Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

В. Н. Ярмолик, А. П. Занкович, С. С. Портянко

ЭЛЕМЕНТЫ ТЕОРИИ ИНФОРМАЦИИ

Практикум
для студентов специальности
«Программное обеспечение информационных технологий»
дневной и дистанционной форм обучения

Минск 2007

УДК 621.391.1(075)
ББК 32.811 я 7
Я 75

Р е ц е н з е н т
доцент кафедры ЭВМ БГУИР,
кандидат технических наук В. В. Ракуш

Ярмолик, В. Н.

Я 75 Элементы теории информации : практикум для студ. спец. «Программное обеспечение информационных технологий» дневн. и дист. форм обуч. / В. Н. Ярмолик, А. П. Занкович, С. С. Портянко. – Минск : БГУИР, 2007. – 39 с. : ил.

ISBN 978-985-488-108-9

В практикуме рассматриваются практические вопросы криптографического преобразования информации в компьютерных системах. Рассмотрены наиболее актуальные вопросы предметной области – симметричные и асимметричные алгоритмы; блочные, роторные и потоковые шифры, алгоритмы электронной цифровой подписи и хеширования. По каждой теме приводятся теоретические сведения, практические способы реализации алгоритмов и набор заданий разной степени сложности.

УДК 621.391.1(075)
ББК 32.811 я 7

ISBN 978-985-488-108-9

СОДЕРЖАНИЕ

1. КРИПТОАНАЛИЗ МЕТОДОВ ПРОСТОЙ ПОДСТАНОВКИ

Простейшие шифры подстановки (substitution) реализуют замену каждого символа исходного текста на один из символов алфавита шифротекста. В общем случае подстановочный шифр описывается таблицей подстановки, состоящей из двух строк и *n* столбцов. Количество столбцов таблицы подстановки соответствует количеству различных символов в алфавите исходного текста. Верхняя строка таблицы подстановки содержит все возможные символы исходного текста, а нижняя – соответствующие им символы шифротекста.

Моноалфавитные шифры характеризуются однозначным соответствием символов исходного текста и символов шифротекста. В случае когда алфавиты исходного текста и шифротекста состоят из одного и того же множества символов, алфавит шифротекста представляет собой простую перестановку лексикографического порядка символов в алфавите исходного текста. При выполнении шифрования каждый символ исходного текста заменяется соответствующим ему символом шифротекста.

Таблица подстановки, описывающая моноалфавитный шифр, преобразующий строчные буквы русского алфавита, будет состоять из 33 столбцов, что соответствует количеству букв в алфавите. Рассмотрим такую таблицу на следующем примере:

а	**б**	**в**	**г**	**д**	**е**	**ё**	**ж**	**з**	**и**	**й**	**к**	**л**	**м**	**н**	**о**	**п**	**р**	**с**	**т**	**у**	**ф**	**х**	**ц**	**ч**	**ш**	**щ**	**ъ**	**ы**	**ь**	**э**	**ю**	**я**
е	л	ц	н	й	и	в	а	ш	у	ь	т	с	я	ф	ы	э	ж	о	р	к	ч	м	ъ	х	п	м	б	г	ё	з	щ	д

В соответствии с приведённой таблицей шифрование строки «`знание - сила`» будет выполнено следующим образом:

«з» → «ш»;

«н» → «ф»;

…

«а» → «е».

В результате шифрования будет получен шифротекст «`шфефуи - оусе`».

Таблица подстановки, описывающая ключ моноалфавитного шифра, преобразующего ASCII-коды (однобайтовые значения), будет состоять из 256 столбцов. Таблица подстановки для шифра, осуществляющего подстановку 32-битных значений, будет состоять из 2^{32} столбцов, что уже вызовет сложности её хранения и передачи. По этой причине на практике вместо таблиц подстановок используются функции подстановки, аналитически описывающие соответствие между порядковыми номерами символов исходного текста в алфавите исходного текста и порядковыми номерами символов шифротекста в алфавите шифротекста.

Предположим, алфавит исходного текста *M* состоит из *n* символов $M=\{a_0, a_1, ..., a_n\}$, тогда алфавит шифротекста *C* будет представлять собой

n-символьный алфавит $C=\{f(a_0), f(a_1), ..., f(a_n)\}$, где функция f, выполняющая отображение $M \rightarrow C$, и будет являться функцией подстановки. В общем случае функция подстановки любого моноалфавитного шифра может быть задана в виде полинома степени t:

$$E_k(a) = (k_0 + k_1 \cdot a + k_2 \cdot a^2 + ... + k_{t-1} \cdot a^{t-1} + k_t \cdot a^t) \bmod n. \quad (1)$$

Примером простейшего моноалфавитного шифра является шифр *Цезаря*. Строки таблицы подстановки для шифра Цезаря представляют собой сдвинутые друг относительно друга на l позиций алфавиты исходного текста. Сам Цезарь использовал для шифрования величину сдвига $l = 3$. Функция подстановки для шифра Цезаря будет задаваться полиномом нулевой степени:

$$E_k(a) = (a + k) \bmod n. \quad (2)$$

Поскольку для моноалфавитных шифров каждый символ исходного текста при шифровании может заменяться одним единственным символом шифротекста, для криптоанализа данных шифров возможно применение анализа частот встречаемости символов в шифротексте. Данная атака основана на том факте, что в естественных языках частоты встречаемости различных букв могут существенно отличаться. Так, в английском языке наиболее часто встречаемой является буква «e», а наименее встречаемой – буква «z». Зная типичную частоту встречаемости каждого из символов алфавита исходного текста, можно воссоздать использованную при шифровании таблицу постановки, ставя каждому из символов исходного текста в соответствие символ шифротекста, частота встречаемости которого в зашифрованном тексте является наиболее близкой к типичной частоте встречаемости символа алфавита исходного текста.

Подстановочные шифры, называемые *полиалфавитными*, используют более чем одну таблицу подстановки. Использование нескольких таблиц (алфавитов) обеспечивает возможность нескольких вариантов подстановки символов исходного сообщения, что повышает криптостойкость шифра.

Классическим полиалфавитным шифром является шифр *Виженера* (Vigenere Cipher). Так же как и в случае шифра Цезаря, данный шифр может быть задан таблицами подстановки, состоящими из n столбцов, где n – размер алфавита исходного текста. Количество таблиц подстановки для случая шифра Виженера будет равняться m, где m – длина ключевого слова (период ключа). Ключевое слово задаёт количество символов, на которое смещены относительно исходного алфавиты шифротекста в каждой из m таблиц подстановок.

Рассмотрим работу шифра Виженера на простом примере. Пусть дано ключевое слово «MOUSE». Тогда правила подстановки будут задаваться 5-ю таблицами, которые для компактности можно объединить в одну.

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**
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
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
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
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
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

Пусть необходимо зашифровать текст «CRYPTOGRAPHY AND DATA SECURITY». Сперва запишем символы ключевого слова под символами исходного текста. Поскольку ключ в методе Виженера – периодический, повторим ключевое слово столько раз, сколько нам потребуется, чтобы закрыть весь исходный текст.

```
CRYPTOGRAPHY AND DATA SECURITY
MOUSEMOUSEMOUSEMOUSEMOUSEMOUSE
```

Каждый из символов ключа указывает, которую из таблиц подстановки нам необходимо использовать для подстановки рассматриваемого символа исходного текста. Символ ключа «M» указывает, что соответствующий символ шифротекста необходимо выбирать из таблицы подстановки, алфавит шифротекста в которой сдвинут относительно алфавита исходного текста на Pos(«M») – Pos(«A») = 12 позиций. Таким образом, первому символу исходного текста будет соответствовать символ шифротекста «C». Проведя аналогичную процедуру для всех символов исходного текста, получим искомый шифротекст:

«OFSHXAULSTTM SRP XSXM MWGGFCLC».

Как видно из примера, в результате шифрования по методу Виженера символы исходного текста «P» и «H» были преобразованы в один и тот же подстановочный элемент «T». Таким образом, отсутствует однозначное соответствие между символами исходного и зашифрованного текстов, что делает применение атаки на шифротекст путём частотного анализа «в лоб» невозможным.

Поскольку ключ шифратора Виженера является периодическим, зашифрованный текст можно представить как m текстов, зашифрованных по методу Цезаря. В рассмотренном примере для текста «CRYPTOGRAPHY AND DATA SECURITY» символы с позициями 1, 6, ..., 26 шифровались по методу Цезаря с ключом $k = 12$; символы с позициями 2, 7, ..., 27 – ключом $k = 14$; с позициями 3, 8, ..., 28 – ключом $k = 20$; с позициями 4, 9, ..., 29 – ключом $k = 18$; с позициями 5, 10, ..., 30 – ключом $k = 4$.

```
                  CRYPTOGRAPHY AND DATA SECURITY
k = 12 («M»):     C    O    H    D    A    U
k = 14 («O»):      R    G    Y    -    -    R
k = 20 («U»):       Y    R    -    D    S    I
k = 18 («S»):        P    A    A    A    E    T
k = 4  («E»):         T    P    N    T    C    Y
```

Таким образом, зная длину ключевого слова шифра Виженера (период ключа), можно произвести взлом шифротекста, выполнив анализ частот встречаемости символов в отдельности для каждого из *m* компонентов шифротекста.

Одним из методов определения длины ключевого слова, использованного при шифровании текста по методу Виженера, является метод *Касиски* (Kasiski).

Данный метод основан на предположении, что наличие повторяющихся *l*-грамм (*l*-символьных последовательностей) в зашифрованном тексте будет в большинстве случаев обусловлено наличием соответствующих повторяющихся *l*-грамм в исходном тексте. Предполагается, что случайное появление в шифротексте повторяющихся *l*-грамм маловероятно.

Одинаковым *l*-граммам, присутствующим в исходном тексте, будут соответствовать одинаковые *l*-граммы, расположенные на тех же позициях в шифротексте, только в том случае, если при шифровании они будут преобразованы с использованием тех же *l* символов ключа. Это условие будет выполняться для всех повторяющихся *l*-грамм, расположенных друг от друга на расстояниях, кратных длине ключевого слова шифра.

Тест Касиски состоит из следующих шагов:

1. Анализируется шифротекст на предмет присутствия в нём повторяющихся *l*-грамм.

2. Для каждой из встретившихся в шифротексте более одного раза *l*-граммы вычисляются расстояния между её соседними вхождениями.

3. Вычисляется наибольший общий делитель полученного на предыдущем шаге множества расстояний с учётом того, что среди найденных повторений *l*-грамм могут в незначительном количестве присутствовать случайные повторения. Полученное значение и будет являться длиной ключевого слова.

Эксперименты показывают, что данный метод является достаточно эффективным при анализе зашифрованных текстов на русском и английском языках в случае, если в тексте присутствуют повторяющиеся *l*-граммы длиной в три и более символов.

Задания

1. Реализовать программное средство, осуществляющее шифрование и дешифрование текстового файла, содержащего текст на заданном языке.

2. Реализовать программное средство, осуществляющее криптоанализ зашифрованного по методу Виженера текста. Для криптоанализа использовать тест Касиски.

3. Провести экспериментальное исследование зависимости вероятности успешного проведения атаки по методу Касиски от длины шифротекста.

4. Провести экспериментальное исследование зависимости вероятности успешного проведения атаки по методу Касиски от длины использованного при шифровании ключевого слова.

ПРИЛОЖЕНИЕ 1. Частотность букв английского языка.

A 0.08167	H 0.06094	O 0.07507	V 0.00978
B 0.01492	I 0.06966	P 0.01929	W 0.0236
C 0.02782	J 0.00153	Q 0.00095	X 0.0015
D 0.04253	K 0.00772	R 0.05987	Y 0.01974
E 0.12702	L 0.04025	S 0.06327	Z 0.00074
F 0.0228	M 0.02406	T 0.09056	
G 0.02015	N 0.06749	U 0.02758	

ПРИЛОЖЕНИЕ 2. Частотность букв русского языка.

