0x_5 \oplus x_2x_6 \oplus x_3x_6 \oplus x_5x_6 \oplus x_4x_7 \oplus \\ \oplus x_5x_7 \oplus x_6x_7 \oplus x_8 \oplus x_0x_8 \oplus 1;$$

$$y_3 = x_0 \oplus x_1x_2 \oplus x_0x_3 \oplus x_2x_4 \oplus x_5 \oplus x_0x_6 \oplus x_1x_6 \oplus x_4x_7 \oplus x_0x_8 \oplus \\ \oplus x_1x_8 \oplus x_7x_8;$$

$$y_4 = x_0x_1 \oplus x_1x_3 \oplus x_4 \oplus x_0x_5 \oplus x_3x_6 \oplus x_0x_7 \oplus x_6x_7 \oplus x_1x_8 \oplus x_2x_8 \oplus \\ \oplus x_3x_8;$$

$$y_5 = x_2 \oplus x_1x_4 \oplus x_4x_5 \oplus x_0x_6 \oplus x_1x_6 \oplus x_3x_7 \oplus x_4x_7 \oplus x_6x_7 \oplus x_5x_8 \oplus \\ \oplus x_6x_8 \oplus x_7x_8 \oplus 1;$$

$$y_6 = x_0 \oplus x_2x_3 \oplus x_1x_5 \oplus x_2x_5 \oplus x_4x_5 \oplus x_3x_6 \oplus x_4x_6 \oplus x_5x_6 \oplus x_7 \oplus \\ \oplus x_1x_8 \oplus x_3x_8 \oplus x_5x_8 \oplus x_7x_8;$$

$$y_7 = x_0x_1 \oplus x_0x_2 \oplus x_1x_2 \oplus x_3 \oplus x_0x_3 \oplus x_2x_3 \oplus x_4x_5 \oplus x_2x_6 \oplus x_3x_6 \oplus \\ \oplus x_2x_7 \oplus x_5x_7 \oplus x_8 \oplus 1;$$

$$y_8 = x_0x_1 \oplus x_2 \oplus x_1x_2 \oplus x_3x_4 \oplus x_1x_5 \oplus x_2x_5 \oplus x_1x_6 \oplus x_4x_6 \oplus x_7 \oplus \\ \oplus x_2x_8 \oplus x_3x_8.$$

Результирующие таблицы  $S_7$  и  $S_9$  приведены в табл. П33 и П34 соответственно.

## Таблицы замен

Таблица П33

54	50	62	56	22	34	94	96	38	6	63	93	2	18	123	33
55	113	39	114	21	67	65	12	47	73	46	27	25	111	124	81
53	9	121	79	52	60	58	48	101	127	40	120	104	70	71	43
20	122	72	61	23	109	13	100	77	1	16	7	82	10	105	98
117	116	76	11	89	106	0	125	118	99	86	69	30	57	126	87
112	51	17	5	95	14	90	84	91	8	35	103	32	97	28	66
102	31	26	45	75	4	85	92	37	74	80	49	68	29	115	44
64	107	108	24	110	83	36	78	42	19	15	41	88	119	59	3

Таблица П34

167	239	161	379	391	334	9	338	38	226	48	358	452	385	90	397
183	253	147	331	415	340	51	362	306	500	262	82	216	159	356	177
175	241	489	37	206	17	0	333	44	254	378	58	143	220	81	400
95	3	315	245	54	235	218	405	472	264	172	494	371	290	399	76
165	197	395	121	257	480	423	212	240	28	462	176	406	507	288	223
501	407	249	265	89	186	221	428	164	74	440	196	458	421	350	163
232	158	134	354	13	250	491	142	191	69	193	425	152	227	366	135
344	300	276	242	437	320	113	278	11	243	87	317	36	93	496	27
487	446	482	41	68	156	457	131	326	403	339	20	39	115	442	124
475	384	508	53	112	170	479	151	126	169	73	268	279	321	168	364
363	292	46	499	393	327	324	24	456	267	157	460	488	426	309	229
439	506	208	271	349	401	434	236	16	209	359	52	56	120	199	277
465	416	252	287	246	6	83	305	420	345	153	502	65	61	244	282
173	222	418	67	386	368	261	101	476	291	195	430	49	79	166	330
280	383	373	128	382	408	155	495	367	388	274	107	459	417	62	454
132	225	203	316	234	14	301	91	503	286	424	211	347	307	140	374
35	103	125	427	19	214	453	146	498	314	444	230	256	329	198	285
50	116	78	410	10	205	510	171	231	45	139	467	29	86	505	32
72	26	342	150	313	490	431	238	411	325	149	473	40	119	174	355

**Таблица П34 (окончание)**

185	233	389	71	448	273	372	55	110	178	322	12	469	392	369	190
1	109	375	137	181	88	75	308	260	484	98	272	370	275	412	111
336	318	4	504	492	259	304	77	337	435	21	357	303	332	483	18
47	85	25	497	474	289	100	269	296	478	270	106	31	104	433	84
414	486	394	96	99	154	511	148	413	361	409	255	162	215	302	201
266	351	343	144	441	365	108	298	251	34	182	509	138	210	335	133
311	352	328	141	396	346	123	319	450	281	429	228	443	481	92	404
485	422	248	297	23	213	130	466	22	217	283	70	294	360	419	127
312	377	7	468	194	2	117	295	463	258	224	447	247	187	80	398
284	353	105	390	299	471	470	184	57	200	348	63	204	188	33	451
97	30	310	219	94	160	129	493	64	179	263	102	189	207	114	402
438	477	387	122	192	42	381	5	145	118	180	449	293	323	136	380
43	66	60	455	341	445	202	432	8	237	15	376	436	464	59	461

## П8. Алгоритм MARS

Таблицы  $S_0$  и  $S_1$  алгоритма MARS (см. разд. 3.34) приведены в табл. П35 и П36.

**Таблица П35**

09d0c479	28c8ffe0	84aa6c39	9dad7287	7dff9be3	d4268361
c96da1d4	7974cc93	85d0582e	2a4b5705	1ca16a62	c3bd279d
0f1f25e5	5160372f	c695c1fb	4d7ff1e4	ae5f6bf4	0d72ee46
ff23de8a	b1cf8e83	f14902e2	3e981e42	8bf53eb6	7f4bf8ac
83631f83	25970205	76afe784	3a7931d4	4f846450	5c64c3f6
210a5f18	c6986a26	28f4e826	3a60a81c	d340a664	7ea820c4
526687c5	7eddd12b	32a11d1d	9c9ef086	80f6e831	ab6f04ad
56fb9b53	8b2e095c	b68556ae	d2250b0d	294a7721	e21fb253
ae136749	e82aae86	93365104	99404a66	78a784dc	b69ba84b
04046793	23db5c1e	46cae1d6	2fe28134	5a223942	1863cd5b

## Таблицы замен

Таблица П35 (продолжение)

c190c6e3	07dfb846	6eb88816	2d0dcc4a	a4ccae59	3798670d
cbfa9493	4f481d45	eafc5ca5	db1129d6	b0449e20	0f5407fb
6167d9a8	d1f45763	4daa96c3	3bec5958	ababa014	b6cccd201
38d6279f	02682215	8f376cd5	092c237e	bfc56593	32889d2c
854b3e95	05bb5b43	7dcfd5dcd	a02e926c	fae527e5	36a1c330
3412e1ae	f257f462	3c4f1d71	30a2e809	68e5f551	9c61ba44
5ded0ab8	75ce09c8	9654f93e	698c0cca	243cb3e4	2b062b97
0f3b8d9e	00e050df	fc5d6166	e35f9288	c079550d	0591aee8
8e531e74	75fe3578	2f6d829a	f60b21ae	95e8eb8d	6699486b
901d7d9b	fd6d6e31	1090acef	e0670dd8	dab2e692	cd6d4365
e5393514	3af345f0	6241fc4d	460da3a3	7bcf3729	5bf1d1e0
14aac070	1587ed55	3afd7d3e	d2f29e01	29a9d1f6	efb10c53
cf3b870f	b414935c	664465ed	024acac7	59a744c1	1d2936a7
dc580aa6	cf574ca8	040a7a10	6cd81807	8a98be4c	accea063
c33e92b5	d1e0e03d	b322517e	2092bd13	386b2c4a	52e8dd58
58656dfb	50820371	41811896	e337ef7e	d39fb119	c97f0df6
68fea01b	a150a6e5	55258962	eb6ff41b	d7c9cd7a	a619cd9e
bcf09576	2672c073	f003fb3c	4ab7a50b	1484126a	487ba9b1
a64fc9c6	f6957d49	38b06a75	dd805fcd	63d094cf	f51c999e
1aa4d343	b8495294	ce9f8e99	bffcd770	c7c275cc	378453a7
7b21be33	397f41bd	4e94d131	92cc1f98	5915ea51	99f861b7
c9980a88	1d74fd5f	b0a495f8	614deed0	b5778eea	5941792d
fa90c1f8	33f824b4	c4965372	3ff6d550	4ca5fec0	8630e964
5b3fbdbd6	7da26a48	b203231a	04297514	2d639306	2eb13149
16a45272	532459a0	8e5f4872	f966c7d9	07128dc0	0d44db62
afc8d52d	06316131	d838e7ce	1bc41d00	3a2e8c0f	ea83837e
b984737d	13ba4891	c4f8b949	a6d6acb3	a215cdce	8359838b
6bd1aa31	f579dd52	21b93f93	f5176781	187dfdde	e94aeb76
2b38fd54	431de1da	ab394825	9ad3048f	dfea32aa	659473e3

**Таблица П35 (окончание)**

623f7863	f3346c59	ab3ab685	3346a90b	6b56443e	c6de01f8
8d421fc0	9b0ed10c	88f1a1e9	54c1f029	7dead57b	8d7ba426
4cf5178a	551a7cca	1a9a5f08	fcd651b9	25605182	e11fc6c3
b6fd9676	337b3027	b7c8eb14	9e5fd030		

**Таблица П36**

6b57e354	ad913cf7	7e16688d	58872a69	2c2fc7df	e389cc66
30738df1	0824a734	e1797a8b	a4a8d57b	5b5d193b	c8a8309b
73f9a978	73398d32	0f59573e	e9df2b03	e8a5b6c8	848d0704
98df93c2	720a1dc3	684f259a	943ba848	a6370152	863b5ea3
d17b978b	6d9b58ef	0a700dd4	a73d36bf	8e6a0829	8695bc14
e35b3447	933ac568	8894b022	2f511c27	ddfbcc3c	006662b6
117c83fe	4e12b414	c2bca766	3a2fec10	f4562420	55792e2a
46f5d857	ceda25ce	c3601d3b	6c00ab46	efac9c28	b3c35047
611dfee3	257c3207	fdd58482	3b14d84f	23becb64	a075f3a3
088f8ead	07adf158	7796943c	facabf3d	c09730cd	f7679969
da44e9ed	2c854c12	35935fa3	2f057d9f	690624f8	1cb0baf9
7b0dbdc6	810f23bb	fa929a1a	6d969a17	6742979b	74ac7d05
010e65c4	86a3d963	f907b5a0	d0042bd3	158d7d03	287a8255
bba8366f	096edc33	21916a7b	77b56b86	951622f9	a6c5e650
8cea17d1	cd8c62bc	a3d63433	358a68fd	0f9b9d3c	d6aa295b
fe33384a	c000738e	cd67eb2f	e2eb6dc2	97338b02	06c9f246
419cf1ad	2b83c045	3723f18a	cb5b3089	160bead7	5d494656
35f8a74b	1e4e6c9e	000399bd	67466880	b4174831	acf423b2
ca815ab3	5a6395e7	302a67c5	8bdb446b	108f8fa4	10223eda
92b8b48b	7f38d0ee	ab2701d4	0262d415	af224a30	b3d88aba
f8b2c3af	daf7ef70	cc97d3b7	e9614b6c	2baebff4	70f687cf
386c9156	ce092ee5	01e87da6	6ce91e6a	bb7bcc84	c7922c20
9d3b71fd	060e41c6	d7590f15	4e03bb47	183c198e	63eeb240

## Таблицы замен

Таблица П36 (окончание)

2ddbf49a	6d5cba54	923750af	f9e14236	7838162b	59726c72
81b66760	bb2926c1	48a0ce0d	a6c0496d	ad43507b	718d496a
9df057af	44b1bde6	054356dc	de7ced35	d51a138b	62088cc9
35830311	c96efca2	686f86ec	8e77cb68	63e1d6b8	c80f9778
79c491fd	1b4c67f2	72698d7d	5e368c31	f7d95e2e	a1d3493f
dcd9433e	896f1552	4bc4ca7a	a6d1baf4	a5a96dcc	0bef8b46
a169fda7	74df40b7	4e208804	9a756607	038e87c8	20211e44
8b7ad4bf	c6403f35	1848e36d	80bdb038	1e62891c	643d2107
bf04d6f8	21092c8c	f644f389	0778404e	7b78adb8	a2c52d53
42157abe	a2253e2e	7bf3f4ae	80f594f9	953194e7	77eb92ed
b3816930	da8d9336	bf 447469	f26d9483	ee6faed5	71371235
de425f73	b4e59f43	7dbe2d4e	2d37b185	49dc9a63	98c39d98
1301c9a2	389b1bbf	0c18588d	a421c1ba	7aa3865c	71e08558
3c5cfcaa	7d239ca4	0297d9dd	d7dc2830	4b37802b	7428ab54
aaaa0347	4b3fbb85	692f2f08	134e578e	36d9e0bf	ae8b5fcf
edb93ecf	2b27248e	170eb1ef	7dc57fd6	1e760f16	b1136601
864e1b9b	d7ea7319	3ab871bd	cfa4d76f	e31bd782	0dbeb469
abb96061	5370f85d	ffb07e37	da50d0fb	ebc977b6	0b98b40f
3a4d0fe6	df4fc26b	159cf22a	c298d6e2	2b78ef6a	61a94ac0
ab561187	14eea0f0	df0d4164	19af70ee		

**П9. Алгоритм s<sup>2</sup>DES**

Таблицы замен  $S_1 \dots S_8$  алгоритма s<sup>2</sup>DES (см. разд. 3.15) приведены, соответственно, в табл. П37–П44.

