

Учреждение образования  
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**П. П. Урбанович, Н. П. Шутько**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ  
ПО ДИСЦИПЛИНАМ  
«ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ  
ИНФОРМАЦИОННЫХ СИСТЕМ»  
И «КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ  
ЗАЩИТЫ ИНФОРМАЦИИ»**

**В 2-х частях**

**Часть 2. Криптографические и стеганографические  
методы защиты информации**

*Рекомендовано  
учебно-методическим объединением по образованию  
в области информатики и радиоэлектроники  
в качестве учебно-методического пособия для студентов  
учреждений высшего образования по специальностям  
1-40 01 01 «Программное обеспечение  
информационных технологий»,  
1-40 05 01 «Информационные системы и технологии  
(по направлениям)» направлению специальности  
1-40 05 01-03 «Информационные системы и технологии  
(издательско-полиграфический комплекс)»,  
1-98 01 03 «Программное обеспечение информационной  
безопасности мобильных систем»*

Минск 2020

УДК [004.056+003.26](075.8)

ББК 32.972я73

У69

Р е ц е н з е н т ы:

кафедра управления информационными ресурсами

Института управленческих кадров Академии управления

при Президенте Республики Беларусь (заведующий кафедрой  
кандидат физико-математических наук, доцент *Б. В. Новыши*);

заведующий кафедрой информационных радиотехнологий  
учреждения образования «Белорусский государственный университет  
информатики и радиоэлектроники» доктор технических наук,  
профессор *Н. И. Листопад*

*Все права на данное издание защищены. Воспроизведение всей книги или ее части не может быть осуществлено без разрешения учреждения образования «Белорусский государственный технологический университет».*

**Урбанович, П. П.**

У69

Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 2. Криптографические и стеганографические методы защиты информации : учеб.-метод. пособие для студентов учреждений высшего образования по специальностям 1-40 01 01 «Программное обеспечение информационных технологий», 1-40 05 01 «Информационные системы и технологии (по направлениям)» направлению специальности 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем» / П. П. Урбанович, Н. П. Шутько. – Минск : БГТУ, 2020. – 226 с.

ISBN 978-985-530-860-8.

Издание содержит материалы для подготовки и выполнения лабораторных работ в соответствии с учебными планами дисциплин «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». Вторая часть пособия включает работы, относящиеся к криптографии, стеганографии, а также к нейрокриптографии. Задания к каждой лабораторной работе дополнены необходимыми теоретическими сведениями с примерами. Первая часть была издана в 2019 г.

Пособие предназначено для студентов ИТ-специальностей, изучающих дисциплины, относящиеся к указанной предметной области. Может быть полезно и магистрантам, научным работникам, объектами исследований которых являются методы и средства обеспечения информационной безопасности.

УДК [004.056+003.26](075.8)

ББК 32.972я73

**ISBN 978-985-530-860-8 (Ч. 2)**

**ISBN 978-985-530-763-2**

© УО «Белорусский государственный  
технологический университет», 2020

© Урбанович П. П., Шутько Н. П., 2020

## **ПРЕДИСЛОВИЕ**

В рамках существующих стандартов образования и учебных планов ряда специальностей (в том числе 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», 1-40 01 01 «Программное обеспечение информационных технологий» (специализация 1-40 01 01-10 «Программирование Интернет-приложений»), 1-47 01 02 «Дизайн электронных и веб-изданий», 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем») предусматривается изучение студентами комплекса специальных дисциплин, направленных на овладение теоретическими знаниями и практическими навыками в анализируемой предметной области. К числу таких дисциплин относятся: «Защита информации и надежность информационных систем» (специальность 1-40 05 01-03), «Криптографические методы защиты информации» (1-98 01 03, 1-40 01 01-10, 1-47 01 02).

Настоящее учебно-методическое пособие охватывает часть тем, предусмотренных учебными программами перечисленных учебных дисциплин. Основу пособия составляют теоретические сведения и практические задания, относящиеся к криптографическим и стеганографическим методам защиты данных в информационных системах.

Материал пособия включает 14 лабораторных работ. По каждой выполненной работе студент оформляет отчет по установленной форме. Для защиты работы преподавателю представляется электронный вариант отчета. При оформлении отчета полезно ознакомиться с содержанием второй и третьей глав учебно-методического пособия [1].

Книга является логическим продолжением и дополнением к учебно-методическому пособию [2].

В основу материалов некоторых лабораторных работ легли результаты совместных исследований сотрудников кафедры информационных систем и технологий БГТУ и Института математики и информатики Католического университета Люблина (Польша).

Авторы выражают признательность М. Г. Савельевой за помощь в подготовке рукописи пособия.

## **Лабораторная работа № 1**

# **ОСНОВЫ ТЕОРИИ ЧИСЕЛ И ИХ ИСПОЛЬЗОВАНИЕ В КРИПТОГРАФИИ**

**Цель:** приобретение практических навыков выполнения операций с числами для решения задач в области криптографии и разработка приложений для автоматизации этих операций.

