О г л а в л е н и е

Глава 1. Основные понятия и определения

1.1. Криптография

Проблемой защиты информации путем ее преобразования занимается ***криптология*** (***cryptology***). В переводе с гречецкого языка ***kryptos*** означает тайный, а ***logos*** - сообщение. ***Криптология*** разделяется на два направления - ***криптографию*** и ***криптоанализ***. Цели этих направлений прямо противоположные.

Криптография – занимается поиском и исследованием методов преобразования информации с целью ее шифрования. В криптографии различают два основных преобразования это непосредственно ***шифрование*** и обратный ему процесс ***дешифрирование***. Целью криптографии является обеспечение конфиденциальности (секретности) и аутентичности (подлинности) передаваемых сообщений

Криптоанализ - это область знаний, которая занимается исследованием возможностей расшифровывания информации без знания криптографических ключей.

В качестве информации подлежащей шифрованию и дешифрированию, будут рассматриваться ***исходные тексты*** (открытые тексты) и ***зашифрованные тексты*** (шифротексты), построенные с использованием некоторых ***алфавитов***. Под этими терминами в дальнейшем будем понимать следующее.

Алфавит - конечное множество элементов (знаков) используемых для представления текстов (***messages***).

Исходный текст (***Plaintext***) - упорядоченный набор из элементов алфавита исходных текстов.

Зашифрованный текст (***Ciphertext)*** - упорядоченный набор из элементов алфавита зашифрованных текстов.

Шифр – это секретный метод записи, в соответствии с которым исходный текст преобразуется в зашифрованный текст (шифротекст).

Шифрирование (***Enciphering***) – это процесс преобразования исходного текста в шифротекст.

Дешифрирование (***Deciphering***) – это обратный процесс по отношению к шифрованию, в результате которого шифротекст преобразуется в исходный текст.

Шифрование и дешифрирование зависят от криптографического ключа или ключей, количество которых зависит от класса используемой криптосистемы. Различают два основных класса криптосистем:

- симметричные одноключевые криптосистемы (с секретным колючем);
- асимметричные двухключевые криптосистемы (с открытым ключом).

На следующем рисунке приведена схема, поясняющая процедуру шифрования и дешифрирования для общего случая произвольной криптографической системы.

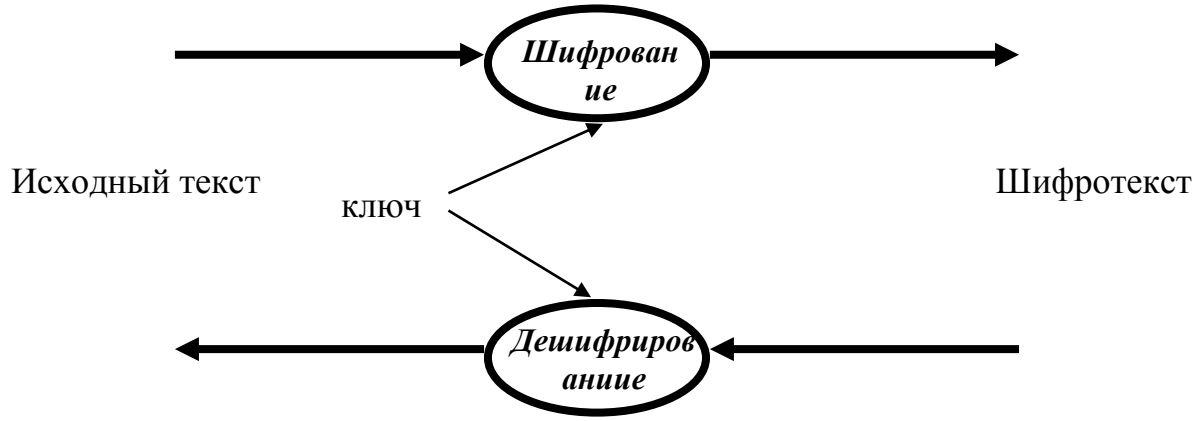


Рис.1.1. Процедура шифрования и дешифрирования

1.2. Простейшие *перестановочные* шифры

Существует два основных вида простейших алгоритмов шифрования. ***Перестановочные шифры*** (***Permutation Ciphers***) выполняющие перестановку символов в исходных данных. Например, с помощью шифра "железнодорожная изгородь" буквы исходного текста изображаются в виде железнодорожной изгороди, и затем читаются построчно. Простой пример приведенный на рис.1.2 иллюстрирует этот метод. В качестве исходного текста используется английское слово CRYPTOGRAPHY в результате шифрования получим CTARPORPYYGH – соответствующий ему шифротекст. Ключ к шифротексту задаётся "высотой" изгороди, которая в данном примере равна 3.

C R Y P T O G R A P H Y

⇓

C T A
R P O R P Y
Y G H

⇓

C T A R P O R P Y Y G H

Рис.1.2. Перестановочный шифр “железнодорожная изгородь”

Для расшифровки закодированного текста необходимо выполнить обратную процедуру.

1.3. Простейшие подстановочные шифры

Подстановочные шифры (***Substitution Cipher***) реализуют замену символов исходного текста на подстановочные элементы, которые могут быть символами алфавита исходных текстов. Простейший вид подстановочного шифра использует в качестве элементов подстановки циклически сдвинутый алфавит исходных текстов на k позиций, где k – это ключ к шифру. Этот вид шифра часто называют шифром Цезаря, так как аналогичный алгоритм использовал Юлий Цезарь для случая k=3. На рис.1.3 приведена иллюстрация метода Цезаря.

C R Y P T O G R A P H Y

⇓

F U B S W R J U D S K B

Рис.1.3. Подстановочный шифр Цезаря

В современных криптосистемах, используемых на практике, процедура подстановки обычно комбинируется с процедурой перестановки. Например, хорошо известный на практике алгоритм шифрования ***Data Encryption Standard (DES)***, кодирует 64-битные блоки исходного текста, используя комбинацию процедур подстановки и перестановки.

Код – это специальный вид подстановочного шифра, который использует "кодовую таблицу" в качестве ключа. Слова или фразы исходного текста помещаются в кодовую таблицу вместе с их подстановочными элементами шифротекста, например, для слова CRYPTOGRAPHY может быть использован код 7905. Таким образом, получение в криптограмме кода 7905 означает слово CRYPTOGRAPHY.

1.4. Криптоанализ

Криптоанализ – это наука и практика изучающая методы взлома шифров. Шифр может быть взломан, если возможно определить исходный текст или ключ из шифротекста, или определить ключ из пары "шифротекст – исходный текст". Существует несколько основных методов атак, а именно атаки, использующие только шифротекст, атаки с известным исходным текстом, и избранным исходным текстом.

В первом случае криптоаналитик должен определить ключ исключительно из перехваченного шифротекста, при этом, возможно, что известными будут метод шифрования, язык исходного текста, предмет обсуждения и даже используемые в исходном тексте слова. Например, сообщение, описывающее место расположения зарытого клада, возможно, будет содержать слова: ЗАРЫТЫЙ, КЛАД, СЕВЕР, ПОВОРОТ, ПРАВО, КИЛОМЕТР, и т.д.

Во втором случае криптоаналитик знает несколько пар "шифротекст – исходный текст". Например, предполагается, что кодированное сообщение, передающееся с пользовательского терминала на компьютер, перехватывается криптоаналитиком, который знает, что сообщение начинается со стандартного заголовка "LOGIN"

В третьем случае криптоаналитик в состоянии завладеть шифротекстом, который соответствует определённому исходному тексту. Это наиболее предпочтительный случай для криптоаналитика.

Шифр ***безоговорочно безопасен (абсолютно секретен)***, если вне зависимости от того, сколько шифротекста перехвачено, в нём всё равно будет недостаточно информации, чтобы однозначно определить исходный текст.

Шифр считается ***защищенным по вычислениям*** (***strong***), если он не может быть взломан систематическим анализом доступных ресурсов.

1.5. Безопасность данных

Современная криптография защищает данные, передаваемые по сверхскоростным электронным линиям или хранящиеся в компьютерных системах. Существуют две основные цели, которые должны достигаться в результате использования криптографических систем. Это ***скрытность*** (или ***секретность***), то есть предотвращение несанкционированного доступа к данным, и ***достоверность*** (или ***целостность***), - предотвращение несанкционированного изменения данных.

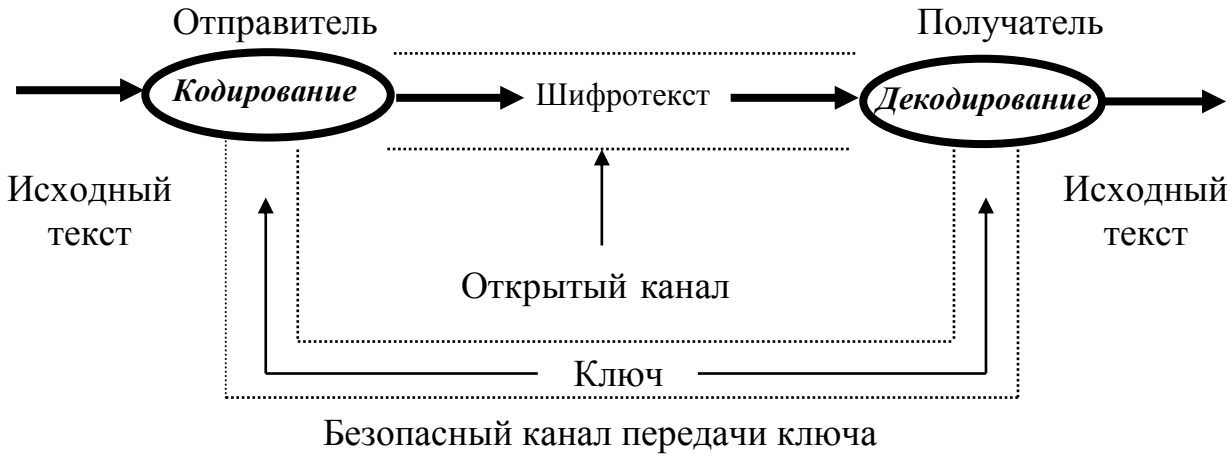


Рис.1.4. Классический информационный канал

Информация, передаваемая по открытому каналу, подвержена пассивному перехвату, что угрожает секретности, и активному перехвату, что угрожает достоверности. ***Пассивный перехват*** (***eavesdropping***) приводит к перехвату сообщений, обычно без их изменения, ***активный перехват*** (***tampering***) приводит к умышленному изменению потока данных.

1.6. Криптосистемы

В общем случае криптосистема описывается пятью компонентами.

1. ***Пространством исходных текстов***, *M.*

2. ***Пространством шифротекстов***, *C.*

3. ***Пространством ключей***, *K.*

4. Семейством ***шифрующих*** (***кодирующих***) ***преобразований***, E_k: $M \Rightarrow C$.

5. Семейством ***дешифрирующих*** (***декодирующих***) ***преобразований***, D_k: $C \Rightarrow M$.

Каждое кодирующее преобразование E_k определяется кодирующим алгоритмом E, общим для всех преобразований в семействе, и ключом k, который отличает его от других преобразований. Аналогично, каждое декодирующее преобразование D_k определяется декодирующим алгоритмом D и ключом k. Для заданного k, D_k это инверсное преобразование по отношению к E_k; это значит что $D_k(E_k(M))=M$ для каждого сообщения исходного текста M. В заданной криптосистеме преобразования E_k и D_k зависят от параметров, полученных на основании k или непосредственно от k.

Произвольная криптосистема должна отвечать определенным требования и характеристикам. К основным требованиям, предъявляемым к криптосистемам, относят.

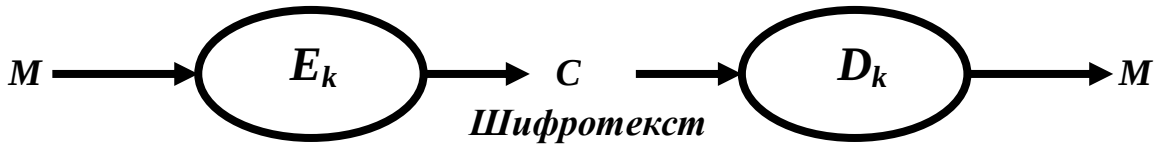


Рис.1.5. Криптосистема

1.Криптосистема должна быть проста в применении.

2.Кодирующие и декодирующие преобразования должны быть корректны для всех ключей из пространства допустимых криптографических ключей.

3.Безопасность системы должна зависеть только от секретности ключей, а не от секретности алгоритмов E и D.

1.7. Требования к секретности и достоверности

Существует ряд требований к секретности и достоверности передаваемой информации.

Требования к секретности

1.Для криптоаналитика должно быть вычислительно невозможным определение декодирующего преобразования D_k на основании

перехваченного шифротекста *C*, даже если соответствующий исходный текст *M* ему известен.

2.Для криптоаналитика должно быть вычислительно невозможным определение исходного текста *M* на основании перехваченного шифротекста *C*.

Требования к достоверности

1.Для криптоаналитика должно быть вычислительно невозможным определение кодирующего преобразования E_k для заданного *C*, даже если соответствующий исходный текст *M* ему известен.

2. Для криптоаналитика должно быть вычислительно невозможным нахождение шифротекста *C'* такого, что $D_k(C')$ будет являться корректным исходным текстом, определённым в пространстве исходных текстов *M*.

Процесс криптографического закрытия данных может осуществляться как программно, так и аппаратно. Аппаратная реализация отличается существенно большей стоимостью, однако ей присущи и преимущества: высокая производительность, простота, защищенность и т.д. Программная реализация более практична, допускает известную гибкость в использовании.

Для современных криптографических систем защиты информации сформулированы следующие общепринятые требования:

- зашифрованное сообщение должно поддаваться чтению (дешифрированию) только при наличии криптографического ключа;
- число операций, необходимых для определения использованного ключа шифрования по фрагменту шифрованного сообщения и соответствующего ему открытого текста, должно быть не меньше чем для общего числа возможных ключей;
- число операций, необходимых для расшифровывания информации путем перебора всевозможных ключей должно иметь строгую нижнюю оценку и выходить за пределы возможностей современных компьютеров (с учетом возможности использования сетевых вычислений);
- знание алгоритма шифрования не должно влиять на надежность защиты;
- незначительное изменение ключа должно приводить к существенному изменению вида зашифрованного сообщения;
- структурные элементы алгоритма шифрования должны быть неизменными;
- дополнительные биты, вводимые в сообщение в процессе шифрования, должны быть полностью и надежно скрыты в шифрованном тексте;
- длина шифрованного текста должна быть равной длине исходного текста;
- не должно быть простых и легко устанавливаемых зависимостей между ключами, последовательно используемыми в процессе шифрования;
- любой ключ из множества возможных ключей должен обеспечивать надежную защиту информации;

- алгоритм должен допускать как программную, так и аппаратную реализацию, при этом изменение длины ключа не должно вести к существенному качественному ухудшению алгоритма шифрования.

1.8. Классификация криптосистем

В соответствии с классификацией Симмонса (***Simmons***) криптосистемы разделяются на ***симметричные*** (***одноключевые***) криптосистемы и ***асимметричные*** (***двухключевые***) криптосистемы. Возможны варианты с использованием большего, чем два количества ключей. В ***симметричных*** криптосистемах кодирующий и декодирующий ключ одинаковый (или легко получаемый один из другого). Это значит, что преобразования E_k и D_k также легко получаются одно из другого. До недавнего времени, все криптосистемы были одноключевыми. Поэтому, одноключевые системы часто упоминаются как ***обычные*** (или ***классические***) криптографические системы. *DES* – это классическая криптографическая система.

Одноключевые криптографические системы представляют собой замечательный способ кодирования личных файлов пользователя. Каждый пользователь *A* имеет личные преобразования E_k и D_k для кодирования и декодирования файлов.

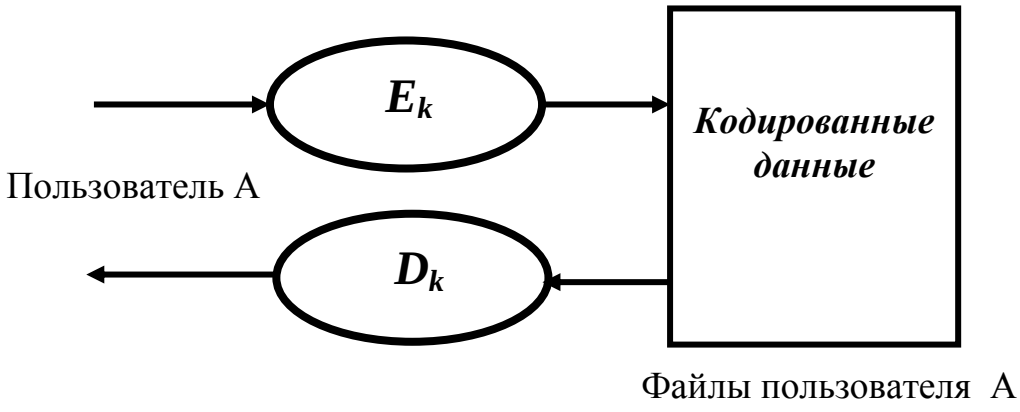


Рис.1.6. Кодирование личных файлов с помощью одного ключа

При создании баз данных с различными правами пользователей возможно использование двухключевых криптосистем

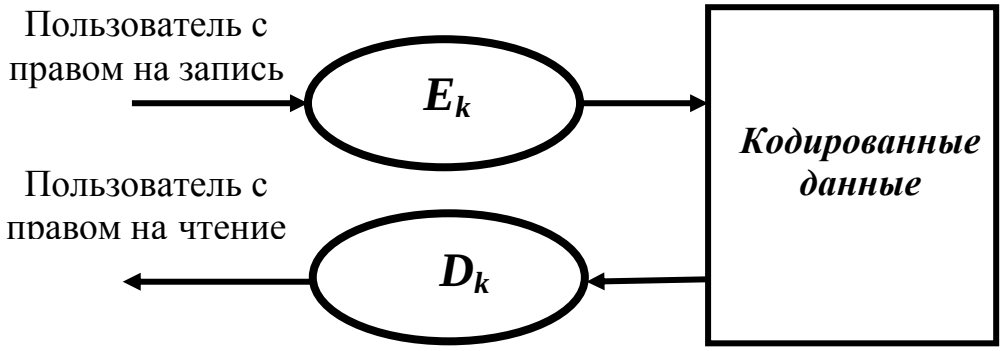


Рис.1.7. Кодирование файлов с раздельными ключами чтения/записи

В данном случае E_k является ключом известным только пользователю с правом записи, а D_k ключом для чтения известным пользователю с правом чтения.

1.8. Системы с открытым ключом

В системах с открытым ключом каждый пользователь (A) имеет ***открытое кодирующее преобразование*** $\boldsymbol{E_A}$, которое может храниться в открытой базе данных, и ***секретное декодирующее преобразование*** $\boldsymbol{D_A}$, известное только этому пользователю. Секретное преобразование D_A описывается секретным ключом, а открытое преобразование E_A – открытым ключом, полученным из секретного преобразования D_A при помощи односторонних вычислений. Определяющим требованием к вычислению E_A из D_A является требование вычислительной невозможности определения D_A из E_A или преобразования эквивалентного D_A.

В таких системах секретность и достоверность обеспечиваются независимыми преобразованиями. Предположим, пользователь A хочет послать сообщение M другому пользователю B. Если A знает открытое преобразование E_B пользователя B, то A может секретно передать M пользователю B путём пересылки шифротекста $C=E_B(M)$. При получении шифротекста C пользователь B декодирует C используя своё секретное преобразование D_B и в результате получает $D_B(C)=D_B(E_B(M))=M$. Передаваемая информация является секретной, однако при полном отсутствии достоверности (целостности). В качестве пользователя A может выступать абсолютно любой пользователь в том числе и злоумышленник который знает открытый ключ пользователя B.

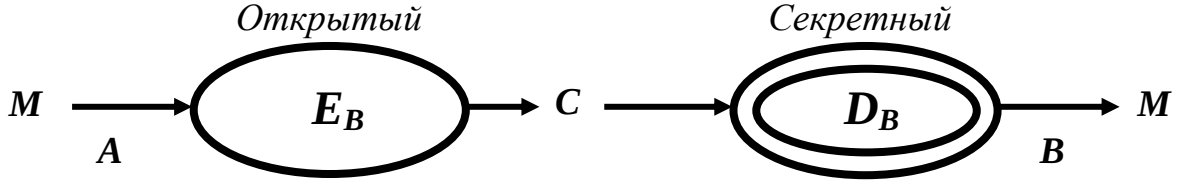


Рис.1.8. Секретность в системе с открытым ключом

Для достижения достоверности, *M* должно быть преобразовано пользователем *A* с помощью собственного секретного преобразования D_A. Игнорируя секретность, *A* посылает шифрограмму $C=D_A(M)$ пользователю *B*. По получении шифрограммы, пользователь *B* использует открытое преобразование E_A отправителя *A* и в результате преобразования получает $E_A(C)=E_A(D_A(M))=M$. В данном случае достигается достоверность (целостность) информации при полном отсутствии ее секретности. Любой из пользователей, в том числе и злоумышленник, может дешифрировать и прочитать передаваемое сообщение.

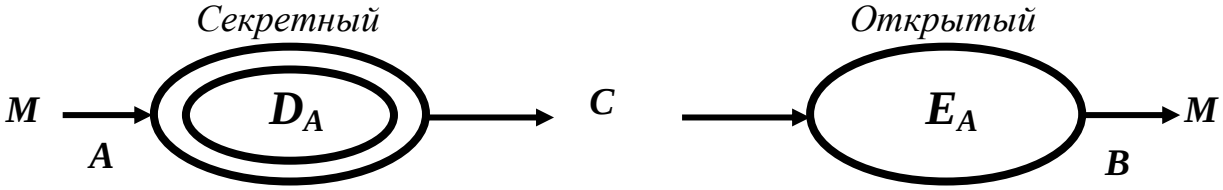


Рис.1.9. Достоверность в системах с открытым ключом

Чтобы достичь одновременно и секретности и достоверности, отправитель и получатель должны применить два этапа преобразования. Отправитель *A* генерирует шифротекст $C=E_B(D_A(M))$, а *B* восстанавливает *M* согласно правилу: $E_A(D_B(C))=E_A(D_B(E_B(D_A(M))))= E_A(D_A(M))=M$.

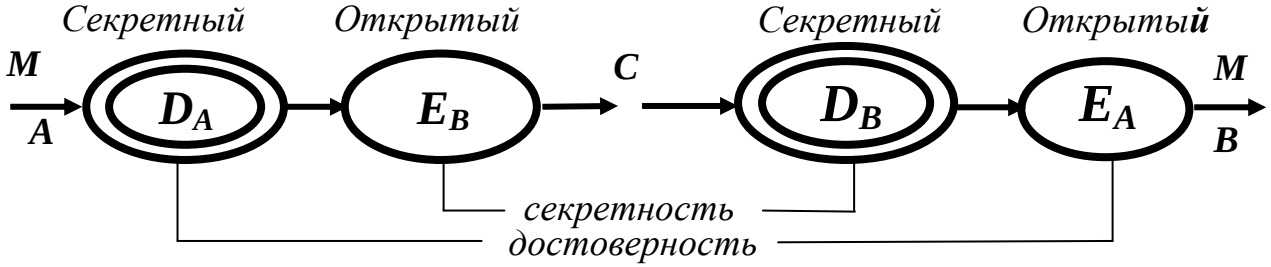


Рис.1.10. Секретность и достоверность в системах с открытым ключом

Как видно из приведенной диаграммы последовательное применение открытых и закрытых преобразований пользователей позволяет достичь одновременно секретности и целостности передаваемой информации.

1.9.Цифровые подписи

Цифровая подпись – может быть интерпретирована как свойство принадлежности подписанного документа пользователю или процессу, примененному для его подписи. Пусть *B* будет получателем сообщения *M*,

подписанного пользователем *A*. Тогда подпись пользователя *A* должна удовлетворять следующим требованиям:

1.*B* должен быть в состоянии подтвердить подпись *A* на *M*.

2.Никому, в том числе и *B*, нельзя подделать подпись пользователя *A*.

3.В случае, когда *A* будет отрицать свою подпись на сообщении *M*, должно быть возможно судье или какой-либо третьей стороне разрешить спор между *A* и *B*.

При соблюдении требований приведенных выше цифровая подпись обеспечивает ***аутентификацию и достоверность*** передаваемого сообщения аналогично обычной ручной подписи. Получатель сообщения воспринимает принятое сообщение как целостный не измененный третей стороной документ, подписанный пользователем *A*.

Системы с открытым ключом обеспечивают простую схему осуществления цифровой подписи. Преобразование D_A пользователя *A*, секретное для остальных пользователей, может служить его цифровой подписью. Получатель *B* сообщения *M*, подписанного *A* (то есть, преобразованного с помощью D_A) получает гарантию принадлежности полученных данных отправителю. Для *B* или ещё кого-либо невозможно подделать подпись *A* на другом сообщении, а для *A* невозможно отрицать подписание документа (принимая, что D_A не было украдено или потеряно). Поскольку обратное преобразование E_A открыто, получатель *B* может быстро определить подлинность подписи, а судья может разрешить любые споры между *A* и *B*. Таким образом:

1.*A* подписывает *M*, вычисляя $C = D_A(M)$.

2.*B* проверяет подпись *A* удостоверяясь, что $E_A(C)$ восстанавливает *M*.

3.Судья решает спор между *A* и *B* проверяя, какое $E_A(C)$ восстановит *M* тем же путём, как это было сделано пользователем *B*.

Более подробно обо всех вопросах, затронутых в данной главе, будет изложено в последующих разделах.

Глава 2. Алгоритмы шифрования
2.1. Перестановочные шифры

Многие используемые ранее шифры просты как в понимании, так и в применении для шифрования данных. Показано, что достаточно только изменить порядок слов, букв, или правило чтения для того чтобы превратить сообщение в секретный код. Самые простые примеры перестановочного шифрования достаточно примитивны по своей сути, однако требуют некоторых усилий для восприятия зашифрованного текста. Наиболее простым шифром подобного типа является алгоритм, реализующий объединение слов исходного текста. Шифрование в данном случае будет заключаться в удалении пробелов в исходном тексте и представлении его в виде щифротекста. Использование заглавных букв позволяет усложнить код и сделать его тяжелее для чтения и расшифрирования.

Например, зашифрованный текст:

THISISHARDCODEFORMANYPEOPLE,

является достаточно сложно воспринимаемым текстом для многих пользователей. Хотя на самом деле он соответствует легко восстанавливаемому исходному тексту

This Is Hard Code For Many People.

Следующим шагом усложнения перестановочного алгоритма шифрования является разбиение зашифрованного текста на блоки. В этом случае сообщение разбивается на блоки по два, три или больше символов. Тот же исходный текст для случая блочного представления по два символа будет представлен шифротекстом

TH IS IS HA RD CO DE FO RM AN YP EO PL E.

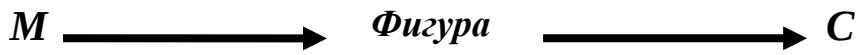
Весьма эффективной для шифрования может быть запись слов исходного текста в обратной последовательности. Написание слов, предложений или полного сообщения в обратном порядке (задом наперед) может быть весьма эффективным способом шифрования. Так, для того же случая исходного текста “This Is Hard Code For Many People“ зашифрованный текст имеет вид

SIHT SI DRAH EDOC ROF YNAM ELPOEP.

Обратный порядок чтение слова SIHT позволяет получить исходное слово THIS.

В общем случае перестановочный шифр переставляет символы исходного текста согласно определенной схеме. Эта перестановка может

быть описана при помощи некоторой геометрической фигуры. В этом случае процедура шифрования состоит из двух шагов.



Правило записи (***Write-in***) Правило чтения (***Take-off***)

Рис.2.1. Общая структура перестановочных шифраторов

Согласно общей схеме, первоначально исходный текст записывается в виде определенной геометрической фигуры в соответствии с некоторым ***правилом записи*** (***Write-in***), затем записанный таким образом текст читается согласно некоторому ***правилу чтения*** (***Take-off***). Ключом подобной схемы шифрования является правило записи *Write-in* и правило чтения *Take-off*.

Пример 2.1. Предположим, что обычный текст CRYPTOGRAPHY записан согласно правилу записи в матрицу (3 строки и 4 столбца) следующим образом:

1	**2**	**3**	**4**
C	R	Y	P
T	O	G	R
A	P	H	Y

Правило чтения заключается в считывании столбцов в следующей последовательности 3-1-4-2, тогда полученный зашифрованный текст будет иметь вид - YGHCTAPRYROP. Для дешифрования используется обратная процедура.

В общем виде процедуру шифрования в соответствии с перестановочными алгоритмами можно описать следующим образом. Исходный текст M представляется в виде блоков данных из d символов $M=m_1,...,m_d\ m_{d+1},\ldots,m_{2d},...,$ Тогда зашифрованный текст представляется как совокупность блоков исходного текста преобразованных в соответствии с функцией f. В результате получим

$$E_K(M)=m_{f(1)},...,m_{f(d)},m_{f(d+1)},...,m_{f(2d)},...$$

Дешифрование использует обратную перестановку.

Пример 2.2. Предположим, что $d=4$ и функция перестановки f имеет вид

i	1	2	3	4
$f(i)$	3	1	4	2

таким образом, исходный текст делится на блоки по 4 бита каждый, затем для каждого блока первый символ исходного текста перемещают на вторую позицию, второй знак на четвертую позицию и т.д. В результате получим

M = C R Y P T O G R A P H Y

$E_K(M)$ = Y C P R G T R O H A Y P

Подобно перемещению столбцов, периодический шифр перестановки может рассматриваться как перемещение столбцов матрицы, в которой обычный текст записан строками.

Пример 2.3. Предположим, что $d=12$ и геометрическая фигура - матрица, состоит из четырех строк и трех столбцов, а функция перестановки столбцов f имеет форму

i	1	2	3	4
f(i)	3	1	4	2

Таким образом, исходный текст делится на блоки по 12 бит каждый, тогда каждый блок имеет вид матрицы 3×4, где третий столбец каждой матрицы записывается как первая часть блока зашифрованного текста, первый столбец соответствует второй части блока и т.д. Для исходного текста M= HERE IS A SECRET MESSAGE ENCIPHERED BY TRANSPOSITION, деление на блоки и запись в виде матриц имеет вид:

H E R E	T M E S	P H E R	S P O S
I S A S	S A G E	E D B Y	I T I O
E C R E	E N C I	T R A N	N

Тогда шифротекст будет преобразован к виду:

$E_K(M)$=RARHIEESEESCEGCTSESEIMANEBAPETRYNHDROISINSOPT

Одна из наиболее известных модификаций метода перестановок типа простая перестановка столбцов использует ключевое слово или фразу в качестве криптографического ключа. Например, слово CONVENIENCE, используемое как ключ, определяет порядковый номер для каждого символа в слове согласно следующему правилу. Буквам ключевого слова назначаются порядковые номера, начиная с номера 1. Порядок их назначения вначале определяется в соответствии с алфавитом исходного текста, а в случае, когда один и тот же символ появляется дважды, нумерация определяется порядком их следования в ключевом слове.

Пример 2.4. Ключевое слово CONVENIENCE определяет количество столбцов для записи исходных текстов, а буквы этого слова определяют порядок чтения столбцов текста. Таким образом, получим следующую нумерацию столбцов.

C	O	N	V	E	N	I	E	N	C	E
1	***10***	***7***	***11***	***3***	***8***	***6***	***4***	***9***	***2***	***5***

Здесь буква *C* определяет столбец с номером 1. Отмерим, что в ключевом слове отсутствуют буквы, предшествующие букве *C*, то есть, буквы *A* и *B*. Вторая буква *C* в ключевом слове определяет второй столбец. Следующей буквой английского алфавита, используемой в ключевом слове, является буква *E*, которая определяет 3, 4 и 5 столбцы матрицы.

Для исходного текста *M*=HERE IS A SECRET MESSAGE ENCIPHERED BY TRANSPOSITION получим следующую запись

C	O	N	V	E	N	I	E	N	C	E
1	***10***	***7***	***11***	***3***	***8***	***6***	***4***	***9***	***2***	***5***
H	E	R	E	I	S	A	S	E	C	R
E	T	M	E	S	S	A	G	E	E	N
C	I	P	H	E	R	E	D	B	Y	T
R	A	N	S	P	O	S	I	T	I	O
N										

Используя правило чтения, определенное ключевым словом, получим следующий зашифрованный текст *C*=HECRN CEYI ISEP SGDI RNTO AAES RMPN SSRO EEBT ETIA EEHS.

Пример 2.5. Развитием предыдущего метода является использование нумерации букв ключевого слова для задания правила чтения, а также правила записи. Правило записи определяет длину строки. Здесь первая строка оканчивается на номере столбца 1. Конечный символ второй строки записывается под номером 2 и так далее.

C	O	N	V	E	N	I	E	N	C	E
1	***10***	***7***	***11***	***3***	***8***	***6***	***4***	***9***	***2***	***5***
H										
E	R	E	I	S	A	S	E	C	R	
E	T	M	E	S						
S	A	G	E	E	N	C	I			
P	H	E	R	E	D	B	Y	T	R	A
N	S	P	O	S	I	T				
I	O	N								

В результате получим следующий зашифрованный текст *C*=HEESPNI RR SSEES EIY A SCBT EMGEPN ANDI CT RTAHSO IEERO.

Следующим достаточно интересным перестановочным шифратором является метод "поворачивающейся решетки", который был использован Германией во времена первой мировой войны.

Суть метода заключается в том, что квадратная решетка, разделенная на сетку квадратов, часть из которых имеют отверстия, помещается на листе бумаги. Исходный текст записывается на бумаге через отверстия, затем решетка поворачивается на 90 градусов, и далее сообщение продолжает записываться в позициях решетки, где имеются отверстия. Данная процедура продолжается для всех четырех возможных положений решетки.

Сущность проектирования такой решетки состоит в делении квадратной решетки на зоны, нумеруя квадраты в каждой четверти так, чтобы при вращении решетки соответствующие квадраты имели тот же самый номер. Затем выбираются по одному квадрату с каждым порядковым номером для выполнения перфорирования.

Пример 2.6. Пример проектирования поворачивающейся решетки состоящей из 25 квадратов.

1	2	3	4	1
4	5	6	5	2
3	6	7	6	3
2	5	6	5	4
1	4	3	2	1

	2		4	1
2	5	6	5	
3	6			3
4		6	5	4
1		3	2	1

Пример 2.7. Пример поворачивающейся решетки и ее использования для сообщения: *M*=HERE IS A SECRET MESSAGE WRITE.

В первом раунде получим

	2		4	1
4	5	6	5	
3	6			3
2		6	5	4
1		3	2	1

H		E		
				R
		E	I	
	S			
	A			

Во втором раунде имеем

1	2	3	4	
		6	5	2
3	6	*	6	
2	5		5	4
1	4	3		1

H		E		S
E	C			R
		E	I	R
	S	E		
	A		T	

В третьем раунде получим

1	2	3		1
4	5	6		2
3		*	6	3
	5	6	5	4
1	4		2	

H		E	M	S
E	C		E	R
	S	E	I	R
S	S	E		
	A	A	T	G

В четвертом раунде имеем

1		3	4	1
4	5		5	2
	6	*	6	3
2	5	6		
	4		2	

H	E	E	M	S
E	C	W	E	R
R	S	E	I	R
S	S	E	I	T
E	A	A	T	G

Окончательно получим зашифрованный текст *C*=HEEMS ECWER RSEIR SSEIR SSEIT EAATG.

2.2. Подстановочные шифры

Простейшие ***шифры подстановки*** (***замены***) реализуют замену каждого символа исходного текста на соответствующий символ зашифрованного текста. Для того чтобы зашифровать исходный текст согласно простейшему подстановочному шифру необходимо наличие таблицы подстановки, которая определяет соответствие символов исходного текста символам шифротекста.

Простейший шифр замены заменяет каждый символ алфавита исходных текстов, обозначенного как *M*, на соответствующий символ алфавита шифротекстов, обозначенного как *C*. Чаще всего *C* представляет собой простую перестановку лексикографического порядка символов в алфавите *M*.

Предположим, алфавит *M* состоит из *n* символов $M=\{a0,a1,...,an\}$, тогда *C* представляет собой *n*-символьный алфавит $C=\{f(a0),f(a1),...,f(an)\}$, где функция *f* выполняет отображение $M \rightarrow C$, то есть, непосредственную замену каждого знака из *M* соответствующим знаком из *C*. Ключом простейшего подстановочного шифра является ***таблица подстановки***, которая может задаваться в явном виде или описываться некоторой функцией *f*.

Для того чтобы зашифровывать, сообщение $M=m1m2...$, представляющее собой исходный текст, используется таблица подстановки, и в результате получается зашифрованное сообщение $E_K(M)=f(m1)f(m2)\ldots$

Пример 2.7. Предположим, что функция *f* определяет соответствие символов английского алфавита $M = \{A, B, ...\}$ символам того же алфавита, но представленного в другой последовательности, определенной следующей таблицей подстановки:

M:ABCDEFGHIJKLMNOPQRSTUVWXYZ
C:YARMOLIKBCDEFGHJNPQSTUVWXZ.

Тогда исходный текст *M*=CRYPTOGRAPHY будет зашифрован как: *C*=RPXJSHIPYJKX.

В предыдущем примере с целью уменьшения размерности ключа использовалось ключевое слово (в данном случае YARMOLIK), которое позволяет однозначно определить всю таблицу подстановки. Схема использования ключевого слова достаточно проста. Первоначально под первыми буквами алфавита исходных текстов записывается ключевое слово без повторяющихся символов. Если символ повторяется в ключевом слове несколько раз, то для построения таблицы подстановки используется только одно его значение, а остальные игнорируются. Затем символы исходного алфавита, которые не были использованы в ключевом слове, в упорядоченном порядке записываются в таблицу подстановки. Очевидны различные модификации приведенного алгоритма генерирования таблицы подстановки как по месту записи в таблице подстановки ключевого слова, так и по правилам ее генерирования.

Существует достаточно много способов генерирования таблиц подставки, являющихся ключом подстановочного шифра. Одним из возможных путей, как показано ниже, может быть так называемый ***метод компаса***. Сущность данного метода состоит в том, что в системе прямоугольных координат выписываются символы алфавита вдоль каждой линии координат по часовой стрелке. Штрих пунктирные стрелки на приведенном ниже рисунке показывают направление, используемое для заполнения диаграммы по спирали. Как только все символы алфавита окажутся вписанными в диаграмму компаса, затем они объединяют в строку символов. И, наконец, исходный алфавит используется для построения результирующей таблицы подстановки

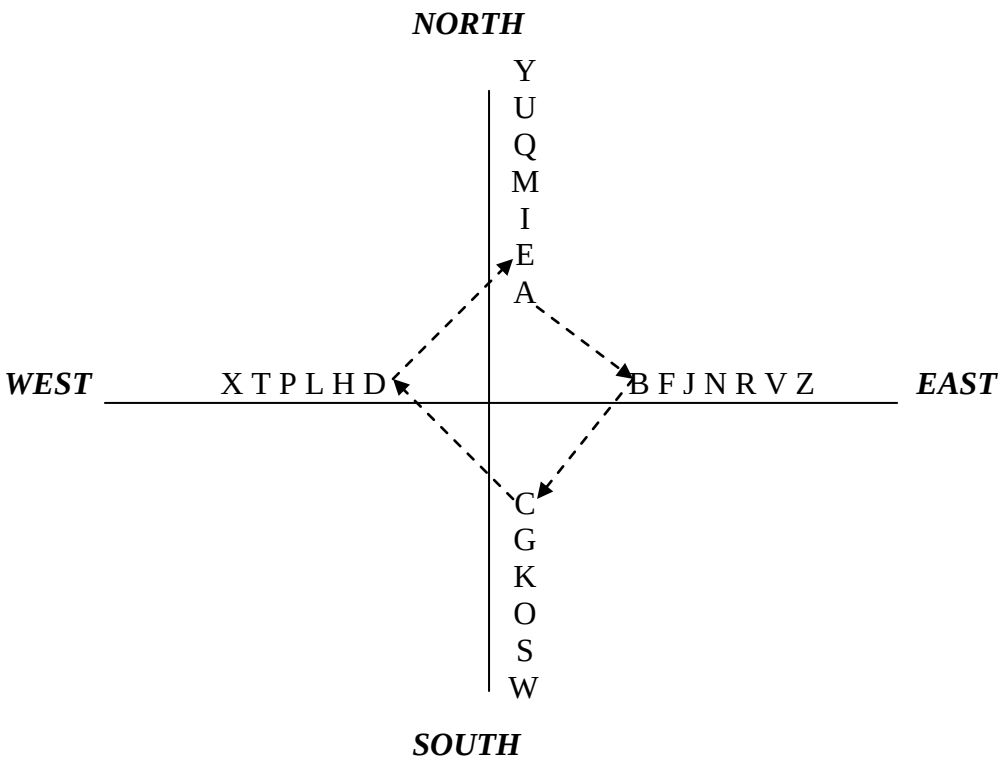


Рис.2.2. Общая структура метода компаса

В результате получим ключ, представленный в виде таблицы подстановки

M:ABCDEFGHIJKLMNOPQRSTUVWXYZ
C:YUQMIEAZVRNJFBWSOKGCXTPLHD

Суть рассмотренного выше метода генерирования таблицы подстановки заключается в том, что правило компаса позволяет формально описать таблицу подстановки или точнее формализовать правило ее построения.

Аналогичным по своей сути является метод ***говорящих часов***, предложенный ***Angie Wimer***. В соответствии с данным методом, алфавит исходных текстов записывается по окружности, и каждый символ соединяется с другим случайно выбранным символом путем проведения хорды, соединяющей две точки окружности, соответствующие символам исходного алфавита.

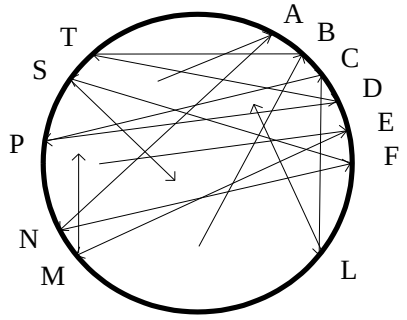


Рис.2.3. Общая структура метода говорящих часов

Дальнейшие попытки формализовать описание таблицы подстановки привели к появлению так называемых ***Полибианских квадратов***. За два века до нашей эры греческий писатель и историк *Полибий* изобрел для целей шифрования таблицу размером 5×5, заполненную в случайном порядке буквами греческого алфавита, например, как показано на рис 2.4.

α	ρ	κ	θ	φ
β	χ	ω	π	υ
δ	ψ	ο	γ	η

ζ	μ		ϕ	σ
ν	ξ	ε	τ	ϖ

Рис.2.4. Полибианский квадрат

При шифровании очередная буква исходного текста заменялась буквой, расположенной ниже ее в том же столбце квадрата. Если буква текста оказывалась в нижней строке таблицы, то для шифротекста выбирали самую верхнюю букву из того же столбца.

Пример 2.8. Например, для исходного текста M=ψξζψσα получается шифротекст C=μρνμϖβ.

Концепция полибианского квадрата оказалась весьма плодотворной и нашла свое практическое применение и развитие в последующих криптографических алгоритмах.

2.3. Шифратор Цезаря

Последующие изыскания в части создания эффективных подстановочных шифраторов были направлены в сторону поиска математического описания процедуры шифрования. В этом случае таблица подстановки присутствует в неявном виде, что существенно упрощает как саму процедуру шифрования, так и использование подобных систем на практике.

Классическим примером подобных криптосистем является ***метод Цезаря***. Свое название этот шифр получил по имени римского императора Гая Юлия Цезаря, который использовал этот шифр при переписке с Цицероном (около 50 лет до нашей эры).

Согласно методу Цезаря, при шифровании исходного текста каждая буква заменяется другой буквой того же алфавита путем смещения по алфавиту от исходной буквы на k букв. При достижении последнего символа алфавита осуществляется циклический переход к его началу. Согласно легенде, Цезарь использовал подобный шифр для k=3.

Математически метод Цезаря описывается выражением

$$f(a)=(a+k) \bmod n,$$

где n - размерность алфавита (количество символов), k-криптографический ключ, а величина a обозначает символ исходного текста его порядковым номером в алфавите M.

Для английского алфавита, используя последовательную нумерацию букв 0-A, 1-B, 2-C, 3-D, 4-E, 5-F, 6-G, 7-H, 8-I, 9-J, 10-K, 11-L, 12-M, 13-N, 14-O, 15-P, 16-Q, 17-R, 18-S, 19-T, 20-U, 21-V, 22-W, 23-X, 24-Y, 25-Z, процедура шифрования, предложенная Цезарем, будет описываться соотношением

$$f(a)=(a+3) \bmod 26.$$

Следует отметить, что как *a*, так и *f(a)* представляют собой номера букв в исходном алфавите. При шифровании буквы *G* исходного алфавита, имеющей номер 6, получим *f(a)*=6+3=9, что соответствует букве *J*, используемой в качестве подстановочного элемента в шифротексте.

Пример 2.9. Например, для случая использования шифра Цезаря при шифровании исходного текста *M*=CRYPTOGRAPHY получим *C*=FUBSWRJ UDSKB.

Более сложное преобразование алфавита исходного текста основывается на использовании так называемого метода ***децимаций***. Шифраторы подобного вида описываются соотношением, основанном на операции умножения

$$f(a)=(k\times a) \bmod n,$$

где *k* и *n* являются взаимно простыми числами. Данное требование необходимо для выполнения однозначного декодирования зашифрованного текста.

Пример 2.10. Например, для исходного текста *M*=CRYPTOGRAPHY и *k*=3 получим $f(a)=(3\times a) \bmod 26$, соответственно *C*=GZUTFQSZATVU.