А 0.07821	Ж 0.01082	Н 0.0685	Ф 0.00132	Ы 0.01854
Б 0.01732	З 0.01647	О 0.11394	Х 0.00833	Ь 0.02106
В 0.04491	И 0.06777	П 0.02754	Ц 0.00333	Э 0.0031
Г 0.01698	Й 0.01041	Р 0.04234	Ч 0.01645	Ю 0.00544
Д 0.03103	К 0.03215	С 0.05382	Ш 0.00775	Я 0.01979
Е 0.08567	Л 0.04813	Т 0.06443	Щ 0.00331	
Ё 0.0007	М 0.03139	У 0.02882	Ъ 0.00023	

2. ПОТОКОВЫЕ КРИПТОСИСТЕМЫ

Основная идея потоковых криптосистем заключается в шифровании исходного текста M с помощью криптографического ключа K, длина которого равна длине текста. Каждый бит шифротекста C_i является функцией соответствующих битов исходного текста и ключевого потока:

$$C_i = E_{Ki}(M_i) = M_i \oplus K_i, \qquad M_i, K_i, C_i \in \{0,1\}. \tag{3}$$

При дешифровании выполняется обратное преобразование D_{Ki}:

$$D_{Ki}(C_i) = C_i \oplus K_i = (M_i \oplus K_i) \oplus K_i = M_i. \tag{4}$$

Символом «⊕» обозначена операция сложения «ИСКЛЮЧАЮЩЕЕ-ИЛИ». Благодаря линейным свойствам этой операции при шифровании и дешифровании используется одинаковый ключевой поток K. Очевидно, что в этом случае длина K должна быть равна длине передаваемого сообщения. Однако обмен ключами большого размера зачастую невозможен. Поэтому на практике для формирования ключевого потока используют генераторы псевдослучайной последовательности (рис. 1). Начальные параметры I генераторов на стороне отправителя и получателя должны совпадать, они являются секретным ключом алгоритма. Псевдослучайная последовательность каждого генератора обладает определенным периодом, после которого значения повторяются. Поэтому необходимо выбирать такие генераторы ключа, чтобы этот период превышал длину шифруемой информации.

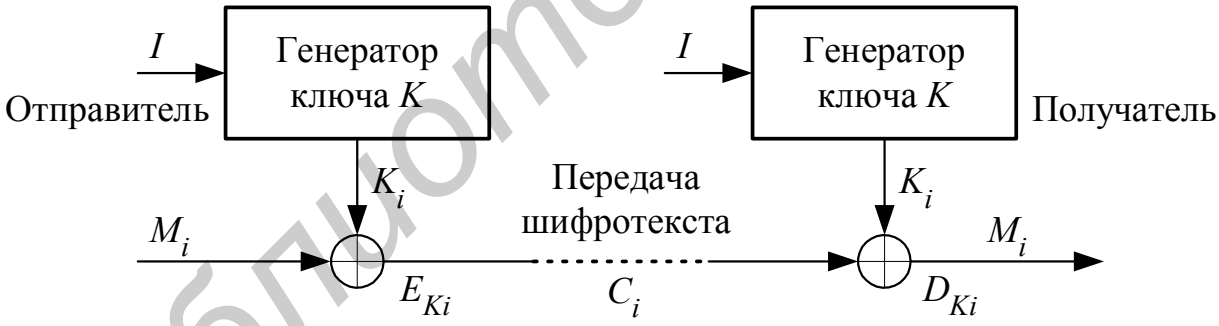

Рис. 1. Схема потоковой криптосистемы

Для корректной работы потоковых криптосистем необходимо, чтобы передающая и принимающая сторона имели синхронизированные генераторы ключа K. Искажение отдельных символов не влияет на расшифровку остальных символов шифротекста. Добавление, удаление или дублирование символов шифротекста нарушает синхронизацию ключевой и текстовой последовательностей, и все последующие символы расшифровываются некорректно.

Рассмотрим генераторы ключей на основе сдвиговых регистров с линейной обратной связью LFSR (Linear Feedback Shift Register). Они достаточно просто реализуются в программном и аппаратном виде, обладают высокой скоростью генерации и большим периодом ключа. Регистр LFSR состоит из двух

частей: сдвигового регистра, выполняющего сдвиг своих разрядов влево на один разряд, и функции обратной связи, вычисляющей вдвигаемое в первый разряд значение. В обобщенном виде структура LFSR представлена на рис. 2.

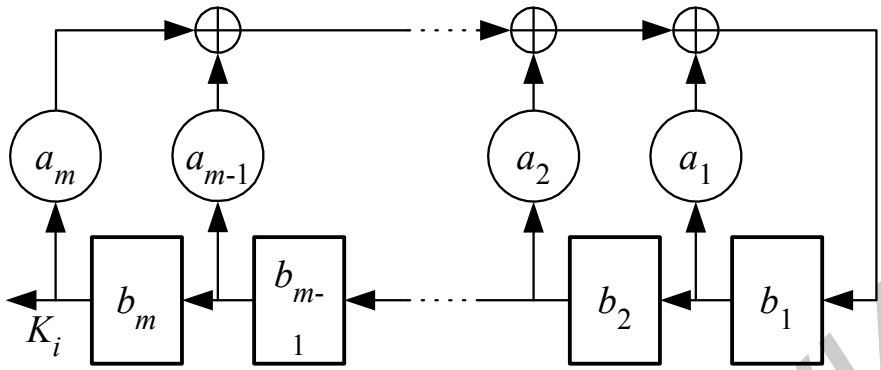

Рис. 2. Обобщенная схема LFSR

Структуру конкретного регистра LFSR принято задавать с помощью характеристического (задающего) многочлена:

$$P(x) = a_m x^m + a_{m-1} x^{m-1} + \ldots + a_2 x^2 + a_1 x + 1, \; a_k \in \{0,1\}, \; k = 1, \ldots, m. \qquad (5)$$

Степень многочлена m задает длину сдвигового регистра. Ненулевые коэффициенты a_k определяют разряды регистра, которые будут участвовать в формировании вдвигаемого в первый разряд значения. Через b_k ($b_k \in \{0,1\}$) обозначены текущие значения разрядов LFSR ($k = 1, \ldots, m$). Новые значения разрядов b^*_k ($b^*_k \in \{0,1\}$) вычисляются по следующему закону:

$$b^*_k = \begin{cases} \sum_{j=1}^{m} \oplus a_j b_j, & k = 1 \quad \text{– функция обратной связи} \\ b_{k-1}, & k = 2, \ldots, m \quad \text{– сдвиг.} \end{cases} \qquad (6)$$

Таким образом, новое значение для всех разрядов, кроме первого, берется из ближайшего младшего разряда. Символом $\sum\oplus$ обозначена многоместная операция сложения «ИСКЛЮЧАЮЩЕЕ-ИЛИ». Она используется для сложения значений из разрядов, которые отмечены ненулевыми коэффициентами a_k характеристического многочлена. Полученная сумма вдвигается в первый разряд LFSR. Очередной бит ключа K_i для потоковой криптосистемы равен значению старшего разряда LFSR b_m.

Пример 1. Построим сдвиговый регистр с линейными обратными связями, задаваемый характеристическим многочленом $P(x) = x^4 + x + 1$.

Схема регистра приведена на рис. 3. Если начальное состояние регистра $I = 1111$, то последовательность генерируемых состояний имеет вид

1111→1110→1101→1010→0101→1011→0110→1100→
1001→0010→0100→1000→0001→0011→0111→1111

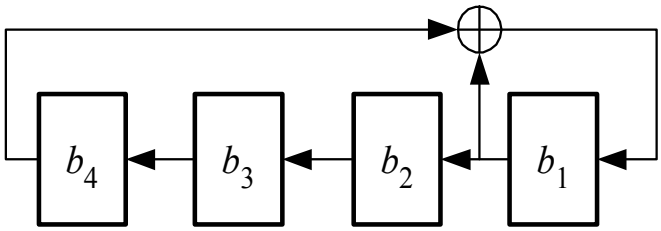

Рис. 3. LFSR на основе многочлена $P(x) = x^4 + x + 1$

Последнее состояние совпадает с начальным, после этого указанная последовательность повторяется. Последовательность сгенерированных бит ключа: `111101011001000 1`.

Свойства генерируемой последовательности определяются постоянными коэффициентами характеристического многочлена a_i. При их соответствующем выборе генерируемая последовательность K будет иметь максимально возможный период, равный $2^m - 1$. Последовательность максимально возможного для данного генератора периода называется М-последовательностью. Основная задача синтеза генератора на основе LFSR – нахождение характеристического многочлена, позволяющего сформировать М-последовательность. Для того чтобы конкретный LFSR имел максимальный период, соответствующий многочлен должен быть примитивным. В общем случае не существует простого способа нахождения примитивных многочленов данной степени, проще выбирать многочлен случайным образом и проверять, является ли он примитивным. Эта задача аналогична задаче проверки на простоту случайно выбранного целого числа и решается с помощью вычислительной техники. Табл. 1 содержит некоторые примитивные многочлены.

Таблица 1

Примитивные многочлены

m	$P(x)$	m	$P(x)$	m	$P(x)$
23	$x^{23} + x^5 + 1$	29	$x^{29} + x^2 + 1$	35	$x^{35} + x^2 + 1$
24	$x^{24} + x^4 + x^3 + x + 1$	30	$x^{30} + x^{16} + x^{15} + x + 1$	36	$x^{36} + x^{11} + 1$
25	$x^{25} + x^3 + 1$	31	$x^{31} + x^3 + 1$	37	$x^{37} + x^{12} + x^{10} + x^2 + 1$
26	$x^{26} + x^8 + x^7 + x + 1$	32	$x^{32} + x^{28} + x^{27} + x + 1$	38	$x^{38} + x^6 + x^5 + x + 1$
27	$x^{27} + x^8 + x^7 + x + 1$	33	$x^{33} + x^{13} + 1$	39	$x^{39} + x^4 + 1$
28	$x^{28} + x^3 + 1$	34	$x^{34} + x^{15} + x^{14} + x + 1$	40	$x^{40} + x^{21} + x^{19} + x^2 + 1$

Использование LFSR для создания потоковых криптосистем предполагает наличие уязвимости, связанной с линейным характером генерируемой последовательности. Обладая парой «исходный текст – шифротекст» длиной всего $2m$ бит, взломщик может восстановить характеристический многочлен и дешифровать все сообщение. Для защиты от этой атаки следует увеличивать размер используемого LFSR или использовать более сложные схемы генерации ключа. Рассмотрим, для примера, генератор Геффе (Geffe), представленный на рис. 4.

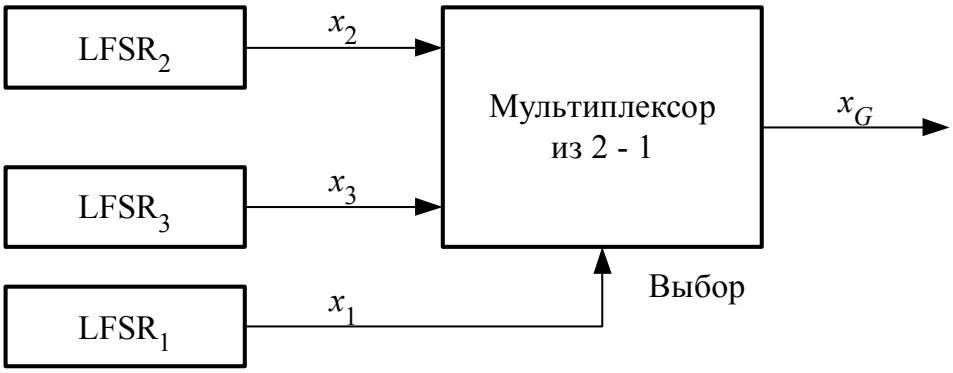

Рис. 4. Генератор Геффе

В нем используются три регистра с линейной обратной связью, объединённые нелинейным образом. Два регистра LFSR являются входами мультиплексора, а третий – управляет его выходом. Через x_1, x_2 и x_3 обозначены выходы трёх LFSR, выход генератора Геффе x_G описывается так: $x_G = (x_1$ *and* $x_2)$ *or* (*not* x_1 *and* x_3). Период данного генератора равен $(2^{m1}-1)(2^{m2}-1)(2^{m3}-1)$, где $m1$, $m2$ и $m3$ – длины первого, второго и третьего LFSR соответственно.