**Задачи:**

1. Закрепить теоретические знания по высшей арифметике.
2. Научиться практически решать задачи с использованием простых и взаимно простых чисел, вычислений по правилам модульной арифметики и нахождению обратных чисел по модулю.
3. Ознакомиться с особенностями реализации готового программного средства L\_PROST и особенностями выполнения с его помощью операций над простыми числами.
4. Разработать приложение для реализации указанных преподавателем операций с числами.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения эксперимента с использованием приложения и результатов эксперимента.

### **1.1. Теоретические сведения**

В основе современной криптографии лежит *теория чисел*.

Теория чисел, или **высшая арифметика**, – раздел математики, изучающий натуральные числа и иные похожие величины. В зависимости от используемых методов в теории чисел рассматривают несколько направлений. Нас будут интересовать вопросы *делимости целых чисел, вычисления наибольшего общего делителя (НОД), разложение числа на простые множители, малая теорема Ферма, теорема Эйлера, элементы теории вычетов*.

#### **1.1.1. Основные понятия и определения**

**Определение 1.** Множество всех *целых чисел* (обозначим буквой  $Z$ ) есть набор всех **действительных чисел** без дробной части:  $\{..., -3, -2, -1, 0, 1, 2, 3, ...\}$ .

**Определение 2.** *Натуральные числа* являются подмножеством целых чисел и образуют множество  $N$ :  $\{1, 2, 3, \dots\}$ .

**Определение 3. Делимость** – одно из основных понятий теории чисел. Если для некоторого целого числа  $a$  и натурального числа  $b$  существует целое число  $q$ , при котором  $bq = a$ , то говорят, что число  $a$  *делится на*  $b$ . В этом случае  $b$  называется **делителем** числа  $a$ , а  $a$  называется **кратным** числу  $b$ . При этом используются следующие обозначения:

$a : b$  –  $a$  делится на  $b$ , или  $b | a$  –  $b$  делит  $a$ .

Из последнего определения следует, что:

- любое натуральное число является делителем нуля;
- единица является делителем любого целого числа;
- любое натуральное число является делителем самого себя.

**Определение 4.** Делитель  $a$  называется **собственным делителем** числа  $b$ , если  $1 < |a| < |b|$ , и **несобственным** – в противном случае.

**Пример 1.**  $4 | 20$ ; число 4 делит число 20, так как  $20 = 4 \cdot 5$ . При этом число 4 является собственным делителем числа 20.

*Свойство 1 собственного делителя:* положительный наименьший собственный делитель составного числа  $n$  не превосходит  $\sqrt{n}$ .

**Определение 5.** Всякое целое число  $a$  можно представить с помощью положительного целого числа  $b$  равенством вида  $a = bq + r$ ,  $0 \leq r \leq b$ . Число  $q$  называется **неполным частным**, а число  $r$  – **остатком** отделения  $a$  на  $b$ .

### 1.1.2. Простые и составные числа

Каждое натуральное число, большее единицы, делится по крайней мере на два числа: на 1 и на само себя.

! **Если число не имеет делителей, кроме самого себя и единицы, то оно называется простым, а если у числа есть еще делители, то составным.**

**Определение 6.** Натуральное число  $n$  называется **простым**, если  $n > 1$  и не имеет положительных делителей, отличных от 1 и  $n$ .

Простое число не делится без остатка ни на одно другое число.

**Пример 2.** Первые 10 простых чисел: 2, 3, 5, 7, 11, 13, 17, 19, 23 и 29. Простыми также являются числа 73, 2521, 2 365 347 734 339. Количество простых чисел бесконечно велико.

Перечислим несколько *важных свойств простых чисел*.

*Свойство 1.* Любое составное число представляется уникальным образом в виде произведения простых чисел; иначе еще говорят, что *разложение числа на простые множители однозначно*.

Это свойство вытекает из основной теоремы арифметики.

**Основная теорема арифметики. Всякое натуральное число  $n$ , кроме 1, можно представить как произведение простых множителей:**

$$n = p_1 p_2 p_3 \dots p_z, z > 1. \quad (1.1)$$

**Пример 3.** Целое число  $1\ 554\ 985\ 071 = 3 \cdot 3 \cdot 4\ 463 \cdot 38\ 713$  – произведение четырех простых чисел, два из которых совпадают.

**Пример 4.** Целое число  $39\ 616\ 304 = 2 \cdot 13 \cdot 7 \cdot 2 \cdot 23 \cdot 13 \cdot 2 \times 13 \cdot 2 \cdot 7 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 7 \cdot 7 \cdot 13 \cdot 13 \cdot 13 \cdot 23$ .

Порядок записи сомножителей после последнего знака равенства соответствует *канонической форме*.

Для того чтобы представить относительно небольшое число в виде простых сомножителей, достаточно уметь делить числа столбиком. Однако при этом следует придерживаться некоторых простых правил. Для первого деления нужно выбрать наименьшее простое число, большее 1, которое делит исходное число без остатка. Частное от первого деления также нужно разделить с учетом указанных ограничений. Процесс деления продолжаем до тех пор, пока частным не будет 1. Рассмотрим это на примерах.

