**Министерство образования Республики Беларусь**

**Учреждение образования**
**«Гомельский государственный технический**
**университет имени П. О. Сухого»**

**Кафедра «Информационные технологии»**

**И. А. Мурашко**

# ЗАЩИТА КОМПЬЮТЕРНОЙ ИНФОРМАЦИИ

**ПОСОБИЕ**

**Электронный аналог печатного издания**

**Гомель 2015**

УДК 003.26(075.8)
ББК 32.811.4я73
М91

Рецензент: зав. каф. «Информатика» ГГТУ им. П. О. Сухого
канд. физ.-мат. наук *Т. В. Тихоненко*

**Мурашко, И. А.**

М91 Защита компьютерной информации : пособие по одноим. дисциплине для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» днев. и заоч. форм обучения / И. А. Мурашко. – Гомель : ГГТУ им. П. О. Сухого, 2015. – 48 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-258-8.

Содержит основной теоретический материал дисциплины «Защита компьютерной информации». Данное пособие может быть полезно студентам для изучения криптографических методов защиты информации, представленной в электронной форме.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной и заочной форм обучения.

**УДК 003.26(075.8)**
**ББК 32.811.4я73**

**ISBN 978-985-535-258-8**

# ПРЕДИСЛОВИЕ

Широкое применение компьютерных технологий и постоянное увеличение объема информационных потоков вызывает постоянный рост интереса к криптографии. В последнее время увеличивается роль программных средств защиты информации, просто модернизируемых, не требующих крупных финансовых затрат в сравнении с аппаратными криптосистемами. Современные методы шифрования гарантируют практически абсолютную защиту данных, но всегда остается проблема надежности их реализации.

Пособие по дисциплине «Защита компьютерной информации» разработано для студентов направления специальности 1-40 01 02 «Информационные системы и технологии (в проектировании и производстве)» дневной и заочной форм обучения. Оно предусматривает изучение методов, алгоритмов и программных средств защиты компьютерной информации. Целью пособия является формирование у студентов знаний в области криптографических методов защиты информации.

В пособии большое внимание уделяется изучению математических основ криптографии, реализации основных криптографических преобразований, алгоритмов работы действующих стандартов шифрования. Рассмотрены симметричные и асимметричные алгоритмы шифрования, функции хэширования, алгоритмы формирования электронной цифровой подписи. Кратко рассмотрены принципы защиты компьютерной информации, принципы проверки подлинности электронных документов, методы и алгоритмы защиты электронных документов от несанкционированного изменения.

# 1. ВВЕДЕНИЕ В КРИПТОГРАФИЮ

## 1.1. Основные термины

Информация, циркулирующая в компьютерах или передаваемая через компьютерные сети, становится уязвимой вследствие возможности ее незаконного перехвата и несанкционированного использования [1]. Для защиты компьютерной информации применяют различные методы и программно-аппаратные средства [2]–[4]. Можно выделить следующие основные способы защиты компьютерной информации [5]. Во-первых, организационно-технические методы, целью которых является исключение возможности доступа посторонних к компьютерным средствам, в которых хранится информация [1]. Во-вторых, криптографические методы, которые предполагают шифрование информации [6]–[9]. В-третьих, средства проверки подлинности электронных документов, которые позволяют защитить электронный документ от несанкционированной модификации [10]–[12]. В данном пособии основное внимание уделяется криптографическим методам защиты компьютерной информации.

*Криптография* занимается поиском и исследованием методов преобразования информации с целью ее шифрования. Прямо противоположные цели преследует *криптоанализ*, который занимается исследованием возможности расшифрования информации без знания криптографических ключей [7]. Эти два направления составляют *криптологию* – науку, объектом исследований которой является защита информации посредством ее преобразования.

В рамках криптографии выделяют два основных направления – шифрование (или кодирование) и дешифрование (или декодирование) информации [3]. В качестве информации, подлежащей шифрованию и дешифрованию, будем рассматривать *исходные тексты* или *открытые тексты*, построенные с использованием некоторого алфавита. *Алфавит* – конечное множество элементов (символов), используемых для предоставления исходных текстов. *Исходный текст* – упорядоченный набор из элементов алфавита, используемого для представления исходного текста. *Зашифрованный текст* – упорядоченный набор из элементов алфавита, используемого для представления зашифрованного текста (или *шифротекста*).

*Шифр* – метод записи, в соответствии с которым исходный текст преобразуется в шифротекст.

Как правило, алгоритмы шифрования (дешифрования) не являются секретной информацией. Более того, они формализованы в виде

стандартов (например, *DES – Data Encryption Standard*, ГОСТ 28147–89). Соответственно, секретность сообщению придает некоторая дополнительная информация, или ключ, который известен только узкому кругу лиц (которым разрешен доступ к этому сообщению). Процедура преобразования исходного текста в шифротекст представлена на рис. 1.1.

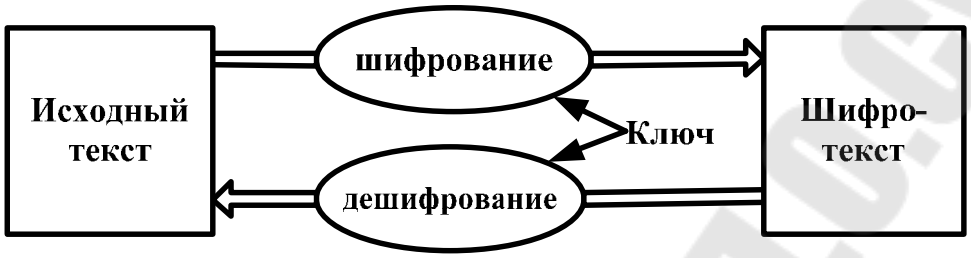



*Рис. 1.1.* Процедура шифрования

Все криптосистемы в зависимости от алгоритма шифрования делятся на симметричные и асимметричные, потоковые и блочные (рис. 1.2) [7].

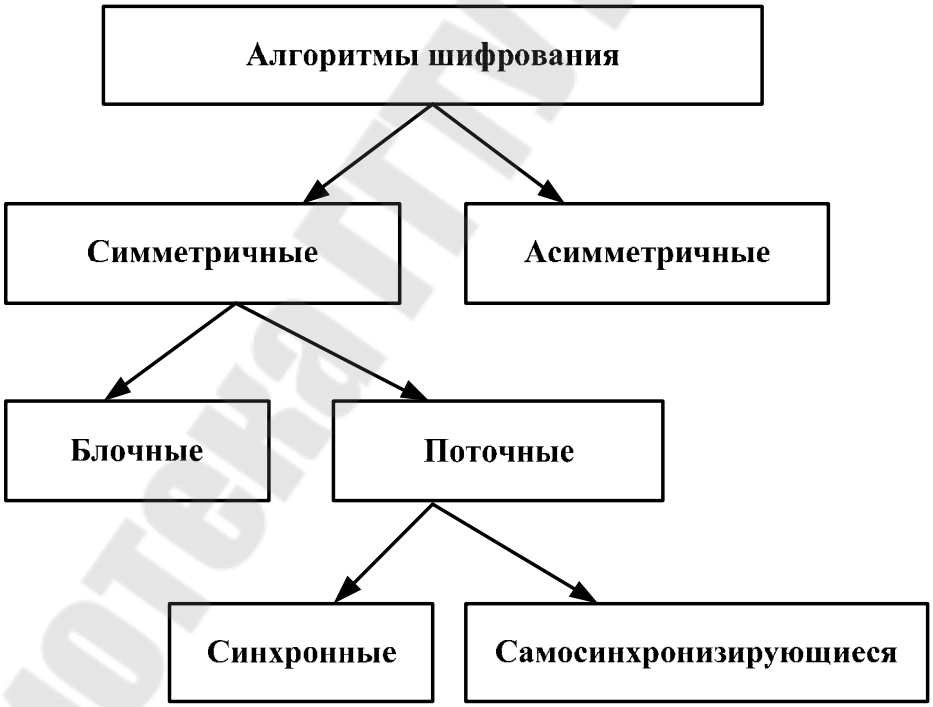



*Рис. 1.2.* Классификация криптосистем

Симметричные криптосистемы предполагают использование только одного ключа для шифрования и расшифровки сообщений. Примерами таких систем являются широко известные стандарты шифрования *DES*, *AES*, ГОСТ 28147–89.

Симметричные криптосистемы в зависимости от процедуры обработки входных сообщений делятся на блочные и потоковые. В блочных алгоритмах исходный текст предварительно разбивается на блоки фиксированной длины (как правило, 64, 128 или 256 бит). Каждый блок шифруется независимо друг от друга. В поточных алгоритмах исходный текст шифруется побитно или побайтно.

Асимметричные криптосистемы используют как минимум два ключа: один для шифрования, второй для расшифровки сообщений.

Шифр может быть взломан, если возможно определить исходный текст или ключ от шифра текста или определить ключ из пары «шифротекст – исходный текст» [3].

Существует несколько *видов атак*:

– атаки, использующие только шифротекст;

– атаки с известным исходным текстом;

– атаки с частично известным исходным текстом.

В первом случае криптоаналитик должен определить ключ исключительно из перехваченного шифротекста, при этом возможно ему будут известны метод шифрования, язык исходного текста, предмет обслуживания, использованные в тексте слова.

Во втором случае криптоаналитик знает несколько пар «шифротекст – исходный текст».

В третьем случае криптоаналитик знает шифротекст, соответствующий определенному исходному тексту.

Шифр *безоговорочно безопасен* (абсолютно секретен), если в независимости от того, сколько шифротекста перехвачено, в нем все равно будет недостаточно информации, чтобы однозначно определить исходный текст [3].

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

Современная криптография защищает данные, передаваемые по линиям связи или хранящиеся в компьютерной системе. Целью защиты является обеспечение:

– скрытости (секретности) данных, т. е. предотвращение несанкционированного доступа к данным;

– достоверности (целостности), т. е. предотвращение несанкционированного изменения данных.

На рис. 1.3 показан безопасный канал передачи данных.

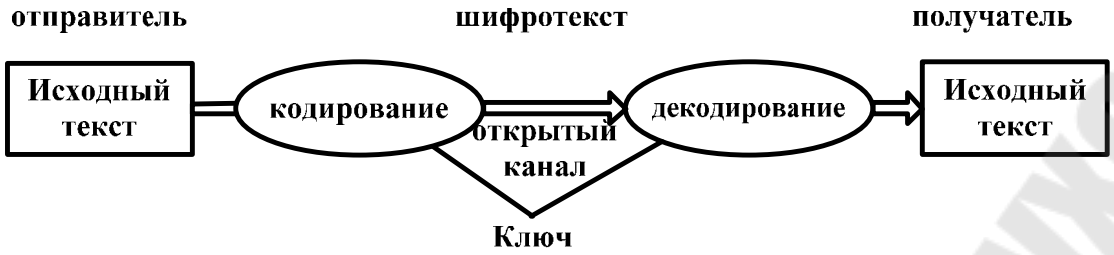



*Рис. 1.3.* Канал передачи данных

Информация, передаваемая по открытому каналу, подвержена перехвату. Различают пассивный перехват, который несет угрозу секретности данных, и активный перехват, который угрожает достоверности. *Пассивный перехват* приводит к перехвату сообщений без изменений. *Активный перехват* приводит к изменению данных.

Различные криптосистемы имеют различную степень защиты. Соответственно, к ним предъявляются определенные требования по секретности и достоверности.

*Требования к секретности*:

1. Для криптоаналитика должно быть вычислительно невозможно определение декодирующего преобразования на основании перехваченного шифротекста, даже если соответствующий исходный текст ему известен.

2. Для криптоаналитика должно быть вычислительно невозможно определение исходного текста на основании перехваченного шифротекста.

*Требования к достоверности*:

1. Для криптоаналитика должно быть вычислительно невозможно определение кодирующего преобразования для заданного шифротекста, даже если исходный текст ему известен.

2. Для криптоаналитика должно быть невозможно нахождение такого шифротекста, который при расшифровке даст корректный исходный текст, определенный в пространстве исходных текстов.

Процесс криптографического закрытия данных может осуществляться как программно, так и аппаратно. При аппаратной реализации достигается высокая производительность шифрования и расшифровки сообщений, простота в использовании и высокая степень защищенности. Однако стоимость такой криптосистемы, как правило, значительно выше, чем при программной реализации. Кроме того, ограничен круг используемых алгоритмов. Программная реализация более практична и гибка в использовании, стоимость ее значительно ниже, однако ниже и степень ее защищенности.

Для современных криптосистем сформулированы следующие требования:

– знание алгоритма шифрования не должно влиять на надежность защиты;

– зашифрованное сообщение должно поддаваться чтению (дешифрованию) только при наличии криптографического ключа;

– незначительное изменение ключа должно приводить к существенному изменению вида зашифрованного сообщения;

– длина шифрованного текста должна быть равна длине исходного текста;

– структурные элементы алгоритма шифрования должны быть неизменными;

– число операций, необходимых для расшифрования информации путем перебора всевозможных ключей, должно иметь строгую нижнюю оценку и выходить за пределы возможностей современных компьютеров (с учетом возможностей использования сетевых вычислений);

– любой ключ из множества возможных ключей должен обеспечивать надежную защиту информации.