Благодаря простоте реализации, высокой скорости работы и сравнительно высокой криптостойкости потоковые шифраторы получили широкое распространение для шифрования информации средней степени секретности. Например, алгоритм A5, построенный на основе трех LFSR с прореженными обратными связями, входит в состав стандарта мобильной связи GSM.

Возможны и другие способы генерации ключевой последовательности. Например, генератор ключевого потока K в алгоритме RC4 работает на основе подстановочной таблицы S (S-бокса) из 256 символов. На первом шаге S-бокс заполняется линейно: $S[0] = 0$, $S[1] = 1$, …, $S[255] = 255$. Затем начальное значение S-бокса меняется на основе пользовательского секретного ключа U по следующему алгоритму:

```
j := 0;
for i := 0 to 255
j := (j + S[i] + U[i mod length(U)]) mod 256;
swap(S[i], S[j]);
```

Эта подстановка является функцией от ключа изменяемой длины. Процедура swap меняет местами значения таблицы S с заданными индексами.

Полученное значение S-бокса используется для побайтной генерации ключевого потока K. Генератор потока имеет два счетчика i и j, инициализируемых нулевым значением. На каждом шаге генерации выполняются следующие операции:

```
i := (i + 1) mod 256
j := (j + S[i]) mod 256
swap(S[i], S[j])
K := S[(S[i] + S[j]) mod 256]
```

Байт *K* складывается операцией «ИСКЛЮЧАЮЩЕЕ-ИЛИ» с байтом открытого текста для получения байта шифротекста либо с байтом шифротекста для получения байта исходного текста. Шифрование происходит весьма быстро – примерно в 10 раз быстрее DES-алгоритма. Типичная реализация выполняет 19 машинных команд на каждый байт текста. Алгоритм RC4 может принимать примерно 256! · 2562 = 21700 возможных состояний. S-бокс медленно изменяется в процессе работы: параметр *i* обеспечивает изменение каждого элемента, а *j* отвечает за то, чтобы эти элементы изменялись случайным образом. Шифр обладает иммунитетом к методам линейного и дифференциального криптоанализа и до сих пор в нем не обнаружены короткие циклы.

Алгоритм RC4, в отличие от алгоритмов на основе сдвиговых регистров LSFR, больше ориентирован на программную реализацию, поскольку работает не с битами, а с целыми байтами исходного текста. Благодаря своей скорости и защищенности, он нашел широкое применение в криптографических системах. Например, он является частью протокола безопасного обмена информацией SSL.

Задания

1. Реализуйте систему потокового шифрования файлов с помощью генератора ключевой последовательности на основе линейного сдвигового регистра с обратной связью $LFSR_1$.
2. Модифицируйте программу из задания 1 путем реализации схемы Геффе с тремя регистрами $LFSR_1$, $LFSR_2$ и $LFSR_3$ (размерность регистров приведена в табл. 2).
3. Реализуйте алгоритм потокового шифрования RC4.

Таблица 2

Варианты заданий

Степень многочлена	Номер варианта							
	1	2	3	4	5	6	7	8
$LFSR_1$	23	24	25	26	27	28	29	30
$LFSR_2$	31	32	33	34	35	36	37	38
$LFSR_3$	39	40	23	24	25	26	27	28

3. РОТОРНЫЕ КРИПТОСИСТЕМЫ

Наиболее известная роторная криптосистема называлась «Энигма» и использовалась для защиты связи между командованием и подводными лодками немецкой армии в годы второй мировой войны. Конструкция «Энигмы» была основана на системе из трех роторов, осуществлявших замену 26 букв латинского алфавита. Каждый ротор имел 26 входных контактов на одной стороне и столько же выходных контактов – на другой. Внутри каждого ротора проходили провода, связывавшие входные и выходные контакты между собой. Выходные контакты первого ротора соединялись с входными контактами второго ротора. Когда оператор нажимал на какую-либо букву на клавиатуре машины, электрический ток подавался на входной контакт первого ротора, соответствующий этой букве. Ток проходил через первый ротор и поступал на выходной контакт, соответствующий какой-либо другой букве. Затем ток проходил последовательно через второй и третий роторы и подавался на неподвижный рефлектор (от лат. reflecto – обращаю назад, отражаю). В конструкции рефлектора 26 контактов разбивались на пары, контакты внутри каждой пары были соединены между собой. Таким образом, рефлектор заменял каждую букву на парную ей. Ток, прошедший через рефлектор, подавался назад, на систему роторов. Он вновь проходил через три ротора, но в обратном порядке. В конце концов на световом табло «Энигмы» загоралась одна из 26 лампочек, соответствовавшая зашифрованной букве (рис. 5). Самым важным свойством «Энигмы» являлось вращение роторов. Первый ротор после каждого преобразования буквы поворачивался на одну позицию. Второй ротор поворачивался на одну позицию после того, как первый ротор совершал полный оборот, т.е. после 26-ти преобразованных букв. Наконец, третий ротор поворачивался на одну позицию после того, как второй ротор совершал полный оборот, т.е. после 676-ти зашифрованных букв.

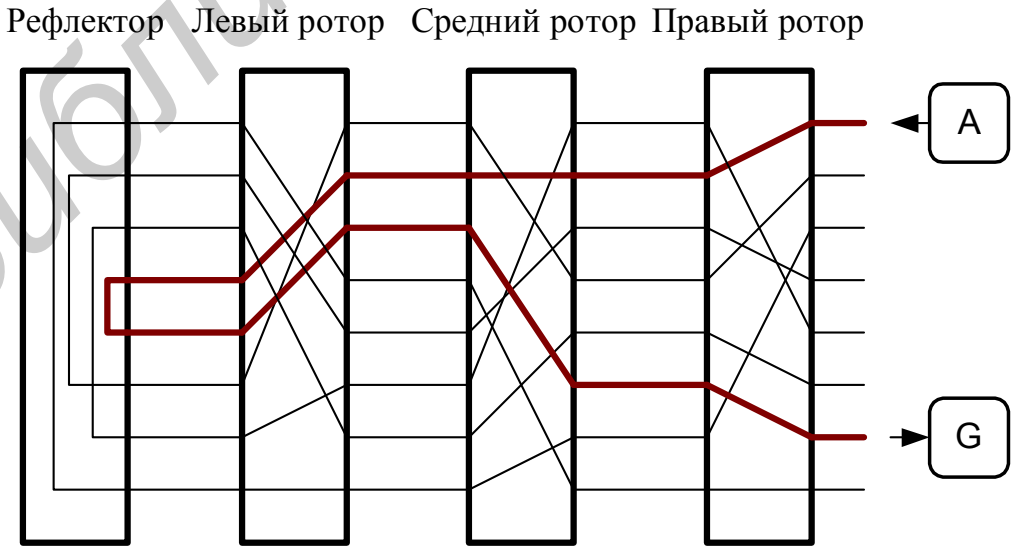

Рис. 5. Устройство шифровальной машины «Энигма»

Благодаря рефлектору, «Энигма» на каждом шаге осуществляла перестановку букв внутри пар, и если, к примеру, буква «N» заменялась на «S», то при том же положении роторов буква «S» менялась на «N» (ток тек по тем же проводам, но в другую сторону). Поэтому для расшифровки сообщения достаточно было вновь пропустить его через машину, восстановив предварительно изначальное положение роторов. Таким образом, начальное положение роторов играло роль ключа шифрования. Каждый оператор имел специальную книгу, задававшую положение роторов для каждого дня. В этом заключалась очевидная слабость данной системы шифрования: достаточно было завладеть книгой и машиной, чтобы раскрыть все секреты. Что и произошло осенью 1942 года, когда войсками союзников была захвачена немецкая подводная лодка U-571.

В общем случае криптосистема на основе роторной машины осуществляет полиалфавитную подстановку с длинным периодом . Пусть роторная машина состоит из банка t роторов (дисков) R_1, R_2, …, R_t. Каждый ротор R_i задает функцию подстановки f_i символов открытого текста символами зашифрованного текста. После каждого шага шифрования любой ротор может быть смещен (повернут) на одну из N позиций, и каждая позиция изменяет функцию соответствия между символами исходного и зашифрованного текста. Если диск R_i находится в позиции j_i, то выполняемая им эффективная подстановка символа m_i описывается выражением

$$F_i(a) = ((f_i(m_i - j_i) \ \mathbf{mod}\ N) + j_i) \ \mathbf{mod}\ N. \qquad (7)$$

Машина, состоящая из t роторов, осуществляет эффективную подстановку символов шифротекста по закону

$$E_{k_i}(m_i) = F_t(F_{t-1}(\ldots(F_1(m_i)\ldots)). \qquad (8)$$

Ключ шифрования k_i состоит из исходных функций подстановки f_1, …, f_t и текущих позиций роторов j_1, …, j_t. Как только шифруется очередной символ, один или более роторов изменяют свою позицию, изменяя этим и сам ключ. Машина, состоящая из t роторов, не сможет вернуться в начальное состояние, пока не произведет N^t успешных операций шифрования. Закон изменения позиций роторов необязательно должен быть линейным. В общем случае для получения новой позиции ротора может использоваться генератор псевдослучайных чисел, что практически исключает возможность узнать о перемещениях роторов, что защищает от корреляционных атак. Главным требованием является совпадение закона генерации последовательности поворотов у отправителя и получателя шифрсообщения.

Рассмотрим пример генератора на основе одного сдвигового регистра с линейной обратной связью LFSR длиной 80 бит (примитивный многочлен $P(x) = x^{80} + x^{79} + x^{43} + x^{42} + 1$) (рис. 6). Три байта RCB (Rotor Control Byte) с позициями (I, J, K) выбираются из регистра LFSR случайно и определяют перемещение каждого из трех роторов на текущем шаге шифрования. Причем значения (I, J, K) также являются частью ключа.

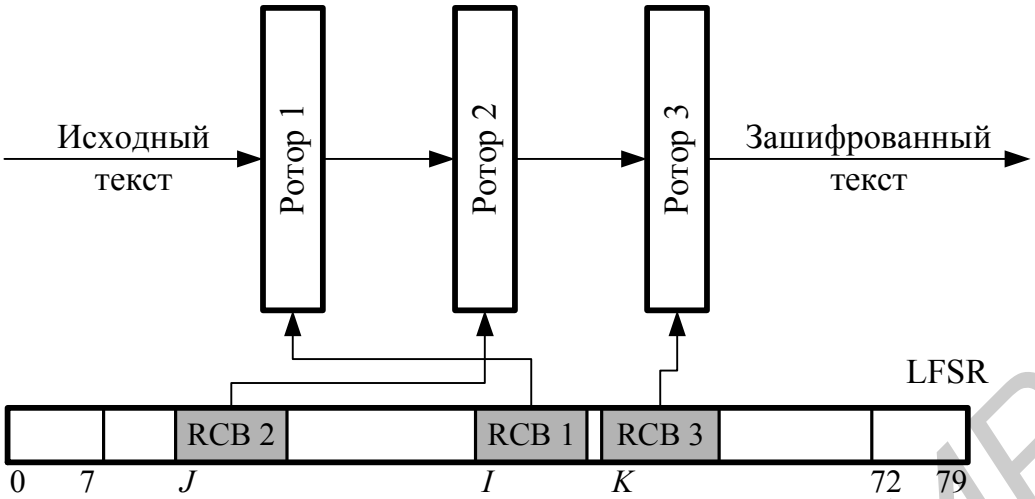

Рис. 6. Управление положением трех роторов с помощью одного линейного сдвигового регистра с обратной связью

Функция подстановки для каждого ротора задается с помощью соответствующей таблицы. Для шифрования сообщения байтами она должна задавать 256 однозначных правил замены входных символов символами шифротекста. При отсутствии рефлектора таблицы подстановки на сторонах отправителя и получателя должны задавать взаимообратные правила – отправитель ищет символ шифротекста по его индексу в первой таблице, а получатель по пришедшему символу находит соответствующий ему индекс с помощью второй таблицы.

Для увеличения периода ключа в каждом роторе может быть организовано шифрование с обратной связью по шифротексту, когда символ, полученный в результате шифрования на предыдущем шаге, используется для шифрования следующего символа (рис. 7).