Таблица П37

12	14	1	15	11	10	8	4	7	9	5	0	3	2	13	6
3	5	4	12	9	14	0	8	2	7	10	1	13	6	15	11
10	7	9	11	15	13	2	5	14	6	1	4	12	3	8	0
6	0	5	8	14	7	1	3	12	4	2	13	11	15	10	9

Таблица П38

2	12	4	6	3	0	8	5	10	11	15	7	13	1	14	9
14	8	3	11	9	13	10	2	5	0	1	6	7	12	15	4
4	13	14	3	1	10	5	7	9	15	8	12	11	2	0	6
0	11	1	15	4	5	12	14	7	10	9	13	6	8	2	3

Таблица П39

5	10	7	12	13	2	0	4	6	14	11	15	3	1	9	8
3	13	6	14	2	0	15	12	1	5	10	7	4	11	8	9
9	2	11	6	1	13	10	15	14	12	3	5	0	8	4	7
1	8	14	11	5	15	9	3	7	6	4	0	12	10	13	2

Таблица П40

13	2	12	11	3	1	5	9	15	6	8	0	14	10	4	7
9	1	14	5	11	7	12	4	8	15	0	6	3	2	13	10
14	5	15	13	7	2	9	6	0	12	10	11	4	8	1	3
6	10	5	3	2	11	14	0	7	4	1	8	9	13	15	12

Таблица П41

10	2	9	11	8	7	6	3	5	13	12	15	0	4	14	1
2	1	0	5	11	8	15	7	12	9	14	6	3	13	10	4
0	5	8	6	4	3	13	14	9	1	15	11	2	10	7	12
7	14	11	13	12	2	10	9	1	8	0	4	5	6	3	15

Таблица П42

0	11	7	10	12	9	14	6	1	3	5	15	2	4	8	13
3	1	4	5	0	12	8	7	10	2	14	13	6	9	15	11
5	12	15	6	7	11	10	13	0	8	9	14	4	2	1	3
4	8	10	11	6	5	7	1	14	15	12	2	3	13	9	0

**Таблицы замен****Таблица П43**

5	0	11	14	10	2	9	8	13	3	12	6	4	7	1	15
10	12	13	4	9	1	3	0	6	8	5	15	14	11	2	7
6	11	12	9	0	3	4	14	1	7	8	13	10	2	15	5
9	4	7	0	3	11	2	1	15	5	6	8	12	13	10	14

**Таблица П44**

7	13	4	14	10	15	8	2	11	9	6	3	1	12	5	0
6	14	10	12	5	7	0	1	2	13	11	4	8	9	15	3
10	6	12	1	11	9	14	3	13	15	4	5	0	2	8	7
5	3	15	6	0	1	13	9	4	10	14	8	12	7	11	2

**П10. Алгоритм s<sup>3</sup>DES**

Таблицы замен  $S_1 \dots S_8$  алгоритма s<sup>3</sup>DES (см. разд. 3.15) приведены, соответственно, в табл. П45–П52.

**Таблица П45**

15	8	3	14	4	2	9	5	0	11	10	1	13	7	6	12
6	15	9	5	3	12	10	0	13	8	4	11	14	2	1	7
9	14	5	8	2	4	15	3	10	7	6	13	1	11	12	0
10	5	3	15	12	9	0	6	1	2	8	4	11	14	7	13

**Таблица П46**

13	14	0	3	10	4	7	9	11	8	12	6	1	15	2	5
8	2	11	13	4	1	14	7	5	15	0	3	10	6	9	12
14	9	3	10	0	7	13	4	8	5	6	15	11	12	1	2
1	4	14	7	11	13	8	2	6	3	5	10	12	0	15	9

Таблица П47

13	3	11	5	14	8	0	6	4	15	1	12	7	2	10	9
4	13	1	8	7	2	14	11	15	10	12	3	9	5	0	6
6	5	8	11	13	14	3	0	9	2	4	1	10	7	15	12
1	11	7	2	8	13	4	14	6	12	10	15	3	0	9	5

Таблица П48

9	0	7	11	12	5	10	6	15	3	1	14	2	8	4	13
5	10	12	6	0	15	3	9	8	13	11	1	7	2	14	4
10	7	9	12	5	0	6	11	3	14	4	2	8	13	15	1
3	9	15	0	6	10	5	12	14	2	1	7	13	4	8	11

Таблица П49

5	15	9	10	0	3	14	4	2	12	7	1	13	6	8	11
6	9	3	15	5	12	0	10	8	7	13	4	2	11	14	1
15	0	10	9	3	5	4	14	8	11	1	7	6	12	13	2
12	5	0	6	15	10	9	3	7	2	14	11	8	1	4	13

Таблица П50

4	3	7	10	9	0	14	13	15	5	12	6	2	11	1	8
14	13	11	4	2	7	1	8	9	10	5	3	15	0	12	6
13	0	10	9	4	3	7	14	1	15	6	12	8	5	11	2
1	7	4	14	11	8	13	2	10	12	3	5	6	15	0	9

Таблица П51

4	10	15	12	2	9	1	6	11	5	0	3	7	14	13	8
10	15	6	0	5	3	12	9	1	8	11	13	14	4	7	2
2	12	9	6	15	10	4	1	5	11	3	0	8	7	14	13
12	6	3	9	0	5	10	15	2	13	4	14	7	11	1	8

**Таблицы замен****Таблица П52**

13	10	0	7	3	9	14	4	2	15	12	1	5	6	11	8
2	7	13	1	4	14	11	8	15	12	6	10	9	5	0	3
4	13	14	0	9	3	7	10	1	8	2	11	15	5	12	6
8	11	7	14	2	4	13	1	6	5	9	0	12	15	3	10

**П11. Алгоритм s<sup>5</sup>DES**

Таблицы замен  $S_1\dots S_8$  алгоритма s<sup>5</sup>DES (см. разд. 3.15) приведены, соответственно, в табл. П53–П60.

**Таблица П53**

9	10	15	1	4	7	2	12	6	5	3	14	8	11	13	0
2	13	8	4	11	1	14	7	12	3	15	9	5	6	0	10
10	12	4	7	9	2	15	1	3	6	13	8	14	5	0	11
4	11	1	13	14	7	8	2	10	0	6	3	9	12	15	5

**Таблица П54**

6	3	5	0	8	14	11	13	9	10	12	7	15	4	2	1
9	6	10	12	15	0	5	3	4	1	7	11	2	13	14	8
5	8	3	14	6	13	0	11	10	15	9	2	12	1	7	4
6	3	15	9	0	10	12	5	13	8	2	4	11	7	1	14

**Таблица П55**

11	5	8	2	6	12	1	15	7	14	13	4	0	9	10	3
7	8	1	14	11	2	13	4	12	3	6	9	5	15	0	10
8	11	1	12	15	6	2	5	4	7	10	9	3	0	13	14
13	2	4	7	1	11	14	8	10	9	15	0	12	6	3	5

Таблица П55

13	11	8	14	3	0	6	5	4	7	2	9	15	12	1	10
10	0	3	5	15	6	12	9	1	13	4	14	8	11	2	7
6	5	11	8	0	14	13	3	9	12	7	2	10	1	4	15
9	12	5	15	6	3	0	10	7	11	2	8	13	4	14	1

Таблица П55

12	6	2	11	5	8	15	1	3	13	9	14	0	7	10	4
15	0	12	5	3	6	9	10	4	11	2	8	14	1	7	13
1	12	15	5	6	11	8	2	4	7	10	9	13	0	3	14
6	3	10	0	9	12	5	15	13	4	1	14	7	11	8	2

Таблица П58

14	8	2	5	9	15	4	3	7	1	12	6	0	10	11	13
1	13	11	8	2	4	7	14	10	6	0	15	5	9	12	3
4	2	9	15	14	8	3	5	10	7	0	12	13	1	6	11
8	11	7	4	13	1	14	2	5	0	9	10	6	15	3	12

Таблица П59

4	13	10	3	7	0	9	14	2	1	15	6	12	11	5	8
9	0	15	10	12	6	5	3	14	7	1	13	11	8	2	4
13	10	3	9	0	7	14	4	8	6	5	12	11	1	2	15
10	3	12	6	5	9	0	15	4	8	11	1	14	7	13	2

Таблица П60

1	10	2	12	15	9	4	7	14	3	5	0	8	6	11	13
14	13	7	11	2	4	1	8	0	10	9	6	5	15	12	3
10	15	12	1	9	2	7	4	13	0	6	11	3	5	8	14
4	8	1	2	7	11	13	14	10	5	15	12	0	6	3	9

# Литература

1. Беляев А. В. Методы и средства защиты информации (курс лекций) // <http://wall.tms.ru>.
2. Беляев А. В., Панасенко С. П., Петренко С. А. Перспективы прикладной криптографии // Защита информации. Конфидент. 2001. № 6. с. 70–78.
3. Брассар Ж. Современная криптология. — Пер. с англ.: М.: Полимед, 1999 — 176 с.
4. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. — М.: Госстандарт СССР, 1989.
5. Киви Б. Конкурс на новый криптостандарт AES // <http://byrd.narod.ru>.
6. Панасенко С. П. Атаки на алгоритмы шифрования // BYTE. 2004. № 11. с. 77–79.
7. Панасенко С. П. Конкурсы AES и NESSIE // Мир ПК. 2004. № 12. с. 88–92.
8. Панасенко С. П. Конкурс NESSIE и алгоритм MISTY1 // Банки и технологии. 2004. № 5. с. 76–79.
9. Панасенко С. П. Назначение и структура алгоритмов шифрования // iXBT. — <http://www.ixbt.com> — 2006.
10. Панасенко С. П. «Неудачники» конкурса AES: алгоритм шифрования E2 // Банки и технологии. 2003. № 6. с. 82–85.
11. Панасенко С. П. Современные алгоритмы шифрования // BYTE. 2003. № 8. с. 18–22.
12. Панасенко С. П. Стандарт шифрования США // Мир и безопасность. 2003. № 6. с. 29–31.
13. Панасенко С. П. NESSIE — конкурс криптоалгоритмов // Управление безопасностью. 2004. № 3. с. 23–25.
14. Панасенко С. П., Батура В. П. Основы криптографии для экономистов: учебное пособие. Под ред. Л. Г. Гагариной. — М.: Финансы и статистика, 2005 — 176 с.

15. Панасенко С., Удовицкий А. Алгоритм шифрования SAFER+ // Банки и технологии. 2004. № 2. с. 60–64.
16. Петров А. А. Компьютерная безопасность: криптографические методы защиты. — М.: ДМК, 2000. 448 с.
17. Положение ФАПСИ «Система сертификации средств криптографической защиты информации». Октябрь 1993 // <http://www.ancud.ru>.
18. Приказ ФАПСИ от 23.09.1999 г. № 158. «Об утверждении положения о порядке разработки, производства, реализации и использования средств криптографической защиты информации с ограниченным доступом, не содержащей сведений, составляющих государственную тайну» // Российская газета. 2000. № 18.
19. Применко Э. А., Винокуров А. А. Сравнение российского стандарта шифрования алгоритма ГОСТ 28147-89 и алгоритма Rijndael, выбранного в качестве нового стандарта шифрования США // Системы безопасности. 2001. № 1. с. 79—82.
20. Религии мира. Справочник. — Пер. с англ.: М.: Белфаксиздатгрупп, 1994.
21. Романец Ю. В., Тимофеев П. А., Шаньгин В. Ф. Защита информации в компьютерных системах и сетях. 2-е издание — М.: Радио и связь, 2001 — 376 с.
22. Ростовцев А. Г., Маховенко Е. Б. Два подхода к анализу блочных шифров // <http://www.ssl.stu.neva.ru> — Проблемы информационной безопасности. Компьютерные системы. 2002. № 1.
23. Ростовцев А. Г., Маховенко Е. Б., Филиппов А. С., Чечулин А. А. О стойкости ГОСТ 28147-89 // <http://www.ssl.stu.neva.ru>. — СПбГПУ.
24. Соколов А. В., Шаньгин В. Ф. Защита информации в распределенных корпоративных сетях и системах. — М.: ДМК Пресс, 2002. 656 с.
25. Фейстель Х. Криптография и компьютерная безопасность. — Пер. с англ.: А. Винокуров — <http://avin.chat.ru> (Перевод по изданию: Feistel H. Cryptography and Computer Privacy, Scientific American, May 1973, Vol. 228, No. 5, pp. 15–23).
26. Шенон К. Теория связи в секретных системах. — Пер. с англ.: В. Ф. Писаренко. — <http://info.phys.msu.su>. — 1949.
27. Шнайер Б. Курс самоподготовки по криптоанализу блочных шифров. — Пер. с англ: Быбин С. С. — <http://bybin.narod.ru>.
28. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — Пер. с англ.: М.: Издательство ТРИУМФ, 2002. 816 с.
29. About 3GPP // <http://www.3gpp.org>.

Литература

30. Adams C. RFC 2144: The CAST-128 Encryption Algorithm // Entrust Technologies, May 1997 — <http://www.ietf.org>.
31. Adams C., Gilchrist J. RFC 2612: The CAST-256 Encryption Algorithm // Entrust Technologies, June 1999 — <http://www.ietf.org>.
32. Advanced Encryption Standard (AES). Questions and Answers // <http://csrc.nist.gov> — January 28, 2002.
33. AES Round 1 Information // <http://csrc.nist.gov> — January 26, 2001.
34. Anderson R., Biham E. Two Practical and Provably Secure Block Ciphers: BEAR and LION // <http://citeseer.ist.psu.edu> — 1995.
35. Anderson R., Biham E., Knudsen L. Serpent: A Proposal for the Advanced Encryption Standard // <http://csrc.nist.gov>.
36. Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES) // <http://csrc.nist.gov> — Department of Commerce — National Institute of Standards and Technology — Federal Register: September 12, 1997.
37. Aoki K., Ichikawa T., Kanda M., Matsui M., Moriai S., Nakajima J., Tokita T. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms // <http://www.mirrors.wiretapped.net> — July 13, 2000.
38. Aoki K., Ichikawa T., Kanda M., Matsui M., Moriai S., Nakajima J., Tokita T. Specification of Camellia — a 128-bit Block Cipher // <http://www.cosic.esat.kuleuven.be> — March 10, 2000.
39. ARIA. Korean Standard Block Cipher Algorithm // <http://www.nrsi.re.kr> (ARIA Home Page).
40. Baldwin R., Rivest R. RFC 2040: The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms // <http://www.ietf.org> — October 1996.
41. Bar-El H., Choukri H., Naccache D., Tunstall M., Whelan C. The Sorcerer's Apprentice Guide to Fault Attacks // <http://citeseer.ist.psu.edu>.
42. Barkan E., Biham E., Keller N. Instant Ciphertxt-Only Cryptanalysis of GSM Encrypted Communication // <http://www.cs.technion.ac.il> — 2006.
43. Barreto P. S. L. M., Rijmen V. The Anubis Block Cipher // <http://www.cosic.esat.kuleuven.be>.
44. Barreto P. S. L. M., Rijmen V. The Khazad Legacy-Level Block Cipher // <http://www.cosic.esat.kuleuven.be><sup>1</sup>.