Комбинацией двух приведенных выше методов шифрования является ***афинное преобразование***, описываемое выражением

$$f(a)=(k_0\times a+ k_1) \bmod n.$$

Здесь криптографический ключ состоит из двух частей k_0 и k_1. Увеличение составных частей ключа приводит к использованию в качестве алгоритма шифрования полиномиального преобразования.

2.4. Шифраторы, использующие различные алфавиты

Все ранее рассмотренные примеры простейших подстановочных алгоритмов шифрования основывались на использовании одного и того же алфавита для исходных текстов и шифротекстов.

Чаще всего подстановочные элементы шифротекста кодируются цифровыми эквивалентами. Примером подобного подхода может быть метод, основанный на использовании ключа в виде некоторого предложения либо словосочетания в котором используются все буквы алфавита исходных текстов. Например, фраза ключевого словосочетания, имеющего определенный смысл, “*The quick brown fox jumps over the lazy dog*” содержит все символы английского алфавита. Отметим, что повторение символов исходного текста еще больше усложняет шифр, так как в данном случае одной букве исходного текста будет соответствовать несколько подстановочных элементов. При построении ключевой фразы необходимо

придерживаться ограничений на длину выбираемых слов, которая не должна превышать девяти символов.

THE	QUICK	BROWN	FOX	JUMPS	OVER	THE	LAZY	DOG
1	2	3	4	5	6	7	8	9

Рис.2.5. Ключевая фраза, определяющая таблицу подстановки

Так, в соответствии с приведенной таблицей символу исходного текста *T* соответствую подстановочные элементы 11 и 71.

Пример 2.11. Для исходного текста *M*=JOHNSON IS SPY получим

J	O	H	N	S	O	N	I	S	S	P	Y
51	33	12	35	55	33	35	23	55	55	54	84

Тогда шифротекст принимает вид *C*=513312355533352355555484.

Весьма оригинальным шифром, использующим различные алфавиты для исходных текстов и шифротекстов, является шифр ***церковного двора*** (***The Churchyard cipher***), который был использован для шифрования текста на надгробном камне.

В соответствии с шифром церковного двора, на надгробной плите на одном из кладбищ Нью-Йорка в 1794 был выгравирован текст

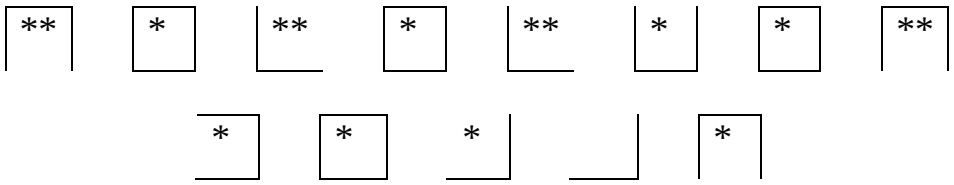


Рис.2.6. Криптограмма шифра церковный двор

Для того, что бы прочитать зашифрованный текст необходимо знать соответствующий криптографический ключ, который в данном случае представляется в виде таблицы подстановки следующего вида:

A*	B*	C*
D*	E*	F*
G*	H*	I-J*

K**	L**	M**
N**	O**	P**
Q**	R**	S**

T	U	V
W	X	Y
Z		

Данный пример показывает многообразие возможных реализаций простейших подстановочных шифраторов.

2.5. Свойства подстановочных шифров

Определяющим недостатком всех ранее рассмотренных простейших подстановочных шифраторов является сохранение частотных свойств исходных текстов в шифротекстах. Простейшие криптографические атаки основаны на анализе частот появления символов в шифротексте, что является основой для взлома подобных криптосистем. В общем случае простейший подстановочный шифр легко взламывается, используя для атаки распределение вероятностей символов в исходном тексте. Как было показано в ряде литературных источников, буквы английского алфавита могут быть разделены на подмножества с различной частотой использования в исходных текстах В англоязычных текстах выделяют высокую, среднюю, низкую, и редкую повторяемость букв английского алфавита:

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

Сравнивая частоты повторяемости букв в зашифрованном тексте с ожидаемыми частотами, специалист криптоаналитик может сопоставить букву зашифрованного текста с буквой обычного текста и таким образом расшифровать зашифрованный текст. Ниже приведена средняя оценка процентного использования букв английского алфавита в исходных текстах.

E	12.31	L	4.03	B	1.62
T	9.59	D	3.65	G	1.61
A	8.05	C	3.20	V	0.93
O	7.94	U	3.10	K	0.52
N	7.19	P	2.29	Q	0.20
I	7.18	F	2.28	X	0.20
S	6.59	M	2.25	J	0.10
R	6.03	W	2.03	Z	0.09
H	5.14	Y	1.88		

Рис.2.7. Процентная оценка использования букв английского алфавита

Следует отметить, что приведенная оценка у различных авторов имеет незначительное отличие, что соответствует статистическому разбросу и спецификой исследований авторов.

Распределение биграмм и триграмм также весьма полезно для взлома криптограмм. Наиболее часто встречаемые в английском языке пары букв и их употребление в процентном отношении представлены в следующей таблице.

TH	HE	AN	IN	ER	RE	ES	ON	EA	TI	AT	ST	…
3.15	2.51	1.72	1.69	1.54	1.48	1.45	1.45	1.31	1.28	1.24	1.21	…

Наиболее часто встречаемые триграммы в английских текстах распределены в следующей последовательности:

THE AND THA ENT ION TIO FOR NDE HAS NCE EDT TIS OFT STH MEN.

Сохранение частотных свойств исходных текстов в шифротекстах является основой для простейшего рода атак для взлома подобных криптосистем, что является свидетельством низкой криптостойкости простейших подстановочных криптосистем.

2.6. Омофонные шифраторы

Система ***омофонов*** (***гамофонов***) обеспечивает простейшую защиту от криптографических атак, основанных на анализе частот появления символов в шифротексте. Система омофонов является одноалфавитной хотя при этом каждый символ исходного текста имеет несколько подстановочных элементов - омофонов.

Предварительно каждому символу исходного текста ставится в соответствие несколько подстановочных элементов – омофонов. Их количество прямо пропорционально частоте использования данного символа в исходных текстах. Далее, при шифровании каждый символ исходного текста заменяется омофоном, выбранным из множества омофонов, соответствующих данному конкретному символу. Процедура выбора омофона из множества возможных значений для данного символа должна выполняться случайным образом равновероятно.

Пример 2.12. Предположим, что в качестве омофонов для английских букв использованы двухзначные целые числа, лежащие в диапазоне между 00 и 99. Количество омофонов для конкретного символа алфавита (например, английский алфавит) исходных текстов выбирается пропорционально относительной частоте букв английского языка в исходных текстах, кроме того, один и тот же омофон используется как подстановочный элемент только для одной буквы английского языка. Возможное сопоставление целых двухзначных чисел омофонов выбранным буквам английского языка приведено в следующей таблице

A 23, 25, 97, 95, 89, 33, 12, 11, 34
G 44, 77, 35, 51
C 87, 41
H 59, 90, 00, 26, 36
O 66, 02, 15, 22, 09, 83, 54

P 04, 58
R 38, 07, 94, 30, 56, 67
T 55, 71, 72, 80, 01, 12, 29, 50, 68
Y 88

Рис.2.8. Соответствие английских букв множествам омофонов

Тогда для исходного текста *M*=CRYPTOGRAPHY возможный вариант шифротекста имеет вид *C*= 87 07 88 58 72 54 51 30 97 04 00 88.

Улучшение качества омофонного шифратора достигается увеличением количества омофонов, используемых при шифровании, однако здесь необходимо отметить, что это приводит к усложнению процедур шифрования и дешифрования.

2.7. Шифратор Билла

Шифратор Билла (***Beale Cipher***) является классическим примером омофонного шифратора. Ключом данного шифратора является Декларация Независимости США. В данном случае используется гипотеза о среднестатистической природе текста представляющего собой Декларацию Независимости США. Предполагается, что частота использования букв в данном документе равняется частоте использования английских букв в любом другом тексте независимо от их местоположения в слове. Текст Декларации записывается построчно по десять слов в строке. Количество слов определяется требованиями качества омофонного шифратора. Большее их число позволяет более полно нивелировать частотные свойства исходного текста. Предположим, используется 1000 строк, тогда первые десять строк будут выглядеть следующим образом.

001 When, in the course of human events, it becomes necessary
011 for one people to dissolve the political bands, which have
021 connected them with another, and to assume among the Powers
031 of the earth the separate and equal station to which
041 the Laws of Nature and of Nature’s God entitle them,
051 a decent respect to the opinions of mankind requires that
061 they should declare the causes which impel them to the
071 separations. We hold these truths to be self-evident, that
081 all men are created equal, that they are endowed by
091 there Creator with certain unalienable rights; that among
… …

Рис.2.9. Ключ шифратора Билла

Билл зашифровывал каждый символ в сообщении исходного текста, подставляя номер слова Декларации, которое начиналось с того же символа.

Символ *W*, например, может быть зашифрован омофонами 001, 019, 040, 066, 072.

Пример 2.13. Предположим, исходный текст состоит из одного слова *M*=PLAIN. В результате шифрования получим: *C* =013 042 081 008 044.

2.8. Биграммный шифр Плейфейра

Шифр Плейфейра (***Playfair Cipher***), предложенный в 1854 году, является наиболее известным биграммным шифром замены. Данный шифр применялся в Великобритании во время первой мировой войны.

В отличие от всех ранее рассмотренных шифров, шифр Плайфейра основан на шифровании одновременно не одного, а двух символов, что также позволяет видоизменить частотные зависимости характерные для исходных текстов.

Основой данного шифра является шифрующая таблица со случайно расположенными буквами алфавита исходных текстов. Такая таблица представляет собой сеансовый ключ. Для удобства запоминания шифрующей таблицы подобно, как и для подстановочных таблиц используется ключевое слово или фраза, записываемая при заполнении начальных строк таблицы.

Для случая английского языка шифрующая таблица задается матрицей

Ключ шифра Плэйфэйра

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

5×5, состоящей из 25 позиций. Позиция для символа *J* соответствует позиции для символа *I* .

Процедура шифрования включает следующие этапы.

1.Исходный текст *M* разбивается на пары символов *M=m1m2*… (биграммы).

2.Последовательность биграмм *m1m2*... открытого текста *M* преобразуется с помощью шифрующей таблицы в последовательность биграмм *c1c2*… шифротекста *C* по следующим правилам.

2.1.Если буквы биграммы исходного текста *m1* и *m2* находятся в одной и той же строке шифрующей матрицы, то *c1* и *c2* представляют собой два символа справа от *m1* и *m2*, соответственно. Здесь первый столбец матрицы, является столбцам справа по отношении к последнему столбцу. Например, если *m1m2=EH* то *c1c2=FD.*

2.2.Если *m1* и *m2* находятся в одном и том же столбце, то *c1* и *c2* принимают значения, соответствующие символам ниже *m1* и *m2*, соответственно. Первая строка считается строкой ниже последней. Например, если *m1m2= FW* то *c1c2=QR.*

2.3.Если *m1* и *m2* находятся в различных строках и столбцах, то *c1* и *c2* соответствуют двум другим углам прямоугольника, имеющего *m1* и *m2*, в качестве двух исходных углов, при этом *c1* находится в той же строке что и *m1*, а *c2* находится в той же строке что и *m2*. Например, если *m1m2=KT*, то *c1c2=CQ* как видно из следующей диаграммы

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

2.4.Если *m1=m2* тогда пустой (нулевой) символ (например, *X*) вставляется в исходный текст между *m1* и *m2*, чтобы устранить равенство *m1=m2*. Например, если *m1m2=KK*, тогда *m1m2m3=KXK* и соответственно *c1c2=DW*.

2.5.Если исходный текст имеет нечетное число знаков, пустой символ добавляется в конец текста для получения четного числа символов исходного текста.

Пример 2.14. Исходный текст *M*=CRYPTOGRAPHY. В результате шифрования согласно алгоритма Плейфейра получим *C*=KOANZCFMIVDO.

Следует отметить, что шифрование биграммами существенно повышает стойкость шифров к взлому, однако частотные свойства распределения биграмм по-прежнему является ключом для злоумышленника.

С целью упрощения процедур шифрования и дешифрования была предложена модификация данного метода путем использования четырех шифрующих матриц. Причем две матрицы (второй и четвертый квадрант) используются для задания символов исходного текста, а матрицы первого и третьего квадранта для получения символов шифротекста. Подобная модификация предполагает меньшее число правил по сравнению с шифратором Плэйфэйра.

Пример 2.15. Предположим, шифрующие матрицы имеют следующий вид.

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

При шифровании диаграммы *m1m2=HW* получим *c1c2=TD*.

2.9. Расщепленный шифр

Идея ***расщепленного шифра*** (***Bifid cipher***) заключается в представлении каждого символа исходного текста в виде совокупностей частей этого символа, которые легко выделяются при шифровании. Затем для получения символа шифротекста используются части нескольких, как правило, последовательных символов исходного текста. Данная идея была предложена Деластелом (***Delastelle***). Суть его идеи состояла в последовательном использовании расщепления и объединения при получении шифротекста.

Подобно, как и для методологии шифрования Плэйфэйра, основой данного метода является шифрующая матрица, представляющая собой ключ шифра. Это предполагает использование номеров координат строк и столбцов для получения шифротекста. Выбрав в качестве шифрующей матрицы прямоугольник, состоящий из пяти строк и пяти столбцов, пронумеруем их последовательными номерами. В результате получим:

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

Тогда каждой букве английского алфавита можно поставить в соответствие двухзначное целое число, которое представляется двумя частями, а именно первой и второй цифрой, соответственно номером строки и номером столбца.

При шифровании исходный текст представляется блоками, состоящими из нескольких символов. Например, исходный текст *M*=THIS IS MY SECRET MESSAGE представляется в виде блоков состоящих из пяти символов, то есть *M*=THISI SMYSE CRETM ESSAG E. Далее каждому символу ставится в соответствие его код, состоящий из двух частей на основании шифрующей матрицы, приведенной выше. В результате получим.

T	H	I	S	I
1	1	5	5	5
1	4	5	3	5

S	M	Y	S	E
5	2	4	5	3
3	3	4	3	5

C	R	E	T	M
4	1	3	1	2
1	5	5	1	3

E	S	S	A	G
3	5	5	4	4

E
3

5	3	3	5	2		5

Приведенная выше таблица кодировки используется для получения кодов символов шифротекста. Первоначально цифры кодов считываются по строкам в рамках каждого блока. Тогда для первого блока получим 1155514535, для второго 5245333435, третьего 4131215513, четвертого 3554453352 и для пятого 35. Далее, используя для первого блока ту же шифрующую матрицу, получим следующие символы *T*=11, *I*=55, *F*=51, *A*=45, *E*=35. Отметим, что при получении символа *T* шифротекста использовались две первые цифры (части) кодов символов *T* и *H* исходного текста. Данное правило справедливо для всех символов шифротекста. Каждый из них получается из частей двух символов исходного текста. Окончательный результат шифрования в соответствии с расщепленным шифром принимает следующий вид *C*=TIFAEBAOQECNLIVEDAOBE.

Весьма эффективной модификацией ***расщепленного шифра*** (***Bifid cipher***) в котором символы исходного текста представляются двумя расщепляемыми частями, является ***Trifid cipher*** шифратор, для которого каждый символ исходного текста расщепляется на три части.

Как и в оригинальном методе, данная модификация базируется на шифрующей матрице, являющейся ключом при шифровании и дешифрировании. Каждый символ в подобной матрице кодируется тремя цифрами 1, 2 и 3. Трехзначные числа, задаваемые тремя цифрами, позволяют закодировать 3×3×3=27 символов, аналогично, как и для оригинального метода, двузначные числа, задаваемые пятью цифрами, позволили закодировать 5×5=25 символов. Пример подобной матрицы имеет следующий вид.

W 111	N 211	C 311	A 112	E 212	X 113	K 113	Q 213	I 313
M 121	O 221	T 321	& 122	V 222	J 322	B 123	R 223	F 323
Z 131	L 231	U 331	Y 132	P 232	G 332	H 133	S 233	D 333

Аналогично оригинальному методу и в данном случае исходный текст разделяется на блоки, состоящие из символов исходного текста. В данном случае длина блока не должна делиться нацело на три. Выберем длину блока равную семь, тогда исходный текст будет закодирован следующим образом.

T	H	I	S	I	S	M
3	1	3	2	3	2	1
2	3	1	3	1	3	2
1	3	3	3	3	3	1

Y	S	E	C	R	E	T
1	2	2	3	2	2	3
3	3	1	1	2	1	2
2	3	2	1	3	2	1

M	E	S	S	A	G	E
1	2	2	2	1	3	2
2	1	3	3	1	3	1
1	2	3	3	2	2	2

Очевидно, что при кодировании использовалась шифрующая таблица являющаяся ключом. Для получения символов шифротекста используются

три последовательные цифры в строке для каждого блока. Так первый блок и последующие блоки отобразятся в символы шифротекста следующим образом 313-I, 232-P, 123-B, 131-Z, 321-T, 333-D, 331-U, 122-&, 322-J, 333-D, 112-A 122-&, 321-T, 321-T, 122-&, 213-Q, 221-O, 331-U, 311-C, 233-S, 222-V. Соответственно *C*=IPBZTDU&JDA&TT&QOUCSV.

Расщепленный шифр считается одним из наиболее стойких ручных (***pencil-and-paper ciphers***) шифров. Его модификации по-прежнему вызывают интерес у любителей кроссвордов и головоломок.

2.10. Шифры, использующие коды переменной длины

Большие возможности криптографии часто использовались разведками разных стран для шифрования секретной информации, передаваемой их резидентами. Следующий метод шифрования (***The Straddling Checkerboard***) использовался разведкой СССР. Суть данного метода основана на использование так называемых ***префиксных кодов*** или кодов с переменной длиной. Подобно уже рассмотренным ранее шифрам, основой данного метода также является шифрующая матрица. На основании данной матрицы каждому символу исходного текста ставится в соответствие цифровой код, причем конкретный символ может быть зашифрован одной либо двумя десятичными цифрами.

Шифрующая матрица состоит из десяти столбцов и трех строк. Для нумерации столбцов используются все десять десятичных цифр. Их последовательность случайна. В первой строке записывается только восемь символов исходного текста. Символы A, T, O, N, E, S, I, R которые соответствуют ключевой фразе “At One Sir” являются ключом для построения остальной части матрицы. Примером такого ключа является следующая матрица.

	9	8	2	7	0	1	6	4	3	5
	A	T		O	N	E		S	I	R
2	B	C	D	F	G	H	J	K	L	M
6	P	Q	U	V	W	X	Y	Z	.	/

В первой строке матрицы пустыми (без символов) остаются только две позиции которые могут быть любыми из десяти возможных. В последующие строки матрицы последовательно записываются символы алфавита исходных текстов, которые не использовались в ключевой фразе. Для нумерации второй и третьей строки матрицы используются цифры соответствующие номерам столбцов с отсутствующими символами.

Структура шифрующей матрицы определяет алгоритм шифрования. Так, символы первой строки кодируются одной цифрой, а символы второй и третьей строки кодируются двумя цифрами - номером строки и номером столбца.

Пример 2.16. Предположим, исходный текст имеет вид *M*=SEND

MONEY. Тогда в результате шифрования в соответствии с ранее полученной шифрующей матрицей получим *C*=410222570166.

Очевидно, самым безопасным из простейших (***pencil-and-paper ciphers***) шифров является ***VIC шифр,*** разработанный в Советском Союзе, по крайней мере, одному из его агентов. Это был шифр, в котором сообщение было написано на найденной в 1953 году в Нью-Йорке мальчиком, продавцом газет части микрофильма.

Для использования данного шифра агент должен владеть тремя частями секретного ключа для шифрования и дешифрирования сообщений.

Агент должен был запомнить.

1.Шесть цифр, которые задаются в виде даты.

2.Первые 20 символов ключевой фразы, которые были началом популярной песни.

3.Пять случайных цифр для использования их в качестве указателя сообщения либо, что есть то же самое, сеансового ключа.

Пусть в качестве даты будет выбрано 4 июля 1776, тогда в цифровой форме имеем число 741776. В качестве случайных пяти цифр возьмем 77651. Ключевая фраза "I dream of Jeannie with t…"была выбрана как начальные слова популярной песни. Все три части ключа далее используются для генерирования шифрующей матрицы, которая для каждого сеанса будет уникальной, отличной от других в силу различного указателя сообщения.

Получение шифрующей матрицы состоит из следующих основных этапов:

1.На первом шаге от пяти случайных цифр указателя сообщения вычитаются пять старших цифр даты.

	7	7	6	5	1
(-)	7	4	1	7	7
	—	—	—	—	—
	0	3	5	8	4

При вычитании переносы между разрядами не учитываются.

2.На втором шаге на основании фразы ключа из 20 символов получаются 20 цифр. Первоначально фраза делится на две половины, и в пределах каждой половины, порядковый номер 1 назначается символу, который раньше всего встречается в алфавите, порядковый номер 2 - следующему символу и так далее. Порядковый номер 0 назначается символу, который встречается последним в упорядоченном алфавитном порядке символов. Таким образом, для фразы-ключа "I dream of Jeannie with t" мы имеем:

I	D	R	E	A	M	O	F	J	E
6	2	0	3	1	8	9	5	7	4

A	N	N	I	E	W	I	T	H	T
1	6	7	4	2	0	5	8	3	9

3.На данном шаге результат первого шага расширяется до десяти цифр

путем ***цепного сложения*** (***chain addition***). Начиная с группы цифр (в этом случае пяти, а позже мы выполним ту же самую операцию с группой из десяти цифр), последовательно складываем по две цифры, начиная с первых двух, записывая результат сложения следующей цифрой результирующего числа.

При выполнении операции сложения переносы игнорируются. В результате получим.

					0+3=	3+5=	5+8=	8+4=	4+3=
0	3	5	8	4	3	8	3	2	7

4.Результат третьего этапа 0 3 5 8 4 3 8 3 2 7 суммируется с первой частью ключевой фразы представленной в виде числа 6 2 0 3 1 8 9 5 7 4. Здесь переносы также игнорируются. В результате получим 6 2 0 3 1 8 9 5 7 4 + 0 3 5 8 4 3 8 3 2 7 = 6 5 5 1 5 1 7 8 9 1.

5.Используя предыдущий результат, полученный в пункте 4, и вторую часть ключевой фразы получается цифровой код из десяти цифр согласно следующей диаграмме

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*0*
(2)	1	6	7	4	2	0	5	8	3	9
(4)	6	5	5	1	5	1	7	8	9	1
	0	***2***	***2***	***1***	***2***	***1***	***5***	***8***	***3***	***1***

Первый символ 0 определяется значением второй строки 0 которое соответствует индексу 6 указанному в первой строке. Второй и третий символы результата принимает значение 2, так как значение 2 во второй строке имеет индекс 5. Результат преобразования 0 2 2 1 2 1 5 8 3 1.

6.Используя результат предыдущего этапа и применяя операцию цепного сложения получим 50 новых псевдослучайных значений. В результате будем иметь: 0 2 2 1 2 1 5 8 3 1 * 2 4 3 3 3 6 3 1 4 3 * 6 7 6 6 9 9 4 5 7 9 * 3 3 2 5 8 3 9 2 6 2 * 6 5 7 3 1 2 1 8 8 8 * 1 2 0 4 3 3 9 6 6 9.

7.Последние десять псевдослучайных цифр используем для организации перестановок десятичных цифр. В результате на основании 1 2 0 4 3 3 9 6 6 9 получим 1 2 0 5 3 4 8 6 7 9. Преобразование начинается с цифры 1 и далее по возрастанию с последним нулем. По сути, преобразование заключается в исключении повторяющихся цифр. Так в коде 1 2 0 4 3 3 9 6 6 9 первой повторяющейся цифрой была цифра 3, второй дубликат которой, заменяется следующей цифрой 4. Тогда последующие цифры исходного кода увеличиваются на единицу вплоть до следующей повторяющейся цифры 6.

8.Окончательный результат 1 2 0 5 3 4 8 6 7 9 используется для кодирования столбцов шифрующей матрицы.

	1	2	0	5	3	4	8	6	7	9
	A	T		O	N	E		S	I	R
0	B	C	D	F	G	H	J	K	L	M
8	P	Q	U	V	W	X	Y	Z	.	/

Полученная таким образом матрица используется для шифрования и дешифрирования передаваемых сообщений.

2.11. Метод диаграммы Порта

Метод ***диаграммы (таблицы) Порта*** (***Giovanni Baptista della Porta***) был предложен в 565 году. Данный метод основан на использовании таблицы Порта и ключевого слова. Таблица Порта имеет следующий вид.

Таблица 2.1.

Таблица Порта

AB	A	B	C	D	E	F	G	H	I	J	K	L	M
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
CD	A	B	C	D	E	F	G	H	I	J	K	L	M
	Z	N	O	P	Q	R	S	T	U	V	W	X	Y
EF	A	B	C	D	E	F	G	H	I	J	K	L	M
	Y	Z	N	O	P	Q	R	S	T	U	V	W	X
GH	A	B	C	D	E	F	G	H	I	J	K	L	M
	X	Y	Z	N	O	P	Q	R	S	T	U	V	W
IJ	A	B	C	D	E	F	G	H	I	J	K	L	M
	W	X	Y	Z	N	O	P	Q	R	S	T	U	V
KL	A	B	C	D	E	F	G	H	I	J	K	L	M
	V	W	X	Y	Z	N	O	P	Q	R	S	T	U
MN	A	B	C	D	E	F	G	H	I	J	K	L	M
	U	V	W	X	Y	Z	N	O	P	Q	R	S	T
OP	A	B	C	D	E	F	G	H	I	J	K	L	M
	T	U	V	W	X	Y	Z	N	O	P	Q	R	S
QR	A	B	C	D	E	F	G	H	I	J	K	L	M
	S	T	U	V	W	X	Y	Z	N	O	P	Q	R
ST	A	B	C	D	E	F	G	H	I	J	K	L	M
	R	S	T	U	V	W	X	Y	Z	N	O	P	Q
UV	A	B	C	D	E	F	G	H	I	J	K	L	M
	Q	R	S	T	U	V	W	X	Y	Z	N	O	P
WX	A	B	C	D	E	F	G	H	I	J	K	L	M
	P	Q	R	S	T	U	V	W	X	Y	Z	N	O
YZ	A	B	C	D	E	F	G	H	I	J	K	L	M
	O	P	Q	R	S	T	U	V	W	X	Y	Z	N

Рис.2.10. Таблица Порта для английского языка

Для шифрования сообщения (например, *M*=LOOK UNDER THE COUCH) необходимо использовать ключ (например, *K*= JACKET) таким образом, чтобы буквы ключа последовательно записывались под буквами исходного текста как это показано на диаграмме.

L	O	O	K	U	N	D	E	R	T	H	E	C	O	U	C	H
J	A	C	K	E	T	J	A	C	K	E	T	J	A	C	K	E

На следующем шаге необходимо использовать таблицу Порта, чтобы получить зашифрованное сообщение. Символы ключевого слова (*JACKET* в нашем примере), используются для определения строки, в которой выбирается символ шифруемого сообщения. В примере выше *J* – первый символ ключевого слова. Таким образом, необходимо определить местонахождение *J* на левой части таблицы Порта. Как видно, *J* находится в 5-ой строке таблицы Порта. В этой же строке в правой части находим шифруемый символ *L*, а ниже под этим символом будет находиться символ шифротекста *U*. Результирующий шифротекст имеет вид *C*=UBCS JJZRF LSVYBIXS

2.12. Шифратор Виженера

Система шифрования Виженера (***Vigenere Cipher***) впервые была опубликована в 1586 году и является одной из старейших и наиболее известных много алфавитных систем шифрования.

Шифратор Виженера, подобно как и шифр Порта, по своей сути использует развитие идеи Цезаря. Отличается только тем, что число реальных таблиц подстановки зависит от длины криптографического ключа и используемого алфавита. Важной особенностью шифратора Виженера является чередование использования таблиц подстановки в зависимости от последовательности символов используемого ключа. Этот шифр много алфавитной подстановки можно описать таблицей шифрования называемой ***таблицей (квадратом) Виженера***. Ниже приведен пример таблицы Виженера.

Таблица 2.2.

Таблица Виженера

	a	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***	***k***	***l***	***m***	***n***	***o***	***p***	***q***	***r***	***s***	***t***	***u***	***v***	***w***	***x***	***y***	***z***
a	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
b	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a
c	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b
d	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c
e	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d
f	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
g	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f
h	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g
i	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h
j	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i
k	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j
l	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k
m	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l
n	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m
o	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n
p	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
q	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
r	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q
s	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r
t	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
u	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
y	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
w	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v
x	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w
y	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x
z	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y

Рис.2.11. Таблица Виженера для английского языка

В таблице Виженера каждая строка представляет собой циклически сдвинутую на один символ предыдущую строку таблицы таким образом, что каждая строка по своей сути является таблицей подстановки шифратора Цезаря для конкретного значения ключа.

Таблица Виженера используется для шифрования и дешифрирования. Верхняя строка таблицы Виженера используется для задания символов исходных текстов, а левый столбец для задания символов криптографического ключа.

При шифровании исходного сообщения его записывают в строку, а под ним ключевое слово либо фразу. Если ключ оказался короче исходного текста, то его циклически повторяют необходимое число раз. На каждом шаге шифрования в верхней строке таблицы Виженера находят очередную букву исходного текста, а в левом столбце - очередное значение символа ключа. В результате очередная буква шифротекста находится на пересечении столбца определенного символом исходного текста и строки, соответствующей строке символа ключа.

При шифровании слова *M*=CRYPTOGRAPHY по методу Виженера для

ключа RAND предварительно исходный текст и ключевое слова запишем в виде двух строк.

C	R	Y	P	T	O	G	R	A	P	H	Y
R	A	N	D	R	A	N	D	R	A	N	D

Тогда первая буква исходного текста *C* определяет третий столбец таблицы Виженера а буква *R* ключа семнадцатую строку таблицы на пересечении которых находится символ шифротекста *T*. Окончательный результат шифрования имеет вид *C*=TRLSKOTURPUB.

Различают три возможных варианта использования криптографического ключа: прямое использование (***Straight Keyword***); прогрессивный ключ (***Progressive Key***); самогенерирующийся ключ (***Auto Key***). Рассмотрим использование всех трех вариантов для исходного сообщения "*Wish you were here*".

Пример 2.17. Для шифрования сообщения "Wish you were here", используем ключ SIAMESE. Тогда для первого случая прямого использования ключа получим:

M=	W	I	S	H	Y	O	U	W	E	R	E	H	E	R	E
K=	S	I	A	M	E	S	E	S	I	A	M	E	S	E	S
C=	O	Q	S	T	C	G	Y	O	M	R	Q	L	W	V	W

Идея использования прогрессивного ключа заключается в циклическом сдвиге символов ключа на одну позицию в упорядоченном алфавите символов при повторном применении ключа. Тогда для ключа SIAMESE при повторном его использовании по прогрессивной схеме имеем TJBNFTF, а при третьем UKCOGUG, и так далее.

Пример 2.18. Для шифрования сообщения "Wish you were here", используем ключ SIAMESE и прогрессивную схему его применения получим.

M=	W	I	S	H	Y	O	U	W	E	R	E	H	E	R	E
K=	S	I	A	M	E	S	E	T	J	B	N	F	T	F	U
C=	O	Q	S	T	C	G	Y	P	N	S	R	M	X	W	Y

В случае самогенерирующегося ключа в качестве его последующих символов используется исходный текст.

Пример 2.19. Для шифрования сообщения "Wish you were here", используем самогенерирующийся ключ SIAMESE. В результате имеем.

M=	W	I	S	H	Y	O	U	W	E	R	E	H	E	R	E
K=	S	I	A	M	E	S	E	W	I	S	H	Y	O	U	W

C= O Q S T C G Y S M J L F S L W

Модификациями метода Вижинера могут быть алгоритмы, в которых используются другие алфавиты, в том числе и различные алфавиты для исходных текстов и криптографических ключей, а также другой принцип генерирования строк шифрующей таблицы.

2.13. Роторные машины

Простейший шифр замены использует единственную таблицу подстановки, что приводит к тому, что частотные свойства исходного текста трансформируются без изменений в шифротекст. Развития простейшего шифра замены, рассмотренные выше, позволяют несколько снизить указанный недостаток, однако они оказались далеки от практического использования.

Первым методом, заслуживающим внимания при практическом использовании в настоящее время, является метод ***роторных машин***, который реально использует огромное количество различных таблиц подстановки.

Основная идея метода роторных машин лежит в многократных либо много алфавитных заменах. В 1568, Олберти (***Alberty***) издал рукопись, описывающую шифрующий диск, который определял многократные замены. Диск устанавливал соответствие *n* (где *n* - как пример может быть числом английских символов) символам исходного текста на внешнем диске (кольце) ротора возможные замены символами зашифрованного текста на внутреннем диске ротора. В зависимости от взаимного положения дисков, которое меняется путем поворота одного из дисков по отношению к другому, задаются различные таблицы подстановки. Очень важным развитием, полученным Олберти, безусловно, является динамическое изменение таблиц подстановки в процессе шифрования. Эта идея оказалась весьма конструктивной для практических систем шифрования.

В начале XX века были изобретены электромеханические устройства шифрования позволившие автоматизировать процесс шифрования. Принцип работы таких машин был основан на много алфавитной замене символов исходного текста в соответствии с идеями Олберти. В литературе широко описаны американская машина SIGABA (M-134), английская TYPEX, японская PURPLE и наиболее известная немецкая ENIGMA.

Главной деталью роторной машины является ***ротор*** (или диск) с электропроводящими перемычками внутри. На каждой стороне диска расположены равномерно по окружности *n* электрических контактов, где *n* – число символов алфавита исходных текстов. Каждый контакт на передней стороне диска соединен с одним из контактов на внутренней стороне. В результате электрический сигнал, соответствующий символу исходных текстов, будет заменен символом шифротекста. Соответствие символов исходных текстов символам шифротекстов определяется таблицей

подстановки и реализуется в роторе внутренними электропроводящими перемычками. Коммутация перемычек может быть изменена путем изменения соединений перемычек. Для фиксированной коммутации один ротор позволяет сгенерировать *n* таблиц подстановки путем поворота одного диска ротора по отношению к другому.

Роторная машина, как правило, состоит из нескольких роторов и механизма изменения положения дисков ротора с каждым зашифрованным символом. Механизм движения роторов может быть различным как по направлению, так и по количеству позиций на которое изменяется положение дисков ротора. Простейшим принципом движения является принцип одометра, использованный в немецкой машине ENIGMA. При шифровании машиной ENIGMA одного символа исходного текста правый крайний ротор поворачивается на одну позицию. Когда данный ротор (и любой другой) переместится на *n* позиций и совершит полный оборот ротор, расположенный справа от него, перемешается на одну позицию. Таким образом, для каждого символа исходного текста роторная машина будет иметь определенное состояние роторов, которому соответствует своя таблица подстановки. Подобным образом последовательно роторная машина принимает все возможные состояния, число которых для *m* роторов и *n* символов ровняется n^m. Для английского алфавита и, например, двух роторов в роторной машине число возможных состояний роторной машины или что, то же самое, таблиц подстановки равняется $n^m=26^2$. Учитывая то, что количество возможных таблиц подстановки в случае английского алфавита ровняется ≈26! количество роторов роторной машины, как правило, не превышает десяти. Попытка упростить реализацию роторной машины привела к повторному использованию одного и того же ротора при шифровании одного символа исходного текста. Структура подобной роторной машины приведена на следующем рисунке. В данном случае роторная машина состоит из трех роторов *R1, R2, R3* и рефлектора (*Reflec.*) (отражателя). В результате данная машина реально использует шесть роторов при шифровании каждого символа исходного текста.

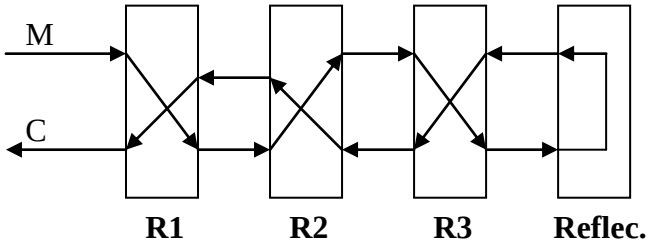


Рис.2.12. Роторная машина

Ключом роторной машины могут быть различные ее параметры, например, такие как:

- количество роторов и их порядок подключения.

- коммутация каждого ротора, определяющая его уникальную таблицу подстановки.

- механизм перемещения дисков роторов.

Развитием классической роторной машины может быть ее электронная версия, где основным элементом является запоминающее устройство (ЗУ), которое выполняет функции ротора. Содержимое ячеек запоминающего устройства определяется двоичными кодами символов шифротекста, а адреса соответствуют кодам символов исходных текстов. Для случая, когда алфавиты шифротекстов и исходных текстов тождественны и представляют собой 256 символов ASCII кодов, запоминающее устройство будет содержать 256 ячеек, каждая из которых имеет восемь бит. Во все ячейки ЗУ записываются ASCII коды в случайной последовательности таким образом, что каждому адресу ЗУ соответствует его ячейка с конкретным ASCII кодом. По своей сути такое ЗУ реализует простейшую таблицу подстановки, когда коды символов исходных текстов представляют собой адреса ЗУ, а содержимое ячеек - коды символов шифротекстов. Структурная схема электронного ротора приведена на рис.2.13.

Запоминающее устройство (RAM)	
Адрес ячейки ЗУ	Ячейка ЗУ
0	01010101
1	10010011
2	11000011
…	…
255	00011101

Рис.2.13. Электронный ротор

Приведенный ротор представляет собой статическую структуру, в которой отсутствует механизм вращения. Реализация данного механизма возможна на базе операции суммирования кода адреса ЗУ (символа шифротекста) с некоторой изменяющейся последовательностью двоичных кодов. Например, для случая равновероятных псевдослучайных последовательностей чисел (ПСЧ) будем иметь реализацию механизма сдвига на случайное количество позиций.

На рис. 2.14 приведен пример возможной структуры роторной машины состоящей из одного ротора.

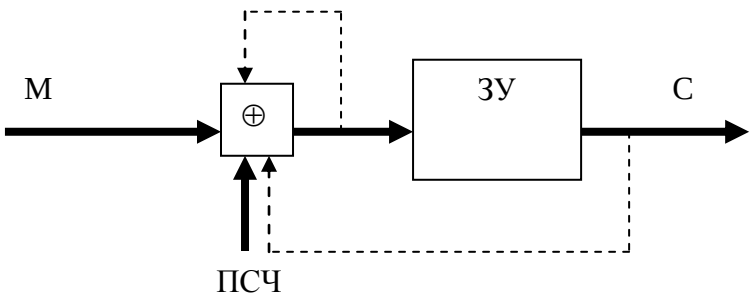


Рис.2.14. Электронная роторная машина

На основании ASCII кода символа исходного текста M и текущего значения кода псевдослучайного числа формируется адрес ЗУ как их поразрядная сумма по модулю два. Возможно использование большего числа аргументов для получения значения адреса ЗУ, а именно предыдущего адреса ЗУ и/или предыдущего значения символа шифротекста. Этот случай показан на рисунке пунктирной линией. Далее в соответствии со сформированным таким образом адресом считывается значение ячейки ЗУ, что и будет представлять собой код символа шифротекста.

В заключение следует отметить, что роторные машины позволяют достичь хорошего сочетания трех главных требований предъявляемых к криптографическим системам, а именно:

1.Высокой криптостойкости.

2.Высокого быстродействия.

3.Простаты реализации как программной, так и аппаратной.

2.14. Шифратор Вернама

Шифратор Вернама (***Gilbert Vernam***), или что есть то же самое, шифратор типа бегущий ключ был предложен Вернамом в 1917. Основная идея данного метода заключается в использовании ключа такой же длины, как и длина исходного текста и простейшего алгоритма шифрования, например поразрядной операции сложения по модулю кодов символов исходных текстов с кодами символов ключа. Данный шифратор часто называется одноразовым шифротором или шифратором одноразового блокнота (***one time pad***).

В качестве ключа может быть использована последовательность детерминированных символов полученных, например, из некоторой известной книги или любого другого документа доступного отправителю и получателю сообщения.

Пример 2.20. Предположим, что в качестве ключа будем использовать текст из учебника “Cryptography and Data Security”, который начинается с начала второго параграфа, а в качестве исходного текста “THE TREASURE IS BURIED...” тогда в процессе шифрования по методу Вижинера получим.

M =	t	h	e	t	r	e	a	s	u	r	e	i	s	b	u	r	i	e	d
K =	t	h	e	s	e	c	o	n	d	c	i	p	h	e	r	i	s	a	n
C =	m	o	i	l	v	g	o	f	x	t	m	x	z	f	l	z	a	e	q

Для общего случая пусть $M=m_1m_2...$ обозначает последовательность бит исходного текста, а $K=k_1k_2...$ последовательность бит ключа. Тогда в соответствии с шифром Вермана шифротекст будет иметь вид

$C=E_K(M)=c_1c_2...$, где $c_i=(m_i \oplus k_i)$ **mod** $2, i=1,2,...$.

Шифр Вернама может быть эффективно реализован с использованием последних достижений микроэлектроники, на базе элементарной операции “исключающее ИЛИ”. В соответствии с указанной операцией, каждой паре исходный текста/ключ (m_i/k_i) ставиться в соответствие символ шифротекста $c_i=m_i \oplus k_i$. Так как $k_i \oplus k_i = 0$ для любого значения $k_i=0$ или 1, дешифрирование выполняется аналогичным образом: $c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i$.

Как будет показано далее, шифр Вернама является наиболее близким приближением к абсолютно секретной криптосистеме.

Глава 3. Элементы математических основ криптографии
3.1. Введение в теорию чисел

Все множество ***натуральных чисел*** состоит их двух подмножеств: ***действительных чисел*** и ***целых чисел.*** Наиболее часто используемым подмножеством в криптографии и различных приложениях по защите информации является подмножество целых чисел. В свою очередь целые числа делятся на ***простые*** и ***составные*** числа.

Целое число d является делителем n тогда и только тогда, когда это число может быть представлено в виде произведения, то есть $n=kd$, и обозначается как $d \mid n$.

Целое число p, $p>1$ является ***простым***, если его делителями являются только 1 и p. Последовательность простых чисел начинается с чисел 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113,….

Натуральные целые числа, имеющие больше двух делителей, называются ***составными***. Таким образом, все натуральные целые числа делятся на простые и составные числа.

Любое целое $n>1$ может быть представлено единственным образом как произведение простых чисел в соответствующих степенях. Представление натурального целого числа в виде произведения простых чисел называется ***каноническим разложением Евклида***, а процедура получения такого разложения ***разложением на простые сомножители*** или ***факторизацией*** числа.

На настоящий момент не известны полиномиальные алгоритмы факторизации чисел, хотя и не доказано, что таких алгоритмов не существует. На этом факте базируется большинство из существующих криптосистем.

Весьма значимым с точки зрения криптографических приложений является факт того, что не известен эффективный алгоритм разложения чисел на множители, кроме того, не было получено никакой конструктивной нижней оценки временной сложности такого разложения. Более того, не известно никаких эффективных методов даже в таком простом случае, когда необходимо восстановить два простых числа p и q на основании их произведения $n=pq$. Единственной альтернативой является последовательное деление числа n на все простые числа. Поэтому получение простых чисел и в особенности больших простых чисел является одной из первоочередных задач криптографии.

Для оценки этой проблемы приведем несколько тривиальных теорем.

Теорема 3.1. ***(Евклида)*** Существует бесконечное множество простых чисел.

Доказательство: Предположим, что это множество конечно и состоит из простых чисел $p1, p2, p3,\ldots,pk$, тогда получили противоречие, заключающееся в том, что число

$$\left(\prod_{i=1}^{k} p_i\right)+1,$$

не делится ни на одно простое число *p1, p2, p3,…,pk*, тогда как оно делится на 1 и на самого себя, а это значит, что это число простое. #

Важным выводом приведенной теоремы является тот факт, что простых чисел существует бесконечное множество.

Теорема 3.2. Для сколь угодно большого положительного целого числа *k>1*, на числовой оси существует *k* последовательно идущих друг за другом составных чисел.

Доказательство: Число *(k+1)!=2×3×4 × … ×(k+1)* делится на любое из следующих чисел *2,3,4,…,(k+1).* Тогда числа, следующие последовательно в числовом ряду целых чисел *(k+1)!+2, (k+1)!+3, (k+1)!+4,…, (k+1)!+(k+1),* являются составными числами вследствие того факта, что первое число делится, по крайней мере, на *2,* второе на *3* и т. д. #

Приведенная теорема свидетельствует о сложности нахождения простых чисел, так как их удельный вес по сравнению с составными числами является несравненно меньшим.

Реальный подсчет простых чисел в каждой сотне целых чисел для интервала от 1 до 1000, то есть в первой сотне от 1 до 100, во второй сотне от 101 до 200 и так далее равен: 25, 21, 16, 16, 17, 14, 16, 14, 15, 14. Аналогичный подсчет для интервала от 1 000 001 до 1 001 000 равен: 6, 10, 8, 8, 7, 7, 10, 5, 6, 8. Еще меньше простых чисел в интервале от 10 000 001 до 10 001 000, а именно: 2, 6, 6, 6, 5, 4, 7, 10, 9, 6.

Количество целых чисел для произвольного интервала может быть оценено с помощью следующей теоремы.

