[ИНФОРМАЦИОННАЯ БЕЗОПАСТНОСТЬ]

[Институт ИИБС, Кафедра ИСКТ]

[Шумейко Е.В.]

АЛГОРИТМЫ СИММЕТРИЧНОГО ШИФРОВАНИЯ
Часть 2

Алгоритм Blowfish

Алгоритм Blowfish

Blowfish является сетью Фейштеля, у которой количество итераций равно 16. Длина блока равна 64 битам, ключ может иметь любую длину в пределах 448 бит. Хотя перед началом любого шифрования выполняется сложная фаза инициализации, само шифрование данных выполняется достаточно быстро.

Алгоритм предназначен в основном для приложений, в которых ключ меняется нечасто, к тому же существует фаза начального рукопожатия, во время которой происходит аутентификация сторон и согласование общих параметров и секретов. Классическим примером подобных приложений является сетевое взаимодействие. При реализации на 32-битных микропроцессорах с большим кэшем данных *Blowfish* значительно быстрее DES.

Алгоритм состоит из двух частей: расширение ключа и шифрование данных. Расширение ключа преобразует ключ длиной, по крайней мере, 448 бит в несколько массивов *подключей* общей длиной 4168 байт.

Алгоритм Blowfish

В основе алгоритма лежит сеть Фейштеля с 16 итерациями. Каждая итерация состоит из перестановки, зависящей от ключа, и подстановки, зависящей от ключа и данных. Операциями являются XOR и сложение 32-битных слов.
Blowfish использует большое количество *подключей*. Эти ключи должны быть вычислены заранее, до начала любого шифрование или дешифрования данных. Элементы алгоритма:

1. P - массив, состоящий из восемнадцати 32-битных *подключей*: P1, P2, ..., P18.
2. Четыре 32-битных *S-boxes* с 256 входами каждый. Первый индекс означает номер *S-box*, второй индекс - номер входа.
3. $S_{1,0}$, $S_{1,1}$, … $S_{1,255}$;
4. $S_{2,0}$, $S_{2,1}$, … $S_{2,255}$;
5. $S_{3,0}$, $S_{3,1}$, … $S_{3,255}$;
6. $S_{4,0}$, $S_{4,1}$, … $S_{4,255}$;

Метод, используемый для вычисления этих *подключей*, будет описан ниже.

Алгоритм Blowfish

Шифрование

Входом является 64-битный элемент данных X, который делится на две 32-битные половины, X_1 и Xr.

$X_1 = X_1$ XOR Pi

Xr = F (X_1) XOR Xr

Swap X_1 and Xr

Функция F

Разделить X_1 на четыре 8-битных элемента A, B, C, D.

$F(X_1) = ((S_{1,A} + S_{2,B} \bmod 2^{32}) \text{ XOR } S_{3,C}) + S_{4,D} \bmod 2^{32}$

Дешифрование отличается от шифрования тем, что Pi используются в обратном порядке.

Алгоритм Blowfish

Генерация подключей

Подключи вычисляются с использованием самого *алгоритма Blowfish*.

1. Инициализировать первый P-массив и четыре *S-boxes* фиксированной строкой.
2. Выполнить операцию XOR P1 с первыми 32 битами ключа, операцию XOR P2 со вторыми 32 битами ключа и т.д. Повторять цикл до тех пор, пока весь P-массив не будет побитово сложен со всеми битами ключа. Для коротких ключей выполняется конкатенация ключа с самим собой.
3. Зашифровать нулевую строку *алгоритмом Blowfish*, используя *подключи*, описанные в пунктах (1) и (2).
4. Заменить P1 и P2 выходом, полученным на шаге (3).
5. Зашифровать выход шага (3), используя *алгоритм Blowfish* с модифицированными *подключами*.
6. Заменить P3 и P4 выходом, полученным на шаге (5).
7. Продолжить процесс, заменяя все элементы P-массива, а затем все четыре *S-boxes*, выходами соответствующим образом модифицированного *алгоритма Blowfish*.

Для создания всех *подключей* требуется 521 итерация.

Алгоритм IDEA

IDEA (International Data Encryption Algorithm) является блочным симметричным алгоритмом шифрования, разработанным Сюдзя Лай (Xuejia Lai) и Джеймсом Массей (James Massey) из швейцарского федерального института технологий. Первоначальная версия была опубликована в 1990 году. Пересмотренная версия алгоритма, усиленная средствами защиты от дифференциальных криптографических атак, была представлена в 1991 году и подробно описана в 1992 году.

IDEA является одним из нескольких симметричных криптографических алгоритмов, которыми первоначально предполагалось заменить DES.

Алгоритм IDEA

Принципы разработки

IDEA является блочным алгоритмом, который использует 128-битовый *ключ для шифрования* данных блоками по 64 бита. Целью разработки *IDEA* было создание относительно стойкого криптографического алгоритма с достаточно простой реализацией.

Алгоритм IDEA

Криптографическая стойкость

Следующие характеристики *IDEA* характеризуют его криптографическую стойкость:

1. **Длина блока**: длина блока должна быть достаточной, чтобы скрыть все статистические характеристики исходного сообщения. С другой стороны, сложность реализации криптографической функции возрастает экспоненциально в соответствии с размером блока. Использование блока размером в 64 бита в 90-е годы означало достаточную силу. Более того, использование режима шифрования CBC говорит о дальнейшем усилении этого аспекта алгоритма.
2. **Длина ключа**: длина ключа должна быть достаточно большой для того, чтобы предотвратить возможность простого перебора ключа. При длине ключа 128 бит *IDEA* считается достаточно безопасным.

Алгоритм IDEA

3. **Конфузия**: зашифрованный текст должен зависеть от ключа сложным и запутанным способом.
4. **Диффузия**: каждый бит незашифрованного текста должен влиять на каждый бит зашифрованного текста. Распространение одного незашифрованного бита на большое количество зашифрованных битов скрывает статистическую структуру незашифрованного текста. Определить, как статистические характеристики зашифрованного текста зависят от статистических характеристик незашифрованного текста, должно быть непросто. *IDEA* с этой точки зрения является очень эффективным алгоритмом.

В *IDEA* два последних пункта выполняются с помощью трех операций. Это отличает его от DES, где все постороено на использовании операции XOR и маленьких нелинейных *S-boxes*.

Алгоритм IDEA

Каждая операция выполняется над двумя 16-битными входами и создает один 16-битный выход. Этими операциями являются:

1.Побитовое исключающее OR обозначаемое как ⊕

2.Сумма целых по модулю 216 (по модулю 65536), при этом входы и выходы трактуются как беззнаковые 16-битные целые. Эту операцию обозначим как +.

3.Умножение целых по модулю 216 + 1 (по модулю 65537), при этом входы и выходы трактуются как беззнаковые 16-битные целые, за исключением того, что блок из одних нулей трактуется как 216. Эту операцию обозначим как •

Алгоритм IDEA

Эти три операции являются несовместимыми в том смысле, что:

- Не существует пары из трех операций, удовлетворяющих дистрибутивному закону. Например
  $a \bullet (b + c) <> (a \bullet b) + (a \bullet c)$
- Не существует пары из трех операций, удовлетворяющих ассоциативному закону. Например
  $a + (b \oplus c) <> (a + b) \oplus c$

Использование комбинации из этих трех операций обеспечивает комплексную трансформацию входа, делая криптоанализ более трудным, чем в таком алгоритме как DES, основанном исключительно на функции XOR.

Алгоритм IDEA

Шифрование

Рассмотрим общую схему шифрования *IDEA*. Как и в любом алгоритме шифрования, здесь существует два входа: незашифрованный блок и ключ. В данном случае незашифрованный блок имеет длину 64 бита, ключ имеет длину 128 бит.

Алгоритм IDEA состоит из восьми *раундов*, за которыми следует заключительное преобразование. Алгоритм разделяет блок на четыре 16-битных подблока. Каждый *раунд* получает на входе четыре 16-битных подблока и создает четыре 16-битных выходных подблока. Заключительное преобразование также получает на входе четыре 16-битных подблока и создает четыре 16-битных подблока. Каждый *раунд* использует шесть 16-битных ключей, заключительное преобразование использует четыре *подключа*, т.е. всего в алгоритме используется 52 *подключа*.

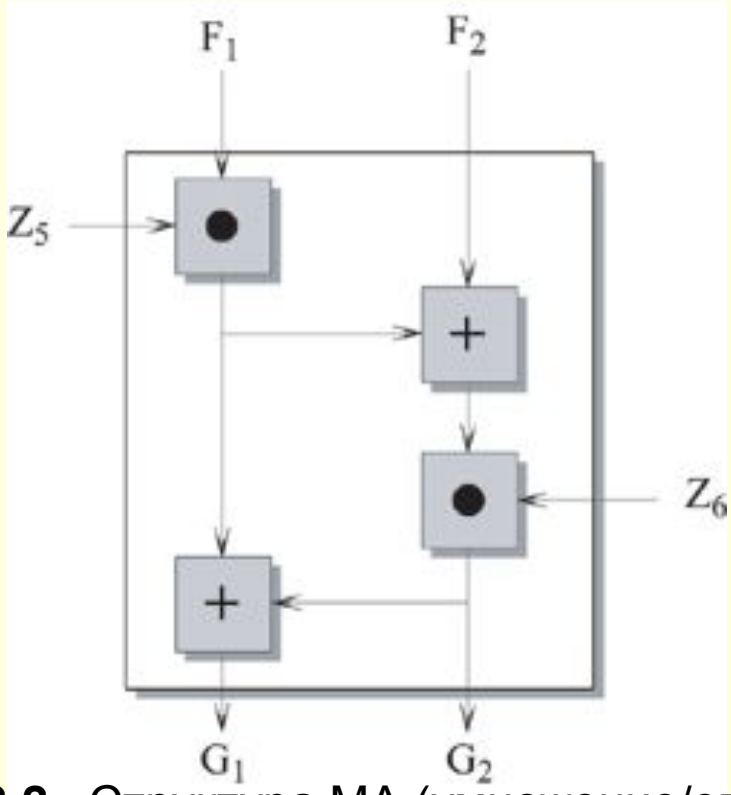
Алгоритм IDEA

Рис. 3.1. Алгоритм IDEA

Алгоритм IDEA

Последовательность преобразований отдельного раунда

Рассмотрим последовательность преобразований отдельного *раунда*. Одним из основных элементов алгоритма, обеспечивающих диффузию, является структура, называемая МА (умножение/сложение):



На вход этой структуре подаются два 16-битных значения и два 16-битных *подключа*, на выходе создаются два 16-битных значения. Исчерпывающая компьютерная проверка показывает, что каждый бит выхода этой структуры зависит от каждого бита входов незашифрованного блока и от каждого бита *подключей*. Данная структура повторяется в алгоритме восемь раз, обеспечивая высокоэффективную диффузию.