---

<sup>1</sup> Существуют два варианта данного документа, не отличающиеся какими-либо атрибутами. Один из вариантов описывает исходную версию алгоритма Khazad, второй — модифицированную.

45. Barreto P. S. L. M., Rijmen V., Nakahara J. Jr., Preneel B., Vandewalle J., Kim H. Y. Improved Square Attacks Against Reduced-Round Hierocrypt // <http://citeseer.ist.psu.edu>.
46. Baudron O., Gilbert H., Granboulan L., Handschuh H., Harley R., Joux A., Nguyen P., Noilhan F., Pointcheval D., Pornin T., Poupart G., Stern J., Vaudenay S. DFC Update // <http://citeseer.ist.psu.edu>.
47. Baudron O., Gilbert H., Granboulan L., Handschuh H., Joux A., Nguyen P., Noilhan F., Pointcheval D., Pornin T., Poupart G., Stern J., Vaudenay S. Report on the AES Candidates // <http://csrc.nist.gov>.
48. Ben-Aroya I., Biham E. Differential Cryptanalysis of Lucifer // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — 1993.
49. Bernstein D. J. Cache-timing attacks on AES // <http://cr.yp.to> — April 04, 2005 — The University of Illinois at Chicago.
50. Biham E. A Note on Comparing the AES Candidates // <http://csrc.nist.gov> — Technion, Haifa, Israel.
51. Biham E. Comment on Selecting the Cipher for the AES Second Round // <http://csrc.nist.gov> — Technion — Israel Institute of Technology, Haifa, Israel.
52. Biham E. Cryptanalysis of Ladder-DES // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — 1997.
53. Biham E. Cryptanalysis of Multiple Modes of Operation // <http://bybin.narod.ru> — Technion, Haifa, Israel — 1994.
54. Biham E. Cryptanalysis of Triple-Modes of Operation // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — 1996.
55. Biham E. How to Forge DES-Encrypted Messages in  $2^{28}$  Steps // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — 1996.
56. Biham E. New Types of Cryptanalytic Attacks Using Related Keys (Revised Version) // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — 1992.
57. Biham E. On Matsui's Linear Cryptanalysis // <http://bybin.narod.ru> — Technion, Haifa, Israel — 1994.
58. Biham E. On Modes of Operation (Abstract) // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — February 22, 1994.
59. Biham E., Biryukov A. An Improvement of Davies' Attack on DES // <http://citeseer.ist.psu.edu> — Technion, Haifa, Israel — 1994.
60. Biham E., Biryukov A. How to Strengthen DES Using Existing Hardware // <http://download.planetmirror.com> — Technion, Haifa, Israel.

Литература

61. Biham E., Biryukov A., Dunkelman O., Richardson E., Shamir A. Initial Observations on Skipjack: Cryptanalysis of Skipjack-3XOR // <http://www.cs.technion.ac.il> — Technion — 1998.
62. Biham E., Biryukov A., Ferguson N., Knudsen L. R., Schneier B., Shamir A. Cryptanalysis of Magenta // <http://www.macfergus.com>.
63. Biham E., Biryukov A., Shamir A. Cryptanalysis of Skipjack Reduced to 31 Rounds using Impossible Differentials // <http://www.cs.technion.ac.il> — Technion — 1998.
64. Biham E., Biryukov A., Shamir A. Miss in the Middle Attacks on IDEA, Khufu and Khafre // <http://www.wizdom.weizmann.ac.il>.
65. Biham E., Dunkelman O., Furman V., Mor T. Preliminary Report on the NESSIE Submissions: Anubis, Camellia, Khazad, IDEA, Misty1, NIMBUS, and Q // <http://www.cosic.esat.kuleuven.be> — Technion, Haifa, Israel.
66. Biham E., Dunkelman O., Keller N. A Related-Key Rectangle Attack on the Full KASUMI // <http://www.cs.technion.ac.il> — 2005.
67. Biham E., Dunkelman O., Keller N. A Simple Related-Key Attack on the Full SHACAL-1 // <http://www.cosic.esat.kuleuven.be>.
68. Biham E., Dunkelman O., Keller N. Differential-Linear Cryptanalysis of Serpent // <http://citeseer.ist.psu.edu> — Technion, Haifa, Israel.
69. Biham E., Dunkelman O., Keller N. Rectangle Attacks on 49-Round SHACAL-1 // <http://vipe.technion.ac.il> — Technion, Haifa, Israel.
70. Biham E., Dunkelman O., Keller N. Related-Key Boomerang and Rectangle Attacks // <http://vipe.technion.ac.il> — 2005.
71. Biham E., Furman V. Differential Cryptanalysis of Nimbus // <http://www.cosic.esat.kuleuven.be> — November 29, 2000.
72. Biham E., Furman V., Misztal M., Rijmen V. Differential Cryptanalysis of Q // <http://citeseer.ist.psu.edu> — February 11, 2001.
73. Biham E., Keller N. Cryptanalysis of Reduced Variants of Rijndael // <http://csrc.nist.gov> — Technion — Israel Institute of Technology, Haifa, Israel.
74. Biham E., Knudsen L. Cryptanalysis of the ANSI X9.52 CBCM Mode // <http://www.cs.technion.ac.il> — Technion, Haifa, Israel — 1998.
75. Biham E., Shamir A. Differential Cryptanalysis of DES-like Cryptosystems // <http://citeseer.ist.psu.edu> — The Weizmann Institute of Science, Israel — July 19, 1990.

76. Biham E., Shamir A. Differential Cryptanalysis of Feal and N-Hash // <http://cs.technion.ac.il> — The Weizmann Institute of Science, Rehovot, Israel.
77. Biham E., Shamir A. Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer // <http://www.cs.technion.ac.il> — The Weizmann Institute of Science, Rehovot, Israel.
78. Biham E., Shamir A. Differential Cryptanalysis of the full 16-round DES // <http://citeseer.ist.psu.edu> — Technion, Haifa, Israel — 1991.
79. Biham E., Shamir A. Differential Fault Analysis of Secret Key Cryptosystems // <http://citeseer.ist.psu.edu> — Technion — Israel Institute of Technology, Haifa, Israel — 1997.
80. Biham E., Shamir A. Power Analysis of the Key Scheduling of the AES Candidates // <http://www.nist.gov>.
81. Biham E., Shamir A. Research announcement: A new Cryptanalytic attack on DES // <http://jya.com> — Draft — October 18, 1996.
82. Biryukov A. Analysis of Involutional Ciphers: Khazad and Anubis // <http://citeseer.ist.psu.edu> — 2003 — Katholieke Universiteit Leuven, Belgium.
83. Biryukov A. Block Ciphers and Stream Ciphers: The State of the Art // <http://homes.esat.kuleuven.be> — Katholieke Universiteit Leuven, Belgium.
84. Biryukov A., The Boomerang Attack on 5 and 6-round Reduced AES // <http://www.cosic.esat.kuleuven.be> — 2004 — Katholieke Universiteit Leuven, Belgium.
85. Biryukov A., De Canniere C. Block Ciphers and Systems of Quadratic Equations // <http://citeseer.ist.psu.edu> — Katholieke Universiteit Leuven, Belgium.
86. Biryukov A., De Canniere C. Linear Cryptanalysis // <http://homes.esat.kuleuven.be>.
87. Biryukov A., De Canniere C., Dellkrantz G. Cryptanalysis of SAFER++ // <http://homes.esat.kuleuven.be> — 2003 — K. U. Leuven, Belgium.
88. Biryukov A., De Canniere C., Lano J., Ors S. B., Preneel B. Security and Performance Analysis of ARIA // <http://www.nrsi.re.kr> — Version 1.2 — Katholieke Universiteit Leuven, Belgium — January 7, 2003.
89. Biryukov A., Kushilevitz E. From Differential Cryptanalysis to Ciphertext-Only Attacks // <http://citeseer.ist.psu.edu> — Technion — Israel Institute of Technology, Haifa, Israel.

90. Biryukov A., Kushilevitz E. Improved Cryptanalysis of RC5 // <http://citeseer.ist.psu.edu> — Technion — Israel Institute of Technology, Haifa, Israel.
91. Biryukov A., Wagner D. Advanced Slide Attacks // <http://citeseer.ist.psu.edu>.
92. Biryukov A., Wagner D. Slide Attacks // <http://citeseer.ist.psu.edu>.
93. Blaze M. A Cryptographic File System for Unix // <http://citeseer.ist.psu.edu> — AT&T Bell Laboratories, Holmdel, NJ, USA.
94. Blaze M. High-Bandwidth Encryption with Low-Bandwidth Smartcards // <http://citeseer.ist.psu.edu> — AT&T Bell Laboratories, Murray Hill, NJ — December 3, 1995.
95. Blaze M., Diffie W., Rivest R. L., Schneier B., Shimomura T., Thompson E., Wiener M. Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security // <http://www.schneier.com> — January 1996.
96. Blaze M., Schneier B. The MacGuffin Block Cipher Algorithm // <http://www.schneier.com>.
97. Blomer J., Seifert J.-P. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES) // <http://wwwcs.uni-paderborn.de> — 2003.
98. Blunden M., Escott A. Related Key Attacks on Reduced Round KASUMI // <http://www.kryptosec.co.uk>.
99. Den Boer B. Cryptanalysis of F.E.A.L // <http://bybin.narod.ru>.
100. Boneh D., DeMillo R. A., Lipton R. J. On the Importance of Checking Cryptographic Protocols for Faults // <http://citeseer.ist.psu.edu> — Bellcore, Morristown, NJ.
101. Borisov N., Chew M., Johnson R., Wagner D. Multiplicative Differentials // <http://citeseer.ifi.unizh.ch> — University of California at Berkeley.
102. Borst J. The Block Cipher: Grand Cru // <http://www.cosic.esat.kuleuven.be> — K. U. Leuven, Heverlee, Belgium.
103. Borst J. Weak Keys of Crypton // <http://csrc.nist.gov> — K. U. Leuven, Heverlee, Belgium.
104. Braziler Y. The Statistical Evaluation of the NESSIE Submission Hierocrypt-11 // <http://www.cosic.esat.kuleuven.be> — Technion, Israel — November 28, 2001.
105. Brickell E., Denning D., Kent S., Maher D., Tuchman W. SKIPJACK Review. Interim Report // <http://www.cs.georgetown.edu> — July 28, 1993.
106. Broersma M. New Xbox security cracked by Linux fans // <http://news.zdnet.co.uk> — ZDNet UK — October 14, 2002.

107. Brown L. The LOKI91 Block Cipher // <http://www.unsw.adfa.edu.au> — April 30, 1999.
108. Brown L., Kwan M., Pieprzyk J., Seberry J. Improving Resistance to Differential Cryptanalysis and the Redesign of LOKI // <http://www.unsw.adfa.edu.au> — Australian Defence Force Academy, Canberra, Australia.
109. Brown L., Pieprzyk J. Introducing the new LOKI97 Block Cipher // <http://www.unsw.adfa.edu.au> — June 12, 1998.
110. Brown L., Pieprzyk J., Seberry J. LOKI — A Cryptographic Primitive for Authentication and Secrecy Applications // <http://www.unsw.adfa.edu.au> — Australian Defence Force Academy, Canberra, Australia.
111. Burwick C., Coppersmith D., D'Avignon E., Gennaro R., Halevi S., Jutla C., Matyas S. M. Jr., O'Connor L., Peyravian M., Safford D., Zunic N. MARS — a candidate cipher for AES // <http://www.ibm.com> — IBM Corporation — Revised, September, 22 1999.
112. Campbell K., Wiener M. DES is not a Group // <http://www3.sympatico.ca> — Bell-Northern Research, Ottawa, Canada.
113. Carter G., Dawson E., Nielsen L. Key Schedule Classification of the AES Candidates // <http://www.nist.gov>.
114. Chari S., Jutla C., Rao J., Rohatgi P. A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards // <http://www.nist.gov>.
115. Charnes C., O'Connor L., Pieprzyk J., Safavi-Naini R., Zheng Y. Further Comments on the Soviet Encryption Algorithm // <http://citeseer.ist.psu.edu> — University of Wollongong, NSW, Australia — June 1, 1994.
116. Cheon J. H., Kim M. J. Kim K. Impossible Differential Cryptanalysis of Hie-rocrypt-3 Reduced to 3 Rounds // <http://www.math.snu.ac.kr>.
117. Cheon J. H., Kim M., Kim K., Lee J.-Y., Kang S. Improved Impossible Differential Cryptanalysis of Rijndael and Crypton // <http://citeseer.ist.psu.edu> — 2001.
118. Cid C., Murphy S., Robshaw M. Computational and Algebraic Aspects of the Advanced Encryption Standard // <http://www.isg.rhul.ac.uk> — University of London, Egham, U. K. — 2004.
119. Ciet M., Piret G., Quisquater J.-J. Related-Key and Slide Attacks: Analysis, Connections, and Improvements (Extended Abstract) // <http://citeseer.ist.psu.edu> — 2002 — Universite catholique de Louvain, Louvain-la-Neuve, Belgium.
120. Coppersmith D., Johnson D., Matyas S. A proposed mode for triple-DES encryption // <http://www.research.ibm.com> — IBM — 1996.