Теорема 3.3. Отношение количества простых чисел $\pi(x)$ находящихся в интервале от *2* до *x* к величине равной *x/ln(x)* стремиться к единице при *x* стремящемся к бесконечности, то есть

$$\lim_{x\to\infty} \frac{\pi(x)}{x/\ln(x)} = 1,$$

где *ln(x)* есть натуральный логарифм от *x.*

Количественно приведенная теорема иллюстрируется таблицей.

x	$\pi(x)$	*x/ln(x)*	$\pi(x)/(x/\ln(x))$
1 000	168	145	1,159
10 000	1 229	1 086	1,132
100 000	9 592	8 686	1,104
1 000 000	78 498	72 382	1,084
10 000 000	664 579	620 421	1,071
100 000 000	5 761 455	5 428 681	1,061
1 000 000 000	50 847 476	48 254 942	1,054

Один из первых алгоритмов, позволяющих уменьшить сложность определения простоты целых чисел, был предложен еще во времена Евклида и базируется на использовании следующей теоремы.

Теорема 3.4. Если целое число $n(n>1)$ не делится ни на одно из простых чисел не большее чем $(n)^{1/2}$, то это число есть простое число.

Доказательство: Пусть n есть составное число, и соответственно может быть выражено как произведение двух сомножителей $n=ab$, где $1<a<n$; $1<b<n$. Числа a и b не могут быть больше, чем $(n)^{1/2}$ одновременно. #

Алгоритм получения простых чисел описывается следующей теоремой.

Теорема 3.5. (Эратосфена)

1) Если в наборе целых чисел $2,3,4,..,N$, удалить все числа, которые делятся на первые r простых чисел $2,3,5,7,\ldots,p_r$, тогда первое (наименьшее) не удаленное число будет простым.

2) Если в наборе целых чисел $2,3,4,..,N$, удалить все числа, которые делятся на простые числа меньшие или равные $(N)^{1/2}$, тогда все оставшиеся числа будут простыми числами p принадлежащими интервалу $(N)^{1/2}<p\leq N$.

Доказательство:

1) Любое составное число n делится, по крайней мере, на одно простое число меньшее, чем n.

2) Каждое составное число n, такое что $(N)^{1/2}< n\leq N$ делится, по крайней мере, на одно простое $p_i\leq(n)^{1/2}\leq(N)^{1/2}$ то есть на одно из чисел 2, 3,…, p_r ($p_r<(N)^{1/2}\leq p_{r+1}$) и, следовательно, будет вычеркнуто. #

Пример 3.1. В качестве примера рассмотрим пятьдесят первых чисел числового ряда, то есть

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49											

В результате удаления всех чисел кратных 2, 3, 5 и 7 получим множество простых чисел: 11,13,17,19,23,29,31,37,41,43,47, которое удовлетворяет вышеприведенной теореме.

Попытки определить формальным образом простые числа приводили к отрицательному результату.

Так простые числа ***Эйлера*** (***Euler's***) могут быть сгенерированы в соответствие с формулой x^2-x+41, для целых чисел x принадлежащих интервалу, $0<x<40$. Очевидно, что для больших значений x данная формула неприменима.

Простые числа ***Ферма*** (***Fermat's***): $3,5,17,257,65537$, генерируются в соответствии с формулой 2^K где $K=2^k$, для целых значений k:

Простые числа ***Мерсена*** (***Mercen's***) могут быть сгенерированы по формуле 2^n-1, для простых чисел $n=2,3,5,7,13,17,19,31,61$. Интерес к числам Мерсена не ослабевает.

Наибольшее известное на данный момент простое число $M=2^{25964951}-1$ содержит 7816230 десятичных цифр. Это 42-е известное простое число Мерсенна было найдено 18 февраля 2005 года в проекте по распределённому поиску простых чисел Мерсенна GIMPS.

Предыдущее по величине известное простое число $M=2^{24036583}-1$ (из 7235733 десятичных цифр). Это 41-е простое число Мерсенна, также было найдено GIMPS 15 мая 2004.

Числа Мерсенна выгодно отличаются от остальных наличием эффективного ***теста простоты***, носящего имя ***Люка-Лемера***. Благодаря этому тесту простые числа Мерсенна давно удерживают рекорд как самые большие известные простые числа. За нахождение простого числа, состоящего из более чем 10^7 десятичных цифр, назначена награда в 100000 долларов США.

Составные числа могут быть представлены в канонической форме

$$a = \prod_{i=1}^{k} p_i^{\alpha_i}$$

где p_i - простые числа, α_i - целое число, а a - составное число.

Пример 3.2. $a=120=2^3\cdot 3^1\cdot 5^1$

Общим делителем чисел $a_1,a_2,a_3,\ldots,a_n$ является целое число d, такое, что $d \mid a_1,\ d \mid a_2,\ d \mid a_3,\ldots,\ d \mid a_n$.

Наибольший общий делитель чисел $a_1,a_2,a_3,\ldots,a_n$ это наибольший целый делитель d, который может быть поделен любым общим делителем этих чисел $d=(a_1,a_2,a_3,\ldots,a_n)$.

Пример 3.3. $(6,15,27)=3$.

Теорема 3.6. Если

$$a_1 = \prod_{i=1}^{k} p_i^{\alpha_i} \qquad a_2 = \prod_{i=1}^{k} p_i^{\beta_i} \quad \ldots \quad a_n = \prod_{i=1}^{k} p_i^{\gamma_i},$$

тогда наибольший общий делитель (НОД) будет равен:

$$(a_1, a_2, \ldots, a_n) = \prod_{i=1}^{k} p_i^{\min(\alpha_i,\beta_i,\ldots,\gamma_i)}.$$

Пример 3.4. Если $6=2^1\cdot 3^1$, $15=3^1\cdot 5^1$, $27=3^3$, тогда НОД$(6,15,27)=$ $=2^{\min\{1,0,0\}}\cdot 3^{\min\{1,1,3\}}\cdot 5^{\min\{0,1,0\}}=2^0\cdot 3^1\cdot 5^0=3$.

Теорема 3.7. Если

$$a_1 = \prod_{i=1}^{k} p_i^{\alpha_i} \qquad a_2 = \prod_{i=1}^{k} p_i^{\beta_i} \quad \ldots \quad a_n = \prod_{i=1}^{k} p_i^{\gamma_i},$$

тогда наименьшее общее кратное (НОК) будет равно:

$$НОК(a_1, a_2, \ldots, a_n) = \prod_{i=1}^{k} p_i^{\max(\alpha_i,\beta_i,\ldots,\gamma_i)}.$$

Пример 3.5. Если $6=2^1\cdot3^1$, $15=3^1\cdot5^1$, $27=3^3$, тогда НОК$(6,15,27)=$ $=2^{max\{1,0,0\}}\cdot3^{max\{1,1,3\}}\cdot5^{max\{0,1,0\}}=2^1\cdot3^3\cdot5^1=270$.

3.2. Алгоритм Евклида

Проблема нахождения наибольшего общего делителя является одной из часто используемых процедур в криптографии, которая в частности позволяет определить взаимную простоту целых чисел. Одним из исторически первых инструментов определения взаимной простоты является хорошо известный алгоритм Евклида. Методологической основой указанного алгоритма является следующая теорема.

Теорема 3.8. Если $a=bq+r$, тогда наибольший общий делитель чисел a,b равен наибольшему общему делителю чисел b,r, то есть $(a,b)=(b,r)$.

Доказательство:

Пусть $d=(a,b)$, тогда из утверждения $d \mid a$ и $d \mid b$ можно сделать вывод, что d является делителем bq, а также делителем разности чисел a и bq таким образом d является делителем $a-bq=r$.#

Теорема 3.9. (Алгоритм Евклида) Для любых целых чисел $a>0$ и $b>0$ таких, что $a>b$, и b не является делителем a, для некоторого s, существуют целые числа $q_0,q_1,q_2,\ldots,q_s$ и $r_1,r_2,\ldots,r_s$ такие, что $b>r_1>r_2>\ldots>r_s>0$ и $a=bq_0+r_1$, $b=r_1q_1+r_2$, $r_1=r_2q_2+r_3,\ldots,$ $r_{s-2}=r_{s-1}q_{s-1}+r_s$, $r_{s-1}=r_sq_s$, и соответственно $(a,b)=r_s$

Доказательство последней теоремы следует из последовательного применения теоремы 3.8 для следующих пар чисел (a,b), (b,r_1), $(r_1,r_2),\ldots,$ (r_{s-1},r_s), для которых выполняется равенство $(a,b)=(b,r_1)=(r_1,r_2)=\ldots=(r_{s-1},r_s)$.

В виде вычислительной процедуры алгоритм Евклида можно представить следующим кодом.

```
Алгоритм Эвклида
begin
        g0:=a;
        g1:=b;
        while  gi≠0 do
        begin
                gi+1:= gi-1 mod gi;
                i:=i+1;
        end
        gcd:= gi-1          {gcd-Greatest Common Divisor (НОД)}
end
```

Пример 3.9. Определим НОД для целых чисел *1173* и *323*, *(1173,323) =?*. Так как $1173=323\cdot3+204$ то НОД для целых чисел *1173* и *323* равен НОДу для чисел *323* и *204*. Повторяя подобное разложение чисел $323=204\cdot1+119;$

204=119 ·1+85; 119=85·1+34; 85=34·2+17; 34=17·2 окончательно получим, что *(1173,323) =17.*

В соответствии с приведенным выше алгоритмом Евклида получим:

gi+1	***:=***	***gi-1 mod gi;***
204	*:=*	*1173 mod 323*
119	*:=*	*323 mod 204*
85	*:=*	*204 mod 119*
34	*:=*	*119 mod 85*
17	*:=*	*85 mod 34*
0	*:=*	*34 mod 17*

Одним из существенных недостатков алгоритма Евклида является применение операции деления для его реализации, что заметно увеличивает его вычислительную сложность. Развитием алгоритма Евклида является ***бинарный алгоритм.*** Этот алгоритм является расширением алгоритма Евклида и основан на следующих очевидных утверждениях:

1. Если и *a* и *b* чётные числа, тогда *(a,b)=2(a/2,b/2);*
2. Если *a* чётное, а *b* нечётное число, то *(a,b)=(a/2,b);*
3. В соответствии с теоремой 3.8 *(a,b)=(b,a-b);*
4. Если и *a* и *b* нечётные числа, то *a-b* является чётным числом.

Пример 3.10. Определим НОД для целых чисел *1173* и *323, (1173,323) =?.* Последовательно применяя утверждения, приведенные выше, получим следующие равенства *(1173,323)=(323,850)=(323,425)=(323,102)=(323,51)= (51,272)=(51,136)=(51,68)=(51,34)=(51,17)=(17,34)=(17,17)=17.* Отметим, что в данном примере от проблемы определения НОД для чисел *(1173,323)* перешли к такой же проблеме для чисел *(323,850)* на основании утверждения 3. А последующий переход выполнен на основании утверждения 2 и так далее.

Числа $a_1,a_2,a_3,\ldots,a_n$ называются ***взаимно простыми*** тогда и только тогда, когда $(a_1,a_2,a_3,\ldots,a_n)=1$.

Числа $a_1,a_2,a_3,\ldots,a_n$, называются ***попарно взаимно простыми*** тогда и только тогда, когда для любых *i* и $j\neq i$ $(a_i,a_j)=1$.

<u>Теорема 3.10.</u> Если *(a,b)=1,* тогда для любых целых чисел *n* и *m* $(a^n,b^m)=1$, справедливо также и обратное утверждение, если $(a^n,b^m)=1$, для любых целых чисел *n* и *m* то выполняется равенство *(a,b)=1.*

Доказательство: Предположим, что выполняется равенство *(a,b)=1.* Тогда, если в каноническом представление числа $a=p_1^{\alpha 1},p_2^{\alpha 2},\ldots,p_k^{\alpha k}$ для некоторого показателя степени αi выполняется неравенство $\alpha i>0$, то $\gamma i=0$ для канонического представления числа $b=p_1^{\gamma 1},p_2^{\gamma 2},\ldots,p_k^{\gamma k}$. Это следует из равенства *(a,b)=1.* Аналогично в случае, когда $n\alpha i>0$ получим $\gamma i=0$. #

3.3. Сравнения

Два целых числа a и b ***сравнимы*** (***конгруэнтны***) ***по модулю*** натурального числа m, если при делении на m они дают одинаковые остатки. Другими словами, a и b ***сравнимы по модулю*** m, если их разность $a - b$ делится на m. Например, 32 и 39 сравнимы по модулю 7, так как $32 = 7 \cdot 4 + 4$, $39 = 7 \cdot 5 + 4$. Утверждение a и b сравнимы по модулю m записывается в виде:

$$a \equiv b \ \mathbf{mod}\ m.$$

Последнее выражение называется ***сравнением (конгруэнцией)***.

Пример 3.11. Следующие равенства являются сравнениями $32 \equiv 5 \bmod 9$; $48 \equiv 12 \bmod 9$; $17 \equiv 7 \bmod 5$.

Числа a и b ***несравнимы по модулю*** m, если их разность $a - b$ не делится на m. Этот факт обозначается неравенством $a \not\equiv b \bmod m$. Очень часто для представления сравнения $a \equiv b \bmod m$ используют символ равенства (=), тогда $a = b \bmod m$ означает, что a и b сравнимы по модулю m, а $a \neq b \bmod m$, что не сравнимы по модулю m.

В случае, когда в сравнении $a = b \bmod m$ величина $b < m$, b называется ***вычетом*** a по модулю m.

Для сравнений существует несколько полезных лемм:

Лемма 3.1. Если $a \equiv b$ **mod** m, тогда для любого целого k будет справедливо $ka \equiv kb$ **mod** m.

Лемма 3.2. Если $ka \equiv kb$ **mod** m, и $(k,m)=1$, то $a \equiv b$ **mod** m.

Лемма 3.3. Если $ka \equiv kb$ **mod** km, где k и m любые целые числа, то $a \equiv b$ **mod** m.

Лемма 3.4. Если $a \equiv b$ **mod** m, и $c \equiv d$ **mod** m, то $a + c \equiv b + d$ **mod** m.

Лемма 3.5. Если $a_1 \equiv b_1$ **mod** m, и $a_2 \equiv b_2$ **mod** m,…, $a_n \equiv b_n$ **mod** m, то $a_1 + a_2 + a_3 + \ldots + a_n \equiv b_1 + b_2 + b_3 + \ldots + b_n$ **mod** m.

Лемма 3.6. Если $a \equiv b$ **mod** m, и $c \equiv d$ **mod** m, то $a \cdot c \equiv b \cdot d$ **mod** m.

Лемма 3.7. Если $a_1 \equiv b_1$ **mod** m, и $a_2 \equiv b_2$ **mod** m,…, $a_n \equiv b_n$ **mod** m, то $a_1 \cdot a_2 \cdot a_3 \cdot \ldots \cdot a_n \equiv b_1 \cdot b_2 \cdot b_3 \cdot \ldots \cdot b_n$ **mod** m.

Лемма 3.8. Если $a \equiv b$ **mod** m, то для любого целого $k > 0$ будет справедливо $a^k \equiv b^k$ **mod** m.

Теория сравнений эффективно используется для выполнения различных вычислений при реализации криптографических алгоритмов. Основой для реализации вычислительных процедур является так называемая ***модулярная арифметика***.

Модульная арифметика основана на следующей классической лемме, используемой во многих технических приложениях.

Лемма 3.9. *(a∗b)* **mod** *m = [(a* **mod** *m)∗(b* **mod** *m)]* **mod** *m*, где ∗ это любая из следующих операций *"+", "-"* или *"×"*.

Приведенная лемма показывает, что вычисление *(a∗b)* **mod** *m* в модульной арифметике даёт тот же результат, что и в случае обычной целочисленной арифметике, однако существенно упрощает вычисление результата *(a∗b)* **mod** *m.*

Пример 3.12. *7·9* **mod** *5 =[(7* **mod** *5)·(9* **mod** *5)]* **mod** *5=3.*

Заметим, что принцип модульной арифметики также применяется и для возведения в степень, так как операция возведения в степень равносильна повторяющемуся выполнению операции умножения.

Пример 3.13. Рассмотрим выражение 3^5 **mod** *7*. Возможны три очевидных алгоритма получения искомого результата. Этот результат может быть вычислен путём возведения *3* в степень *5,* и затем получения результата по **mod** *7* или в соответствии с двумя алгоритмами приведенными ниже.

В первом случае число *3* последовательно умножается на предыдущий результат умножения до тех пор, пока не будет получено значение 3^5. Отметим, что для данного примера необходимо выполнить *4* операции умножения, а для общего случая a^z *mod m* необходимо выполнить *z-1* операцию умножения. Далее необходимо выполнить операцию деления для определения искомого значения 3^5 **mod** *7.* Отметим неоправданно высокую вычислительную сложность такого алгоритма и, в первую очередь, в части операции возведения в степень. Значительно меньшую вычислительную сложность можно достичь, используя быстрые алгоритмы возведения в степень.

Во втором случае, при использовании быстрых алгоритмов возведения в степень, искомый результат для 3^5 **mod** *7* может быть получен в следующей последовательности:

1. Величина *3* возводится в квадрат $3^2=3\times3=9.$
2. Используя значение 3^2, получаем $3^4=3^2\times3^2=9\times9=81.$
3. Вычисляется 3^5, как $3^5=3^4\times3=81\times3=243.$
4. Выполняя операцию деления, получим 3^5 **mod** *7=243* **mod** *7=5.*

В данном случае, для нашего примера 3^5 **mod** *7*, количество операций умножения равняется *3*, а для общего случая (a^z *mod m*) необходимо не более чем $2(\lceil \log_2 z \rceil - 1)$ операций умножения. Очевидным недостатком второго алгоритма является большая размерность операндов, которые могут быть существенно больше чем величина *m.*

Третий алгоритм основан на использовании основополагающей леммы 3.9. Все промежуточные вычисления будут выполняться по модулю *7* или для общего случая (a^z *mod m*) по модулю *m,* в следующей последовательности.

1. Величина *3* возводится в квадрат по модулю *7*, в результате получим 3^2 **mod** *7=3·3* **mod** 7=2

2. Для получения 3^4 **mod** 7 используем предыдущий результат, тогда, в соответствии с леммой 3.9, получим 3^4 **mod** $7=[(3^2 \textbf{mod} 7)\times(3^2 \textbf{mod} 7)] \textbf{mod} 7= 2\times 2$ **mod** $7=4$.

3. Для получения окончательного результата, выполним следующие вычисления 3^5 **mod** $7=[(3^4 \textbf{mod} 7)\times(3 \textbf{mod} 7)] \textbf{mod} 7= 4\times 3$ **mod** $7=5$.

Для общего случая (a^z mod m) последний алгоритм можно представить в виде следующей вычислительной процедуры.

Алгоритм быстрого возведения в степень

```
begin "возвращаем x=a^z mod m"
a1:=a; z1:=z;
x:=1;
while z1≠0 do "x(a1z1 mod m)=a^z mod m"
    begin
        while z1 mod 2=0 do
            begin "возводим в квадрат a1 пока z1 чётно"
            z1:= z1 div 2;
            a1:= (a1×a1) mod m;
            end;
        z1:= z1–1;
        x:=(x·a1 ) mod m "умножение"
    end;
    fastexp:=x;
end
```

Пример 3.14. Рассмотрим вычисление $x=5^{10}$ **mod** $7=5^{(1010)}$ **mod** 7 с использованием алгоритма быстрого возведения в степень. Здесь $10_{10}=1010_2$. В последующей диаграмме приведена последовательность вычислений в соответствии с указанным алгоритмом.

```
a1:=5; z1:=10; x:=1;
    z1≠0; (10≠0);
        z1 mod 2=0; (10 mod 2=0);
            z1 div 2=5; (10/2=5);
            a1:= a1·a1 mod m=4; (5·5 mod 7=4);
        z1 mod 2≠0; (5 mod 2≠0);
    z1:= z1–1=4; (5-1=4);
    x:= (x·a1 ) mod m =4; (1·4 mod 7=4);
    z1≠0; (4≠0);
        z1 mod 2=0; (4 mod 2=0);
            z1 div 2=2; (4/2=2);
            a1:= a1·a1 mod m=2; (4·4 mod 7=2);
        z1 mod 2=0; (2 mod 2=0);
            z1 div 2=1; (2/2=1);
```

$a_1 := a_1 \cdot a_1 \bmod m = 4;\ (2 \cdot 2 \bmod 7 = 4);$

$z_1 \bmod 2 \neq 0;\ (1 \bmod 2 \neq 0);$

$z_1 := z_1 - 1 = 0;\ (1-1=0);$

$z_1 = 0;\ (0=0);$

$x := (x \cdot a1\ ) \bmod m = 2;\ (4 \cdot 4 \bmod 7 = 2);$

Таким образом, результирующее значение $x = 5^{10}$ **mod** 7 ровняется 2.

3.4. Теорема Эйлера

С понятием сравнения тесно связана величина ***вычета*** $r\ (0<r<m)$, числа a по модулю m. ***Вычетом*** числа a по модулю m называется остаток от деления величины a на m. Согласно приведенному определению, если $a = r$ **mod** m, $(0<r<m)$, то $m \mid (a-r)$. Отсюда, справедливо равенство $a-r=qm$ или $a=qm+r$.

Если набор из m целых чисел $\{a_i\}=\{a_1,a_2,\ldots,a_m\}$ формирует полный набор чисел, сравнимых по ***модулю*** m с каждым значением r_i в системе вычетов $\{0,1,2,\ldots,m-1\}$, то этот набор $\{a_i\}$ называется ***полной системой вычетов по модулю m.*** Так, для любого целого a_i, выполняется сравнение $a_i = r_j \bmod m$ где r_j является уникальным значением неповторяющимся для остальных чисел множества $\{a_i\}$.

Пример 3.15. Набор целых чисел $\{16,12,19,48,65\}$ является полной системой вычетов по модулю 5. Действительно, $16=1 \bmod 5$, $12=2 \bmod 5$, $19=4 \bmod 5$, $48=3 \bmod 5$, $65=0 \bmod 5$ и мы получаем полный набор вычетов $\{0,1,2,3,4\}$.

Множество целых чисел $\{a_i\}=\{a_1,a_2,\ldots,a_n\}$ формирует ***класс вычета по модулю m,*** если все вычеты для данного множества одинаковы и равны r.

Пример 3.16. Набор целых чисел $\{16,21,56,91,106\}$ является классом вычета по модулю 5. Действительно, $16=1 \bmod 5$, $21=1 \bmod 5$, $56=1 \bmod 5$, $91=1 \bmod 5$, $106=1 \bmod 5$ и мы имеем одинаковый вычет 1 для всех чисел исходного множества.

Большое практическое значение в криптографии имеет известная теорема Ферма.

Теорема 3.11. **(Fermat).** Если p простое число и $(a,p)=1$, где a целое, тогда

$$a^{p-1} = 1 \textbf{ mod } p.$$

Доказательство: Пусть для заданных взаимно простых величин p и a, $(a,p)=1$ существует $p-1$ положительных произведений $a,2a,3a,\ldots,(p-1)a$. Тогда любая пара ia, ja $(i \neq j)$ этих произведений не сравнима по модулю p, то есть

$$ia \neq ja \bmod p,$$

что следует из леммы 3.2 для сравнений приведенной ранее. Следовательно, каждое произведение имеет свой уникальный ненулевой вычет r_i , $(1 \le r_i \le p-1)$. Эти вычеты $\{1,2,3,\ldots,(p-1)\}$ для множества чисел $\{a,2a,3a,\ldots,(p-1)a\}$. расположены в произвольном порядке и формируют полную систему вычетов $\{r_\alpha, r_\beta, r_\chi, \ldots, r_\lambda\}=\{1,2,3,\ldots,(p-1)\}$, где $a=r_\alpha \bmod p$, $2a=r_\beta \bmod p$, $3a=r_\chi \bmod p, \ldots, (p-1)a=r_\lambda \bmod p$.

На основе леммы 3.7 для сравнений $a=r_\alpha \bmod p$, $2a=r_\beta \bmod p$, $3a=r_\chi \bmod p, \ldots, (p-1)a=r_\lambda \bmod p$ получим $a\cdot 2a\cdot 3a\cdot\ldots\cdot(p-1)\cdot a=r_\alpha\cdot r_\beta \cdot r_\chi,\cdot\ldots\cdot r_\lambda \bmod p$. Учитывая, что $\{r_\alpha,r_\beta,r_\chi,\ldots,r_\lambda\}=\{1,2,3,\ldots,(p-1)\}$ и выполнив перестановку, будем иметь $a\cdot 2a\cdot 3a\cdot\ldots\cdot(p-1)\cdot a = 1\cdot 2\cdot 3\cdot\ldots(p-1) \bmod p$. Окончательно $a^{p-1}\cdot(p-1)!=(p-1)! \bmod p$. Поскольку $(p-1)!$ и p взаимно простые целые числа, то есть $((p-1)!,p)=1$, тогда окончательно получаем $a^{p-1}=1$ **mod** p.#

Отметим, что теорема Ферма справедлива только для простых чисел p. Обобщением данной теоремы является теорема Эйлера, которая основана на использовании функции Эйлера. Определим данную функцию для целого числа n.

Функцией Эйлера (Euler's) $\psi(n)$ целого числа $n\ge 1$ является количество целых чисел, которые меньше чем n и взаимно просты с n. Для небольших значений n эта функция принимает следующие значения $\psi(1)=0$, $\psi(2)=1$, $\psi(3)=2$, $\psi(4)=2$, $\psi(5)=4$, $\psi(6)=2$, $\psi(7)=6$, $\psi(8)=4$, $\psi(9)=6$, $\psi(10)=4$, $\psi(11)=10,\ldots$. Несложно показать, что согласно приведенному определению, если n есть простое число p, то $\psi(p)=p-1$.

Для функции Эйлера справедлив ряд утверждений и теорем. Одной и наиболее часто используемых в криптографии теорем относительно функции Эйлера является следующая теорема.

Теорема 3.12. Если $n=pq$, где p и q $(p\ne q)$ простые числа, то $\psi(n)=\psi(p)\psi(q)=(p-1)(q-1)$.

Доказательство: Имеем множество $\{0,1,2,\ldots,pq-1\}$ из pq целых чисел, которые меньше чем $n=pq$. Все эти числа взаимно простые с $n=pq$, за исключением $(p-1)$ чисел $\{q,2q,3q,\ldots,(p-1)q\}$, кратных q, $(q-1)$ чисел $\{p,2p,3p,\ldots,(q-1)p\}$, кратных p и 0. Тогда, $\psi(pq)=pq-(p-1)-(q-1)-1=pq-p-q+1=(p-1)(q-1)$.#

Пример 3.17. $\psi(10)=\psi(2\cdot 5)=\psi(2)\psi\cdot(5)=1\cdot 4=4$.

Теорема 3.13. Если p простое число, и $k>0$ целое число, то $\psi(p^k)=p^k-p^{k-1}=p^{k-1}(p-1)$.

Доказательство: Множество целых чисел, которые меньше чем p^k и не взаимно простые с p^k, включает числа $\{p,2p,3p,\ldots,(p^{k-1}-1)p\}$. Это значит, что среди p^k-1 чисел, меньших, чем p^k, исключая ноль, есть $p^{k-1}-1$ целых, не взаимно простых с p^k. Тогда, $\psi(p)=p^k-1-(p^{k-1}-1)=p^k-p^{k-1}$.#

Пример 3.18. $\psi(8)=\psi(2^3)=2^3-2^2=8-4=4$.

Теорема 3.14. Функция Эйлера является мультипликативной для произведения целых чисел, если эти числа являются взаимно простыми, то есть $\psi(n\cdot m)=\psi(n)\psi\cdot(m)$, если $(n,m)=1$.

Данная теорема позволяет сформулировать алгоритм вычисления функции Эйлера для произвольного целого числа *a*.

Первоначально представим число в виде канонического разложения Евклида $a=p_1^{\alpha 1} p_2^{\alpha 2}\dots p_r^{\alpha r}$, тогда в силу того что $p_i^{\alpha i}$ и $p_j^{\alpha j}$ являются взаимно простыми для любых $i\neq j$, используя предыдущую теорему получим $\psi(a)=\psi(p_1^{\alpha 1})\psi(p_2^{\alpha 2})\dots\psi(p_r^{\alpha r})=(p_1^{\alpha 1}-p_1^{\alpha 1-1})(p_2^{\alpha 2}-p_2^{\alpha 2-1})\dots(p_r^{\alpha r}-p_r^{\alpha r-1})=a(1-1/p_1)(1-1/p_2)\dots(1-1/p_r)$.

Пример 3.19. $\psi(2700)=?\ \ 270=2^2 3^3 5^2$. $\psi(2700)=2700(1-1/2)(1-1/3)(1-1/5)=720$.

Теорема 3.15. (Euler's). Если $n\geq 0$ положительное целое число, и $(a,n)=1$, где *a* целое, то

$$a^{\psi(n)}=1 \ \mathbf{mod}\ n.$$

Доказательство: Пусть $\{r_1,r_2,r_3,\dots,r_{\psi(n)}\}$ представляет собой приведенную систему вычетов по модулю *n*, тогда для $(a,n)=1$ числа ar_1, ar_2, $ar_3,\dots$, $ar_{\psi(n)}$ образуют ту же приведенную систему вычетов. То есть, $ar_1=r_\alpha \bmod n$, $ar_2=r_\beta \bmod n$, $ar_3=r_\chi \bmod n,\dots,ar_{\psi(n)}=r_\lambda \bmod p$, где $\{r_\alpha,r_\beta,r_\chi,\dots,r_\lambda\}$ это есть те же значения вычетов $\{r_1, r_2, r_3,\dots,r_{\psi(n)}\}$, переставленные в ином порядке. Используя аналогичный подход, который применялся при доказательстве теоремы 3.11, перемножив правую и левую части сравнений $ar_1=r_\alpha \bmod n$, $ar_2=r_\beta \bmod n$, $ar_3=r_\chi \bmod n$, $\dots, ar_{\psi(n)}=r_\lambda \bmod p$, получим: $a^{\psi(n)} r_1\cdot r_2\cdot r_3,\cdot\dots\cdot r_{\psi(n)}= r_\alpha\cdot r_\beta\cdot r_\chi,\cdot\dots\cdot r_{\psi(n)} \bmod n$. Учитывая, что $\{r_1,r_2, r_3,\dots,r_{\psi(n)}, m\}=1$, окончательно получим $a^{\psi(n)}=1 \ \mathbf{mod}\ n$.#

Пример 3.20. $3^{10} \bmod 11=?$. Согласно теореме Ферма $3^{10}=1 \bmod 11$, где $p=11$ есть простое число, и $a=3$ есть целое взаимно простое с $p=11$, то есть $(11,3)=1$.

Пример 3.21. $3^{12} \bmod 26=?$. Согласно теореме Эйлера $3^{12} \bmod 26=1$, где $n=26$, $\psi(26)=\psi(2\cdot 13)=\psi(2)\psi\cdot(13)=1\cdot 12=12$.

3.5. Линейные сравнения

Под линейным сравнением (линейной конгруэнцией) понимают выражение вида

$$ax=b \ \mathbf{mod}\ n,\ b<n,$$

где *a*, *b* и *n* есть целые числа, и $b<n$, а *x* $(x<n)$ неизвестное целое число, удовлетворяющее линейному сравнению.

Существует три возможных случая для линейного сравнения относительно его решения *x*, а именно линейное сравнение может: 1) не иметь решений; 2) иметь одно решение; 3) иметь множество решений, удовлетворяющих этому линейному сравнению.

Теорема 3.16. Если наибольший общий делитель d чисел a и n $(d=(a,n))$ не является делителем b, то линейное сравнение $ax=b$ **mod** n не имеет решений.

Доказательство: Предположим противоположное, что есть решение x_0, которое удовлетворяет линейному сравнению, то есть $ax_0=b$ **mod** n. Согласно теореме 3.16 d является делителем a и n, откуда следует, что d должен являться делителем ax_0 и nq, так же как и делителем $ax_0–nq=b$. Здесь q есть положительное целое число. Тогда получим противоречие: по условию теоремы d не является делителем b, а в случае наличия решения x_0 d должно является делителем b, значит, линейное сравнение $ax=b$ **mod** n не имеет решений, если $d=(a,n)$ не является делителем b. #

Пример 3.22. Линейное сравнение $2x=1 \ mod \ 4$ не имеет решения, так как наибольший общий делитель $d=2$ чисел $a=2$ и $n=4$ $(2=(2,4))$ не является делителем $b=1$. Действительно, ни одно из возможных целочисленных значений $x<4$ не удовлетворяет сравнению$2x=1 \ mod \ 4$.

Теорема 3.17. Если наибольший общий делитель d чисел a и n равен единице $(a,n)=1$, то есть, a и n являются взаимно простыми числами, то линейное сравнение $ax=b$ **mod** n имеет одно решение.

Доказательство: Предположим, что существует полная система вычетов $\{0,1,2,\ldots,n-1\}$ по модулю n. Тогда согласно тому, что a и n взаимно простые целые числа, множество чисел $\{0\cdot a, 1\cdot a, 2\cdot a,\ldots, (n-1)\cdot a\}$ образует полную систему вычетов по модулю n. Тогда среди всех целых чисел есть одно ax_0 и только одно с вычетом равным b.#

Пример 3.18. Линейное сравнение $2x=1 \ mod \ 3$ имеет одно решение $x_0=2$, так как $a=2$ и $n=3$ являются взаимно простыми числами.

Теорема 3.17 позволяет сформулировать задачу нахождения решения линейного сравнения для случая, когда a и n являются взаимно простыми числами. Здесь возможны два случая в зависимости от величины b.

Для $b=1$ линейное сравнение принимает вид $ax=1$ **mod** n, где неизвестная величина $x=a^{-1}$ является ***мультипликативной инверсной величиной*** по отношению к a. Тогда $aa^{-1}=1$ **mod** n.

Для вычисления мультипликативной инверсной величины возьмём два линейных сравнения: исходное сравнение $ax=1$ **mod** n в соответствии с условием теоремы 3.17 и сравнение $1=a^{\psi(n)}$ **mod** n соответствующее теореме Эйлера, затем, используя лемму 3.6, перемножим левую и правую части этих сравнений. Как результат получим $ax=a^{\psi(n)}$ **mod** n. Используя лемму 3.2, разделим правую и левую части последнего равенства $ax=a^{\psi(n)}$ **mod** n на a, получим

$$x=a^{\psi(n)-1} \ \textbf{mod} \ n,$$

что и является основным расчетным соотношением для вычисления мультипликативной инверсной величины. Для случая, когда n является простым числом $x=a^{n-2}$ **mod** n.

Пример 3.19. Найти решение линейного сравнения $3x=1 \ mod \ 7$.

Так как *7* это простое число, тогда $x=a^{n-2}$ **mod** $n=3^{7-2}$ **mod** $7=3^5$ **mod** $7=5$.

Пример 3.20. Найти решение линейного сравнения $4x=1 \bmod 9$.

Функция Эйлера целого числа $n=9$ вычисляется как $\psi(9)=6$, Тогда $x=a^{\psi(n)-1}$ **mod** $n=4^{6-1}$ **mod** $9=4^5$ **mod** $9=7$.

Для $b\neq 1$ линейное сравнение принимает вид $ax=b$ **mod** n, где неизвестная величина x будет вычисляться согласно соотношению

$$x=ba^{\psi(n)-1} \textbf{ mod } n,$$

а для случая, когда n является простым числом по формуле $x=ba^{n-2}$ **mod** n.

Пример 3.21. Найти решение линейного сравнения $3x=3 \bmod 7$.

Принимая во внимание, что $(3,7)=1$, а *7* есть простое число $x=ba^{n-2}$ **mod** $n=3\times 3^{7-2}$ **mod** $7=3^6$ **mod** $7=1$.

Теорема 3.18. Если наибольший общий делитель d чисел a и n является делителем числа b $(d \mid b)$, то существует d решений линейного сравнения вида $ax=b$ **mod** n.

Доказательство: Согласно условию теоремы, d является делителем a, n и b. Тогда из линейного сравнения $ax=b$ **mod** n получим $a_1dx=b_1d$ **mod** n_1d. Используя лемму 3.3, получим линейное сравнение вида $a_1x=b_1$ **mod** n_1, где $(a_1,n_1)=1$. Данное сравнение $a_1x=b_1$ **mod** n_1, имеет одно решение x_0. Целые числа того же класса по модулю n/d будут решениями для исходного сравнения $ax=b$ **mod** n. То есть $x_1=x_0$ **mod** n, $x_2=x_0+n/d$ **mod** n, $x_3=x_0+2n/d$ **mod** $n,\ldots,$
$x_d=x_0+((d-1)n)/d$ **mod** n.#

Пример 3.22. Найти решения для следующего линейного сравнения $6x=4 \bmod 10$.

Принимая во внимание, что $(6,10)=2$ и *2* это делитель *4*, получим сравнение $3x=2 \bmod 5$. Решением последнего сравнения будет $x_0=ba^{n-2}$ **mod** $n=2\times 3^{5-2}$ **mod** $5=2\times 3^3$ **mod** $5=4$. Тогда решениями сравнения $6x=4 \bmod 10$ будут $x_1=x_0$ **mod** $n=4$ **mod** $10=4$; $x_2=x_0+n/d$ **mod** $n=4+10/2$ **mod** $10=9$.

3.6. Элементы теории информации

Неоценимый вклад в криптографию внес основоположник теории информации Клод Шеннон (***K.Shannon***). В 1949 K.Shannon опубликовал свои теоретические исследования по криптографии, основанные на полученных им ранее результатах в теории информации.

Одним из главных результатов по криптографии можно считать его теоретическое обоснование возможности создания ***абсолютно секретных криптосистем***. Он определил теоретическую секретность шифра по неопределённости возможного исходного текста, на основании полученного шифротекста. Соответственно, если вне зависимости от того, сколько шифротекста перехвачено, нельзя получить никакой информации

относительно исходного текста, то шифр обладает ***идеальной секретностью.***

Теория информации измеряет количество информации в сообщении по среднему количеству бит, необходимых для оптимального кодирования всех возможных сообщений. Например, поле "пол" в базе данных содержит только один бит информации, потому что оно может быть закодировано одним битом (*Male* может быть представлено как “0”, *Female* как “1”). Если поле представлено ASCII символами, кодирующими строки символов *Male* и *Female*, это потребует больше места на носителях информации, но не будет содержать в себе большего объема информации.

Количество информации в сообщении формально измеряется ***энтропией*** сообщения, которая базируется на понятии ***количества информации***.

Пусть $X_1,...,\ X_n$ это n возможных сообщений, возникающих с вероятностями $p(X_1),...,p(X_n)$, сумма этих вероятностей $p(X_i),\ i=1,...,n$ равна 1. Тогда получение сообщения X_i можно оценить количеством полученной информации, которое вычисляется как $F(X_i)=-\log_2 p(X_i)=\log_2(1/p(X_i))$. Очевидно, что при получении сообщения вероятность которого крайне мала будет получено большое количество информации, что следует из выражения $\log_2(1/p(X_i))$ и наоборот для событий с большой вероятностью будет получено мало информации. Действительно, если $p(X_i)=1$ количество информации $F(X_i)=\log_2(1/p(X_i))=\log_2 1=0$. То есть, события, происходящие с вероятность единица, не дают никакой информации.

Под ***энтропией*** понимают среднее количество информации при получении одного из возможных сообщений. Численно энтропия представляет собой средневзвешенное количество информации и вычисляется по формуле.

$$H(X) = -\sum_{i=1}^{n} p(X_i)\log_2(p(X_i)) = \sum_{i=1}^{n} p(X_i)\log_2(1/p(X_i)).$$

Интуитивно, каждый элемент $\log_2(1/p(X_i))$ в последнем выражении представляет собой число бит, необходимых для оптимального кодирования сообщения X_i. Действительно, при оптимальном кодировании сообщения (события), а в нашем случае, например, символа исходного текста необходимо использовать меньшее число бит для кодирования часто встречающегося символа, а для редко встречающегося символа большее число бит. Тогда в среднем для кодирования сообщения, состоящего из множества символов, будет использовано оптимальное суммарное количество бит.

Поскольку $1/p(X)$ уменьшается при увеличении $p(X)$, оптимальное кодирование использует короткие коды для часто встречающихся сообщений за счёт использования длинных кодов для редких сообщений. Этот принцип применён в ***коде Морзе***, где наиболее часто используемые буквы представлены самыми короткими кодами.

Код Хаффмана является оптимальным кодом, ассоциированным с буквами, словами, машинными инструкциями или фазами. Односимвольный код ***Хаффмана*** часто используется для минимизации больших файлов.

Пример 3.23. Пусть $n=3$, и пусть три сообщения представлены событиями $A,B,$ и C, где $p(A)=1/2$ и $p(B)=p(C)=1/4$. Тогда $\log_2(1/p(A))=\log_2 2= 1;\ \log_2(1/p(B))=\log_2(1/p(C))=\log_2 4= 2,$ что подтверждает наши предыдущие наблюдения о том, что в часто встречающемся сообщении для оптимального кодирования нужно минимальное число бит.

Пример 3.24. Предположим, необходимо оптимально закодировать пол клиента в базе данных. Имеются две возможности (два события) *Male* и *Female* примерно с одинаковыми вероятностями $p(Male)=p(Female)=1/2$. Тогда значение энтропии будет вычисляться как

$$H(X)=p(Male)\log_2(1/p(Male))+ p(Female)\log_2(1/p(Female))=$$
$$=(1/2)(\log_2 2)+(1/2)(\log_2 2)=1,$$

что подтверждает наши предыдущие наблюдения о том, что в поле базы данных пол оптимально будет закодирован одним битом. С другой стороны независимо от того как закодирован пол клиента в базе данных он содержит в себе 1 бит информации.

Следующий пример иллюстрирует применение энтропии для определения содержания сообщения.

Пример 3.25. Пусть $n=3$, и пусть три сообщения представлены буквами $A,B,$ и C, где $p(A)=1/2,\ p(B)=p(C)=1/4$. Тогда $H(X)=(1/2)\log_2 2+2(1/4)\log_2 4 =0.5+1.0=1.5.$

Оптимальное кодирование может быть достигнуто с использованием однобитного кода для кодирования, A в силу того, что вероятность этого сообщения наибольшая, и двухбитных кодов для B и C. Например, A может быть закодировано битом 0, в то время как B и C могут кодироваться двумя битами каждый: 10 и 11. Применяя подобное кодирование, последовательность, состоящая из восьми букв ABCAABAC, кодируется, как 12-битная последовательность 010110010011 как показано ниже:

A	B	C	A	A	B	A	C
0	10	11	0	0	10	0	11

Среднее число бит на букву равно 12/8=1,5, что соответствует нашим предварительным наблюдениям. Действительно, при получении одного из трех сообщений $A,B,$ или C среднее ожидаемое количество информации равняется 1,5.

3.7. Частотный анализ

Достижения в теории информации позволили формальным образом исследовать исходные тексты, представленные на конкретном языке, и использовать эти результаты для взлома криптосистем. Одним из таких методов анализа является ***частотный анализ***. В соответствии с данным

методом, распределение букв в криптотексте сравнивается с распределением букв в алфавите исходного сообщения. Вероятность успешного вскрытия повышается с увеличением длины криптотекста.

Пусть для заданного языка определено множество сообщений длиной N символов, тогда ***частота (скорость) языка*** для сообщений X длиной N определяется как

$$r=H(X)/N,$$

где величина r определяет среднее число бит информации в каждом символе сообщения.

Простейший способ определения частоты языка (***абсолютной частоты R***), основывается на предположении, что все буквы алфавита языка имеют одинаковую вероятность появления во всех возможных сообщениях так же, как и всевозможные последовательности букв алфавита в сообщениях равновероятны. Если алфавит языка содержит L букв, тогда ***абсолютная частота*** может быть получена так:

$$R=\log_2 L,$$

Для английского языка $L=26$, тогда $R=\log_2 L=\log_2 26 =4,7$.

Абсолютная частота языка $R=4,7$ определяет максимальное число бит информации, которое может быть получено при получении одной буквы сообщения. Или, максимальное количество бит, необходимых для кодирования сообщения, представленного на английском языке.

Реальная частота английского языка значительно меньше абсолютной частоты. Это происходит потому, что английский язык, так же как и другие неискусственные языки, слишком многословный или, что то же самое, избыточный. Например, фраза “occurring frequently” ("часто появляющийся") может быть сокращена на 58% как “crng frg” без потери информации.

Существуют множество различных таблиц распределений букв в том или ином языке, но ни одна из них не содержит окончательной информации - даже порядок букв может отличаться в различных таблицах. Распределение букв очень сильно зависит от типа текста: проза, разговорный язык, технический язык и т.п. Наиболее часто встречающиеся распределения для английского и русского языка приведены в таблице 3.1 и таблице 3.2.

Таблица 3.1.

Частота букв английского языка

A	0.0804	B	0.0154	C	0.0306
D	0.0399	E	0.1251	F	0.0230
G	0.0196	H	0.0554	I	0.0726
J	0.0016	K	0.0067	L	0.0414
M	0.0253	N	0.0709	O	0.0760
P	0.0200	Q	0.0011	R	0.0612
S	0.0654	T	0.0925	U	0.0271
V	0.0099	W	0.0192	X	0.0019
Y	0.0173	Z	0.0009		

Таблица 3.2.

Частота букв русского языка

А	0.062	Л	0.053	Ц	0.004
Б	0.014	М	0.026	Ч	0.012
В	0.038	Н	0.053	Ш	0.006
Г	0.013	О	0.090	Щ	0.003
Д	0.025	П	0.023	Ы	0.016
Е	0.072	Р	0.040	Ь,Ъ	0.014
Ж	0.007	С	0.045	Э	0.003
З	0.016	Т	0.053	Ю	0.006
И	0.062	У	0.021	Я	0.018
Й	0.010	Ф	0.002		
К	0.028	Х	0.009		

Хотя нет таблицы, которая может учесть все виды текстов, но есть характерные черты общие для всех таблиц, например, в английском языке буква *E* всегда возглавляет таблицу частот встречаемости, а *T* идет на второй позиции. *A* и *O* почти всегда третьи. Кроме того, девять букв английского языка *E, T, A, O, N, I, S, R, H* всегда имеют частоту выше, чем любые другие. Эти девять букв занимают примерно 70% английского текста. Ниже приведены соответствующие таблицы для различных языков.