**Пример 5.** Представить числа 144 и 39 616 304 в виде простых сомножителей. Порядок действий виден на рис. 1.1.

144	2	1-й шаг	39 616 304	2
72	2	2-й шаг	19 808 152	2
36	2		9904 076	2
18	2		4 952 038	2
9	3		2 476 019	7
3	3		353 717	7
1			50 531	13
		...	3 887	13
			299	13
			23	23
			1	

**Рис. 1.1.** Пояснение к разложению чисел на простые сомножители

Таким образом,  $144 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3$ . Результат операций над другим числом был представлен выше в примере 4.

*Свойство 2.* Простых чисел бесконечно много, причем существует примерно  $n / \ln(n)$  простых чисел, меньших числа  $n$ .

*Свойство 3.* Наименьший простой делитель составного числа  $n$  не превышает  $\sqrt{n}$ , поэтому для проверки простоты числа достаточно проверить его делимость на 2 и на все нечетные (а еще лучше простые) числа, не превосходящие  $\sqrt{n}$ ; как видим, данное свойство коррелирует со свойством 1 собственного делителя.

Из соотношения  $n = qr$  натуральных чисел, больших единицы, следует, что либо  $p$ , либо  $q$  принадлежит отрезку от 2 до  $\sqrt{n}$ .

Поиск сомножителей числа  $n$  может вестись, например, перебором всех простых чисел до  $\sqrt{n}$ . Однако если множители – большие простые числа, то на их поиск может потребоваться много времени.

! Сложность решения задачи разложения **больших чисел на простые сомножители**, известной как «*проблема факторизации*», определяет криптоустойчивость некоторых алгоритмов асимметричной криптографии, в частности алгоритма RSA.

*Свойство 4.* Любое четное число, большее 2, представимо в виде суммы двух простых чисел, а любое нечетное, большее 5, представимо в виде суммы трех простых чисел.

*Свойство 5.* Для любого натурального  $n$ , большего 1, существует хотя бы одно простое число на интервале от  $n$  до  $2n$ .

**Определение 7.** Натуральное число  $n$  называется **составным**, если  $n > 1$  и имеет по крайней мере один положительный делитель, отличный от 1 и  $n$ .

Единица не считается ни простым числом, ни составным.

**Пример 6.** Числа 17, 31 – простые, а числа 14, 15 – составные (14 делится на 2 и на 7, 15 делится на 3 и на 5).

Вернемся к собственному делителю.

*Свойство 2 собственного делителя.* Положительный наименьший собственный делитель составного числа  $n$  есть **простое число**.

Так как простое число не делится ни на какое другое, кроме себя самого, очевидный способ проверки числа  $n$  на простоту – разделить  $n$  на все числа  $(n - 1)$  и проанализировать наличие остатка от деления. Этот способ «в лоб» часто реализуется в компьютерных программах. Однако перебор может оказаться достаточно трудоемким, если на простоту нужно проверить число с количеством цифр в несколько десятков.

Существуют правила, способные заметно сократить время вычислений [4].

*Правило 1.* Воспользоваться свойством 3 простых чисел (см. выше).

**Пример 7.** Проверим, является ли число 287 простым числом. Для этого найдем наименьший собственный делитель этого числа: проверяем все простые числа от 2 до  $\sqrt{287}$ , т. е. от 2 до 13 (берется наибольшее простое число, не превышающее значение корня квадратного;  $\sqrt{287} \approx 16,94$ ).

Получаем: ни одно из вышеуказанных простых чисел не является делителем числа 287. Таким образом, число 287 является простым.

*Правило 2.* Если последняя цифра анализируемого числа является четной, то это число заведомо составное.

*Правило 3.* Числа, делящиеся на 5, всегда оканчиваются пятеркой или нулем. Если младшим разрядом анализируемого числа являются 5 или 0, то такое число не является простым.

*Правило 4.* Если анализируемое число делится на 3, то и сумма его цифр тоже обязательно делится на 3.

**Пример 8.** Анализируемое число: 136 827 658 235 479 371. Сумма цифр этого числа равна:  $1 + 3 + 6 + 8 + 2 + 7 + 6 + 5 + 8 + 2 + 3 + 5 + 4 + 7 + 9 + 3 + 7 + 1 = 87$ . Это число делится на 3 без остатка:  $87 = 29 \cdot 3$ . Следовательно, и наше число тоже делится на 3 и является составным.

*Правило 5.* Основано на *свойстве делимости на 11*. Нужно из суммы всех нечетных цифр числа вычесть сумму всех четных его цифр. Четность и нечетность определяется счетом от младшего разряда. Если получившаяся разность делится на 11, то и анализируемое число тоже на него делится.

**Пример 9.** Анализируемое число: 2 576 562 845 756 365 782 383. Сумма его четных цифр (подчеркнуты) равна:  $8 + 2 + 7 + 6 + 6 + 7 + 4 + 2 + 5 + 7 + 2 = 56$ . Сумма нечетных:  $3 + 3 + 8 + 5 + 3 + 5 + 5 + 8 + 6 + 6 + 5 = 57$ . Разность между ними равна 1. Это число не делится на 11, а следовательно, 11 не является делителем анализируемого числа.

*Правило 6.* Основано на *свойстве делимости на 7 и 13*. Нужно разбить анализируемое число на тройки цифр, начиная с младших разрядов. Просуммировать числа, стоящие на нечетных позициях, и вычесть из них сумму чисел на четных. Проверить делимость результата на числа 7 и 13.