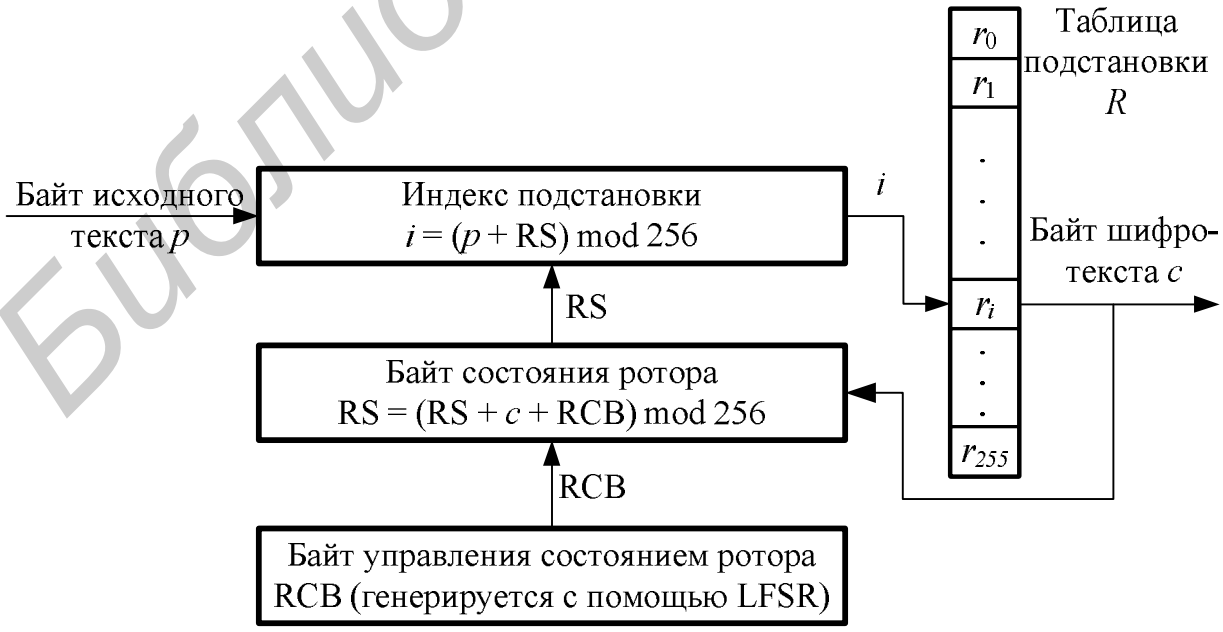

Рис. 7. Реализация ротора с обратной связью по шифротексту

Как следует из рис. 7, значение байта открытого текста складывается по модулю 256 с байтом текущего состояния ротора RS (Rotor State). Значение RS меняется после каждого шага шифрования на основе значения RCB из генератора поворотов ротора, предыдущего зашифрованного байта c и предыдущего состояния ротора RS. Результат сложения i используется как индекс элемента r из таблицы подстановки $R = \{r_0, r_1, \ldots, r_{255}\}$. Элемент r_i используется в качестве результата шифрования c (если ротор R – последний) или в качестве входного байта p для следующего ротора.

Длина ключа для представленного алгоритма составляет $3 \cdot 256 + 10 + 3 + 3 = 784$ байтов. Пользователь не способен запомнить такое количество данных, поэтому необходимо предусмотреть возможность генерации ключевых данных на основе пользовательского пароля.

Главными достоинствами рассмотренной криптосистемы являются простота программной и аппаратной реализации, а также высокие криптостойкость и скорость шифрования. Для кодирования одного байта необходимо трижды выполнить три операции сложения и одну операцию адресации по индексу, а также один такт работы регистра LFSR (десять сдвигов). Всего $(3 + 1) \cdot 3 + 10 = 22$ элементарных операций с 8-битными операндами.

Задания

1. Реализовать криптографический алгоритм, использованный в электронно-механической роторной криптосистеме «Энигма».
2. Реализовать криптографический алгоритм с использованием усовершенствованной роторной машины, использующей регистр LFSR в качестве генератора поворотов роторов по псевдослучайному закону и обратную связь по шифротексту в роторах.
3. Разработать криптосистему, основанную на криптографическом алгоритме из задания 1 или 2. Криптосистема должна обеспечивать выполнение трех функций: генерацию ключа, шифрование текста, дешифрование текста. Проверить статистические свойства криптосистемы с помощью диаграммы распределения символов (частоты встречаемости в тексте) для открытого и зашифрованного текста.

4. СИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ. АЛГОРИТМ IDEA

Алгоритм IDEA (International Data Encryption Algorithm) относится к классу симметричных шифраторов. Данный алгоритм был разработан в 1990 г. в качестве альтернативы алгоритму DES (Data Encryption Standard). В основе алгоритма лежит идея смешанного преобразования, которое случайным образом равномерно распределяет исходный текст по всему пространству шифротекста.

Смешанные преобразования реализуются при помощи перемежающихся последовательностей замен и простых операций перестановок. Преобразование данных производится по блокам, размер которых равен 64 битам. Длина ключа в алгоритме IDEA составляет 128 бит.

Каждый 64-битный блок рассматривается как четыре 16-битных подблока, которые преобразуются с использованием следующих целочисленных операций.

1. Побитное сложение по модулю 2 (XOR) двух 16-битных операндов, которое будем обозначать как $\oplus$.

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

3. Умножение двух чисел без знака по модулю $2^{16}+1$. Результат операции умножения усекается до длины в 16 бит. При вычислении данной операции существует исключение для кода со всеми нулями, который при умножении рассматривается как число 2^{16}. Данную операцию будем обозначать как $\odot$.

Процедура шифрования состоит из восьми одинаковых раундов и дополнительного 9-го выходного раунда (рис. 8, *а*).

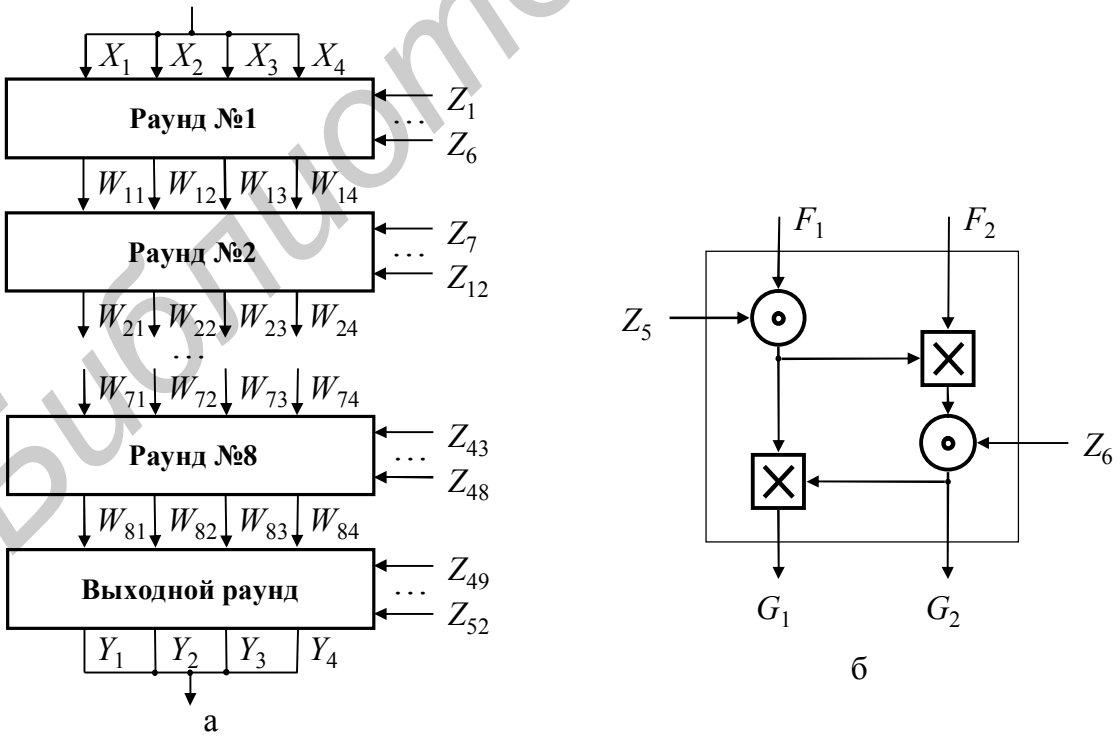

Рис. 8. Алгоритм IDEA:
а – схема процедуры шифрования; б – мультипликативно-аддитивная структура

На выходе 9-го раунда формируется содержимое четырёх 16-битных подблоков, образующих блок шифротекста.

Основной частью каждого раунда является мультипликативно-аддитивная структура (рис. 8, *б*).

Здесь F_1 и F_2 – 16-битные значения, полученные из открытого текста, Z_5 и Z_6 – 16-битные подключи.

Все операнды, участвующие в выполнении процедуры шифрования, имеют размерность 16 бит.

На рис. 9 приведена схема выполнения первого раунда алгоритма IDEA.

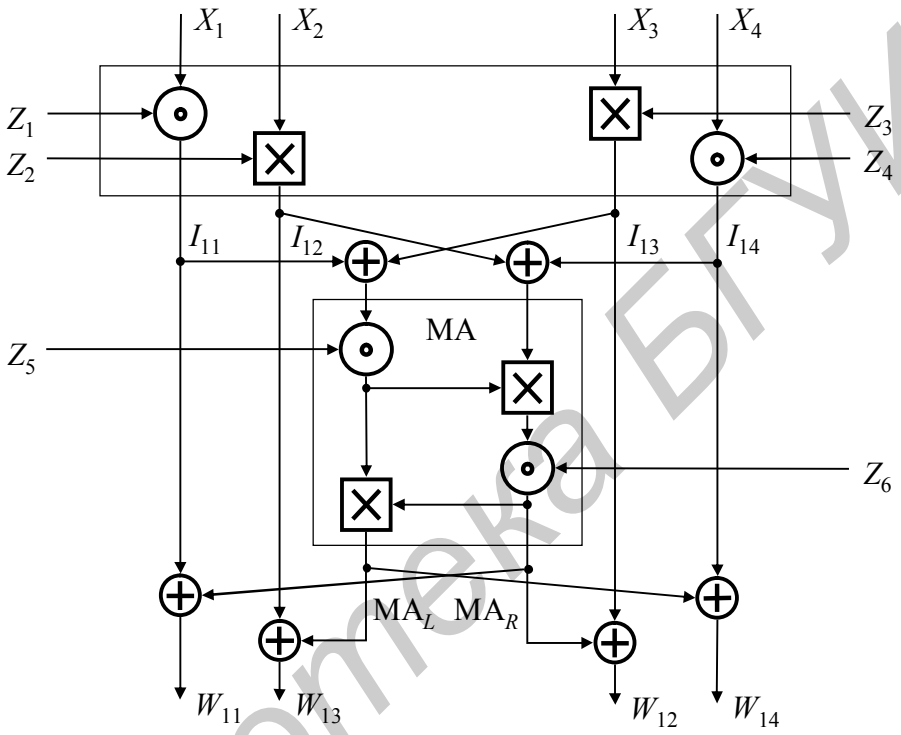

Рис. 9. Первый раунд шифрования алгоритма IDEA

Данные, получаемые на выходе *i*-го раунда шифрования, подаются на вход (*i*+1)-го раунда. Входными данными 1-го раунда являются четыре 16-битных подблока (X_1, X_2, X_3, X_4) 64-битного блока исходного текста.

Схема выполнения 9-го раунда шифрования приведена на рис. 10.

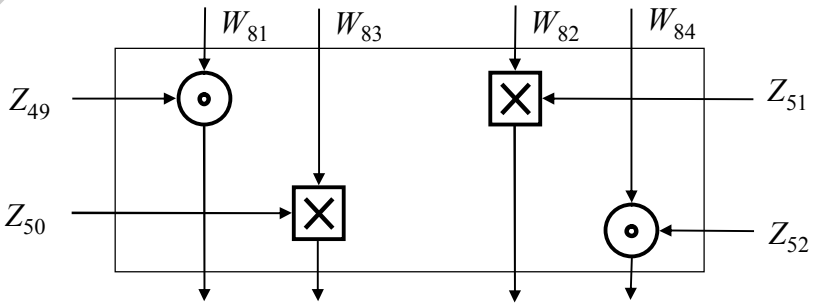

Рис. 10. Девятый раунд шифрования алгоритма IDEA

Следует обратить внимание на то, что второй и третий подблоки промежуточного значения W меняются местами после выполнения каждого раунда шифрования кроме восьмого.