## 1.2. Простейшие перестановочные шифры

### 1.2.1. Принцип работы перестановочного алгоритма

С древних времен для сокрытия смысла записанного сообщения люди использовали различные хитрости. Например, удаляли пробелы и писали слова только большими (только малыми) буквами.

| Это лекция по алгоритмам→ ЭТОЛЕКЦИЯПОАЛГОРИТМАМ |
|---|

Следующим шагом усложнения является разбиение зашифрованного текста на блоки.

| ЭТОЛ ЕКЦИ ЯПОА ЛГОР ИТМА М |
|---|

Эффективным способом шифрования является запись слов в обратном порядке.

| ОТЭ ЯИЦКЕЛ ОП МАМТИРОГЛА |
|---|

В общем случае перестановочный шифр переставляет символы исходного текста по определенной схеме. Перестановка может быть представлена в виде геометрической фигуры (рис. 1.4).

| Исходный текст *M* | ==========> Правило записи | **Фигура** | ==========> Правило чтения | Зашифрованный текст *C* |
|---|---|---|---|---|

*Рис. 1.4.* Принцип работы перестановочного алгоритма шифрования

Пример (рис. 1.5): матрица 2 строки, 5 столбцов. Запись построчная. Чтение по столбцам сверху вниз 4, 1, 2, 5, 3.

Исходный текст *M*: шифрование

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | **ш** | **и** | **ф** | **р** | **о** |
| 2 | **в** | **а** | **н** | **и** | **е** |

Шифротекст *C*: ришвиаоефн

*Рис. 1.5.* Пример работы перестановочного алгоритма

Формально эта процедура записывается следующим образом: исходный текст $M$ разбивается на блоки $M = m_1, m_2, \ldots m_i$, все блоки одинаковой длины. Тогда зашифрованный текст будет представлен как совокупность блоков исходного текста, преобразованных в соответствии с функцией шифрования $f$. Все символы исходного текста появятся в зашифрованном тексте. Дешифрация выполняется обратным образом.

### 1.2.2. Шифр «Железнодорожная изгородь»

Пусть имеется правило записи текста следующего вида (рис. 1.6). Здесь число обозначает номер символа в исходном сообщении.

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

*Рис. 1.6.* Пример правила записи

Геометрическая фигура соответствует изгороди. В этом случае исходный текст «ЭТОЛЕКЦИЯПОШИФРАМ» будет записан следующим образом (рис. 1.7).

Э Е Я И М
Т Л К И П Ш Ф А
О Ц О Р

*Рис. 1.7.* Шифр «Железнодорожная изгородь»

При использовании правила чтения по строкам слева направо начиная с первой строки будет получен следующий шифротекст:

«ЭЕЯИМТЛКИПШФАОЦОР».

### 1.2.3. Ключевое слово или ключевая фраза

Одной из наиболее известных модификаций метода перестановки является использование *ключевого слова* или *фразы* в качестве правила перестановки столбцов.

*Пример*: Слово КРИПТОГРАФИЯ может быть использовано как ключ (табл. 1.1). Буквам ключевого слова назначаются номера, начиная с первого, в соответствии с алфавитом. Если буква встречается несколько раз, то нумерация определяется порядком следования повторяющейся буквы в ключевом слове (запись построчно, чтение по столбцам, начиная с первого столбца).

*Таблица 1.1*

**Процедура шифрования с ключом**

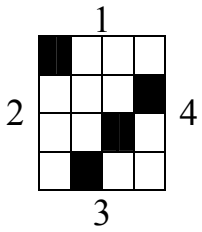
| К | Р | И | П | Т | О | Г | Р | А | Ф | И | Я |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 3 | 7 | 10 | 6 | 2 | 9 | 1 | 11 | 4 | 12 |
| Э | Т | О | Л | Е | К | Ц | И | Я | П | О | А |
| Л | Г | О | Р | И | Т | М | А | М | Ш | И | Ф |
| Р | О | В | А | Н | И | Я | | | | | |

В результате будет получен следующий шифротекст

«ЯМЦМЯООВОИЭЛРКТИЛРАТГОИАЕИНПШАФ».

### 1.2.4. Метод поворачивающейся решетки

Идея метода заключается в следующем: исходный текст записывается через отверстия в решетке, которая по мере заполнения поворачивается на 90º. Предварительно текст разбивается на блоки (в данном случае блок равен 16 символам). Пример решетки представлен на рис. 1.8.




– вырезанные ячейки (куда вписываем текст)

*Рис. 1.8.* Пример решетки

Зашифруем исходный текст, предварительно разбив его на блоки по 16 символов: «ЭТОЛЕКЦИЯПОКРИПТ|ОГРАФИИАБВ…», запись – построчная через прорези, слева направо и сверху вниз (рис. 1.9).

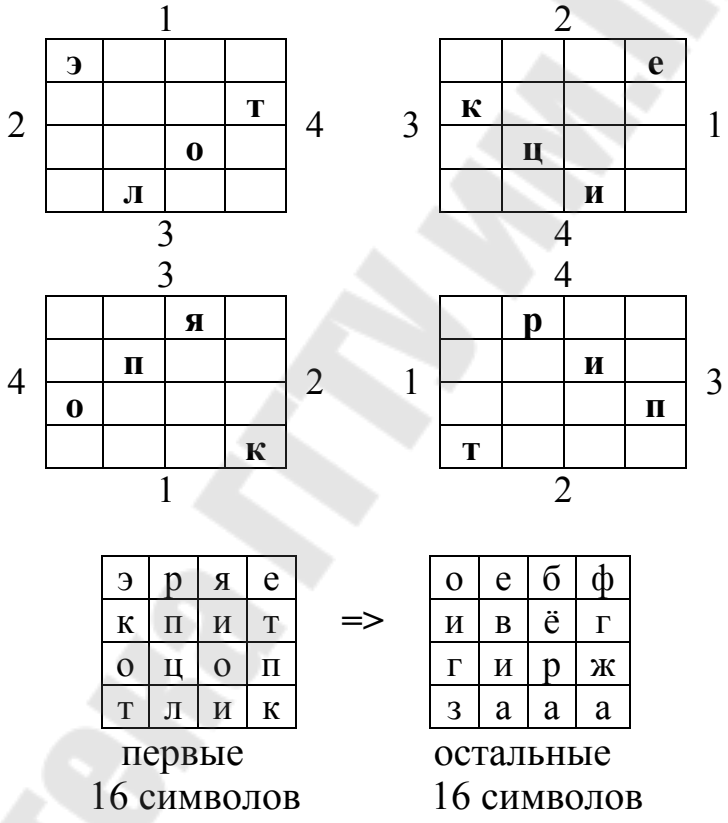



*Рис. 1.9.* Процедура шифрования

Криптотекст: «ЭРЯЕКПИТОИОПТЛИК|ОЕБФИВЁГГИ…».

Изготовить решетку достаточно просто (рис. 1.10):

– строится матрица (4 х 4);

– ячейки матрицы, которые, при повороте матрицы на 90º, занимают одинаковое положение, нумеруются одинаково;

– вырезается один из квадратов с одинаковым номером.

| | | | |
|---|---|---|---|
| **1** | 2 | 3 | 1 |
| 3 | 4 | 4 | **2** |
| 2 | 4 | **4** | 3 |
| 1 | **3** | 2 | 1 |

*Рис. 1.10.* Процедура шифрования

Элементами ключа в данном методе являются размеры решетки, прорези в решетке, алгоритм вращения решетки.

## 1.3. Простейшие подстановочные шифры

Исследования с целью 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 шифр Цезаря. Свое название этот шифр получил по имени римского императора Гая Юлия Цезаря, который использовал этот шифр при переписке с Цицероном.

Таблица подстановки присутствует в неявном виде, т. е. символ шифротекста вычисляется по математическому выражению

$$c_i = (a_i + k) \bmod n,$$

где $a_i$ – символ исходного текста; $k$ – ключ; $n$ – мощность алфавита.

В шифре Цезаря используется $k = 3$. Тогда 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

$$c_i = (a_i + 3) \bmod 26.$$

В данном случае и $a_i$, и $c_i$ представляют собой номера букв в исходном алфавите. При шифровании буквы *B* исходного алфавита, имеющей номер 6, получим $c_i = 1 + 3 = 4$, что соответствует букве *E*, используемой в качестве подстановочного элемента в шифротексте.

Развитием этого метода является использование операции умножения вместо операции сложения (необходимо обеспечить требование $(k, n) = 1$):

$$c_i = (a_i \cdot k) \bmod n.$$

То есть номера символов в шифротексте в $k$ раз больше номеров символов исходного текста, например (при $k = 3$, $n = 26$, $(3,26) = 1$):

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | ... | 24 | 25 |
| A | B | C | D | ... | Y | Z |
| 0 | 3 | 6 | 9 | ... | 20 | U |
| A | D | F | G | ... | 23 | X |

Аффинное преобразование является развитием двух предыдущих методов. Символы шифротекста вычисляются по следующей формуле:

$$c_i = (k_1 a_i + k_2) \bmod n.$$

В данном случае используется 2 ключа: $k_1$ и $k_2$. Причем накладывается требование взаимной простоты $(k_1, n) = 1$.

Подстановочные шифры используют взаимно-однозначное соответствие, т. е. определенному символу исходного текста соответствует конкретный символ шифротекста. Это делает их уязвимыми для атак типа частотный анализ.

Достижения в теории информации позволили формальным образом исследовать исходные тексты, представленные на конкретном языке, и использовать эти результаты для взлома криптосистем.

Суть частотного анализа заключается в следующем: распределение букв в криптотексте сравнивается с распределением букв в алфавите исходного сообщения, делается замена символов шифротекста на символы, имеющие примерно одинаковую частоту встречаемости в исходных текстах. Вероятность успешного взлома повышается с увеличением длины криптотекста. Существует достаточно много различных таблиц распределения частоты встречаемости букв [3]. Наиболее часто встречающиеся буквы русского и английского языка представлены в табл. 1.2.

*Таблица 1.2*

**Частота встречаемости букв русского и английского языка**

| Английский алфавит | | Русский алфавит | |
|---|---|---|---|
| **буква** | **частота встречаемости** | **буква** | **частота встречаемости** |
| *E* | 0,1251 | О | 0,109 |
| *T* | 0,0925 | Е | 0,087 |
| *A* | 0,0804 | А | 0,075 |
| *O* | 0,0726 | И | 0,075 |
| *I* | 0,0726 | Н | 0,064 |
| *N* | 0,0709 | Т | 0,064 |
| *S* | 0,0654 | С | 0,055 |
| *R* | 0,0612 | Р | 0,048 |
| *H* | 0,055 | В | 0,046 |

Кроме того, часто используют частоту встречаемости комбинаций из двух (трех) идущих подряд букв. Так, в английском алфавите наиболее часто встречаются комбинации *TH*, *EN*, с другой стороны, комбинация *OZ* никогда не встречается в осмысленных сообщениях.

Доля имеющих смысл последовательностей букв любого языка понижается с увеличением этой последовательности. Например, в русском языке часто встречается ЕЕ, зато три ЕЕЕ встречаются только в слове *длинношеее*.

Подстановочные шифры используют взаимно-однозначное соответствие, т. е. определенному символу исходного текста соответствует конкретный символ шифротекста. Это делает их уязвимыми для атак типа частотный анализ. Система омофонов обеспечивает простейшую защиту от таких атак. Основная идея заключается в том, каждый символ исходного текста имеет несколько подстановочных элементов. Процедура выбора подстановочного элемента должна выполняться случайно.

Следующий подстановочный шифр также обеспечивает защиту от простейших атак типа частотный анализ. Шифр Виженера изобретался многократно, первые упоминания датированы 16 в. Свое название получил в 19 в. по имени французского дипломата Блеза Виженера. Система шифрования на основе данного шифра является одной из наиболее известных многоалфавитных систем шифрования. Особенностью шифратора Виженера является использование различных таблиц подстановки в зависимости от последовательности символов используемого ключа. Основой шифра является таблица (или квадрат) Виженера (табл. 1.3). В строки и столбцы таблицы записан алфавит, причем каждая последующая строка (столбец) таблицы является циклическим сдвигом предыдущей на один разряд.

*Таблица 1.3*

**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* |
| ***v*** | *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* |

При шифровании исходное сообщение записывают в строку, а под ним ключевое слово или фразу. Если ключ короче, чем сообщение, то его дублируют. На каждом шаге шифрования в верхней строке таблицы находят очередную букву исходного сообщения, а в левом столбце – очередной символ ключа. В результате символ шифротекста будет находиться на пересечении данной строки и данного столбца.