121. Courtois N. T. Is AES a Secure Cipher? // <http://www.cryptosystem.net>.
122. Courtois N. The Best Differential Characteristics and Subtleties of the Biham-Shamir Attacks on DES // <http://eprint.iacr.org> — Axalto Cryptographic Research & Advanced Security, Cedex, France — September 2004.
123. Courtois N., Castagnos G., Goubin L. What do DES S-boxes Say to Each Other? // <http://eprint.iacr.org> — Axalto Cryptographic Research & Advanced Security, Cedex, France.
124. Courtois N. T., Goubin L. An Algebraic Masking Method to Protect AES Against Power Attacks // <http://eprint.iacr.org>.
125. Crowley P. Mercy: a fast large block cipher for disk sector encryption // <http://citeseer.ist.psu.edu> — DataCash Ltd.
126. CRYPTREC Report 2000 // <http://www.ipa.go.jp> — March 2001 — Information Technology Promotion Agency, Japan.
127. CRYPTREC Report 2002 // <http://www.ipa.go.jp> — March 2003 — Information-technology Promotion Agency, Japan.
128. Daemen J. Cipher and Hash Function Design Strategies Based on Linear and Differential Cryptanalysis // <http://homes.esat.kuleuven.be> — 1995.
129. Daemen J., Govaerts R., Vandewalle J. Weak Keys for IDEA // <http://bybin.narod.ru> — 1993.
130. Daemen J., Knudsen L., Rijmen V. The Block Cipher Square // <http://www.esat.kuleuven.ac.be>.
131. Daemen J., Peeters M., Van Assche G., Rijmen V. Nessie Proposal: Noekeon // <http://www.cosic.esat.kuleuven.be> — 2000.
132. Daemen J., Rijmen V. AES Proposal: Rijndael // <http://csrc.nist.gov> — Document version 2 — 03/09/99.
133. Daemen J., Rijmen V. Answer to "new observations on Rijndael" // <http://www.esat.kuleuven.ac.be> — August 11, 2000.
134. Daemen J., Rijmen V. Rijndael for AES // <http://citeseer.ist.psu.edu> — March 24, 2000.
135. Damgaard I. B., Knudsen L. R. Multiple Encryption with Minimum Key // <http://citeseer.ist.psu.edu> — Aarhus University, Denmark.
136. Dhem J.-F., Koeune F., Leroux P.-A., Mestre P., Quisquater J.-J., Willems J.-L. A practical implementation of the timing attack // <http://citeseer.ist.psu.edu> — June 15, 1998 — Universite catholique de Louvain, Louvain-la-Neuve, Belgium.

137. Dottax E. Results on the implementations of Khazad, MISTY1 and SAFER++ on a 8051 cpu // <http://www.cosic.esat.kuleuven.be> — Ecole Normale Supérieure — September 17, 2002.
138. Dunkelman O. Comparing MISTY1 and KASUMI // <http://citeseer.ist.psu.edu> — December 31, 2002.
139. Dunkelman O., Keller N. Boomerang and Rectangle Attacks on SC2000 // <http://citeseer.ist.psu.edu> — July 3, 2001.
140. Dunkelman O., Keller N., Kim J. Related-Key Rectangle Attack on the Full SHACAL-1 // <http://www.cosic.esat.kuleuven.be>.
141. Dusart P., Letourneux G., Vivolo O. Differential Fault Analysis on A.E.S // <http://citeseer.ist.psu.edu> — October 01, 2002.
142. Dusse S., Hoffman P., Ramsdell B., Lundblade L., Repka L. RFC 2311: S/MIME Version 2 Message Specification // <http://www.ietf.org> — March 1998.
143. Ehrsam W. F., Meyer C. H. W., Powers R. L., Smith J. L., Tuchman W. L. Product Block Cipher System for Data Security, United States Patent № 3962539, June 8, 1976 // <http://www.freepatentonline.com>.
144. Feistel H. Block Cipher Cryptographic System, United States Patent № 3798359, March 19, 1974 // <http://www.freepatentonline.com>.
145. Feistel H. Centralized Verification System, United States Patent № 3798605, March 19, 1974 // <http://www.freepatentonline.com>.
146. Feistel H. Step Code Ciphering System, United States Patent № 3798360, March 19, 1974 // <http://www.freepatentonline.com>.
147. Ferguson N., Kelsey J., Lucks S., Schneier B., Stay M., Wagner D., Whiting D. Improved Cryptanalysis of Rijndael // <http://www.schneier.com>.
148. Ferguson N., Schneier B. Cryptanalysis of Akelarre // <http://www.schneier.com> — July 23, 1997.
149. Ferguson N., Schroepel R., Whiting D. A simple algebraic representation of Rijndael // <http://citeseer.ist.psu.edu> — Draft May 05, 2001.
150. FIPS 46-3. Data Encryption Standard (DES) // <http://csrc.nist.gov> — Reaffirmed October 25, 1999.
151. FIPS 81. DES Modes of Operation // <http://csrc.nist.gov> — 1980 December 2.
152. FIPS Publication 180-2. Specifications for the Secure Hash Standard // <http://csrc.nist.gov> — August 1, 2002.
153. FIPS Publication 197. Specification for the Advanced Encryption Standard // <http://csrc.nist.gov> — November 26, 2001.

154. Fleming R. An attack on a weakened version of TEA // <http://groups.google.com> — sci.crypt posting — October 22, 1996.
155. Fluhrer S. R. Cryptanalysis of the Mercy Block Cipher // <http://citeseer.ist.psu.edu> — Cisco Systems, Inc., San Jose, CA.
156. Fuller J., Millan W. On Linear Redundancy in the AES S-box // <http://eprint.iacr.org> — 2002 — Queensland University of Technology, Brisbane, Australia.
157. Furman V. Differential Cryptanalysis of Nimbus // <http://maths.utime.cn> — Technion, Haifa, Israel — 2002.
158. Gandolfi K., Mourtel C., Olivier F. Electromagnetic Analysis: Concrete Results // <http://citeseer.ist.psu.edu> — 2001 — Gemplus Card International, France.
159. General Report on the Design, Specification and Evaluation of 3GPP Standard Confidentiality and Integrity Algorithms (Release 4) // <http://www.3gpp.org> — 3GPP TR 33.908 V4.0.0 (2001-09).
160. Georgoudis D., Leroux D., Chaves B. S. Specification of the Algorithm. The "FROG" Encryption Algorithm // <http://www.tecapro.com> — Tecnologia Apropriada, June 15, 1998.
161. Gilbert H., Chauvaud P. A Chosen Plaintext Attack of the 16-round Khufu Cryptosystem // <http://citeseer.ist.psu.edu> — France Telecom, Issy-les-Moulineaux, France.
162. Gilbert H., Girault M., Hoogvorst P., Noilhan F., Pornin T., Poupart G., Stern J., Vaudenay S. Decorrelated Fast Cipher: an AES Candidate // <http://citeseer.ist.psu.edu> — May 19, 1998.
163. Giraud C. DFA on AES // <http://citeseer.ist.psu.edu> — Oberthur Card Systems, Puteaux, France.
164. Gladman B. Some Informal Reflections on Rijndael and Twofish // <http://csrc.nist.gov> — March 15, 2000 — Worcester, UK.
165. Gorska A., Gorski K., Kotulski Z., Paszkiewicz A., Szczepanski J. New Experimental Results in Differential-linear Cryptanalysis of Reduced Variants of DES // <http://www.ipt.gov.pl>.
166. Granboulan L. Flaws in differential cryptanalysis of Skipjack // <http://www.di.ens.fr> — Ecole Normale Supérieure.
167. Granboulan L., Nguyen P., Noilhan F., Vaudenay S. DFCv2 // <http://citeseer.ist.psu.edu>.

168. Grosek O., Nemoga K., Zanechal M. Why use bijective S-boxes in GOST-algorithm // <http://www.mat.savba.sk> — Slovak Academy of Sciences, Bratislava — July 30, 1997.
169. Hall C., Kelsey J., Rijmen V., Schneier B., Wagner D. Cryptanalysis of SPEED // <http://citeseer.ist.psu.edu>.
170. D'Halluin C., Bijnens G., Rijmen V., Preneel B. Attack on Six Rounds of Crypton // <http://www.cosic.esat.kuleuven.ac.be> — Katholieke Universiteit Leuven, Heverlee, Belgium.
171. Handschuh H., Heys H. M. A Timing Attack on RC5 // <http://www.gemplus.com>.
172. Handschuh H., Knudsen L. R., Robshaw M. J. Analysis of SHA-1 in Encryption Mode // <http://citeseer.ist.psu.edu> — 2001.
173. Handschuh H., Naccache D. SHACAL // <http://www.cosic.esat.kuleuven.be> — 2000 — Gemplus, Issy-les-Moulineaux.
174. Handschuh H., Naccache D. SHACAL // <http://www.cosic.esat.kuleuven.be> — 2001 — Gemplus, Issy-les-Moulineaux.
175. Handschuh H., Preneel B. On the Security of Double and 2-key Triple Modes of Operation // <http://www.gemplus.com> — 1999.
176. Harpes C. A Generalization of Linear Cryptanalysis applied to SAFER // <http://citeseer.ist.psu.edu> — March 9, 1995 — Swiss Federal Institute of Technology, Zurich.
177. Harpes C., Kramer G. G., Massey J. L. A Generalization of Linear Cryptanalysis and the Applicability of Matsui's Piling-up Lemma // <http://citeseer.ist.psu.edu> — May 19, 1995 — Swiss Federal Institute of Technology, Zurich.
178. Hernandez J. C., Sierra J. M., Ribagorda A., Ramos B., Mex-Perera J. C. Distinguishing TEA from a random permutation: Reduced round versions of TEA do not have the SAC or do not generate random numbers (Abstract) // <http://cat.inist.fr> — 2001.
179. Heys H. M. A Timing Attack on RC5 // <http://citeseer.ist.psu.edu> — Memorial University of Newfoundland, St. John's, Newfoundland, Canada.
180. Heys H. M. A Tutorial on Linear and Differential Cryptanalysis // <http://www.enqr.mun.ca> — Memorial University of Newfoundland, St. John's, Canada.
181. Hong S., Hong D., Ko Y., Chang D., Lee W., Lee S. Differential cryptanalysis of TEA and XTEA (Abstract) // <http://cat.inist.fr> — 2003 — Korea University, Seoul.

182. Hopwood D. Standard Cryptographic Algorithm Naming // <http://www.users.zetnet.co.uk> — Version 1.0.20a — 22 October 2002.
183. Howard P. GSM and UMTS Security // <http://www.isg.rhul.ac.uk> — Vodafone Group — 2005.
184. Iwata T., Yagi T., Kurosawa K. On the Pseudorandomness of KASUMI Type Permutations // <http://citeseer.ist.psu.edu> — July 3, 2003.
185. Jakobsen T., Knudsen L. R. The Interpolation Attack on Block Ciphers // <http://citeseer.ist.psu.edu> — 1997.
186. Jacobson M. J. Jr., Huber K. The MAGENTA Block Cipher Algorithm // <http://www.gel.ulaval.ca> — Deutsche Telekom AG, Darmstadt, Germany — June 8, 1998.
187. Johnson D. B. Future Resiliency: A Possible New AES Evaluation Criterion // <http://csrc.nist.gov> — Certicom, 1999.
188. Kaliski B. S. Jr., Rivest R. L., Sherman A. T. Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DES) // <http://theory.lcs.mit.edu> — MIT Laboratory for Computer Science, Cambridge, MA, USA — 1988.
189. Kaliski B. S. Jr., Robshaw M. J. B. Linear Cryptanalysis Using Multiple Approximations and FEAL // <http://bybin.narod.ru>.
190. Kaliski B. S. Jr., Yin Y. L. On the Security of the RC5 Encryption Algorithm // <http://citeseer.ist.psu.edu> — RSA Laboratories Technical Report TR-602, Version 1.0 — September 1998.
191. Kanda M., Moriai S., Aoki K., Ueda H., Ohkubo M., Takashima Y., Ohta K., Matsumoto T. E2 — a Candidate Cipher for AES // <http://info.isl.ntt.co.jp> — Nippon Telegraph and Telephone Corporation, 1998.
192. Kang J.-S., Preneel B., Ryu H., Chung K. I., Park C. H. Pseudorandomness of Basic Structures in the Block Cipher KASUMI // <http://etrij.etri.re.kr> — ETRI Journal, Volume 25, Number 2, April 2003.
193. Keliher L., Meijer H., Tavares S. E. High Probability Linear Hulls in Q // <http://www.cosic.esat.kuleuven.be>.
194. Kelsey J. Re: Chaining ciphers // <http://cypherpunks.venona.com>.
195. Kelsey J., Kohno T., Schneier B. Amplified Boomerang Attack Against Reduced-Round MARS and Serpent // <http://citeseer.ist.psu.edu>.
196. Kelsey J., Schneier B. Key-Schedule Cryptanalysis of DEAL // <http://www.schneier.com> — Counterpane Systems.
197. Kelsey J., Schneier B., Wagner D. Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES // <http://www.schneier.com> — 1996.

198. Kelsey J., Schneier B., Wagner D. Key Schedule Weaknesses in SAFER+ // <http://www.schneier.com>.
199. Kelsey J., Schneier B., Wagner D. Mod  $n$  Cryptanalysis, with Applications Against RC5P and M6 // <http://www.schneier.com>.
200. Kelsey J., Schneier B., Wagner D. Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA // <http://www.schneier.com>.
201. Kilian J., Rogaway P. How to Protect DES Against Exhaustive Key Search // <http://citeseer.ist.psu.edu> — Full version. — July 28, 1997.
202. Kim J. Combined Differential, Linear and Related-Key Attacks on Block Ciphers and MAC Algorithms // <http://www.cosic.esat.kuleuven.be> — November 2006 — Katholieke Universiteit Leuven.
203. Kim J., Moon D., Lee W., Hong S., Lee S., Jung S. Amplified Boomerang Attack Against Reduced-Round SHACAL // <http://www.iacr.org> — Korea University, Seoul, Korea.
204. Kim K. Construction of DES-like S-boxes Based on Boolean Function Satisfying the SAC // <http://citeseer.ist.psu.edu> — Electronics and Telecommunications Research Institute — Daejeon, Korea.
205. Kim K., Lee S., Park S., Lee D. How to Strengthen DES against Two Robust Attacks // <http://vega.icu.ac.kr> — Korea-Japan Joint Workshop on Information Security and Cryptology — January 24-25, 1995, Inuyama, Japan.
206. Kim K., Lee S., Park S., Lee D. Securing DES S-boxes against Three Robust Cryptanalysis // <http://vega.icu.ac.kr> — Electronics and Telecommunications Research Institute — Taejon, Korea.
207. Kim K., Park S., Lee S. Reconstruction of  $s^2$ DES S-boxes and their Immunity to Differential Cryptanalysis // <http://vega.icu.ac.kr> — Korea-Japan Joint Workshop on Information Security and Cryptology — October 24-26, 1993, Seoul, Korea.
208. Knudsen L. R. Analysis of Camellia // <http://info.isl.ntt.co.jp> — April 28, 2000.
209. Knudsen L. R. Block Ciphers — Analysis, Design, and Applications // <http://www.daimi.au.dk> — July 1, 1994.
210. Knudsen L. R. Cryptanalysis of LOKI91 // <http://citeseer.ist.psu.edu> — Aarhus Universitet, Denmark.
211. Knudsen L. R. DEAL — A 128-bit Block Cipher // <http://www2.mat.dtu.dk> — February 21, 1998 — Revised May 15, 1998.