На каждом из девяти раундов используются значения 16-битных итерационных ключей Z_i, которые получаются путём преобразования исходного 128-битного ключа K.

Первые 8 итерационных ключей $Z_1 \ldots Z_8$ берутся как восемь последовательных частей 128-битного ключа. Для получения следующих восьми итерационных ключей 128-битное значение ключа K циклически сдвигается на 25 бит влево и ключи $Z_9 \ldots Z_{16}$ вновь берутся как его 8 последовательных частей. Данный процесс повторяется до тех пор, пока не будут получены все 52 итерационных ключа.

Процедура дешифрования состоит из тех же девяти раундов, но только выполняемых с использованием иных значений итерационных ключей. Итерационные ключи дешифрования получают из итерационных ключей шифрования на основе таблицы соответствия (табл. 3).

Таблица 3

Значения ключей, используемых в алгоритме IDEA для дешифрования

Итерация (раунд)	Обозначение	Эквивалентное обозначение
1	$U_1, U_2, U_3, U_4, U_5, U_6$	$Z_{49}^{-1}, -Z_{50}, -Z_{51}, Z_{52}^{-1}, Z_{47}, Z_{48}$
2	$U_7, U_8, U_9, U_{10}, U_{11}, U_{12}$	$Z_{43}^{-1}, -Z_{45}, -Z_{44}, Z_{46}^{-1}, Z_{41}, Z_{42}$
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	$U_{43}, U_{44}, U_{45}, U_{46}, U_{47}, U_{48}$	$Z_7^{-1}, -Z_9, -Z_8, Z_{10}^{-1}, Z_5, Z_6$
9	$U_{49}, U_{50}, U_{51}, U_{52}$	$Z_1^{-1}, -Z_2, -Z_3, Z_4^{-1}$

При этом выполняются следующие соотношения:

$$Z_j^{-1} \odot Z_j = 1 \bmod (2^{16}+1); \tag{9}$$

$$-Z_j \boxtimes Z_j = 0 \bmod 2^{16}. \tag{10}$$

Таким образом, для ключа Z_j значение, обозначаемое как $-Z_j$, является аддитивным инверсным по модулю 2^{16}, а значение, обозначаемое как Z_j^{-1} – мультипликативным инверсным по модулю $2^{16}+1$.

Порядок использования итерационных ключей при шифровании показан на рис. 11.

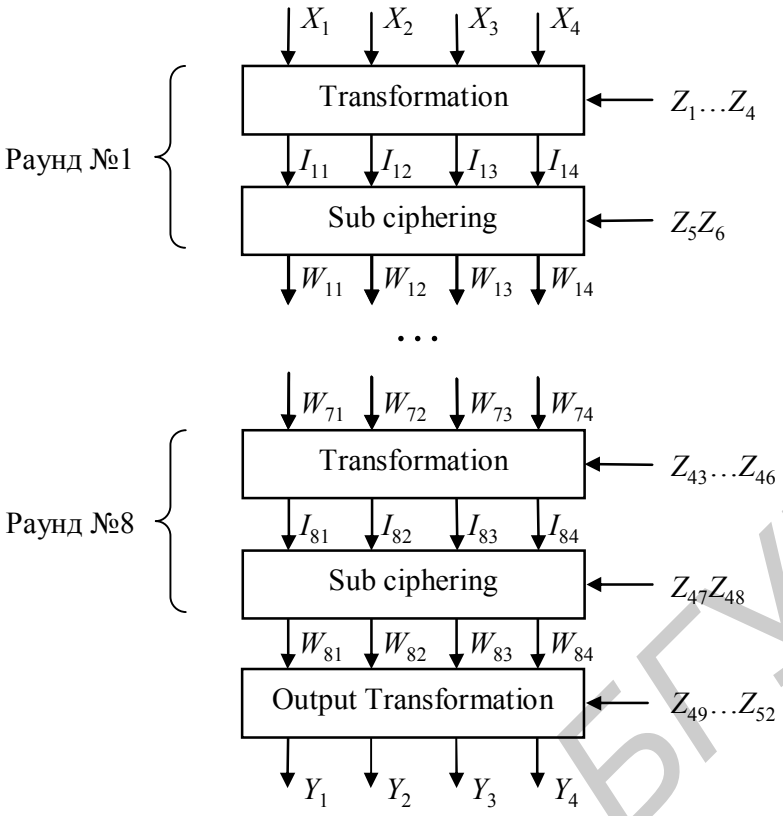

Рис. 11. Порядок использования итерационных ключей алгоритма IDEA

При выполнении дешифрования раунды алгоритма выполняются в таком же порядке. На вход первого раунда подаётся четыре 16-битных подблока 64-битного блока шифротекста. Значения, полученные после выполнения выходного раунда, являются подблоками 64-битного блока исходного текста. Отличие от процедуры шифрования заключается в том, что вместо ключей $Z_1 \ldots Z_{52}$ используются ключи $U_1 \ldots U_{52}$.

Задания

1. Разработать программное средство, выполняющее шифрование по алгоритму IDEA заданного файла с произвольным содержимым. Ключ шифрования подаётся в виде бинарного файла длиной 16 байт.

2. Разработать программное средство, выполняющее дешифрование заданного файла, зашифрованного по алгоритму IDEA. Ключ шифрования подается в виде бинарного файла длиной 16 байт.

5. АРИФМЕТИКА ЧИСЕЛ БОЛЬШОЙ РАЗРЯДНОСТИ

Размерность обрабатываемых в вычислительных машинах чисел обычно ограничивается размерностью машинного слова. Типичная переменная целочисленного типа занимает в памяти машины 8, 16, 32 или 64 бит. Для многих криптографических алгоритмов требуются числа намного большего размера. Например, рекомендуемый размер открытого ключа для алгоритма RSA составляет 4 Кбит. Рассмотрим реализацию базовых арифметических операций над целыми числами большого размера. Для представления цифр больших чисел удобно использовать систему счисления с основанием b, равным 2^m, где m – размер машинного слова. Это наиболее компактный способ представления больших чисел, позволяющий хранить все цифры в массиве слов-переменных.

Алгоритм сложения

Алгоритм сложения неотрицательных чисел достаточно прост: цифры числа складываются, начиная с младших разрядов к старшим. Если зафиксировано переполнение (т. е. при сложении получена цифра, большая максимально возможной в данной системе счисления), то происходит перенос значения в следующий разряд. Рассмотрим реализацию сложения неотрицательных n-разрядных целых чисел $(u_{n-1}, \ldots, u_0)_b$ и $(v_{n-1}, \ldots, v_0)_b$ по основанию b. Следующий алгоритм формирует их сумму $(w_n, w_{n-1}, \ldots, w_0)_b$, причем $w_n \in \{0, 1\}$:

```
ADD(u, v, n)
j := 0; k := 0;
while j < n
      do w_j := u_j + v_j + k
           if w_j ≥ b
              then w_j := w_j – b
                   k := 1
              else k := 0
           j := j + 1
w_n := k
return (w_n, …, w_0)
```

Заметим, что при работе этого алгоритма всегда выполняются соотношения $u_j + v_j + k \le (b - 1) + (b - 1) + 1 < 2b$, так что размер результата суммирования не превышает $\log_2 2b = b + 1$ разрядов. Приведенный алгоритм может использоваться и для сложения отрицательных чисел. Для этого следует использовать их представление в дополнительном коде.

Алгоритм умножения

Умножение больших чисел может быть выполнено традиционным школьным способом «в столбик». Однако вместо использования массива про-

межуточных результатов гораздо эффективнее добавлять к произведению каждую новую строку немедленно после ее вычисления.

Если множимое состоит из m слов, множитель – из n слов, то произведение занимает не более $m + n$ слов, независимо от того, выполняется знаковое или беззнаковое умножение. Рассмотрим реализацию умножения неотрицательных целых чисел $(u_{m-1}, \ldots, u_0)_b$ и $(v_{n-1}, \ldots, v_0)_b$ по основанию b. Следующий алгоритм формирует их произведение $(w_{m+n-1}, \ldots, w_0)_b$:

```
MULTIPLY (w, v, m, n)
for j = 0, …, m – 1 do w_j := 0;
j := 0;
while j < n
    do if v_j > 0
        then i := 0; k := 0;
            while i < m
                do t := u_i · v_j + w_{i+j} + k:
                    w_{i+j} := t mod b;
                    k := ⌊t / b⌋;
                    i := i + 1;
            w_{j+m} := k;
        else w_{j+m} := 0;
    j := j + 1;
return (w_{m+n-1}, …, w_0)
```

На каждом шаге алгоритма умножения выполняются неравенства

$$0 \le t < b^2,\ 0 \le k < b.$$

Умножение больших чисел выполняется проще для беззнаковых операндов. Знак произведения получается как результат операции «ИСКЛЮЧАЮЩЕЕ-ИЛИ» над разрядами знака множителей.

Умножение целых чисел может быть существенно ускорено. Например, пусть $u = b^n U_1 + U_0$, $v = b^n V_1 + V_0$. Тогда (алгоритм Карацубы)

$$\begin{aligned} uv &= b^{2n} U_1 V_1 + b^n (U_1 V_0 + U_0 V_1) + U_0 V_0 = \\ &= b^{2n} U_1 V_1 + b^n ((U_0 + V_0)(U_1 + V_1) - U_0 V_0 - U_1 V_1) + U_0 V_0 = \\ &= b^{2n} C_0 + b^n (C_2 - C_1 - C_0) + C_1, \end{aligned} \qquad (11)$$

где $C_0 = U_1 V_1$, $C_1 = U_0 V_0$, $C_2 = (U_0 + V_0)(U_1 + V_1)$.

Алгоритм Карацубы сводит задачу умножения двух чисел к нескольким задачам умножения чисел меньшей разрядности. Разбиение может осуществляться рекурсивно до тех пор, пока разрядность не уменьшится до поддерживаемой аппаратно (т.е. пока n не достигнет размера машинного слова). В этом случае число элементарных умножений для алгоритма Карацубы асимптотически сходится к $O(n^{\log_2 3})$.

Пример: $13 \cdot 27 = 100 \cdot 1 \cdot 2 + 10 \cdot (3 \cdot 7 + 1 \cdot 2) + 3 \cdot 7 = 100 \cdot 2 + 10 \cdot (36 - 21 - 2) + 21 = 200 + 130 + 21 = 351$.

Обобщением алгоритма Карацубы является алгоритм Тома–Кука, в котором множители могут разбиваться более чем на две части. Максимальной скоростью на сегодняшний день обладает алгоритм умножения на основе быстрого преобразования Фурье. В этом случае цифры произведения получаются как коэффициенты свертки цифр множителей, посчитанные с учетом переносов значений между коэффициентами.

Деление

Основная сложность при реализации классического метода деления «в столбик» состоит в необходимости угадывать разряды частного на каждой итерации алгоритма. Этот процесс должен быть формализован. Прежде всего заметим, что деление m-разрядного числа на n-разрядное ($m > n$) сводится к последовательности делений $(n + 1)$-разрядных чисел u на n-разрядное число v, причем $0 \le u/v < b$, где b – основание системы счисления. Таким образом, необходимо построить алгоритм для нахождения $q = \lfloor u/v \rfloor$, $u = (u_n, u_{n-1}, \ldots, u_0)_b$ и $v = (v_n, v_{n-1}, \ldots, v_0)_b$.