# 2. МАТЕМАТИЧЕСКИЕ ОСНОВЫ КРИПТОГРАФИИ

## 2.1. Целые числа

В криптографии широкое применение нашли целые числа большой разрядности. Целые числа бывают простые и составные.

Целое число является *простым*, если его делителями являются только единица и само это число. Целое число $d$ является *делителем* $n$, если можно записать $n = k \cdot d$ и записывается $d \mid n$. Например: 2, 3, 5, 7, 11, … являются простыми. Числа, имеющие более двух делителей, являются *составными*.

Любое целое число $n > 1$ может быть представлено единственным образом как произведение простых чисел в соответствующих степенях. Такое представление называется *каноническим разложением Евклида*:

$$n = \prod_{i=1}^{k} p_i^{a_i}, \qquad (2.1)$$

где $p_i$ – простые числа; $a_i$ – целые числа.

*Пример*: $250 = 2^1 \cdot 5^3$.

Процедура получения такого разложения называется разложением на простые сомножители или *факторизацией* числа. Факторизация больших чисел имеет большую вычислительную сложность. На настоящий момент не известны быстрые алгоритмы разложения чисел на простые сомножители. На этом факте основана одна из распространенных криптосистем с открытым ключом – *RSA*.

## 2.2. Алгоритмы получения простых чисел

Одной из основных задач криптографии является получение простых чисел, в особенности большой разрядности. Другими словами данную задачу можно сформулировать следующим образом: доказать, что данное число является простым. Евклид показал, что существует бесконечное множество простых чисел, однако их удельный вес среди других чисел невелик. Так, если в первой сотне (от 1 до 100) есть 25 простых чисел, то среди сотни чисел от 900 до 1000 их 14, далее их удельный вес уменьшается (в сотне от 10 000 001 до 10 000 100 простых чисел всего два) [3].

Одним из первых алгоритмов получения простых чисел является решето Эратосфена. Идея этого алгоритма состоит в следующем:

– выписывают все целые числа от 2 до $n$ (2, 3, 4, …, $n$);

– берут первое число (сначала это 2) и зачеркивают в списке все числа, делящиеся нацело на два;

– находят следующее не вычеркнутое число и зачеркивают все числа, делящиеся на это число (кроме него самого);

– алгоритм повторяют для всех чисел, не превосходящих $\sqrt{n}$.

Все не вычеркнутые числа в списке являются простыми числами в диапазоне от 2 до $n$.

*Пример*: $n = 30$, $\sqrt{30} = [5, 47] = 5$.

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.

Берут первое число 2 и зачеркивают в списке все числа, делящиеся нацело на два 2: 2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29.

Берут следующее не вычеркнутое число 3 и зачеркивают в списке все числа, делящиеся нацело на два 3: 2, 3, 5, 7, 11, 13, 17, 19, 23, 25, 29.

Берут следующее не вычеркнутое число 5 и зачеркивают в списке все числа, делящиеся нацело на два 5: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

Следующее не вычеркнутое число $7 > 5$, поэтому алгоритм заканчивает работу. В списке остались только простые числа в диапазоне от 2 до 30.

В настоящее время не существует эффективных алгоритмов для получения простых чисел. Однако существует ряд простых чисел, которые могут быть получены с помощью следующих формул:

1. Числа Мерсенна имеют вид $M_p = 2^p - 1$, где $p$ – простое число. Наибольшим известным простым числом по состоянию на февраль 2013 г. является число Мерсенна $2^{57885161} - 1$. Оно содержит 17 425 170. Его нашли 25 января 2013 г. на математическом факультете университета *UCLA* в рамках проекта по распределенному поиску простых чисел Мерсенна *GIMPS* [13]. Числа Мерсенна выгодно отличаются от остальных наличием эффективного теста простоты: теста Люка–Лемера. Благодаря ему простые числа Мерсенна давно удерживают рекорд как самые большие известные простые. За нахождение простых чисел из более чем 100 000 000 и 1 000 000 000 десятичных цифр *EFF* назначила денежные призы соответственно в 150 000 и 250 000 долларов США. Ранее *EFF* уже присуждала призы за нахождение простых чисел из 1 000 000 и 10 000 000 цифр [13].

2. Числа Ферма имеют вид: $F_p = 2^{2^n} + 1$, $n$ – неотрицательное целое число. Эффективным тестом простоты является тест Пепина. По состоянию на ноябрь 2011 г. известно только 5 простых чисел Ферма (для $n$ = 0, 1, 2, 3, 4), и высказана гипотеза, что других простых чисел Ферма нет [13].

3. Простые числа Эйлера могут быть найдены по следующей формуле: $E_p = x^2 - x + 41$, для $0 < x < 40$.

*Пример*: $x = 7 \Rightarrow E_p = 7^2 - 7 + 41 = 83$ – простое число.

4. Числа Вудала могут быть найдены по следующей формуле: $W_p = n \cdot 2^n - 1$.

5. Числа Прота могут быть найдены по следующей формуле: $P = k \cdot 2^n + 1$, причем $k$ нечетно и $k < 2^n$. Числа Каллена являются частным случаем чисел Прота при $k = n$. Числа Ферма являются частным случаем чисел Прота при $k = 1$ и $n = 2^m$ [13].

## 2.3. Наибольший общий делитель

Составные числа могут быть представлены в канонической форме (2.1).

**Общим делителем** чисел $a_1, a_2, a_3, \ldots a_N$ является целое число $d$ такое, что $d|a_1, d|a_2, \ldots d|a_N$.

**Наибольший общий делитель** (НОД) чисел $a_1, a_2, a_3, \ldots a_N$ – это наибольший делитель $d$, который может быть поделен любым делителем этих чисел. Наибольший общий делитель чисел ($a$, $b$) обозначается как: НОД ($a$, $b$), или ($a$, $b$), или *gcd* ($a$, $b$) – *Greatest Common Divisor*. В дальнейшем будем использовать обозначение ($a$, $b$). *Например*: (30, 15, 25) = 5.

Нахождение НОД является одной из наиболее распространенных процедур в криптографии. В частности, НОД определяет взаимную простоту чисел. Запись ($a$, $b$) = 1 означает, что $a$ и $b$ – взаимно простые числа. *Например*: (13, 15) = 1, т. е. эти числа не имеют общего делителя (кроме единицы). Числа 3 и 15 не являются взаимно простыми, так как (3, 15) = 3 ≠ 1.

Одним из первых инструментов определения взаимной простоты чисел стал *алгоритм Евклида*, который основан на следующем утверждении: если $a$ можно представить как $b \cdot q + r$, то наибольший общий делитель чисел $a$ и $b$ равен наибольшему общему делителю чисел $b$ и $r$, т. е. ($a$, $b$) = ($b$, $r$).

Рассмотрим пример: (512, 375) = [так как $512 = 375 \cdot 1 + 137$] => (375, 137) = [$375 = 137 \cdot 2 + 101$] => (137, 101) = [$137 = 101 \cdot 1 + 36$] => (101, 36) = [$101 = 36 \cdot 2 + 29$] => (36, 29) = [$36 = 29 \cdot 1 + 7$] => (29, 7) = [$29 = 7 \cdot 4 + 1$] => (7, 1) = 1.

Недостатком алгоритма Евклида является применение операций умножения и деления, которые для многоразрядных чисел выполняются достаточно медленно. Более эффективным способом вычисления наибольшего общего делителя является бинарный алгоритм, который основан на следующих утверждениях [8]:

– $(0, n) = n$; $(m, 0) = m$; $(m, m) = m$; $(1, n) = 1$; $(m, 1) = 1$;

– если $m$ и $n$ – четные, то $(m, n) = 2 \cdot (m/2, n/2)$;

– если $m$ – четное, $n$ – нечетное, то $(m, n) = (m/2, n)$;

– если $m$ и $n$ – нечетные, то $(m, n) = ((n – m)/2, m)$.

*Пример*: (375, 95) = [так как 375 и 95 – нечетные] => ((375 – 95), 95) = (280, 95) = [280 – четное, 95 – нечетное] => (140, 95) = [140 – четное, 95 – нечетное] => (70, 95) = [70 – четное, 95 – нечетное] => (35, 95) = [35 и 95 – нечетные] => (35, (95 – 35)) = (35, 60) = [60 – четное, 35 – нечетное] => (35, 30) = [35 – нечетное, 30 – четное] => (35, 15) = [35 и 15 – нечетные] => ((35 – 15), 15) = (20, 15) = [20 – четное, 15 – нечетное] => => (10, 15) = [10 – четное, 15 – нечетное] => (5, 15) = [5 и 15 – нечетные] => => (5, (15 – 5)) = (5, 10) = [5 – нечетное, 10 – четное] => (5, 5) = 5.

Числа $a_1, a_2, a_3, \ldots, a_n$ называются *взаимно простыми*, если их наибольший общий делитель равен единице, или $(a_1, a_2, \ldots, a_n) = 1$.

## 2.4. Сравнения

Два целых числа $a$ и $b$ *сравнимы* (конгруэнтны) по модулю натурального числа $m$, если при делении на $m$ они дают одинаковые остатки. *Например*: 32 и 39 сравнимы по модулю 7, так как $32 = 4 \cdot 7 + 4$, $39 = 5 \cdot 7 + 4$. Записывается в следующем виде: $a \equiv b \bmod m$.

В общем случае числа $a$ и $b$ *сравнимы по модулю* $m$, если их разность $a – b$ делится нацело на $m$. Числа $a$ и $b$ *несравнимы по модулю* $m$, если их разность не делится нацело на $m$. Если $a = b \bmod m$, и $b < m$, то $b$ называется *вычетом* $a$ по модулю $m$.

Рассмотрим основные свойства сравнений:

– если $a = b \bmod m$, то для любого целого $k$ будет справедливо: $k \cdot a = k \cdot b \bmod m$;

– если $k \cdot a = k \cdot b \bmod m$ и $(k, m) = 1$, то $a = b \bmod m$;

– если $k \cdot a = k \cdot b \bmod k \cdot m$, то $a = b \bmod m$;

– если $a = b \bmod m$, с $= d \bmod m$, то $a + c = (b + d) \bmod m$;

– если $a_1 = b_1 \bmod m$, $a_2 = b_2 \bmod m$ и т. д., то $(a_1 + a_2 + \ldots + a_n) = (b_1 + b_2 + b_n) \bmod m$;

– если $a = b \bmod m$ и $c = d \bmod m$, то $a \cdot c = b \cdot d \bmod m$;

– если $a = b \bmod m$, то для любого целого $k$: $a^k = b^k \bmod m$.

Для операций сложения, вычитания и умножения справедливо следующее соотношение: $(a \cdot b) \bmod m = [(a \bmod m) \cdot (b \bmod m)] \bmod m$. Здесь «·» обозначена одна из трех операций: сложение (+), вычитание (–), умножение (·). Применение данного выражения позволяет существенно упростить вычисления.

*Пример*: $(8 \cdot 11) \bmod 5 = [(8 \bmod 5) \cdot (11 \bmod 5)] \bmod 5 = [3 \cdot 1] \bmod 5 = 3$.

Рассмотрим, как можно использовать данное соотношение для упрощения возведения целых чисел в степень по модулю некоторого числа. Типовая реализация алгоритма возведения в степень $a^n \bmod r$ предполагает вычисление $a^n$, а затем нахождение остатка по модулю $r$. В общем случае требуется $(n - 1)$ операция умножения и одна операция деления. Например, $7^5 \bmod 11 = 7 \cdot 7 \cdot 7 \cdot 7 \cdot 7 \bmod 11 = 16\,807 \bmod 11 = 10$. Вычисления можно ускорить, если использовать следующее преобразование: $7^5 = (7^2)^2 \cdot 7$. В этом случае требуется только три операции умножения. Однако как в первом, так и во втором случае промежуточные результаты имеют большую разрядность. Соответственно, избавиться от чисел большой разрядности можно в том случае, если все промежуточные результаты брать по модулю.

Например: $7^5 \bmod 11 = (((7^2 \bmod 11)^2 \cdot \bmod 11) \cdot 7) \bmod 11 = (((5)^2 \cdot \bmod 11) \cdot 7) \bmod 11 = ((3) \cdot 7) \bmod 11 = 21 \bmod 11 = 10$.

Вычисления можно оптимизировать, если разложить показатель степени на слагаемые, представляющие собой степени числа 2 (другими словами, перевести показатель в двоичную форму).

Например: требуется вычислить $a^{25} \bmod r$. Представим $25 = 2^4 + 2^3 + 2^1$. Тогда $a^{25} \bmod r = (a^{16} \bmod r) \cdot (a^8 \bmod r) \cdot (a^1 \bmod r)^2 \bmod r = (((((a^2 \bmod r)^2 \bmod r)^2 \bmod r)^2 \bmod r) \cdot (((a^2 \bmod r)^2 \bmod r)^2 \bmod r) \cdot a$. В этом случае требуется только 9 умножений (а если сохранить результат $(a^8 \bmod r)$, то только 6).