212. Knudsen L. R. Iterative Characteristics of DES and  $s^2$ -DES // <http://citeseer.ist.psu.edu> — Aarhus University.
213. Knudsen L. R. A Key-schedule Weakness in SAFER K-64 // <http://citeseer.ist.psu.edu> — 1995 — Ecole Normale Supérieure, Paris, France.
214. Knudsen L. R. Quadratic relations in Khazad and Whirlpool // <http://www.cosic.esat.kuleuven.be> — June 27, 2002.
215. Knudsen L. R. Some thoughts on the AES process // <http://csrc.nist.gov> — April 15, 1999.
216. Knudsen L. R. Why SAFER K Changed Its Name // <http://citeseer.ist.psu.edu> — April 1996 — Ecole Normale Supérieure, Paris.
217. Knudsen L. R., Berson T. A. Truncated Differentials of SAFER // <http://citeseer.ist.psu.edu>.
218. Knudsen L. R., Meier W. Improved Differential Attacks on RC5 // <http://citeseer.ist.psu.edu>.
219. Knudsen L., Raddum H. A first report on Whirlpool, NUSH, SC2000, Noekeon, Two-Track-MAC and RC6 // <http://www.cryptonessie.org> — March 07, 2001.
220. Knudsen L. R., Raddum H. On Noekeon // <http://www.cosic.esat.kuleuven.be> — April 24, 2001.
221. Knudsen L. R., Rijmen V. On The Decorrelated Fast Cipher (DFC) and Its Theory // <http://www.esat.kuleuven.ac.be> — University of Bergen.
222. Knudsen L. R., Rijmen V. Two Rights Sometimes Make a Wrong // <http://citeseer.ist.psu.edu> — Katholieke Universiteit Leuven, Heverlee, Belgium.
223. Knudsen L. R., Rijmen V. Weaknesses in LOKI97 // <http://www.unsw.adfa.edu.au> — January 29, 1999 — University of Bergen, Norway.
224. Knudsen L. R., Rijmen V., Rivest R. L. Robshaw M. J. B. On the Design and Security of RC2 // <http://theory.lcs.mit.edu>.
225. Knudsen L. R., Robshaw M. J. B., Wagner D. Truncated Differentials and Skipjack // <http://www.cs.berkeley.edu>.
226. Knudsen L., Wagner D. Integral Cryptanalysis // <http://citeseer.ist.psu.edu> — December 11, 2001.
227. Ko Y., Hong S., Lee W., Lee S., Kang J.-S. Related Key Differential attacks on 27 round of XTEA and full rounds of GOST (Abstract) // <http://www.springerlink.com> — 2004.

228. Kocher P. C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems // <http://citeseer.ist.psu.edu> — Cryptography Research, Inc., San Francisco, USA.
229. Kocher P., Jaffe J., Jun B. Differential Power Analysis // <http://citeseer.ist.psu.edu> — 1999 — Cryptography Research, Inc., San Francisco, CA, USA.
230. Kuhn U. Cryptanalysis of Reduced-Round MISTY // <http://citeseer.ist.psu.edu> — Dresdner Bank AG, Frankfurt, Germany.
231. Kuhn U. Improved Cryptanalysis of MISTY1 // <http://coblitz.codeen.org> — Dresdner Bank AG, Frankfurt, Germany — 2002.
232. Kwan M. The Design of the ICE Encryption Algorithm // <http://www.darkside.com.au> — 1997.
233. Kwon D., Kim J., Park S., Sung S. H., Sohn Y., Song J. H., Yeom Y., Yoon E.-J., Lee S., Lee J., Chee S., Han D., Hong J. New Block Cipher: ARIA // <http://www.nrsi.re.kr>.
234. Lai X., Massey J. A Proposal for a New Block Encryption Standard // <http://bybin.narod.ru> — 1990.
235. Lai X., Massey J., Murphy S. Markov Ciphers and Differential Cryptanalysis // <http://citeseer.ist.psu.edu> — June 14, 1991.
236. Law Y. W. Key Management and Link-Layer Security of Wireless Sensor Networks. Energy-Efficient Attack and Defence. Dissertation to obtain the doctor's degree at the University of Twente // <http://www.ctit.utwente.nl> — University of Twente, Netherlands — 2005.
237. Lebedev A. N., Volchkov A. A. NUSH. Cryptographic Algorithms Based upon the Block Cipher Called «NUSH» // <http://www.cosic.esat.kuleuven.be> — LAN Crypto, Int., Moscow, Russia.
238. Lee C., Kim J., Sung J., Hong S., Lee S. Provable Security for an RC6-like Structure and a MISTY-FO-like Structure against Differential Cryptanalysis // <http://www.cosic.esat.kuleuven.be> — 2006.
239. Lim C. H. A revised version of Crypton — Crypton V1.0 // <http://citeseer.csail.mit.edu> — Future Systems, Inc., Seoul, Korea.
240. Lim C. H. Specification and Analysis of CRYPTON Version 1.0 // <http://citeseer.csail.mit.edu> — Future Systems, Inc., Seoul, Korea — December 22, 1998.
241. Loeb L. Exit DES, enter Rijndael // <http://www.ibm.com> — IBM — November 01, 2000.

242. Lu J., Kim J., Keller N., Dunkelman O. Differential and Rectangle Attacks on Reduced-Round SHACAL-1 // <http://www.cosic.esat.kuleuven.be> — 2006.
243. Lu J., Kim J., Keller N., Dunkelman O. Related-Key Rectangle Attack on 42-Round SHACAL-2 // <http://www.cosic.esat.kuleuven.be>.
244. Lucks S. Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys // <http://citeseer.ist.psu.edu> — University of Mannheim, Germany.
245. Lucks S. BEAST: A fast block cipher for arbitrary blocksizes // <http://citeseer.ist.psu.edu> — Georg-August-Universitat, Gottingen, Germany.
246. Lucks S. On the Security of the 128-Bit Block Cipher DEAL // <http://th.informatik.uni-manheim.de> — University of Mannheim, Germany.
247. Machado A. W. The Nimbus Cipher // <http://www.cosic.esat.kuleuven.be> — Gauss Informatica, Brazil.
248. Mangard S., Pramstaller N., Oswald E. Successfully Attacking Masked AES Hardware Implementations // <http://www.iaik.tu-graz.ac.at> — 2005 — Graz University of Technology, Austria.
249. Massacci F., Marraro L. Towards the Formal Verification of Ciphers: Logical Cryptanalysis of DES // <http://citeseer.ist.psu.edu>.
250. Massey J. L. Announcement of a Strengthened Key Schedule for the cipher SAFER (Secure And Fast Encryption Routine) for both 64 and 128 bit key lengths // <http://groups.google.com> — September 12, 1995.
251. Massey J. L. SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm // <http://citeseer.ist.psu.edu> — 1994 — Swiss Federal Institute of Technology, Zurich.
252. Massey J. L. SAFER K-64: One Year Later // <http://citeseer.ist.psu.edu> — 1995 — Swiss Federal Institute of Technology, Zurich, Switzerland.
253. Massey J. L., Khachatrian G. H., Kuregian M. K. Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES). Submission Document // <http://mcrypt.hellug.gr> — June 12, 1998 — Cylink Corporation, Sunnyvale, CA.
254. Massey J. L., Khachatrian G. H., Kuregian M. K. Nomination of SAFER++ as Candidate Algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) // <http://cosic.esat.kuleuven.be> — September 26, 2000 — Cylink Corporation, Santa Clara, CA, USA.
255. Matsui M. Block Cipher Algorithms MISTY1 and MISTY2 // <http://www.cosic.esat.kuleuven.be> — Edition 2.1 — December 16, 1996.

256. Matsui M. Linear Cryptanalysis Method for DES Cipher // <http://homes.esat.kuleuven.be> — Mitsubishi Electric Corporation, Japan.
257. Matsui M. New Block Encryption Algorithm MISTY // <http://www.cosic.esat.kuleuven.be> — Mitsubishi Electric Corporation, Kamakura, Kanagawa, Japan.
258. Matsui M. Specification of MISTY1 — a 64-bit Block Cipher // <http://www.cosic.esat.kuleuven.be> — Version 1.00 — Mitsubishi Electric Corporation — September 18, 2000.
259. Matsui M. Supporting Document of MISTY1. Version 1.10 // <http://www.cosic.esat.kuleuven.be> — Mitsubishi Electric Corporation — September 25, 2000.
260. Matsui M., Tokita T. MISTY, KASUMI and Camellia Cipher Algorithm Development // <http://global.mitsubishielectric.com> — Mitsubishi Electric — December 2002.
261. McBride L. Q — A Proposal for NESSIE. v2.00 // <http://www.cosic.esat.kuleuven.be> — Mack One Software, Galveston, USA.
262. McLoone M., McCanny J. V. Very High Speed 17 Gbps SHACAL Encryption Architecture (Abstract) // <http://www.springerlink.com> — 2003 — Queen's University of Belfast, Northern Ireland.
263. Menezes A., van Oorschot P., Vanstone S. Handbook of Applied Cryptography. CRC Press, 1996 // <http://www.cacr.math.uwaterloo.ca>.
264. Merkle R. C. Fast Software Encryption Functions // <http://citeseer.ist.psu.edu> — Xerox Parc, Palo Alto, CA — 1998.
265. Merkle R. C. Method and Apparatus for Data Encryption // <http://www.freepatentsonline.com> — United State Patent № 5003597, March 26, 1991.
266. Merkle R., Hellman M. On the Security of Multiple Encryption // <http://www.cs.biu.ac.il> — July 1981.
267. Mirza F. Block Ciphers And Cryptanalysis // <http://members.aol.com> — Royal Holloway University of London.
268. Mister S. Properties of the Building Blocks of Serpent // <http://csrc.nist.gov> — Entrust Technologies — May 15, 2000.
269. Moen V. Integral Cryptanalysis of Block Ciphers // <http://www.ii.uib.no> — May 6, 2002.
270. Moon D., Hwang K., Lee W., Lee S., Lim J. Impossible differential cryptanalysis of reduced round XTEA and TEA (Abstract) // <http://cat.inist.fr> — 2002 — Korea University, Seoul.

Литература

271. Morin P. Provably Secure and Efficient Block Ciphers // <http://cg.scs.carleton.ca> — Carleton University.
272. Muller F., A New Attack Against Khazad // <http://www.iacr.org> — DCSSI Crypto Lab, Issy-les-Moulineaux, France.
273. Murphy S. An Analysis of SAFER // <http://www.isg.rhul.ac.uk> — 22 July 1997 — University of London, Egham, U. K.
274. Murphy S. The Cryptanalysis of FEAL-4 with Twenty Chosen Plaintexts // <http://bybin.narod.ru>.
275. Murphy S., Robshaw M. J. B. Essential Algebraic Structure Within the AES // <http://citeseer.ist.psu.edu> — University of London, Egham, U. K.
276. Murphy S., Robshaw M. Further Comments on the Structure of Rijndael // <http://www.isg.rhul.ac.uk> — University of London, Egham, U. K. — 17 August, 2000.
277. Murphy S., Robshaw M. New Observations on Rijndael. Preliminary Draft // <http://www.isg.rhul.ac.uk> — University of London, Egham, U. K. — 7 August, 2000.
278. Nakahara J. Jr. An Update to Linear Cryptanalysis of SAFER++ // <http://esat.kuleuven.be> — March 12, 2003 — Katholieke Universiteit Leuven, Belgium.
279. Nakahara J. Jr. Cryptanalysis and Design of Block Ciphers // <http://citeseer.ist.psu.edu> — Katholieke Universiteit Leuven — June 2003.
280. Nakahara J. Jr. The Statistical Evaluation of the NESSIE Submission NUSH // <http://www.cosic.esat.kuleuven.be> — Katholieke Universiteit Leuven, Belgium — October 22, 2001.
281. Nakahara J. Jr., Preneel B., Vandewalle J. Impossible Differential Attacks on Reduced-Round SAFER Ciphers // <http://citeseer.ist.psu.edu> — March 12, 2003 — Katholieke Universiteit Leuven, Belgium.
282. Nakahara J. Jr., Preneel B., Vandewalle J. More Square Attacks on Rijndael Block Cipher // <http://citeseer.ist.psu.edu> — Katholieke Universiteit Leuven, Leuven-Heverlee, Belgium — Version 1.02 — January 16, 2002.
283. Nechvatal J., Barker E., Bassham L., Burr W., Dworkin M., Foti J., Roback E. Report on the Development of the Advanced Encryption Standard (AES) // <http://csrc.nist.gov> — National Institute of Standards and Technology.
284. Nechvatal J., Barker E., Dodson D., Dworkin M., Foti J., Roback E. Status report on the first round of the development of the advanced encryption standard // <http://csrc.nist.gov> — National Institute of Standards and Technology.

285. Needham R. M., Wheeler D. J. Tea extensions // <http://www.cl.cam.ac.uk> — October 1996.
286. Nyberg K. Cryptographic Algorithms for UMTS // <http://www.tcs.hut.fi> — Nokia Group, Finland — July 2004.
287. Ohkuma K., Sano F., Muratani H., Motoyama M., Kawamura S. Specification of Hierocrypt-L1 // <http://www.cosic.esat.kuleuven.be> — Toshiba Corporation — September 25, 2000.
288. Ohkuma K., Sano F., Muratani H., Motoyama M., Kawamura S. Specification on a Block Cipher: Hierocrypt-L1 (revised) // <http://www.cosic.esat.kuleuven.be> — Toshiba Corporation — September 2001.
289. Ohkuma K., Sano F., Muratani H., Motoyama M., Kawamura S. Specification of Hierocrypt-3 // <http://www.cosic.esat.kuleuven.be> — Toshiba Corporation — September 25, 2000.
290. Ohta K., Aoki K. Linear Cryptanalysis of the Fast Data Encipherment Algorithm // <http://bybin.narod.ru>.
291. Ohta H., Matsui M. RFC 2994: A Description of the MISTY1 Encryption Algorithm // <http://tools.ietf.org> — Mitsubishi Electric Corporation — November 2000.
292. Ohta K., Moriai S., Aoki K. Improving the Search Algorithm for the Best Linear Expression // <http://bybin.narod.ru>.
293. Oreku G. S., Li J., Pazynyuk T., Mtenzi F. J. Modified S-box to Archive Accelerated GOST // <http://paper.ijcsns.org> — International Journal of Computer Science and Network Security, VOL. 7 No. 6, June 2007.
294. Oswald E., Preneel B. A Theoretical Evaluation of some NESSIE Candidates regarding their Susceptibility towards Power Analysis Attacks // <http://www.iaik.tugraz.at> — Katholieke Universiteit Leuven, Belgium — October 4, 2002.
295. Outerbridge R. AES Candidate DEAL // <http://csrc.nist.gov> — Interac Association — August 21, 1998.
296. Parker M. G. Generalized S-box Nonlinearity // <http://www.ii.uib.no> — 2003 — University of Bergen, Norway.
297. Patel S., Ramzan Z., Sundaram G. Sha-zam: A Block Cipher. Fast as DES, Secure as SHA // <http://theory.ics.mit.edu>.
298. Patel S., Ramzan Z., Sundaram G. Sha-zam: Short Circuiting Cryptanalysis // <http://citeseer.ist.psu.edu>.
299. Peacham D., Thomas B. A DFA attack against the AES key schedule // <http://www.siventure.com> — October 26, 2006 — SiVenture.