Таблица 3.3.

Частота встречаемости девяти букв в различных языках

Русский		Английский		Немецкий		Французский		Итальянский		Финский	
О	0.090	E	0.125	E	0.184	E	0.159	E	0.118	A	0.121
Е	0.072	T	0.092	N	0.114	A	0.094	A	0.117	I	0.106
А	0.062	A	0.080	I	0.080	I	0.084	I	0.113	T	0.098
И	0.062	O	0.076	R	0.071	S	0.079	O	0.098	N	0.086
Н	0.053	I	0.073	S	0.070	T	0.073	N	0.069	E	0.081
Т	0.053	N	0.071	A	0.054	N	0.072	L	0.065	S	0.078
С	0.045	S	0.065	T	0.052	R	0.065	R	0.064	L	0.059
Р	0.040	R	0.061	U	0.050	U	0.062	T	0.056	O	0.055
В	0.038	H	0.055	D	0.049	L	0.053	S	0.050	K	0.052
#	0.515	#	0.699	#	0.726	#	0.741	#	0.750	#	0.736

Используя частоту встречаемости букв английского алфавита как распределение вероятностей $p(X_i)$ для вычисления энтропии получим значение частоты языка равную $r=H(1\text{-grams})/1=4.15$. Как видим, реальные частоты встречаемости букв в английском тексте заметно уменьшили количество информации, которое они содержат.

Отметим, что в двух предыдущих случаях вычисления частоты языка была использована гипотеза отсутствия зависимости последовательностей букв в исходных текстах, хотя очевидно, что такая зависимость, безусловно, существует.

Известны статистические распределений частоты биграмм (двух символов) исходных текстов. Например, для английского языка такие сочетания букв как *TH* и *EN* возникают гораздо чаще, чем другие. Некоторые биграммы (например, OZ) никогда не возникают в сообщениях, содержащих смысловую информацию (исключение составляют акронимы). Используя подобное распределение для пар букв, частота английского языка будет иметь величину $r=H(2\text{-}grams)/2=3.62$.

Доля значащих (имеющих смысл) последовательностей букв любого языка понижается, когда длина последовательности увеличивается. Например, в английском языке можно встретить сочетание *BB* из двух букв *B,* и практически невозможно встретить триграммы *BBB.* Учитывая частоту распределения триграмм в английском языке, численное значение частоты языка примет значение $r=H(3\text{-}grams)/3=3.22$.

Частота языка (значение энтропии на один символ) будет иметь максимально близкую величину к истинному значению при использовании статистических результатов распределения N-грамм для возрастающих значений N. Когда N возрастает, энтропия на символ уменьшается, потому что для больших N количество сообщений, содержащих смысл, резко уменьшается по отношению к произвольному случайному набору из N букв. Показано, что в зависимости от структуры текста, использованного языка и других факторов реальное значении энтропии на один символ для больших значений N принимает значение из диапазона $r=1\div1,5$.

Как видно, реальное значение количества информации, содержащееся в одном символе сообщения заметно меньше абсолютной частоты $R=4,7$. С другой стороны, такое их соотношение свидетельствует об избыточности языков. Формально избыточность языка с частотой r и абсолютной частотой R определена как $D=R-r$. Для $R=4.7$ и частоты $r=1$, $D=3.7$, что свидетельствует о том, что английский язык на 79% избыточен; для $r=1.5$, $D=3.2$, предполагается избыточность на 68%.

3.8. Абсолютная секретность

Как уже отмечалось ранее, основным вкладом К.Шеннона в теорию и практику защиты информация явилось его теоретическое обоснование возможности создания абсолютно секретных криптосистем. В своем анализе криптографических систем он выделял три вида информации:

1.Исходные сообщения *M,* описываемые вероятностями *p(M)* появления сообщений *M,* где $\sum_{M} P(M) = 1$.

2.Зашифрованные сообщения *C*, описываемые вероятностями *p(C)* появления шифротекстов *C*, где $\sum_C P(C) = 1$.

3.Ключи *K*, описываемые вероятностями *p(K)* появления ключей *K*, где $\sum_K P(K) = 1$.

Если $P_C(M)$ есть вероятность того, что на основании исходного сообщения *M* было получено зашифрованное сообщение *C*, тогда согласно определению К.Шеннона ***абсолютная секретность*** будет достигнута тогда и только тогда, когда будет выполняться следующее равенство.

$$P_C(M)=P(M).$$

Здесь *P(M)* есть вероятность того, что на основании исходного сообщения *M* может быть получено любое другое зашифрованное сообщение. Приведенное равенство свидетельствует о том, что перехват зашифрованного сообщения *C* не даёт криптоаналитику никакой дополнительной информации об исходном сообщении.

Необходимым и достаточным ***условием абсолютной секретности*** является выполнение для каждого *C* следующего равенства

$$P_M(C)=P(C),$$

которое для конкретного зашифрованного сообщения *C* должно выполняться для всех *M*.

Условие абсолютной секретности $P_M(C)=P(C)$ означает, что вероятность $P_M(C)$ получения зашифрованного сообщения *C* при шифровании исходного сообщения *M* такая же, что и вероятность *P(C)* получения *C* при шифровании некоторого другого сообщения *M'* (шифрование *M'* выполнялось другим ключом).

Таким образом, абсолютная секретность возможна, в случае использования, по крайней мере, столько абсолютно случайных криптографических ключей, сколько существует возможных исходных сообщения, которые кодируются этими ключами.

Следующий рисунок показывает абсолютно секретную криптосистему с четырьмя равновероятными исходными сообщениями и четырьмя равновероятными ключами.

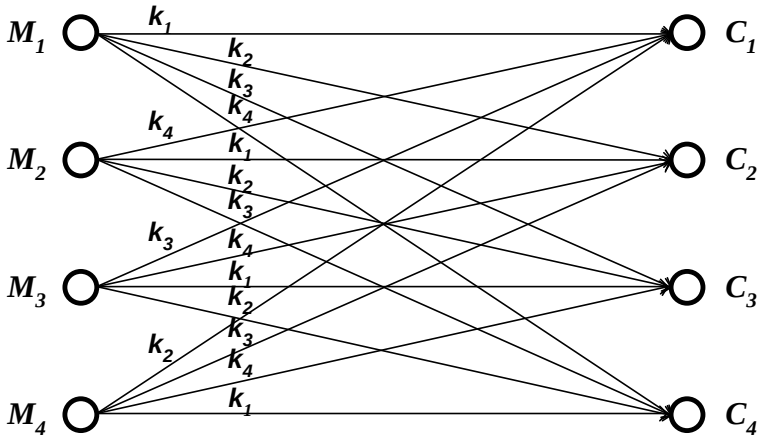


Рис.3.1. Абсолютно секретная криптосистема

Здесь $P_C(M)=P(M)=1/4$, и $P_M(C)=P(C)=1/4$ для всех M и C. Криптоаналитик перехватив одно из зашифрованных сообщений C_1 C_2 C_3 или C_4, не сможет определить какой из четырех возможных исходных текстов был зашифрован. Отправитель исходного сообщения мог использовать для шифрования один из четырех равновероятных ключей k_1, k_2, k_3 или k_4 ($p(k_1)=1/4$, $p(k_2)=1/4$, $p(k_3)=1/4$, и $p(k_4)=1/4$) и поэтому любое из сообщений M_1 M_2 M_3 или M_4 могло быть зашифровано.

Абсолютная секретность требует, чтобы количество ключей было, по крайней мере, столько же, сколько и возможных исходных сообщений. В противном случае, появятся некоторые исходные сообщения M такие, что для данного C не будет существовать ключа k_i, декодирующего C в M, а это значит $P_C(M)=0$. Таким образом, криптоаналитик может исключить некоторое возможное исходное сообщение из дальнейшего анализа, тем самым, повышая вероятность взлома шифрограммы.

Шифр, использующий неповторяющуюся случайную последовательность элементов ключа при его размерности равной размерности исходного сообщения, называется системой шифрования типа ***одноразовый блокнот***. Одноразовый блокнот это единственный шифр, позволяющий достичь абсолютной секретности.

Реализация системы шифрования типа одноразовый блокнот впервые была сформулирована Вернамом в 1917 (см. раздел 2.14). Пусть $M=m_1m_2...$ определяет бинарный поток исходного сообщения, и $K=k_1k_2...$ определяет поток бит ключа, тогда последовательность бит шифротекста $C=E_K(M)=c_1c_2...$, где $c_i=(m_i+k_i) \bmod 2$, $i=1,2,...$. Подобный шифр легко реализуется с использованием логической операции XOR для каждой пары значений исходный текст/ключ $c_i=m_i \oplus k_i$. Поскольку $k_i \oplus k_i =0$ для $k_i =0$ или 1, декодирование происходит той же операцией: $c_i \oplus k_i = m_i \oplus k_i \oplus k_i=m_i$.

Пример 3.26. $M=0111001101010101$, $K=0101011100101011$, здесь последовательность бит ключа представлена потоком случайных бит с вероятностями $p(0)=p(1)=0.5$.

Процедура шифрованияя:

$C=M \oplus K=0111001101010101 \oplus 0101011100101011=0010010001111110$.

Процедура дешифрованияя:

$M=C \oplus K=0010010001111110 \oplus 0101011100101011=0111001101010101$.

Очевидным недостатком абсолютно секретных систем типа система одноразового блокнота является проблема распределения ключей между пользователями.

Глава 4. Подстановочно перестановочные шифры
4.1. Стандарт шифрования данных DES

Стандарт шифрования данных (***Data Encryption Standard - DES***) в настоящее время является одним из наиболее популярных методов шифрования. Он часто используется на практике и широко обсуждается в литературе. Основой данного метода шифрования является совместное использование двух простейших операций, таких как подстановки и перестановки.

Идея комбинированного использования операций подстановки и перестановки принадлежит К.Шеннону и является средством для достижения высокого качества криптосистем, хорошей аппроксимации абсолютной секретности. Совместное применение подстановок и перестановок позволяет получить "комбинированное преобразование", которое произвольно распределяет исходные сообщения равномерно по набору всех возможных зашифрованных сообщений. Смешанные преобразования могут быть созданы, например, путём применения преобразования состоящего в последовательном применении перестановок и подстановок. Хорошее качество достигается путем итерационного применения идентичных подстановочно-перестановочных преобразований

Впервые этот подход реализован в виде шифра ***LUCIFER***, созданного на фирме IBM Фейстлом (***Feistl***). LUCIFER использует преобразование исходного сообщения, переменно применяя подстановки и перестановки. Структура данного алгоритма представлена на рис.4.1.

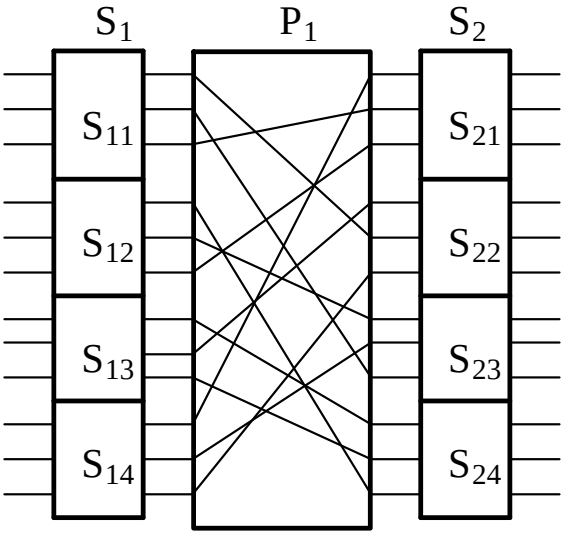


Рис.4.1. Структурная схема шифратора LUCIFER

Процедура подстановки шифруемого блока S_i из 12 бит состоит из 4 операций подстановки S_{i1} ,...,S_{i4}, меньшей размерности, каждая из которых управляет трехбитным блоком данных. Это необходимо чтобы уменьшить сложность операций подстановки, которая реализуется с помощью таблиц подстановки. Перестановка P_i переставляет (перемешивает) все двенадцать

бит, полученные как результат подстановки, в некоторой случайной последовательности. Например, для подстановки P_1 первый символ переставляется на пятую позицию, второй, на девятую, третий на вторую и так далее. Таким образом, последовательно выполняется несколько итераций для достижения хорошего качества шифрования.

В 1977 Национальное Бюро Стандартов США представило стандарт шифрования данных (***Data Encryption Standard - DES***) для использования в несекретных приложениях правительства США, а также различными неправительственными организациями. Алгоритм шифрования был разработан на фирме IBM и был производным от криптосистемы LUCIFER.

DES кодирует 64-битные блоки данных с использованием 56-битного ключа. Существует мнение, что 56-битный ключ являлся достаточно сильным на ранних стадиях использования стандарта, а в настоящее время его размерность недостаточна.

Алгоритм стандарта DES, который используется и для кодирования и для декодирования, в обобщенном виде представлен на рисунке 4.2. Входной блок $T= t_1t_2,...,t_{64}$ состоящий из 64 бит, предварительно перемешивается согласно исходной перестановке *IP*, $T_0=IP(T)$. После прохождения через 16 итераций функции *F*, он перемешивается согласно обратной перестановке IP^{-1} и формирует итоговый результат. Перестановки *IP* и IP^{-1} заданы в таблице 4.1.

Таблица 4.1.

Таблицы перестановки IP и IP^{-1}

перестановка *IP*	перестановка IP^{-1}
58 50 42 34 26 18 10 2	40 8 48 16 56 24 64 32
60 52 44 36 28 20 12 4	39 7 47 15 55 23 63 31
62 54 46 38 30 22 14 6	38 6 46 14 54 22 62 30
64 56 48 40 32 24 16 8	37 5 45 13 53 21 61 29
57 49 41 33 25 17 9 1	36 4 44 12 52 20 60 28
59 51 43 35 27 19 11 3	35 3 43 11 51 19 59 27
61 53 45 37 29 21 13 5	34 2 42 10 50 18 58 26
63 55 47 39 31 23 15 7	33 1 41 9 49 17 57 25

Эти таблицы читаются слева направо, сверху вниз. Например, *IP* перемещает $T=t_1t_2...t_{64}$ в $T_0=t_{58}t_{50}...t_7$. Обе таблицы фиксированы. Между начальной и последней перестановками алгоритм выполняет 16 итераций функции *F*, которая комбинирует подстановку и перестановку. Пусть T_i обозначает результат *i*-й итерации, а L_i и R_i обозначают левую и правую половины T_i соответственно, таким образом, $T_i = L_i R_i$, где $L_i = t_1t_2...t_{32}$, $R_i = t_{33}t_{34}...t_{64}$. Тогда $L_i=R_{i-1}$, $R_i = L_{i-1} \oplus F(R_{i-1},K_i)$, где $\oplus$ есть знак операции поразрядного сложения по модулю два (XOR), а K_i - это 48-битный ключ для *i*-й итерации.

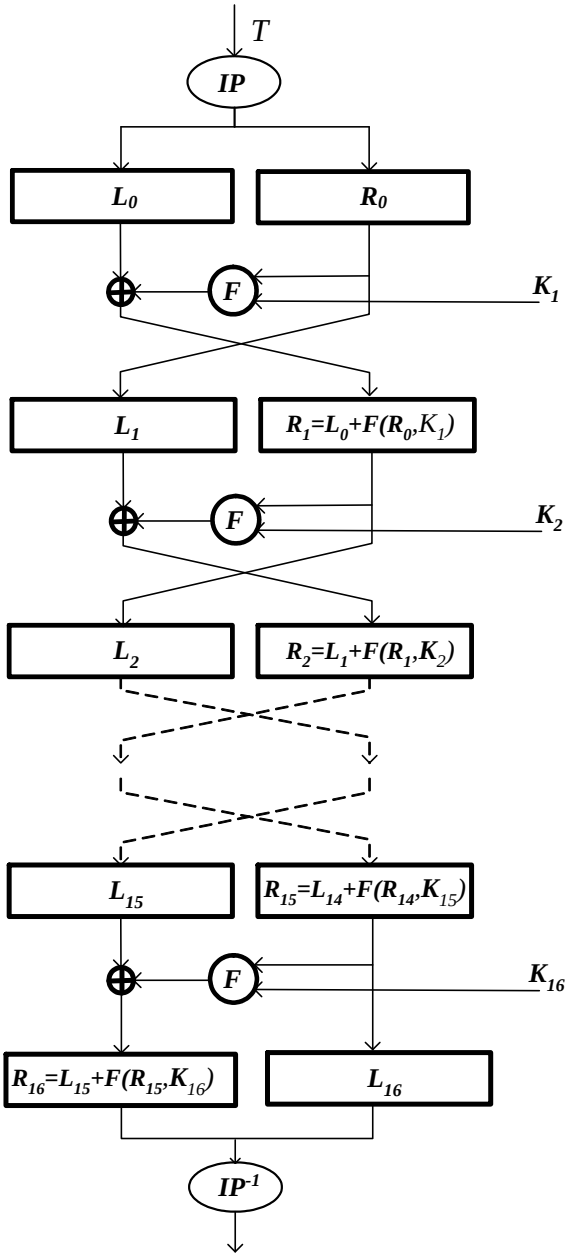


Рис. 4.2. Диаграмма DES

На следующем рисунке (Рис.4.3) представлена функция *F(Ri-1,Ki,)*. Первый операнд данной функции *Ri-1* увеличивается до 48-битного блока *E(Ri-1)* используя таблицу 4.2 побитного выбора *E*. Эта таблица применяется для перестановок, таким образом, что некоторые биты *Ri-1* выбираются более одного раза.

Таблица 4.2.

Таблицы перестановок, использованные для реализации функции F(Ri-1,Ki,)

Таблица побитного выбора ***E***	Перестановка ***P***
32 1 2 3 4 5	16 7 20 21
4 5 6 7 8 9	29 12 28 17
8 9 10 11 12 13	1 15 23 26
12 13 14 15 16 17	5 12 31 10
16 17 18 19 20 21	2 8 24 14
20 21 22 23 24 25	32 27 3 9
24 25 26 27 28 29	19 13 30 6
28 29 30 31 32 1	22 11 4 25

В результате для $Ri\text{-}1=r_1r_2...r_{32}$, имеем $E(Ri\text{-}1)=r_{32}r_1r_2...r_{32}r_1$. Далее полученный результат $E(Ri\text{-}1)$ поразрядно суммируется по модулю два с 48 битным значением ключа Ki, а полученный результат разбивается на восемь 6-битных блоков $B_1B_2...B_8$, где $E(Ri\text{-}1)\oplus Ki=B_1B_2...B_8$. Каждый 6-битный блок B_j затем используется в качестве входных данных подстановочной функции F, реализованной с использованием восьми, так называемых, S-боксов (***S-box***). Данная функция *S-box*, используя 6-битный блок B_j, возвращает 4-битный блок $S_i(B_j)$ как результат подстановки. Результирующие 4-битные блоки данных объединяются вместе в окончательный 32-битный блок, который затем перемешивается с помощью перестановки P, показанной в таблице 4.2. Таким образом, 32-битный блок, возвращаемый функцией $F(Ri\text{-}1,Ki,)$, будет равен $P(S_1(B_1)...S_8(B_8))$. Блок-диаграмма функции $F(Ri\text{-}1,Ki,)$, (S_i) представлена на следующем рисунке.

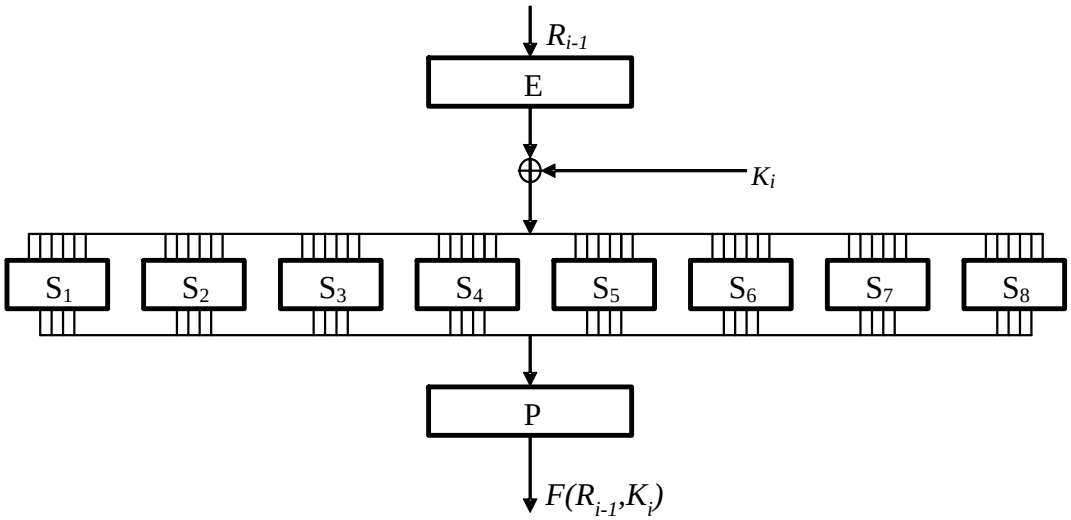


Рис. 4.3. Функция F(Ri-1,Ki,)

Как отмечалось ранее, каждый *S-box* S_i отображает 6-битный блок $B_j=b_1b_2b_3b_4b_5b_6$ в 4-битный блок как показано в таблице 4.3. для случая S_1.

Таблица 4.3.

Функция подстановки(S-box) S1

Строка / Столбец $(b_2b_3b_4b_5)$																
(b_1b_6)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Процесс подстановки осуществляется следующим образом. Четверичное число 0, 1, 2 или 3, соответствующее b_1b_6, выделяет строку в таблице, в то время как шестнадцатеричное число 0,1,2, … или 15, соответствующее $b_2b_3b_4b_5$ выделяет столбец. Тогда значение $S_i(B_j)$ это есть 4-битное шестнадцатеричное число, расположенное в строке и столбце определяемыми кодами b_1b_6 и $b_2b_3b_4b_5$.

Пример 4.1. Если B=010100, то возвращается значение, находящееся в строке с номером 0 и столбце с номером 10. Это значение равняется 6, которое представлено в двоичном виде как 0110.

Для каждой из 16 итераций используется новое значение ключа K_i, длина которого равняется 48 бит. Значение ключа K_i вычисляется из начального ключа K, как показано на рисунке 4.4.

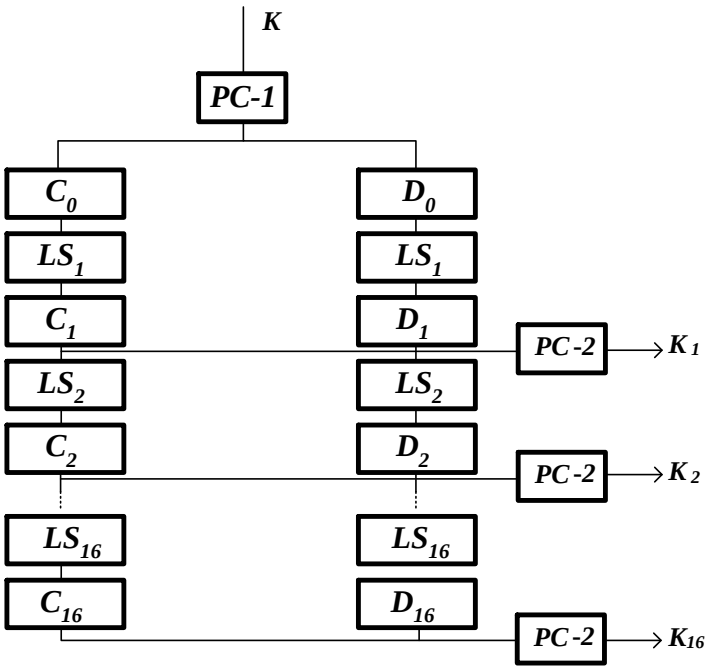


Рис. 4.4. Диаграмма вычисления значений ключа

Ключ K представляет собой 64-битный блок с 8 битами контроля на четность расположенными в позициях 8, 16, 24, 32, 40, 48, 56, 64. Таблица 4.4 (PC-1) первоначальной перестановки выполняет две функции. Во-первых, удаляет биты четности и, во-вторых, перемешивает биты 56-битного исходного ключа K.

Таблица 4.4.

Таблицы перестановки используемые при генерирование ключей

Перестановка ключа PC-1

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
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

Перестановка ключа PC-2

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

Таким образом, каждая *i*-я итерация использует различный 48-битный ключ K_i полученный из исходного ключа *K*. Рис. 4.4. иллюстрирует процедуру генерирования ключей для каждой итерации. По окончании преобразований в соответствии с таблицей PC-1 результат PC-1(*K*) разбивается на две половины *C* и *D* по 28 бит каждая. Блоки *C* и *D* затем последовательно циклически сдвигаются влево, чтобы получить каждый ключ K_i. Пусть C_i и D_i являются значениями *C* и *D*, используемыми для получения ключа K_i. Тогда значения C_i и D_i определяются согласно

соотношениям $C_i=LS_i(C_{i-1})$, $D_i=LS_i(D_{i-1})$. В последнем выражении LS_i обозначает операцию циклического сдвига влево на заданное количество позиций, указанных в таблице 4.5 для каждой из итераций формирования ключа K_i. Затем ключ K_i вычисляется как преобразование в соответствии с таблицей перестановки *PC-2*: $K_i=PC\text{-}2(C_i,D_i)$.

Таблица 4.5.

Количество # позиций циклического сдвига LS_i

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Декодирование в соответствии с описанным стандартом выполняется, используя тот же алгоритм, отличием является только последовательность использования криптографических ключей. Так, ключ K_{16} используется в первой итерации, K_{15} во второй, и т. д., и наконец ключ K_1 используется в шестнадцатой итерации. Необходимость изменения порядка подачи ключей объясняется тем, что процедура дешифрирования является обратной процедурой по отношению к шифрованию. Заметим, что, несмотря на то, что порядок ключей обратный, алгоритм таковым не является.

Ключевым элементом для понимания процедуры дешифрования является определяющее свойство операции сложения по модулю два и взаимообратная процедура перестановок *IP* и IP^{-1}. Для указанных перестановок выполняются соотношения $IP(IP^{-1}(T))=IP^{-1}(IP(T))=T$. Для реализации обратной последовательности преобразований, например, для получения R_{i-1} и L_{i-1} на основании R_i и L_i используем обратную последовательность действий. Так для получения R_{i-1}, используем тривиальное действие $R_{i-1}=L_i$ в соответствии с алгоритмом DES. В тоже время как $L_{i-1}=R_i \oplus F(L_i,K_i) \oplus F(L_i,K_i)=R_i$. Здесь используется свойство операции сложения по модулю два $F(L_i,K_i) \oplus F(L_i,K_i)=0$.

Очевидно, что криптографическая устойчивость будет уменьшаться, если внутренние ключи на каждом шаге либо на нескольких шагах алгоритма будут одинаковыми. Поэтому следует избегать варианта, когда $K_1=K_2=\ldots=K_{16}$. Последнее равенство определяет множество ***слабых ключей*** *K*, которые удовлетворяют вышеуказанному условию. Слабые ключи возникают в тех случаях, когда все биты значения блока C_0 равны 1 или 0, и все биты блока D_0 равны 1 или 0, то есть $k_{49}=k_{41}=k_{33}=\ldots=k_{57}=0$ или *1* и $k_{55}=k_{47}=k_{39}=\ldots=k_{63}=0$ или *1*.

Совокупность из четырех слабых ключей, которые представлены следующими внешними ключами (с паритетом) представлена в таблице 4.6.

Таблица 4.6.

Слабые ключи DES

01	01	01	01	01	01	01	01

1F	1F	1F	1F	1F	1F	1F	1F
E0	E0	E0	E0	E0	E0	E0	E0
FE	FE	FE	FE	FE	FE	FE	FE

Также можно выделить другой набор ключей, называемых полуслабыми ключами. Их особенность заключается в том, что есть только 2 различных внутренних ключа, повторяющихся по 8 раз каждый. Полуслабый ключ получается тогда когда блок C или D содержит набор битов 0101...0101 или 1010...1010, а другой регистр (D или C) содержит набор битов 0000...0000, 1111...1111, 0101...0101, 1010...1010.

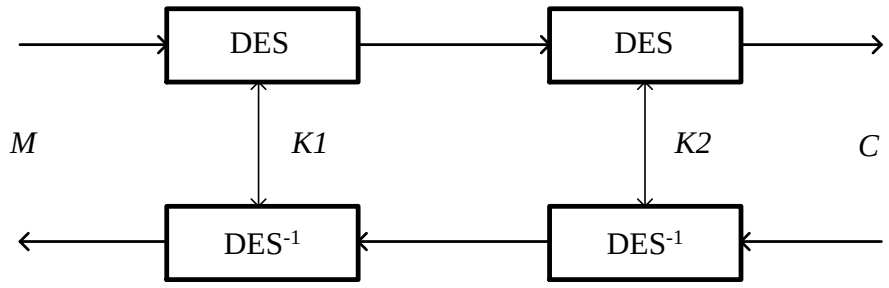
Как правило, реализация *DES* предполагает аппаратное решение в виде микросхемы с целью обеспечения высокого быстродействия, хотя возможна и программная реализация, позволяющая вносить модификации в данный алгоритм.

Выделяют два основных недостатка *DES*.

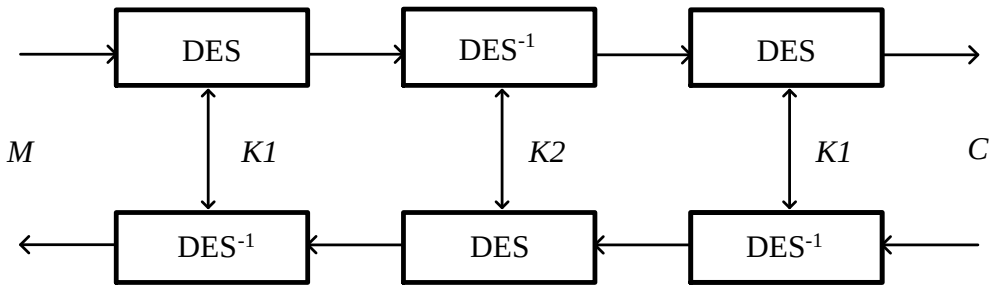
1.Размер ключа равный 56 бит может не обеспечить достаточную криптостойкость алгоритма.

2.Функция *F* подстановки, являющаяся одним из основных элементов, защищающих *DES* от взлома, может содержать в себе скрытые слабости.

С целью улучшения свойств алгоритма *DES* используют различные его модификации, среди которых выделяют подходы, направленные на увеличение размерности ключа. На рис.4.5 приведены наиболее часто используемые алгоритмы, основанные на многократном применении алгоритма DES.



а)



б)

Рис.4.5. Многократное шифрование с помощью DES

На приведенном рисунке аббревиатура *DES* используется для обозначения процедуры шифрования, а DES^{-1} для дешифрирования. Исходное сообщение и шифротекст обозначены как *M* и *C*. Как видно, обе модификации позволяют увеличить размерность ключа в два раза до 112 бит. Наиболее предпочтительной структурой является тройной *DES* (рисю 4.5б), который достаточно долго рассматривался как возможный кандидат на новый криптографический стандарт.

4.2. Международный алгоритм шифрования данных IDEA

Международный алгоритм шифрования данных (***International Data Encryption Algorithm – IDEA***) принадлежит к классу симметричных шифраторов. Основные принципы *IDEA* были опубликованы в 1990 (***J.Massey***), как альтернатива широко используемому в то время алгоритму *DES*. Подобно стандарту *DES* и в алгоритме *IDEA* используется основополагающая идея смешанных преобразований, которые случайным образом распределяют открытый текст равномерно по всему пространству зашифрованного текста.

Смешанные преобразования могут быть созданы, например, с применением преобразований последовательных перемежающихся последовательностей замен и простых операций перестановок. Как и *DES* алгоритм *IDEA* является блочным шифратором, с таким же размером блока равным 64 битам, однако длина ключа в этом алгоритме заметно больше и равняется 128 битам. Для увеличения качества шифрования к алгоритму предъявляются следующие требования:

1. ***Длина блока данных.*** Существует два основных требования для длины блока данных. Первое – это сложность алгоритма, которая требует уменьшения блока данных. Второе – это статистическая зависимость между блоками данных, которая может быть снижена увеличением длины блока данных. Как было доказано, наиболее подходящая длина блока данных для современных приложений равна 64 битам. Следует отметить, что размерность блока равная 64 битам, является неким неофициальным стандартом, используемым практически во всех симметричных криптосистемах.

2. ***Длина ключа***. Длина ключа должна быть достаточной, чтобы отражать так называемые прямые атаки (то есть, попытку перебрать все возможные значения ключа). Ключ с размером 128 бит выглядит хорошим решением, как сегодня так и в обозримом будущем.

3. ***Запутанность*** (***Confusion***)**.** Зависимость зашифрованного текста от открытого текста и ключа должна быть максимально сложной, запутанной и не очевидной.

4. ***Распространение*** (***Diffusion***)**.** Каждый бит открытого текста, также как и каждый бит ключа, должен быть принят в расчет при получении каждого бита зашифрованного текста таким образом, чтобы каждый бит шифротекста зависел от каждого бита исходного текста и ключа.

Для достижения высокого уровня запутывания над 16-битными блоками данных выполняются следующие операции:

1. Побитное сложение по модулю два (***XOR***) над двумя 16 битными операндами, которое обозначено, как $\oplus$.

2. Сложение двух целых 16 битных операндов по модулю 2^{16}, обозначенное как $\boxtimes$

3. Умножение двух целых чисел без знака по модулю $2^{16}+1$. Существует одно исключение для кода со всеми нулями. Этот код при выполнении операции умножения рассматривается как число 2^{16}, а сама операция обозначается, как $\odot$.

Для этих операция верны следующие соотношения:

$$a\odot(b\boxtimes c) \neq (a\odot b)\boxtimes(a\odot c),$$
$$a\boxtimes(b\oplus c) \neq (a\boxtimes b)\oplus c.$$

Все представленные операции могут быть легко реализованы и имеют небольшую сложность. Для более полного понимания рассмотренные операции показаны в следующей таблице 4.7 для случая, когда длина операндов *X* и *Y* равна двум битам. В этом случае операции сложения $\boxtimes$ выполняется по модулю $2^2=4$, а умножения $\odot$ выполняются по модулю $2^2+1=5$. Так, например, $2\boxtimes3 = 1 \bmod 4$, а $2\odot3 = 1 \bmod 5$. В случае нулевого кода операнда при выполнении операции умножения интерпретируется как $2^2=4$. Тогда, например, для случая, когда *X*=2 а *Y*=0 получим $2\odot0 = 2\odot4 = 8 = 3 \bmod 5$. При значениях операндов *X*=1 и *Y*=0 имеем $1\odot0 = 1\odot4 = 4 = 0 \bmod 5$. Последний результат объясняется тем, что младшие два разряда числа 4 представляют собой 0.

Таблица 4.7.

Пример выполнения операций в алгоритме IDEA

X	Y	X$\boxtimes$Y	X$\odot$Y	X$\oplus$Y
0 (00)	0 (00)	0 (00)	1 (01)	0 (00)
0 (00)	1 (01)	1 (01)	0 (00)	1 (01)
0 (00)	2 (10)	2 (01)	3 (11)	2 (10)
0 (00)	3 (11)	3 (11)	2 (10)	3 (11)
1 (01)	0 (00)	1 (01)	0 (00)	1 (01)
1 (01)	1 (01)	2 (10)	1 (01)	0 (00)
1 (01)	2 (10)	3 (11)	2 (10)	3 (11)
1 (01)	3 (11)	0 (00)	3 (11)	2 (10)
2 (10)	0 (00)	2 (10)	3 (11)	2 (10)
2 (10)	1 (01)	3 (11)	2 (10)	3 (11)
2 (10)	2 (10)	0 (00)	0 (00)	0 (00)

2 (10)	3 (11)	1 (01)	1 (01)	1 (01)
3 (11)	0 (00)	3 (11)	2 (10)	3 (11)
3 (11)	1 (01)	0 (00)	3 (11)	2 (10)
3 (11)	2 (10)	1 (01)	1 (01)	1 (01)
3 (11)	3 (11)	2 (10)	0 (00)	0 (00)

Для обеспечения приемлемого уровня распространения был разработан главный блок, как мультипликативно-аддитивная структура, представленная на рис.4.6.

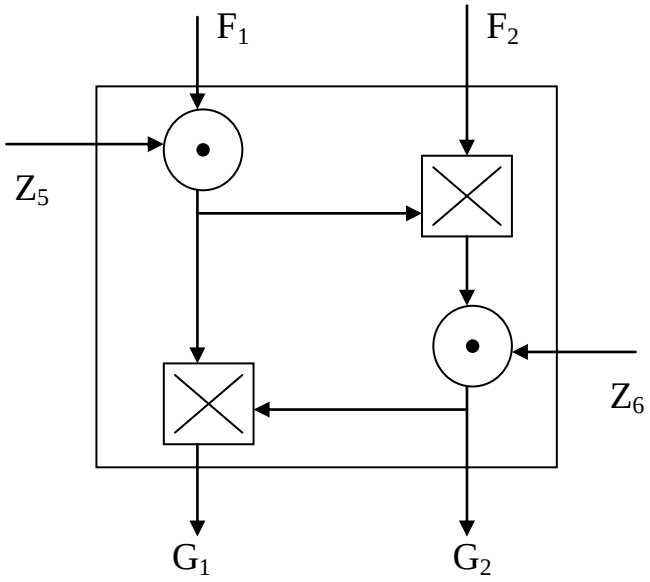


Рис.4.6. Мультипликативно-аддитивная структура.

Здесь *F1* и *F2* 16-битные значения, полученные из открытого текста. *Z5* и *Z6* - 16-ти битные подключи.

Процедура шифрования в соответствии с алгоритмом *IDEA* состоит из восьми идентичных итераций и последней девятой выходной итерации. Общий вид алгоритма *IDEA* в режиме шифрования представлен на рис.4.7.

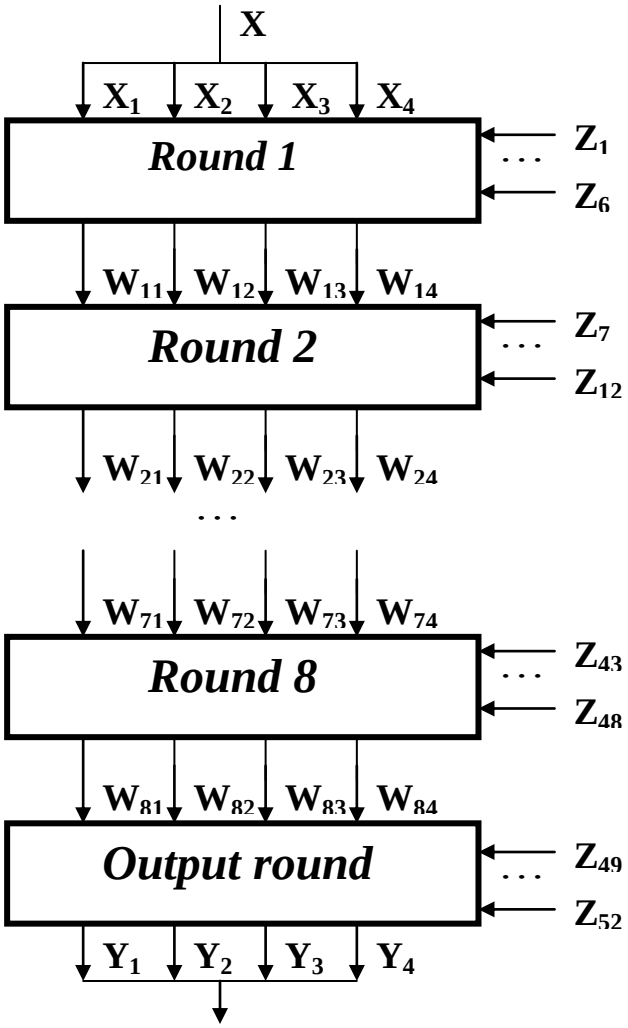


Рис.4.7. Процедура шифрования в соответствии с алгоритмом IDEA

Необходимо отметить, что все операнды, участвующие в выполнении процедуры шифрования, имеют размерность 16 бит. Это в равной мере относится к входным значениям X_i, выходным Y_i, промежуточным W_{ij} и значениям ключей Z_{rl}.

Основой данного алгоритма является структура одной итерации, которая определяет основные свойства и характеристики рассматриваемого метода шифрования.

На следующем рисунке приведена диаграмма одной итерации (первой) алгоритма IDEA.

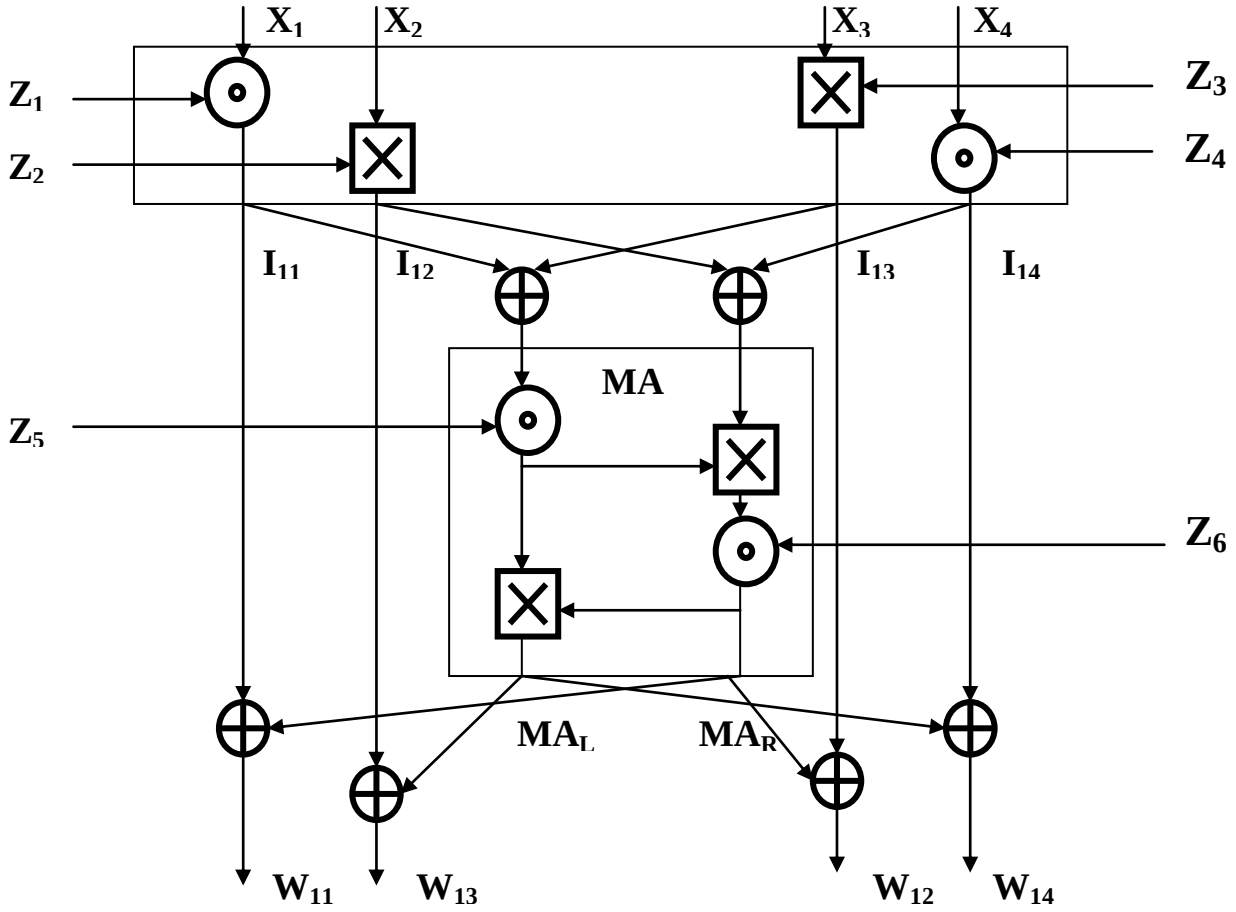


Рис.4.8. Первый цикл шифрования алгоритма IDEA

Выходная итерация отличается от восьми предыдущих в части позволяющей использовать данный алгоритм для целей дешифрации.

Структурная схема девятого этапа шифрования приведена на следующем рисунке.

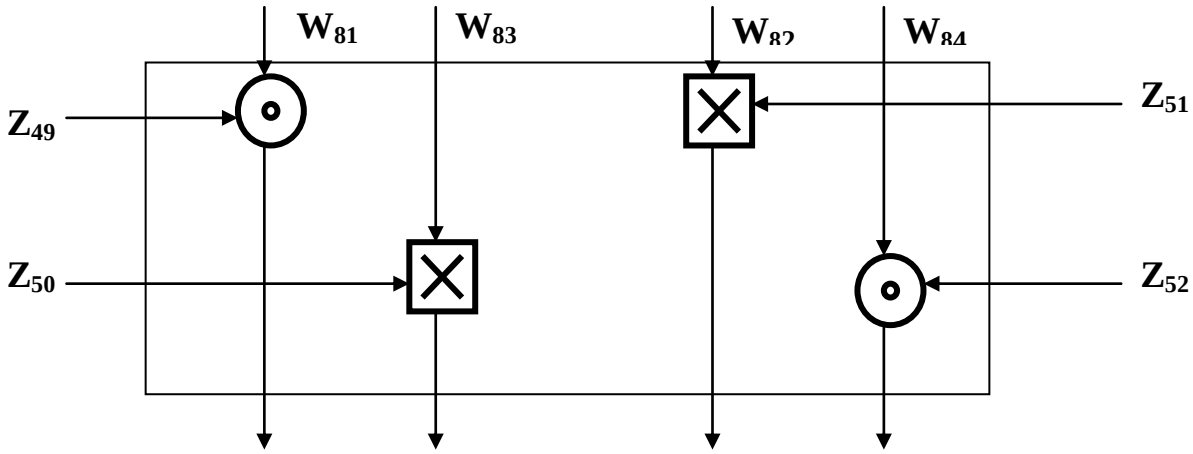


Рис.4.9. Девятый цикл шифрования алгоритма IDEA

Как видно из приведенных выше диаграмм, на каждой из итераций используются новые значения ключей, которые получаются на основании исходного 128 битного ключа *Z*. Для этого применяется генератор итерационных ключей.