Рис. 3.2. Структура МА (умножение/сложение)

Алгоритм IDEA

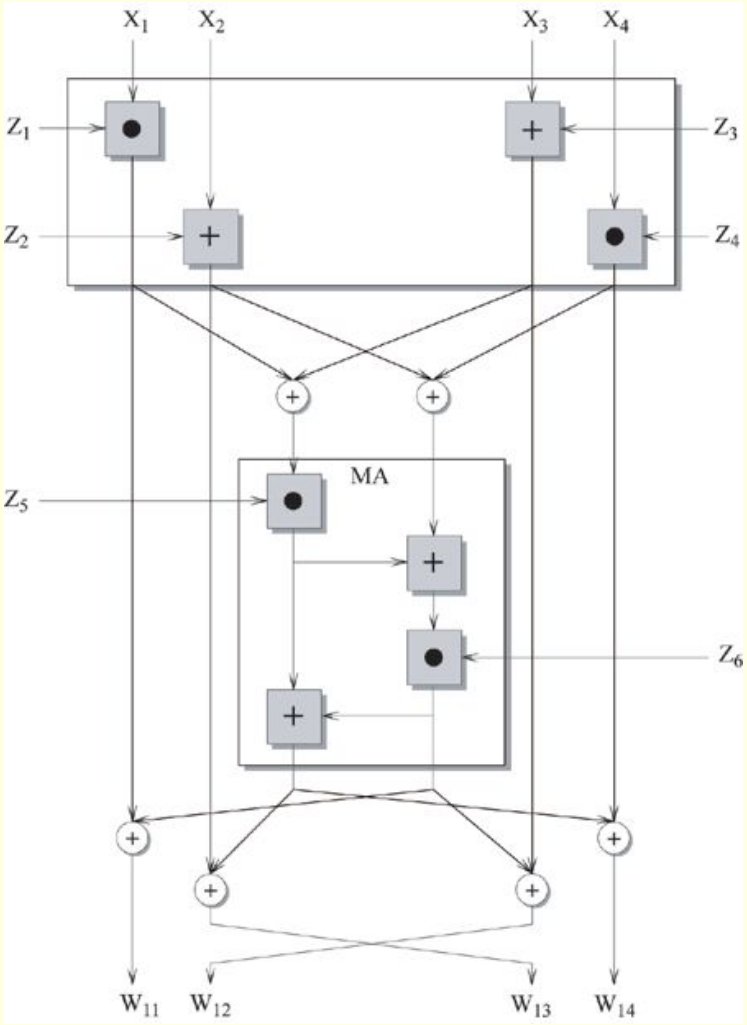


Рис. 3.3. I-ый раунд IDEA

Раунд начинается с преобразования, которое комбинирует четыре входных подблока с четырьмя *подключами*, используя операции сложения и умножения. Четыре выходных блока этого преобразования комбинируются, используя операцию XOR для формирования двух 16-битных блоков, которые являются входами MA структуры. Кроме того, MA структура имеет на входе еще два *подключа* и создает два 16-битных выхода.

Алгоритм IDEA

В заключении четыре выходных подблока первого преобразования комбинируются с двумя выходными подблоками MA структуры, используя XOR для создания четырех выходных подблоков данной итерации. Заметим, что два выхода, которые частично создаются вторым и третьим входами (X_2 и X_3), меняются местами для создания второго и третьего выходов (W_{12} и W_{13}). Это увеличивает перемешивание битов и делает алгоритм более стойким для дифференциального криптоанализа. Рассмотрим девятый *раунд алгоритма*, обозначенный как заключительное преобразование. Это та же структура, что была описана выше. Единственная разница состоит в том, что второй и третий входы меняются местами. Это сделано для того, чтобы дешифрование имело ту же структуру, что и шифрование. Заметим, что девятая стадия требует только четыре входных *подключа*, в то время как для первых восьми стадий для каждой из них необходимо шесть входных *подключей*.

Алгоритм IDEA

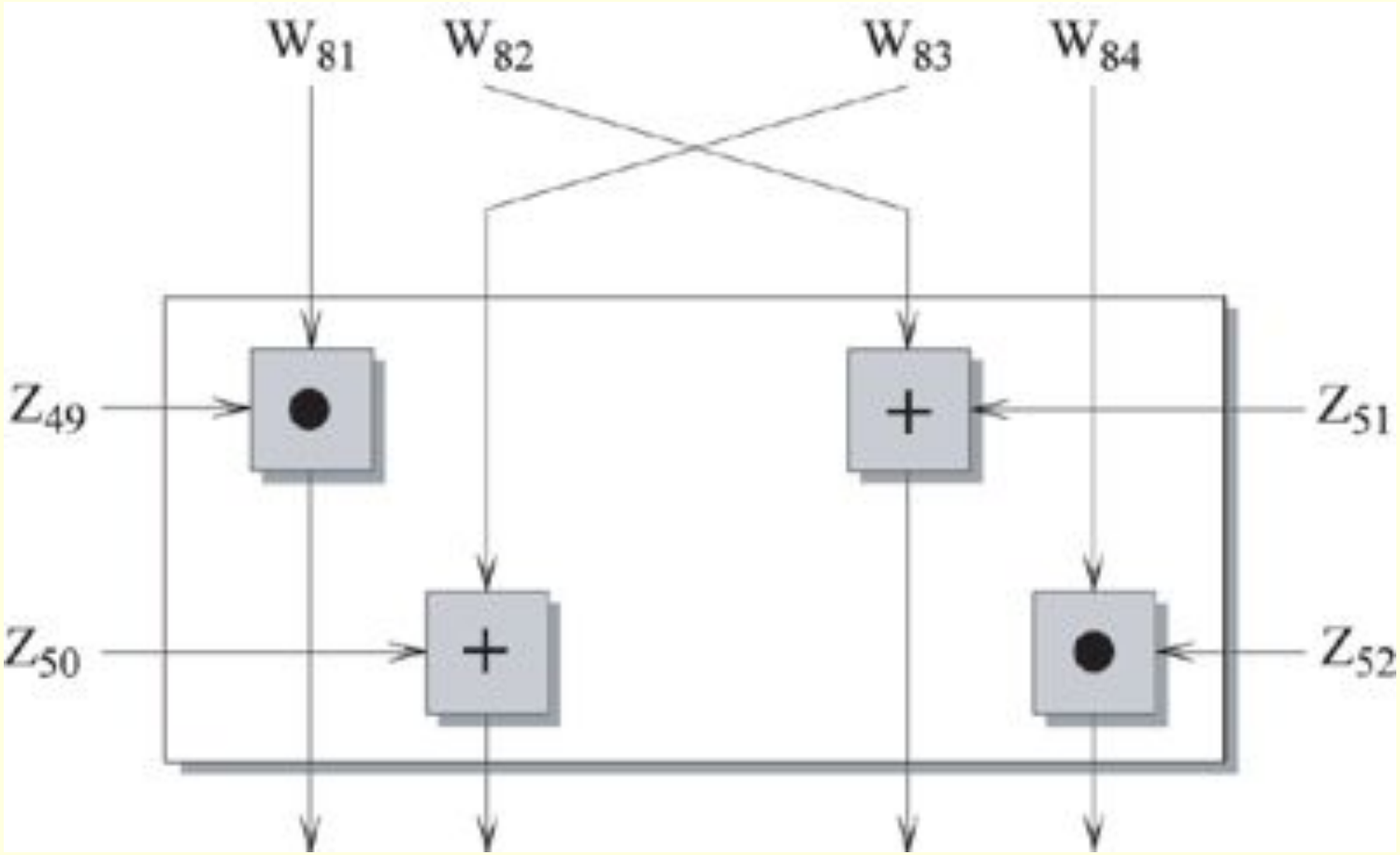


Рис. 3.4. Заключительное преобразование

Алгоритм IDEA

Создание подключей

Пятьдесят два 16-битных *подключа* создаются из 128-битного *ключа шифрования* следующим образом. Первые восемь *подключей*, которые обозначим как Z_1, Z_2, ..., Z_8, получаются непосредственно из ключа, при этом Z_1 равен первым 16 битам, Z_2 равен следующим 16 битам и т.д. Затем происходит циклический сдвиг ключа влево на 25 битов, и создаются следующие восемь *подключей*. Эта поцедура повторяется до тех пор, пока не будут созданы все 52 *подключа*.

$$Z_1 = Z_{[1..16]} \quad Z_{25} = Z_{[76..91]}$$
$$Z_7 = Z_{[97..112]} \quad Z_{31} = Z_{[44..59]}$$
$$Z_{13} = Z_{[90..105]} \quad Z_{37} = Z_{[37..52]}$$
$$Z_{19} = Z_{[83..98]} \quad Z_{43} = Z_{[30..45]}$$

Хотя на каждом *раунде* за исключением первого и восьмого используются только 96 битов *подключа*, множество битов ключа на каждой итерации не пересекаются, и не существует отношения простого сдвига между *подключами* разных *раундов*. Это происходит потому, что на каждом *раунде* используется только шесть *подключей*, в то время как при каждой ротации ключа получается восемь *подключей*.