**Пример 10.** Анализируемое число: 2 576 562 845 756 365 782 383.

Просуммируем числа, стоящие на нечетных позициях, и вычтем из них сумму чисел на четных:  $(383 + 365 + 845 + 576) - (782 + 756 + 562 + 2) = 67$ . Это число не делится ни на 7, ни на 13, а значит, и делителями заданного числа они не являются.

**Определение 8.** Если два простых числа отличаются на 2, то их называют **числами-близнецами**.

Таких чисел не очень много. Например, ими являются 5 и 7, 29 и 31, 149 и 151.

Всякое натуральное число  $n > 1$  либо является простым числом, либо имеет простой делитель.

Воспользуемся перечисленными свойствами для определения простоты числа 2009. Это число не делится на 2 (так как оно нечетно), не делится также на 3 (сумма его цифр  $2 + 9 = 11$  не делится на 3), не делится и на 5. Воспользуемся далее свойством 6: попробуем разделить 2009 на 7; в результате получается целый результат: 287. Таким образом, получен ответ: число 2009 – составное.

Понятно, что в криптографии используются числа, проверка на простоту которых производится гораздо дольше, и для работы с этими числами требуются специальные программные средства. К вопросу проверки чисел на простоту мы еще вернемся. Здесь же отметим, что первый *алгоритм нахождения простых чисел*, не превышающих  $n$ , был придуман Эратосфеном во II в. до н. э. и известен сейчас как «*решето Эратосфена*». Его суть в последовательном исключении из списка целых чисел от 1 до  $n$  (или из сокращенного диапазона, например от  $m$  до  $n$ ,  $1 < m \leq n$ ) чисел, кратных 2, 3, 5 и другим простым числам, уже найденным «решетом». Как видим, описанное выше свойство 2 простых чисел и положено в основу рассматриваемого алгоритма.

Для нахождения всех простых чисел не больше заданного числа  $n$  в соответствии с «*решетом Эратосфена*» нужно выполнить следующие шаги:

1) выписать подряд все целые числа от двух (либо от  $m$ ) до  $n$  ( $2, 3, 4, \dots, n$ ). Пусть некоторая переменная (например,  $s$ ) изначально равна 2 – первому простому числу;

2) удалить из списка числа от  $2s$  до  $n$ , считая шагами по  $s$  (это будут числа, кратные  $s$ :  $2s, 3s, 4s, \dots$ );

3) найти первое из оставшихся чисел в списке, большее чем  $s$ , и присвоить значению переменной  $s$  это число;

4) повторять шаги 2 и 3, пока возможно.

**Пример 11.** Примем  $n = 15$ .

Шаг 1. Выпишем числа от 2 до 15: 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 11, 12, 13, 14, 15.

Шаг 2. Удалим из списка числа с учетом  $s = 2$ : 2, 3, 5, 7, 9, 11, 13, 15. В этом списке первое число, большее чем  $s = 2$ , это 3. Текущему  $s$  присваивается новое значение:  $s = 3$ .

Шаг 3. Удалим из списка числа с учетом  $s = 3$ : 2, 3, 5, 7, 11, 13, 15. В этом списке первое число, большее чем  $s = 3$ , это 5.

Текущему  $s$  присваивается новое значение:  $s = 5$ .

Шаг 4. Удалим из списка числа с учетом  $s = 5$ : 2, 3, 5, 7, 13. В этом списке первое число, большее чем  $s = 5$ , это 7. Однако в этом списке уже нет чисел, кратных текущему значению  $s$ , т. е. 7.

Таким образом, числа 2, 3, 5, 7, 13 являются простыми в диапазоне от 1 до 15. Как видим, количество таких чисел – 5.

Вспомним свойство 2 *простых чисел* и посмотрим, как оно «работает» для нашего примера. Вычислим  $n / \ln(n) = 15 / \ln 15 \approx 5,5$ . Результат (с учетом округления до целого) близок к истинному: количеству простых чисел от 1 до  $n = 15$ .

**Пример 12.** Найти все простые числа из промежутка [800; 830].

Воспользуемся свойством 3 простых чисел и вычислим  $\sqrt{830} \approx 28,8$ , т. е. меньше 29. Запишем числа из заданного диапазона и удалим последовательно все числа, делящиеся на простые числа от 2 до 28. Такими простыми числами являются: 2, 3, 5, 7, 11, 13, 17, 19, 23. После выполнения всех операций в «решете» останутся числа: 809, 811, 821, 823.

Помимо рассмотренного алгоритма Эратосфена, который является наименее эффективным, в настоящее время разработаны и используются другие алгоритмы. Описание и программную реализацию этих алгоритмов можно найти, например, в известной и популярной у разработчиков криптографических приложений книге [5].

### 1.1.3. Взаимно простые числа и ф-функция

Понятие *делимости чисел* (см. определение 3) является одним из важных в теории чисел. С этим понятием, а также с его производным – *общим делителем* (см. определение 4) связаны другие

важнейшие (в частности, для криптографии) понятия: *наибольшего общего делителя* (НОД) и *взаимно простых чисел*.