Функцией *Эйлера* $\varphi(n)$ целого числа $n \geq 1$ является *количество* целых чисел, которые меньше, чем $n$, и взаимно просты с $n$.

Значения функции Эйлера для $n = 1, 2, \ldots, 9$ представлены в табл. 2.1.

*Таблица 2.1*

**Значение функции Эйлера**

| ***n*** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| **φ(*n*)** | 0 | 1 | 2 | 2 | 4 | 2 | 6 | 4 | 6 |

Вычислить функцию Эйлера можно по следующим выражениям.

1. Для простого числа $p$ функция Эйлера равна $\varphi(p) = p - 1$.

2. Для числа $n$, которое можно представить в виде произведения простых чисел $n = p \cdot q$ ($p$ и $q$ – простые числа), функция Эйлера равна $\varphi(n) = \varphi(p) \cdot \varphi(q) = (p - 1) \cdot (q - 1)$.

3. Если $p$ – простое число и $k > 0$ – целое число, то
$\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p - 1)$.

4. Для произвольного случая необходимо представить целое число $n$ в виде канонического разложения Евклида: $n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_r^{a_r}$, тогда $\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_r})$.

*Пример*: $n = 4500 = 2^2 \cdot 3^2 \cdot 5^3$. $\varphi(4500) = 4500 \cdot (1 - \frac{1}{2}) \cdot (1 - \frac{1}{3}) \cdot (1 - \frac{1}{5}) = 1200$.

Теорема *Эйлера*: если $n \geq 0$ – целое число и $(a, n) = 1$, то

$$a^{\varphi(n)} = 1 \bmod n.$$

*Пример*: $10^{220} \bmod 253 = [(10, 253) = 1, \varphi(253) = \varphi(11 \cdot 23) = \varphi(11) \times \times \varphi(23) = 10 \cdot 22 = 220] = 1$.

## 2.5. Решение линейных сравнений

Под *линейным сравнением* понимают выражение вида
$\boldsymbol{a \cdot x = b \bmod n}$, где $a$, $b$, $n$ – целые числа, $b < n$.

В общем случае существует 3 возможных ситуации:

1) линейное сравнение не имеет решения;

2) линейное сравнение имеет 1 решение;

3) линейное сравнение имеет множество решений.

Рассмотрим первый случай. Если $(a, n) = d$ не является делителем $b$, то линейное сравнение $a \cdot x = b \bmod n$ не имеет решения.

Например: $2 \cdot x = 1 \bmod 4$. В этом случае $d = (2, 4) = 2$, а 2 не является делителем $b = 1$, следовательно, линейное сравнение не имеет решения. Проверим это утверждение полным перебором. Переменная $x$ может принять только четыре значения: 0, 1, 2, 3. Пусть $x = 0$, тогда $2 \cdot x \bmod 4 = 0 \neq 1$. Пусть $x = 1$, тогда $2 \cdot x \bmod 4 = 2 \neq 1$. Пусть $x = 2$, тогда $2 \cdot x \bmod 4 = 0 \neq 1$. Пусть $x = 3$, тогда $2 \cdot x \bmod 4 = 2 \neq 1$.

Рассмотрим второй случай. Если $(a, n) = 1$, т. е. $a$ и $n$ являются взаимно простыми числами, то линейное сравнение $a \cdot x = b \bmod n$ имеет одно решение.

Например, $2 \cdot x = 1 \bmod 3$. Здесь $(2, 3) = 1$. Найдем решение полным перебором. Переменная $x$ может принять только три значения: 0, 1, 2. Пусть $x = 0$, тогда $2 \cdot x \bmod 3 = 0 \neq 1$. Пусть $x = 1$, тогда $2 \cdot x \bmod 3 = 2 \neq 1$. Пусть $x = 2$, тогда $2 \cdot x \bmod 3 = 1$. Получили, решением является $x = 2$.

Рассмотрим третий случай. Если $(a, n) = d$ и $d$ является делителем $b$, то линейное сравнение $a \cdot x = b \bmod n$ имеет $d$ решений.

Например, $6 \cdot x = 4 \bmod 10$. Здесь $(6, 10) = 2$. Найдем решение полным перебором. Переменная $x$ может принять десять значений: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Подставляя эти значения в сравнение, получим, что решением являются $x = 4$ и $x = 9$.

Рассмотрим формальный метод решения линейных сравнений. Ограничимся случаем, когда имеется только одно решение, т. е. $a$ и $n$ являются взаимно простыми числами. Пусть $b = 1$. Тогда $a \cdot x = 1 \bmod n$, следовательно, $x = a^{-1}$. Другими словами, $x$ является мультипликативной инверсной величиной по отношению к $a$ по модулю $n$. Как показано в [3], решение может быть найдено как: $x = a^{\varphi(n)-1} \bmod n$.

*Пример*: $2 \cdot x = 1 \bmod 3$. В этом случае $(2, 3) = 1$, $\varphi(3) = 2$, поэтому $x = 2^{2-1} \bmod 3 = 2$.

Пусть $b \neq 1$. Тогда $a \cdot x = b \bmod n$. Как показано в [3], решение может быть найдено как: $x = b \cdot a^{\varphi(n)-1} \bmod n$.

*Пример*: $4 \cdot x = 3 \bmod 9$. В этом случае $(4, 9) = 1$, $\varphi(9) = 6$, поэтому $x = 3 \cdot 4^{6-1} \bmod 9 = 3$.

# 3. СИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ

## 3.1. Криптографический стандарт *DES*

До 70-х гг. прошлого века криптография являлась достаточно засекреченной наукой и применялась в основном в военной сфере. С развитием сетей передачи данных и широким применением компьютеров в гражданских областях появилась потребность безопасного хранения информации в системах коллективного доступа. Поэтому в 1972 г. Национальное бюро стандартов (*NBS*) выступило инициатором создания стандартного криптографического алгоритма. Алгоритм должен быть относительно недорогим, легко доступным, поддерживать аппаратную и программную реализацию, обеспечивать высокий уровень безопасности и возможность сертификации. Такой алгоритм был разработан специалистами *IBM* на базе шифра *LUCIFER*. В 1976 г. стандарт шифрования данных *DES* (*Data Encryption Standard*) был принят в качестве федерального стандарта и разрешен к использованию во всех несекретных правительственных коммуникациях. В 1981 г. Американский национальный институт стандартов (*ANSI*) одобрил *DES* в качестве стандарта для частного сектора и финансовой сферы. Этот стандарт позволяет шифровать 64-битный блок данных, используя 56-битный ключ. Для шифрования и дешифрования сообщений используется один и тот же алгоритм. Различие между шифрованием и расшифровкой заключается только в последовательности использования раундовых ключей.

Основу *DES* составляет многократное использование простейших криптографических преобразований – подстановок и перестановок. Одна итерация (или раунд) шифрования использует только простейшие арифметические операции, что позволяет легко выполнить как аппаратную, так и программную реализацию. Всего при шифровании блока данных используется 16 итераций.

Рассмотрим работу алгоритма (рис. 3.1). Входной 64-битный блок данных (*T*) перемешивается в соответствии с таблицей *IP* (рис. 3.2, *а*). Номер символа в блоке лежит в пределах от 0 до 63, т. е. состоит из двух десятичных цифр. Самый левый столбец таблицы соответствует десяткам, а самая верхняя строка соответствует единицам. В ячейке таблицы записан номер символа, который попадет в данную позицию после перестановки. Например, в позицию десятого символа попадет 44-й символ.

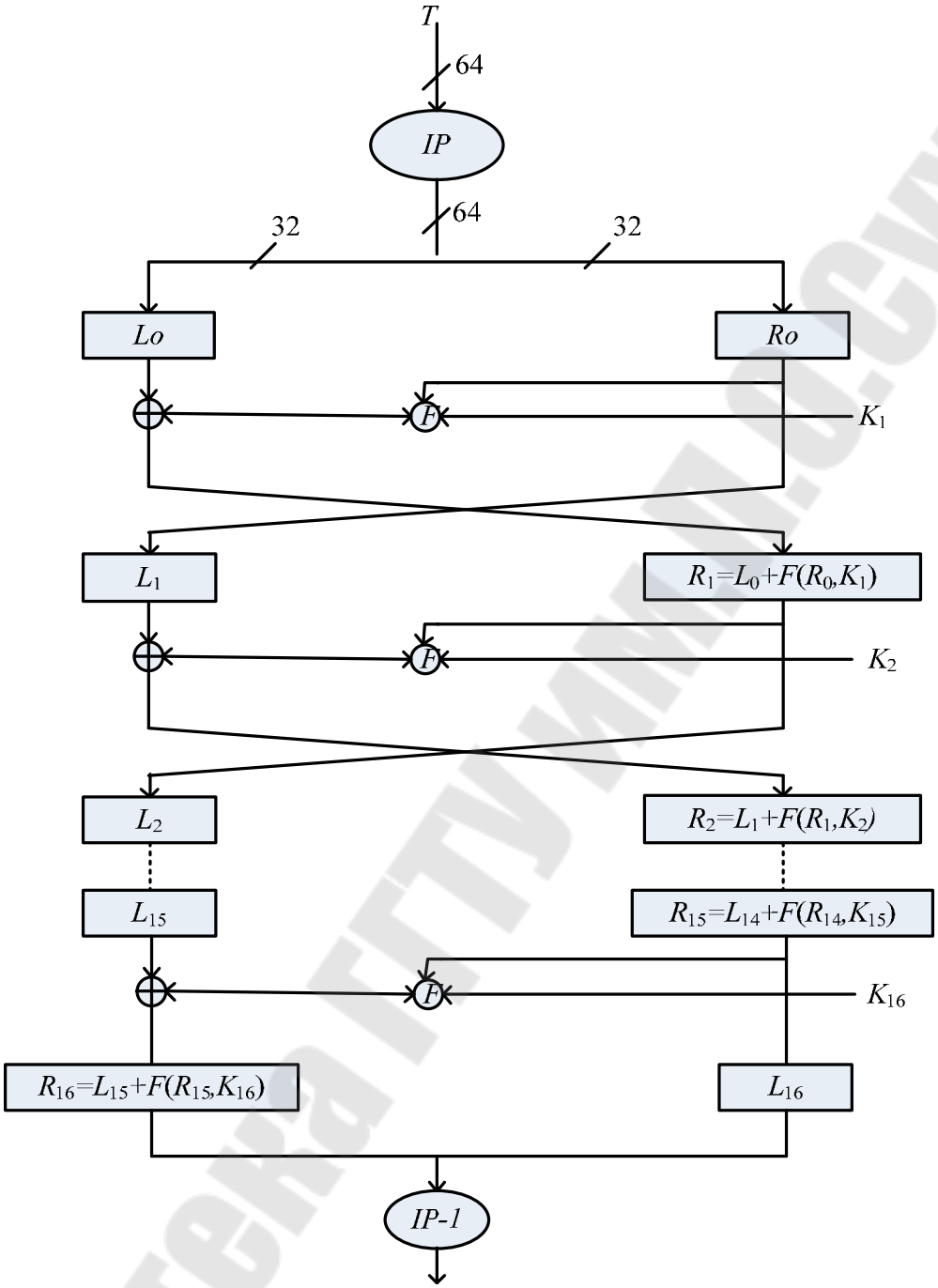



*Рис. 3.1.* Алгоритм *DES*

После прохождения через 16 итераций шифрования он перемешивается согласно обратной таблице перестановки $IP^{-1}$ (рис. 3.2, *б*). Входная и выходная перестановки связаны взаимоотношением

$$IP^{-1}\,(IP(T)) = T.$$

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 | 60 | 52 |
| **1** | 44 | 36 | 28 | 20 | 12 | 4 | 62 | 54 | 46 | 38 |
| **2** | 30 | 22 | 14 | 6 | 64 | 56 | 48 | 40 | 32 | 24 |
| **3** | 16 | 8 | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| **4** | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | 61 | 53 |
| **5** | 45 | 37 | 29 | 21 | 13 | 5 | 63 | 55 | 47 | 39 |
| **6** | 31 | 23 | 15 | 7 | | | | | | |

*а*)

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 | 39 | 7 |
| **1** | 47 | 15 | 55 | 23 | 63 | 31 | 38 | 6 | 46 | 14 |
| **2** | 54 | 22 | 62 | 30 | 37 | 5 | 45 | 13 | 53 | 21 |
| **3** | 61 | 29 | 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| **4** | 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 | 34 | 2 |
| **5** | 42 | 10 | 50 | 18 | 58 | 26 | 33 | 1 | 41 | 9 |
| **6** | 49 | 17 | 57 | 25 | | | | | | |

*б*)

*Рис. 3.2*. Таблица входных перестановок *IP* (*а*)
и выходных перестановок $IP^{-1}$ (*б*)

После входной перестановки 64-битный блок делится на две части – правую ($R_0$) и левую ($L_0$). В $L_0$ попадают первые 32 бита, в $R_0$ – последние 32 бита. Далее происходит 16 итераций шифрования, которые описываются следующими соотношениями:

$$L_i = R_{i-1};$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i),$$

где $\oplus$ – побитовая операция сложения по модулю два; $i$ – номер итерации шифрования; $K_i$ – значение ключа на $i$-й итерации.