Алгоритм IDEA

Дешифрование

Процесс дешифрования аналогичен процессу шифрования. Дешифрование состоит в использовании зашифрованного текста в качестве входа в ту же самую структуру *IDEA*, но с другим набором ключей. Дешифрующие ключи $U_1, \ldots, U_{52}$ получаются из шифрующих ключей следующим образом:

1. Первые четыре *подключа* i-ого *раунда* дешифрования получаются из первых четырех *подключей* (10 - i) -го*раунда* шифрования, где стадия заключительного преобразования считается 9-м *раундом*. Первый и четвертый ключи дешифрования эквивалентны мультипликативной инверсии по модулю $(2^{16} + 1)$ соответствующих первого и четвертого *подключей* шифрования. Для *раундов* со 2 по 8 второй и третий *подключи* дешифрования эквивалентны аддитивной инверсии по модулю (2^{16}) соответствующих третьего и второго *подключей* шифрования. Для *раундов* 1 и 9 второй и третий *подключи* дешифрования эквивалентны аддитивной инверсии по модулю (216) соответствующих второго и третьего *подключей* шифрования.

Алгоритм IDEA

Для первых восьми *раундов* последние два *подключа* i *раунда* дешифрования эквивалентны последним двум *подключам* (9 - i) *раунда* шифрования.

Для мультипликативной инверсии используется нотация Zj^{-1}, т.е.:

$$Z_j \cdot Z_j^{-1} = 1 \bmod (2^{16} + 1)$$

Так как $2^{16} + 1$ является простым числом, каждое ненулевое целое $Zi \leq 2^{16}$ имеет уникальную мультипликативную инверсию по модулю $(2^{16} + 1)$. Для аддитивной инверсии используется нотация (-Zj), таким образом, мы имеем: $-Zj + Zj = 0 \bmod (2^{16})$

Для доказательства того, что алгоритм дешифрования с соответствующими *подключами* имеет корректный результат, рассмотрим одновременно процессы шифрования и дешифрования. Каждый из восьми *раундов* разбит на две стадии преобразования, первая из которых называется трансформацией, а вторая шифрованием.

Алгоритм IDEA

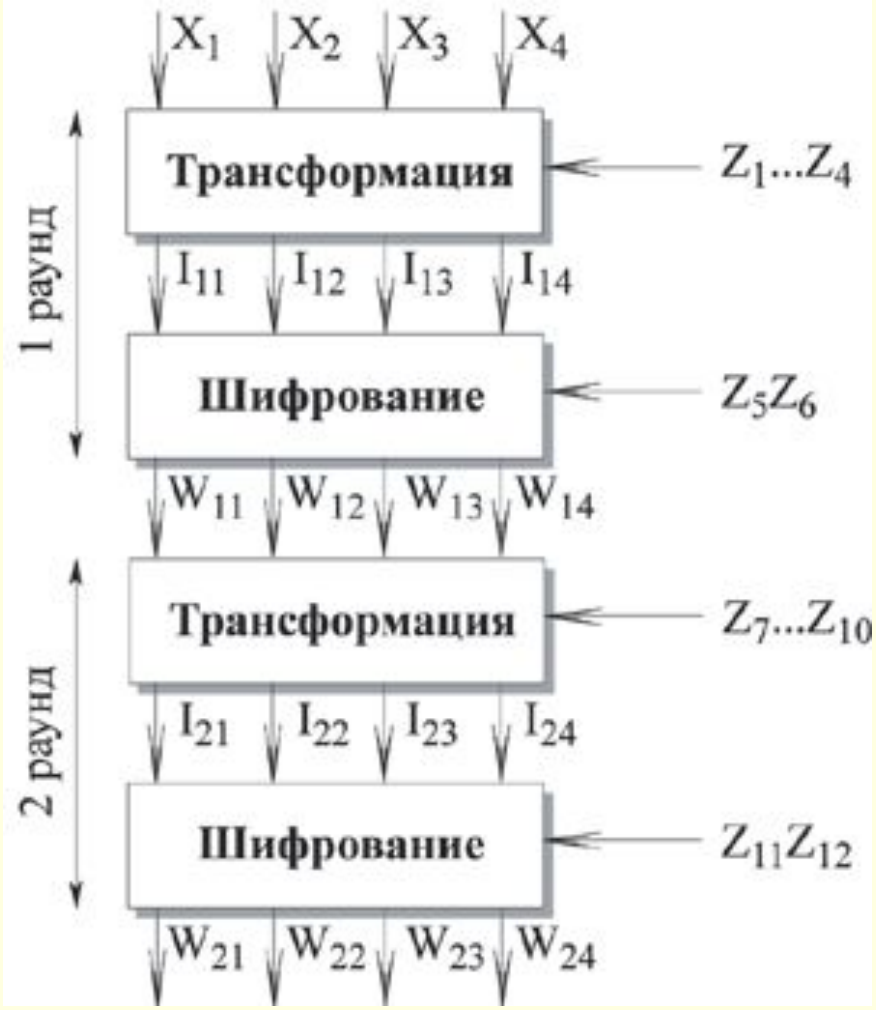


Рис. 3.5. Шифрование IDEA

Алгоритм IDEA

Рис. 3.6. Дешифрование IDEA

Алгоритм IDEA

Рассмотрим преобразования, выполняемые в прямоугольниках на обоих рисунках. При шифровании поддерживаются следующие соотношения на выходе трансформации:

$$Y_1 = W_{81} \cdot Z_{49} \quad Y_3 = W_{82} + Z_{51}$$
$$Y_2 = W_{83} + Z_{50} \quad Y_4 = W_{84} \cdot Z_{52}$$

Подставляя соответствующие значения, получаем:

$$J_{11} = Y_1 \cdot Z_{49}^{-1} = W_{81} \cdot Z_{49} \cdot Z_{49}^{-1} = W_{81}$$
$$J_{12} = Y_2 + -Z_{50} = W_{83} + Z_{50} = W_{83} + Z_{50} + -Z_{50} = W_{83}$$
$$J_{13} = Y_3 + -Z_{51} = W_{82} + Z_{51} + -Z_{51} = W_{82}$$
$$J_{14} = Y_4 \cdot Z_{52}^{-1} = W_{84} \cdot Z_{52} \cdot Z_{52}^{-1} = W_{84}$$

Алгоритм IDEA

Таким образом, выход первой стадии процесса дешифрования эквивалентен входу последней стадии процесса шифрования за исключением чередования второго и третьего блоков. Теперь рассмотрим следующие отношения:

$$W_{81} = I_{81} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

$$W_{82} = I_{83} \oplus MA_R(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

$$W_{83} = I_{82} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

$$W_{84} = I_{84} \oplus MA_L(I_{81} \oplus I_{83}, I_{82} \oplus I_{84})$$

Алгоритм IDEA

Где MAR(X, Y) есть правый выход MA структуры с входами X и Y , и $MA_L(X, Y)$ есть левый выход MA структуры с входами X и Y. Теперь получаем

$$V_{11} = J_{11} \oplus MA_R\,(J_{11} \oplus J_{13},\ J_{12} \oplus J_{14}) =$$
$$W_{81} \oplus MA_R(W_{81} \oplus W_{82},\ W_{83} \oplus W_{84}) =$$
$$I_{81} \oplus MA_R(I_{81} \oplus I_{83},\ I_{82} \oplus I_{84}) \oplus$$
$$MA_R[\ I_{81} \oplus MA_R(I_{81} \oplus I_{83},\ I_{82} \oplus I_{84}) \oplus I_{83} \oplus$$
$$MA_R(I_{81} \oplus I_{83},\ I_{82} \oplus I_{84}),\ I_{82} \oplus$$
$$MA_L(I_{81} \oplus I_{83},\ I_{82} \oplus I_{84}) \oplus I_{84} \oplus$$
$$MA_L(I_{81} \oplus I_{83},\ I_{82} \oplus I_{84})\ ] =$$
$$I_{81} \oplus MA_R(I_{81} \oplus I_{83},\ I_{82} \oplus I_{84}) \oplus$$
$$MA_R(I_{81} \oplus I_{83},\ I_{82}\quad I_{84}) =$$
$$I_{81}$$

Алгоритм IDEA

Аналогично мы имеем

$$V_{12} = I_{83}$$
$$V_{13} = I_{82}$$
$$V_{14} = I_{84}$$

Таким образом, выход второй стадии процесса дешифрования эквивалентен входу предпоследней стадии процесса шифрования за исключением чередования второго и третьего подблоков. Аналогично можно показать, что

$$V_{81} = I_{11}$$
$$V_{82} = I_{13}$$
$$V_{83} = I_{12}$$
$$V_{84} = I_{14}$$

Наконец, так как выход трансформации процесса дешифрования эквивалентен первой стадии процесса шифрования за исключением чередования второго и третьего подблоков, получается, что выход всего процесса шифрования эквивалентен входу процесса шифрования.