Условие $u/v < b$ может быть переформулировано как $u/b < v$, т. е. $(u_n, u_{n-1}, \ldots, u_1)_b < v = (v_n, v_{n-1}, \ldots, v_0)_b$. Частное q должно быть единственным целым числом, таким, что $0 \le u - qv < v$. Попытаемся угадать q как

$$q^* = \min\left(\left\lfloor \frac{u_n b + u_{n-1}}{v_{n-1}} \right\rfloor, b-1\right). \qquad (12)$$

В данном случае $q^* > q$, при этом если q^* превышает q, то превышение незначительно. Кроме того, если $v_{n-1} \ge \lfloor b/2 \rfloor$, то $q^* - 2 \le q \le q^*$. Это условие носит название условия нормализации. Его можно обеспечить, домножив делимое и делитель на $\lfloor b/(v_{n-1}+1) \rfloor$. Дополнительно можно показать, что если $q^* v_{n-2} > br^* + u_{n-2}$, то $q^* < q$, где $r^* = u_n b + u_{n-1} + q^* v_{n-1}$. В противном случае $q \in \{q^*, q^* - 1\}$.

Исходя из этих соображений, алгоритм вычисления частного m и остатка n от деления $(u_{m+n-1}, \ldots, u_0)_b$ на $(v_{n-1}, \ldots, v_0)_b$ имеет вид

DIVIDE(u, v, m, n)
$d := \lfloor b/(v_{n-1}+1) \rfloor$;
$(u_{m+n}, u_{m+n-1}, \ldots, u_0)_b := d \cdot (u_{m+n-1}, \ldots, u_0)_b$
$(v_{n-1}, \ldots, v_0)_b := d \cdot (v_{n-1}, \ldots, v_0)_b$
$j := m$
while $j \ge 0$
  do $q^* := \lfloor (u_{j+n} b + u_{j+n-1})/v_{n-1} \rfloor$
    $r^* := (u_{j+n} b + u_{j+n-1}) \bmod v_{n-1}$
    if $(q^* = b) \vee (q^* v_{n-2} > br^* + u_{j+n-2})$
      then $q^* := q^* - 1$

$r^* := r^* + v_{n-1}$

if $(r^* < b) \vee (q^* v_{n-2} > b r^* + u_{j+n-2})$

then $q^* := q^* - 1$

$r^* := r^* + v_{n-1}$

$(u_{j+n}, \ldots, u_j)_b := (u_{j+n}, \ldots, u_j)_b - q^*(v_{n-1}, \ldots, v_0)_b$

if $(u_{j+n}, \ldots, u_j)_b < 0$

then NegFlag := true

$(u_{j+n}, \ldots, u_j)_b := (u_{j+n}, \ldots, u_j)_b + b^{n+1}$

else NegFlag := false

$q_j = q^*$

if NegFlag = true

then $q_j := q_j - 1$

$(u_{j+n}, \ldots, u_j)_b := (u_{j+n}, \ldots, u_j)_b + (0, v_{n-1}, \ldots, v_0)_b$

$j := j - 1$

return $((q_m, \ldots, q_1 q_0)_b, (u_{n-1}, \ldots, u_0)_b / d)$

Некоторые фрагменты этого алгоритма выполняются очень редко, что затрудняет отладку.

Задания

1. Реализуйте алгоритмы «в столбик» для вычисления суммы, произведения и частного двух целых чисел большой разрядности.
2. Реализуйте алгоритмы Карацубы для умножения целых чисел большой разрядности.
3. Сравните скорость работы и затраты памяти для реализованных в заданиях 1 и 2 алгоритмов умножения целых чисел большой разрядности.

6. АСИММЕТРИЧНЫЕ КРИПТОСИСТЕМЫ. АЛГОРИТМ RSA

Криптосистема RSA была предложена в 1977 году тремя учёными: Роном Ривестом, Ади Шамиром и Леонардом Адлеманом. Алгоритм шифрования данных RSA является асимметричным, поскольку для шифрования и расшифрования данных используются два различных ключа, называемых открытым и закрытым. Принципиальное отличие криптосистемы RSA от симметричных криптосистем заключается в том, что раскрытие ключа, которым было произведено шифрование, не влечёт за собой раскрытия исходного текста. Данный факт допускает пересылку ключа шифрования без использования каких-либо защищённых каналов связи. Это обеспечивается свойством криптосистемы RSA, в соответствии с которым использованный для шифрования данных открытый ключ не подходит для их расшифрования. Для расшифрования используется закрытый ключ, который не может быть получен из открытого.

Шифрование в системе RSA производится путём возведения исходного текста X ($0 \le X < r$) в степень открытого ключа K_O по модулю большого числа r:

$$Y = E_{K_O}(X) = X^{K_O} \bmod r. \tag{13}$$

Расшифрование производится путём возведения шифротекста Y в степень закрытого ключа K_C по модулю r :

$$Y^{K_C} \bmod r = (X^{K_O} \bmod r)^{K_C} \bmod r = X^{K_O \cdot K_C} \bmod r = X \bmod r. \tag{14}$$

Возможность получения исходного текста из зашифрованного обеспечивается свойством открытого и закрытого ключей, для которых должно выполняться равенство

$$(K_O \cdot K_C) \bmod \varphi(r) = 1, \tag{15}$$

где $\varphi(r)$ – функция Эйлера от r. Закрытый ключ является мультипликативным инверсным по модулю $\varphi(r)$ для открытого. По теореме Эйлера

$$a^{\varphi(r)} = 1 \bmod r, \tag{16}$$

где $(a, r) = 1$.

Известно, что если

$$a = b \bmod r, \tag{17}$$

то

$$a^n = b^n \bmod r. \tag{18}$$

Из этого следует, что

$$X^{n \cdot \varphi(r)} = 1 \bmod r. \tag{19}$$

Поскольку для модулярной арифметики справедливо

$$m \cdot a = m \cdot b \bmod r,$$

то мы можем умножить левую и правую части равенства (19) на X:

$$X^{n \cdot \varphi(r)+1} = X \bmod r. \tag{20}$$

Если представить показатель степени $n \cdot \varphi(r)+1$ как произведение значений K_O и K_C, то получим

$$K_O \cdot K_C = 1 \bmod \varphi(r). \qquad (21)$$

Процедура формирования параметров алгоритма RSA выглядит следующим образом:

1) генерируются два больших простых числа: p и q, $p \neq q$;

2) вычисляется произведение данных чисел: $r = p \cdot q$;

3) находится функция Эйлера от r: $\varphi(r) = (p-1) \cdot (q-1)$;

4) генерируется значение открытого ключа K_O, исходя из следующих условий: $K_O < \varphi(r)$, $(K_O, \varphi(r)) = 1$;

5) вычисляется мультипликативное инверсное по модулю $\varphi(r)$ для K_O. Полученное значение будет являться закрытым ключом K_C.

Для вычисления мультипликативного инверсного можно использовать расширение алгоритма Евклида. Расширенный алгоритм Евклида позволяет найти значения x_1 и y_1, при которых выполняется равенство $x_1 \cdot a + y_1 \cdot b = d_1$, где $d_1 = \text{НОД}(a, b)$. Если $a > b$ и числа a, b – взаимно простые, т. е. $d_1 = 1$, то y_1 является мультипликативным инверсным для b по модулю a.

```
EUCLIDEX(a; b)
d0:=a; d1:=b;
x0:=1; x1:=0;
y0:=0; y1:=1;
while d1>1 do
begin
    q:=d0 div d1;
    d2:=d0 mod d1;
    x2:=x0–q*x1;
    y2:=y0–q*y1;
    d0:=d1; d1:=d2;
    x0:=x1; x1:=x2;
    y0:=y1; y1:=y2;
end
return (x1; y1; d1)
```

Если расширенный алгоритм Евклида используется для вычисления мультипликативного инверсного и в результате выполнения алгоритма получено отрицательное значение, то к нему необходимо прибавить величину a.

Для выполнения процедур шифрования и расшифрования можно использовать алгоритм быстрого возведения в степень по модулю, позволяющий для положительных целых a, z и n вычислить $x = a^z \bmod n$.

```
FASTEXP(a; z; n)
a1:=a; z1:=z; x:=1;
while z1≠0 do
begin
    while (z1 mod 2)=0 do
    begin
        z1:=z1 div 2;
        a1:=(a1*a1) mod n;
    end
    z1:=z1–1;
    x:=(x*a1) mod n;
end
return (x)
```

Рассмотрим пример использования алгоритма RSA:

1) выберем два простых числа: $p = 41$ и $q = 59$;

2) вычислим их произведение $q = p \cdot q = 2419$;

3) вычислим функцию Эйлера: $\varphi(r) = (p–1)\cdot(q–1) = 40 \cdot 58 = 2320$;

4) выберем $K_O = 157$; (2320, 157) = (157, 122) = (122, 35) = (35, 17) = 1;

5) найдем значение K_C, для которого выполняется: $157 \cdot K_C = 1 \bmod \varphi(r) = 1 \bmod 2320$. Положив a равным $\varphi(r)$ и b равным K_O, по расширенному алгоритму Евклида вычислим $K_C = y_1 = 133$.

В качестве примера зашифруем строку «BSUIR». Символам исходного текста соответствуют следующие десятичные ASCII-коды:

«B» : 66;
«S» : 83;
«U» : 85;
«I» : 73;
«R» : 82.

Таким образом, исходный текст M будет представлять собой последовательность чисел: M={66, 83, 85, 73, 82}. Тогда компоненты шифротекста C будут получены следующим образом:

$C[1] = M[1]^{K_O} \bmod r = 66^{157} \bmod 2419 = 1425$;

$C[2] = M[2]^{K_O} \bmod r = 83^{157} \bmod 2419 = 575$;

$C[3] = 85^{157} \bmod 2419 = 1473$;

$C[4] = 73^{157} \bmod 2419 = 483$;

$C[5] = 82^{157} \bmod 2419 = 2296$.

Для расшифрования шифротекста C={1425, 575, 1473, 483, 2296} воспользуемся закрытым ключом K_C:

$M[1] = C[1]^{K_C} \bmod r = 1425^{133} \bmod 2419 = 66$;

$M[2] = 575^{133} \bmod 2419 = 83$;

$M[3] = 1473^{133} \bmod 2419 = 85;$

$M[4] = 483^{133} \bmod 2419 = 73;$

$M[5] = 2296^{133} \bmod 2419 = 82.$

Таким образом, мы вновь получили исходный текст {66, 83, 85, 73, 82}, который соответствует строке «BSUIR».

Субъект, желающий получать зашифрованные по алгоритму RSA сообщения, публикует сгенерированную им пару значений (K_O, r) в общедоступном месте. Значение закрытого ключа K_C, а также значения p и q хранятся в секрете.

Для того чтобы организовать секретный канал связи, абоненты A и B посылают друг другу свои открытые ключи, не заботясь о том, что их значения узнает кто-то посторонний. При обмене сообщениями абонент A шифрует отправляемые данные открытым ключом абонента B, а абонент B шифрует отправляемые данные открытым ключом абонента A. Для расшифрования полученных сообщений участники переписки используют свои секретные ключи.

Криптостойкость системы RSA основывается на сложности определения закрытого ключа по открытому, поскольку для этого потребуется вычислить функцию Эйлера от значения параметра r, для чего необходимо разложить r на множители. На сегодняшний день данная задача не имеет эффективного решения, а использование традиционных методов поиска множителей для чисел размерностью порядка 2^{2048} (именно такие числа рекомендуется использовать в алгоритме RSA) не позволяет осуществить взлом криптосистемы за разумное время.

Задания

1. Разработать программное средство, выполняющее вычисление открытого ключа (K_O) алгоритма RSA и побайтовое шифрование данным ключом по алгоритму RSA произвольного файла. Значения параметров p, q и K_C, а также имя входного файла задаются пользователем. Программа должна осуществлять проверку ограничений на вводимые пользователем значения параметров алгоритма.

2. Разработать программное средство, выполняющее расшифрование файла, каждый 16-битный блок которого представляет собой зашифрованное по алгоритму RSA 8-битное значение. Значения модуля r и *закрытого* ключа K_C задаются пользователем.

3. Разработать программное средство, выполняющее дешифрование (взлом) файла, каждый 16-битный блок которого представляет собой зашифрованное по алгоритму RSA 8-битное значение. Значения модуля r и *открытого* ключа K_O, которым был *зашифрован* файл, задаются пользователем.

7. ЭЛЕКТРОННАЯ ЦИФРОВАЯ ПОДПИСЬ

При электронном документообороте традиционные методы подтверждения подлинности документа при помощи оттиска печати или поставленной от руки подписи являются непригодными. Вместо этого используется электронная цифровая подпись (ЭЦП) под электронным документом, представляющая собой небольшой объём данных, передаваемых вместе с документом при его пересылке.