Рассмотрим преобразование $F$, включающее в себя перестановки и подстановки. Схема преобразования представлена на рис. 3.3. Сначала 32-битный операнд $R_{i-1}$ расширяется до 48 бит при помощи таблицы $E$ (рис. 3.4). Полученное 48-битное слово поразрядно суммируется по модулю два с ключом $K_i$. Результат разбивается на 8 блоков по 6 бит. Каждый 6-разрядный блок поступает на вход перестановочной функции, реализованной при помощи $S$-боксов ($S_1 – S_8$).

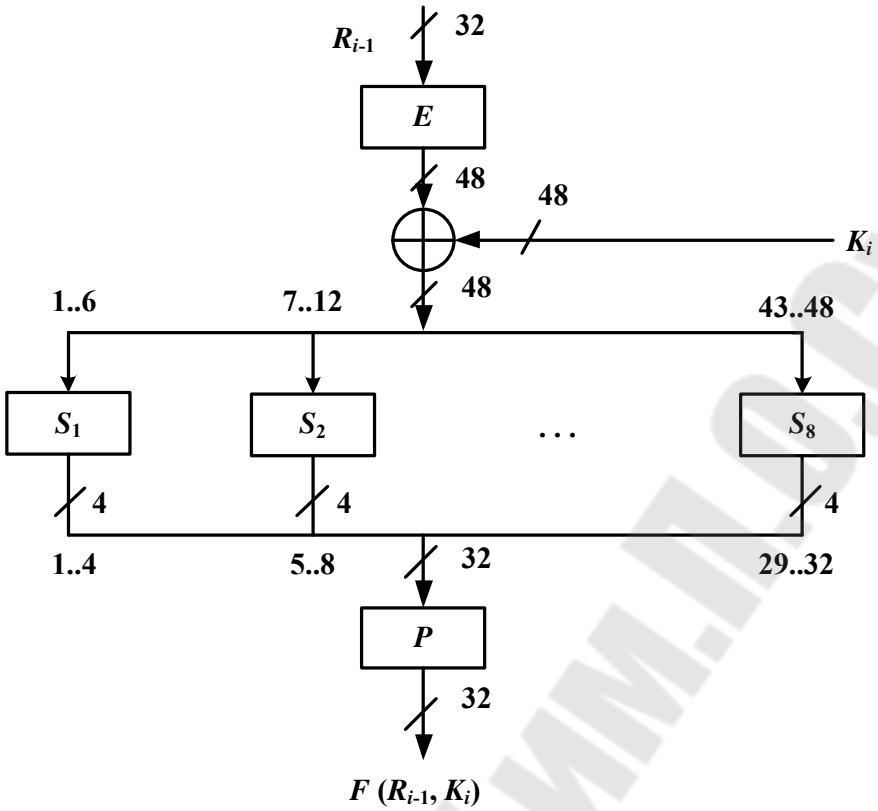



*Рис. 3.3*. Преобразование *F*

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 32 | 1 | 2 | 3 | 4 | 5 | 4 | 5 | 6 | 7 |
| **1** | 8 | 9 | 8 | 9 | 10 | 11 | 12 | 13 | 12 | 13 |
| **2** | 14 | 15 | 16 | 17 | 16 | 17 | 18 | 19 | 20 | 21 |
| **3** | 20 | 21 | 22 | 23 | 24 | 25 | 24 | 25 | 26 | 27 |
| **4** | 28 | 29 | 28 | 29 | 30 | 31 | 32 | 1 | | |

*Рис. 3.4.* Таблица *E*

Каждый *S*-бокс возвращает 4-битовое значение. Результаты объединяются в 32-битный блок и поступают на вход таблицы перестановки *P* (рис. 3.5).

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 | 1 | 15 |
| **1** | 23 | 26 | 5 | 12 | 31 | 10 | 2 | 8 | 24 | 14 |
| **2** | 32 | 27 | 3 | 9 | 19 | 13 | 30 | 6 | 22 | 11 |
| **3** | 4 | 25 | | | | | | | | |

*Рис. 3.5*. Таблица *P*

*S*-бокс отображает 6-битовый блок $B_j = b_1\ b_2\ b_3\ b_4\ b_5\ b_6$ в соответствии со своей таблицей подстановки. Сначала из $B_j$ выделяются крайние биты $b_1$ и $b_6$, которые являются номером строки. Оставшиеся биты определяют номер столбца. В табл. 3.1 приведен пример таблицы подстановки *S*-бокса $S_1$. Пример: $B_j = \underline{1}10001\underline{1}$, получили: 11 = 3 – номер строки, 1000 = 8 – номер столбца. Таким образом, $S_j$ (110001) = = 5 = 0101.

*Таблица 3.1*

**Таблица подстановки *S*-бокса $S_1$**

| **Строка** | | **Номер столбца ($b_2\ b_3\ b_4\ b_5$)** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $b_1$ | $b_6$ | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 0 | 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0 | 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 1 | 0 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 1 | 1 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

Рассмотрим процедуру вычисления ключа. Для стандарта *DES* входной ключ является 64-битным. Однако каждый восьмой разряд не используется (это разряды 8, 16, 24, 32, 40, 48, 56, 64, которые используются как контрольные разряды), поэтому в шифровании используется только 56 разрядов. После этого для каждой из 16 итераций шифрования и дешифрования используется новое значение ключа $K_i$, разрядность которого равна 48 битам. Вычисление ключей происходит по следующей схеме (рис. 3.6).

Таблица $PC_1$ удаляет контрольные разряды и перемешивает оставшиеся биты в соответствии с рис. 3.7. Результат делится на две части пополам. Получаем $C_0$ и $D_0$.

Затем каждая из частей циклически сдвигается влево на число позиций в зависимости от номера итерации, которое определяется таблицей сдвига (табл. 3.2).

*Таблица 3.2*

**Таблица сдвигов**

| ***i*** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Сдвиг** | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Далее из 56 бит формируется 48-разрядный ключ в соответствии с таблицей $PC_2$, представленной на рис. 3.8.

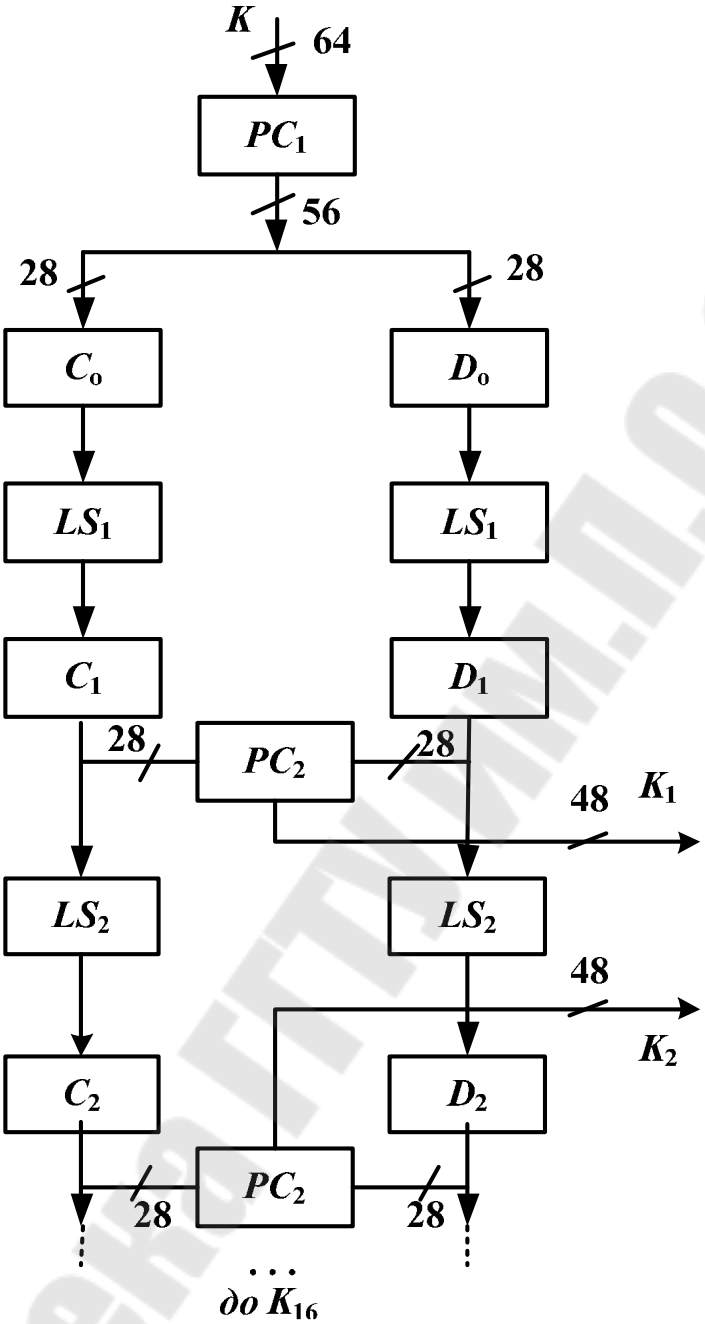



*Рис. 3.6*. Схема вычисления ключа $K_i$

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 58 | 50 |
| **1** | 42 | 34 | 26 | 18 | 10 | 2 | 59 | 51 | 43 | 35 |
| **2** | 27 | 19 | 11 | 3 | 60 | 52 | 44 | 36 | 63 | 55 |
| **3** | 47 | 39 | 31 | 23 | 15 | 7 | 62 | 54 | 46 | 38 |
| **4** | 30 | 22 | 14 | 6 | 61 | 53 | 45 | 37 | 29 | 21 |
| **5** | 13 | 5 | 28 | 20 | 12 | 4 | | | | |

*Рис. 3.7.* Таблица $PC_1$

| | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 14 | 17 | 11 | 24 | 1 | 5 | 3 | 28 | 15 | 6 |
| **1** | 21 | 10 | 23 | 19 | 12 | 4 | 26 | 8 | 16 | 7 |
| **2** | 27 | 20 | 13 | 2 | 41 | 52 | 31 | 37 | 47 | 55 |
| **3** | 30 | 40 | 51 | 45 | 33 | 48 | 44 | 49 | 39 | 56 |
| **4** | 34 | 53 | 46 | 42 | 50 | 36 | 29 | 32 | | |

*Рис. 3.8*. Таблица $PC_2$

Дешифрование по стандарту *DES* выполняется по тому же самому алгоритму. Отличием является только последовательность использования ключей. Ключ, который используется при шифровании на первой итерации, в дешифровании используется на последней итерации и т. д.

На рис. 3.9 представлена типовая реализация *DES*. Для шифрования и дешифрования используется один ключ *K*_1. Однако разрядность ключа в 56 бит не всегда удовлетворяет пользователя. Поэтому появились модификации этого алгоритма: двойной *DES* (рис. 3.10, *а*) и тройной *DES* (рис. 3.10, *б*).

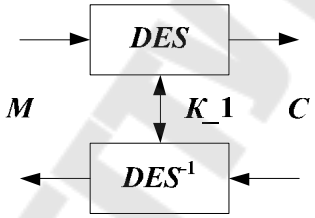
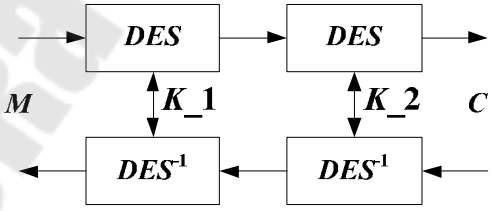
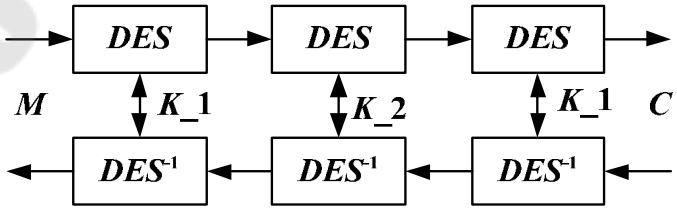



*Рис. 3.9*. Типовая реализация *DES*




*а*)




*б*)

*Рис. 3.10*. Модификации *DES*: *а* – двойной *DES*; *б* – тройной *DES*

Обе модификации позволяют увеличить разрядность ключа до 112 бит. Предпочтение следует отдать последней модификации, которая рассматривалась как кандидат на новый криптографический стандарт, но не была утверждена, так как были разработаны более сильные алгоритмы.