Алгоритм ГОСТ 28147

Алгоритм ГОСТ 28147 является отечественным стандартом для алгоритмов симметричного шифрования. ***ГОСТ 28147*** разработан в 1989 году, является блочным алгоритмом шифрования, длина блока равна 64 битам, длина ключа равна 256 битам, количество *раундов* равно 32. Алгоритм представляет собой классическую сеть Фейштеля.

$$L_i = R_i$$

$$R_i = L_i \quad f\ (R_{i-1},\ K_i)$$

Функция F проста. Сначала правая половина и i-ый *подключ* складываются по модулю 2^{32}. Затем результат разбивается на восемь 4-битовых значений, каждое из которых подается на вход *S-box*. *ГОСТ 28147* использует восемь различных *S-boxes*, каждый из которых имеет 4-битовый вход и 4-битовый выход. Выходы всех *S-boxes* объединяются в 32-битное слово, которое затем циклически сдвигается на 11 битов влево. Наконец, с помощью XOR результат объединяется с левой половиной, в результате чего получается новая правая половина.

Алгоритм ГОСТ 28147

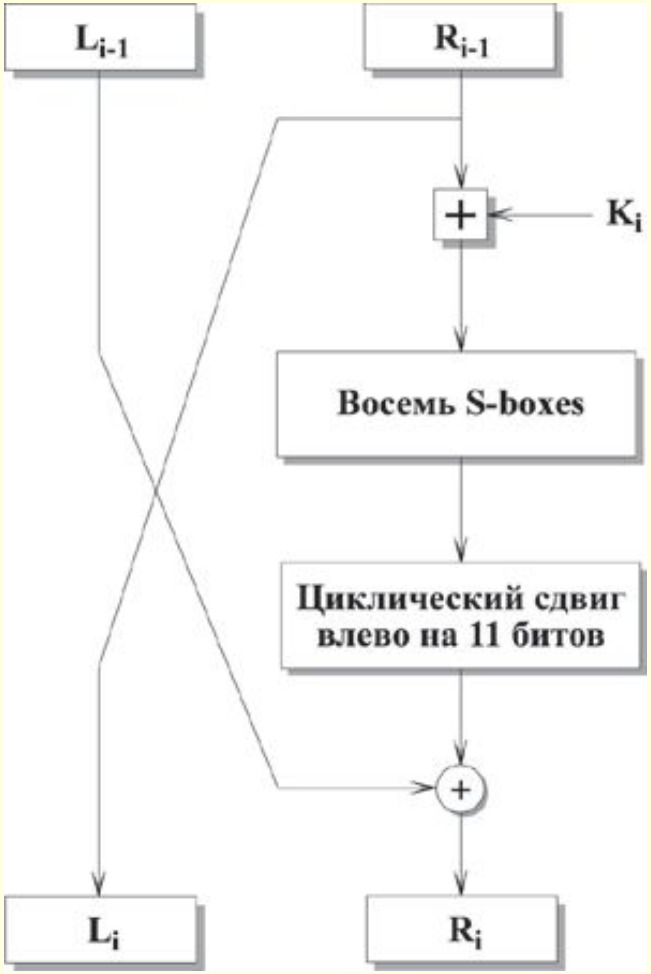


Рис. 3.7. I-ый раунд ГОСТ 28147

Алгоритм ГОСТ 28147

Генерация ключей проста. 256-битный ключ разбивается на восемь 32-битных *подключей*. Алгоритм имеет 32 *раунда*, поэтому каждый *подключ* используется в четырех *раундах* по следующей схеме:

Раунд	1	2	3	4	5	6	7	8
Подключ	1	2	3	4	5	6	7	8
Раунд	9	10	11	12	13	14	15	16
Подключ	1	2	3	4	5	6	7	8
Раунд	17	18	19	20	21	22	23	24
Подключ	1	2	3	4	5	6	7	8
Раунд	25	26	27	28	29	30	31	32
Подключ	8	7	6	5	4	3	2	1

Алгоритм ГОСТ 28147

1-ый S-box	4	10	9	2	13	8	0	14
	6	11	1	12	7	15	5	3
2-ой S-box	14	11	4	12	6	13	15	10
	2	3	8	1	0	7	5	9
3-ий S-box	5	8	1	13	10	3	4	2
	14	15	12	7	6	0	9	11
4-ый S-box	7	13	10	1	0	8	9	15
	14	4	6	12	11	2	5	3
5-ый S-box	6	12	7	1	5	15	13	8
	4	10	9	14	0	3	11	2
6-ой S-box	4	11	10	0	7	2	1	13
	3	6	8	5	9	12	15	14
7-ой S-box	13	11	4	1	3	15	5	9
	0	10	14	7	6	8	2	12
8-ой S-box	1	15	13	0	5	7	10	4
	9	2	3	14	6	11	8	12

Считается, что стойкость *алгоритма ГОСТ 28147* во многом определяется структурой *S-boxes*. Долгое время структура *S-boxes* в открытой печати не публиковалась. В настоящее время известны *S-boxes*, которые используются в приложениях Центрального Банка РФ и считаются достаточно сильными. Напомню, что входом и выходом *S-box* являются 4-битные числа, поэтому каждый *S-box* может быть представлен в виде строки цифр от 0 до 15, расположенных в некотором порядке. Тогда порядковый номер цифры будет являться входным значением *S-box*, а сама цифра - выходным значением *S-box*.

Алгоритм ГОСТ 28147

Основные различия между DES и *ГОСТ 28147* следующие:

- DES использует гораздо более сложную процедуру создания *подключей*, чем *ГОСТ 28147*. В *ГОСТ* эта процедура очень проста.
- В DES применяется 56-битный ключ, а в *ГОСТ 28147* - 256-битный. При выборе сильных *S-boxes ГОСТ 28147* считается очень стойким.
- У *S-boxes* DES 6-битовые входы и 4-битовые выходы, а у *S-boxes ГОСТ 28147* 4-битовые входы и выходы. В обоих алгоритмах используется по восемь *S-boxes*, но размер *S-box ГОСТ 28147* существенно меньше размера *S-box* DES.
- В DES применяются нерегулярные перестановки P, в *ГОСТ 28147* используется 11-битный циклический сдвиг влево. Перестановка DES увеличивает лавинный эффект. В *ГОСТ 28147* изменение одного входного бита влияет на один *S-box* одного *раунда*, который затем влияет на два *S-boxes* следующего *раунда*, три *S-boxes* следующего и т.д. В *ГОСТ 28147* требуется 8 *раундов* прежде, чем изменение одного входного бита повлияет на каждый бит результата; DES для этого нужно только 5 *раундов*.
- В DES 16 *раундов*, в *ГОСТ 28147* - 32 *раунда*, что делает его более стойким к дифференциальному и линейному криптоанализу.

Алгоритм ГОСТ 28147

Режимы выполнения алгоритмов симметричного шифрования
Для любого симметричного блочного алгоритма шифрования определено четыре режима выполнения.
ECB - Electronic Codebook - каждый блок из 64 битов незашифрованного текста шифруется независимо от остальных блоков, с применением одного и того же *ключа шифрования*. Типичные приложения - безопасная передача одиночных значений (например, криптографического ключа).
CBC - Chipher Block Chaining - вход криптографического алгоритма является результатом применения операции XOR к следующему блоку незашифрованного текста и предыдущему блоку зашифрованного текста. Типичные приложения - общая блокоориентированная передача, аутентификация.

Алгоритм ГОСТ 28147

CFB - Chipher Feedback - при каждом вызове алгоритма обрабатывается J битов входного значения. Предшествующий зашифрованный блок используется в качестве входа в алгоритм; к J битам выхода алгоритма и следующему незашифрованному блоку из J битов применяется операция XOR, результатом которой является следующий зашифрованный блок из J битов. Типичные приложения - потокоориентированная передача, аутентификация.

OFB - Output Feedback - аналогичен *CFB*, за исключением того, что на вход алгоритма при шифровании следующего блока подается результат шифрования предыдущего блока; только после этого выполняется операция XOR с очередными J битами незашифрованного текста. Типичные приложения - потокоориентированная передача по зашумленному каналу (например, спутниковая связь).

Алгоритм ГОСТ 28147

Режим ECB

Данный режим является самым простым режимом, при котором незашифрованный текст обрабатывается последовательно, блок за блоком. Каждый блок шифруется, используя один и тот же ключ. Если сообщение длиннее, чем длина блока соответствующего алгоритма, то оно разбивается на блоки соответствующей длины, причем последний блок дополняется в случае необходимости фиксированными значениями. При использовании данного режима одинаковые незашифрованные блоки будут преобразованы в одинаковые зашифрованные блоки.

ECB-режим идеален для небольшого количества данных, например, для шифрования ключа сессии.

Существенным недостатком *ECB* является то, что один и тот же блок незашифрованного текста, появляющийся более одного раза в сообщении, всегда имеет один и тот же зашифрованный вид. Вследствие этого для больших сообщений *ECB* режим считается небезопасным. Если сообщение имеет много одинаковых блоков, то при криптоанализе данная закономерность будет обнаружена.

Алгоритм ГОСТ 28147

Режим CBC

Для преодоления недостатков *ECB* используют способ, при котором одинаковые незашифрованные блоки преобразуются в различные зашифрованные. Для этого в качестве входа алгоритма используется результат применения операции XOR к текущему незашифрованному блоку и предыдущему зашифрованному блоку.