**Определение 9.** Наибольшее целое число, которое делит без остатка числа  $a$  и  $b$ , называется *наибольшим общим делителем* этих чисел – НОД ( $a, b$ ).

**Пример 13.** Делителями числа  $a = 24$  являются: 1, 2, 4, 6, 8, 12, 24; делителями числа  $b = 32$  являются: 1, 2, 4, 8, 16, 32. Как видим, НОД ( $24, 32$ ) = 8.

Понятно, что значение НОД можно вычислять для неограниченного ряда чисел.

Простым и эффективным средством вычисления НОД ( $a, b$ ) является *алгоритм Евклида* (примеры его использования приведены в [3]). В основе алгоритма лежит определение 5. В соответствии с этим определением используется цепочка вычислений двумя исходными (начальными) числами  $a$  и  $b$ :

$$a_i = b_i q_i + r_i, \quad 0 \leq r_i \leq b_i. \quad (1.2)$$

При  $i = 0$  в выражении (1.2)  $a_i$  и  $b_i$  соответствуют как раз числам  $a$  и  $b$ . Последний ненулевой остаток ( $r_i, i \geq 0$ ) соответствует НОД ( $a, b$ ).

**Пример 14.** Пусть  $a = 1234$ ,  $b = 54$ . Найти НОД.

$$1234 = 54 \cdot 22 + 46;$$

$$54 = 46 \cdot 1 + 8;$$

$$46 = 8 \cdot 5 + 6;$$

$$8 = 6 \cdot 1 + 2;$$

$$6 = 2 \cdot 3 + 0.$$

Последний ненулевой остаток равен 2, поэтому НОД ( $1234, 54$ ) = 2.

Чтобы найти НОД нескольких чисел (например,  $a, b, c$ ), достаточно найти НОД двух чисел (например, НОД ( $a, b$ ) =  $d$ ), потом НОД полученного (НОД ( $a, b$ )) и следующего числа (НОД ( $c, d$ )), и т. д.

Таким образом, чтобы вычислить НОД  $k$  чисел, нужно последовательно вычислить  $(k - 1)$  НОД. Последнее вычисление дает искомый результат.

**Определение 10.** *Взаимно простыми* являются целые числа, наибольший общий делитель которых равен 1.

**Пример 15.** Взаимно простыми являются числа 11 и 7, 11 и 4, хотя число 4 само по себе не является простым.

**Теорема 1.** Целые числа  $a$  и  $b$  взаимно прости тогда и только тогда, когда существуют такие целые числа  $u$  и  $v$ , что выполняется равенство

$$au + bv = 1. \quad (1.3)$$

**Теорема 2.** Если НОД  $(a, b) = d$ , то справедливо следующее соотношение (соотношение Безу):

$$au + bv = d. \quad (1.4)$$



**Формула (1.4) называется также реализацией «расширенного алгоритма Евклида».** Этот алгоритм состоит из двух этапов: собственно алгоритма Евклида и вычислений на основе обратных подстановок или последовательного выражения остатков в каждом из шагов предыдущего этапа с соответствующим приведением подобных на каждом шаге.

**Пример 16.** Для демонстрации обратимся к примеру 14, который составляет первый из указанных этапов. Ниже приведена табл. 1.1, из которой можно легко понять, как по алгоритму Евклида вычисляются остатки.

Таблица 1.1  
Реализация алгоритма Евклида для примера 14

$1234 = 54 \cdot 22 + 46$	$46 = 1234 - 54 \cdot 22$
$54 = 46 \cdot 1 + 8$	$8 = 54 - 46 \cdot 1$
$46 = 8 \cdot 5 + 6$	$6 = 46 - 8 \cdot 5$
$8 = 6 \cdot 1 + 2$	$2 = 8 - 6 \cdot 1$

Обратные подстановки, или проход вверх, начинаются от записи равенства в нижней строке правого столбца таблицы:  $2 = 8 - 6 \cdot 1$ . Далее вместо цифры 6 подставляется ее значение из равенства строкой выше:  $2 = 8 - (46 - 8 \cdot 5) \cdot 1$  и т. д. Полная цепочка подстановок и преобразований выглядит так:  $2 = 8 - (46 - 8 \cdot 5) \cdot 1 = 8 - 46 + 8 \cdot 5 = 8 \cdot 6 - 46 = (54 - 46) \cdot 6 - 46 = 54 \cdot 6 - 46 \cdot 6 - 46 = 54 \cdot 6 - 46 \cdot 7 = 54 \cdot 6 - (1234 - 54 \cdot 22) \cdot 7 = 54 \cdot 6 - 1234 \cdot 7 + 54 \cdot 154 = 54 \cdot 160 + (-7) \cdot 1234 = 8640 - 8638$ . Из выражения перед последним знаком равенства (выделено) следует, что для нашего примера  $u = -7$  и  $v = 160$  в соответствии с формой записи в выражении (1.4).