Любая система электронной цифровой подписи включает в себя процедуру подписания электронного документа и процедуру проверки данной подписи.

Наибольшим удобством использования обладают схемы цифровой подписи на основе криптосистем с открытым ключом. Для таких схем при постановке подписи используется закрытый ключ автора электронного документа, а при проверке подписи – открытый ключ.

Важной составляющей любой системы электронной цифровой подписи является процедура вычисления дайджеста сообщения, необходимого для защиты передаваемой информации от случайного либо преднамеренного искажения. Для вычисления дайджеста используется криптографическая хеш-функция, формирующая хеш-образ отправляемого сообщения.

Алгоритм безопасного хеширования SHA-1

Алгоритм безопасного хеширования SHA-1 был опубликован в 1995 году в качестве замены использовавшегося до этого алгоритма хеширования SHA-0, в котором была обнаружена уязвимость.

Для сообщения произвольной длины l, не превышающей 2^{64} бит, алгоритм SHA-1 формирует 160-битный хеш-образ.

Процедура формирования хеш-образа состоит из следующих шагов.

1. Весь исходный текст разбивается на блоки по 512 бит. В случае если длина исходного текста не кратна 512 битам, производится его выравнивание за счёт добавления в конец бита со значением 1, m нулевых битов и 64-битного представления значения длины исходного сообщения (рис. 12).

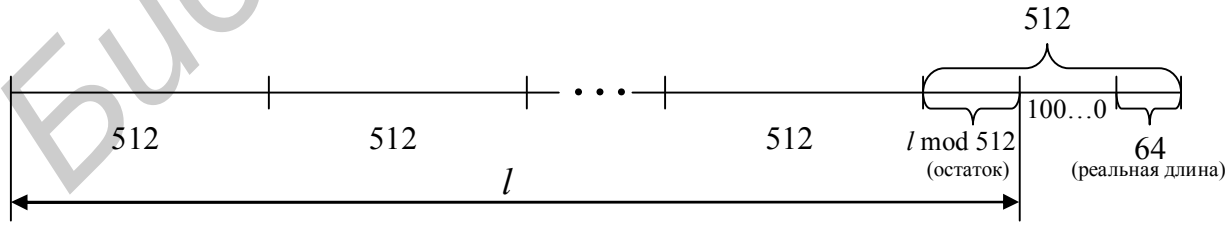

Рис. 12. Выравнивание исходного сообщения

2. Инициализируется пять 32-битных рабочих переменных A, B, C, D, E:

$A \leftarrow 67452301_{16}$

$B \leftarrow EFCDAB89_{16}$

$C \leftarrow 98BADCFE_{16}$
$D \leftarrow 10325476_{16}$
$E \leftarrow C3D2E1F0_{16}$

3. Выполняется обработка очередных 512 бит исходного текста. Для этого значения переменных A, B, C, D, E копируются в переменные a, b, c, d, e и далее для t от 1 до 80 выполняется преобразование значений данных переменных по схеме, изображенной на рис. 13.

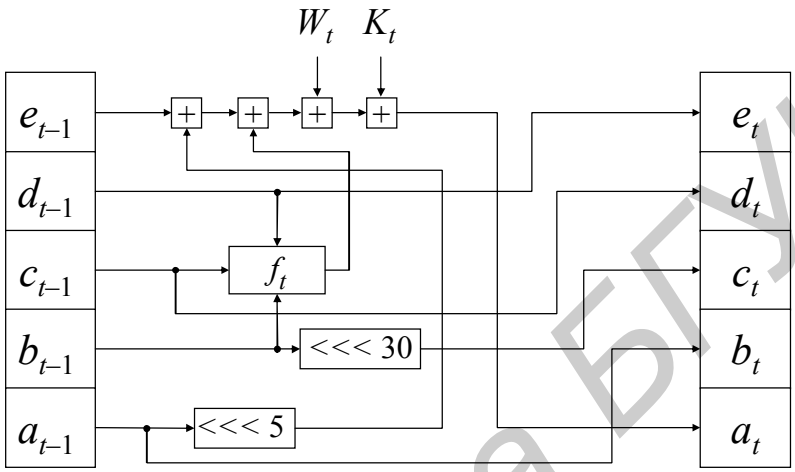

Рис. 13. Схема итерации алгоритма SHA-1

Каждая из 80 итераций может быть записана следующим образом:

$TMP \leftarrow (a <<< 5) + f_t(b, c, d) + e + W_t + K_t;$
$e \leftarrow d;$
$d \leftarrow c;$
$c \leftarrow (b <<< 30);$
$b \leftarrow a;$
$a \leftarrow TMP,$

где «+» – операция сложения по модулю 2^{32}, $f_t(X, Y, Z)$ – нелинейная функция, имеющая следующий вид:

$$f_t(X,Y,Z) = \begin{cases} (X \,\&\, Y) \mid (!X \,\&\, Z), & t = \overline{1,20}, \\ X \oplus Y \oplus Z, & t = \overline{21,40}, \\ (X \,\&\, Y) \mid (X \,\&\, Z) \mid (Y \,\&\, Z), & t = \overline{41,60}, \\ X \oplus Y \oplus Z, & t = \overline{61,80}, \end{cases} \quad (22)$$

где «&» – побитовая операция «И», «|» – побитовая операция «ИЛИ», «!» – операция побитового инвертирования, «⊕» – операция побитового сложе-

ния по модулю 2. Параметр K_t принимает четыре различных значения в зависимости от номера текущей итерации:

$K_t = \texttt{5A827999}_{16}, \quad t = \overline{1,20}$;

$K_t = \texttt{6ED9EBA1}_{16}, \quad t = \overline{21,40}$;

$K_t = \texttt{8F1BBCDC}_{16}, \quad t = \overline{41,60}$;

$K_t = \texttt{CA62C1D6}_{16}, \quad t = \overline{61,80}$.

«<<<» – операция циклического сдвига на 30 либо 5 бит влево, W_t – одно из шестнадцати 32-битных слов 512-битного блока сообщения при $t = \overline{1,16}$ либо значение, определяемое в соответствии со следующим выражением при $t = \overline{17,80}$:

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1. \qquad (23)$$

4. Значения переменных a, b, c, d, e независимо друг от друга складываются по модулю 2^{32} со значениями переменных A, B, C, D, E, в которые затем и помещаются полученные результаты.

5. Шаги 3–4 выполняются до тех пор, пока не будет обработан весь текст.

После обработки последнего блока текста значение хеш-образа формируется как $ABCDE$.

Алгоритм цифровой подписи RSA

Математическая схема электронной цифровой подписи по алгоритму RSA была предложена в 1977 году сотрудниками Массачусетского технологического института США. Данная система цифровой подписи стала первым практическим решением задачи подписи электронных документов при помощи криптосистем с открытым ключом. Процедура вычисления цифровой подписи в данной системе использует криптографическое преобразование по алгоритму RSA.

В соответствии с данной системой цифровой подписи, субъект, желающий пересылать подписанные им документы, должен сформировать два ключа алгоритма RSA: открытый и закрытый.

Пару значений (K_O, r), которая является открытым ключом подписи, отправитель передаёт всем возможным получателям его сообщений. Именно эти значения будут использоваться для проверки подлинности и принадлежности отправителю полученных от него сообщений.

Значение K_C сохраняется отправителем в секрете. Данное значение вместе с модулем r является секретным ключом, который будет использоваться отправителем для постановки подписей под своими сообщениями.

Схема использования алгоритма цифровой подписи на базе RSA для обмена двух абонентов подписанными сообщениями показана на рис. 14.

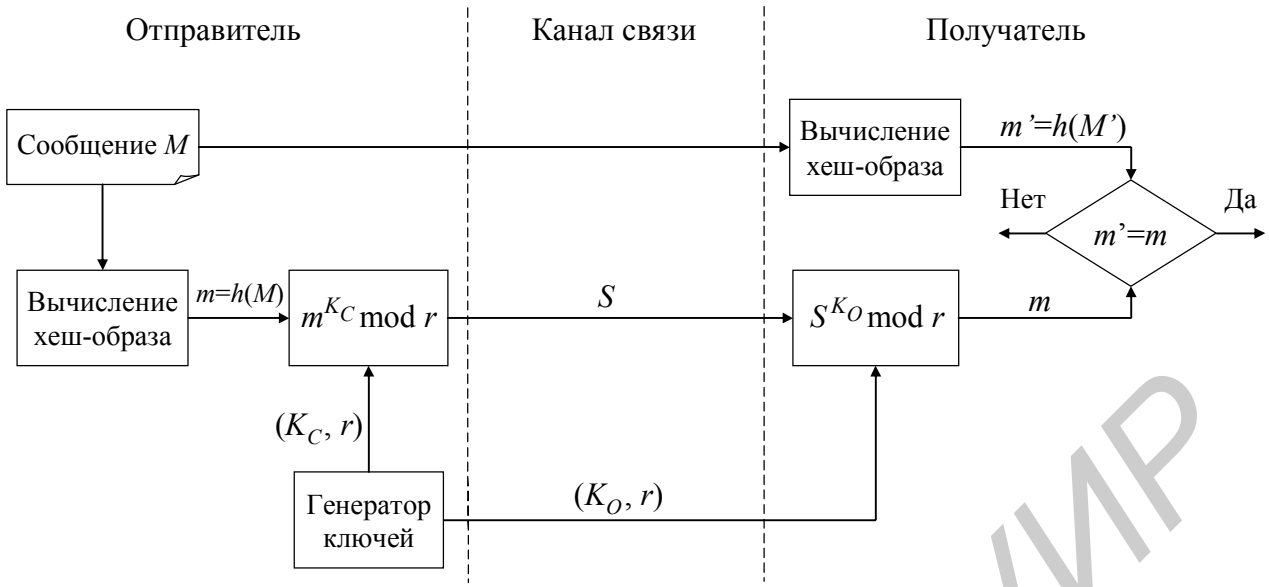

Рис. 14. Схема использования цифровой подписи на базе RSA

Допустим, что получатель уже располагает открытым ключом подписи отправителя. Процедура подписи отправителем сообщения M будет состоять из следующих шагов.

1. Отправитель сжимает сообщение M при помощи криптографической хеш-функции h в целое число $m = h(M)$.

2. Отправитель вычисляет значение цифровой подписи S для сообщения M на основе ранее полученного значения хеш-образа m и значения своего закрытого (секретного) ключа подписи K_C. Для этого используется преобразование, аналогичное преобразованию, выполняемому при шифровании по алгоритму RSA:

$$S = m^{K_C} \bmod r. \tag{24}$$

Пара (M, S), представляющая собой подписанное отправителем сообщение, передаётся получателю. Сформировать подпись S мог только обладатель закрытого ключа K_C.

Процедура проверки получателем подлинности сообщения и принадлежности его отправителю состоит из следующих шагов.

1. Получатель сжимает полученное сообщение M' при помощи криптографической хеш-функции h, идентичной той, которая была использована отправителем, в целое число m'.

2. Получатель выполняет расшифрование открытым ключом K_O отправителя дайджеста m оригинального сообщения, преобразуя значение подписи S по алгоритму RSA:

$$m = S^{K_O} \bmod r. \tag{25}$$

3. Получатель сравнивает полученные значения m' и m. Если данные значения совпадают, т. е.

$$S^{K_O} \bmod r = h(M'), \tag{26}$$

то получатель признает полученное сообщение подлинным и принадлежащим отправителю.

Фальсификация сообщения при его передаче по каналу связи возможна только при получении злоумышленником секретного ключа K_C либо за счет проведения успешной атаки против использованной для вычисления дайджеста сообщения хеш-функции. При использовании достаточно больших значений p и q определение секретного значения K_C по открытому ключу (K_O, r) является чрезвычайно трудной задачей, соответствующей по сложности разложению модуля r на множители. Используемые в реальных приложениях хеш-функции обладают характеристиками, делающими атаку против цифровой подписи практически неосуществимой.

Задания

1. Реализовать алгоритм вычисления хеш-функции SHA-1 для файла с произвольным размером и содержимым.