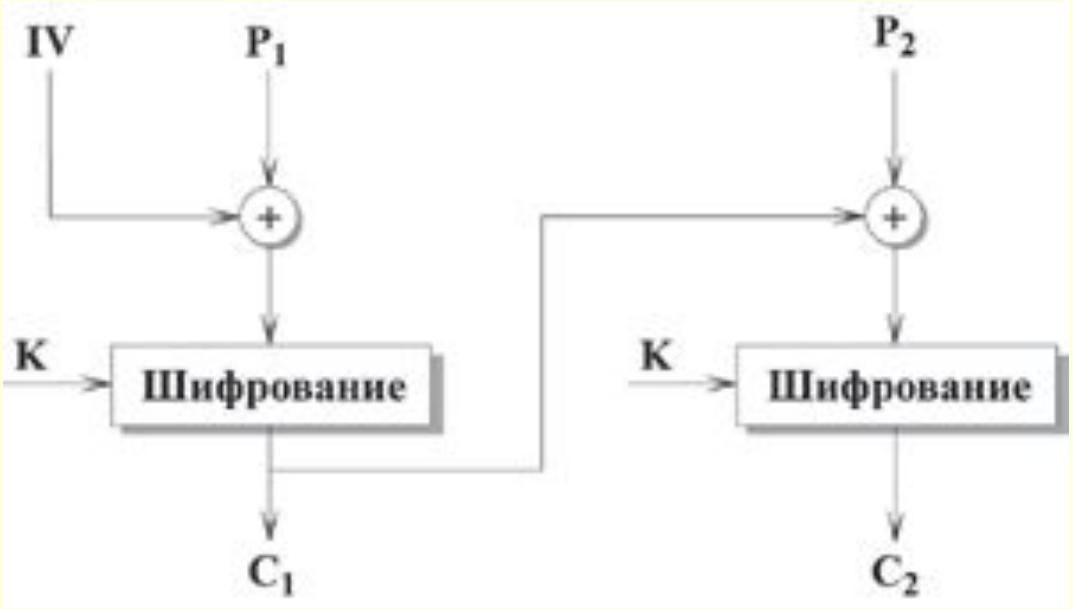


Рис. 3.8. Шифрование в режиме CBC

Алгоритм ГОСТ 28147

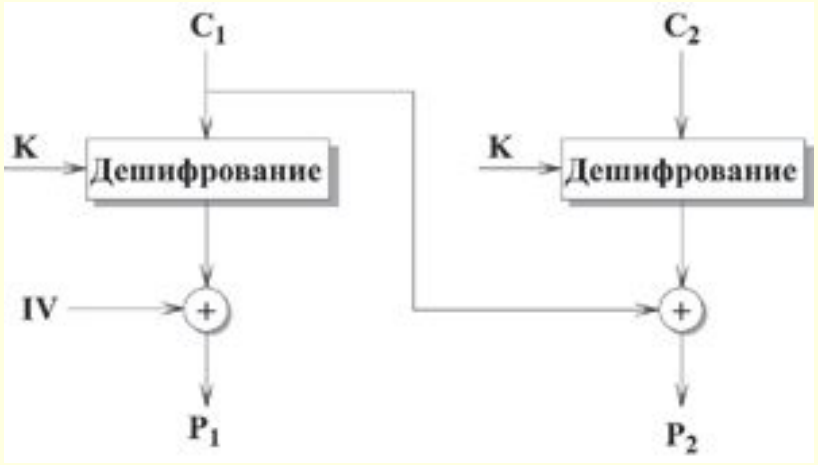


Рис. 3.9. Дешифрование в режиме CBC

Для получения первого блока зашифрованного сообщения используется инициализационный вектор (IV), для которого выполняется операция XOR с первым блоком незашифрованного сообщения. При дешифровании для IV выполняется операция XOR с выходом дешифрирующего алгоритма для получения первого блока незашифрованного текста.
IV должен быть известен как отправителю, так и получателю. Для максимальной безопасности IV должен быть защищен так же, как ключ.

Алгоритм ГОСТ 28147

Режим CFB

Блочный алгоритм предназначен для шифрования блоков определенной длины. Однако можно преобразовать блочный алгоритм в поточный алгоритм шифрования, используя последние два режима. Поточный алгоритм шифрования устраняет необходимость разбивать сообщение на целое число блоков достаточно большой длины, следовательно, он может работать в реальном времени. Таким образом, если передается поток символов, каждый символ может шифроваться и передаваться сразу, с использованием символьно ориентированного режима блочного алгоритма шифрования.

Одним из преимуществ такого режима блочного алгоритма шифрования является то, что зашифрованный текст будет той же длины, что и исходный.

Будем считать, что блок данных, используемый для передачи, состоит из J бит; обычным значением является J=8. Как и в режиме *CBC*, здесь используется операция XOR для предыдущего блока зашифрованного текста и следующего блока незашифрованного текста. Таким образом, любой блок зашифрованного текста является функцией от всего предыдущего незашифрованного текста.

Алгоритм ГОСТ 28147

Рассмотрим шифрование. Входом функции шифрования является регистр сдвига, который первоначально устанавливается в инициализационный вектор IV. Для левых J битов выхода алгоритма выполняется операция XOR с первыми J битами незашифрованного текста P1 для получения первого блока зашифрованного текста C1. Кроме того, содержимое регистра сдвигается влево на J битов, и C1 помещается в правые J битов этого регистра. Этот процесс продолжается до тех пор, пока не будет зашифровано все сообщение.

При дешифровании используется аналогичная схема, за исключением того, что для блока получаемого зашифрованного текста выполняется операция XOR с выходом алгоритма для получения незашифрованного блока.