Первые 8 итерационных ключей $Z_1,Z_2,\ldots,Z_8$, следующие друг за другом, являются последовательными частями криптографического ключа *K*. Отметим, что каждый итерационный ключ имеет размерность 16 бит. Затем криптографический ключ *K* циклически сдвигается на 25 бит влево и последующие восемь итерационных ключей получаются как копия следующих друг за другом частей сдвинутого на 25 бит ключа. Эта процедура повторяется до тех пор, пока не будут получены все 52 итерационных ключа.

Если *K* обозначить, как Z[1…128] тогда первые 8 итерационных ключа для всех восьми циклов будут равны Z_1=Z[1…16], Z_7=Z[97…112], Z_{13}=Z[90…105], Z_{19}=Z[83…98], Z_{25}=Z[76…91], Z_{31}=Z[44…59], Z_{37}=Z[34 …52], Z_{43}= Z[30…45]. Полный перечень итерационных ключей приведен в таблице 4.8.

Таблица 4.8.

Значения ключей используемых в алгоритме IDEA для шифрования

Итерация	Обозначение	Эквивалентное обозначение
#1	$Z_1Z_2Z_3Z_4Z_5Z_6$	Z[1…96]
#2	$Z_7Z_8Z_9Z_{10}Z_{11}Z_{12}$	Z[97…128;26…89]
#3	$Z_{13}Z_{14}Z_{15}Z_{16}Z_{17}Z_{18}$	Z[90…128;1…25;51…82]
#4	$Z_{19}Z_{20}Z_{21}Z_{22}Z_{23}Z_{24}$	Z[83…128;1…50]
#5	$Z_{25}Z_{26}Z_{27}Z_{28}Z_{29}Z_{30}$	Z[76…128;1…43]
#6	$Z_{31}Z_{32}Z_{33}Z_{34}Z_{35}Z_{36}$	Z[44…75;101…128;1…36]
#7	$Z_{37}Z_{38}Z_{39}Z_{40}Z_{41}Z_{42}$	Z[37…100;126…128;1…29]
#8	$Z_{43}Z_{44}Z_{45}Z_{46}Z_{47}Z_{48}$	Z[30…125]
Выходной цикл	$Z_{49}Z_{50}Z_{51}Z_{52}$	Z[23…86]

Для дешифрирования используются итерационные ключи, полученные на основании ключей шифрования. При их получении применяются следующие соотношения.

$$Z_j^{-1} \odot Z_j = 1 \ \mathbf{mod}\ (2^{16}+1);$$

$$-Z_j \boxtimes Z_j = 0 \ \mathbf{mod}\ 2^{16},$$

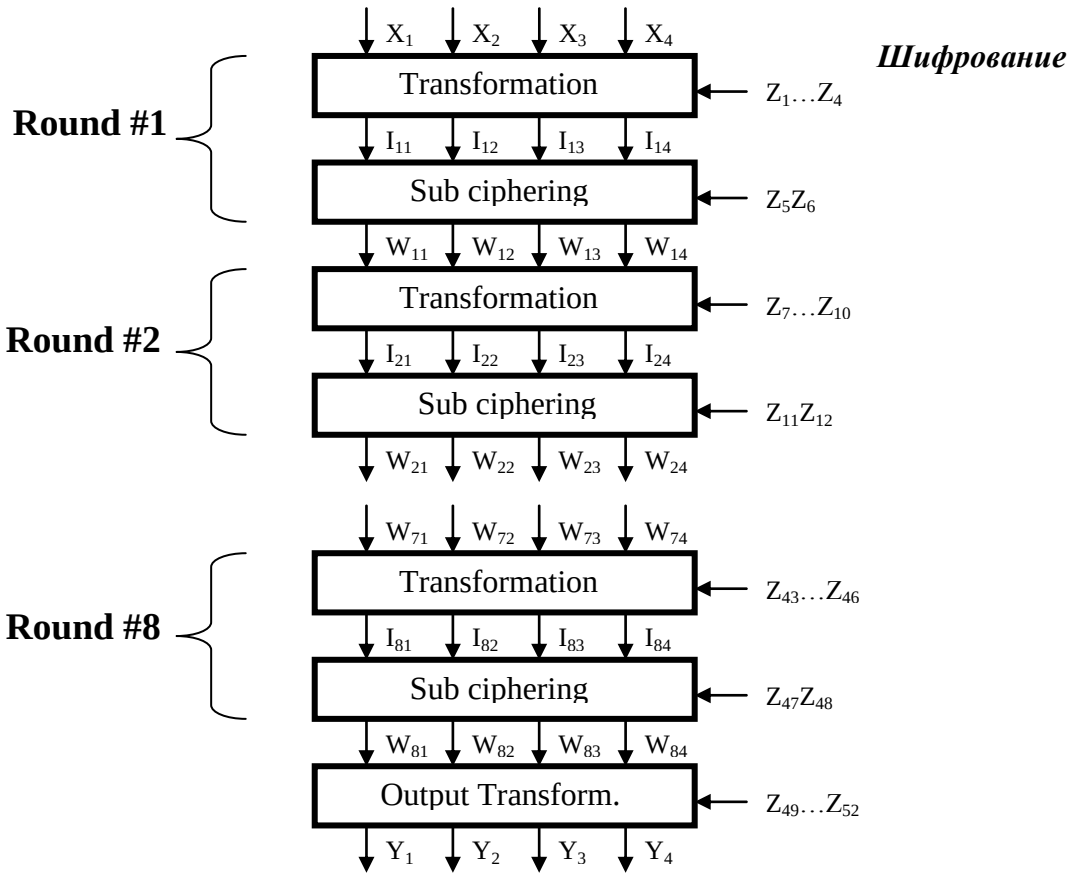
где Z_j^{-1}является мультипликативной инверсной, а $-Z_j$ – аддитивно инверсной величиной. Данные соотношения позволяют построить процедуру дешифрирования взаимообратную процедуре шифрования.

Таблица 4.9.

Значения ключей используемых в алгоритме IDEA для дешифрирования

Итерация	Обозначение	Эквивалентное обозначение
#1	$U_1U_2U_3U_4U_5U_6$	Z_{49}^{-1},$-Z_{50}$,$-Z_{51}$, Z_{52}^{-1},Z_{47},Z_{48}
#2	$U_7U_8U_9U_{10}U_{11}U_{12}$	Z_{43}^{-1},$-Z_{45}$,$-Z_{44}$, Z_{46}^{-1},Z_{41},Z_{42}
#3	$U_{13}U_{14}U_{15}U_{16}U_{17}U_{18}$	Z_{37}^{-1},$-Z_{39}$,$-Z_{38}$, Z_{40}^{-1},Z_{35},Z_{36}
#4	$U_{19}U_{20}U_{21}U_{22}U_{23}U_{24}$	Z_{31}^{-1},$-Z_{33}$,$-Z_{32}$, Z_{34}^{-1},Z_{29},Z_{30}
#5	$U_{25}U_{26}U_{27}U_{28}U_{29}U_{30}$	Z_{25}^{-1},$-Z_{27}$,$-Z_{26}$, Z_{28}^{-1},Z_{23},Z_{24}
#6	$U_{31}U_{32}U_{33}U_{34}U_{35}U_{36}$	Z_{19}^{-1},$-Z_{21}$,$-Z_{20}$, Z_{22}^{-1},Z_{17},Z_{18}
#7	$U_{37}U_{38}U_{39}U_{40}U_{41}U_{42}$	Z_{13}^{-1},$-Z_{15}$,$-Z_{14}$, Z_{16}^{-1},Z_{11},Z_{12}
#8	$U_{43}U_{44}U_{45}U_{46}U_{47}U_{48}$	Z_7^{-1},$-Z_9$,$-Z_8$, Z_{10}^{-1},Z_5,Z_6
Выходной цикл	$U_{49}U_{50}U_{51}U_{52}$	Z_1^{-1},$-Z_2$,$-Z_3$, Z_4^{-1}

Так же как и *DES*, алгоритм *IDEA* используется как для шифрования, так и для дешифрирования. Отличием является только значения используемых ключей, представленных в предыдущих таблицах. Схематично шифрование и дешифрирование представлено на следующем рисунке.



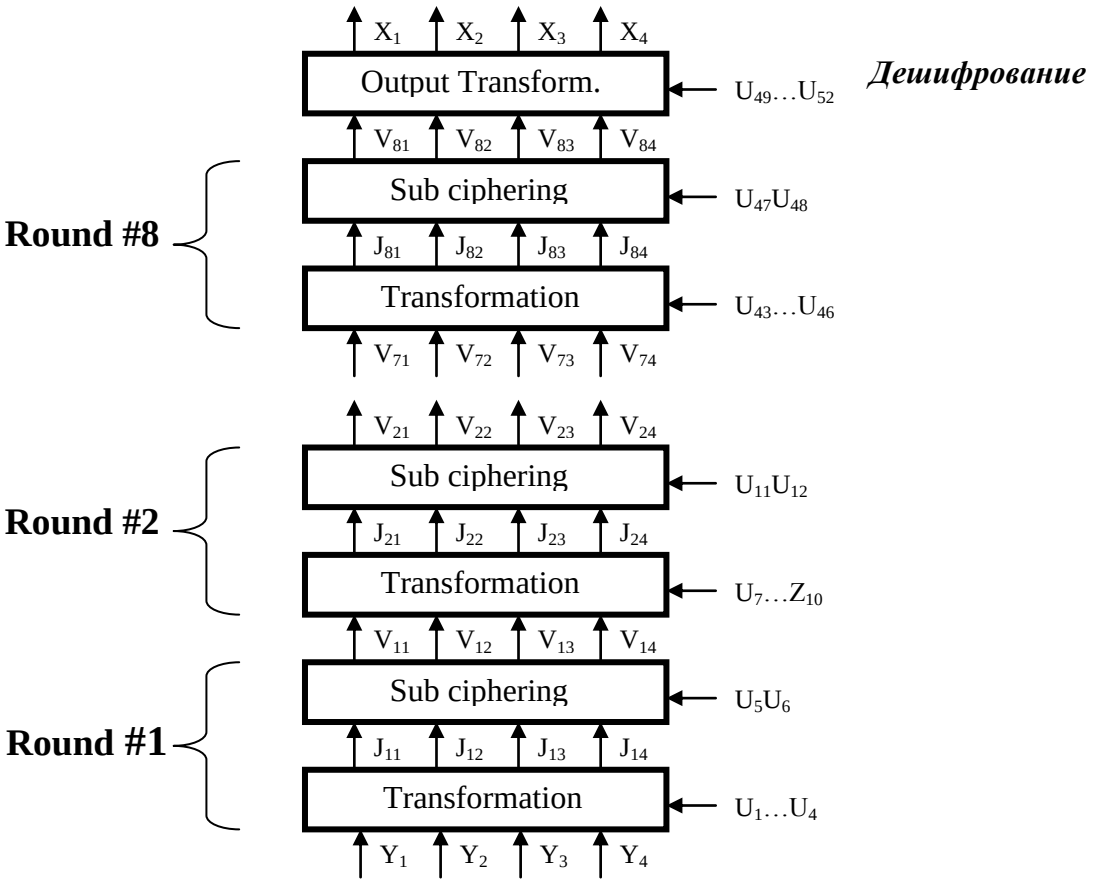


Рис.4.10. Процедуры шифрования и дешифрирования по алгоритму IDEA

Отметим, что на данных диаграммах для более полного понимания обратимости процедуры шифрования каждая итерация (Round) представлена двумя последовательными процедурами.

Рассмотрим последние этапы процедур шифрования и дешифрирования:

При шифровании на последнем этапе выполняются следующие преобразования $Y1=W81 \odot Z49$, $Y2=W83 \boxtimes Z50$, $Y3=W82 \boxtimes Z51$, $Y4=W84 \odot Z52$,

Соответственно при дешифрировании на первом этапе выполняются преобразования $J11=Y1 \odot U1$, $J12=Y2 \boxtimes U2$, $J13=Y2 \boxtimes U3$, $J14=Y4 \odot U4$, после подстановки действительных значений подключей получим

$$J11=Y1 \odot Z49^{-1}=W81 \odot Z49 \odot Z49^{-1} = W81$$
$$J12=Y2 \boxtimes -Z50 =W83 \boxtimes Z50 \boxtimes -Z50 = W83$$
$$J13=Y3 \boxtimes -Z51 =W82 \boxtimes Z51 \boxtimes -Z51 = W82$$
$$J14=Y4 \odot Z52^{-1}=W84 \odot Z52 \odot Z52^{-1} = W84$$

Итак, результат первого этапа дешифрирования является таким же, как и значения, полученные перед последним этапом шифрования, что свидетельствуем о обратимости алгоритма IDEA.

Далее рассмотрим следующие этапы, соответствующие процедурам шифрования и дешифрирования.

$$W81 = I81 \oplus MA_R(I81 \oplus I83,\ I82 \oplus I84)$$
$$W82 = I82 \oplus MA_R(I81 \oplus I83,\ I82 \oplus I84)$$
$$W83 = I83 \oplus MA_L(I81 \oplus I83,\ I82 \oplus I84)$$
$$W84 = I84 \oplus MA_L(I81 \oplus I83,\ I82 \oplus I84)$$

Здесь $MA_R(A,B)$ означает правую часть выходного значения для MA (см. рис 4.8), также, как $MA_L(A,B)$ – левую часть.

Затем в соответствии с процедурой дешифрирования (см. рис 4.8 и рис. 4.10) можно вычислить значения $V11, V12, V13, V14$, Тогда

$$V11 = J11 \oplus MA_R(J11 \oplus J13,\ J12 \oplus J14) =$$
$$= W81 \oplus MA_R(W81 \oplus W83,\ W82 \oplus W84) =$$
$$= I81 \oplus MA_R(I81 \oplus I83, I82 \oplus I84) \oplus MA_R[(I81 \oplus MA_R(I81 \oplus I83, I82 \oplus I84) \oplus I83 \oplus MA_R(I81 \oplus I83, I82 \oplus I84), I82 \oplus MA_L(I81 \oplus I83, I82 \oplus I84) \oplus I84 \oplus MA_L(I81 \oplus I83, I82 \oplus I84)] =$$
$$= I81 \oplus MA_R(I81 \oplus I83, I82 \oplus I84) \oplus MA_R(I81 \oplus I83, I82 \oplus I84) =$$
$$= I81$$

Также можно легко показать, что $V12 = I12$, $V13 = I13$ и $V14 = I14$.

Таким образом, приведенные преобразования показывают работоспособность алгоритма IDEA в режиме шифрования и дешифрирования.

4.3. Симметричный блочный шифратор BLOWFISH

Блочный ***шифратор BLOWFISH*** принадлежит к классу стандартных симметричных шифраторов. Основные принципы BLOWFISH были опубликованы в 1994 Брюсом Шнейером (***Bruce Schneier***), как очередная альтернатива стандарту DES. При проектировании криптосистемы BLOWFISH было достигнуто выполнение следующих требований.

1.***Производительность***. Процедуры шифрования и дешифрирования требуют минимального процессорного времени. BLOWFISH кодирует 64 битные блоки данных на 32 разрядном микропроцессоре в течение 18 циклов на байт.

2.***Объем памяти***. BLOWFISH требует для своей реализации менее чем 5 Кбайт памяти.

3.***Простота реализации***. Структура BLOWFISH очень проста для реализации и в то же время обладает высокой криптостойкостью.

4.***Размерность ключа***. Длина ключа должна быть достаточной, чтобы противостоять так называемым прямым атакам путем перебора всех

возможных значений ключей. В то же время он должен обеспечивать высокую скорость работы процедур шифрования и дешифрирования. Вот почему BLOWFISH предлагает использование ключей различной длины. Длина колеблется от 32 до 448 бит

5.***Длина блока данных***. BLOWFISH работает с 64-битными блоками открытого текста и получает 64-битные зашифрованные блоки. В настоящее время BLOWFISH используется в огромном количестве программных продуктов и имеет прекрасные оценки специалистов.

Рассмотрим основные компоненты шифратора BLOWFISH.

Ключ. BLOWFISH позволяет использовать ключи длиной от одного 32 битного слова K_1 до четырнадцати 32 битных слов.

$$K_1,K_2,\ldots,K_j,\ 1\le j\le 14.$$

Подключи. Подключи генерируются на основании исходного ключа и хранятся в *P*-массиве состоящем из восемнадцати 32-битных слов

$$P_1,P_2,\ldots,P_{18}.$$

S-матрицы. В шифраторе BLOWFISH определяются четыре *S*-матрицы. Каждая матрица состоит из 256 32-битных слов.

$$S_{1,0},\ S_{1,1},\ldots,S_{1,255},$$
$$S_{2,0},\ S_{2,1},\ldots,S_{2,255},$$
$$S_{3,0},\ S_{3,1},\ldots,S_{3,255},$$
$$S_{4,0},\ S_{4,1},\ldots,S_{4,255}.$$

Перед использованием шифратора BLOWFISH первоначально вычисляются значения подключей *P* и *S*-матриц. Алгоритм вычисления итерационных ключей и *S*-матриц состоит из следующих этапов.

1.Для инициализации *P*-массива и *S*-матриц используется число π таким образом, что P_1 равняется 32 битам числа π (дробной его части), следующие 32 бита присваиваются P_2 и так далее. Вначале инициализируется *P*-массив, затем последовательно четыре *S*-матрицы. В шестнадцатеричной системе счисления это выглядит следующим образом: P_1=243F6A88, P_2=85 A308D3 ,…, $S_{4,254}$= 57FDFE3, $S_{4,255}$=3AC372E6.

2.Выполняется побитовая XOR операция между *P*-массивом и *K*-массивом, повторяя значение ключа, если это необходимо. Например, для максимальной длины ключа имеем $P_1=P_1\oplus K_1,\ P_2=P_2\oplus K_2,\ldots,\ P_{14}=P_{14}\oplus K_{14},\ P_{15}=P_{15}\oplus K_1,\ \ldots,\ P_{18}=P_{18}\oplus K_4$.

3.Основываясь на текущих значениях массива *P* и *S*-матриц, шифруется 64 битный блок данных, состоящий из 64 нулей. Затем 64 бита зашифрованных данных используются в качестве новых значений P_1 и P_2, а также как незашифрованный блок данных для следующей итерации шифрования. Новые значения *P* и массивов *S* используются в последующих итерациях шифрования для получения новых значений для $P_3,P_4,\ldots,P_{18},S_{1,0},S_{1,1},S_{1,2},\ldots\ S_{4,254},S_{4,255}$. Эта процедура может быть представлена как $P_1,P_2=E_{P,S}[0],\ P_3,P_4=E_{P,S}[P_1||P_2],\ P_5,P_6=E_{P,S}\ [P_3||P_4],\ \ldots,$

$P17,P18=E_{P,S}$ [$P15||P16$], $S1,0,S1,1=E_{P,S}$ [$P17||P18$], $S1,2,S1,3=E_{P,S}$ [$S1,0||S1,1$],…,$S4,254,S4,255=E_{P,S}$ [$S4,252||S4,253$].

Структура шифратора BLOWFISH во многом повторяет структуру стандарта DES. Он также состоит из последовательно выполняемых итераций, на каждой из которых используются значения итерационных ключей.

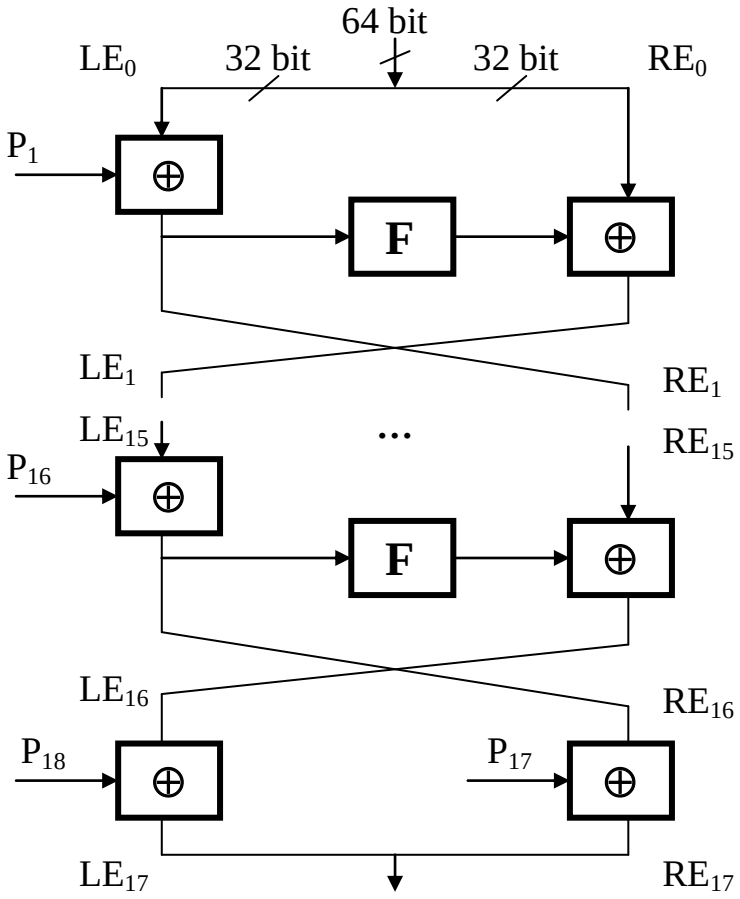


Рис.4.11. Процедуры шифрования BLOWFISH

Подобно, как и в DES исходный блок шифруемых данных разбивается на две части: правую *RE0* и левую *LE0* части по 32 бита. Затем левая часть *LE0* поразрядно складывается по модулю два со значением первого подключа *P1*. Результат операции сложения является аргументом функции *F*, значение которой суммируется с правой частью *RE0*. Подобная последовательность операций повторяется для последующих итераций. Аналитически шифрование можно описать в виде следующего алгоритма

For i=1 ***to*** 16 ***do***
$RE_i=LE_{i-1}\oplus P_i;$
$LE_i=F[RE_i]\oplus RE_{i-1};$
$LE_{17}=RE_{16}\oplus P_{18};$
$RE_{17}=LE_{16}\oplus P_{17}.$

При дешифрировании подключи *P* используются в обратной последовательности в соответствии со следующей блок схемой.

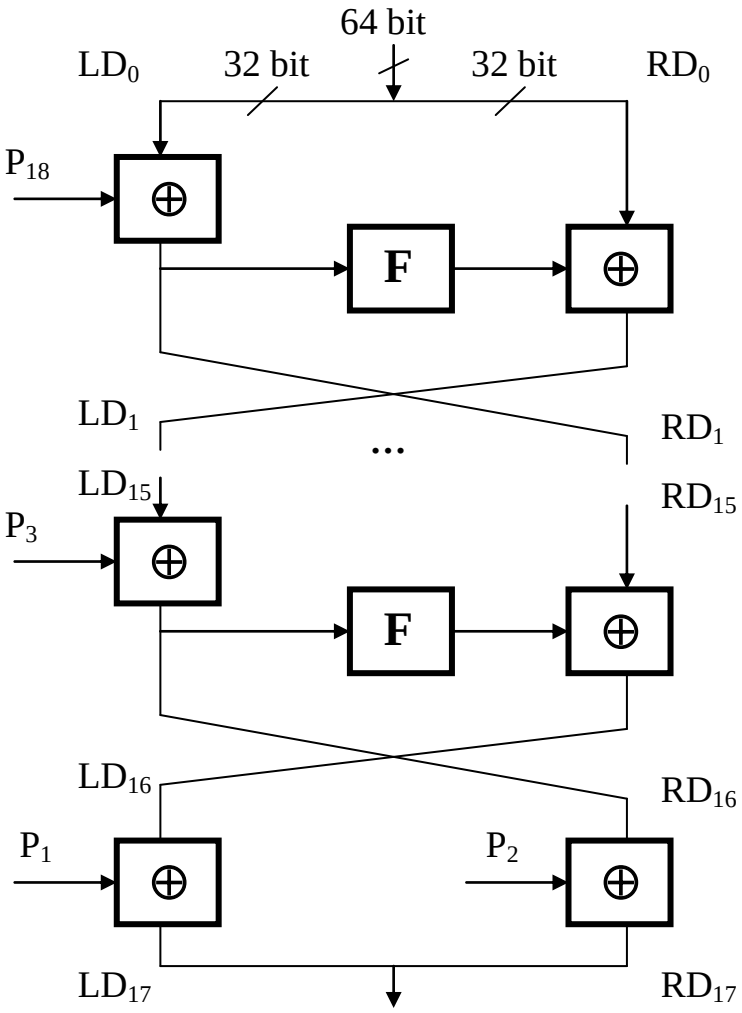


Рис.4.12. Процедуры дешифрирования BLOWFISH

Аналитически шифрование можно описать в виде следующего алгоритма.

For i=1 ***to*** 16 ***do***
RDi=LDi-1⊕P19-i;
LDi=F[RDi]⊕RDi-1;
LD17=RD16 ⊕P1;
RD17=LD16 ⊕P2;

Детальная структурная схема реализации одной итерации BLOWFISH приведена на рис.4.13.

Штрих-пунктиром на приведенном рисунке показана функция *F*, которая является основным элементом каждой итерации. Четыре *S*-матрицы используются как таблицы подстановки.

Для реализации BLOWFISH необходимо использовать 2 операции: операцию сложение (+) по модулю 2^{16}; побитовое исключающее или (⊕-сложение по модулю два)**.**

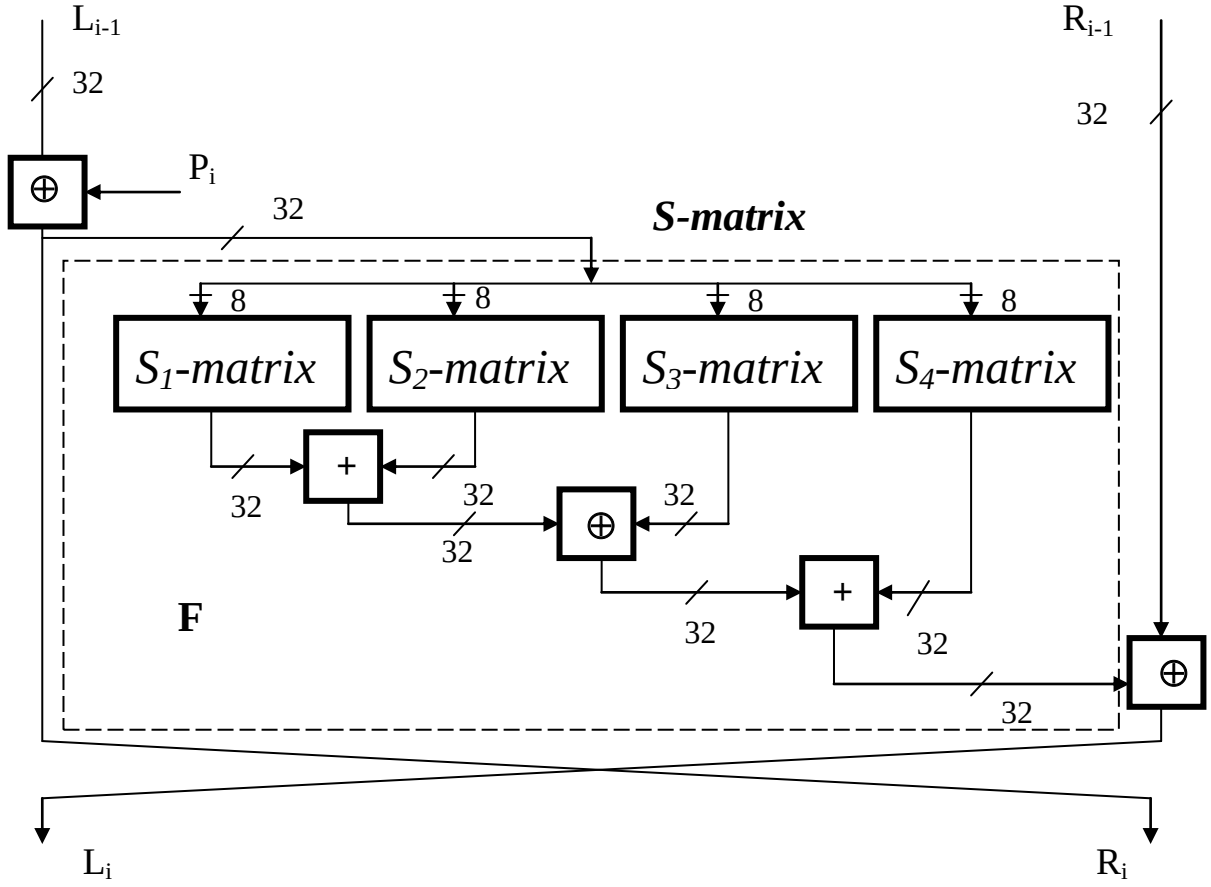


Рис.4.13. Структурная схема реализации одной итерации BLOWFISH

Все итерации в процессе шифрования и дешифрирования включают составную функцию *F*, основным элементом которой является операция подстановки, определенная *S*-матрицами.

Первоначально 32 битные входные значения делятся на 4 части *a, b, c, d* по 8 бит каждая. В соответствии с 8-битным кодом, являющимся индексом элемента *S*-матрицы, получается новое 32-битное значение. Четыре *S*-матрицы являются генераторами операндов для реализации функции *F*. Общее выражение для этой функции имеет вид.

$$F(a,b,c,d)=(S1,a+S2,b)\oplus S3,c+S4,d.$$

BLOWFISH имеет следующие достоинства.

1.В сравнении с DES алгоритмом *S*-матрицы для BLOWFISH зависят от сеансового ключа и поэтому всегда их элементы имеют новые значения.

2.В каждой итерации подвергаются преобразованию данные как левой, так и правой частей блока.

3.Скорость шифрования и дешифрирования для BLOWFISH самая высокая в сравнении с классическими симметричными алгоритмами.

В таблице 4.12 приведена сравнительная характеристика эффективности алгоритмов шифрования рассмотренных в предыдущих разделах данной главы.

Таблица 4.10.

Сравнительная характеристика алгоритмов шифрования

Алгоритм	Количество циклов на одну итерацию	Количество итераций	Количество циклов для шифрования блока данных
BLOWFISH	9	16	144
DES	18	16	288
IDEA	50	8	400
Triple DES	18	48	856

Анализ приведенной таблицы показывает высокую эффективность рассмотренных алгоритмов, которые являются основой подавляющего большинства реальных приложений, использующих криптографические алгоритмы.

4.4. Стандарт AES

4.4.1. Общие сведения об алгоритме AES

Как отмечалось в предыдущих разделах, криптографический стандарт DES в свое время получил достаточно широкое распространение и до сих пор широко используется в различных приложениях. Однако на текущий момент этот стандарт находит все меньшее применение. В основном, из-за небольшой длина его ключа, которая составляет 56 бит, что чрезвычайно мало на современном этапе развития ЭВМ. Кроме того, возросшие возможности современной вычислительной техники позволяют обрабатывать большие массивы данных в соответствии с классическими алгоритмами практически на лету и в состоянии реализовывать более сложные в вычислительном отношении методы шифрования.

В 1997 году Американский институт стандартизации (NIST) объявил конкурс на новый стандарт симметричного криптографического алгоритма ***AES*** (***Advanced Encryption Standard)***. На сей раз уже были учтены основные недостатки известных симметричных криптосистем и к разработке были подключены самые крупные центры по криптографии со всего мира. Кроме того, сформулированы требования, которым должен был отвечать кандидат в стандарты. Основными из предложенных требований к кандидату являлись длина криптографического ключа, которая должна была быть не менее чем 128 бит, а также эффективная реализация алгоритма как аппаратно, так и программно. Кроме того, шифр во многом должен был повторять

характеристики предыдущего стандарта, такие как симметричный принцип шифрования исходного сообщения по блокам и ряд других.

На первом этапе конкурса в соответствии с весьма жесткими требованиями были определены наиболее вероятные победители конкура, представленные в таблице 4.11.

Таблица 4.11.

Кандидаты на стандарт AES

Алгоритм	Создатель	Страна	Быстродействие (200МГц)
MARS	IBM	US	8 Мбайт/с
RC6	R.Rivest & Co	US	12 Мбайт/с
Rijndael	***V.Rijmen & J.Daemen***	***BE***	***7 Мбайт/с***
Serpent	*Universities*	IS, UK, NO	2 Мбайт/с
TwoFish	B.Schneier & Co	US	11 Мбайт/с

Все эти алгоритмы были признаны достаточно стойкими и успешно противостоящими всем широко известным методам взлома. 2 октября 2000 года NIST объявил о своем выборе – победителем конкурса стал бельгийский алгоритм ***RIJNDAEL***. С этого момента с алгоритма-победителя сняты все патентные ограничения – его можно будет использовать в любом криптографическом приложении без отчисления каких-либо средств создателю.

Алгоритм ***AES*** (***Rijndael***) был разработан двумя специалистами по криптографии ***V.Rijmen*** и ***J.Daemen*** из Бельгии. Он является нетрадиционным блочным шифром, поскольку не использует стандартную итерационную схему Фейстеля.

Для AES величина N_b означает число 32-битных слов по отношению к 128 битному ***входному блоку, выходному блоку*** или блоку внутреннего ***состояния*** криптографической системы ($128=N_b \times 32$, откуда $N_b=4$). Величина N_k представляет собой количество 32-битных слов криптографического ключа в зависимости от его длины 128, 192 или 256 бит. Соответственно, N_k равняется 4, 6 или 8. Количество раундов шифрования алгоритма принимает значение 10, 12 или 14. Данные, преобразуемые согласно алгоритму AES, представляются в виде двухмерного массива байт размером 4x4, 4x6 или 4x8 в зависимости от установленной длины блока. Далее на соответствующих этапах преобразования производятся либо над независимыми столбцами, либо над независимыми строками, либо вообще над отдельными байтами в таблице.

Для описания алгоритма AES используются следующие обозначения.

1.Все байты данных представляются в векторной форме ($b_7,b_6,b_5,b_4,b_3,b_2,b_1,b_0$), которая соответствует полиномиальному представлению в виде.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 = \sum_{i=0}^{7} b_i x^i.$$

Например (01001011)$\rightarrow x^6+x^3+x+1$.

2.Массив байт $a_0,a_1,a_2,\ldots,a_{15}$ входного блока $ip_0,ip_1,ip_2,\ldots,ip_{126}$, ip_{127} длины 128 бит представляется как $a_0=(ip_0,ip_1,\ldots,ip_7)$, $a_1=(ip_8,ip_9,\ldots,ip_{15})$, …, $a_{15}=(ip_{121},ip_{122},\ldots,ip_{127})$, где ip_k обозначает $input_k$ для k=0,1,2,…,127. В общем случае для данных различной размерности, например, в зависимости от длины ключа, имеем $a_n=(ip_{8n},ip_{8n+1},\ldots,ip_{8n+7})$, где $n\leq 16$.

3.Все внутренние операции преобразования в соответствии со стандартом AES выполняются над двухмерными массивами, называемыми ***состоянием*** (***state***). Состояние представляется как четыре строки байт $S_{r,c}$, где $0\leq r<4$ и $0\leq c< N_b$. Входной массив данных копируется в состояние по следующей схеме.

$S_{r,c},=in(r+4c)$, где $0\leq r<4$ и $0\leq c<N_b$,

а состояние копируется в блок выходных данных как

$out(r+4c)=S_{r,c}$, где $0\leq r<4$ и $0\leq c<N_b$,

где $S(r,c)$ отдельный байт состояния. Операции перехода от входных данных к состоянию и от состояния к выходным данным проиллюстрированы в следующей таблице.

Таблица 4.12.

Отображение данных

in_0	in_1	in_2	in_3	$\Rightarrow$	$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$	$\Rightarrow$	out_0	out_1	out_2	out_3
in_4	in_5	in_6	in_7		$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$		out_4	out_5	out_6	out_7
in_8	in_9	in_{10}	in_{11}		$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$		out_8	out_9	out_{10}	out_{11}
in_{12}	in_{13}	in_{14}	in_{15}		$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$		out_{12}	out_{13}	out_{14}	out_{15}

При реализации алгоритма AES применяются следующие математические операции, определенные над конечными полями, в качестве аргументов которых выступают байты данных.

Сложение (***Addition***) двух элементов в конечном поле реализуется путем сложения по модулю два коэффициентов полиномиального представления слагаемых. Например, $(x^5+x^3+x^2+1)+(x^7+x^5+x+1)=(x^7+x^3+x^2+x)$, что в бинарной форме запишется как (00101101)+(10100011)=(10001110), а шестнадцатеричной как (*2d*)+(*a3*)=(*8e*).

Умножение (***Multiplication***) полиномов над GF(2^8) соответствует умножению полиномов по модулю примитивного полинома $m(x)=x^8+x^4+x^3+x+1$ (1(00011011)=(*1b*)). Например, $(x^6+x^5+x^4+x+1)\times(x^7+x^5+x^2+1)$ ***mod*** $x^8+x^4+x^3+x+1=x^7+x^6+x^5+x+1$. В силу того, что операция умножения является ассоциативной операцией, для нее выполняется равенство.

$$a(x)(b(x)+c(x))=a(x)b(x)+a(x)c(x).$$

Элемент (01)=(00000001) называется ***мультипликативным идентификатором*** (***multiplicative identity***). Для каждого полинома $b(x)$ существует инверсный полином по модулю $m(x)$. Выделяют операцию

умножения двоичного полинома $b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x^1+b_0x^0=\sum_{i=1}^{7}b_ix^i$ на x=(02) по модулю $m(x)=x^8+x^4+x^3+x+1$. Эта операция на уровне байта реализуется, как сдвиг влево на один бит и сложение результата сдвига по модулю два с байтом (*1b*), соответствующим примитивному полиному *m(x).* Эта операция обозначается как ***xtime***(). Умножение на x^k реализуется как повторяющееся k раз выполнение операции *xtime*().

Пример 4.2. Рассмотрим вычисление произведения (57)×(13)=?. Первоначально получим вспомогательные результаты для (57)=(01010111). Соответственно,

(57)×(02)=*xtime*(57)=(10101110)=(*ae*)

(57)×(04)=*xtime*(*ae*)=(01011100)⊕(00011011)=(01000111)=(47)

(57)×(08)=*xtime*(47)=(10001110)=(*8e*)

(57)×(10)=*xtime*(*8e*)=(00011100)⊕(00011011)=(00000111)=(07)

Тогда окончательный результат будет вычисляться как

(57)×(13)=(57)×{(01)⊕(02)⊕(10)}=(57)⊕(01)×(02)⊕(57)×(10)=(57)⊕(ae)⊕(07)=(01010111)⊕(10101110)⊕(00000111)=(11111110)=(*fe*).

Более сложными представляются операции сложения и умножения полиномов вида $a(x)=a_3x^3+a_2x^2+a_1x^1+a_0 \Leftrightarrow a=(a_0,a_1,a_2,a_3)$ и $b(x)=b_3x^3+b_2x^2+b_1x^1+b_0 \Leftrightarrow b=(b_0,b_1,b_2,b_3)$, где используются не бинарные коэффициенты, а коэффициенты представляющие собой байты.

Тогда операция ***сложения*** будет выполняться как

$$a(x)+b(x)=(a_3\oplus b_3)x^3+(a_2\oplus b_2)x^2+(a_1\oplus b_1)x^1+(a_0\oplus b_0).$$

Операция ***умножения*** $c(x)=a(x)\times b(x)$ для полиномов с коэффициентами представленными байтами имеет вид.

$$c(x)=a(x)\times b(x)=c_6x^6+c_5x^5+c_4x^4+c_3x^3+c_2x^2+c_1x^1+c_0,$$

где

$c_0=a_0b_0$ $\quad$ $c_4=a_3b_1\oplus a_2b_2\oplus a_1b_3$

$c_1=a_1b_0\oplus a_0b_1$ $\quad$ $c_5=a_3b_2\oplus a_2b_3$

$c_2=a_2b_0\oplus a_1b_1\oplus a_0b_2$ $\quad$ $C_6=a_3b_3$

$c_3=a_3b_0\oplus a_2b_1\oplus a_1b_2\oplus a_0b_3$

Следующим этапом операции умножения, согласно алгоритму AES, является уменьшение степеней коэффициентов полинома $c(x)$ по модулю (x^4+1) таким образом, что x^i **mod** $(x^4+1)=x^{\,i \bmod 4}$. Тогда модулярное произведение $a(x)\otimes b(x)$ полиномов $a(x)$ и $b(x)$ определяется как

$$d(x)=a(x)\otimes b(x)=d_3x^3+d_2x^2+d_1x^1+d_0,$$

где

$$d_0=a_0b_0\oplus a_3b_1\oplus a_2b_2\oplus a_1b_3$$
$$d_1=a_1b_0\oplus a_0b_1\oplus a_3b_2\oplus a_2b_3$$
$$d_2=a_2b_0\oplus a_1b_1\oplus a_0b_2\oplus a_3b_3$$
$$d_3=a_3b_0\oplus a_2b_1\oplus a_1b_2\oplus a_0b_3$$

В матричной форме

$$\begin{vmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{vmatrix} = \begin{vmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{vmatrix} \times \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix}$$

4.4.2. Расширение криптографического ключа в алгоритме AES

Увеличение размерности ключа (***key expansion***) K заключается в получении общего числа слов ключа равного $N_b(N_r+1)$ с использованием следующих операций над словами.

Операция ***RotWord***() заключается в выполнении над четырьмя байтами $[a_0,a_1,a_2,a_3]$ циклической перестановки. В результате получается $[a_1,a_2,a_3,a_0]$.

Операция ***SubWord***() заключается в выполнении над четырьмя байтами $[a_0,a_1,a_2,a_3]$ операции подстановки для каждого байта в соответствии с таблицей 4.13 подстановки (***S-box***).

Таблица 4.13.

AES S-box

		y															
		0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*a*	*b*	*c*	*d*	*e*	*f*
x	*0*	*63*	*7c*	*77*	*7b*	*f2*	*6b*	*6f*	*c5*	*30*	*01*	*67*	*2b*	*fe*	*d7*	*ab*	*76*
	1	*ca*	*82*	*c9*	*7d*	*fa*	*59*	*47*	*f0*	*ad*	*d4*	*a2*	*af*	*9c*	*a4*	*72*	*c0*
	2	*b7*	*fd*	*93*	*26*	*36*	*3f*	*f7*	*cc*	*34*	*a5*	*e5*	*f1*	*71*	*d8*	*31*	*15*
	3	*04*	*c7*	*23*	*c3*	*18*	*96*	*05*	*9a*	*07*	*12*	*80*	*t2*	*eb*	*27*	*b2*	*75*
	4	*09*	*83*	*2c*	*1a*	*1b*	*6e*	*5a*	*a0*	*52*	*3b*	*d6*	*b3*	*29*	*e3*	*2f*	*84*
	5	*53*	*d1*	*00*	*ed*	*20*	*fc*	*b1*	*5b*	*6a*	*cb*	*be*	*39*	*4a*	*4c*	*58*	*cf*
	6	*d0*	*ef*	*aa*	*fb*	*43*	*4d*	*33*	*85*	*45*	*f9*	*02*	*7f*	*50*	*3c*	*9f*	*a8*
	7	*51*	*a3*	*40*	*8f*	*92*	*9d*	*38*	*f5*	*bc*	*b6*	*da*	*21*	*10*	*ff*	*f3*	*d2*
	8	*cd*	*0c*	*13*	*ec*	*5f*	*97*	*44*	*17*	*c4*	*a7*	*7e*	*3d*	*64*	*5d*	*19*	*73*
	9	*60*	*81*	*4f*	*dc*	*22*	*2a*	*90*	*88*	*46*	*ee*	*b8*	*14*	*de*	*5e*	*0b*	*db*
	a	*e0*	*32*	*3a*	*0a*	*49*	*06*	*24*	*5c*	*c2*	*d3*	*ac*	*62*	*91*	*95*	*e4*	*79*
	b	*e7*	*c8*	*37*	*6d*	*d5*	*4e*	*a9*	*6c*	*56*	*f4*	*ea*	*ea*	*65*	*7a*	*ae*	*08*
	c	*ba*	*78*	*25*	*2e*	*1c*	*a6*	*b4*	*c6*	*e8*	*dd*	*74*	*1f*	*4b*	*bd*	*8b*	*8a*
	d	*70*	*3e*	*b5*	*66*	*48*	*03*	*f6*	*0e*	*61*	*35*	*57*	*b9*	*86*	*c1*	*1d*	*9e*
	e	*e1*	*f8*	*98*	*11*	*69*	*d9*	*8e*	*94*	*9b*	*1e*	*87*	*e9*	*ce*	*55*	*28*	*df*
	f	*8c*	*a1*	*89*	*0d*	*bf*	*f6*	*42*	*68*	*41*	*99*	*2d*	*0f*	*b0*	*54*	*bb*	*16*

Операция ***Rcon[i]*** заключается в формировании константы вида *Rcon[i]* $=[x^{i-1}, (00), (00), (00)]$, где первый байт генерируется из двоичного кода, содержащего все нули кроме *i*-ой позиции определяемой (x^{i-1}).