Исследованием целых чисел занимался швейцарский математик Леонард Эйлер (Leonard Euler). Один из важных вопросов его исследования: сколько существует натуральных чисел, не превосходящих некоторое число  $n$  и взаимно простых с  $n$ ? Ответ на этот вопрос связан с каноническим разложением числа  $n$  на простые множители (см. выше основную теорему арифметики и пример 4). Так, если

$$n = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}, \quad (1.5)$$

где  $p_1, p_2, \dots, p_n$  – разные простые множители, то число  $\phi(n)$  натуральных чисел, не превосходящих  $n$  и взаимно простых с  $n$ , можно точно определить по формуле

$$\phi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \cdots \left(1 - \frac{1}{p_n}\right). \quad (1.6)$$



**Количество натуральных чисел, не превосходящих  $n$  и взаимно простых с  $n$ , называется функцией Эйлера и обозначается  $\phi(n)$ .**

**Пример 17.** Определить количество натуральных чисел, не превосходящих 12 и взаимно простых с 12.

Взаимно простыми с 12 будут четыре числа 1, 5, 7, 11, т. е.  $\phi(12) = 4$  – получено методом «ручного» подсчета.

Теперь подсчитаем  $\phi(12)$  по формуле (1.5). Вспомнив примеры 4 и 5, запишем каноническое разложение числа 12:  $12 = 2 \cdot 2 \cdot 3 = 2^2 \cdot 3$ , т. е.  $p_1 = 2, p_2 = 3$ . Теперь подсчитаем функцию Эйлера:

$$\phi(12) = 12 \cdot (1 - 1/2) \cdot (1 - 1/3) = 4.$$



**Если  $p$  – простое число, то  $\phi(p) = p - 1$ , если числа  $p$  и  $q$  являются простыми и  $p \neq q$ , то**

$$\phi(p) = (p - 1)(q - 1). \quad (1.7)$$

#### 1.1.4. Модулярная арифметика и обратные числа по модулю

Понятие «**модулярная арифметика**» ввел немецкий ученый К. Ф. Гаусс. В этой арифметике мы интересуемся остатком от деления числа  $a$  на число  $n$  ( $n$  – натуральное число и  $n > 1$ ). Если таким остатком является число  $b$ , то можно записать:

$$a \equiv b \pmod{n}, \text{ или } a \equiv b \text{ mod } n.$$

Такая формальная запись читается как « $a$  сравнимо с  $b$  по модулю  $n$ ».

При целочисленном (в том числе и нулевом) результате деления числа  $a$  на число  $n$  справедливо:  $a = b + kn$ .

**Пример 18.** При  $a = 13$  и  $n = 4$  имеем  $b = 1$ , т. е.  $13 = 1 + 3 \cdot 4$ . Для данного примера справедлив вывод: число 13 по модулю 4 равно 1, или числа 13 и 1 равны по модулю 4.

**Пример 19.** Справедливы следующие сравнения чисел:

$$-5 \equiv 7 \pmod{4} \equiv 11 \pmod{4} \equiv 23 \pmod{4} \equiv 3 \pmod{4}.$$

Иногда  $b$  называют **вычетом** по модулю  $n$ , а числа  $a$  и  $b$  называют **сравнениями** (по модулю  $n$ ).

Модулярная арифметика так же *коммутативна, ассоциативна и дистрибутивна*, как и обычная арифметика. В силу этих свойств сравнения можно почленно складывать, вычитать, умножать, возводить в степень ( $a^m \pmod{n} \equiv (a \pmod{n})^m$ , если  $a \equiv b \pmod{n}$ , то  $a^m \equiv b^m \pmod{n}$ ); другие примеры см. в [3]).

Указанные свойства позволяют упрощать сложность и время выполнения многих вычислений. Криптография использует множество вычислений по модулю  $n$ , потому что задачи типа вычисления *дискретных логарифмов* и квадратных корней очень трудны. Кроме того, с вычислениями по модулю удобнее работать, потому что они ограничивают диапазон всех промежуточных величин и результата.

**Обратные числа в модулярной арифметике.** Традиционное отношение чисел в арифметике: обратное к 7 равно  $7^{-1} = 1/7$ , так как  $7 \cdot (1/7) = 1$ . В модулярной арифметике запись уравнения в виде

$$ax \equiv 1 \pmod{n} \quad (1.8)$$

предусматривает поиск таких значений  $x$  и  $k$ , которые удовлетворяют равенству

$$ax = nk + 1. \quad (1.9)$$

Общая задача решения уравнения (1.8) может быть сформулирована следующим образом: найти такое  $x$ , что

$$1 \equiv ax \pmod{n}. \quad (1.10)$$

Уравнение (1.10) имеет единственное решение, если  $a$  и  $n$  – взаимно простые числа, в противном случае – решений нет.

Уравнение (1.10) можно переписать в ином виде:

$$a^{-1} \equiv x \pmod{n}. \quad (1.11)$$

**Пример 20.** При  $a = 5$  и  $n = 14$  получим  $x = 3/5^{-1} \pmod{14} \equiv 3 \pmod{14}$ , так как  $5 \cdot 3 \pmod{14} \equiv 1$ .

Если НОД  $(a, n) = 1$ , то  $a^{-1}a \equiv 1 \pmod{n}$ , где  $a^{-1}$  – число, обратное  $a$  по модулю  $n$ .