Алгоритм ГОСТ 28147

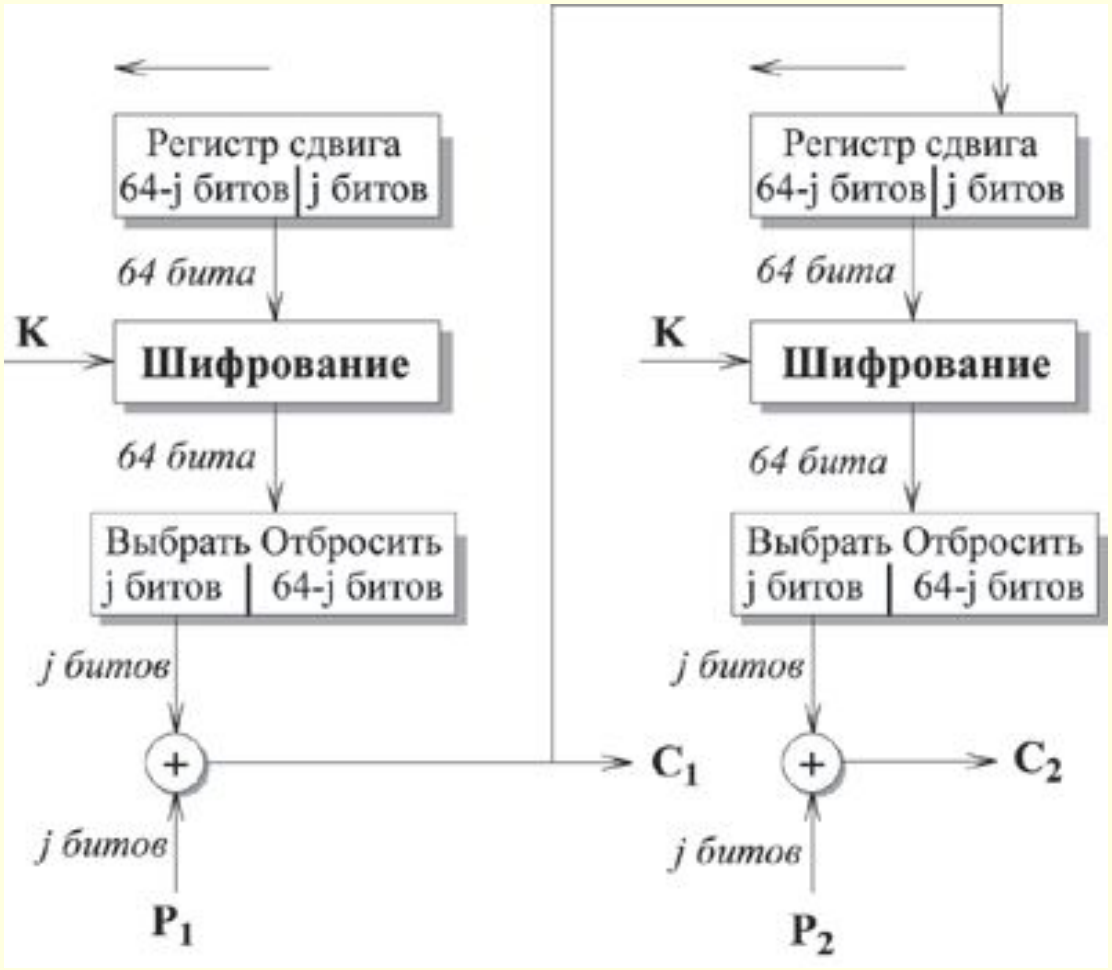


Рис. 3.10. Шифрование в режиме CFB

Алгоритм ГОСТ 28147

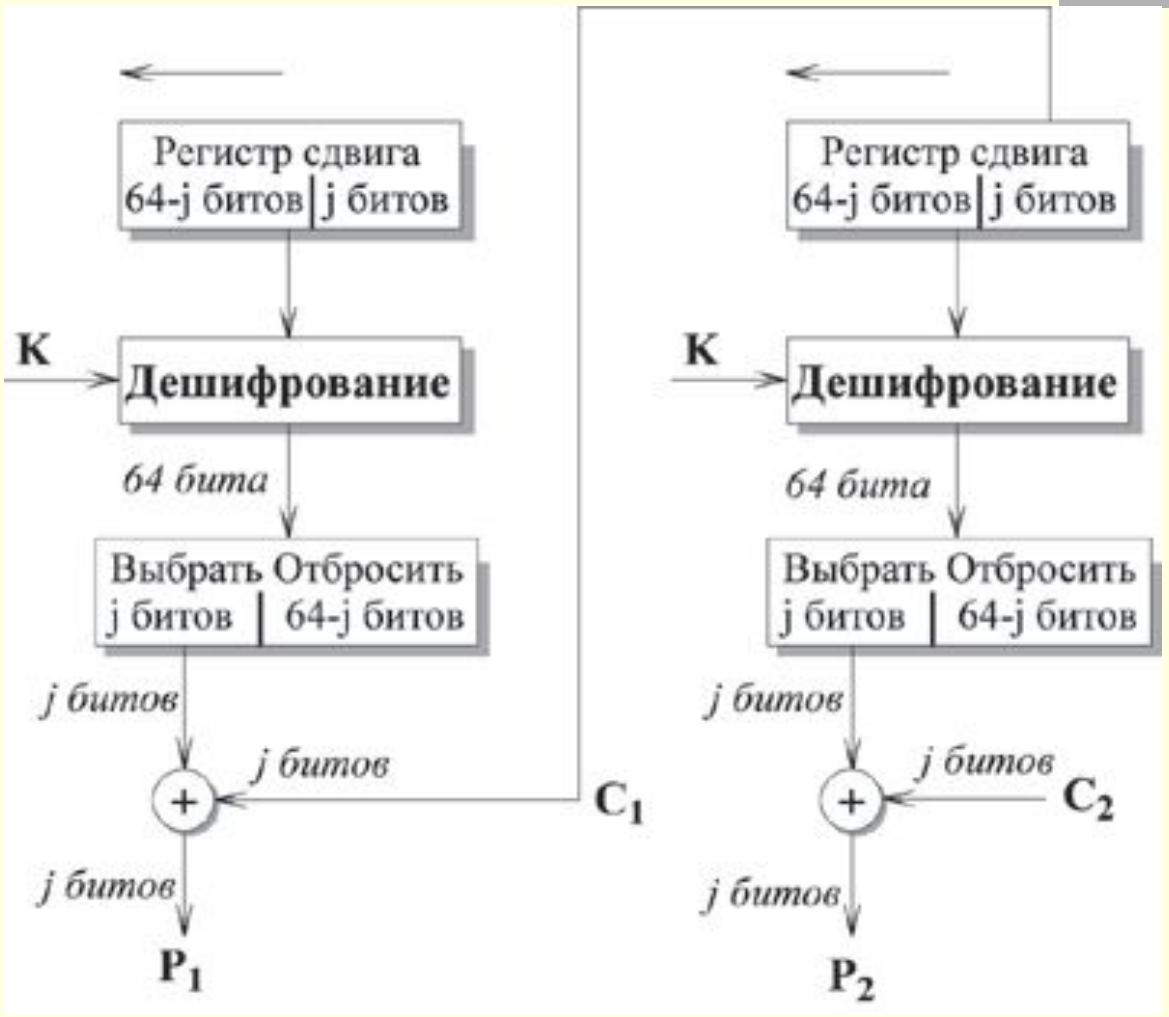


Рис. 3.11. Дешифрование в режиме CFB

Алгоритм ГОСТ 28147

Режим OFB

Данный режим подобен режиму *CFB*. Разница заключается в том, что выход алгоритма в режиме *OFB* подается обратно в регистр, тогда как в режиме *CFB* в регистр подается результат применения операции XOR к незашифрованному блоку и результату алгоритма.

Основное преимущество режима *OFB* состоит в том, что если при передаче произошла ошибка, то она не распространяется на следующие зашифрованные блоки, и тем самым сохраняется возможность дешифрования последующих блоков. Например, если появляется ошибочный бит в Ci, то это приведет только к невозможности дешифрования этого блока и получения Pi. Дальнейшая последовательность блоков будет расшифрована корректно. При использовании режима *CFB* Ci подается в качестве входа в регистр и, следовательно, является причиной последующего искажения потока. Недостаток *OFB* в том, что он более уязвим к атакам модификации потока сообщений, чем *CFB*.