Литература

300. Pieprzyk J., Tombak L. Soviet Encryption Algorithm // <http://citeseer.ist.psu.edu> — University of Wollongong — June 1, 1994.
301. Piret G.-F. Block Ciphers: Security Proofs, Cryptanalysis, Design, and Fault Attacks // <http://www.di.ens.fr> — Universite Catolique de Louvain — Janvier 2005.
302. Piret G., Quisquater J.-J. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad // <http://www.dice.ucl.ac.be> — 2003 — Universite Catolique de Louvain, Belgium.
303. Piret G., Quisquater J.-J. Integral Cryptanalysis on reduced-round SAFER++ // <http://citeseer.ist.psu.edu> — Universite catolique de Louvain, Louvain-la-Neuve, Belgium.
304. Piret G., Quisquater J.-J. Security of the MISTY Structure in the Lubym-Rackoff Model: Improved Results // <http://www.dice.ucl.ac.be> — 2004 — UCL Crypto Group, Belgium.
305. Portfolio of recommended cryptographic primitives // <http://www.cryptonessie.org> — NESSIE consortium, February 27, 2003.
306. Poupard G., Vaudenay S. Decorrelated Fast Cipher: an AES Candidate well suited for low cost smart cards applications // <http://csrc.nist.gov> — Ecole Normale Supérieure, Paris, France.
307. Preneel B., Biryukov A., Oswald E., Van Rompay B., Granboulan L., Dottax E., Murphy S., Dent A., White J., Dichtl M., Pyka S., Schafheutle M., Serf P., Biham E., Barkan E., Dunkelman O., Quisquater J.-J., Ciet M., Sica F., Knudsen L., Parker M., Raddum H. Public Report D20: NESSIE security report // <http://www.cosic.esat.kuleuven.be> — February 19, 2003 — Version 2.0.
308. Preneel B., Bosselaers A., Ors S. B., Biryukov A., Granboulan L., Dottax E., Murphy S., Dent A., White J., Dichtl M., Pyka S., Serf P., Biham E., Barkan E., Dunkelman O., Ciet M., Sica F., Knudsen L., Raddum H. NESSIE Public Report D18: Update on the selection of algorithms for further investigation during the second round // <http://www.cosic.esat.kuleuven.be> — Version 1.0 — March 11, 2002.
309. Preneel B., Nuttin M., Rijmen V., Buelens J. Cryptanalysis of the CFB mode of the DES with a reduced number of rounds // <http://citeseer.ist.psu.edu> — Katholieke Universiteit Leuven, Belgium.
310. Preneel B., Rijmen V., Bosselaers A. Recent Developments in the Design of Conventional Cryptographic Algorithms // <http://www.cosic.esat.kuleuven.ac.be> — Katholieke Universiteit Leuven, Belgium — 18 September 1998.

311. Preneel B., Van Rompay B., Granboulan L., Martinet G., Dichtl M., Schafheutle M., Serf P., Biblivitz A., Biham E., Dunkelman O., Ciet M., Quisquater J.-J., Sica F. NESSIE Public Report D14: Report on the Performance Evaluation of NESSIE Candidates I // <http://www.cosic.esat.kuleuven.be> — Version 3.3 — November 20, 2001.
312. Preneel B., Van Rompay B., Granboulan L., Martinet G., Murphy S., Shipsey R., White J., Dichtl M., Serf P., Schafheutle M., Biham E., Dunkelman O., Ciet M., Quisquater J.-J., Sica F., Knudsen L., Raddum H. NESSIE Phase I: Selection of Primitives // <http://www.cryptonessie.org> — September 23, 2001.
313. Preneel B., Van Rompay B., Granboulan L., Martinet G., Murphy S., Shipsey R., White J., Dichtl M., Serf P., Schafheutle M., Biham E., Dunkelman O., Furman V., Ciet M., Quisquater J.-J., Sica F., Knudsen L., Raddum H. NESSIE Public Report D13: Security Evaluation of NESSIE First Phase // <http://www.cosic.esat.kuleuven.be> — September 23, 2001.
314. Preneel B., Van Rompay B., Ors S. B., Biryukov A., Granboulan L., Dottax E., Dichtl M., Schafheutle M., Serf P., Pyka S., Biham E., Barkan E., Dunkelman O., Stolin J., Ciet M., Quisquater J.-J., Sica F., Raddum H., Parker M. NESSIE Public Report D21: Performance of Optimized Implementations of the NESSIE Primitives // <http://www.cosic.esat.kuleuven.be> — February 20, 2003.
315. Quisquater J.-J., Standaert F.-X. Exhaustive Key Search of the DES: Updates and Refinements // <http://cr.ypt.to> — Universite Catholique de Louvain, Belgium.
316. Raddum H. The Statistical Evaluation of the NESSIE Submission CS-Cipher // <http://citeseer.csail.mit.edu> — The University of Bergen, Norway — February 18, 2002.
317. Raddum H. The Statistical Evaluation of the NESSIE Submission Khazad // <http://www.cosic.esat.kuleuven.be> — The University of Bergen, Norway — February 18, 2002.
318. Raddum H., Knudsen L. R. A Differential Attack on Reduced-Round SC2000 // <http://citeseer.ist.psu.edu> — University of Bergen — 30<sup>th</sup> April 2001.
319. RAND Corporation. A Million Random Digits with 100,000 Normal Deviates. — Glencoe, IL: Free Press Publishers, 1955.
320. RC2 source code. sci.crypt anonymous post // <http://groups.google.com> — January 29, 1996.

Литература

321. Reddy Andem V. A Cryptanalysis of the Tiny Encryption Algorithm // <http://cs.ua.edu> — 2003 — University of Alabama, Tuscaloosa, Alabama.
322. Rijmen V. Reference Implementation of SHARK // <ftp://ftp.esat.kuleuven.ac.be> — K. U. Leuven — December, 1995.
323. Rijmen V., Daemen J., Preneel B., Bosselaers A., De Win E. The Cipher SHARK // <ftp://ftp.esat.kuleuven.ac.be> — Katholieke Universiteit Leuven, Heverlee, Belgium.
324. Rijmen V., Preneel B. Cryptanalysis of McGuffin // <ftp://ftp.esat.kuleuven.ac.be> — Katholieke Universiteit Leuven, Heverlee, Belgium.
325. Ritter T. Ladder-DES: A Proposed Candidate to Replace DES // <http://zone-h.org> — Ritter Software Engineering, Austin, Texas, USA — February 22, 1994.
326. Ritter T. Linear Cryptanalysis: A Literature Survey // <http://www.cyphersbyritter.com> — 1997.
327. Rivest R. RFC 2268: A Description of the RC2® Encryption Algorithm // <http://www.ietf.org> — MIT Laboratory for Computer Science and RSA Data Security, Inc. — March 1998.
328. Rivest R. L. The RC5 Encryption Algorithm // <http://citeseer.ist.psu.edu> — MIT Laboratory for Computer Sciense, Cambridge — Revised March 20, 1997.
329. Rivest R. L., Robshaw M. J. B., Sidney R., Yin Y. L. The RC6 Block Cipher // <http://www.rsasecurity.com> — Version 1.1 — August 20, 1998.
330. Robshaw M. J. B. Block Ciphers // <http://citeseer.ist.psu.edu> — RSA Laboratories Technical Report — Version 2.0 — August 2, 1995.
331. Rogaway P. The Security of DESX // <http://www.cs.ucdavis.edu> — University of California, Davis, CA, USA — Draft 2.0. — July 5, 1996.
332. Rogaway P., Coppersmith D. A Software-Optimized Encryption Algorithm // <http://citeseer.ist.psu.edu> — Last revised September 18, 1997.
333. Russell M. D. Tinyness: An Overview of TEA and Related Ciphers // <http://www-users.cs.york.ac.uk> — 27<sup>th</sup> Feb 2004 — Draft v0.3.
334. Saarinen M.-J. A chosen key attack against the secret S-boxes of GOST // <http://citeseer.ist.psu.edu> — August 12, 1998.
335. Saarinen M.-J. O. Cryptanalysis of Block Ciphers Based on SHA-1 and MD5 // <http://www.m-js.com> — Helsinki University of Technology, Finland.
336. SAFER+ // <http://fn2.freenet.edmonton.ab.ca>.

337. Satoh A., Morioka S. Unified Hardware Architecture for 128-bit Block Ciphers AES and Camellia // <http://islab.oregonstate.edu> — IBM Japan Ltd.
338. Savard J. LUCIFER: the first block cipher // <http://www.quadibloc.com>.
339. Savard J. SAFER (Secure And Fast Encryption Routine) // <http://www.quadibloc.com>.
340. Schneier B. The Blowfish encryption algorithm // <http://www.schneier.com>.
341. Schneier B. The Blowfish encryption algorithm — one year later // <http://www.schneier.com>.
342. Schneier B. Description of a new variable-length key 64-bit block cipher (blowfish) // <http://www.schneier.com>.
343. Schneier B. Products that use Blowfish // <http://www.schneier.com>.
344. Schneier B. Speed comparisons of block ciphers on a Pentium // <http://www.schneier.com>.
345. Schneier B. Status of TEA // <http://groups.google.com> — sci.crypt posting — September 11, 1998.
346. Schneier B., Kelsey J. Unbalanced Feistel Networks and Block-Cipher Design // <http://www.schneier.com> — Counterpane Systems.
347. Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., Ferguson N. Twofish: A 128-bit Block Cipher // <http://www.schneier.com> — June 15, 1998.
348. Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., Ferguson N., Kohno T., Stay M. The Twofish Team's Final Comments on AES Selection // <http://csrc.nist.gov> — May 15, 2000.
349. Schramm K., Paar C. Higher Order Masking of the AES // <http://www.crypto.ruhr-uni-bochum.de> — 2006 — Ruhr University Bochum, Germany.
350. Schroeppel R. AES Comments // Public Comments Regarding The Advanced Encryption Standard (AES) Development Effort. Round 2 Comments — <http://csrc.nist.gov> — University of Arizona — May 15, 2000.
351. Schroeppel R. Hasty Pudding Cipher Specification // <http://www.cs.arizona.edu> — June 1998, revised May 1999.
352. Schroeppel R. The Hasty Pudding Cipher: One Year Later // <http://www.cs.arizona.edu> — June 12, 1999.
353. Schroeppel R. The Hasty Pudding Cipher: Specific NIST Requirements // <http://www.cs.arizona.edu> — June 1998.

354. Schroepel R. Tweaking the Hasty Pudding Cipher // <http://www.cs.arizona.edu> — May 14, 1999.
355. Schroepel R., Orman H. An Overview of the Hasty Pudding Cipher // <http://www.cs.arizona.edu> — July 1998.
356. Schubert A., Meyer V., Anheier W. Reusable Cryptographic VLSI Core Based on the SAFER K-128 Algorithm with 251.8 Mbit/s Throughput // <http://www.item.uni-bremen.de> — University of Bremen, Germany, 1998.
357. Shimoyama T., Yanami H., Yokoyama K., Takenaka M., Itoh K., Yajima J., Torii N., Tanaka H. Specification and Supporting Document of the Block Cipher SC2000 // <http://www.cosic.esat.kuleuven.be> — 2000.
358. Skipjack and KEA algorithm specifications // <http://csrc.nist.gov> — Version 2.0 — May 29, 1998.
359. Skorobogatov S. P., Anderson R. J. Optical Fault Induction Attacks // <http://www.cl.cam.ac.uk> — 2002 — University of Cambridge, United Kingdom.
360. Smith J. L. Recirculating Block Cipher Cryptographic System, United States Patent № 3796830, March12, 1974 // <http://www.freepatentsonline.com>.
361. Song B., Seberry J. Further Observations on the Structure of the AES Algorithm // <http://citeseer.ist.psu.edu> — University of Wollongong, Australia.
362. Specification of ARIA // <http://www.nrsi.re.kr> — January 2005 — National Security Research Institute.
363. Specification of E2 — a 128-bit Block Cipher // <http://info.isl.ntt.co.jp> — Nippon Telegraph and Telephone Corporation, June 14, 1998.
364. Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 1: f8 and f9 Specification (Release 6) // <http://www.3gpp.org> — 3GPP TS 35.201 V6.1.0 (2005-09).
365. Specification of the 3GPP Confidentiality and Integrity Algorithms. Document 2: KASUMI Specification // <http://portal.etsi.org> — Version: 1.0 — December 23, 1999.
366. Standaert F.-X., Piret G., Quisquater J.-J. Cryptanalysis of Block Ciphers: A Survey // <http://citeseer.ist.psu.edu> — Universite catolique de Louvain, Belgium — 2003.
367. Stern J., Vaudenay S. CS-Cipher // <http://www.cosic.esat.kuleuven.be> — Ecole Normale Supérieure.
368. Sugita M., Kobara K., Imai H. Security of Reduced Version of the Block Cipher Camellia against Truncated and Impossible Differential Cryptanalysis // <http://www.iacr.org>.