Справедливо также:

$$x^{-1} \equiv y \pmod{n} \Rightarrow y^{-1} \equiv x \pmod{n}. \quad (1.12)$$

В силу приведенных рассуждений и обоснований выражению (1.12) удовлетворяют такие числа, при которых выполняется равенство

$$xy + kn = 1, \quad (1.13)$$

где  $k$  – целое число (результат деления  $xy/n$ ).

Нахождение чисел, обратных по модулю, легко реализуется с помощью расширенного алгоритма Евклида (см. пример 16 и программную реализацию алгоритма в [3]).

**Пример 21.** Решить уравнение  $7y \equiv 1 \pmod{40}$  или  $y^{-1} \equiv 7 \pmod{40}$ .

Находим НОД  $(7, 40)$  – прямая прогонка (алгоритм Евклида):

$$40 = 7 \cdot 5 + 5,$$

$$7 = 5 \cdot 1 + 2,$$

$$5 = 2 \cdot 2 + 1, \text{ т. е. } \text{НОД}(7, 40) = 1.$$

Обратная подстановка (приведение (1.12) к форме (1.13)):

$1 = 5 - 2 \cdot 2 = 5 - 2 \cdot (7 - 5 \cdot 1) = 5 \cdot 3 + 7 \cdot (-2) = (40 - 7 \cdot 5) \cdot 3 + 7 \cdot (-2) = 3 \cdot 40 + 7 \cdot (-17) = kn + xy = 1 \pmod{n}$ , или  $7 \cdot (-17) = 7y$ , так как  $-17 \pmod{40} = 23$ , то  $y = 23$ : число 23 является обратным числу 7 по модулю 40.

**Малая теорема Ферма.** Если  $n$  – простое число, а число  $a$  не кратно  $n$ , то справедливо:

$$a^n \equiv 1 \pmod{n}. \quad (1.14)$$

В соответствии с *обобщением Эйлера* приведенной теоремы, если НОД  $(a, n) = 1$ , то справедливо:

$$a^{\phi(n)} \pmod{n} \equiv 1. \quad (1.15)$$

Последнее выражение можно переписать в следующем виде:

$$a^{-1} \pmod{n} \equiv a^{\phi(n)-1} \pmod{n}. \quad (1.16)$$

**Пример 22.** Найти число, обратное 5 по модулю 7. Число 7 является простым. Поэтому  $\phi(7) = 7 - 1 = 6$ . Теперь с помощью выражения (1.16) получаем:

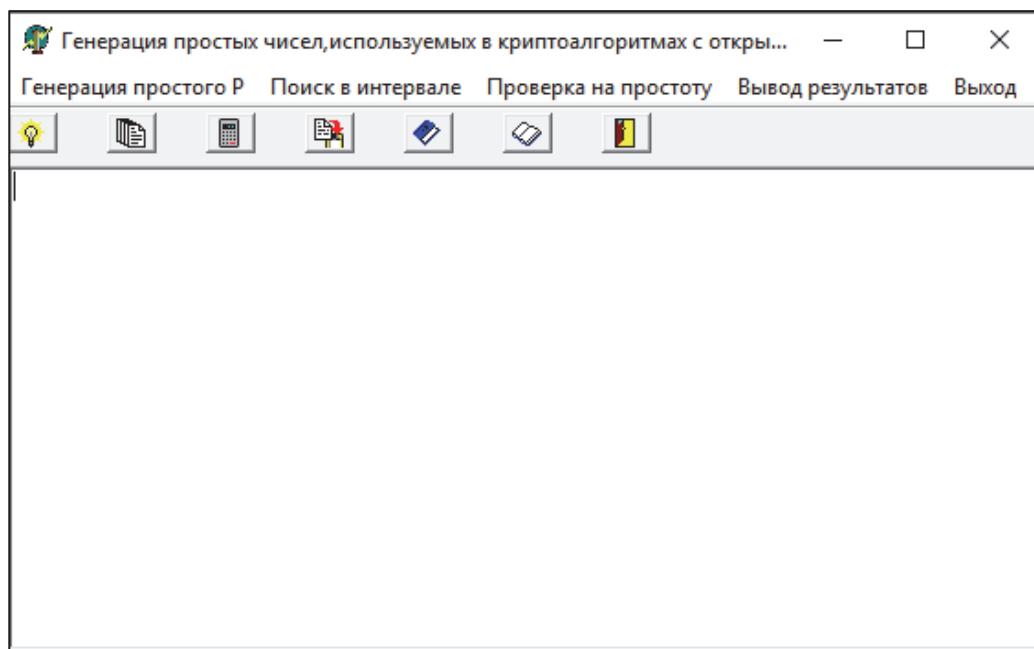
$$5^{6-1} \pmod{7} \equiv 5^5 \pmod{7} \equiv 3.$$

Таким образом,  $5^{-1} \pmod{7} \equiv 3$ , или  $5 \cdot 3 \equiv 1 \pmod{7}$ .

## 1.2. Практическое задание

*Рекомендация!* Перед выполнением практического задания можно познакомиться с функционалом программного средства *L\_PROST*, являющегося приложением на компакт-диске к пособию [6], особенностями выполнения с его помощью операций над простыми числами.