Алгоритм ГОСТ 28147

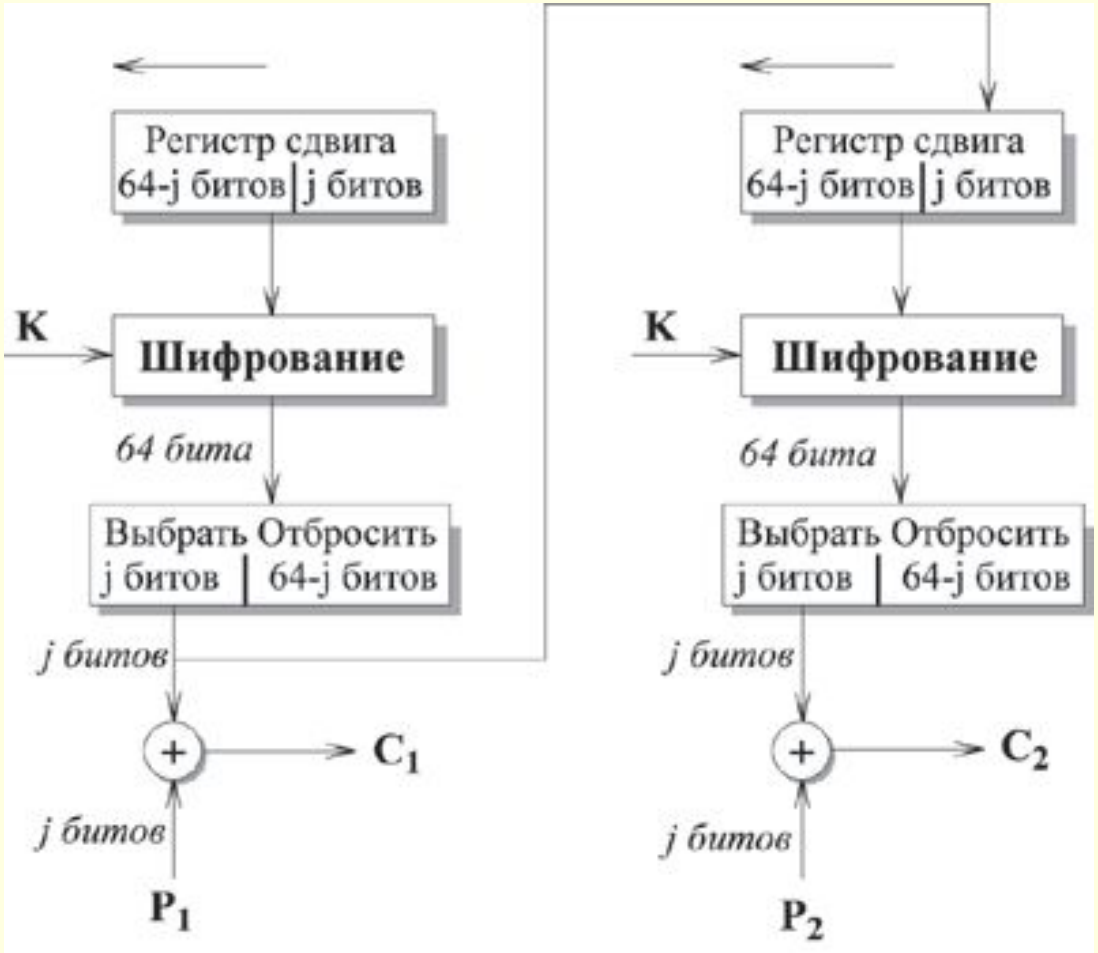


Рис. 3.12. Шифрование в режиме OFB

Алгоритм ГОСТ 28147

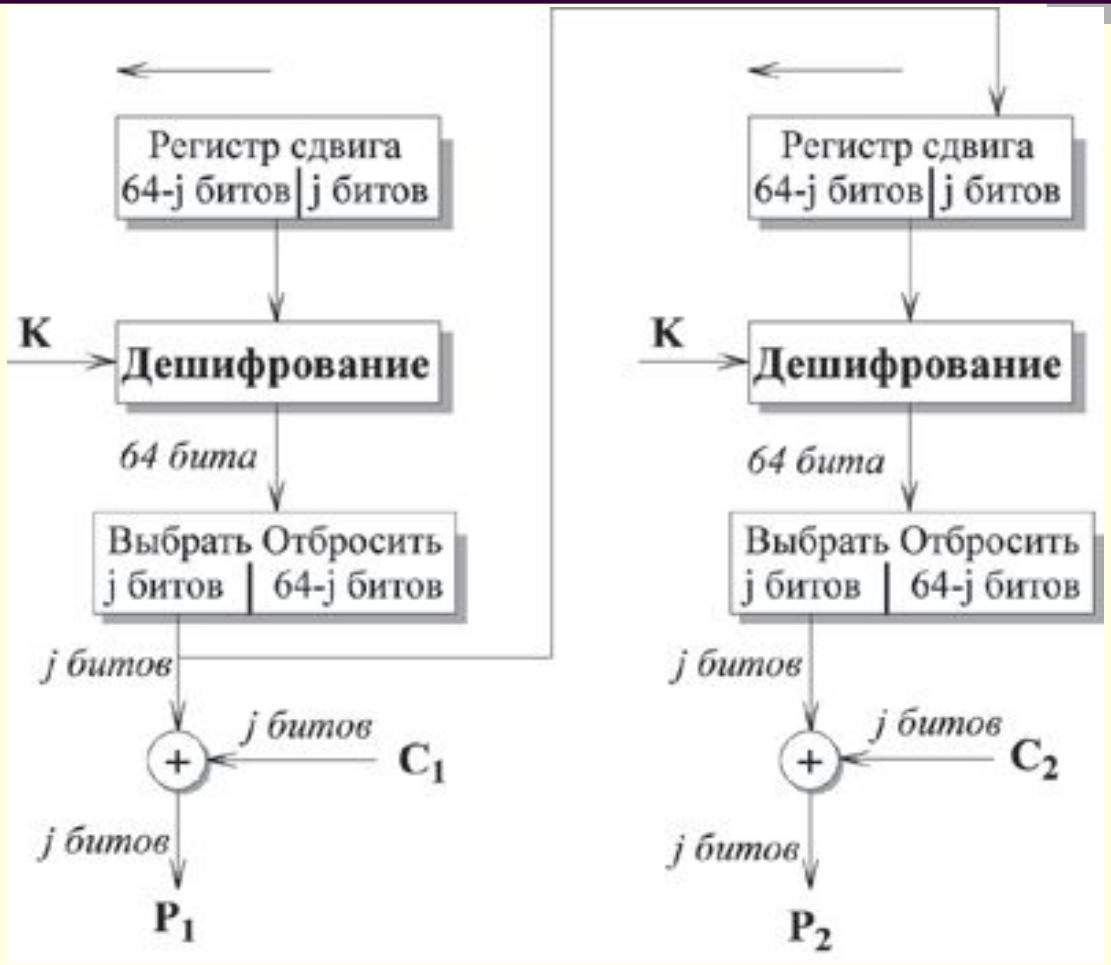


Рис. 3.13. Дешифрование в режиме OFB

Создание случайных чисел

Создание случайных чисел

Случайные числа играют важную роль при использовании криптографии в различных сетевых приложениях, относящихся к безопасности. Сделаем краткий обзор требований, предъявляемых к случайным числам в приложениях сетевой безопасности, а затем рассмотрим несколько способов создания случайных чисел.

Требования к случайным числам

Большинство алгоритмов сетевой безопасности, основанных на криптографии, использует случайные числа. Например:

1. Схемы взаимной аутентификации. В большинстве сценариев аутентификации и распределения ключа используются nonces для предотвращения атак повтора (replay-атак). Применение действительно случайных чисел в качестве nonces не дает противнику возможности вычислить или угадать nonce.
2. Ключ сессии, созданный KDC или кем-либо из участников.

Двумя основными требованиями к последовательности случайных чисел являются случайность и непредсказуемость.

Создание случайных чисел

Случайность

Обычно при создании последовательности *псевдослучайных чисел* предполагается, что данная последовательность чисел должна быть случайной в некотором определенном статистическом смысле. Следующие два критерия используются для доказательства того, что последовательность чисел является случайной:

1. Однородное распределение: распределение чисел в последовательности должно быть однородным; это означает, что частота появления каждого числа должна быть приблизительно одинаковой.
2. Независимость: ни одно значение в последовательности не должно зависеть от других.

Хотя существуют тесты, показывающие, что последовательность чисел соответствует некоторому распределению, такому как однородное распределение, теста для "доказательства" независимости нет. Тем не менее, можно подобрать набор тестов для доказательства того, что последовательность является зависимой. Общая стратегия предполагает применение набора таких тестов до тех пор, пока не будет уверенности, что независимость существует

Создание случайных чисел

Непредсказуемость

В приложениях, таких как взаимная аутентификация и генерация ключа сессии, нет жесткого требования, чтобы последовательность чисел была статистически случайной, но члены последовательности должны быть непредсказуемы. При "правильной" случайной последовательности каждое число статистически не зависит от остальных чисел и, следовательно, непредсказуемо. Однако правильные случайные числа на практике используются достаточно редко, чаще последовательность чисел, которая должна быть случайной, создается некоторым алгоритмом. В данном случае необходимо, чтобы противник не мог предугадать следующие элементы последовательности, основываясь на знании предыдущих элементов и используемого алгоритма.

Создание случайных чисел

Источники случайных чисел

Источники действительно случайных чисел найти трудно. Физические генераторы шумов, такие как детекторы событий ионизирующей радиации, газовые разрядные трубки и имеющий течь конденсатор могут быть такими источниками. Однако эти устройства в приложениях сетевой безопасности применяются ограниченно. Проблемы также вызывают грубые атаки на такие устройства. Альтернативным решением является создание набора из большого числа случайных чисел и опубликование его в некоторой книге. Тем не менее, и такие наборы обеспечивают очень ограниченный источник чисел по сравнению с тем количеством, которое требуется приложениям сетевой безопасности. Более того, хотя наборы из этих книг действительно обеспечивает статистическую случайность, они предсказуемы, так как противник может получить их копию. Таким образом, шифрующие приложения используют для создания случайных чисел специальные алгоритмы. Эти алгоритмы детерминированы и, следовательно, создают последовательность чисел, которая не является статистически случайной. Тем не менее, если алгоритм хороший, полученная последовательность будет проходить много тестов на случайность. Такие числа часто называют ***псевдослучайными числами***.

Рассмотрим несколько алгоритмов генерации случайных чисел.

Создание случайных чисел

Генераторы псевдослучайных чисел

Первой широко используемой технологией создания случайного числа был алгоритм, предложенный Лехмером, который известен как метод линейного конгруента. Этот алгоритм параметризуется четырьмя числами следующим образом:

m	Модуль (основание системы)	$m > 0$
a	Множитель	$0 \le a < m$
c	Приращение	$0 \le c < m$
X0	Начальное значение или зерно (seed)	$0 \le X_0 < m$

Последовательность случайных чисел {Xn} получается с помощью следующего итерационного равенства:

$$X_{n+1} = (a\,X_n + c) \bmod m$$

Если m, a и c являются целыми, то создается последовательность целых чисел в диапазоне

$$0 \le X_n < m.$$

Создание случайных чисел

Выбор значений для a, c и m является критичным для разработки хорошего генератора случайных чисел.
Очевидно, что m должно быть очень большим, чтобы была возможность создать много случайных чисел. Считается, что m должно быть приблизительно равно максимальному положительному целому числу для данного компьютера. Таким образом, обычно m близко или равно 2^{31}.
Существует три критерия, используемые при выборе генератора случайных чисел:

1. Функция должна создавать полный период, т.е. все числа между 0 и m до того, как создаваемые числа начнут повторяться.
2. Создаваемая последовательность должна появляться случайно. Последовательность не является случайной, так как она создается детерминированно, но различные статистические тесты, которые могут применяться, должны показывать, что последовательность случайна.
3. Функция должна эффективно реализовываться на 32-битных процессорах.

Создание случайных чисел

Значения a, c и m должны быть выбраны таким образом, чтобы эти три критерия выполнялись. В соответствии с первым критерием можно показать, что если m является простым и c = 0, то при определенном значении a период, создаваемый функцией, будет равен m-1. Для 32-битной арифметики соответствующее простое значение $m = 2^{31} - 1$. Таким образом, функция создания *псевдослучайных чисел* имеет вид:

$$X_{n+1} = (a\, X_n) \bmod (2^{31} - 1)$$

Только небольшое число значений a удовлетворяет всем трем критериям. Одно из таких значений есть $a = 7^5 = 16807$, которое использовалось в семействе компьютеров IBM 360. Этот генератор широко применяется и прошел более тысячи тестов, больше, чем все другие генераторы *псевдослучайных чисел*.

Создание случайных чисел

Сила алгоритма линейного конгруента в том, что если сомножитель и модуль (основание) соответствующим образом подобраны, то результирующая последовательность чисел будет статистически неотличима от последовательности, являющейся случайной из набора 1, 2, ..., m-1. Но не может быть случайности в последовательности, полученной с использованием алгоритма, независимо от выбора начального значения X_0. Если значение выбрано, то оставшиеся числа в последовательности будут предопределены. Это всегда учитывается при криптоанализе.

Если противник знает, что используется алгоритм линейного конгруента, и если известны его параметры ($a = 7^5$, $c = 0$, $m = 2^{31} - 1$), то, если раскрыто одно число, вся последовательность чисел становится известна. Даже если противник знает только, что используется алгоритм линейного конгруента, знания небольшой части последовательности достаточно для определения параметров алгоритма и всех последующих чисел. Предположим, что противник может определить значения X_0, X_1, X_2, X_3. Тогда :

Создание случайных чисел

$X_1 = (a X_0 + c) \bmod m$
$X_2 = (a X_1 + c) \bmod m$
$X_3 = (a X_2 + c) \bmod m$

Эти равенства позволяют найти a, c и m.
Таким образом, хотя алгоритм и является хорошим генератором *псевдослучайной последовательности чисел*, желательно, чтобы реально используемая последовательность была непредсказуемой, поскольку в этом случае знание части последовательности не позволит определить будущие ее элементы. Эта цель может быть достигнута несколькими способами. Например, использование внутренних системных часов для модификации потока случайных чисел. Один из способов применения часов состоит в перезапуске последовательности после N чисел, используя текущее значение часов по модулю m в качестве нового начального значения. Другой способ состоит в простом добавлении значения текущего времени к каждому случайному числу по модулю m.

Создание случайных чисел

Криптографически созданные случайные числа
В криптографических приложениях целесообразно шифровать получающиеся случайные числа. Чаще всего используется три способа.