## 3.2. Алгоритм шифрования данных *IDEA*

Первую версию алгоритма разработали в 1990 г. Лай Сюэцзя (*Xuejia Lai*) и Джеймс Мэсси (*James Massey*) из Швейцарского института *ETH Zürich* в качестве замены *DES* и назвали ее *PES* (*Proposed Encryption Standard*). Затем, после публикации работ Бихама и Шамира по дифференциальному криптоанализу *PES*, алгоритм был улучшен с целью усиления криптостойкости и назван *IPES* (англ. *Improved Proposed Encryption Standard*). Через год его переименовали в *IDEA* (*International Data Encryption Algorythm*). Как и *DES*, этот алгоритм основан на последовательности операций подстановок и перестановок. Патент на этот алгоритм принадлежит швейцарской фирме *Ascom*, которая разрешила бесплатное некоммерческое использование (применяется в общедоступном пакете конфиденциальной версии электронной почты *PGP*).

При разработке алгоритма *IDEA* старались учесть следующие требования. Сложность алгоритма требует уменьшать размер блока данных. С другой стороны, чем больше блок, тем меньше статистическая зависимость между блоками данных. Как компромисс выбрано значение 64 бита. Длина ключа также имеет сильное влияние на эффективность алгоритма. Чем длиннее ключ, тем выше криптостойкость, однако тем ниже скорость шифрования. Компромисс – 128 бит. Кроме того, авторы постарались сделать максимально сложной и неочевидной зависимость шифротекста от исходного текста.

В *IDEA* применяются 52 раундовых ключа размером 16 бит. Исходный текст делится на четыре группы по 16 бит.

Для достижения качества шифрования были введены следующие операции.

1. Побитовое сложение по модулю два 16-разрядных операндов. Обозначается как *XOR* или $\oplus$.

2. Сложение целых 16-разрядных операндов по модулю $2^{16}$. Обозначается как $\boxtimes$. Сложение представляет собой обычную операцию по модулю 65536 с переносом.

3. Умножение целых 16-разрядных операндов по модулю $2^{16}+1$. Обозначается как $\odot$. Чтобы операция была обратимой, вместо нуля используется код 65536.

Таблицы истинности для двухбитных операндов представлены на рис. 3.11.

| ⊕ | | | | |
|---|---|---|---|---|
| x \ y | 00 | 01 | 10 | 11 |
| 00 | 00 | 01 | 10 | 11 |
| 01 | 01 | 00 | 11 | 10 |
| 10 | 10 | 11 | 00 | 01 |
| 11 | 11 | 10 | 01 | 00 |

| ⊠ | | | | |
|---|---|---|---|---|
| x \ y | 00 | 01 | 10 | 11 |
| 00 | 01 | 00 | 11 | 10 |
| 01 | 01 | 10 | 11 | 00 |
| 10 | 10 | 11 | 00 | 01 |
| 11 | 11 | 00 | 01 | 10 |

| ⊙ | | | | |
|---|---|---|---|---|
| x \ y | 00 | 01 | 10 | 11 |
| 00 | 01 | 00 | 11 | 10 |
| 01 | 00 | 01 | 10 | 11 |
| 10 | 11 | 10 | 00 | 01 |
| 11 | 10 | 11 | 01 | 00 |

*Рис. 3.11.* Арифметические операции *IDEA*

Рассмотрим процедуру шифрования.

Исходный незашифрованный 64-битный блок делится на четыре подблока по 16 бит (обозначим их *A*, *B*, *C* и *D*). В процессе шифрования все преобразования совершаются над 16-битными числами. Для шифрования и расшифрования используют один и тот же алгоритм, состоящий из восьми идентичных раундов и одного выходного (рис. 3.12). Используется 54 ключа $K_1$–$K_{52}$ (по шесть ключей в каждом раунде и четыре ключа в выходном).

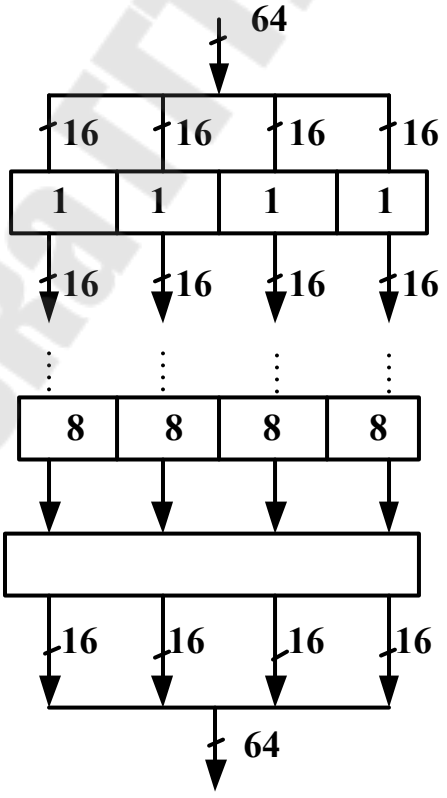



*Рис. 3.12*. Процедура шифрования

*Первый раунд шифрования.* Вначале вычисляются:
$A = A \odot K_1$; $B = B \boxtimes K_2$; $C = C \boxtimes K_3$; $D = D \odot K_4$.
Затем находят:
$E = A \oplus C$;
$F = B \oplus D$;
$E = E \odot K_5$;
$F = F \boxtimes E$;
$F = F \odot K_6$;
$E = E \boxtimes F$;
$A = A \oplus F$;
$C = C \oplus F$;
$B = B \oplus E$;
$D = D \oplus E$.
Затем меняются местами $B$ и $C$.

Так повторяется восемь раз, причем на последующем раунде используются новые ключи (на втором раунде $K_7$–$K_{12}$; на третьем раунде $K_{13}$–$K_{18}$ и т. д.). После восьмого раза $B$ и $C$ местами не меняются.

В выходном (девятом) раунде находим:
$A = A \odot K_{49}$;
$B = B \boxtimes K_{50}$;
$C = C \boxtimes K_{51}$;
$D = D \odot K_{52}$.

Для ускорения шифрования разработаны специальные аппаратные средства. Для расшифрования используется взаимообратная процедура. Ключи дешифрования получаются из ключей шифрования по формулам:
$Z^{-1}_j \odot Z_j = 1 \bmod (2^{16} + 1)$;
$-Z_j \boxtimes Z_j = 0 \bmod 2^{16}$.

## 3.3. Стандарт AES

В 1997 г. *NIST* (Американский институт стандартизации) объявил конкурс на новый стандарт симметричного криптографического алгоритма. Основными требования к кандидату были: длина ключа – не меньше 128 бит; простота программной и аппаратной реализации, высокая скорость шифрования. Победителем стал алгоритм *Rijndael*, который лег в основу нового стандарта шифрования *AES* (*Advanced Encryption Standard*). С этого момента с алгоритма сняты все патентные ограничения, т. е. его можно использовать в любом приложении.

Алгоритм представляет собой симметричный блочный шифр, который работает с блоками данных длиной 128 бит и использует

ключи длиной 128, 192 и 256 бит. Алгоритм *Rijndael* может работать и с другими длинами блоков данных и ключей, но эта возможность в стандарт не вошла. По заявлению правительства США при использовании 128-битного ключа для взлома шифрования потребуется 149 триллионов лет.

При шифровании данные разбиваются на 4 слова по 32 бита ($N_b = 4$). Криптографический ключ может иметь длину: 128, 196, 256 бит ($N_k = 4, 6, 8$). В зависимости от размера ключа количество раундов шифрования может быть: 10, 12, 14.

В *AES* все байты данных представляются в векторной форме $b_7, b_6, \ldots, b_0$. Векторам соответствует полиномиальное представление:

$$b_7x^7 \oplus b_6x^6 \oplus \ldots \oplus b_1x \oplus b_0x^0 = \sum_{i=0}^{7} b_i x^i .$$

*Пример*:

$$00110101 = 0 \cdot x^7 \oplus 0 \cdot x^6 \oplus 1 \cdot x^5 \oplus 1 \cdot x^4 \oplus 0 \cdot x^3 \oplus 1 \cdot x^2 \oplus 0 \cdot x^1 \oplus 1 \cdot x^0 = x^5 \oplus x^4 \oplus x^2 \oplus 1,$$

где $\oplus$ – это сумма по модулю два.

Входной блок данных (128 бит) разбивается на 16 байт $in_0, in_1, \ldots, in_{15}$. Все внутренние преобразования выполняются над двумерными массивами, называемыми *состояниями* (*state*), которые представляются как матрица $S_{r,c}$, где $0 \le r \le 4$, $0 \le c \le 4$. Тогда процесс обработки данных можно представить следующим образом (рис. 3.13).

При реализации *AES* применяют следующие математические операции, определенные над конечными полями (аргументами являются байты):

1. Поразрядное сложение по модулю два ($\oplus$ или *XOR*).

*Пример*: $156 \oplus 53 = 169$.

Полиномиальное представление.

$156 = 1\,0\,0\,1\,1\,1\,0\,0 = x^7 \oplus x^4 \oplus x^3 \oplus x^2$;

$53 = 0\,0\,1\,1\,0\,1\,0\,1 = x^5 \oplus x^4 \oplus x^2 \oplus 1$.

Тогда:

$(x^7 \oplus x^4 \oplus x^3 \oplus x^2) \oplus (x^5 \oplus x^4 \oplus x^2 \oplus 1) = x^7 \oplus x^5 \oplus x^3 \oplus 1$.

Поразрядное сложение удобно выполнять в столбик:

1 0 0 1 1 1 0 0 = 156

0 0 1 1 0 1 0 1 = 53

1 0 1 0 1 0 0 1 = 169

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

*Рис. 3.13*. Последовательность обработки данных

2. Умножение.

В *AES* умножение выполняется по модулю примитивного полинома: $m(x) = x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$ (в векторной форме: 00011011).

*Пример*:

Выполним умножение полиномов без учета модуля:

$(x^6 \oplus x^5 \oplus x^4 \oplus x \oplus 1) \cdot (x^7 \oplus x^5 \oplus x^2 \oplus 1) = x^{13} \oplus x^{11} \oplus x^8 \oplus \oplus x^6 \oplus x^{12} \oplus x^{10} \oplus$
$\oplus x^7 \oplus x^5 \oplus x^{11} \oplus x^9 \oplus x^6 \oplus x^4 \oplus x^8 \oplus x^6 \oplus x^3 \oplus x \oplus x^7 \oplus x^5 \oplus x^2 \oplus 1 = x^{13} \oplus$
$\oplus x^{12} \oplus x^{10} \oplus x^9 \oplus x^6 \oplus x^4 \oplus x^3 \oplus x^2 \oplus x \oplus 1.$

Затем найдем значение результата по модулю $m(x)$.

$x^{13} \oplus x^{12} \oplus x^{10} \oplus x^9 \oplus x^6 \oplus x^4 \oplus x^3 \oplus x^2 \oplus x \oplus 1$ | $x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1$

$x^{13} \oplus x^9 \oplus x^8 \oplus x^6 \oplus x^5$ | $x^5 \oplus x^4 \oplus x^2$

---

$x^{12} \oplus x^{10} \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^3 \oplus x^2 \oplus x \oplus 1$

$x^{12} \oplus x^8 \oplus x^7 \oplus x^5 \oplus x^4$

---

$x^{10} \oplus x^7 \oplus x^3 \oplus x^2 \oplus x \oplus 1$

$x^{10} \oplus x^6 \oplus x^5 \oplus x^3 \oplus x^2$

---

$x^7 \oplus x^6 \oplus x^5 \oplus x \oplus 1$

Получили:

$$(x^6 \oplus x^5 \oplus x^4 \oplus x \oplus 1) \cdot (x^7 \oplus x^5 \oplus x^2 \oplus 1) \bmod (x^8 \oplus x^4 \oplus x^3 \oplus x \oplus 1) = x^7 \oplus x^6 \oplus x^5 \oplus x \oplus 1.$$

Для упрощения вычислений выделяют операцию умножения полинома $a(x)$ на $x$ по модулю $m(x)$: $a(x) \cdot x \mod m(x)$. Эта операция реализуется следующим образом: байт множимого сдвигается влево на один разряд, в крайний правый бит записывается 0. Если выдвинутый бит равен единице, то результат сдвига поразрядно суммируется с байтом $m(x)$ = 00011011. Для умножения $a(x)$ на $x^k$ по mod $m(x)$ данную процедуру повторяют $k$ раз.

Пример: найти $a(x) \cdot b(x) \bmod m(x)$.