Пример 4.3. Рассмотрим операцию *Rcon[i]* для различных значений *i*.

Rcon[1] = $[x^{i-1}, (00), (00), (00)] = [x^0, (00), (00), (00)] =[(01), (00), (00), (00)]=01000000$.

Rcon[2] = $[x^1, (00), (00), (00)]$ = 02000000

Rcon[3] = $[x^2, (00), (00), (00)]$ = 04000000

Rcon[4] = $[x^3, (00), (00), (00)]$ = 08000000

Rcon[5] = $[x^4, (00), (00), (00)]$ = 10000000

Rcon[6] = $[x^5, (00), (00), (00)]$ = 20000000

Rcon[7] = $[x^6, (00), (00), (00)]$ = 40000000

Rcon[8] = $[x^7, (00), (00), (00)]$ = 80000000

Rcon[9] = $[x^8, (00), (00), (00)] = [x^7 \times x, (00), (00), (00)]$ = 1b000000

$x^7 \times x = xtime(x^7) = xtime(80) = \{leftshift(80)\} \oplus \{1b\}$ = 1b

Rcon[10] = $[x^9, (00), (00), (00)] = [x^8 \times x, (00), (00), (00)]$ = 36000000

Rcon[11] = $[x^{10}, (00), (00), (00)] = [x^9 \times x, (00), (00), (00)]$ = 6c000000

Rcon[12] = $[x^{11}, (00), (00), (00)] = [x^{10} \times x, (00), (00), (00)]$ = d8000000

Непосредственно процедура расширения ключа *K* заключается в построении линейного массива четырех байтовых слов $[w_i]$, $0 \le i < N_b(N_r+1)$ в соответствии с процедурой, представленной в виде кода.

```
Key Expansion (byte key[4×Nk], word w[Nb×(Nr+1)], Nk)
    Begin
        i=1
        while (i<Nk)
            w[i]=word[key[4×i],key[4×i+1],key[4×i+2], key[4×i+3]]
            i=i+1
        end while
        i=Nk
        while (i<Nb×(Nr+1))
            word temp=w[i-1]
                if (i mod Nk=0)
                    temp=SubWord(RotWord(temp))⊕
                    ⊕Rcon[i/Nk]
                else if (Nk=8 and i mod Nk=4)
                    temp=SubWord(temp)
                end if
            w[i]=w[i-Nk]⊕temp
            i=i+1
        end while
    end
```

Пример 4.4. Предположим, что криптографический ключ K = 36 8a c0 f4 ed cf 76 a6 08 a3 b6 78 31 31 27 6e
В соответствии с приведенным алгоритмом, первые четыре слова ключа K для $Nk = 4$ имеют вид $w[0]$ = 368*ac*0*f*4, $w[1]$ = *edcf*76*a*6, $w[2]$ = 08a3b678, $w[3]$ = 3131276*e*.

Вычисления слова $w[4]$ для $i = 4$ выполняются в следующей последовательности. В соответствии с приведенной процедурой *temp* = $w[3]$ = 3131276*e*. Его циклическая перестановка на один байт принимает вид ***RotWord***($w[3]$) = 31276e31. Выбирая каждый байт предыдущего результата ***RotWord***($w[3]$) последовательно применяя подстановки в соответствии с нелинейной функцией подстановки (***S-box***), приведенной в таблице 4.13, получим ***SubWord***(31276e31) = c7cc9fc7.

Далее вычисляется константа ***Rcon***[*i*/*Nk*]=***Rcon***[4/4]=***Rcon***[1]=01000000. Операция сложения по модулю два выполняется как ***SubWord***(c7cc9fc7)⊕ ***Rcon***[01000000]=*c*6*cc*9*fc*7.

Учитывая, что $w[i - Nk] = w[0]$ = 368*ac*0*f*4, окончательно для $w[4]$ имеем $w[4] = c6cc9fc7 \oplus 368ac0f4 = f0465f33$. Аналогично получаются и остальные слова $w[i]$ расширенного ключа.

4.4.3. Шифрование в стандарте AES

Входными данными для AES является 128-битный входной блок, преобразованный в матрицу состояния таким образом, что последовательно формируются столбцы по четыре байта в каждом в соответствии с таблицей 4.14.

Таблица 4.14.

Матрица входного блока AES

a_0	a_4	a_8	a_{12}
a_1	a_5	a_9	a_{13}
a_2	a_6	a_{10}	a_{14}
a_3	a_7	a_{11}	a_{15}

Процедура шифрования выполняется в соответствии со следующим кодом.

```
Cipher (byte in [4×Nb], byte out [4×Nb], word w[Nb×(Nr+1)])
    begin
        byte state [4,Nb]
        State=in
        AddRoundKey (state,w)
            for round=1 step 1 to Nr-1
                SubBytes (state)
                ShiftRows(state)
```

MixColumns(state)
AddRoundKey(state, w+round×Nb)
end for
SubBytes (state)
ShiftRows(state)
AddRoundKey(state, w+Nr×Nb)
out=state
end

При реализации процедуры шифрования используются преобразования *SubBytes (state), ShiftRows(state), MixColumns(state), AddRoundKey(state, w+round×Nb*). Последовательно рассмотрим каждое из них.

Преобразование SubBytes () заключается в выполнении операции подстановки для каждого байта независимо с использованием *S-box* (см. таблицу 4.13), где старшие четыре бита задают номер строки, а младшие - номер столбца.

В соответствии с ***преобразованием ShiftRows*** (), первая строка (строка с номером 0) состояния не сдвигается, а последующие сдвигаются в соответствии с формулой

$$s^*_{r,c} = s^*_{r,(c+shift(r,Nb)) \bmod Nb}, \text{ для } 0 < r < 4 \text{ и } 0 \le c < Nb,$$

где количество сдвигов $shift(r,Nb) = shift(r, 4)$ зависит от номера строки r следующим образом

$$shift(1, 4) = 1;\ shift(2, 4) = 2;\ shift(3, 4) = 3;$$

Преобразование MixColumns () оперирует с состоянием по столбцам, рассматривая столбец как полином над $GF(2^8)$ умножая их по модулю $x^4 + 1$ на фиксированный полином $a(x) = (03)x^3 + (01)x^2 + (01)x + (02)$. Тогда

$$s'(x) = a(x) \otimes s(x)$$

где $s(x)$ есть входной полином, а $s'(x)$ полином результата.

В матричной форме $s'(x)$представляется как

$$\begin{vmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{vmatrix} = \begin{vmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{vmatrix} \times \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix} \text{ для } 0 \le c < Nb$$

Результатом данной операции будет.

$$s'_{0,c} = ((02) \times s_{0,c}) \oplus ((03) \times s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{1,c} = s_{0,c} \oplus ((02) \times s_{1,c}) \oplus ((03) \times s_{2,c}) \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus ((02) \times s_{2,c}) \oplus ((03) \times s_{3,c})$$
$$s'_{3,c} = ((03) \times s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus ((02) \times s_{3,c})$$

В соответствии с ***преобразованием AddRoundKey***() расширенный ключ (***round key***)складывается по модулю два с состоянием. Каждый расширенный

ключ состоит из *Nb* слов. Эти *Nb* слов складываются со столбцом состояния в соответствии с выражением

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round \times Nb+c}] \text{ для } 0 \le c < Nb,$$

где $[w_i]$ есть слова ключа, а *round* принимает значения из диапозона $0 \le round \le Nr$. Сложение с исходным криптографическим ключом выполняется когда $round = 0$,

4.4.4. Дешифрирование в стандарте AES

Дешифрирование в стандарте AES основано на использовании следующих преобразований *InvShiftRows*(), *InvSubBytes*(), *InvMixColumns*() и *AddRoundKey*() для обеспечения инверсной процедуры по отношению к шифрованию.

Преобразование InvShiftRows() является инверсным по отношению к преобразованию *ShiftRows*(). Первая строка (*Row* 0) не сдвигается. Байты последующих трех строк (*Row* 1, *Row* 2, *Row* 3) циклически сдвигаются на различное количество байт, где количество сдвигов *shift (r,Nb)*, для *r*-ой строки и *Nb* = 4, равняется *shift(*1, 4*)* = 1, *shift(*2, 4*)* = 2, *shift(*3, 4*)* = 3, соответственно.

Тогда преобразование ***InvShiftRows***() выполняется как

$S'_{r,(c+shift(r,Nb))\bmod Nb} = s_{r,c}$, для $0 < r < 4$ и $0 \le c < Nb$

Преобразование InvSubBytes() является инверсной операцией подстановки, для которой используется инверсная таблица подстановки (***inverse S-box),*** применяемая для каждого байта состояния. Указанная таблица приведена ниже.

Таблица 4.15.

Инверсная таблица подстановки алгоритма AES

		y															
		0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*a*	*b*	*c*	*d*	*e*	*f*
x	*0*	*52*	*09*	*6a*	*d5*	*30*	*36*	*a5*	*38*	*bf*	*40*	*a3*	*9e*	*81*	*f3*	*d7*	*fb*
	1	*7c*	*e3*	*39*	*82*	*9b*	*2f*	*ff*	*87*	*34*	*8e*	*43*	*44*	*c4*	*de*	*e9*	*cb*
	2	*54*	*7b*	*94*	*32*	*a6*	*c2*	*23*	*3d*	*ee*	*4c*	*95*	*0b*	*42*	*fa*	*c3*	*4e*
	3	*08*	*2e*	*al*	*66*	*28*	*d9*	*24*	*b2*	*76*	*5b*	*a2*	*49*	*6d*	*8b*	*d1*	*25*
	4	*72*	*f8*	*f6*	*64*	*86*	*68*	*98*	*16*	*d4*	*a4*	*5c*	*cc*	*5d*	*65*	*b6*	*92*
	5	*6c*	*70*	*48*	*50*	*fd*	*ed*	*b9*	*da*	*5e*	*15*	*46*	*57*	*a7*	*8d*	*9b*	*84*
	6	*90*	*d8*	*ab*	*00*	*8c*	*bc*	*d3*	*0a*	*f7*	*e4*	*58*	*05*	*b8*	*b3*	*45*	*06*
	7	*d0*	*2c*	*1e*	*8f*	*ca*	*3f*	*of*	*02*	*c1*	*af*	*bd*	*03*	*01*	*13*	*8a*	*6b*
	8	*3a*	*91*	*11*	*41*	*4f*	*67*	*dc*	*ea*	*97*	*f2*	*cf*	*ce*	*f0*	*b4*	*e6*	*73*
	9	*96*	*ac*	*74*	*22*	*e7*	*ad*	*35*	*85*	*e2*	*f9*	*37*	*e8*	*1c*	*75*	*df*	*6e*
	a	*47*	*f1*	*1a*	*71*	*1d*	*29*	*c5*	*89*	*6f*	*b7*	*62*	*0e*	*aa*	*18*	*be*	*1b*
	b	*fc*	*56*	*3e*	*4b*	*c6*	*d2*	*79*	*20*	*9a*	*db*	*c0*	*fe*	*78*	*cd*	*5a*	*F4*
	c	*1f*	*dd*	*a8*	*33*	*88*	*07*	*c7*	*31*	*b1*	*12*	*10*	*59*	*27*	*80*	*ec*	*5f*
	d	*60*	*51*	*7f*	*a9*	*19*	*b5*	*4a*	*0d*	*2d*	*e5*	*7a*	*9f*	*93*	*c9*	*9c*	*ef*

	e	*a0*	*e0*	*3b*	*4d*	*ae*	*2a*	*f5*	*b0*	*c8*	*eb*	*bb*	*3c*	*83*	*53*	*99*	*61*
	f	*17*	*2b*	*04*	*7e*	*ba*	*77*	*d6*	*26*	*e1*	*69*	*14*	*63*	*55*	*21*	*0c*	*7d*

Преобразование InvMixColumns () является инверсным преобразованием по отношению к преобразованию *MixColumns*(). Это преобразование оперирует с состоянием по столбцам, причем каждый столбец рассматривается как полином над полем $GF(2^8)$, который умножается на фиксированный полином $a^{-1}(x)$ по модулю $x^4 + 1$. Если инверсное состояние $s'(x)$ представляется в виде матрицы, тогда

$$s'(x) = a^{-1}(x) \otimes s(x),$$

где $a^{-1}(x) = (0b)x^3 + (0d)x^2 + (09)x + (0e)$ в матричной интерпретации запишется как

$$\begin{vmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{vmatrix} = \begin{vmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{vmatrix} \times \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix} \text{ для } 0 \le c < Nb$$

Результатом данной операции будет.

$s'_{0,c} = ((0e) \times s_{0,c}) \oplus ((0b) \times s_{1,c}) \oplus ((0d) \times s_{2,c}) \oplus ((09) \times s_{3,c})$

$s'_{1,c} = ((09 \times s_{0,c}) \oplus ((0e) \times s_{1,c}) \oplus ((0b) \times s_{2,c}) \oplus ((0d) \times s_{3,c})$

$s'_{2,c} = ((0d) \times s_{0,c}) \oplus ((09) \times s_{1,c}) \oplus ((0e) \times s_{2,c}) \oplus ((0b) \times s_{3,c})$

$s'_{3,c} = ((0b) \times s_{0,c}) \oplus ((0b) \times s_{1,c}) \oplus ((09) \times s_{2,c}) \oplus ((0e) \times s_{3,c})$

Преобразование AddRoundKey () по своей сути является инверсным и поэтому оно используется как при шифровании, так и при дешифрировании без всяких изменений.

Процедура дешифрирования выполняется в соответствии со следующим кодом.

```
Inverse Cipher (byte in [4×Nb], byte out [4×Nb], word dw[Nb×(Nr+1)])
    begin
        byte state [4,Nb]
        state=in
        AddRoundKey (state,dw+Nr×Nb)
            for round=Nr-1 step -1 to 1
                InvSubBytes (state)
                InvShiftRows(state)
                InvMixColumns(state)
                AddRoundKey(state, dw+round×Nb)
            end for
        InvSubBytes (state)
        InvShiftRows(state)
        AddRoundKey(state, dw)
        out=state
    end
```

Все преобразования в шифре стандарта AES имеют строгое математическое обоснование. Сама структура и последовательность операций позволяют выполнять данный алгоритм эффективно как на 8-битных, так и на 32-битных процессорах. В структуре алгоритма заложена возможность параллельного исполнения некоторых операций, что на многопроцессорных рабочих станциях может поднять скорость шифрования в 4 раза.

Глава 5. Потоковые системы шифрования
5.1. Потоковые шифры

Характерной особенностью потоковых шифраторов является использование при их реализации криптографических ключей большой размерности часто равной длине исходного текста.

Различают два типа потоковых шифров ***синхронные*** (***synchronous***) и ***самосинхронизирующиеся*** (***self-synchronizing***) шифраторы. В первом случае последовательность значений ключа является независимой от открытого текста и шифротекста. Понятие синхронный шифратор, связано с тем, что для успешного дешифрирования требуется синхронизация между последовательностью значений ключа и зашифрованным текстом. В то время как для самосинхронизирующихся шифраторов значения ключа зависят либо от исходного текста, либо от шифротекста. Обобщенная структура синхронного потокового шифратора приведена на рис.5.1.

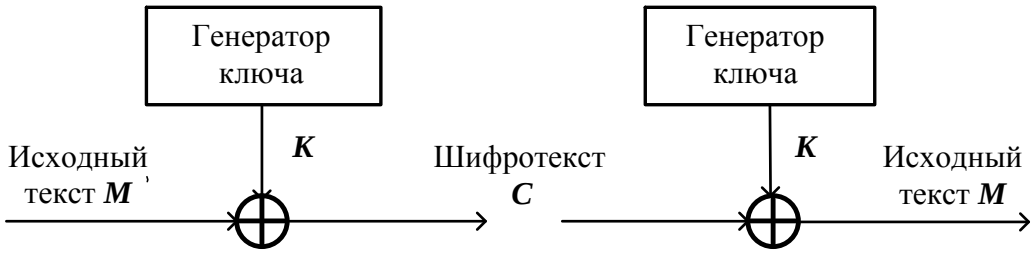


Рис.5.1. Структурная схема синхронного потокового шифратора

Подобная структура шифратора позволяет достичь высокой помехозащищенности и надежности функционирования. Действительно искажение одного символа в зашифрованном тексте приведет к искажению только одного символа исходного текста при дешифрировании. В тоже время нарушение синхронизации является причиной искажения всех последующих символов исходного текста с момента потери синхронного формирования ключа с шифротекстом. Подобная ситуация может возникнуть при удалении либо добавлении символов шифротекста.

С цель восстановления синхронизации используются самосинхронизирующиеся потоковые шифраторы. При их использовании ошибочно добавленный или удаленный бит вызывает только ограниченное количество ошибочных символов в дешифрированном тексте, после чего правильный текст восстанавливается.

Различают принципиальные отличия между симметричными блочными шифраторами (DES, IDEA, BLOWFISH, …) и потоковыми системами шифрования.

1.Блочный шифр одновременно шифрует целый блок данных, как правило, 64 бита, а в потоковых шифраторах процедура шифрования и дешифрирования выполняется поразрядно.

2.В блочном шифре, каждый бит зашифрованного текста является сложной функцией открытого текста, и криптографического ключа. В потоковом шифраторе (см. рис.5.1) каждый бит шифротекста получается в результате поразрядной операции сложения по модулю два очередного бита открытого текста и бита ключа.

3.Блочный шифр не требует задания начального вектора для генератора последовательности символов ключа, а в потоковом шифраторе такой вектор необходим.

4. Блочные шифры подобно как DES, используется в коммерческом секторе, а потоковые шифры используется для специальных приложений и, как правило, широко не обсуждаются.

5.2. Генераторы криптографического ключа

Основная составная часть любого потокового шифратора включает в себя алгоритм генерирования последовательности символов криптографического ключа. По сути, проблема создания эффективных симметричных блочных алгоритмов шифрования в случае потоковых шифраторов перенесена на проблему создания эффективных генераторов последовательностей ключа. Таким образом, генерирование непредсказуемых, как правило, двоичных последовательностей большой длины является одной из важных проблем классической криптографии. Для решения этой проблемы широко используются генераторы двоичных псевдослучайных последовательностей.

Обычно для генерирования последовательностей псевдослучайных чисел применяют аппаратные либо чаще всего программные генераторы, которые, хотя и называются генераторами случайных чисел, на самом деле формируют числовые последовательности, которые по своим свойствам очень похожи на случайные числа. Наиболее часто используемым алгоритмом генерирования псевдослучайных чисел в различных приложениях является ***линейный конгруэнтный генератор,*** описываемый рекуррентным соотношением

$$x_{t+1}=(ax_t+c) \bmod N,$$

где параметр x_0 называется начальным значением (вектором), величина $a\neq 0$ – множителем, параметр c – приращением, а значение N (мощность алфавита) модулем. В случае $c=0$ приведенное соотношение определяет ***мультипликативный конгруэнтный генератор***, а в случае $c\neq 0$ ***смешанный конгруэнтный генератор.*** Существует большое множество конкретных реализаций подобного генератора в разнообразных приложениях. Так мультипликативный конгруэнтный генератор для персональных компьютеров Pentium PC часто использует модуль $N=2^{31}$-

1=2147483647. При этом рекомендуемыми значениями множителя a являются следующие числа: 16807, 630360016, 1078318381, 1203248318, 397204094, 2027812808, 1323257245, 764261123, 112817.

Наиболее часто используемые генераторы псевдослучайных чисел приведены ниже.

1. $x_{t+1}=(1176x_t+1476x_{t-1}+1776x_{t-2}) \bmod (2^{32}-5);$

2. $x_{t+1}=(2^{13}(x_t+x_{t-1}+x_{t-2})) \bmod (2^{32}-5);$

3. $x_{t+1}=(1995x_t+1998x_{t-1}+2001x_{t-2}) \bmod (2^{32}-849);$

4. $x_{t+1}=(2^{19}(x_t+x_{t-1}+x_{t-2})) \bmod (2^{32}-1629);$

5. $x_{t+1}=(5115x_t+1776x_{t-1}+1492x_{t-2}+2111111111x_{t-3}+c_t) \bmod 2^{32};$

$c_t = \lfloor (5115x_{t-1}+1776x_{t-2}+1492x_{t-3}+2111111111x_{t-4}+c_{t-1})/ 2^{32} \rfloor.$

6. *COMBO:* $z_n=(x_n+y_n) \bmod 2^{32},$

$x_n=(x_{n-1}*x_{n-2}) \bmod 2^{32},$

$y_n=(30903y_{n-1}+c_n) \bmod 2^{16},$

$c_n=\lfloor (y_{n-1}+c_{n-1}) /2^{16} \rfloor.$

7. *KISS:* $z_n=(x_n+y_n+u_n) \bmod 2^{32},$

$x_n=(69069x_{n-1}+1) \bmod 2^{32},$

$y_n=y_{n-1}(I_{32}+L^{13})(I_{32}+R^{17})(I_{32}+L^{5}),$

$u_n=(2u_{n-1}+u_{n-2}+c_n) \bmod 2^{32},$

$c_n=\lfloor (2u_{n-2}+u_{n-3}+c_{n-1}) / \bmod 2^{32} \rfloor.$

Определяющим недостатком конгруэнтных шифраторов является их сложность реализации, которая заключается в необходимости выполнения операции умножения. Кроме того, данный класс генераторов является достаточно хорошо изученным и в силу этого легко поддающимся прогнозированию и соответственно взлому криптосистемы.

5.3. Генераторы *M* последовательностей

Генераторы *M* последовательностей являются одними из наиболее часто используемых типов генераторов псевдослучайных последовательностей применяемых в различных приложениях. Для их реализации используется сдвиговый регистр с линейной обратной связью (***LinearFeedback Shift Register - LFSR***). Подобный генератор часто используется как источник криптографического ключа для потоковых систем шифрования. Одна из основных причин широкого применения генераторов *M* последовательностей является сравнительно простая их реализация как программная, так и аппаратная. Кроме того, отмечаются высокие статистические свойства *M* последова- тельностей практически не отличающиеся от свойств случайных последовательностей. В тоже время генераторы *M* последовательностей подобно, как и любые другие генераторы

псевдослучайных последовательностей обеспечивают воспроизводимость выходных последовательностей. Подобные генераторы, далее их будем обозначать LFSR, строятся на основании примитивных порождающих полиномов. Так согласно примитивному порождающему полиному $\varphi(x)=1+x+x^4$, структурная схема LFSR представлена на рис.5.2.

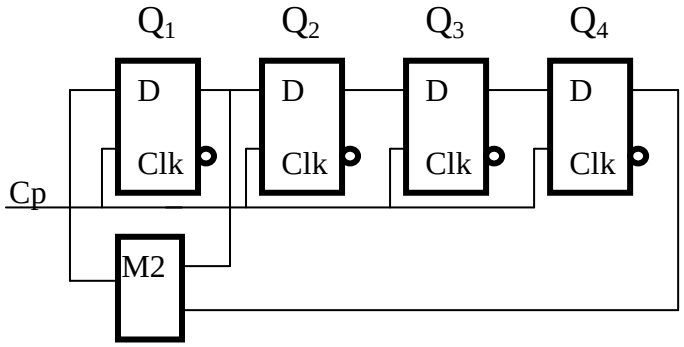


Рис.5.2. Структурная схема LFSR

Схема LFSR приведенного на рис.5.2 состоит из четырехразрядного регистра сдвига на один разряд вправо и двухвходового сумматора по модулю два включенного в цепь обратной связи. Аналитически функционирование приведенного генератора описывается системой уравнений.

$$Q_1(k+1)=Q_1(k)\oplus Q_4(k),$$
$$Q_2(k+1)=Q_1(k),$$
$$Q_3(k+1)=Q_2(k),$$
$$Q_4(k+1)=Q_3(k).$$

Последовательные состояния генератора приведены в таблице 5.1.

Таблица 5.1.

Состояния генератора LFSR

#	$Q_1Q_2Q_3Q_4$	#	$Q_1Q_2Q_3Q_4$
0	1 0 0 0	*8*	1 1 0 1
1	1 1 0 0	*9*	0 1 1 0
2	1 1 1 0	*10*	0 0 1 1
3	1 1 1 1	*11*	1 0 0 1
4	0 1 1 1	*12*	0 1 0 0
5	1 0 1 1	*13*	0 0 1 0
6	0 1 0 1	*14*	0 0 0 1
7	1 0 1 0	*15*	1 0 0 0

Как видно из приведенной таблицы, используя начальное состояние 1000, LFSR генерирует всевозможные четырехразрядные двоичные коды,

кроме нулевого кода (0000). Последовательность генерируемых кодов имеет случайный характер и не зависит от ненулевого начального состояния.

Для любой разрядности двоичных кодов существуют примитивные порождающие полиномы. Полиномы с минимальным количеством ненулевых коэффициентов для разрядностей от 1 и до 28 приведены в таблице 5.2.

Таблица 5.2.

Примитивные порождающие полиномы

$m=deg\varphi(x)$	$\varphi(x)$	$m=deg\varphi(x)$	$\varphi(x)$
1	$1+x$	15	$1+x+x^{15}$
2	$1+x+x^2$	16	$1+x^2+x^3+x^5+x^{16}$
3	$1+x+x^3$	17	$1+x^3+x^{17}$
4	$1+x+x^4$	18	$1+x^7+x^{18}$
5	$1+x^2+x^5$	19	$1+x+x^2+x^5+x^{19}$
6	$1+x+x^6$	20	$1+x^3+x^{20}$
7	$1+x+x^7$	21	$1+x^2+x^{21}$
8	$1+x+x^5+x^6+x^8$	22	$1+x+x^{22}$
9	$1+x^4+x^9$	23	$1+x^5+x^{23}$
10	$1+x^3+x^{10}$	24	$1+x^3+x^4+x^{24}$
11	$1+x^2+x^{11}$	25	$1+x^3+x^{25}$
12	$1+x^3+x^4+x^7+x^{12}$	26	$1+x+x^2+x^6+x^{26}$
13	$1+x+x^3+x^4+x^{13}$	27	$1+x+x^2+x^5+x^{27}$
14	$1+x+x^{11}+x^{12}+x^{14}$	28	$1+x^3+x^{28}$

Для произвольной степени $m=deg\varphi(x)$ порождающего полинома $\varphi(x) = a_0+a_1x+a_2x^2+\ldots+a_{m-1}x^{m-1}+a_mx^m$; $a_m=a_0=1$; $a_i\in\{0,1\}$ существуют две альтернативные структуры реализации LFSR с внешними и внутренними сумматорами по модулю два. На рис.5.3 приведена структура LFSR с внешними сумматорами по модулю два.

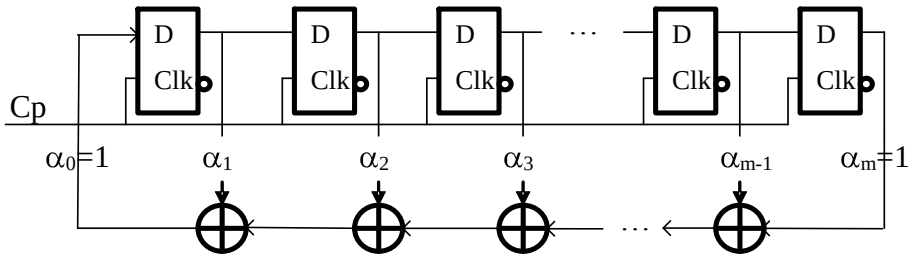


Рис.5.3. Структурные схемы LFSR с внешними сумматорами по модулю два

На рис.5.4 приведена структура LFSR с внутренними сумматорами по модулю два. Альтернативная структура LFSR описывается тем же примитивным порождающим полиномом, что и предыдущая структура.

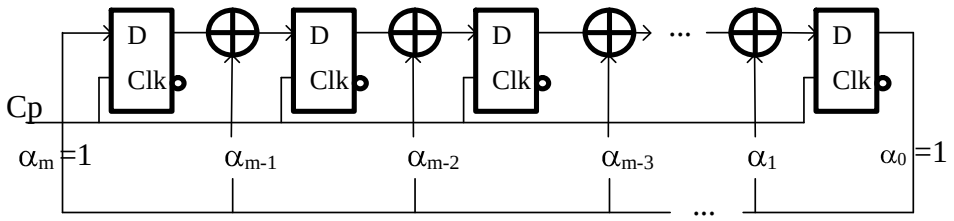
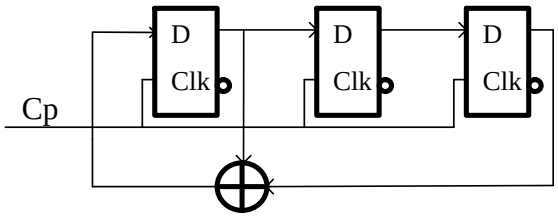


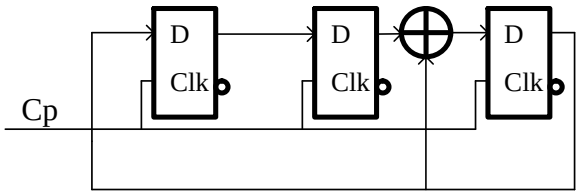
Рис.5.4. Структурные схемы LFSR с внутренними сумматорами по модулю два

Пример 5.1. В качестве примера LFSR с внешними и внутренними сумматорами рассмотрим случай порождающего полинома $\varphi(x)=1+x+x^3$ для которого ниже приведены соответствующие структуры и временные диаграммы формируемых последовательностей.



1 0 0
1 1 0
1 1 1
0 1 1
1 0 1
0 1 0
0 0 1

а)



1 0 0
0 1 0
0 0 1
1 0 1
1 1 1
1 1 0
0 1 1

б)

Рис.5.5. Структурные схемы LFSR для порождающего полинома $\varphi(x)=1+x+x^3$ с внешними** а) **и внутренними** б) **сумматорами по модулю два

Функционирование LFSR описывается следующим математическим соотношением, представленным системой уравнений.

$$a_1(k+1)=\sum_{i=1}^{m}\alpha_i a_i(k);$$
$$a_j(k+1)=a_{j-1}(k),\ j=\overline{2,m},k=0,1,2,\ldots$$

Матричное описание LFSR имеет вид.

$$\begin{vmatrix} a_1(k+1) \\ a_2(k+1) \\ a_3(k+1) \\ \ldots \\ a_m(k+1) \end{vmatrix} = \begin{vmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \ldots & \alpha_{m-1} & \alpha_m \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{vmatrix} \times \begin{vmatrix} a_1(k) \\ a_2(k) \\ a_3(k) \\ \ldots \\ a_m(k) \end{vmatrix}$$

В векторной форме будем иметь

$$A(k+1)=V\times A(k)$$

где V представляет собой порождающую (генерирующую) матрицу, а $A(k)$ вектор столбец текущего и $A(k+1)$ последующего состояния LFSR.

Псевдослучайные последовательности (M последовательности) характеризуются следующими свойствами.

1.Существует $\psi(2^m-1)/m$ примитивных полиномов для заданного значения его старшей степени m, где ψ есть функция Эйлера. Так, например, для $m=3$ $\psi(2^m-1)/m=\psi(2^3-1)/3=6/3=2$. Таким образом, для $m=3$ существует два примитивных полинома $\varphi(x)=1+x+x^3$ и $\varphi(x)=1+x^2+x^3$. Следует отметить, что функция Эйлера принимает большие значения даже небольших значений m.

2.Для заданного полинома $\varphi(x)$ существует инверсный полином $\varphi(x)^{-1}$, который может быть получен согласно следующему отношению

$$\varphi(x)^{-1}=x^m\varphi(x^{-1}).$$

Пример 5.2. Так, например, для порождающего полинома $\varphi(x)=1+x^2+x^5$ $\varphi(x)^{-1}=x^5\varphi(x^{-1})=x^5(1+x^{-2}+x^{-5})=1+x^3+x^5$.

3.Период M последовательности зависит от степени m примитивного порождающего полинома и равняется

$$L=2^m-1.$$

Отметим что LFSR, построенный на базе примитивного порождающего полинома φ(x) степени m, генерирует всевозможные m разрядные двоичные коды кроме кода, состоящего из всех нулей.

4.Для заданного полинома $\varphi(x)$ существует L различных M последовательностей, каждая из которых отличается фазовым сдвигом.

Пример 5.3. Для порождающего полинома $\varphi(x)=1+x+x^4$ существует 15 M последовательностей приведенных в таблице 5.3.

Таблица 5.3.

M последовательности, описываемые полиномом $\varphi(x)=1+x+x^4$

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

5.M последовательность представляет собой псевдослучайную последовательность, в которой вероятность появления нулей и единиц определяется соотношениями.

$$p(a_k=1)=\frac{2^{m-1}}{2^m-1}=\frac{1}{2}+\frac{1}{2^{m+1}-2};$$

$$p(a_k=1)=\frac{2^{m-1}-1}{2^m-1}=\frac{1}{2}-\frac{1}{2^{m+1}-2};$$

Как видно из приведенных соотношений вероятность появления нуля и единицы практически равняется 0.5.

7.Функция автокорреляции M последовательности максимально близка к автокорреляционной функции идеальной случайной последовательности.

Действительно оригинальная *M* последовательность является идентичной в $2^{m-1}-1$ позициях со сдвинутой своей копией на любое число тактов, и будет отличаться в 2^{m-1} позициях.

Так, например последовательность 000111101011001 генерируемая в соответствии с полиномом $\varphi(x)=1+x+x^4$ и ее сдвинутая на две позиции копия 011110101100100 совпадают на семи позициях и имеют различное значение на восьми позициях.

8.Свойство ***сдвига и сложения***. Для любого s $(1 \le s < L)$ существует $r \ne s$ $(1 \le r < L)$ такое что $\{a_k\} \oplus \{a_{k-s}\} = \{a_{k-r}\}$.

$\{a_0\}$ 000111101011001
$\{a_{-2}\}$ 011110101100100
$\{a_{-9}\}$ 011001000111101

9.Среди *L M* последовательностей, полученных на основании примитивного полинома $\varphi(x)$ существует единственная *M* - последовательность, для которой выполняется равенство

$$a_k = a_{2k}, \quad k=0,1,2,....$$

Такая *M* последовательность называется ***характеристикой*** и получается следующим образом. Для заданного примитивного порождающего полинома строится система линейных уравнений

$$a_i = a_{2i}, \; i=0,1,2,..., \; m-1.$$

Не нулевое решение приведенной системы уравнений и есть искомая характеристическая последовательность.

Пример 5.4. В качестве примера рассмотрим процедуру определения характеристической последовательности для полинома $\varphi(x)=1+x+x^4$. Для этого случая система линейных уравнений имеет форму

$$a_0 = a_0;$$
$$a_1 = a_2;$$
$$a_2 = a_4 = a_0 \oplus a_3;$$
$$a_3 = a_6 = a_2 \oplus a_5 = a_1 \oplus a_1 \oplus a_4 = a_0 \oplus a_3.$$

Единственное ненулевое решение $a_0a_1a_2a_3=0111$ удовлетворяющее приведенной системе и есть искомая характеристическая *M* последовательность. В таблице 5.3 эта последовательность приведена под номером 2.

10.Свойство ***децимаций***. Децимация *M* последовательности $\{a_i\}$ по индексу q, $(q=1,2,3,...)$ означает порождение другой последовательности $\{b_j\}$ как q-тые элементы первоначальной *M* последовательности $\{a_i\}$, то есть $b_j = qa_i$. Если наибольший общий делитель периода *L M* последовательности и индекса q равен единице, то есть $(L,q)=1$ то период $\{b_j\}$ будет равен $L=2^m-1$, и децимация называется нормальной.

Пример 5.5. В качестве примера рассмотрим нормальную децимацию *M* последовательности соответствующей полиному $\varphi(x)=1+x+x^4$ по индексу два. В результате получим

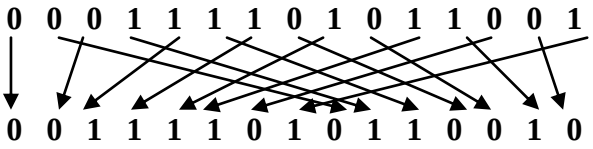


Рис.5.6. Пример децимации М-последовательности

Как видно из предыдущего примера в результате децимации получили *M* последовательность, описываемую тем же полиномом. В общем случае результатом децимации может быть любая *M* последовательность, описываемая любым примитивным порождающим полиномом той же степени *m.*

5.4. Генераторы нелинейных последовательностей

В общем случае генераторы нелинейных последовательностей строятся с использованием нелинейных зависимостей для получения очередного выходного значения. Чаще всего для этих целей используется регистры сдвига с нелинейной обратной связью *F*. Структура подобного генератора включает в себе два основных блока, а именно сдвиговой регистр на один разряд вправо и комбинационную схему, реализующую нелинейную функцию, описывающую обратную связь. Аналогично регистрам сдвига с линейной обратной связью (*LFSR*) подобные устройства называются ***регистрами сдвига с нелинейной обратной связью*** (***Nonlinear Feedback Shift Register*** (***NFSR***))
Общий вид подобного устройства приведен на рисунке 5.7.

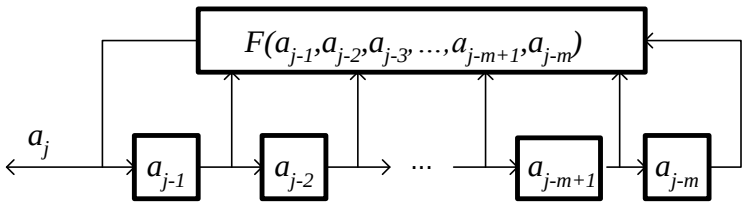


Рис.5.7. Регистр сдвига с нелинейной обратной связью

Хорошо изученной разновидностью подобных генераторов являются генераторы ***последовательностей де-Брейна*** (***De-Bruijn***), характерной чертой, для которых является генерирование всевозможных 2^m двоичных комбинаций, где *m* является разрядностью регистра сдвига, на котором формируется последовательность де Брейна. Существует процедура преобразования *LFSR* в *NFSR* генерирующего последовательность де Брейна. В этом случае нелинейная функция $F(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1},a_{j-m})$ образуется как

сумма по модулю два функции $f(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1},a_{j-m})$, описывающей линейную обратную связь LFSR, с нелинейной функцией $g(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1})=a^*_{j-1},a^*_{j-2},a^*_{j-3},\ldots,a^*_{j-m+2},a^*_{j-m+1}$. Здесь $a^*_i=1\oplus a_i$.

Вид нелинейной функции $g(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1})=a^*_{j-1},a^*_{j-2},a^*_{j-3},\ldots,a^*_{j-m+2},a^*_{j-m+1}$ свидетельствует о том, что данная функция принимает единичное значение только на двух наборах переменных (двух состояний регистра сдвига), а именно на наборе 000…001 и 000…000. Отметим, что в последовательности кодов формируемых LFSR присутствует только код 000…001, и нет больше кодов, у которых все первые *m-1* бита, принимают нулевое значение. Кроме того, независимо от вида конкретного примитивного порождающего полинома, функция $f(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1},a_{j-m})$, описывающая линейную обратную связь LFSR, всегда принимает единичное значение на наборе 000…001.

Таким образом, при появлении на регистре сдвига набора 000…001 обе функции и *g* и *f* принимают единичное значение, а *F* соответственно нулевое в силу чего на регистре сдвига будет получен код 000…000, который будет являться следующим набором аргументов для указанных функций *g*, *f* и *F*. В следующем цикле *g* будет ровняться единице, а *f* нулю, тогда *F*=1, и на регистре сдвига будет получен код 100…000.

Пример 5.6. Простейшим примером генератора последовательности де Брейна является генератор представленный на рисунке 5.8.

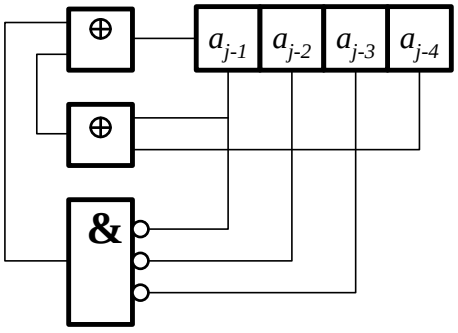


Рис.5.8. Генератор последовательности де Брейна

Данный генератор синтезирован на базе LFSR соответствующего порождающему полиному $\varphi(x)=1\oplus x^1\oplus x^4$ линейная обратная связь, которого описывается функцией $f(a_1,a_2,a_3,a_4)=a_1\oplus a_4$. Нелинейная функция $F(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1},a_{j-m})$ для данного примера принимает вид $F(a_{j-1},a_{j-2},a_{j-3},\ldots,a_{j-m+1},a_{j-m})= a_1\oplus a_4\oplus(a^*_1a^*_2a^*_3)$. Диаграмма состояний генератора приведена в таблице.

Таблица 5.4.

Диаграмма состояний генератора последовательности де Брейна

#	a1	a2	a3	a4	#	a1	a2	a3	a4
1	1	0	0	0	9	1	1	0	1

2	1	1	0	0	10	0	1	1	0
3	1	1	1	0	11	0	0	1	1
4	1	1	1	1	12	1	0	0	1
5	0	1	1	1	13	0	1	0	0
6	1	0	1	1	14	0	0	1	0
7	0	1	0	1	15	0	0	0	1
8	1	0	1	0	16	0	0	0	0

Последовательность де Брейна, как видно из приведенного примера, может быть получена путем добавления нулевого символа к последовательности 2^m-1 бит, таким образом, что на периоде ее повторения образовалась серия из m нулей. Такая последовательность соответственно имеет период равный 2^m и характеризуется наличием на интервале повторения всевозможных кодов длиной m, каждый из которых встречается только один раз. Последнее свойство и определило интерес к использованию последовательностей де Брейна для различных приложений в криптографии.

Частным случаем последовательностей де Брейна являются так называемые ***последовательности Форда***, которые описываются более простым алгоритмом формирования. Последний носит рекуррентный характер и заключается в формировании очередного m-разрядного кода $X_k=(b_2,b_3,b_4,\ldots,b_{m+1})$ на основе предыдущего кода $X_{k-1}=(b_1,b_2,b_3,\ldots,b_m)$, где значение $b_{m+1}\in\{0,1\}$ определено следующим образом. Формируется код вида $X^*_{k-1}=(b_2,b_3,b_4,\ldots,b_m,1)$ и строятся всевозможные его циклические сдвиги, среди них выбирается код $X^{**}_{k-1}=(b_i,b_{i+1},\ldots,b_m,1,b_2,\ldots,b_{i-1})$, представляющий собой наибольшее m-разрядное число. Если $b_2=\ldots=b_{i-1}=0$, то значение b_{m+1} равно $b_{m+1}=b_1\oplus 1$, а в противном случае $b_{m+1}=b_1$. Отметим, что начальным кодом X_0 для формирования последовательности Форда может быть любой код, включая $X_0=(0,0,0,\ldots,0)$.

В качестве примера в таблице 5.5 приведены результаты формирования последовательности Форда для $m=4$.

Таблица 5.5.

Диаграмма состояний генератора последовательности Форда

k	$X_k=$ $=(b_2,b_3,b_4,\ldots,b_{m+1})$	$X^{**}_{k-1}=$ $=(b_i,b_{i+1},\ldots,b_m,1,b_2,\ldots,b_{i-1})$,	b_{m+1}

0	*0 0 0 0*	*1 0 0 0*	$b_1 \oplus 1 = 0 \oplus 1 = 1$
1	*0 0 0 1*	*1 1 0 0*	$b_1 \oplus 1 = 0 \oplus 1 = 1$
2	*0 0 1 1*	*1 1 1 0*	$b_1 \oplus 1 = 0 \oplus 1 = 1$
3	*0 1 1 1*	*1 1 1 1*	$b_1 \oplus 1 = 0 \oplus 1 = 1$
4	*1 1 1 1*	*1 1 1 1*	$b_1 \oplus 1 = 1 \oplus 1 = 0$
5	*1 1 1 0*	*1 1 1 0*	$b_1 = 1$
6	*1 1 0 1*	*1 1 1 0*	$b_1 = 1$
7	*1 0 1 1*	*1 1 1 0*	$b_1 \oplus 1 = 1 \oplus 1 = 0$
8	*0 1 1 0*	*1 1 1 0*	$b_1 = 0$
9	*1 1 0 0*	*1 1 0 0*	$b_1 = 1$
10	*1 0 0 1*	*1 1 0 0*	$b_1 \oplus 1 = 1 \oplus 1 = 0$
11	*0 0 1 0*	*1 0 1 0*	$b_1 \oplus 1 = 0 \oplus 1 = 1$
12	*0 1 0 1*	*1 1 1 0*	$b_1 = 0$
13	*1 0 1 0*	*1 0 1 0*	$b_1 \oplus 1 = 1 \oplus 1 = 0$
14	*0 1 0 0*	*1 1 0 0*	$b_1 = 1$
15	*1 0 0 0*	*1 0 0 0*	$b_1 \oplus 1 = 1 \oplus 1 = 0$
16	*0 0 0 0*	*1 0 0 0*	…
…	…		

Весьма близкими по свойствам к *M* последовательностям и находящими наряду с ними широкое практическое применение являются ***последовательности Голда и Касами***. Последовательность Голда образуется путем суммирования по модулю два двух *M* последовательностей, порождаемых отличными примитивными полиномами $\varphi'(x)$ и $\varphi''(x)$ одной и той же степени *m*. Период данной последовательности равен 2^m-1, а порождающий ее полином представляет собой произведение $\varphi'(x)$ на $\varphi''(x)$.