369. Supporting Document on E2 // <http://info.isl.ntt.co.jp> — Nippon Telegraph and Telephone Corporation, April 1, 1999.
370. Tanaka H., Kaneko T. An Expansion Algorithm for Higher Order Differential Cryptanalysis of Secret Key Ciphers // <http://www.nict.go.jp> — Journal of the National Institute of Information and Communications Technology, Vol. 52, Nos. 1/2, 2005.
371. Tanaka H., Sugio N., Kaneko T. A Study on Higher Order Differential Cryptanalysis of 64 Bit Block Cipher KASUMI // <http://www2.nict.go.jp> — Journal of the National Institute of Information and Communications Technology, vol. 52, Nos. 1/2, 2005.
372. Tang J. A Survey of Cryptanalysis of Rijndael // <http://www.cs.auckland.ac.nz> — The University of Auckland.
373. The AES Lounge // <http://iaik.tu-graz.ac.at>.
374. The Block Cipher Square // <http://www.esat.kuleuven.ac.be>.
375. Van Rompay B. NESSIE Public Report D02: Report on the NESSIE Call // <http://www.cosic.esat.kuleuven.be>.
376. Van Rompay B. NESSIE Public Report D07: Response to the NESSIE Call // <http://www.cosic.esat.kuleuven.be>.
377. Van Rompay B., Knudsen L. R., Rijmen V. Differential cryptanalysis of the ICE encryption algorithm // <http://www.cosic.esat.kuleuven.ac.be> — 1998.
378. Van Rompay B., Rijmen V., Nakahara J. Jr. A first report on CS-Cipher, Hierocrypt, Grand Cru, SAFER++ and SHACAL // <http://www.cosic.esat.kuleuven.be> — Katholieke Universiteit Leuven, Belgium — March 12, 2001.
379. Vaudenay S. On the Need of Multipermutations: Cryptanalysis of MD4 and SAFER // <http://citeseer.ist.psu.edu> — November 16, 1994 — Ecole Normale Supérieure, Paris, France.
380. Vaudenay S. On the Security of CS-Cipher // <http://www.cosic.esat.kuleuven.be> — Ecole Normale Supérieure.
381. Vaudenay S. On the Weak Keys of Blowfish // <http://citeseer.ist.psu.edu>.
382. Vaudenay S. Provable security for block Ciphers by decorrelation // <http://citeseer.ist.psu.edu> — Ecole Normale Supérieure, Paris — June 1998.
383. Vaudenay S. Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS... // <http://lasecwww.epfl.ch> — Swiss Federal Institute of Technology.
384. Vaudenay S., Moriai S. Comparison of the Randomness Provided by Some AES Candidates // <http://www.nist.gov>.

385. Wallen J. Design Principles of the KASUMI Block Cipher // <http://citeseer.ist.psu.edu> — Helsinki University of Technology.
386. Wagner D. The boomerang attack // <http://www.cs.berkeley.edu> — U. C. Berkeley.
387. Wagner D., Ferguson N., Schneier B. Cryptanalysis of FROG // <http://www.nist.gov>.
388. Wernsdorf R. Functions of SERPENT Generate the Alternating Group // <http://csrc.nist.gov> — SIT GmbH, Berlin, Germany — May 12, 2000.
389. What are RC5 and RC6? // <http://www.rsasecurity.com> — RSA Laboratories — 2004.
390. What is RC2? // <http://www.rsasecurity.com> — RSA Laboratories — 2004.
391. Wheeler D., Needham R. A Large Block DES-like Algorithm // <http://www.cl.cam.ac.uk> — Cambridge University, England — November 1994.
392. Wheeler D. J., Needham R. M. Correction to xtea // <http://www.movable-type.co.uk> — Computer Laboratory, Cambridge University, England — October 1998.
393. Wheeler D. J., Needham R. M. TEA, a Tiny Encryption Algorithm // <http://www.cl.cam.ac.uk> — Computer Laboratory, Cambridge University, England.
394. Wiener M. Efficient DES Key Search // <http://citeseer.ist.psu.edu> — Bell-Northern Research, Ottawa, Canada — August 20, 1993.
395. Wikipedia, the free encyclopedia — <http://en.wikipedia.org>.
396. Williams C. S. Proposal for a Tweak to Cylink's AES Candidate algorithm SAFER+ // <http://csrc.nist.gov> — April 14, 1999 — Cylink Corporation, Sunnyvale, CA.
397. Wu W.-L., Feng D.-G. Collision Attack on Reduced-Round Camellia // <http://eprint.iacr.org> — Chinese Academy of Sciences, Beijing, China.
398. Wu W.-L., Feng D.-G. Linear cryptanalysis of NUSH block cipher // <http://www.scichina.com> — Chinese Academy of Sciences — 2002.
399. Wu W., Zhang W., Feng D. Impossible Differential Cryptanalysis of ARIA and Camellia // <http://eprint.iacr.org> — Chinese Academy of Sciences, Beijing, China.
400. Xiao L., Heys H. M. An Improved Power Analysis Attack Against Camellia's Key Schedule // <http://eprint.iacr.org> — September 22, 2005.

401. Yajima J., Takenaka M., Torii N., Itoh K. Results of New Boolean Functions for Serpent S-boxes // <http://csrc.nist.gov> — Fujitsu Laboratories Ltd. — May 11, 2000.
402. Yeh Y.-S., Lin C.-H., Wang C.-C. Dynamic GOST // <http://www.iis.sinica.edu.tw> — September 2, 1998 — Journal of Information Science and Engineering 16, 857-861 (2000).
403. Yeom Y., Park S., Kim I. On the Security of CAMELLIA against the Square Attack // <http://www.iacr.org> — National Security Research Institute, Daejeon, Korea.
404. Yin Y. L. A Note on the Block Cipher Camellia // <http://info.isl.ntt.co.jp> — NTT Multimedia Communication Laboratories, Palo Alto, CA — Version 1.1 — August 18, 2000.
405. Yin Y. L. The RC5 Encryption Algorithm: Two Years On // RSA Laboratories' CryptoBytes, v. 2, n. 3 — Winter 1997 — <http://www.olymp.org>.
406. Zhang W., Wu W.-L., Zhang L., Feng D.-G. Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-192 // <http://www.lois.cn> — 2006 — Chinese Academy of Science, Beijing, China.
407. Zheng Y. The SPEED Cipher // <http://labs.calyptix.com> — 1997 — Monash University, Melbourne, Australia
408. Zheng Y., Matsumoto T., Imai H. Duality between Two Cryptographic Primitives // <http://citeseer.ist.psu.edu> — Yokohama National University, Yokohama, Japan.

# Предметный указатель

## 3

3GPP 270

## 4

4-IPHT 420  
4-PHT 419

## A

Ascom Systec 70  
AT&T 24  
Auscrypt 295

## B

Bellcore 50  
Bluetooth 424

## C

CFS 180  
Cisco Systems 334  
Clipper 457  
CMOS 94  
Counterpane Systems 61,  
160, 483  
CRC32 24  
Cryptography Research 46  
CRYPTREC 133, 348  
CS Communication  
& Systemes 70, 149  
Cylink 63, 394, 408

## D

DataCash 328  
Deutsche Telekom 61, 63, 316

## E

Ecole Normale  
Supérieure 63, 70  
EDE 173, 179

ENS 148, 195

Entrust Technologies 63,  
134, 135

ETH Zurich 265

EUROCRYPT 206

## F

Fast Software Encryption 149

Fortezza 457

France Telecom 195, 295

FTP 453

Fujitsu Laboratories 70, 424

Future Systems 63, 141

## G

Gauss Informatica 348

Gemplus 70, 442

GSM 281

## H

Hifn Incorporated 483

## I

IBM 63, 94, 162, 178, 301, 319

IPHT 399, 412

## K

KEA 460

## L

Lotus 376

Lotus Notes 376

Lucent Technologies 454

## M

MAC 70, 78, 278

Mack One Software 374

Microsoft Xbox 477

Mitsubishi Electric 70, 123,  
207, 270, 334

## N

NBS 161

NIST 61, 65, 70, 96, 148, 457  
NTT 63, 70, 123, 201, 206, 270

## O

OpenSSL 97  
ORCA 24

## P

PGP 270  
PHT 396, 408, 410  
Proton World 351

## R

RAND Corporation 292

RFC 376

RSA Data Security 70, 184,  
376, 381, 385, 391

RSA Laboratories 63, 386, 391

## S

S/MIME 376  
SAGE 270

sci.crypt 376, 404

SHS 442

Siemens 70

Spice 249

SP-сеть 11, 12, 136, 149, 229,  
301, 395, 424, 433

## T

Technion 70, 351

Tecnologia Apropriada 62,  
63, 212

Toshiba 70, 229, 239

**U**

Unix 180  
Usenet 376

**W**

Wide-Trail Strategy 282

Wikipedia 301

**X**

Xerox 288

**A**

Академия Вооруженных Сил Австралии 295

Академия Наук Армении 408

Алгоритм шифрования:

2K-DES 42

2-key Triple DES 173

2x Four-Rung

Ladder-DES 183

3-key Triple DES 174, 176

Aardvark 82

AES (Rijndael) 12, 52, 61, 62, 63, 65, 66, 69, 81, 84, 103, 108, 132, 148, 195, 200, 218, 219, 352, 374, 416, 433, 451, 469

AES-128 88

AES-192 88, 95, 97

AES-256 88, 96

Akelarre 97

Anubis 70, 103, 283, 352, 374

Bear 82, 108, 109, 114, 433, 451, 454

BEAST 114

BEAST-RK 117

Biham-DES 38, 191

Block TEA 480

Block TEA 2 481

Blowfish 10, 42, 45, 52, 118, 218, 382, 469, 483, 493

Camellia 70, 123, 207, 287, 347, 445, 498

CAST-128 10, 134, 135, 500

CAST-256 12, 40, 63, 134, 135, 510

COMP128 47

Corrected Block TEA 481

Crypton 12, 63, 141, 469

Crypton v0.5 141, 147

Crypton v1.0 141, 148, 515

CS-Cipher (CS) 70, 148

DEA 156, 162

DEAL 27, 63, 156

DES 10, 12, 18, 24, 25, 28, 34, 37, 42, 45, 46, 47, 50, 61, 69, 79, 81, 89, 109, 118, 156, 160, 207, 254, 265, 288, 301, 311, 315, 319, 382, 433, 457, 461, 517

DES с независимыми подключами 34, 38, 52, 89, 185

DES3 174

DESX 34, 38, 184

DFC 63, 195, 253

DFCv2 200

Double DES 18, 25, 109, 172, 180, 182, 193

E2 63, 201, 207

f8 277

FEAL 34, 37, 52, 206

FEAL-4 207, 211

FEAL-8 211

FEAL-16 211

FEAL-32 211

FEAL-N 207

FEAL-NX 212

Four-Rung Ladder-DES 181

FROG 12, 53, 62, 63, 212

GDES с независимыми подключами 188

Generalized DES (GDES) 34, 186

GOST⊕ 80

GOST-H 80

Grand Cru 70, 219, 352, 374

Hierocrypt-3 70, 239

Hierocrypt-L1 70, 229, 239

HPC 12, 53, 63, 199, 246

HPC-Extended 247

HPC-Long 247

HPC-Medium 247, 250

HPC-Short 247

HPC-Tiny 247

ICE 34, 254

ICEBERG 259

ICE-n 257, 259

IDEA 43, 45, 52, 70, 82, 97, 118, 265, 279, 382

IPES 265

KASUMI 43, 126, 270, 334, 346, 519

Khafre 43, 52, 288

Khazad 70, 282

Khazad-0 285

Khufu 43, 52, 288

KN 43

Ladder-DES 109, 181

Lion 82, 108, 111, 114, 433, 451, 454

Lioness 82, 108, 112, 451, 454

LOKI 57, 295

LOKI99 56, 57, 295

LOKI91 57, 295

LOKI97 63, 211, 295

Lucifer 10, 58, 162, 171, 301, 319

MacGuffin (McGuffin) 312

Madryga 384

MAGENTA 61, 63, 316

MARS 40, 63, 65, 67, 95, 319, 522

Mercy 328

MISTY 43, 270, 348

MISTY1 70, 123, 127, 270, 279, 280, 287, 334, 445

MISTY2 334, 345

Nimbus 70, 348

Noekeon 38, 70, 351

NUSH 38, 70, 357, 374

PES 265, 268, 279

PURE 43

Q 70, 374

Q 2.0 374

Quadruple DES

(QDES) 109, 181

RC2 33, 375, 381

RC4 333, 381

RC5 10, 33, 37, 45, 52, 67, 97, 218, 364, 381, 391, 462

RC5-32/12/16 33, 382, 386

RC5-32/16 469

RC5-64/16/16 387

RC5KFR 389

RC5P 388

RC5PFR 389

RC5RA 390

RC5-w/R/b 382

RC5XOR 388

RC5XOR-32/12/16 388

- RC6 11, 63, 65, 66, 67, 70,  
95, 382, 391
- RC6-32/20/16 391
- RC6-32/20/24 391
- RC6-32/20/32 391
- RC6-w/R/b 391
- RDES 188
- RDES-1 190
- RDES-2 190
- RDES-4 190
- REDOC-II 118
- Rijndael (AES) 142
- RSA 47, 381
- s<sup>2</sup>DES 34, 190, 525
- s<sup>3</sup>DES 190, 192, 527
- s<sup>4</sup>DES 191
- s<sup>5</sup>DES 79, 191, 529
- SAFER 82, 265, 408, 424
- SAFER K 394, 408
- SAFER K-128 403, 404
- SAFER K-64 58, 394,  
409, 421
- SAFER SK 394, 408
- SAFER SK-128 404, 416
- SAFER SK-40 404
- SAFER SK-64 402, 404, 413
- SAFER+ 11, 27, 63, 218,  
408, 418
- SAFER+/256 415
- SAFER++ 70, 418
- SC2000 70, 374, 424
- SEAL 82, 115
- Serpent 11, 40, 61, 63, 65,  
66, 67, 95, 114, 200, 374,  
416, 432, 486
- Serpent-0 433
- Serpent-1 433
- SHACAL 70, 287, 442
- SHACAL-0 445
- SHACAL-1 442
- SHACAL-2 446
- SHARK 12, 43, 103, 282,  
382, 451
- SHARK\* 451
- Sha-zam 454
- Simplified TEA (STEA) 477
- Skipjack 23, 43, 119, 457
- Skipjack-3XOR 22, 462
- SPEED 462
- Square 12, 103, 142, 433,  
451, 469
- TDEA 174
- TEA 10, 474
- Thin-ICE 34, 257, 258
- TREYFER 42
- Triple DES 18, 27, 52, 109,  
157, 170, 171, 173, 178,  
181, 194, 195
- Twofish 61, 63, 65, 67, 93,  
94, 148, 483
- Twofish-FK 394, 491
- xDES<sup>0</sup> 193
- xDES<sup>1</sup> 193
- xDES<sup>2</sup> 194
- xDES<sup>3</sup> 194
- xDES<sup>i</sup> 193
- XTEA 478
- XXTEA 481
- ГОСТ 28147-89 8, 10, 24,  
42, 53, 73, 119, 185, 253
- Диффи—Хеллмана 460
- Джонса 180
- Мэта Блейза 179
- АНБ 162, 171, 376, 457, 461
- Андерсон, Росс 61, 63,  
108, 432
- Анубис 103
- Аоки, Казумаро 70, 123
- Атака 18
- DEMA 47
- SEMA 47
- slicing 347
- Square 92, 93, 133, 147, 408
- активная 20, 44, 48
- алгебраическая 96
- встреча посередине 25, 59,  
114, 116, 172, 178, 186,  
193, 271
- высокоточным облучением  
шифратора 48
- изменением напряжения  
питания шифратора 48
- изменением тактовой  
частоты шифратора 48
- локальным нагревом  
шифратора 49
- методом бумеранга 38, 95,  
132, 295, 423, 449
- методом грубой силы 23,  
37, 169, 296, 461
- методом Дэвиса 169
- методом интерполяции 42,  
92, 132
- методом поиска  
коллизий 133
- на основе выбранных  
открытых  
текстов 175, 179
- на основе сбоев 48, 52, 97
- на основе шифртекста 23,  
37, 102
- на связанных ключах 53,  
54, 92, 95, 175, 179, 185,  
186, 193, 280, 287, 296,  
357, 381, 402, 415, 449,  
468, 476, 480
- наведением  
электромагнитного  
 поля 49
- нарушением  
электрических контактов
- шифратора 49
- пассивная 19, 44
- по времени  
выполнения 45, 65, 97,  
148, 269, 387
- по побочным каналам 44,  
48, 97, 133
- по потребляемой  
мощности 46, 65, 94,  
97, 133, 141, 148, 287,  
416, 450
- по сообщениям  
об ошибках 47
- с аддитивным  
выбором 21, 40,  
112, 281
- с выбранным открытым  
текстом 20, 269
- с выбранным  
шифртекстом 21
- с известным открытым  
текстом 20, 23
- с известным  
шифртекстом 19
- с помощью невозможных  
дифференциалов 43, 93,  
95, 132, 281, 295, 347,  
408, 462, 477, 480
- с помощью усеченных  
дифференциалов 92,  
132, 287, 480
- сдвиговая 40, 54, 132, 477
- Аутентификация 5, 57, 270,  
304, 424
- Аутерридж, Ричард 63