$a(x) = x^6 \oplus x^5 \oplus x^4 \oplus x \oplus 1 = 01110011,$

$b(x) = x^7 \oplus x^5 \oplus x^2 \oplus 1 = 10100101.$

Найдем $a(x) \cdot x^k$ по mod $m(x)$ (рис. 3.14).

| | | |
|---|---|---|
| $a(x) \cdot x^0 =$ | | 0 1 1 1 0 0 1 1 |
| $a(x) \cdot x =$ | 0 | 1 1 1 0 0 1 1 0 |
| $a(x) \cdot x =$ | 1 | 1 1 0 0 1 1 0 0<br>0 0 0 1 1 0 1 1<br>1 1 0 1 0 1 1 1 |
| $a(x) \cdot x^3 =$ | 1 | 1 0 1 0 1 1 1 0<br>0 0 0 1 1 0 1 1<br>1 0 1 1 0 1 0 1 |
| $a(x) \cdot x^4 =$ | 1 | 0 1 1 0 1 0 1 0<br>0 0 0 1 1 0 1 1<br>0 1 1 1 0 0 0 1 |
| $a(x) \cdot x^5 =$ | 0 | 1 1 1 0 0 0 1 0 |
| $a(x) \cdot x^6 =$ | 1 | 1 1 0 0 0 1 0 0<br>0 0 0 1 1 0 1 1<br>1 1 0 1 1 1 1 1 |
| $a(x) \cdot x^7 =$ | 1 | 1 0 1 1 1 1 1 0<br>0 0 0 1 1 0 1 1<br>1 0 1 0 0 1 0 1 |

*Рис. 3.14.* Результат умножения $a(x) \cdot x^k$ по mod $m(x)$

Затем строки таблицы, соответствующие ненулевым коэффициентам полинома $b(x)$ (рис. 3.15). Таким образом, для умножения данных используется только операции сдвига и сложения по модулю два, которые выполняются очень быстро.

| | |
|---|---|
| 1 0 1 0 0 1 0 1 | $a(x) \cdot x^7$ |
| 1 1 1 0 0 0 1 0 | $a(x) \cdot x^5$ |
| 1 1 0 1 0 1 1 1 | $a(x) \cdot x^2$ |
| 0 1 1 1 0 0 1 1 | $a(x) \cdot x^0$ |
| 1 1 1 0 0 0 1 1 | $x^7 \oplus x^6 \oplus x^5 \oplus x \oplus 1$ |

*Рис. 3.15*. Результат умножения $a(x) \cdot b(x)$ по mod $m(x)$

3. В *AES* также используется умножение полубайтов (коды длиной 4 бита, представленные в виде полинома), которое выполняется по модулю примитивного полинома: $m_2(x) = x^4 \oplus 1$. Однако в этом случае коэффициенты полиномов представляют собой байты.

Пусть $d(x) = a(x) \cdot b(x)$ по mod $m_2(x) = d_3 x^3 \oplus d_2 x^2 \oplus d_1 x^1 \oplus d_0$.

Тогда в матричной форме операция умножения полиномов будет выглядеть следующим образом:

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

Применение данных процедур позволяет значительно повысить скорость шифрования и снизить требования к аппаратной реализации.

# 4. АСИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ

## 4.1. Принципы ассиметричной криптографии

Первые криптографические системы с открытым ключом появились в конце 1970-х гг. От классических алгоритмов они отличаются тем, что для шифрования данных используется один ключ (открытый), а для расшифрования – другой (секретный). Данные, зашифрованные открытым ключом, можно расшифровать только секретным ключом. Следовательно, открытый ключ может распространяться через обычные коммуникационные сети и другие открытые каналы. Таким образом, устраняется главный недостаток стандартных криптографических алгоритмов: необходимость использовать специальные каналы связи для распределения ключей. Разумеется, секретный ключ не может быть вычислен из открытого ключа.

Концепция криптографии с открытыми ключами была выдвинута Уитфилдом Диффи, Мартином Хеллманом и независимо Ральфом Мерклом. Они предложили использовать ключи парами – один ключ для шифрования, а другой для дешифрования, причем так, чтобы невозможно было получить один ключ из другого.

С 1976 г. было предложено множество криптографических алгоритмов с открытыми ключами. Проблема заключается в том, что многие из предложенных алгоритмов небезопасны. Те алгоритмы, которые являются безопасными, зачастую используют слишком большой ключ или размер полученного с их помощью шифротекста намного превышает размер исходного текста. Таким образом, только небольшое количество алгоритмов являются и безопасными, и пригодными для практического использования.

Криптографические системы с открытым ключом не сразу получили признание. Это связано с тем, что в тот момент, когда была открыта криптография с открытыми ключами, Агентство Национальной Безопасности США предложило удобную криптографическую систему, разработанную фирмой *IBM*, в качестве федерального стандарта шифрования данных (*DES*). Данная система критиковалась разработчиками ассиметричной системы из-за недостаточной длины ключа, но большинство производителей поддержали *DES*. По этой причине критика была воспринята как попытка помешать введению стандарта ради продвижения собственной разработки.

Алгоритмы с открытым ключом (ассиметричные алгоритмы) разработаны таким образом, что ключ, используемый для шифрова-

ния, отличается от ключа дешифрования, и ключ дешифрования не может быть получен из ключа шифрования. Название «с открытым ключом» было дано алгоритму по причине того, что ключ шифрования может быть открытым, т. е. находится в свободном доступе. Что касается ключа дешифрования, то он известен только лицам, которые могут расшифровать сообщение. В данных системах ключ шифрования обычно называется открытым, а ключ дешифрования – закрытым или секретным.

Шифрование с открытым ключом обозначается следующим образом:

$$E_{K_1}(M) = C,$$
$$D_{K_2}(E_{K_1}(M)) = M,$$

где $M$ – исходное сообщение; $C$ – шифротекст; $E$ – функция шифрования; $K_1$, $K_2$ – открытый и закрытый ключи.

В криптографии с открытым ключом центральным является понятие однонаправленной функции. Однонаправленная функция – это функция, которая относительно легко вычисляется, однако инвертируется с большим трудом, т. е., зная $x$, просто рассчитать $f(x)$, но по известному значению $f(x)$ сложно вычислить $x$. В данном случае понятие сложно означает, что для вычисления $x$ по $f(x)$ могут потребоваться миллионы лет, даже если для решения этой задачи будут использоваться все компьютеры мира. В качестве примера однонаправленной функции можно рассмотреть разбитую тарелку – ее легко разбить на мелкие кусочки, но сложно снова сложить тарелку их этих кусочков.

Математически строгого доказательства существования однонаправленных функций на данный момент не приведено. Также не существует реальных свидетельств возможности их построения. Но, несмотря на это, существуют функции, которые выглядят как однонаправленные – их можно рассчитать, но нельзя инвертировать.

Однонаправленные функции не используются непосредственно для шифрования. Для целей шифрования предназначены однонаправленные функции с люком – особый тип однонаправленных функций. Они также легко вычисляются в прямом направлении и сложно в обратном, но если известна определенная секретная функция, то можно рассчитать обратную функцию. Математически это можно представить следующим образом: легко вычислить $f(x)$ по заданному $x$, но трудно по известному $f(x)$ вычислить $x$, однако существует секретная информация $y$, позволяющая при знании $f(x)$ и $y$ легко вычислить $x$.

## 4.2. Алгоритм *RSA*

Одним из наиболее известных алгоритмов шифрования с открытым ключом является алгоритм Ривеста–Шамира–Адлемана (*RSA*), который до сих пор используется как для шифрования, так и для цифровой подписи.

Безопасность *RSA* основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших (более 100–200 разрядов) простых чисел. Предполагается, что восстановление открытого текста по шифротексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

Для генерации двух ключей используются два больших случайных простых числа, $p$ и $q$. Для максимальной безопасности рекомендуется выбирать $p$ и $q$ равной длины. Затем рассчитывается произведение $n = p \cdot q$. Вычисляется функция Эйлера $\varphi(n) = (p - 1) \cdot (q - 1)$. После этого случайным образом выбирается ключ шифрования $e$ (открытый ключ), такой, что $e$ и $\varphi(n)$ являются взаимно простыми числами. Для вычисления ключа дешифрования $d$ используется расширенный алгоритм Евклида. Ключ дешифрования $d$ (закрытый ключ) должен удовлетворять следующему требованию: $e \cdot d = 1 \bmod \varphi(n)$. Другими словами, $d$ является взаимообратным к $e$ по модулю $\varphi(n)$. Два простых числа, $p$ и $q$, больше не нужны. Они должны быть отброшены, но не должны быть раскрыты.

Для шифрования сообщения $m$ оно сначала разбивается на цифровые блоки, меньшие $n$. Если нужно зашифровать фиксированное число блоков, их можно дополнить несколькими нулями слева, чтобы гарантировать, что блоки всегда будут меньше $n$. Шифрованное сообщение $c$ будет состоять из блоков $c_i$ той же длины. Формула шифрования выглядит так: $c_i = m_i^e \bmod n$. Для расшифровки сообщения используется аналогичная операция: $m_i = c_i^d \bmod n$.

Приведем пример использования *RSA* для шифрования сообщений. Для простоты будем использовать малые числа (на практике используются числа большой разрядности).

Выберем два случайных простых числа, например $p = 3$, $q = 11$. Затем вычисляем произведение $n = p \cdot q = 33$. Вычисляем функцию Эйлера $\varphi(n) = (3 - 1) \cdot (11 - 1) = 20$. После этого случайным образом выбирается ключ шифрования $e$ (открытый ключ), такой, что $(e, \varphi(n)) = 1$. Пусть $e = 3$. Находим закрытый ключ $d$, удовлетворяющий соотношению: $e \cdot d = 1 \bmod \varphi(n)$. Для этого можно использовать расширение алгоритма

Евклида или решить сравнение $3 \cdot d = 1 \bmod 20$. Решая сравнение, получим $d = 3^{\varphi(20)-1} \bmod 20 = 3^7 \bmod 20 = 7$.

Проверяем: $3 \cdot 7 \bmod 20 = 1$.

Зашифруем слово «*CRYPTO*». Пусть букве *A* соответствует число 0, букве *B* – 1, *C* – 2 и т. д. Тогда сообщение «*CRYPTO*» можно представить в виде последовательности чисел {3, 17, 24, 15, 19, 14}. Зашифруем сообщение, используя открытый ключ $e = 3$:

$c_1 = 3^3 \bmod 33 = 27$;

$c_2 = 17^3 \bmod 33 = 29$;

$c_3 = 24^3 \bmod 33 = 30$;

$c_4 = 15^3 \bmod 33 = 9$;

$c_5 = 19^3 \bmod 33 = 28$;

$c_6 = 14^3 \bmod 33 = 5$.

Получили зашифрованное сообщение {27, 29, 30, 9, 28, 5}. Для расшифровки полученного сообщения необходимо знать секретный ключ $d = 7$. Тогда расшифровка будет проходить аналогично шифрованию.

$m_1 = 27^7 \bmod 33 = 3$;

$m_2 = 29^7 \bmod 33 = 17$;

$m_3 = 30^7 \bmod 33 = 24$;

$m_4 = 9^7 \bmod 33 = 15$;

$m_5 = 28^7 \bmod 33 = 19$;

$m_6 = 5^7 \bmod 33 = 14$.

Таким образом, в результате расшифровки получено исходное сообщение «*CRYPTO*» ({3, 17, 24, 15, 19, 14}).

Криптостойкость алгоритма *RSA* основывается на предположении, что исключительно трудно определить секретный ключ по открытому, поскольку для этого необходимо решить задачу о существовании делителей целого числа, т. е. найти множители параметра *n*. Данная задача не имеет эффективного (полиномиального) решения. Вопрос существования эффективного алгоритма решения данной задачи является до настоящего времени открытым. Традиционные же методы для чисел, состоящих из 200 цифр (именно такие числа рекомендуется использовать), требуют выполнения огромного числа операций (порядка $10^{23}$).

Доказано, что некоторые варианты *RSA* так же сложны, как и разложение на множители, а раскрытие даже нескольких блоков информации по зашифрованному данным алгоритмом шифротексту не легче, чем дешифрование всего сообщения.

Криптоаналитик может перебирать все возможные *d*, пока не подберет правильное значение. Такое вскрытие грубой силой даже менее эффективно, чем попытка разложить *n* на множители.

Алгоритмы с открытыми ключами, как правило, используются не для шифрования сообщений (для этих целей используют более быстрые и надежные симметричные алгоритмы), а для шифрования ключей. Это связано с тем, что алгоритмы с открытыми ключами работают значительно медленнее симметричных алгоритмов. Кроме того, криптографические системы с открытыми ключами уязвимы по отношению к вскрытию с выбранным открытым текстом. Поэтому в большинстве случаев криптография с открытыми ключами используется для засекречивания и распространения сеансовых ключей, которые используются симметричными алгоритмами для закрытия потока сообщений. Такие криптографические системы называются смешанными или гибридными.

Использование криптографии с открытыми ключами для распределения ключей решает очень важную проблему. Для иллюстрации рассмотрим следующий пример.

• Пользователь *A* посылает пользователю *B* свой открытый ключ.

• Пользователь *B* создает случайный сеансовый ключ, шифрует его с помощью открытого ключа пользователя *A* и передает его пользователю *A*.