Более широким классом последовательностей, включающим в качестве подмножества последовательности Голда, является множество последовательностей Касами. Последние могут быть получены в результате двоичного сложения трех *M* последовательностей $\{a_i\}$, $\{b_i\}$ и $\{c_i\}$, порождаемых полиномами $\varphi'(x)$ и $\varphi''(x)$ степени *m* и $\varphi^*(x)$степени *m/2* соответственно, где *m* –четно. Наиболее важное качество последовательностей Касами, а также входящих в них подмножеств последовательностей, является их высокие корреляционные свойства.

5.5. Комбинированные генераторы ключа

Как отмечалось в предыдущих разделах генераторы *M* последовательностей (*LFSR*) являются весьма эффективным средством для генерирования криптографического ключа. Обычно для этих целей можно использовать один *LFSR* или несколько подобных структур, желательно, описываемых различными порождающими полиномами. Если длины формируемых последовательностей являются взаимно простыми и порождающие полиномы примитивными оказывается возможным получить

последовательность бит ключа максимальной длины. ***Сеансовым ключам*** (ключом, используемым для шифрования одного сообщения) может быть начальное состояние используемых *LFSR.*

Очередной бит ключа формируется в соответствии с некоторой функцией, как правило, нелинейной. Эта функция называется ***комбинированной функцией*** (***combining function***), а сами генераторы ключа ***комбинированными генераторами*** (***combination generator***).

Одним из наиболее очевидных решений при построении комбинированных генераторов ключа является ***генератор Геффи*** (***Geffe generator***), В основе данного генератора лежит использование трех *LFSR* и двухвходового мультиплексора как показано на рис.5.9.

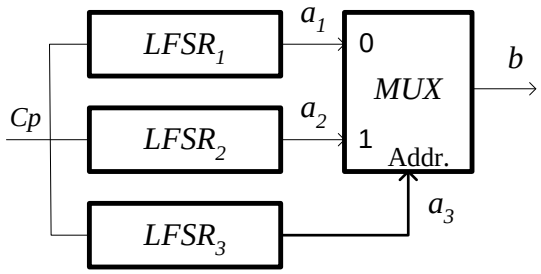


Рис.5.9. Комбинированный генератор Геффи

Два генератора *M* последовательностей $LFSR_1$ и $LFSR_2$ являются источниками последовательностей a_1 и a_2 подаваемыми на информационные входы мультиплексора, а последовательность a_3 формируемая $LFSR_3$, подается на адресный вход мультиплексора. Все три *LFSR* функционируют синхронно при подаче синхронизирующих импульсов *Cp*. Выходная последовательность *b* формируется в соответствии со следующим выражение $b=(a_2a_3)+ (a_1(1 \oplus a_3))$, а период равен наименьшему общему кратному периодов трех *M* последовательностей генерируемых $LFSR_1$, $LFSR_2$ и $LFSR_3$. Если их порождающие полиномы имеют степени *m1*, *m2* и *m3* то максимальный период последовательности *b* будет равен $(2^{m1}-1)\times(2^{m2}-1)\times(2^{m3}-1)$. Обычно $m1 \neq m2 \neq m3$, а их величины близки и, как правило, определяются размерностью сеансового ключа.

Возможны различные модификации генератора Геффи в части увеличения количества используемых *LFSR*, а также режимов их функционирования.

Идея модификации режимов функционирования *LFSR* используется в ***генераторе Бета-Пайпера*** (***Beth-Piper Stop-and-Go generator***). В отличие от предыдущего метода используемые в нем *LFSR* синхронизируются различными сигналами. Для случая трех *LFSR* структурная схема такого генератора приведена на рис.5.10.

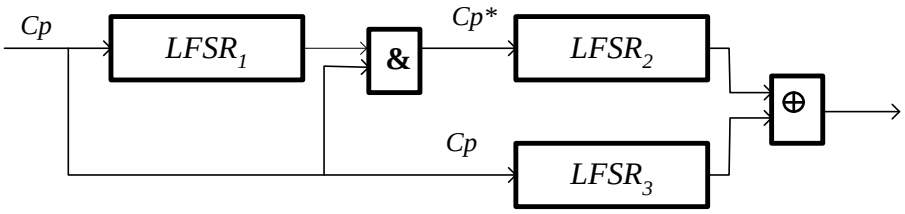


Рис.5.10.Генератор Бета-Пайпера

Как видно из приведенного рисунка, $LFSR_1$ управляет синхронизацией $LFSR_2$, а результирующая выходная последовательность определяется как сумма по модулю два выходных последовательностей формируемых $LFSR_2$ и $LFSR_3$. Функционирование $LFSR_2$ определяет термин ***Stop-and-Go***, действительно в зависимости от выходного значения $LFSR_1$, данный блок ($LFSR_2$) выполняет микрооперацию сдвига либо нет, то есть стоит либо идет. При выборе порождающих полиномов для трех *LFSR* генератора Бета-Пайпера руководствуются теми же требованиями, что и в случае генератор Геффи. Тогда оказывается возможным получение максимально возможной длины генерируемой последовательности.

Видоизмененная структура генератора Бета-Пайпера (***Alternating Stop-and-Go generator***) имеет большее распространение и также основана на использовании трех генераторов *M* последовательностей $LFSR_1$, $LFSR_2$ и $LFSR_3$. Подобно, как и в предыдущих случаях *M* последовательности, генерируемые тремя генераторами, имеет различные взаимно простые периоды. Как видно из рис. 5.11 $LFSR_1$ управляет синхронизацией $LFSR_2$ и $LFSR_3$. Причем если один из них выполняет микрооперацию сдвига (идет) то второй в это время не выполняет (стоит). Выходная последовательность, как и в предыдущих случаях, формируется как сумма по модулю два выходных последовательностей формируемых $LFSR_2$ и $LFSR_3$.

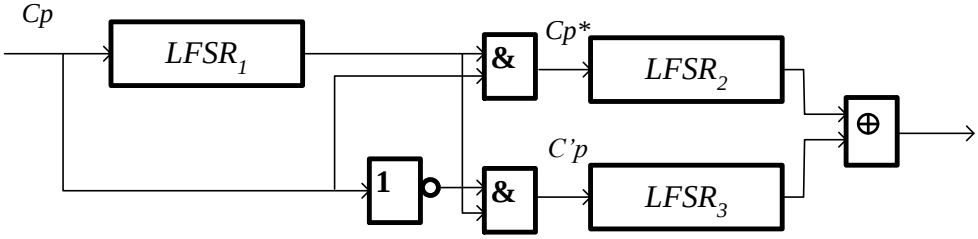


Рис.5.11. Альтернативный генератор Бета-Пайпера

Дальнейшим развитием идеи внесения нелинейных зависимостей по управлению синхронизацией является каскадная схема Голмана (***Gollmann Cascaded Key stream generator***), которая позволяет достигать желаемого качества в зависимости от количества используемых каскадов. Для случая трех каскадов подобный генератор приведен на рис.5.12.

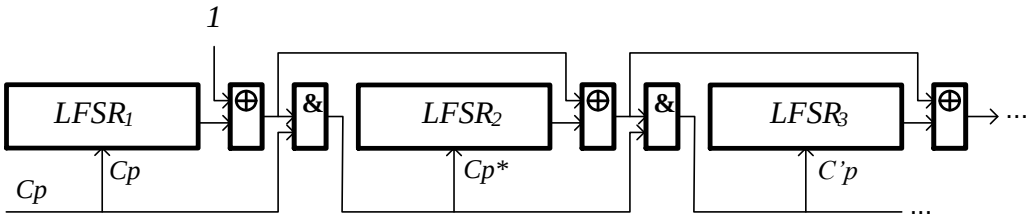


Рис.5.12. Каскадная схема Голмана

Основным достоинством приведенной схемы является возможность ее адаптации для достижения, желаемого качества. Отметим, что синхронизация второго и последующих каскадов управляется предыдущими каскадами. Выполнение микрооперации сдвига либо ее отсутствие для конкретного *LFSR* завит от состояний на выходах всех предыдущих генераторов *M* последовательностей. В тоже время необходимо отметить, что данная микрооперация выполняется для каждого *LFSR* с вероятностью 0,5.

Большое количество *LFSR* использует так называемый ***пороговый генератор*** (***Threshold generator***), приведенный на рис.5.13.

Количество (*m*) *LFSR* выбирается нечётным тогда нелинейная функция *F* формирует выходное значение 0 или 1 в зависимости от соотношения единичных и нулевых значений на выходах генераторов *M* последовательностей.

Если количество единичных значений больше нулевых на выходе генератора формируется единичное значение, в противном случае выходное значение ровняется нулю.

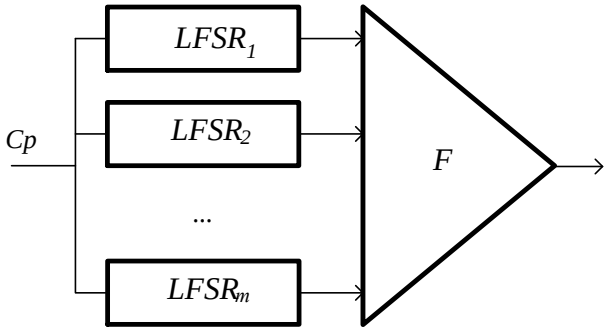


Рис.5.13. Пороговый генератор

Подобно, как и в предыдущих случаях, периоды *M* последовательностей должны быть взаимно простыми, а степени порождающих их полиномов иметь сравнимые величины и, как правило, больше 10.

5.6. Синхронные потоковые шифраторы

Синхронные потоковые шифраторы основаны на синхронном функционировании генераторов ключа у отправителя и получателя

сообщения. Процедура синхронизации обеспечивается ***начальным состоянием*** (***seed***) генераторов псевдослучайных последовательностей криптографического ключа. В случае использования простейших генераторов ключа, таких как генераторы (см. рис. 5.3 и рис. 5.4) *M* последовательностей типа LFSR, синхронизация достигается за счет установки регистров генераторов ключа у отправителя и получателя в одно и тоже состояние. Начальное состояние генератора ключа часто интерпретируется как ***сеансовый ключ***, который используется только для одного сеанса передачи данных, а структура генератора как ***долгосрочный ключ***, используемый для заданного временного промежутка использования криптосистемы. Структура синхронного потокового шифратора приведена на рис. 5.14.

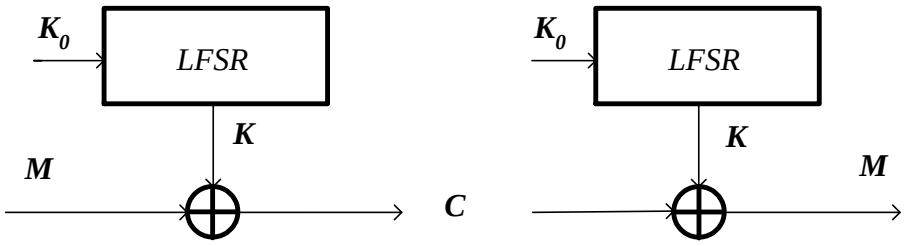


Рис.5.14. Синхронный потоковый шифратор

Здесь *K* представляет собой криптографический ключ, длина которого равняется длине шифруемого сообщения, а K_0 представляет собой сеансовый ключ.

Пример 5.7. В качестве примера рассмотрим потоковый шифратор (рис.5.15.), использующий в качестве ключа *M* последовательность, генерируемую в соответствии с примитивным порождающим полиномом $\varphi(x)=1+x+x^4$. Предположим, что начальное состояние генератора ключа K_0 при шифровании и при дешифрировании сообщения принимает одно и тоже значение равное 0010, что позволит нам достичь синхронной работы криптографической системы в целом. Отметим, что в данном случае сеансовым ключом может быть любой код из четырех символов, кроме кода 0000. В соответствии с выбранным начальным состоянием обоих генераторов у отправителя и получателя сообщения будет сгенерирована одна и та же последовательность ключа *K*=011110101100100….

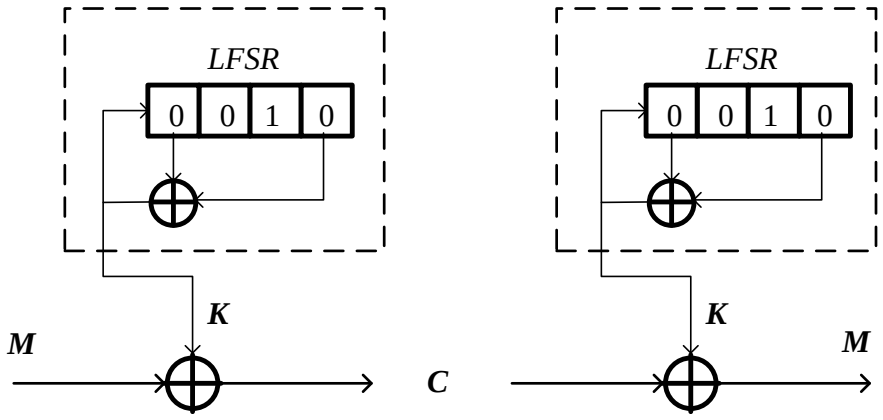


Рис.5.15. Синхронный потоковый шифратор, описываемый полиномом $\varphi(x)=1+x+x^4$.

В качестве исходного текста используем двоичную последовательность M=101011111100001, тогда шифрование будет состоять из последовательного поразрядного сложения по модулю два. В результате получим.

$$\begin{array}{ll} M & = 101011111100001 \\ \oplus & \qquad\qquad \oplus \\ K & = 011110101100100 \\ \hline C & = 110101010000101 \end{array}$$

Получатель сообщения, получив зашифрованный текст C=11010101000 0101, выполнит обратное преобразование.

$$\begin{array}{ll} C & = 110101010000101 \\ \oplus & \qquad\qquad \oplus \\ K & = 011110101100100 \\ \hline\hline M & = 101011111100001 \end{array}$$

К сожалению, многие генераторы псевдослучайных последовательностей криптографических ключей подвержены различного рада атакам и зачастую легко взламываются. В данном случае под взломом понимается получение третей стороной структуры генератора криптографического ключа, то есть так называемого долгосрочного ключа. В случае генератора M последовательности его структура однозначно описывается порождающим полиномом $\varphi(x) = a_0+a_1x+a_2x^2+\ldots+a_{m-1}x^{m-1}+a_mx^m$; $a_m=a_0=1$; $a_i\in\{0,1\}$, и для его взлома злоумышленнику необходимо получит $2m$ бит шифротекста, что почти всегда достижимо и столько же бит соответствующего ему зашифрованного текста. Что тоже является реальной задачей, тогда непосредственно взлом будет состоять в нахождении решения системы из m линейных уравнений неизвестными, в которой будут коэффициенты порождающего полинома, а коэффициенты значениями $2m$

бит последовательности ключа *K*. Такая система уравнений для общего случая имеет.

$$k_{m+1} = \alpha_1 k_m \oplus \alpha_2 k_{m-1} \oplus \alpha_3 k_{m-2} \oplus ... \oplus \alpha_{m-1} k_2 \oplus k_1;$$
$$k_{m+2} = \alpha_1 k_{m+1} \oplus \alpha_2 k_m \oplus \alpha_3 k_{m-1} \oplus ... \oplus \alpha_{m-1} k_3 \oplus k_2;$$
$$k_{m+3} = \alpha_1 k_{m+2} \oplus \alpha_2 k_{m+1} \oplus \alpha_3 k_m \oplus ... \oplus \alpha_{m-1} k_4 \oplus k_3;$$
$$\dots$$
$$k_{2m} = \alpha_1 k_{2m-1} \oplus \alpha_2 k_{2m-2} \oplus \alpha_3 k_{2m-3} \oplus ... \oplus \alpha_{m-1} k_{m=1} \oplus k_m;$$

Здесь коэффициент a_m, так же как и a_0 для любого полинома равен единице.

Пример 5.8. В качестве примера рассмотрим потоковый шифратор, приведенный на рис.5.14. В случае если злоумышленник получит доступ к $2m=2\times4=8$ битам исходного текста M=10101111 и C=11010101. Тогда он сможет восстановить элементы ключа *K,* которые будут равны поразрядной сумме по модулю два исходного текста и шифротекста. Соответственно получим: k_1=1⊕1=0, k_2=0⊕1=1, k_3=1⊕0=1, k_4=0⊕1=1, k_5=1⊕0=1, k_6=1⊕1=0, k_7=1⊕0=1, k_8=1⊕1=0. Система из $m=4$ линейных уравнений имеет вид

$$k_5 = \alpha_1 k_4 \oplus \alpha_2 k_3 \oplus \alpha_3 k_2 \oplus \alpha_4 k_1;$$
$$k_6 = \alpha_1 k_5 \oplus \alpha_2 k_4 \oplus \alpha_3 k_3 \oplus \alpha_4 k_2;$$
$$k_7 = \alpha_1 k_6 \oplus \alpha_2 k_5 \oplus \alpha_3 k_4 \oplus \alpha_4 k_3;$$
$$k_8 = \alpha_1 k_7 \oplus \alpha_2 k_6 \oplus \alpha_3 k_5 \oplus \alpha_4 k_4;$$

Окончательно будем иметь $1=\alpha_1\oplus\alpha_2\oplus\alpha_3$; $0=\alpha_1\oplus\alpha_2\oplus\alpha_3\oplus1$; $1=\alpha_2\oplus\alpha_3\oplus1$ и $0=\alpha_1\oplus\alpha_3\oplus1$. Откуда получим $\alpha_2=0$, $\alpha_3=0$ и $\alpha_1=1$, что соответствует полиному, использованному в примере 5.7.

С целью увеличения качества шифрования можно использовать комбинированный подход применения классических симметричных и потоковых криптосистем.

Пример 5.9. В качестве примера подобных решений рассмотрим случай совместного применения алгоритма шифрования DES и LFSR.В данном случае поток бит ключа формируется на выходе системы шифрования DES. Начальное значение ключа K_0 определяет исходное состояние *LFSR* и может использоваться как криптографический ключ для DES. Его значение удобно рассматривать как сеансовый ключ комбинированной криптосистемы, в которой очередные 64 бита ключа получаются при шифровании, в соответствии с алгоритмом DES, очередного состояния *LFSR.*

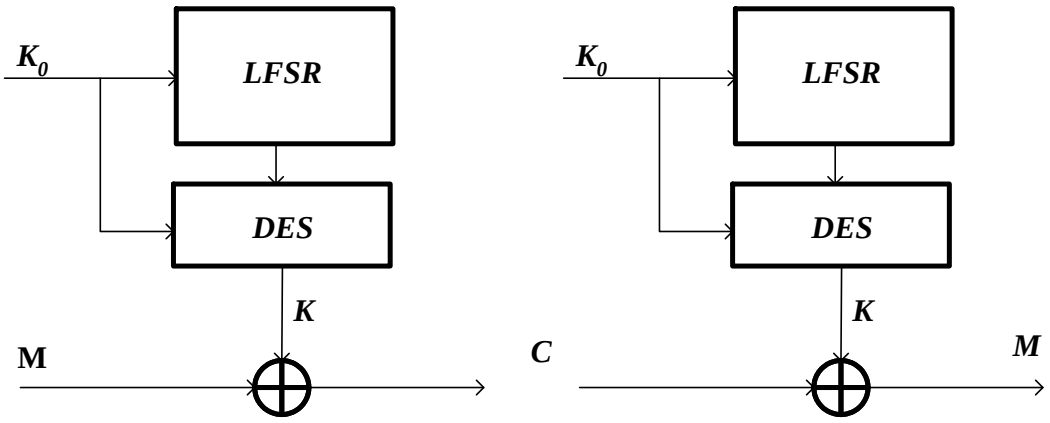


Рис.5.16. Комбинированная криптосистема

Могут быть использованы и другие комбинации классисеских криптографических систем и применения для них принципа потокового шифрования исходных сообщений.

5.7. Самосинхронизирующиеся криптосистемы

Синхронные потоковые криптосистемы наряду с большим числом очевидных достоинств имеют ряд недостатков. Одним из наиболее значимых недостатков является катастрофические последствия для потоковой криптосистемы при нарушении синхронизации при передаче зашифрованного текста. Эффект нарушения синхронизации заключается в пропадании символа или нескольких символов шифротекста, ровно как и их размножении. С момента нарушения синхронизации весь последующий шифротекст будет смещен относительно последовательности ключа, что не позволит правильно дешифрировать исходный текст.

С целью уменьшения эффекта нарушения синхронизации используются так называемые самосинхронизирующиеся потоковые криптосистемы (***Self-synchronizing cipher***) которые подразделяются на ***самосинхронизирующиеся потоковые криптосистемы по шифротексту*** (***ciphertext auto key cipher***) и по ***исходному тексту*** (***plaintext auto key cipher***).

Самосинхронизирующиеся потоковые криптосистемы по шифротексту реализуются в соответствии с примитивным порождающим полиномом $\varphi(x) = a_0+a_1x+a_2x^2+\ldots+a_{m-1}x^{m-1}+a_mx^m$; $a_m=a_0=1$; $a_i\in\{0,1\}$, который определяет структуру блока шифрования и дешифрирования. Общий вид блока шифрования представлен на рис 5.17.

Как видно из приведенной структуры очередной бит ключа K является функцией порождающего полинома $\varphi(x)$ и n предыдущих бит шифротекста C. Начальное состояние регистра сдвига $S_0=s_1s_2\ldots s_{n-1}$ используется как сеансовый ключ и может принимать произвольные значения, в том числе и нулевое значение которое является недопустимым в случае синхронных криптосистем.

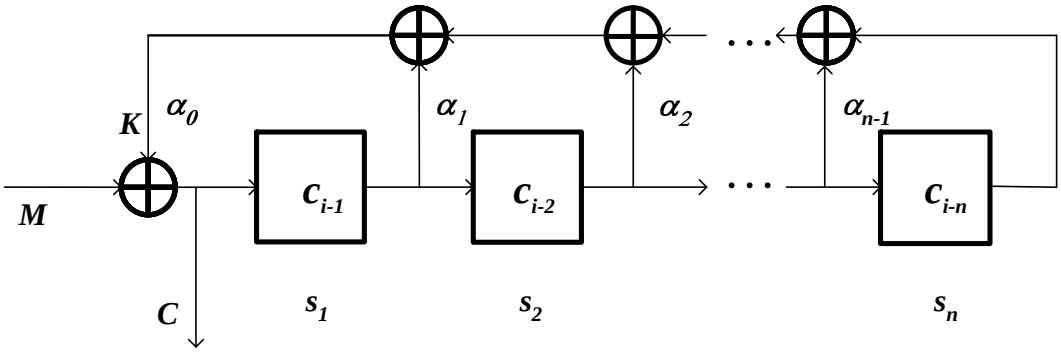


Рис.5.17. Самосинхронизирующаяся криптосистема по шифротексту (режим шифрования)

Аналитически функционирование такого генератора описывается уравнением

$$c_i = \begin{cases} m_i + \sum_{j=1}^{i} \alpha_j c_{i-j} + \sum_{j=i+1}^{n} \alpha_j s_{j-i}, & 0 \le i \le n-1; \\ m_i + \sum_{j=1}^{n} \alpha_j c_{i-j}, & i \ge n, \end{cases}$$

где m_i представляют собой последовательные биты исходной последовательности, c_i биты шифротекста, а операция сложения представляет собой операцию сложения по модулю два. Как видно из приведенного соотношения очередной бит криптографического ключа зависит не более чем от n бит шифротекста. Таким образом, при нарушении синхронизации количество искаженных бит исходного текста будет определяться величиной n. По истечении n тактов после последнего искаженного символа криптосистема будет правильно дешифрировать очередные биты исходного текста.

Для дешифрирования используется структура, выполняющая взаимообратные преобразования. В данном случае использовалось свойство обратимости операции сложения по модулю два. Структурная схема режима дешифрирования приведена на следующем рисунке.

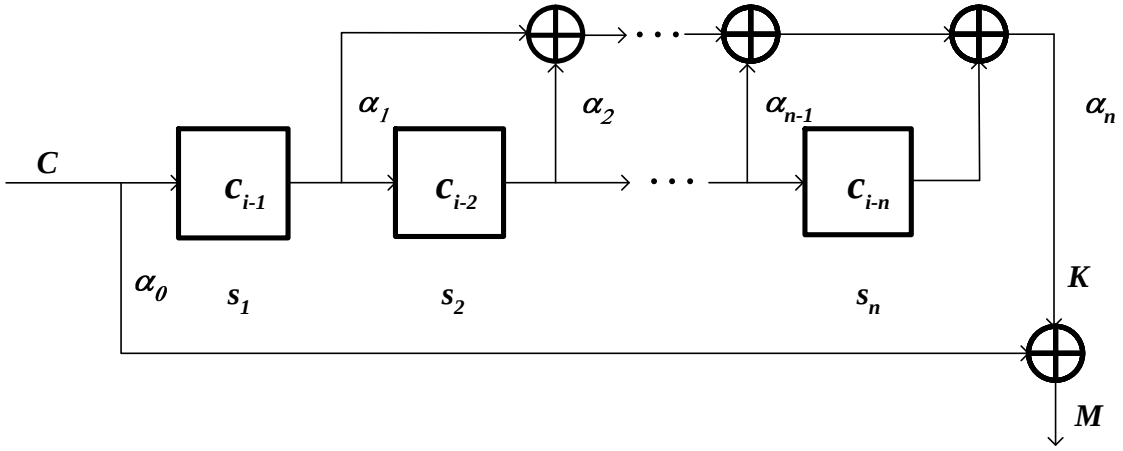


Рис.5.18. Самосинхронизирующаяся криптосистема по шифротексту (режим дешифрирования)

Аналитически данный режим описывается соотношением

$$m_i = \begin{cases} c_i + \sum_{j=1}^{i} \alpha_j c_{i-j} + \sum_{j=i+1}^{n} \alpha_j s_{j-i}, & 0 \le i \le n-1; \\ c_i + \sum_{j=1}^{n} \alpha_j c_{i-j}, & i \ge n. \end{cases}$$

Как видно из последнего соотношения очередной бит исходного текста, получаемый при дешифрировании, зависит не более чем от n последовательных бит шифотекста.

Пример 5.10. Для иллюстрации рассмотренного метода шифрования приведем пример процедуры шифрования и дешифрирования самосинхронизирующейся потоковой криптосистемы описываемой порождающим полиномом $\varphi(x)=1+x+x^4$ для исходного текста M=1110100 и начального состояния (сеансового ключа) S_0==($s_1s_2s_3s_4$)=0011. Схема шифрования подобной криптосистемы приведена на следующем рисунке.

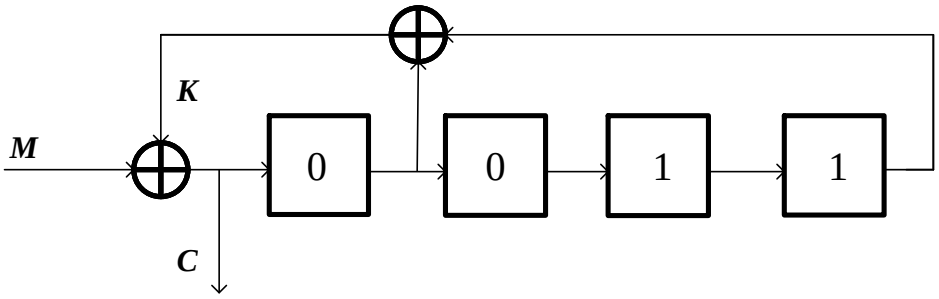


Рис.5.19. Схема шифрования потоковой криптосистемы

Последовательность состояний элементов памяти шифрующего блока потоковой криптосистемы и последовательность бит шифротекста приведены в таблице 5.6.

Таблица 5.6.

Диаграмма состояний шифратора приведенного на рис.5.18

k	***M***	***K***	***C***	c_{i-1}	c_{i-2}	c_{i-3}	c_{i-4}
1	***1***	***1***	***0***	0	0	1	1
2	***1***	***1***	***0***	0	0	0	1
3	***1***	***0***	***1***	0	0	0	0
4	***0***	***1***	***1***	1	0	0	0
5	***1***	***1***	***0***	1	1	0	0
6	***0***	***0***	***0***	0	1	1	0
7	***0***	***1***	***1***	0	0	1	1
8	***1***	***0***	***1***	1	0	0	1

Для дешифрирования используется следующая структура.

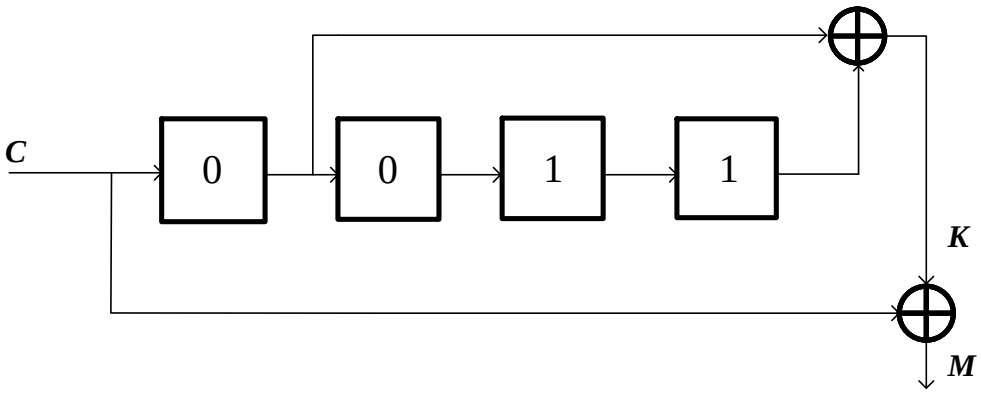


Рис.5.20. Схема дешифрирования потоковой криптосистемы

Временная диаграмма состояний блока дешифрирования приведена в таблице 5.7.

Таблица 5.7.

Диаграмма состояний шифратора приведенного на рис.5.18

k	***C***	***K***	***M***	c_{i-1}	c_{i-2}	c_{i-3}	c_{i-4}
1	***0***	***1***	***1***	0	0	1	1
2	***0***	***1***	***1***	0	0	0	1
3	***1***	***0***	***1***	0	0	0	0
4	***1***	***1***	***0***	1	0	0	0
5	***0***	***1***	***1***	1	1	0	0
6	***0***	***0***	***0***	0	1	1	0
7	***1***	***1***	***0***	0	0	1	1
8	***1***	***0***	***1***	1	0	0	1

Основным достоинством потоковых криптосистем является простота их технической реализации, а большинство проблем и недостатком связаны с генерированием криптографического ключа.

Глава 6. Криптосистемы с открытым ключом
6.1. Алгоритм распределения ключей

Открытая ***система распределения ключей*** (***public key distribution algorithm***), предложенная ***W.Diffie*** и ***M.Hellman,*** впервые была опубликована в 1976 году и предназначалась для распределения ключей по открытому каналу. Авторы предложили алгоритм, основанный на возведении в степень по модулю *q,* где *q* является простым числом. Стойкость предложенного алгоритма основывается на сложности вычисления дискретного логарифма над полем Галуа *GF(q)* с *q-1* элементами *{1,2,...,q-1}.* Рассмотрим пару взаимно инверсных функций:

$$C=\alpha^M \ (\text{mod } q);$$

$$M=\log_\alpha C \ (\text{mod } q) \text{ над полем } GF(q),$$

где *0<M, C<q,* соответственно, *q* простое целое число, а α примитивный элемент (корень) *GF(q).* Вычисление *C* из *M* осуществляется простым возведением в степень, что вычислительно не составляет труда. В тоже время вычисление дискретного логарифма над *GF(q)* процедура намного более сложная и требует для своего решения больших вычислительных затрат. Для больших значений *q* эта задача вычислительно неразрешима. На этом факте и основан рассматриваемый алгоритм, как впрочем, и большинство из известных криптосистем с открытым ключом. Сущность открытой системы распределения ключей состоит в том, что два пользователя используя открытый канал, обмениваются криптографическим ключом, который будет известен только им, двоим. Злоумышленник при попытке получения значения ключа столкнется с проблемой дискретного логарифма над полем *GF(q).*

Предварительно оба пользователя *A* и *B* обмениваются параметрами (α,*q*) конечного поля, где α является примитивным элементом поля, а модуль *q* представляет собой большое простое число. Эта информация является открытой для всех. Далее последовательно выполняются следующие действия.

1.Каждый из двух пользователь *A* и *B* генерирует независимое случайное число. Пользователь *A* число M_A, а пользователь *B* число M_B равновероятно выбранное из набора целых чисел *{1,2,...,q-1},* и каждый из них хранит свое число в секрете.

2.Оба пользователя осуществляют следующие вычисления: *A* вычисляет $C_A = \alpha^{M_A} \bmod q$, а *B* вычисляет $C_B = \alpha^{M_B} \bmod q$.

3.Затем по открытому каналу *A* посылает в адрес *B* результат своего вычисления C_A, а *B* значение C_B. Отметим невозможность восстановления величин M_A, и M_B на основании доступных для злоумышленника значений C_A, и C_B даже если параметры конечного поля (α,*q*) ему известны.

4.Пользователь *A* вычисляет K_A, получая C_B из открытого файла и выполняя вычисления.

$$K_A = C_B^{M_A} \bmod q = (\alpha^{M_B})^{M_A} \bmod q = \alpha^{M_B M_A} \bmod q$$

пользователь B вычисляет общий ключ K_B в такой же форме

$$K_B = C_A^{M_B} \bmod q = (\alpha^{M_A})^{M_B} \bmod q = \alpha^{M_A M_B} \bmod q$$

Таким образом, оба пользователя A и B обменялись секретным ключом $K_A=K_B$, что следует из приведенных выше соотношений. Отметим, что значения M_A, и M_B являются аналогами секретного ключа пользователей A и B, а C_A, и C_B аналогами их открытых ключей. Как результат приведенных действий с секретными и открытыми ключами пользователей оба они получают криптографический ключ $K=K_A=K_B$, который может быть использован ими для секретного общения используя, например DES.

Пример 5.1. Рассмотрим поле Галуа $GF(q)$ где q простое число. Примем α примитивным элементом $GF(q)$ таким, чтобы степени α генерировали все ненулевые элементы $1,2,...,q-1$ поля $GF(q)$. Например, возьмем $\alpha=2$ и $q=11$. Вычисляя степени $\alpha=2^i \pmod{11}$ получим.

i	1	2	3	4	5	6	7	8	9	10	11
$\alpha^i=2^i$	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}
$\alpha^i \bmod 11=2^i \bmod 11$	2	4	8	5	10	9	7	3	6	1	2

Оба пользователя предварительно согласовывают параметры конечного поля $(\alpha,q)=(2,11)$.

1.Первоначально, пользователь A выбирает случайное число $M_A=4$ из набора целых чисел $2^i \pmod{11}=\{1,2,3,...,10\}$ и хранит его в секрете, а B генерирует свое секретное $M_B=9$.

2.Затем оба пользователя выполняют вычисления. A вычисляет

$$C_A = \alpha^{M_A} \bmod q = 2^4 \bmod 11 = 5,$$

а B

$$C_B = \alpha^{M_B} \bmod q = 2^9 \bmod 11 = 6.$$

3.В заключение оба пользователя A и B посылают свои результаты вычислений $C_A=5$ и $C_B=6$ друг другу, для получения общего секретного ключа.

4.Соответственно A вычисляет

$$K_A = C_B^{M_A} \bmod q = 6^4 \bmod 11 = 9,$$

а пользователь B получает общий ключ K_B таким же образом

$$K_B = C_A^{M_B} \bmod q = 5^9 \bmod 11 = 9.$$

Таким образом, каждый пользователь завершает вычисления общего ключа $K=K_A=K_B$, равного 9.

Пример 5.2. Рассмотрим проблему обмена ключом в $GF(2^m)$ для $m=3$. В качестве примитивного полинома $p(x)$ степени $m=3$ возьмем полином $p(x)=1+x+x^3$. Если α является корнем $p(x)$ то ($p(\alpha)=1+\alpha+\alpha^3=0$). Множество элементов поля $GF(2^3)$ сгенерированное в соответствии с порождающим полиномом $p(x)=1+x+x^3$ в виде таблицы показано ниже. Отметим, что

элементы конечного поля Галуа *GF* (2^m) представлены в различной форме. Как трех битные двоичные векторы, в виде полиномиального представления и как экспоненты.

Таблица 5.1.

Элементы поля Галуа

Экспонента	Полином	Вектор
α^0	1	1 0 0
α^1	α	0 1 0
α^2	α^2	0 0 1
α^3	$1+\alpha$	1 1 0
α^4	$\alpha+\alpha^2$	0 1 1
α^5	$1+\alpha+\alpha^2$	1 1 1
α^6	$1+\alpha^2$	1 0 1
α^7	1	1 0 0

Последовательные степени α^0, α^1 и α^2 не требуют, каких либо пояснений об их получении, в то время как α^3 получается как результат вычислений по модулю порождающего полинома, либо с использованием равенства $p(\alpha)=1+\alpha+\alpha^3=0$. Из которого следует, что $\alpha^3=1+\alpha$.

1.Допустим, пользователи *A* и *B* выбрали $M_A=2$ и $M_B=5$, соответственно. Оба значения $M_A=2$ и $M_B=5$ держатся в секрете.

2.Далее оба пользователя выполняют вычисления своих открытых значений, соответственно

$$C_{\text{A}} = \alpha^{M_A} \bmod (1+x+x^3) = \alpha^2 = 001,$$

и

$$C_{\text{B}} = \alpha^{M_B} \bmod (1+x+x^3) = \alpha^5 = 111.$$

3.Оба пользователя обмениваются открытыми значениями C_A = 001 и C_B=111.

4.В результате вычислений

$$K_A=(a^5)^2 \bmod p(x) = a^{10} \bmod p(x) = a^3=110$$

пользователь *A* получает ключ $K=K_A=110$, а пользователь *B* ключ $K=K_B=110$ как результат вычислений

$$K_B=(a^2)^5 \bmod p(x) = a^{10} \bmod p(x) = a^3=110.$$

Таким образом, оба пользователя получили один и тот же криптографический ключ.

6.2. Криптосистемы типа рюкзак

Криптосистемы с открытым ключом неразрывно связаны с понятием ***односторонней функции***. Для таких функций их значение легко вычисляется по заданному аргументу *x*. В тоже время определение *x* по значению функции *f(x)* является трудно вычислимым.

Очень сложно теоретически и практически показать, что ***трудновычислимость*** функции $x=f^{-1}(x)$ (задачи, алгоритма) используемой нами для вычисления является неслучайной и не есть степень нашего невежества в понимании конкретной проблемы. Трудновычислимость это понятие, взятое из теории сложности, которое отражает достижения фундаментальных исследований и в какой то мере свидетельствует о достижениях науки в целом. Теория сложности дает оценку сложности решения той или иной задачи с учетом имеющихся на текущий момент знаний (методов решения этой задачи). Трудновычислимость свидетельствует о том, что на текущий момент не существует известных методов решения задачи за полиномиальное время. Но это не значит, что такие методы не существуют и не будут открыты в будущем. Поэтому в криптографии используется достаточно осторожное определение односторонней функции.

Таким образом, ***криптографической односторонней функцией*** называется функция, отвечающая двум условиям:

1.Значение *f(x)* легко вычисляется на основании значения *x*.

2.Определение *x* из *f(x),* вероятно, будет трудновычислимым.

Одним из первых примеров односторонних функций является задача о рюкзаке (***knapsack problem***), которая может быть сформулирована следующим образом.

Пусть задан набор $K=\{k_1,k_2,k_3,...,k_n\}$, из *n* различных положительных чисел, а также еще одно положительное число *C*. Задачей является нахождение таких k_i, если это возможно, сумма которых равна *C*.

В простейшем случае *C* указывает размер рюкзака, а каждое из чисел k_i указывает размер предмета, который может быть упакован в рюкзак. Задачей является нахождение такого набора предметов, чтобы рюкзак был полностью заполнен. В качестве иллюстрации рассмотрим число *C*=1524 и набор из 10 чисел

$$K = (123, 763, 37, 1451, 830, 333, 621, 745, 971, 201).$$

Заметим, что

$$1524=123+37+830+333+201.$$

Таким образом, мы знаем решение. В принципе решение всегда может быть найдено полным перебором подмножеств *K* и проверкой, какая из их сумм равна *C*. В случае нашего примера это означает перебор $2^{10}=1024$ подмножеств и это вполне осуществимо. В реальных условиях набор $K=\{k_1,k_2,k_3,...,k_n\}$, будет состоять из $n\sim200\div300$ различных чисел, для которых полный перебор не является реальным методом решения. Суть здесь в том, что неизвестны алгоритмы, имеющие существенно меньшую сложность по

сравнению с полным перебором. Поиск среди 2^{300} подмножеств не поддается обработке за реальное время. На сегодняшний день задача о рюкзаке известна как *NP* полная, имеющая экспоненциальную сложность.

Задача о рюкзаке является ярким примером односторонней функции *f(x),* которую определим следующим образом.

Набор $K=\{k_1,k_2,k_3,...,k_n\}$, из n положительных чисел k_i определяет функцию $f(x)$ как двоичное число $0\le f(x)\le 2^n-1$ полученное из номеров индексов i представленных векторами из $n-1$ нулевого символа и одного единичного, таким образом, что позиция единичного символа соответствует позиции числа k_i в множестве K. Число k_1 имеет индекс 000…01, k_2 индекс 000…10,… и k_n имеет индекс 100…00. Тогда $f(1)= f(000...01)=k_1$, $f(2)= f(000...10)=k_2$,..., $f(n)= f(100...00)= k_n$.

Наш предыдущий результат можно записать как $f(565)=f(1000110101)= =1524$.

Используя определение односторонней функции, сформулируем основные компоненты криптографической системы.

Криптографическим ключом будет являться $K=\{k_1, k_2,...,k_n\}$, где k_i целые для $i=1,2,...,n$, а исходным текстом n-битный код $M=\{x_1,x_2,...,x_n\}$, где $x_i=\{0,1\}$. Затем криптосистема типа рюкзак шифрует n-битный открытый текст по следующей формуле

$$C=K\times M= k_1x_1+k_2x_2+...+k_nx_n.$$

Вычисление C – это, как видно из приведенного выражения, достаточно простая процедура, но восстановление M из C и K как уже отмечалось, является трудноразрешимой задачей. Таким образом, в подобной интерпретации имеем криптографическую систему, когда даже при наличии криптографического ключа проблема дешифрирования является в принципе неразрешимой задачей. Для того, что бы алгоритм рюкзака можно было использовать для целей криптографии, однонаправленная функция, описывающая его, должна иметь ***лазейку*** (***trap door***)

Если криптографический ключ K выбран так, что каждый его элемент k_i превосходит сумму предшествующих элементов, то решение задачи о рюкзаке становится очень простым. То есть если компоненты ключа удовлетворяют условию $k_i>k_1+k_2+k_3+...+k_{i-1}$, то проблема дешифрирования реализуется достаточно просто.

При выполнении процедуры шифрования последовательно будут формироваться следующие значения.

$c_1=k_1x_1$;
$c_2=k_1x_1+k_2x_2$;
...
$c_{n-1}=k_1x_1+k_2x_2+...+k_{n-2}x_{n-2}+k_{n-1}x_{n-1}$;
$C=c_n=k_1x_1+k_2x_2+...+k_{n-2}x_{n-2}+k_{n-1}x_{n-1}+k_nx_n$.

где c_n представляет собой закодированный текст C. Теперь, открытый текст M может быть восстановлен из c_n для $i=1,2,...,n$ и ключа K следуя следующей процедуре.

Если $c_n<k_n$, то значение $x_n=0$, это следует из того факта, что в противном случае если $x_n=1$ то $c_n \geq k_n$. Это означает, что если $c_n<k_n$, то коэффициент k_n не был вложен в рюкзак. Тогда соответственно $c_{n-1}=c_n$. Если $c_n \geq k_n$, то набор $x_n=1$ и $c_{n-1}=c_n-k_n$.

Используя ранее полученное значение c_{n-1} можно таким же образом найти x_{n-1} и c_{n-2}. Процедура декодирования продолжается до тех пор, пока $M=\{x_1,x_2,...,x_n\}$, не будет полностью восстановлен.

Отметим, что в подобной интерпретации криптосистема типа рюкзак может быть использована только как симметричная криптосистема, когда только отправитель и получатель знают секретный ключ $K=\{k_1, k_2,...,k_n\}$.

Пример 5.3. Допустим открытый текст, принимает следующий вид $M=\{11001\}$, а криптографический ключ $K=\{151, 187, 426, 1091, 2412\}$. Здесь для компонент ключа выполняется условие $k_i>k_1+k_2+k_3+...+k_{i-1}$.

Шифротекст C будет вычисляться согласно следующей зависимости. $C=K\times M=1\times 151+1\times 187+0\times 426+0\times 1091+1\times 2412=2750$.

Зашифрованный текст $C=c_5=2750$ высылается в адрес получателя информации. На первом этапе дешифрирования получатель восстанавливает x_5 из c_5 и k_5. Получим что $x_5=1$, так как $c_5=2750>k_5=2412$. Полная процедура дешифрирования x_i для $i=5,4,...,1$ имеет вид.

$$c_5=2750>k_5=2412,\ x_5=1;$$
$$c_4=c_5-k_5=338<k_4=1091,\ x_4=0;$$
$$c_3=338<k_3=426,\ x_3=0;$$
$$c_2=338>k_2=187,\ x_2=1;$$
$$c_1=c_2-k_2=151=k_1,\ x_1=1.$$

Таким образом, как и ожидалось, восстановленный открытый текст $M=\{11001\}$.

Однако, как уже отмечалось, ключ K не может использоваться, как открытый ключ для шифрования, потому что любой пользователь, имеющий информацию о ключе, может легко восстановить M из C.

Для использования алгоритма рюкзака для целей создания криптосистем с открытым ключом, в рассмотренный выше алгоритм необходимо внести ряд модификаций. Процедура построения криптосистемы с открытым ключом будет состоять из следующих этапов.