2. Реализовать программное средство, выполняющее генерацию и проверку ЭЦП файла с произвольным содержимым на базе алгоритма RSA с использованием для вычисления хеш-функции ранее реализованного алгоритма SHA-1.

8. КРИПТОСИСТЕМЫ НА ОСНОВЕ ЭЛЛИПТИЧЕСКИХ КРИВЫХ

Эллиптическая кривая над множеством действительных чисел может быть определена как набор точек (x, y), удовлетворяющих уравнению эллиптической кривой вида $y^2 = x^3 + ax + b$ (x, y, a и b – действительные числа), а также некоторый элемент O, называемый неопределенным (нулевым) элементом (рис. 15).

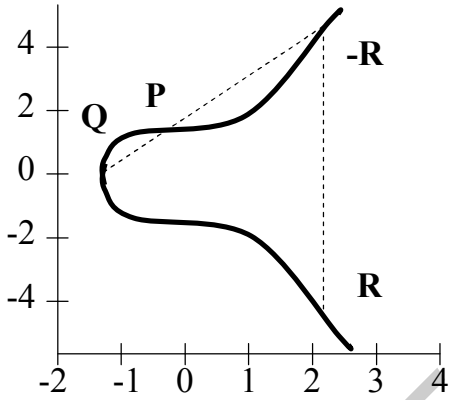

Рис. 15. Эллиптическая кривая $y^2 = x^3 + x + 1$

По определению, эллиптическая кривая обладает следующим свойством: если три ее точки лежат на одной прямой, то их сумма равна O. Это свойство позволяет описать правила сложения и умножения точек эллиптической кривой.

1. Если O – нулевой элемент, то справедливо равенство $O = -O$, а для любой точки P эллиптической кривой имеем $P + O = P$.
2. Прямая, проходящая через точки P и $-P$, является вертикальной прямой, которая не пересекает эллиптическую кривую ни в какой третьей точке. По определению, $P + (-P) = O$.
3. Пусть P и Q – две различные точки эллиптической кривой, и $P \neq -Q$. Проведем через P и Q прямую. Она пересечет эллиптическую кривую только еще в одной точке, называемой $-R$. Точка $-R$ отображается относительно оси X в точку R. Закон сложения: $P + Q = R$.
4. Чтобы сложить точку P с ней самой, нужно провести касательную к кривой в точке P. Если $y_P \neq 0$, то касательная пересечет эллиптическую кривую ровно в одной точке R. Закон удвоения точки P: $P + P = 2P = R$.
5. Умножение точки P на целое положительное k определяется как сумма k точек P: $kP = P + P + P + \ldots + P$.

Эллиптическая кривая может быть использована для построения эллиптической группы, если ее параметры a и b удовлетворяют соотношению $4a^3 + 27b^2 \neq 0 \pmod M$, где M – простое число, $a < M$ и $b < M$.

Эллиптическая группа $E_M(a,b)$ представляет собой набор точек (x, y) с целыми положительными координатами, $x < M$ и $y < M$, которые удовлетворяют соотношению $y^2 = x^3 + ax + b \pmod M$.

Алгоритм формирования элементов эллиптической группы:

1. Для всех значений x ($0 < x < M$) вычисляется значение $x^3 + ax + b \pmod M$.

2. Для каждого значения из шага 1 определяется квадратный корень по модулю M и элемент включается в группу $E_M(a,b)$, если результат положительный.

Пример. Пусть $a = 1$, $b = 0$ и $M = 23$, тогда $y^2 = x^3 + x$. Так как $4 \cdot 1^3 + 27 \cdot 1^2 = 8 \neq 0 \pmod{23}$, то можно построить группу $E_{23}(1,1)$. Существуют 23 точки, которые удовлетворяют уравнению группы, а именно (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). Заметим, что для каждого значения x существует по две точки с симметрией относительно $y = 11{,}5$. В случае эллиптических кривых над действительными числами для каждой точки имеется отрицательная, отображаемая относительно оси x. В случае использования конечной эллиптической группы отрицательные значения координаты y берутся по модулю, в результате чего получаются положительные координаты: $-P = (xP, (-yP \bmod M))$. Например, если $P = (1,5)$, то $-P = (1, (-5 \bmod 23)) = (1,18)$.

Рассмотрим алгоритм сложения точек эллиптической группы. Пусть $P = (x_1, y_1)$ и $Q = (x_2, y_2)$. Тогда $P + Q = (x_3, y_3)$, где $x^3 = \lambda^2 - x_1 - x_2 \pmod{M}$ и $y_3 = \lambda \times (x_1 - x_3 - y_1) \pmod{M}$, а

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) \pmod{M}, & \text{если } P \neq Q, \\ (3x_1^2 + a)/(2y_1) \pmod{M}, & \text{если } P = Q. \end{cases} \qquad (27)$$

Число λ – угловой коэффициент секущей, проведенной через точки $P = (x_1,y_1)$ и $Q = (x_2, y_2)$. При $P = Q$ секущая превращается в касательную, чем и объясняется наличие двух формул для вычисления λ.

Пример. Эллиптическая группа задана уравнением $y^2 = x^3 + 9x + 17 \bmod 23$. Необходимо найти произведение $2P=P+P$ для точки $P = (16, 5)$ из этой группы

$$\lambda = (3x_P^2 + a)/(2y_P) \bmod M = (3 \cdot 16^2 + 9)/(2 \cdot 5) \bmod 23;$$

тогда $10\lambda = 18 \bmod 23$, отсюда

$$\lambda = 18 \cdot 10^{\varphi(23)-1} \bmod 23 = 18 \cdot 10^{21} \bmod 23 = 11;$$

$$x_R = \lambda^2 - 2x_P \bmod M = (11^2 - 2 \cdot 16) \bmod 23 = 20;$$

$$y_R = -y_P + \lambda(x_P - x_P) \bmod M = -5 + 11 \cdot (16 - 20) \bmod 23 = -49 \bmod 23 = 20.$$

В результате получаем $2P = (20,20)$.

Умножение точек эллиптической группы на число определяется аналогично умножению для эллиптических кривых как многократное сложение точки с собой. Если вычислять $P + P + P + \ldots$ достаточно долго, то, т.к. число точек конечно, в конце концов должен быть получен результат O. Всегда можно найти такие a и b $(b > a)$, что $aP = bP$. Это означает, что $cP = O$, где $c = b - a$. Наименьшее c, для которого это справедливо, называется ***порядком*** точки.

Использование эллиптических групп в криптографических целях основано на сложности решения задачи дискретного логарифмирования в эллиптической группе, которая может быть сформулирована так: для заданных точек P и Q найти такое k, чтобы $kP = Q$. Значение k называется логарифмом от Q по ос-

нованию P. Если взять значение k достаточно большим, то задача нахождения k становится практически неосуществимой.

Алгоритм обмена ключами в эллиптической группе

Прежде всего задается большое простое число M, а также параметры a и b для определения эллиптической группы $E_M(a,b)$. Затем выбирается генерирующая точка G такая, чтобы c, для которого $cG = O$, было большим простым числом с «хорошими алгебраическими свойствами». Размер числа c на практике довольно велик ($2^{254} < c < 2^{256}$). Параметры $E_M(a,b)$ и G открыты для всех.

Процедура обмена ключами между пользователями ***A*** и ***B***:

1. Пользователь ***A*** выбирает целое число n_A такое, что $n_A < M$. n_A – секретный ключ пользователя ***A***. Затем пользователь ***A*** генерирует открытый ключ $P_A = n_A G$. Заметим, что P_A – точка из группы $E_M(a,b)$.
2. Пользователь ***B*** выполняет те же действия для получения своего секретного ключа n_B и открытого ключа P_B.
3. Пользователь ***A*** генерирует общий секретный ключ $K = n_A P_B$, а пользователь ***B*** генерирует тот же секретный ключ $K = n_B P_A$.
4. Легко показать, что $n_A P_B = n_A(n_B G) = n_B(n_A G) = n_B P_A$.

Для взлома третья сторона должна вычислить k (n_A или n_B) на основании G и kG, т.е. решить задачу дискретного логарифмирования в эллиптической группе.

Пример. Пусть $M = 211$, $E_{211}(0,-4)$ и $G = (2,2)$. Легко проверить, что $241G = O$, а 241 является простым числом.

1. Секретный ключ $n_A = 121$, тогда открытый ключ $P_A = n_A G = 121 \cdot (2,2) = (115,48)$.
2. Секретный ключ $n_B = 203$, открытый ключ $P_B = n_B G = 203 \cdot (2,2) = (130,203)$.
3. Общий секретный ключ $K = n_A P_B = n_B P_A = 121 \cdot (130,203) = 203 \cdot (115,48) = (161,69)$.

Алгоритм ЭЦП на основе эллиптических кривых (ECDSA)

Алгоритм ECDSA (Elliptic Curve DSA) является аналогом алгоритма цифровой подписи DSA (Digital Signature Algorithm), реализованным с помощью эллиптических групп. Рассмотрим эллиптическую группу $E_M(a,b)$ и генерирующую точку G с порядком q, причем q простое число.

Пользователи ***A*** генерируют свои ключи: секретный n_A и открытый P_A, где $P_A = n_A G$. Для постановки цифровой подписи под сообщением m пользователь ***A***:

1. На основе хеш-функции $h()$ находит хеш-код $h(m)$ от m. В качестве хеш-функции должна использоваться криптографически стойкая функция, например SHA-1.
2. Генерирует случайное число k, $1 < k < q - 1$.
3. Вычисляет значение $kG = (x_1,y_1)$ и $r = x_1 \bmod q$. Если $r = 0$, возвращается на шаг 2.
4. Вычисляет $s = k^{-1}(h(m) + n_A r) \pmod q$. Если $s = 0$, то возвращается на шаг 2.
5. Подпись сообщения m – это пара целых чисел (r,s).

Для проверки цифровой подписи пользователь ***B*** использует ту же эллиптическую группу $E_M(a,b)$, генерирующую точку G, открытый ключ P_A и хеш-функцию h.

1. На основе хеш-функции h определяем хеш-код $h(m)$ от m.
2. Проверяем, принадлежат ли числа $\boldsymbol{r}$ и $\boldsymbol{s}$ интервалу от 1 до $q-1$.
3. Вычисляем $w = s^{-1} \bmod q$.
4. Вычисляем $u_1 = h(m)w \bmod q$ и $u_2 = rw \bmod q$.
5. Вычисляем $u_1G + u_2P_A = (x_1^*, y_1^*)$ и $r^* = x_1^* \bmod q$.
6. Равенство $r^* = r$ удостоверяет подпись пользователя ***A***.

Особое достоинство криптосистем на эллиптических кривых состоит в том, что по сравнению с системами на основе алгоритма RSA они обеспечивают существенно более высокую стойкость при равной трудоемкости или, наоборот, существенно меньшую трудоемкость при равной стойкости. В результате тот уровень стойкости, который достигается в RSA при использовании 1024-битных модулей, в системах на эллиптических кривых реализуется при размере модуля 160 бит, что обеспечивает более простую как программную, так и аппаратную реализацию.

Задания

1. Для заданного M (табл. 4) определить значения a и b, которые позволяют построить эллиптическую группу $E_M(a, b)$.
2. Для найденных в задании 1 параметров сгенерировать все элементы эллиптической группы $E_M(a, b)$.
3. Реализовать алгоритм обмена ключами для эллиптической группы $E_M(a,b)$.
4. Разработать алгоритм цифровой подписи на основе эллиптической группы $E_M(a, b)$.

Таблица 4

Варианты заданий

Параметр	Номер варианта							
	1	2	3	4	5	6	7	8
Модуль M	47	53	59	61	67	71	73	79

Св. план 2007, поз. 60

Учебное издание

Ярмолик Вячеслав Николаевич
Занкович Артем Петрович
Портянко Сергей Сергеевич

Элементы теории информации

Практикум
для студентов специальности
«Программное обеспечение информационных технологий»
дневной и дистанционной форм обучения

Редактор М. В. Тезина
Корректор Е. Н. Батурчик

Подписано в печать 23.05.2007. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,44.
Уч.-изд. л. 2,3. Тираж 100 экз. Заказ 9.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6