**Б**

Баркан, Элад 281  
 Баррето, Пауло 70, 103, 282  
 Бен-Ароя, Ишаи 189, 312  
 Бирюков, Алекс 40, 43, 169,  
     191, 386, 388  
 Бихам, Эли 27, 43, 50, 54, 61,  
     63, 93, 94, 108, 168, 175, 179,  
     186, 189, 191, 211, 281, 294,  
     311, 312, 351, 432  
 Бланден, Марк 280  
 Блейз, Мэт 117, 180, 312  
 Боне, Дэн 50  
 Борст, Йохан 70, 219  
 Браун, Лоуренс 63, 295

**В**

Вагнер, Дэвид 38, 40, 63,  
     295, 348, 381, 402, 468, 476,  
     480, 483  
 Ван Аске, Жиль 70, 351  
 Вектор инициализации 15, 76,  
     78, 179, 216, 251, 455  
 Винер, Майкл 171  
 Воденэ, Серж 47, 70, 122, 148,  
     155, 195, 200  
 Волчков, Алексей  
     Анатольевич 70, 357  
 Ву, Вен-Линг 363, 373

**Г**

Гамма шифра 76, 277  
 Генератор  
     псевдослучайных чисел 5,  
     62, 82, 109, 333  
     случайных чисел 4  
 Георгоудис, Дианелос 212

**Д**

Данкельман, Орр 281  
 Деймен, Джоан 12, 63, 70,  
     142, 219, 282, 351, 433,  
     451, 469  
 ДеМилло, Ричард 50  
 ден Боер, Берт 211  
 Детектор изменения 52  
     напряжения 52  
     освещенности 53  
     синхронизации 52  
     частоты питания 52

Дрезденский банк 280  
 Дублирование вычислений 53

**Е**

Единое расширение ключа  
 SAFER+ 416, 421

**Ж**

Жен, Юлиан 462

**З**

Закладка 62  
 Запас криптостойкости 22,  
     66, 94

**И**

Идентификация 70  
 Избыточные вычисления 53  
 Имитоприставка 24, 78  
 Ин, Икван Лайза 70, 386, 391  
 Ито, Коучи 70  
 Ичикава, Тецуя 70, 123, 334  
 Ишии, Чикаши 280

**Й**

Йокояма, Казухиро 70

**К**

Кавамура, Шиничи 70, 229  
 Казад-Дум 282  
 Калиски, Бертон 386  
 Канда, Масаюки 70, 123  
 Канеко, Тошинобу 280  
 Квадрат 11, 92, 103, 219, 469  
 Квадратичное

    хэширование 455

Кван, Мэтью 254, 295

Квартет 38

Келлер, Натан 93, 281

Келси, Джон 63, 160, 312,  
     381, 402, 468, 476, 480, 483

Килиан, Джо 184

Ким, Кванджо 190

Класс ключей 37, 59, 63, 80,  
     147, 160, 200, 269, 271, 287,  
     386, 450, 477

Ключ 8

    возможно слабый 169, 200

комплémentарный 170,  
     171, 185  
 открытый 3, 5  
 полуслабый 191  
 расшифрования 7  
 раунда 9  
 секретный 3, 5, 18  
 слабый 37, 59, 62, 63, 80,  
     92, 122, 147, 169, 189,  
     191, 200, 218, 254, 269,  
     271, 287, 312, 386, 450  
 шифрования 9  
 эквивалентный 18, 59, 62,  
     160, 169, 185, 191, 253,  
     477, 481  
 Кнудсен, Ларс 42, 58, 61, 63,  
     102, 114, 132, 156, 174, 179,  
     190, 200, 348, 357, 381, 386,  
     402, 404, 432  
 Коллизия 294, 402, 404, 449  
 Конкурс:  
     AES 22, 27, 40, 61, 70, 93,  
     114, 122, 136, 141, 171,  
     195, 206, 212, 219, 247,  
     296, 316, 319, 382, 391,  
     408, 433, 469, 492  
     NESSIE 38, 69, 103, 123,  
     134, 149, 207, 219, 229,  
     270, 282, 287, 345, 351,  
     352, 357, 374, 418, 432,  
     442  
 Контрольная сумма 3, 24, 53  
 Копперсмит, Дон 200, 319  
 Кохер, Пол 45, 67, 269, 387  
 Криптоалгоритмы 3  
     бесключевые 3  
     двухключевые 3  
     одноключевые 3  
 Криптоанализ:  
     дифференциально-  
         линейный 66  
     дифференциальный 25, 27,  
         37, 43, 52, 63, 79, 92, 122,  
         132, 135, 155, 168, 171,  
         185, 186, 188, 190, 193,  
         195, 200, 254, 259, 265,  
         268, 271, 280, 287, 294,  
         312, 316, 322, 347, 357,  
         375, 381, 384, 386, 388,  
         433, 449, 461, 468, 473  
     дифференциальный на  
         основе сбоев 50, 287

# Предметный указатель

- дифференциальный**  
на связанных ключах 81, 281, 476
- дифференциальный**  
по потребляемой мощности 47, 94, 415
- интегральный** 287, 348
- линейный** 34, 63, 79, 92, 132, 155, 168, 185, 186, 190, 193, 200, 254, 271, 287, 296, 347, 357, 364, 375, 381, 384, 386, 388, 408, 433, 449, 473  
на связанных ключах 168
- Криптограмма** 7
- Криптостойкость** 18, 21, 22
- Критерий корректности ключа** 23
- Кроули, Пол 328
- Куреян, Мельсик 70, 408
- Кушилевиц, Эйял 386
- Кюн, Ульрих 280, 347
- Л**
- Лавинный эффект 164
- Лай, Кседжя 70, 265
- Лакс, Стефан 114, 160
- ЛАН Крипто 70, 357
- Лебедев, Анатолий Николаевич 70, 357
- Леру, Дамиан 212
- Лим, Чэ Хун 141
- Липтон, Ричард 50
- Локи 295
- Лоу, Йи Вей 281
- Любы, Майкл 109
- М**
- МакБрайд, Лесли 70, 374
- Маховенко, Елена Борисовна 81
- Мацун, Мицуру 34, 37, 70, 123, 168, 334
- Мачадо, Алексис Уорнер 70, 348
- МВД Сингапура 403
- Мейер, Уилли 386
- Меркль, Ральф 27, 172, 288
- Мерфи, Шон 211
- Мирза, Фаузан 477
- Мияучи, Шоджи 206
- Модификация алгоритма:**  
изменение структуры раунда 22  
усечение количества раундов 22
- Мориаи, Шихо 70, 123
- Морин, Пэт 82
- Мотояма, Масахико 70, 229
- Муратани, Хирофуми 70, 229
- Мэсси, Джеймс 70, 265, 394, 403, 404, 408
- Н**
- Накаджима, Джунко 70, 123
- Насаш, Давид 70, 442
- Нидэм, Роджер 474, 481
- О**
- Обмен ключами 57
- Октет 289
- Отбеливание 57, 184, 332, 375, 486
- Открытый текст 7
- Охкума, Кенджи 70, 229
- П**
- Парадокс дней рождения 42, 56
- Пател, Сарвар 454
- Петерс, Михаэль 70, 351
- Пирэ, Жиль-Франсуа 259
- ПЛИС 24, 65, 259
- Пренел, Барт 316
- Пьепржик, Йозеф 63, 295
- Р**
- Раддум, Хавард 357
- Разность 29
- Ракофф, Чарльз 109
- Рамзан, Зульфикар 454
- Раскрытие:  
полное 18  
частичное 19
- Рассел, Мэтью 477, 479
- Расширение ключа 9, 53, 186  
ТЕМК 175  
на лету 54, 67, 148, 168, 200, 270
- Редди Андем, Викрам 477, 481
- Режим работы** 12, 62, 75, 113, 175, 480  
CBC 14, 76, 78, 175, 216, 251, 291, 481  
CBC|CBC|CBC 175  
CBCM 178  
CFB 15, 76, 175, 481  
ECB 13, 76, 175, 179  
ECB|ECB|OFB 176  
OFB 16, 76, 175, 481  
внешний 175  
внутренний 175  
выработка имитоприставки 78
- гаммирование** 76
- гаммирование с обратной связью** 77
- косвенный 352
- модифицированный OFB** 16
- простая замена 76
- прямой 352
- с маскированием 178
- счетчик 17
- Ривест, Рональд 70, 184, 333, 375, 381, 391
- Риджмен, Винсент 12, 63, 70, 102, 103, 142, 200, 219, 282, 316, 351, 381, 433, 451, 468, 469, 473
- Риттер, Терри 181
- Робшоу, Мэтью 70, 381, 391
- Рогауэй, Филипп 184
- Ростовцев, Александр Григорьевич 81
- РусКрипто 357
- С**
- Сааринен, Маркку-Юхани 481, 483
- Сано, Фумихико 70, 229
- Себерри, Джениффер 63, 295
- Сеть Фейстеля 9, 119, 134, 136, 156, 196, 201, 250, 254, 271, 296, 299, 317, 332, 334, 376, 384, 451, 459
- несбалансированная 110, 273, 312, 464
- обобщенная 11, 313
- расширенная 11, 12, 320
- сбалансированная 56, 110
- Си, язык 292, 315, 479

Сидни, Рэймонд 70, 391  
 Синхропосылка 76, 78  
 Смарт-карта 20, 42, 47, 49,  
     65, 94, 117, 122, 141, 148,  
     199, 206, 219, 253, 288, 300,  
     416, 442  
         CASCADE 48  
 Соримачи, Джун 334  
 Стерн, Жак 70, 148  
 Сугио, Нобуоки 280  
 Сундарам, Ганеш 454

**Т**

Табличная замена 75, 79  
 Такенака, Масахико 70  
 Танака, Хидема 70, 280  
 Технологический Институт  
     г. Цюрих 394  
 Токита, Тошио 70, 123, 334  
 Толкиен, Дж. 282  
 Тории, Наоя 70  
 Точка единственности 24

**У**

Уайтинг, Дуг 63, 483  
 Уилер, Дэвид 474, 481  
 Университет:  
     Carleton 82  
     Royal Holloway 70, 477  
 Беркли 38, 351, 483  
     г. Берген 70  
     г. Геттинген 114  
     г. Йокогама 201, 207  
     г. Любек 62, 70, 72, 219,  
         351, 451  
     г. Мадрид 477

г. Сеул 449, 477  
 г. Токио 424  
 Калифорнийский 295  
 Кембриджский 108, 188,  
     474  
 Манхейма 160  
     шт. Алабама 477  
     шт. Аризона 246

**Ф**

Фейстель, Хорст 10, 301  
 Фенг, Дэнг-Гуо 363, 373  
 Фергюсон, Нильс 63, 102, 483  
 Фларер, Скотт 334  
 Флеминг, Роджер 477  
 Фуке, Пьер-Алан 149  
 Фурман, Владимир 351

**Х**

Хандшух, Хелена 70, 387, 442  
 Характеристика 31  
     итеративная 32  
 Хачатрян, Гурген 70, 408  
 Хейз, Говард 387  
 Хеллман, Мартин 27, 172  
 Холл, Крис 63, 468, 483  
 Хэш-функция 3, 54, 57, 62, 70,  
     82, 109, 114, 160, 402, 404,  
     465, 477  
         MD5 110  
         SHA 82, 442, 454  
         SHA-0 446  
         SHA-1 110, 115, 185,  
             442, 454  
         SHA-256 446  
         Snefru 294

**Ч**

Чавес, Билли Симон 212

**Ш**

Шамир, Эди 27, 36, 43,  
     50, 94, 168, 186, 211,  
     294, 311  
 Шимизу, Акихиро 206  
 Шимояма, Такешি 70  
 Шифрование 7  
     асимметричное 8, 70, 288  
     блочное 4, 70  
     потоковое 5, 70, 114, 277  
     симметричное 4, 8, 70  
 Шифртекст 7  
 Шнайер, Брюс 14, 27, 36,  
     41, 63, 79, 102, 118, 160,  
     161, 174, 188, 194, 212, 268,  
     301, 312, 381, 402, 468, 476,  
     480, 483  
 Шреппель, Ричард 63, 246

**Э**

Экранирование 53  
 Электронная подпись 7, 70,  
     78, 288  
 Эскотт, Эдриан 280

**Я**

Яджима, Джун 70  
 Якобсен, Томас 42  
 Ямагиши, Ацуhiro 334  
 Янами, Хитоши 70