• Пользователь *A* расшифровывает сообщение пользователя *B*, используя свой закрытый ключ для получения сеансового ключа.

Оба пользователя шифруют свои сообщения с помощью одного сеансового ключа, который уничтожается по окончании сеанса связи.

Это значительно уменьшает риск компрометации (раскрытие ключа не криптологическим способом) сеансового ключа. Закрытый ключ также чувствителен к компрометации, но риск значительно меньше, так как в течение сеанса этот ключ используется только один раз для шифрования сеансового ключа.

В асимметричных криптографических системах при шифровании и дешифровании используются различные ключи, причем ключ шифрования открыт и доступен каждому, а ключ дешифрования известен только ограниченному кругу лиц. Безопасность данного алгоритма полностью основана на ключах, а сам алгоритм не является секретным. Эффективность данной системы основана на том, что ключ дешифрования не может быть получен по ключу шифрования.

Широкому распространению асимметричных криптографических систем препятствует то, что, во-первых, скорости их работы примерно в 1000 раз ниже скоростей работы симметричных алгорит-

мов. Несмотря на то, что с развитием вычислительной техники производительность алгоритмов постоянно растет, их распространению мешает то, что с каждым годом значительно увеличиваются объемы информации, подлежащей шифрованию. Во-вторых, данные криптографические системы чувствительны по отношению к вскрытию с выбранным открытым текстом.

Наиболее оптимальным вариантом является использование смешанных криптографических систем, где асимметричный алгоритм используется для шифрования ключей симметричного алгоритма. Гибридные криптографические системы сочетают в себе достоинства обоих алгоритмов и устраняют недостатки, присущие каждому из них.

Различные алгоритмы предоставляют различные степени безопасности. Конечно, любой алгоритм теоретически и зачастую практически можно вскрыть. Но если стоимость взлома алгоритма выше, чем стоимость зашифрованных данных, скорее всего, данные находятся в безопасности. Также если время взлома алгоритма больше времени, в течение которого зашифрованные данные должны находиться в секрете, вероятнее всего, алгоритм обеспечивает требуемый уровень безопасности. Сопоставляя затраты на шифрование данных и эффект от того, что информация не была дешифрована, можно подобрать оптимальный вариант шифрования.

# 5. ЭЛЕКТРОННАЯ ЦИФРОВАЯ ПОДПИСЬ

## 5.1. Понятие электронной цифровой подписи

Электронная цифровая подпись (ЭЦП) – реквизит электронного документа, предназначенный для удостоверения источника данных и защиты данного электронного документа от подделки.

Схема электронной подписи обычно включает в себя:

– алгоритм генерации ключевых пар пользователя;

– функцию вычисления подписи;

– функцию проверки подписи.

При разработке механизма цифровой подписи необходимо решить следующие задачи:

– формирование подписи таким образом, чтобы ее невозможно было подделать;

– обеспечение возможности проверки того, что подпись действительно принадлежит указанному субъекту;

– предотвращение отказа субъекта от своей подписи.

Функция вычисления подписи на основе документа и секретного ключа пользователя вычисляет собственно подпись. В зависимости от алгоритма функция вычисления подписи может быть детерминированной или вероятностной. Детерминированные функции всегда вычисляют одинаковую подпись по одинаковым входным данным. Вероятностные функции вносят в подпись элемент случайности, что усиливает криптостойкость алгоритмов ЭЦП. Однако для вероятностных схем необходим надежный источник случайности (либо аппаратный генератор шума, либо криптографически надежный генератор псевдослучайных бит), что усложняет реализацию [7].

В настоящее время детерминированые схемы практически не используются. Даже в изначально детерминированные алгоритмы сейчас внесены модификации, превращающие их в вероятностные алгоритмы (так, в алгоритм подписи *RSA* вторая версия стандарта *PKCS#1* добавила предварительное преобразование данных, включающее в себя, среди прочего, зашумление).

Функция проверки подписи проверяет, соответствует ли данная подпись данному документу и открытому ключу пользователя. Открытый ключ пользователя доступен всем, так что любой может проверить подпись под данным документом.

Поскольку подписываемые документы переменной (и достаточно большой) длины, в схемах ЭЦП зачастую подпись ставится не на

сам документ, а на его хэш-образ. Для вычисления хэш-образа используются криптографические хэш-функции, что гарантирует выявление изменений документа при проверке подписи. Хэш-функции не являются частью алгоритма ЭЦП, поэтому в схеме может быть использована любая надежная хэш-функция.

Алгоритмы ЭЦП делятся на два больших класса: обычные цифровые подписи и цифровые подписи с восстановлением документа. Обычные цифровые подписи необходимо пристыковывать к подписываемому документу. К этому классу относятся, например, алгоритмы, основанные на эллиптических кривых (*ECDSA*, ГОСТ Р 34.10–2001). Цифровые подписи с восстановлением документа содержат в себе подписываемый документ: в процессе проверки подписи автоматически вычисляется и тело документа. К этому классу относится один из самых популярных алгоритмов – *RSA*.

Следует различать электронную цифровую подпись и код аутентичности сообщения, несмотря на схожесть решаемых задач (обеспечение целостности документа и невозможность отказа от авторства). Алгоритмы ЭЦП относятся к классу асимметричных алгоритмов, в то время как коды аутентичности вычисляются по симметричным схемам [1].

## 5.2. Классическая схема создания цифровой подписи

Классическая схема создания цифровой подписи предполагает выполнение следующих действий (рис. 5.1).

1. Вычислить хеш-образ $m$ исходного сообщения $M$ при помощи хеш-функции $h$.

2. Вычислить цифровую подпись $S$ по хеш-образу сообщения с использованием секретного ключа $K_c$.

3. Сформировать новое сообщение ($M$, $S$), состоящее из исходного сообщения и добавленной к нему цифровой подписи.

Получив подписанное сообщение ($M'$, $S$), получатель должен выполнить следующие действия (принятое сообщение обозначено как $M'$ по причине того, что оно могло быть преднамеренно либо случайно искажено в процессе передачи по каналу связи и может не совпадать с отправленным).

1. Вычислить хеш-образ $m'$ сообщения $M'$ при помощи хеш-функции $h$.

2. С использованием открытого ключа проверки подписи ($K_o$) извлечь хеш-образ $m$ сообщения из цифровой подписи $S$.

3. Сравнить вычисленное значение *m*’ с извлеченным из цифровой подписи значением хеш-образа *m*. Если хеш-образы совпадают, то подпись признается подлинной.


*Рис. 5.1.* Классическая схема создания цифровой подписи

Первой и наиболее известной во всем мире конкретной системой электронной цифровой подписи стала система *RSA*, рассмотренная в предыдущем разделе. При ее использовании сначала необходимо вычислить пару ключей (секретный ключ и открытый ключ). Для этого отправитель сообщения (документа) выбирает два больших простых числа $p$ и $q$, а затем находит их произведение $n = p \cdot q$, находит значение функции Эйлера от данного произведения

$$\varphi(n) = (p - 1) \cdot (q - 1).$$

Далее отправитель вычисляет значение $K_o$: $K_o < \varphi(n)$, НОД $(K_o, \varphi(n)) = 1$ и значение $K_c$ из условий $K_c < \varphi(n)$, $K_o \cdot K_c = 1 \bmod \varphi(n)$.

Пара значений ($K_o$, $n$) является открытым ключом. Эту пару чисел автор передает партнерам по переписке для проверки его цифровых подписей. Значение $K_c$ сохраняется автором как секретный ключ подписи.

Допустим, что отправитель хочет подписать сообщение $M$ перед его отправкой. Сначала сообщение $M$ сжимают с помощью хеш–функции $h$ в целое число $m$: $m = h(M)$.

Затем на основе хеш-образа $m$ и секретного значения $K_c$ вычисляют цифровую подпись $S$ под электронным документом $M$:

$$S = m^{K_c} \bmod n.$$

Для возведения в степень можно воспользоваться алгоритмом быстрого возведения в степень по модулю, позволяющим вычислить $x = a^z \bmod n$ (данный алгоритм рассмотрен в главе 2).

Пара ($M$, $S$) передается получателю как электронный документ $M$, подписанный цифровой подписью $S$, причем подпись $S$ сформирована обладателем секретного ключа $K_c$.

После приема пары ($M'$, $S$) получатель вычисляет хеш-образ сообщения $M'$ двумя различными способами. Прежде всего, он восстанавливает хеш-образ $m$, применяя криптографическое преобразование подписи $S$ с использованием открытого ключа $K_o$:

$$m = S^{K_o} \bmod r.$$

Кроме того, он находит результат хеширования $m'$ принятого сообщения $M'$ с помощью такой же хеш-функции $h$:

$$m' = h(M).$$

Если вычисленные значения совпадают, т. е.

$$S^{K_o} \bmod r = h(M'),$$

то получатель признает пару ($M'$, $S$) подлинной. Фальсификация сообщения при его передаче по каналу связи возможна только при получении злоумышленником секретного ключа $K_c$ либо за счет проведения успешной атаки против хеш-функции. При использовании достаточно больших значений $p$ и $q$ определение секретного значения $K_c$ по открытому ключу ($K_o$, $n$) является чрезвычайно трудной задачей, соответствующей по сложности разложению модуля $n$ на множители. Используемые в реальных приложениях хеш-функции обладают характеристиками, делающими атаку против цифровой подписи практически не осуществимой. Пример – хеш-функция SHA–1, принятая в США в качестве стандарта в 1995 г., формирующая 160-битовый хеш-образ при обработке сообщения блоками по 512 бит. Вероятность коллизии при использовании данной хеш-функции составляет $2^{-160}$ или приблизительно $6{,}84 \cdot 10^{-49}$.

# ЛИТЕРАТУРА

1. О некоторых вопросах защиты информации : постановление Совета Министров Респ. Беларусь от 26 мая 2009 г. № 675 / Нац. реестр правовых актов Респ. Беларусь. – 2009. – № 136. – С. 63–76.

2. Бузов, Г. А. Защита от утечки информации по техническим каналам : учеб. пособие / Г. А. Бузов. – М. : Горячая линия–Телеком, 2005.

3. Ярмолик, В. Н. Криптография, стеганография и охрана авторского права / В. Н. Ярмолик, С. С. Портянко, С. В. Ярмолик. – Минск, 2007. – 242 с.

4. Мельников, В. П. Информационная безопасность и защита информации : учеб. пособие для вузов / В. П. Мельников, С. А. Клейменов, А. М. Петраков ; под ред. С. А. Клейменова. – М. : Академия, 2009. – 331 с.

5. Шаньгин, В. Ф. Защита компьютерной информации. Эффективные методы и средства : учеб. пособие для вузов / В. Ф. Шаньгин. – М. : ДМК, 2008. – 542 с.

6. Харин, Ю. С. Компьютерный практикум по математическим основам криптографии : учеб. пособие / Ю. С. Харин, С. В. Агиевич. – Минск : БГУ, 2001. – 190 с.

7. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы и исходные тексты на языке С / Б. Шнайер. – М. : Триумф, 2002. – 816 с.

8. Хамидуллин, Р. Р. Методы и средства защиты компьютерной информации : учеб. пособие / Р. Р. Хамидуллин, И. А. Бригаднов, А. В. Морозов. – СПб. : СЗТУ, 2005. – 178 с.

9. СТБ 34.101.31–2011. Информационные технологии и безопасность. Защита информации. Криптографические алгоритмы шифрования и контроля целостности. – Минск : Гостстандарт, 2011.

10. Защита информации в системах мобильной связи : учеб. пособие для вузов / А. А. Чекалин [и др.]. – М. : Горячая линия–Телеком, 2005.

11. Ищейнов, В. Я. Защита конфиденциальной информации : учеб. пособие для вузов / В. Я. Ищейнов, М. В. Мецатунян. – М. : Форум, 2009. – 254 с.

12. Технические средства и методы защиты информации : учеб. пособие для вузов / А. П. Зайцев [и др.] ; под ред. А. П. Зайцева, А. А. Шелупанова. – М. : Горячая линия–Телеком, 2009.

13. Простое число. – Режим доступа: http://ru.wikipedia.org/wiki/Простое_число. – Дата доступа: 1.06.2014.

# СОДЕРЖАНИЕ

Учебное электронное издание комбинированного распространения

Учебное издание

**Мурашко** Игорь Александрович

# ЗАЩИТА КОМПЬЮТЕРНОЙ ИНФОРМАЦИИ

## Пособие

**Электронный аналог печатного издания**

Редактор *Н. В. Гладкова*
Компьютерная верстка *Е. Б. Ящук*

Подписано в печать 23.04.15.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 3,02. Уч.-изд. л. 2,89.
Изд. № 121.
http://www.gstu.by

Издатель и полиграфическое исполнение:
Издательский центр
Учреждения образования «Гомельский государственный
технический университет имени П. О. Сухого».
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за №1/273 от 04.04.2014 г.
246746, г. Гомель, пр. Октября, 48