Циклическое шифрование

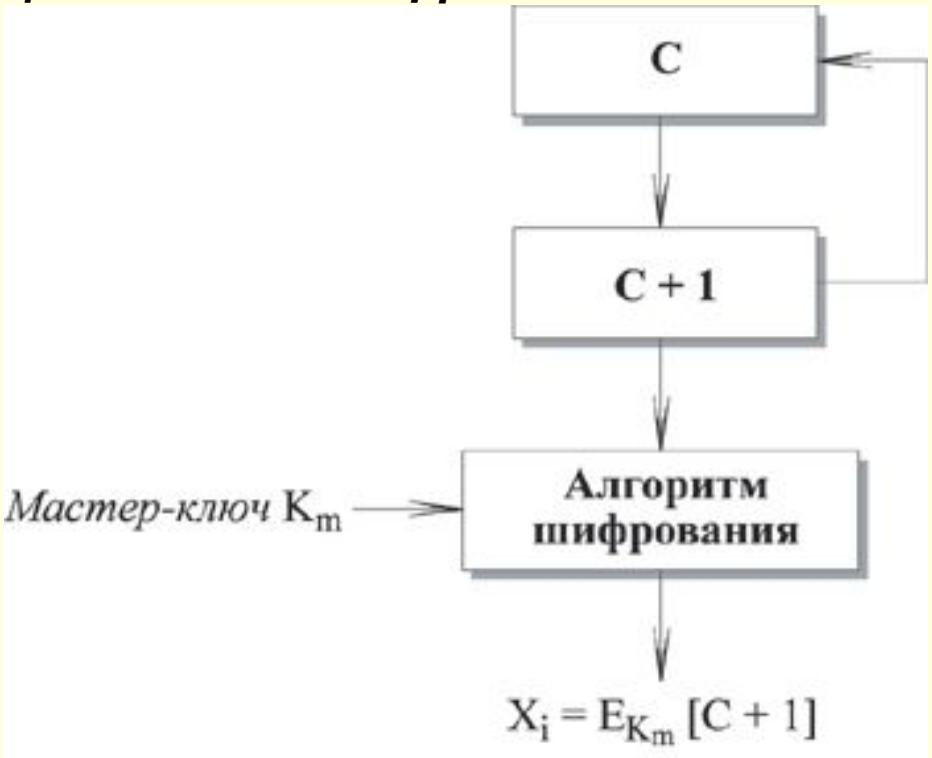


Рис. 3.14. Циклическое шифрование

Создание случайных чисел

В данном случае применяется способ создания ключа сессии из мастер-ключа. Счетчик с периодом N используется в качестве входа в шифрующее устройство. Например, в случае использования 56-битного ключа DES может применяться счетчик с периодом 2^{56}. После каждого созданного ключа значение счетчика увеличивается на 1. Таким образом, псевдослучайная последовательность, полученная по данной схеме, имеет полный период: каждое выходное значение X_0, X_1,...X_{N-1} основано на различных значениях счетчика и, следовательно, $X_0 \neq X1 \neq X_{N-1}$

Так как мастер-ключ защищен, легко показать, что любой секретный ключ не зависит от знания одного или более предыдущих секретных ключей.

Для дальнейшего усиления алгоритма вход должен быть выходом полнопериодического генератора *псевдослучайных чисел*, а не простой последовательностью.

Создание случайных чисел

Режим Output Feedback DES

Режим *OFB* DES может применяться для генерации ключа, аналогично тому, как он используется для потокового шифрования. Заметим, что выходом каждой стадии шифрования является 64-битное значение, из которого только левые j битов подаются обратно для шифрования. 64-битные выходы составляют последовательность *псевдослучайных чисел* с хорошими статистическими свойствами.

Генератор псевдослучайных чисел ANSI X9.17

Один из наиболее сильных генераторов *псевдослучайных чисел* описан в ANSI X9.17. В число приложений, использующих эту технологию, входят приложения финансовой безопасности и PGP.

Создание случайных чисел

Алгоритмом шифрования является тройной DES. Генератор ANSI X9.17 состоит из следующих частей:

1. **Вход:** генератором управляют два псевдослучайных входа. Один является 64-битным представлением текущих даты и времени, которые изменяются каждый раз при создании числа. Другой является 64-битным начальным значением; оно инициализируется некоторым произвольным значением и изменяется в ходе генерации последовательности *псевдослучайных чисел*.
2. **Ключи:** генератор использует три модуля тройного DES. Все три используют одну и ту же пару 56-битных ключей, которая должна держаться в секрете и применяться только для генерации *псевдослучайного числа*.
3. **Выход:** выход состоит из 64-битного *псевдослучайного числа* и 64-битного значения, которое будет использоваться в качестве начального значения при создании следующего числа.

Создание случайных чисел

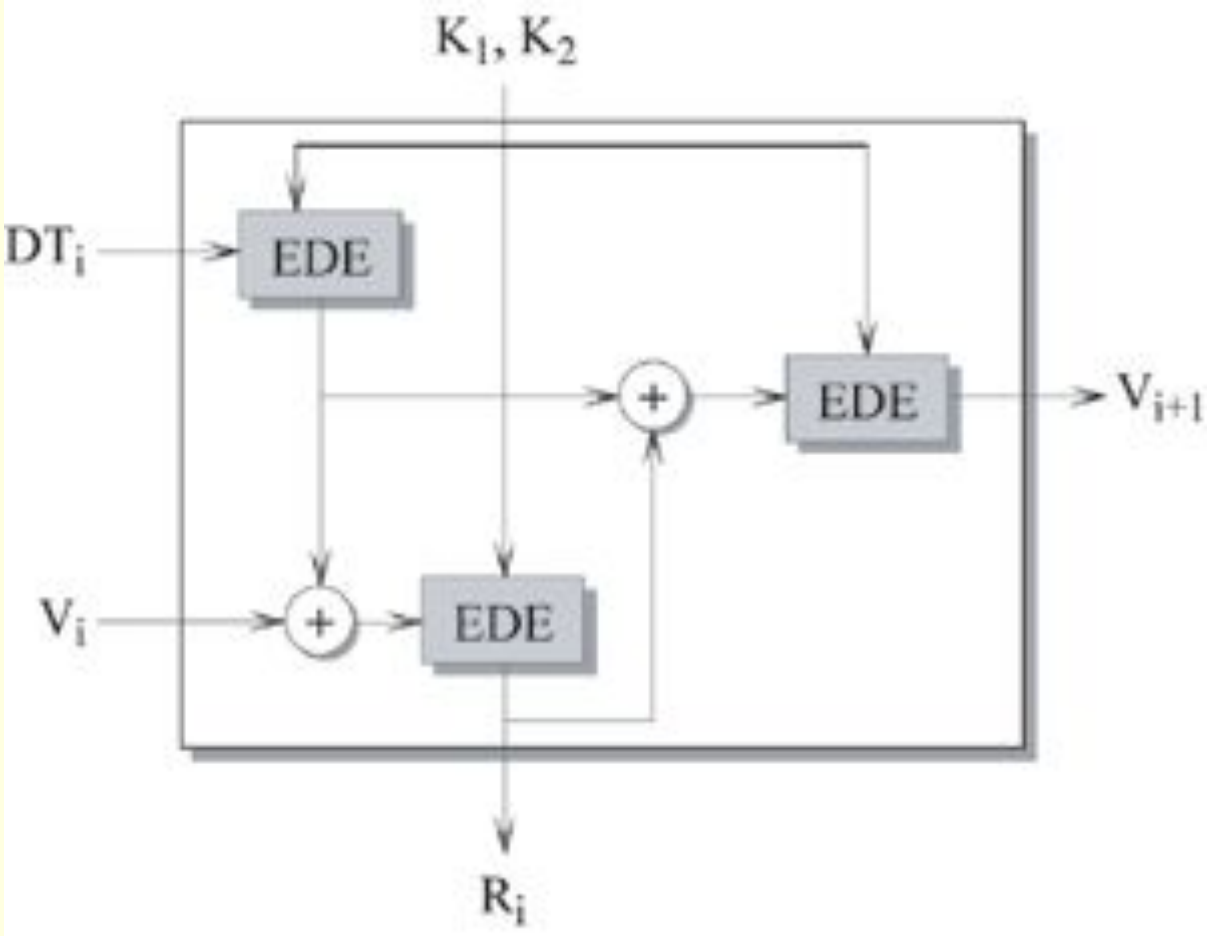


Рис. 3.15. Генератор псевдослучайных чисел ANSI X9.17

Создание случайных чисел

DT_i - значение даты и времени на начало i-ой стадии генерации.
V_i - начальное значение для i-ой стадии генерации.
R_i - *псевдослучайное число*, созданное на i-ой стадии генерации.
K_1, K_2 - ключи, используемые на каждой стадии.

Тогда:

$$R_i = EDE_{K1,K2}\,[\,EDE_{K1,K2}\,[\,DT_i]\quad V_i\,]$$
$$V_{i+1} = EDE_{K1,K2}\,[\,EDE_{K1,K2}\,[\,DT_i]\,R_i]$$

Схема включает использование 112-битного ключа и трех EDE-шифрований. На вход подаются два *псевдослучайных значения*: значение даты и времени и начальное значение очередной итерации, на выходе создаются начальное значение для следующей итерации и очередное *псевдослучайное значение*. Даже если *псевдослучайное число* Ri будет скомпрометировано, вычислить V_{i+1} из Ri невозможно, и, следовательно, следующее *псевдослучайное значение* R_{i+1}, так как для получения V_{i+1} дополнительно выполняются три операции EDE.