1.Первоначально генерируется секретный ключ $K_p^*=\{k_1^*,k_2^*,...,k_n^*\}$для которого выполняется условию $k^*_i>k^*_1+k^*_2+k^*_3+...+k^*_{i-1}$. Этот ключ должен быть известен только одному пользователю.

2.Выбирается значение модуля m, которое должно быть больше, чем $k^*_1+k^*_2+...+k^*_n$, что необходимо для однозначного декодирования шифротекста. Затем определяется величина w, где $w<m$, $(w,m)=1$и вычисляется еще один секретный параметр $v=w^{-1}$, в соответствии с $wv=1 \bmod$

m или что тоже самое $ww^{-1}=1 \ (mod \ m)$. Параметры m, w и v являются секретными.

3.Генерируется открытый ключ как вектор $K_p=wK_p^* \ mod \ m$, где его компоненты вычисляются как $k_i=wk_i^* \ mod \ m, \ i=1,2,...,n$.

4.Открытый ключ $K_p=\{k_1,k_2,...,k_n\}$ является доступным для всех пользователей.

Пример 5.4. Построим криптографическую систему типа рюкзак.

1.Генерируется секретный ключ $K_p^*=\{k_1^*,k_2^*,...,k_n^*\}=\{1, 3, 5, 11, 21, 44, 87, 175, 349, 701 \}$, для которого выполняется условию $k^*_i>k^*_1+k^*_2+k^*_3+...+k^*_{i-1}$.

2.Выбирается модуля $m=1590>k^*_1+k^*_2+...+k^*_n=1397$, а также величина $w=43<m=1590$, которая является взаимно простой с величиной m, то есть $(w,m)=(43, 1590)=1$. Если w и m являются взаимно простыми, то существует единственное решение линейного уравнения $wv=1 \ mod \ m$, которое в данном случае имеет вид $43v=mod \ 1590$. Решением данного уравнения является $v=w^{-1}=37$.

3.Генерируется открытый ключ как вектор K_p где его компоненты вычисляются как $k_i=wk_i^* \ mod \ m, \ i=1,2,...,n$.

k_1=43×1 mod 1590 = 43;
k_2=43×3 mod 1590 = 129;
k_3=43×5 mod 1590 = 215;
k_4=43×11 mod 1590 = 473;
k_5=43×21 mod 1590 = 903;
k_6=43×44 mod 1590 = 302;
k_7=43×87 mod 1590 = 561;
k_6=43×175 mod 1590 = 1165;
k_6=43×349 mod 1590 = 697;
k_6=43×701 mod 1590 = 1523;

4.Открытый ключ $K_p=\{k_1,k_2,...,k_n\}=\{43, 129, 215, 473, 903, 302, 561, 1165, 697, 1523\}$ является открытым для всех пользователей.

Теперь рассмотрим процедуру шифрования и дешифрирования для криптосистемы с открытым ключом типа рюкзак.

Что касается процедуры шифрования, то она полностью повторяет процедуру шифрования описанную ранее $C=K_p \times M= k_1x_1+k_2x_2+...+k_nx_n$. Таким образом, любой пользователь может сформировать шифрограмму на основании открытого ключа K_p получателя информации и исходного сообщения M.

При получении шифрограммы получатель предварительно преобразует (трансформирует) полученный шифротекст из пространства C шифротекстов, полученных на основании открытого ключа K_p, в пространство шифротекстов C^* полученных на основании секретного ключа K_p^*. Для этого выполняется операция умножения C на величину $v=w^{-1}$. В результате получим

$C^*=w^{-1}C \bmod m = w^{-1}(K_p \times M) \bmod m = w^{-1}(k_1x_1+k_2x_2+...+k_nx_n) \bmod m.$

Учитывая, что $k_i=wk_i^* \bmod m$, получим

$C^*=(w^{-1}wk_1^*x_1 \bmod m+ w^{-1}wk_2^*x_2 \bmod m+...+ w^{-1} wk_n^*x_n \bmod m) \bmod m.$

Далее, учитывая соотношения $w^{-1} w=1$, и $k^*_1+k^*_2+...+k^*_n < m$, получим

$C^*=(k_1^*x_1 \bmod m+k_2^*x_2 \bmod m+...+k_n^*x_n \bmod m) \bmod m=$

$=(k_1^*x_1 +k_2^*x_2+...+k_n^*x_n) \bmod m=$

$=(k_1^*x_1 +k_2^*x_2+...+k_n^*x_n).$

Таким образом, $C^*=(k_1^*x_1 +k_2^*x_2+...+k_n^*x_n)$, что позволяет использовать ранее рассмотренную процедуру дешифрирования путем сравнения C^* с k_n^* для получения x_n и таким образом последовательно получить все значения $M=\{x_1,x_2,...,x_n\}$.

6.3. Экспоненциальные криптосистемы

В 1978, ***Pohling*** и ***Hellman*** опубликовали схему шифрования, основанную на вычислении показательной функции в конечном поле. В то же время, ***Riverst, Shamir*** и ***Adleman*** опубликовали схожую схему, но с небольшими изменениями, которые позволили создать реальный метод шифрования с открытым ключом *RSA*.

Обе схемы шифрования (*Pohling-Hellman* и *RSA*) предназначены для шифрования блоков сообщения $M \in \{0,1,...,n-1\}$ путем вычисления экспоненты

$$C=M^e \bmod n,$$

где e и n является ключом для кодирующих преобразований.

Значение сообщения M восстанавливается такой же операцией

$$M=C^d \bmod n,$$

но используется другая экспонента d для ключа.

Кодирование и декодирование могут быть реализованы с использованием быстрого алгоритма возведения в степень $C=fastexp(M,e,n)$ и $M=fastexp(C,d,n)$, который рассматривался в главе 3.

Кодирующие и декодирующие преобразования основаны на прямом использовании теоремы Эйлера, которая является обобщении теоремы Ферма. Теорема Эйлера гласит, что для каждого M взаимно простого с n, выполняется равенство

$$M^{\psi(n)} \bmod n = 1.$$

Это свойство предполагает, что если e и d удовлетворяют соотношению $ed \bmod \psi(n)=1$, где $\psi(n)$ представляет собой функцию Эйлера, то процедура декодирования восстанавливает оригинальный открытый текст. Это доказывается следующей теоремой.

Теорема 6.1. Для целых чисел e и d удовлетворяющих соотношению ed

mod $\psi(n)=1$, где $\psi(n)$ есть функция Эйлера от n, и исходного сообщения $M\in\{0,1,...,n-1\}$ взаимно простого с n, $(M,n)=1$, выполняется равенство

$$(M^e \bmod n)^d \bmod n=M.$$

Доказательство: Согласно условию теоремы имеем $(M^e \bmod n)^d \bmod n=M^{ed} \bmod n$. Равенство $ed \bmod \psi(n)=1$ можно представить как $ed = t\psi(n)+1$, где t есть произвольное целое число.

Таким образом, подставив $ed = t\psi(n)+1$ в исходное соотношение и преобразовав, получим

$$M^{ed} \bmod n=M^{t\psi(n)+1} \bmod n=M\times M^{t\psi(n)} \bmod n=$$
$$= M\times(M^{t\psi(n)} \bmod n)\bmod n =$$
$$=M\times(M^{\psi(n)} \bmod n)^t \bmod n=$$
$$=M\times(1)^t \bmod n = M.\#$$

Из симметрии следует, что процедуры кодирования и декодирования коммутативные операции и соответственно взаимно инверсные. Таким образом,

$$(M^e \bmod n)^d \bmod n=(M^d \bmod n)^e \bmod n=M^{de} \bmod n = M.$$

Процедура создания экспоненциальной криптографической системы заключается в вычислении двух криптографических ключей e и d.

Для этого первоначально выбирается модуль n как большое целое число, чаще всего простое. Затем вычисляется функция Эйлера $\psi(n)$. На следующем шаге определяются криптографические ключи e и d удовлетворяющие условию $ed \bmod \psi(n)=1$. Вначале выбирается d взаимно простое с $\psi(n)$, а затем используется расширенный алгоритм Евклида для вычисления его инверсного значения $e=inv(d,\psi(n))$.

В силу того, что величины e и d симметричны, первоначально может быть выбрано значение e, а затем вычислено $d=inv(e,\psi(n))$.

В рассмотренной схеме шифрования *Pohling* и *Hellman*, модуль n чаще всего выбирается как большое простое число $n=p$. Тогда шифрующее и дешифрующее преобразования имеют вид.

$$C=M^e \bmod p,\ M=C^d \bmod p.$$

Все преобразования выполняются в поле Галуа $GF(p)$. Так как p является простым числом то, $\psi(n)= p-1$, и не требует никаких затрат для вычисления. Это обстоятельство является ключевым, для того чтобы определить условия использования подобной криптосистемы.

Криптографическими ключами для шифрования и дешифрирования являются значения (e,p) и (d,p). Знание одного из них позволяет достаточно просто вычислить второй. Процедура вычисления сводится к решению линейного сравнения с одним неизвестным.

Таким образом, схема шифрования *Pohlnig* и *Hellman* может быть

использована для обычных систем шифрования, когда оба криптографические ключи *e* и *d* содержатся в секрете.

Пример 5.5. Допустим *p=11*, тогда *ψ(p)=p-1=10*. Выберем *d=7* и вычислим *e=inv(7,10)*. Для определения *e* решим линейное сравнение *7e=1 mod 10*. Получим $e=7^{\psi(10)-1} \bmod 10=7^{4-1} \bmod 10=7^3= \bmod 10=3$.

Предположим, что в качестве исходного сообщения выберем *M=5*. Тогда процедура шифрования *M* будет иметь вид

$$C=M^e \bmod p=5^3 \bmod 11=4,$$

а процедура дешифрирования *C* представляется как

$$M=C^d \bmod p=4^7 \bmod 11=5.$$

Как видно из приведенного примера схема предложенная *Pohling* и *Hellman* может быть эффективно использована для реализации классических симметричных криптографических систем

6.4. Криптосистема RSA

В сравнении с экспоненциальными шифраторами надёжность криптосистемы *RSA* основывается на трудности задачи разложения составных чисел на сомножители, что позволяет ее использование как системы с открытым ключом.

Алгоритм RSA (по первым буквам фамилий его создателей ***Rivest-Shamir-Adleman***) основан на свойствах простых чисел, причем очень больших простых чисел. В *RSA*, модуль *n* представляет собой составное число как произведение двух больших простых чисел *p* и *q*, то есть *n=pq*.

Именно модуль, который, является открытым, а его разложение на *p* и *q*, закрытым и является основой алгоритма *RSA*, и главным достижением авторов этого алгоритма позволившим успешно применять его на практике.

Сомножители *p* и *q* модуля *n* хранятся в секрете так как, если злоумышленник разложит, открыто опубликованное число *n* на *p* и *q*, то он сможет найти секретный ключ тем же способом, что и разработчик этой криптосистемы. Таким образом, если задача разложения на множители может быть решена быстро, например каким-то пока неизвестным нам алгоритмом, то взломать криптосистему *RSA* будет легко. Обратное утверждение, показывающее, что если задача разложения на множители сложна, то взломать систему *RSA* трудно, не доказано. Однако за время существования этой криптографической системы никакого иного способа её взлома обнаружено не было.

Определим параметр *n* как результат перемножения двух больших простых чисел *p* и *q*. Выберем большое случайное число *d*, и будем использовать его в качестве одного из криптографических ключей. Значение *d* должно быть взаимно простым с результатом произведения $(p-1)\times(q-1)$. Тогда оказывается возможным нахождение второго числа *e*, для которого верно соотношение

$$(e\times d) \bmod ((p-1)\times(q-1)) = 1$$

Тогда открытым ключом является пара чисел *(e,n)*, а закрытым *(d,n)*.

При шифровании исходный текст рассматривается как последовательность целых блоков целых чисел $M=m_1,m_2,\ldots, m_l$, и над каждым блоком этой последовательности выполняется операция

$$c_i = m_i^e \bmod n.$$

В результате получается последовательность блоков чисел шифротекста $C=c_1,c_2,\ldots c_l$. Декодирование информации происходит по формуле

$$m_i = c_i^d \bmod n.$$

Таким образом, процедура дешифрирование предполагает знание секретного ключа (d,n).

Математическая основа алгоритма *RSA* базируется на теореме 6.1.

Пример 5.6. Предположим, что $p=3$ и $q=7$. Тогда $n=p\times q=21$. В качестве одного из ключей выбираем $d=5$. Далее, решая уравнение $5\times e \bmod 12=1$, где $\psi(n)=(p-1)\times(q-1)=(3-1)\times(7-1)=12$, определяем $e=17$. Таким образом, открытый ключ равен $(17, 21)$, а секретный – $(5, 21)$.

В качестве исходного текста используем последовательность блоков чисел $M=m_1,m_2,m_3,m_4,m_5 = 1,2,3,4,5$, состоящих из одной десятичной цифры. Тогда последовательная процедура шифрования будет иметь вид

$c_1 = 1^{17} \bmod 21 = 01;$
$c_2 = 2^{17} \bmod 21 = 11;$
$c_3 = 3^{17} \bmod 21 = 12;$
$c_4 = 4^{17} \bmod 21 = 16;$
$c_5 = 5^{17} \bmod 21 = 17.$

В результате получим шифрограмму $C=c_1,c_2,c_3,c_4,c_5=01,11,12,16,17$, которая соответствует исходному тексту M. Процедура дешифрирования показана ниже

$m_1 = 01^5 \bmod 21 = 1;$
$m_2 = 11^5 \bmod 21 = 2;$
$m_3 = 12^5 \bmod 21 = 3;$
$m_4 = 16^5 \bmod 21 = 4;$
$m_5 = 175 \bmod 21 = 5;$

Пример 5.7. Предположим $p=53$, а $q=61$, откуда $n=pq=53\times 61=3233$ и $\psi(n)=(53-1)(61-1)=3120$. Полагая $d=791$, и решая линейное уравнение $791e=1 \bmod 3120$, получим $e=71$.

Для кодирования сообщения $M=RENAISSANCE$, предварительно буквы исходного текста должны быть закодированы в цифровую форму. Для этого можно использовать различные системы кодирования. Например, каждый символ исходного текста может быть закодирован двумя десятичными цифрами как $A=00$, $B=01$,..., $Z=25$, и пробел как 26. Тогда исходный текст $M=RE\ \ NA\ IS\ \ SA\ \ NC\ E$ будет представляться как последовательность числовых блоков $M=1704\ 1300\ 0818\ 1800\ 1302\ 0426$.

Первый блок кодируется, как $1704^{71} \bmod 3233=3106$, а все

зашифрованное сообщение примет вид *C=3106 0100 0931 2691 1984 2927.*

Более подробно рассмотрим процедуру проектирования криптографической системы типа *RSA.*

1.Первоначально выбираются два простых числа, *p* и *q*, где $p \neq q$ и представляют собой сравнимые по величине большие простые числа.

2.Вычисляется произведение $n=pq$, которое является открытым, где *p* и *q* случайно выбранные в предыдущем пункте простые числа.

3.Вычисляется функция Эйлера $\psi(n)=(p-1)(q-1)$, которая является секретным значением.

4.Случайным образом выбирается ключ *e* или *d*, который должен быть взаимно простым с $\psi(n)$.

5.Вычисляется мультипликативное инверсное значение по отношению к величине, определенной в п.4 (или *d* или *e*), в соответствии с уравнением $ed \bmod \psi(n)=1$.

6.5. Эллиптические кривые

Понятие эллиптических кривых является относительно новым понятием в криптографии, однако весьма активно обсуждаемым как один из кандидатов на последующее широкое применение для практических нужд защиты информации.

Эллиптические кривые (***Elliptic Curve***) в общем случае описываются кубическими уравнениями вида

$$y^2+axy+by=x^3+cx^2+dx+e.$$

Где *a, b, c, d* и *e* представляют собой действительные числа, которые удовлетворяют достаточно простым требованиям. Определение эллиптических кривых включает понятие элемента *O* называемого ***бесконечным элементом*** (***infinite element***) или нулевым элементом (***zero element***). На следующих рисунках 6.1 и 6.2 приведены два примера эллиптических кривых.

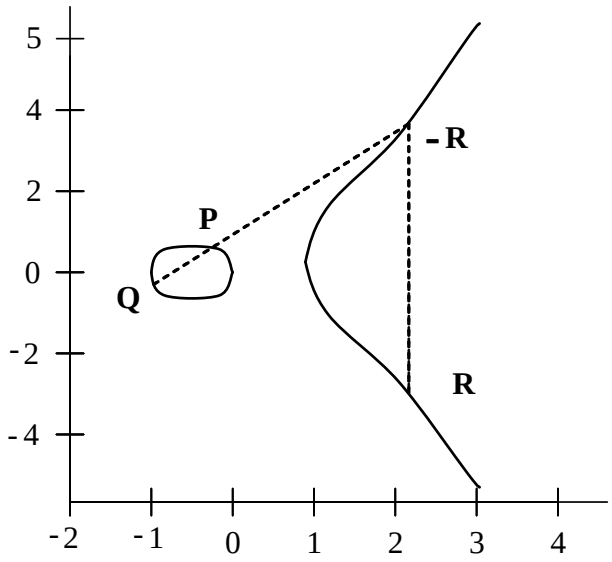


Рис.6.1. Эллиптическая кривая, описываемая уравнением $y^2=x^3-x$

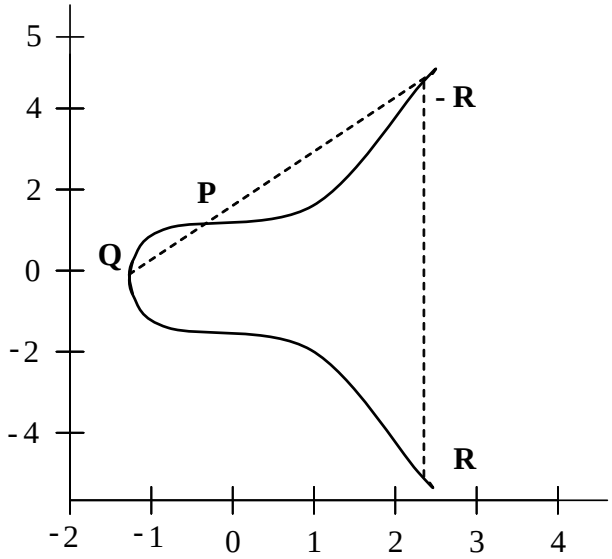


Рис.6.2. Эллиптическая кривая, описываемая уравнением $y^2=x^3+x+1$

Эллиптические кривые над множеством действительных чисел определяются парами чисел (x,y) задающих координаты ***точек*** эллиптической кривой. На практике чаще всего применяются эллиптические кривые, описываемые уравнением

$$y^2=x^3+ax+b,$$

где x, y, a и b являются действительными числами.

Если для эллиптической кривой описываемой уравнением $y^2=x^3+ax+b$ выполняется условие $4a^3 + 27b^2 \neq 0$, тогда эллиптическая кривая $y^2= x^3+ax+ b$ может быть использована для задания множества действительных точек описываемых данной кривой. Это множество составляет группу точек, включающую также и специальную точку O над которыми могут быть

определены различные операции. Главным элементом эллиптической кривой является точка (***point***) $P=(x,y)$, определяемая ее координатами x и y, принадлежащими этой кривой. По определению различают точку $P=(x,y)$ эллиптической кривой и инверсную ей точку $-P=(x,-y)$.

Основополагающим свойством эллиптических кривых является следующее свойство.

Секущая прямая линия всегда пересекает эллиптическую кривую только в трех точках. Исключением является случай вертикальной секущей прямой, для которой третьей точкой является бесконечный элемент O.

Приведенное свойство позволяет определить операцию сложения двух точек эллиптической кривой. Предположим, что P и Q являются двумя различными точками эллиптической кривой и P не равняется $-Q$. Для сложения двух точек P и Q, проводится прямая линия через эти две точки. Согласно свойству эллиптических кривых приведенному выше эта линия пересечет эллиптическую кривую только в еще одной точке, называемой $-R$. Точке $-R$ всегда соответствует симметричная ей точка R (см. рис.6.1 и рис.6.2). Таким образом, операция сложения над группой задаваемой точками эллиптической кривой определяется как

$$P + Q = R.$$

Приведенное определение операции сложения является геометрической интерпретацией операции сложения точек эллиптической кривой.

Для случая, когда точка Q равняется $-P$ прямая линия является вертикальной линией, которая пересекается только в этих двух точках, поэтому в основном для этих целей и был определен бесконечный элемент (нулевой элемент) O. Тогда согласно определению, $P + (-P) = O$. Как результат данного равенства, $P + O = P$. Элемент O называется аддитивным элементом (***additive identity***) либо нулевым аддитивным элементом группы определяемой эллиптической кривой. Все эллиптические кривые имеют аддитивный элемент. Более детально основные свойства операции сложения приведены ниже.

1.Элемент O является нулевым аддитивным элементом, для которого выполняется равенство $O=-O$. Тогда соответственно для любой точки P эллиптической кривой получим $P+O=P$.

2.Прямая линия, проведенная через точки P и $-P$ является вертикальной линией. Тогда согласно определению $P+(-P)=O$.

3.Для $P \neq Q$ и $P \neq -Q$ результатом сложения двух точек будет являться третья точка, принадлежащая эллиптической кривой, то есть $P+Q=R$.

4.В случае, когда $P = Q$, то есть когда точка P складывается сама с собой и $P \neq (x,0)$, в точке P проводится касательная линия к эллиптической кривой. Эта линия пересечет эллиптическую кривую только в одной точке $-R$. Тогда результатом удвоения точки P будет точка R симметричная точке $-R$.

Таким образом

$$P+P=2P=R.$$

5.Для случая, когда $P = (x,0)$, касательная к эллиптической кривой в этой точке будет представлять собой вертикальную линию. Тогда, согласно определению, для такой точки $2P = O$. Для случая вычисления $3P$ будем иметь $2P + P$. Выполнив подстановку вместо $2P$ нулевого элемента O, окончательно получим $P + O = P$. Таким образом, $3P = P$. Для $P = (x,0)$ будут выполняться равенства $3P = P$, $4P = O$, $5P = P$, $6P = O$, $7P = P$, и так далее.

6.Операция умножения точки P на положительное целое число k определяется как сумма k точек P, так что $kP=P+P+P+...+P$.

Несмотря на то, что геометрическая интерпретация операции сложения точек эллиптических кривых является достаточно наглядной иллюстрацией арифметики эллиптических кривых, он не может быть использован для практической реализации вычислений. Для этих целей используется алгебраическое определение операции сложения двух точек $P=(x_P,y_P)$ и $Q=(x_Q,y_Q)$, которое однозначно вытекает из геометрической интерпретации этой операции.

1.Если P и Q являются различными точками эллиптической кривой и P не равняется $-Q$, тогда результатом операции сложения $P+Q=R$ является точка $R=(x_R,y_R)$, координаты которой вычисляются по следующим соотношениям

$$s=(y_P-y_Q)/(x_P-x_Q);$$
$$x_R=s^2-x_P-x_Q;$$
$$y_R=-y_P+s(x_P-x_R).$$

2. Удвоение точки P будет осуществляться подобным образом

$$s=(3x_P^2+a)/(2y_P);$$
$$x_R=s^2-2x_P;$$
$$y_R=-y_P+s(x_P-x_R).$$

Величина a используемая в последнем выражении является одним из параметров эллиптической кривой описываемой уравнением $y^2=x^3+ax+b$.

Очевидно, что даже аналитические соотношения, определяющие операцию сложения точек эллиптической кривой, не позволяют достичь высокой степени точности вычислений. Вычисления являются, как правило, медленными, а результат не точен, что неприемлемо для криптографии, где вычисления должны давать абсолютно точный результат, а процедура вычисления результата выполняться за реальное время.

Для практического применения в криптографии используются конечные поля (поля Галуа) $GF(M)$ и $GF(2^m)$. Так, например конечное поле $GF(M)$ оперирует целыми числами от 0 до $M-1$, где M, как правило, представляет собой большое простое число, а все вычисления выполняются по модулю M.

Если в уравнении $y^2=x^3+ax+b$ параметры a и b принадлежат конечному полю $GF(M)$ и оно не содержит повторяющихся сомножителей ($4a^3+27b^2\neq0$

mod M), то эллиптическая кривая, описываемая таким уравнением, определяет эллиптическую группу $E_M(a,b)$ над полем Галуа *GF(M)*. Эллиптическая группа $E_M(a,b)$ также может быть описана как $y^2=x^3+ax+b \ mod \ M$. Она включает точки, принадлежащие эллиптической кривой $y^2=x^3+ax+b$ координаты которых являются элементами конечного поля *GF(M)*.

Пример 5.8. Рассмотрим эллиптическую группу $E_M(a,b)=E_{23}(1,0)$, описываемую эллиптической кривой $y^2=x^3+x$ в конечном поле *GF(23)*.

Данной группе принадлежит точка (*9,5*) так как ее координаты удовлетворяют уравнению $y^2=x^3+x \ mod \ 23$. Действительно, подставив в уравнение $y^2 \ mod \ M = x^3 + x \ mod \ M$ значения $x=9$ и $y=5$, последовательно получим

$5^2 \ mod \ 23 = 9^3 + 9 \ mod \ 23,$

$5^2 \ mod \ 23 = 729 + 9 \ mod \ 23,$

$25 \ mod \ 23 = 738 \ mod \ 23,$

$2 = 2.$

Таким образом, точка (*9,5*) принадлежит эллиптической группе $E_{23}(1,0)$. Всего этой группе принадлежит 23 точки, и сама группа представляется как {*O* (*0,0*) (*1,5*) (*1,18*) (*9,5*) (*9,18*) (*11,10*) (*11,13*) (*13,5*) (*13,18*) (*15,3*) (*15,20*) (*16,8*) (*16,15*) (*17,10*) (*17,13*) (*18,10*) (*18,13*) (*19,1*) (*19,22*) (*20,4*) (*20,19*) (*21,6*) (*21,17*)}.

Графически эллиптическая группа $E_{23}(1,0)$ показана на рис. 6.3.

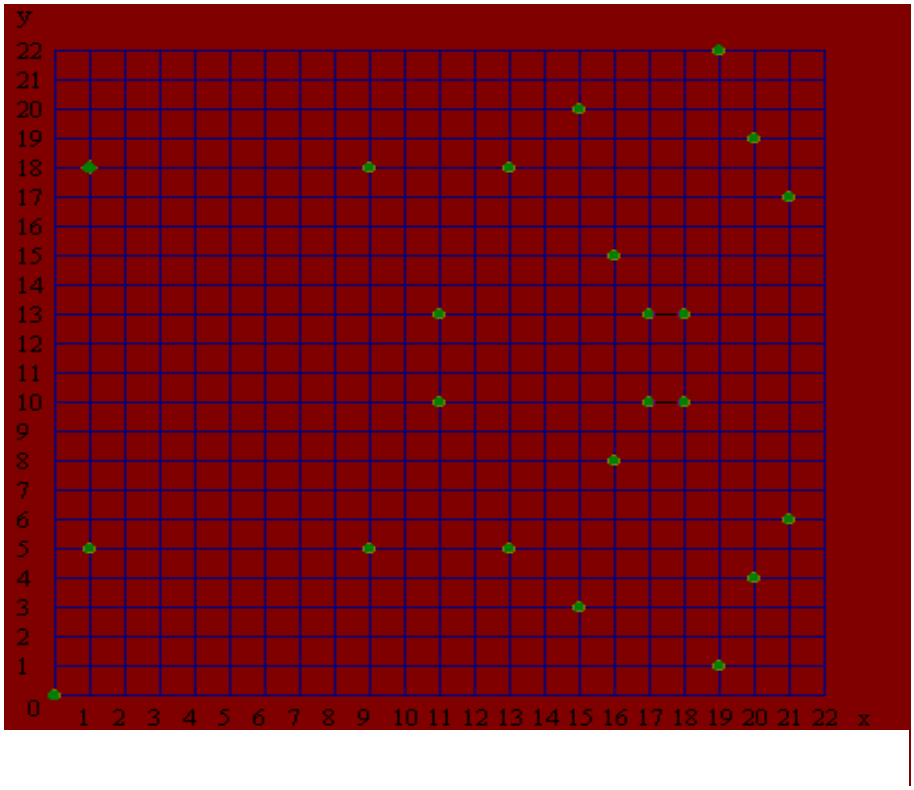


Рис.6.3. Эллиптическая группа $E_{23}(1,0)$

Приведенный пример иллюстрирует структуру эллиптической группы в целом. Можно отметить две характерные черты эллиптической группы. Во-

первых, их природа носит достаточно случайный характер, а во-вторых, не все целочисленные значения конечного поля *GF(23),* равно как и не все возможные точки с координатами, принадлежащими *GF(23)* входят в эллиптическую группу.

Общее количество точек эллиптической группы определяется в соответствии с утверждением Хоссего (Hassego).

$$M+1-2M^{1/2} \le \#E_M(a,b) \le M+1+2M^{1/2}.$$

Анализ последнего соотношения показывает

Пример 5.9. В качестве второго примера рассмотрим эллиптическую группу $E_M(a,b)=E_5(0,1)$, описываемую эллиптической кривой $y^2=x^3+1$ в конечном поле *GF(5).* Данная группа будет состоять из следующих точек {*O*, (*0,1*), (*0,4*), (*2,2*), (*2,3*), (*4,0*)}.

Для эллиптических групп также справедливы все свойства, которые касались эллиптических кривых для действительных чисел. Единственным отличием является выполнение всех операций по модулю *M*. Так отрицательная точка *-P* будет определяться как $-P = (x_P, -y_P \bmod M)$. Например, для эллиптической группы $E_5(0,1)$ и ее точки $-(2,2)$ будем иметь $-(2,2)=(2,-2 \bmod 5)=(2,3)$.

Для эллиптических групп соотношения для определения координат точки *R* являющейся суммой *P+Q* двух исходных точек определяется следующим образом.

1.Если *P* и *Q* являются различными точками эллиптической группы $E_M(a,b)$ и *P* не равняется *–Q*, тогда результатом операции сложения $P+Q=R$ является точка $R=(x_R,y_R)$, координаты которой вычисляются по следующим соотношениям

$$s=(y_P-y_Q)/(x_P-x_Q) \bmod M;$$
$$x_R = s^2 - x_P - x_Q \bmod M;$$
$$y_R = -y_P + s(x_P - x_R) \bmod M.$$

2.Удвоение точки *P* будет осуществляться подобным образом

$$s=(3x_P^2+a)/(2y_P) \bmod M;$$
$$x_R = s^2 - 2x_P \bmod M;$$
$$y_R = -y_P + s(x_P - x_R) \bmod M.$$

Величина *a* является одним из параметров эллиптической группы $E_M(a,b)$ и принадлежит конечному полю *GF(M).* Все вычисления выполняются в конечном поле по модулю *M*.

Пример 5.10. Рассмотрим операцию сложения двух точек (*0,1*) и (*2,2*) принадлежащих эллиптической группе $E_5(0,1)$. В силу того, что $(0,1)\neq(2,2)$ и $(2,2)\neq-(0,1)$, используем первый случай для операции сложения. Тогда $s=(y_P-y_Q)/(x_P-x_Q) \bmod M=(1-2)/(0-2) \bmod 5$. Откуда получим линейное уравнение

$2s=1 \bmod 5$, решением которого является $s=3$. Далее значения координат результирующей точки $R=(x_R,y_R)$ вычисляются как $x_R=s^2-x_P-x_Q \bmod M=(3^2-0-2) \bmod 5=2$; $y_R=-y_P+s(x_P-x_R) \bmod M=-1+3(0-2) \bmod 5=-7 \bmod 5=3$. Окончательно получим точку $R=(x_R,y_R)=(2,3)$

Пример 5.11. Рассмотрим удвоение точки $(2,2)$ принадлежащей этой же эллиптической группе $E_5(0,1)$ тогда в силу того, что $(2,2)=(2,2)$ используем второй случай для операции сложения. Тогда $s=(3x_P^2+a)/(2y_P) \bmod M=(3\times 2^2+0)/(2\times 2) \bmod 5=3$. Откуда значения координат точки $R=(x_R,y_R)$ вычисляются как $x_R=s^2-2x_P \bmod M=(3^2-2\times 2) \bmod 5=0$; $y_R=-y_P+s(x_P-x_R) \bmod M=-2+3(2-0) \bmod 5=4$. В результате получили точку $R=(x_R,y_R)=(0,4)$.

Пример 5.12. Рассмотрим случай, когда $P=-P$. Возьмем точки $(2,2)$ и $(2,3)$ принадлежащей этой же эллиптической группе $E_5(0,1)$ тогда в силу того, что $(2,2)=-(2,3)$ результатом будет точка O. Формально это значение получается при вычислении значения $s=(y_P-y_Q)/(x_P-x_Q) \bmod M = (2-3)/(2-2) \bmod 5=\infty$. Тогда $(2,2)+(2,3)=O$.

Таблица 5.2 для операции сложения точек эллиптической группы $E_5(0,1)$ содержит результаты сложения всевозможных пар точек данной группы.

Таблица 5.2.

Операция сложения точек $E_5(0,1)$

+	O	$(0,1)$	$(0,4)$	$(2,2)$	$(2,3)$	$(4,0)$
O	O	$(0,1)$	$(0,4)$	$(2,2)$	$(2,3)$	$(4,0)$
$(0,1)$	$(0,1)$	$(0,4)$	O	$(2,3)$	$(4,0)$	$(2,2)$
$(0,4)$	$(0,4)$	O	$(0,1)$	$(4,0)$	$(2,2)$	$(2,3)$
$(2,2)$	$(2,2)$	$(2,3)$	$(4,0)$	$(0,4)$	O	$(0,1)$
$(2,3)$	$(2,3)$	$(4,0)$	$(2,2)$	O	$(0,1)$	$(0,4)$
$(4,0)$	$(4,0)$	$(2,2)$	$(2,3)$	$(0,1)$	$(0,4)$	O

Более сложные вычисления необходимо выполнить для эллиптических групп большей размерности.

Пример 5.13. Предположим, имеем эллиптическую группу $E_{23}(9,17)$ описываемую эллиптической кривой $y^2=x^3+9x+17 \bmod 23$. Нетрудно убедится, что точка $P=(16,5)$ принадлежит данной группе. Действительно $y^2 \bmod 23=5^2 \bmod 23=2$ и $16^3+9\times 16+17 \bmod 23=2$.

Вначале вычислим $2P=P+P$. Для этого последовательно определим $s=(3x_P^2+a)/(2y_P) \bmod M=(3\times 16^2+9)/(2\times 5) \bmod 23$. Как результат вычислений получим линейное сравнение $10s=18 \bmod 23$, решением которого в свою очередь будет $s=18\times 10^{\varphi(23)-1} \bmod 23 = 18\times 10^{21} \bmod 23 =11$.

Тогда $x_R=s^2-2x_P \bmod M=(11^2-2\times 16) \bmod 23=20$ и $y_R=-y_P+s(x_P-x_R) \bmod M=-5+11(16-20) \bmod 23=-49 \bmod 23=-3 \bmod 23=20$. Как результат получим новую точку $Q=2P=(20,20)$.

Для вычисления $R=P+Q=3P$ последовательно выполним вычисления $s=(y_P-y_Q)/(x_Q-x_P) \bmod M=(5-20)/(20-16) \bmod 23$. Затем $4s=-15 \bmod 23 =8 \bmod 23$ и $s=8\times4^{\varphi(23)-1} \bmod 23 = 8\times4^{21} \bmod 23 = 2$.

Тогда, $x_R =s^2- x_P -x_Q \bmod M=(22-16-20) \bmod 23=-9 \bmod 23=14$ и $y_R =-y_P+s(x_P -x_R) \bmod M=-5+2(14-16) \bmod 23= -9 \bmod 23=14$. В результате получим точку $R=3P=(14,14)$.

Основной операцией выполняемой при реализации криптографических алгоритмов является операция вычисления скалярного произведения kP, которое реализуется путем последовательно выполнения операции сложения $P+P+P+\ldots$.

При последовательном выполнении подобного сложения на каждом шаге будет получаться результирующая точка, которая также должна принадлежать исходной эллиптической группе. Если операция последовательного добавления точки P к уже полученной сумме $P+P+P+\ldots$ достаточно продолжительна то в силу того, что эллиптическая группа содержит конечное множество точек, наступит такой момент, что для некоторых целых k и l $(l>k)$ наступит равенство $kP=lP$. Из последнего утверждения следует, что для некоторого целого $c=l-k$ будет выполняться равенство $cP=O$. Наименьшее значение c, для которого выполняется равенство $cP=O$, называется ***порядком*** точки P. Для эллиптической группы рассмотренной в примере 5.9 результаты последовательного добавления $P+P+P+\ldots$приведены в таблице 5.3.

Таблица 5.3.

Результат выполнения операции P+P+P+… для E_5(0,1)

+	(0,1)	(0,4)	(2,2)	(2,3)	(4,0)
P	(0,1)	(0,4)	(2,2)	(2,3)	(4,0)
2P	(0,4)	(0,1)	(0,4)	(0,1)	O
3P	O	O	(4,0)	(4,0)	(4,0)
4P	(0,1)	(0,4)	(0,1)	(0,4)	O
5P	(0,4)	(0,1)	(2,3)	(2,2)	(0,4)
6P	O	O	O	O	O

Как видно из приведенной таблицы у каждой точки различный порядок, так точки $(0,1)$ и $(0,4)$ имеют порядок $c=3$, точки $(2,2)$ и $(2,3)$ порядок $c=6$, а точка $(4,0)$ имеет $c=2$.

В реальных криптографических приложениях модуль M эллиптической группы выбирается как большое простое число, содержащее сотни знаков. Кроме того, весьма важным элементом является так называемая генерирующая точка G, порядок которой c должен являться также большим простым числом.

Основой всех существующих криптографических приложения является проблема дискретного логарифма, В случае эллиптических групп эта проблема имеет следующую формулировку.

Для заданных двух точек P и Q эллиптической группы найти такое целое положительное число k для которого выполняется равенство $kP=Q$. Величина k называется дискретным логарифмом от Q по основанию P.

6.7. Квантовая криптография

Развитие новых технологий для создания элементной базы ЭВМ и в первую очередь нанотехнологии, подходя к атомному пределу, неизбежно столкнутся с квантовыми свойствами материи, которые до настоящего времени никак не использовались в сфере информационных технологий. Применение новых свойств материи будет означать радикальную трансформацию идеологии современной вычислительной техники, основанной на известном поведении сигнала (носителя информации), и четко определяемом состоянии носителя этого сигнала.

Из всех квантовых информационных технологий, которые должны прийти на смену существующим приложениям, ближе всего к созданию приложений, пригодных для использования в реальной жизни, подошла квантовая криптография.

Технология квантовой криптографии опирается на фундаментальное свойство природы известное в физике как ***принцип неопределенности Гейзенберга***, сформулированный в 1927г., который базируется на принципиальной неопределенности поведения квантовой системы. Невозможно одновременно получить координаты и импульс частицы, невозможно измерить один параметр фотона, не исказив другой.

Благодаря этому свойству возможно построение каналов передачи данных, защищенных от подслушивания. Отправитель кодирует отправляемые данные, задавая определенные квантовые состояния, а получатель регистрирует эти состояния. Затем получатель и отправитель совместно обсуждают результаты наблюдений. При передаче данных задается и контролируется поляризация фотонов.

Впервые идея квантовой криптографии была предложена в 1984г. ***Чарльзом Беннетом*** и ***Жиль Брассаром,*** которые предложили использовать фотоны в криптографии для получения защищенного канала передачи информации. Для кодирования нулей и единиц они предложили использовать фотоны, поляризованные в различных направлениях, и разработали простую схему квантового распределения ключей шифрования, названную ими ***BB84***. Позднее, в 1991г. идея была развита ***Экертом***.

Сущность предложенной ими идеи заключался в том, что импульс горизонтально поляризованных фотонов проходит через горизонтальный

поляризационный фильтр. Если поворачивать фильтр, то поток пропускаемых фотонов будет уменьшаться до тех пор, пока угол поворота не составит 90 градусов, Тогда ни один фотон из горизонтально поляризованного импульса не будет пропускаться вертикально поляризованным фильтром. При повороте фильтра на 45 градусов он пропустит горизонтально поляризованный фотон с вероятностью 50%.

Реализация идеи заключается в том, что отправителем исходного сообщения свет фильтруется, поляризуется и формируется в виде коротких импульсов малой интенсивности. Поляризация каждого импульса задается отправителем произвольным образом в соответствии с одним из четырех перечисленных состояний (горизонтальная, вертикальная, лево- или право-циркулярная).

Получатель сообщения измеряет поляризацию фотонов, используя произвольную последовательность базовых состояний (ортогональная или циркулярная).

В соответствии с законами квантовой физики, с помощью измерения можно различить лишь два ортогональных состояния: если известно, что фотон поляризован либо вертикально, либо горизонтально, то путем измерения, можно установить, как именно он поляризован. То же самое можно утверждать относительно поляризации под углами 45 и 135 градусов. Однако с достоверностью отличить вертикально поляризованный фотон от фотона, поляризованного под углом 45 градусов, невозможно.

Получатель открыто сообщает отправителю, какую последовательность базовых состояний он использовал. Далее отправитель открыто уведомляет получателя о том, какие базовые состояния использованы получателем корректно. Все измерения, выполненные при неверных базовых состояниях, не совпадающих у получателя и отправителя, отбрасываются. Измерения интерпретируются согласно двоичной схеме: лево-циркулярная поляризация или горизонтальная - 1, право-циркулярная или вертикальная - 0.

Очень важным достоинством канал передачи информации по такому каналу, образованному потоком световых импульсов, является невозможность перехвата сообщения, то есть его подслушивания.

Примером использования квантовой криптографии на практике является ***алгоритм генерирования*** двумя пользователями ***секретного ключа.***

Целью данного алгоритма является получение отправителем и получателем секретного ключа, который будет известен только им обоим.

Первоначально отправитель формирует последовательность фотонных импульсов. Каждый из импульсов случайным образом поляризован в одном из четырех направлений. Возможна вертикальная поляризация (|), горизонтальная (–), лево-циркулярная (\) и право-циркулярная (/). Например, отправитель посылает последовательность из девяти фотонных импульсов, поляризация которых задается, как | / / – \ | | – –.

Получатель настраивает свой детектор произвольным образом (случайно) на измерение серии либо диагонально (×), либо ортогонально (+)

поляризованных импульсов (мерить одновременно и те и другие невозможно).

Предположим, получатель для измерения девяти фотонных импульсов использует поляризацию в следующей последовательности + × + + × × × + ×.

Предположив, что получатель, верно, определил поляризацию фотонов, тогда его регистрирующее устройство будет пропускать либо не пропускать импульс фотонов, что будет означать получение либо единичного символа сообщения, либо нулевого. Если же получатель неверно выбрал поляризацию фильтра, фотон всегда будет проходить либо не проходить через фильтрующее устройство с вероятностью 0,5, так как угол поляризации по отношению к поляризации фильтра составляет 45 градусов. И это не зависит от того, какая поляризация использовалась при передаче исходного сообщения, та, которая соответствует единице или та, которая соответствовала нулю.

Таким образом, при представлении исходного текста как последовательность равновероятных двоичных символов результаты анализа будут представляться как случайная последовательность двоичных цифр.

Для примера, когда отправитель использовал поляризацию | / / – \ | | – –, а получатель + × + + × × × + ×, результаты алгоритма генерирования секретного ключа сведены в таблицу 6.?

Таблица 6.?

Алгоритм генерирования секретного ключа

Отправитель	|	/	/	–	\	|	|	–	–
Получатель	+	×	+	+	×	×	×	+	×
Результат измерений	0	0	0	1	1	0	1	1	0
Анализатор выбран правильно	Да	Да	Нет	Да	Да	Нет	Нет	Да	Нет
Ключ	0	0		1	1			1	

В графе “*Результаты измерения*” представлены результаты физического эксперимента фильтрации потока импульсов фотонов, а конкретные значения свидетельствуют о прохождении импульса фотонов либо нет через конкретный фильтр. Эти результаты являются секретной данными.

Затем по открытому каналу связи получатель сообщает отправителю, какие анализаторы им использовались (см. строку “*Получатель*”), но не сообщает, какие результаты ими были получены.

В ответ отправитель по общедоступному каналу связи сообщает получателю, какие анализаторы он выбрал правильно. Эта информация представлена в строке “*Анализатор выбран правильно*” и она может быть общедоступна. Те импульсы фотонов, для которых получатель неверно выбрал анализатор, отбрасываются и не принимаются для дальнейшего

рассмотрения, а результаты измерения для правильно выбранных результатов могут быть использованы обоими пользователями для криптографического ключа, так как только отправитель и только получатель будут владеть этой информацией. Отправитель в силу того, что он формировал поляризацию сам, а получатель принимал, используя правильную поляризацию своего анализатора.

В среднем, в половине случаем импульсов фотонов будет получен очередной бит ключа, а в остальных случаях информация носит случайный характер в силу неправильно выбранной поляризации анализатора и поэтому игнорируется обоими пользователями.

Если бы злоумышленник производил перехват информации при помощи оборудования, подобного оборудованию Боба, то примерно в 50 процентах случаев он выберет неверный анализатор, не сможет определить состояние полученного им импульса фотонов, и отправит импульс фотонов получателю в состоянии поляризации, выбранной наугад. При этом в половине случаев он выберет неверную поляризацию и, таким образом, примерно в 25 процентах случаев результаты измерений получателя могут отличаться от результатов отправителя.

Для обнаружения подобного перехвата отправитель и получатель выбирают случайный участок ключа и сравнивают его по общедоступному каналу связи. Если процент ошибок велик, то он может быть отнесен на счет злоумышленника, и предыдущий эксперимент полностью аннулируется, а процедура обмена секретным ключом повторяется сначала.