На рис.1.2 представлено основное диалоговое окно программы после запуска исполнительного файла *L\_PROST.EXE*. Как видно, средство позволяет генерировать простые числа, осуществлять проверку чисел на простоту и определять простые числа в заданном интервале.



**Рис. 1.2.** Основное диалоговое окно программного средства *L\_PROST*

1. Используя *L\_PROST*, найти все простые числа в интервале  $[2, n]$ . Значение  $n$  соответствует варианту из табл. 1.2, указанному преподавателем.

Подсчитать количество простых чисел в указанном интервале. Сравнить это число с  $n/\ln(n)$  (см. выше пример 15).

2. Повторить п. 1 для интервала  $[m, n]$ .

Сравнить полученные результаты с «ручными» вычислениями, используя «решето Эратосфена» (см. примеры 11 и 12).

3. Записать числа  $m$  и  $n$  в виде произведения простых множителей (форма записи – каноническая).

4. Проверить, является ли число, состоящее из конкатенации цифр  $m \parallel n$  (табл. 1.2), простым.

5. Найти НОД ( $m, n$ ).

### **Основное задание**

6. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

- вычислять НОД двух либо трех чисел;
- выполнять поиск простых чисел.

7. С помощью созданного приложения выполнить задания по условиям п. 1 и 2.

8. Результаты выполнения работы оформить в виде отчета по установленным правилам.

Таблица 1.2

#### **Варианты задания**

Вариант	$m$	$n$
1	450	503
2	521	553
3	367	401
4	421	457
5	499	531
6	431	471
7	540	577
8	667	703
9	399	433
10	587	621
11	555	591
12	354	397
13	379	411
14	632	663
15	447	477

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. Дать определение понятий: целое число, натуральное число, делимость чисел, собственный делитель, НОД.

2. Сформулировать основную теорему арифметики. Представить примеры ее применения.

3. Пояснить сущность проблемы факторизации и ее связь с прикладной криптографией.
4. Найти НОД:  
пар чисел: 333 и 100; 56 и 200; 99 и 200; 61 и 987; 123 и 456;  
трех чисел: 21, 43, 342; 57, 31, 200; 42, 11, 98.
5. Записать каноническое разложение чисел: 2770; 3780; 6224.
6. Записать соотношение Безу. Показать пример его практического использования.
7. Подсчитать число взаимно простых чисел с числами 2770, 3780, 6224.
8. Сформулировать малую теорему Ферма. Показать примеры ее практического применения.
9. Сформулировать основные свойства модулярной арифметики.
10. Пояснить порядок операций на основе расширенного алгоритма Евклида.
11. Найти числа, обратные к  $a$  по модулю  $n$ :  $a = 41, n = 143$ ;  
 $a = 13, n = 71$ .

## **Лабораторная работа № 2**

### **ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ ШИФРОВ НА ОСНОВЕ ПОДСТАНОВКИ (ЗАМЕНЫ) СИМВОЛОВ**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости подстановочных шифров.
2. Ознакомиться с особенностями реализации и свойствами различных подстановочных шифров на основе готового программного средства (L\_LUX).
3. Разработать приложение для реализации указанных преподавателем методов подстановочного зашифрования/расшифрования.
4. Выполнить исследование криптостойкости шифров на основе статистических данных о частотах появления символов в исходном и зашифрованном сообщениях.
5. Оценить скорость зашифрования/расшифрования реализованных способов шифров.
6. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

#### **2.1. Теоретические сведения**

Лабораторная работа является первой в цикле работ, относящихся к криптографическим шифрам. Основные теоретические сведения и определения из данной предметной области можно найти в [3].

В этом разделе материалов кратко рассмотрим лишь сведения, имеющие непосредственное отношение к цели и задачам лабораторной работы.



**Сущность подстановочного шифрования состоит в том, что исходный текст (из множества  $M$ ) и зашифрованный текст (из множества  $C$ ) основаны на использовании одного и того же или разных алфавитов, а тайной или ключевой информацией является алгоритм подстановки.**

Если исходить из того, что используемые алфавиты являются конечными множествами, то в общем случае каждой букве  $a_x$  алфавита  $A_M$  ( $a_x \in A_M$ ) для создания сообщения  $M_i$  ( $M_i \in M$ ) соответствует буква  $a_y$  или множество букв  $\{A_{xC}\}$  для создания шифртекста  $C_i$  ( $C_i \in C$ ). Важно, чтобы во втором случае любые два множества (например,  $\{A_{xC}\}_b$  и  $\{A_{xC}\}_n$ ,  $b \neq n$ ,  $1 \leq b, n, x, y \leq N$ ,  $N$  – мощность алфавита), используемые для замены разных букв открытого текста, не пересекались:

$$\{A_{xC}\}_b \cap \{A_{xC}\}_n = \emptyset.$$

Если в сообщении  $M_i$  содержится несколько букв  $a_x$ , то каждая из них заменяется на символ  $a_y$  либо на любой из символов  $\{A_{xC}\}$ . За счет этого с помощью одного ключа можно сгенерировать различные  $C_i$  для одного и того же  $M_i$ . Так как множества  $\{A_{xC}\}_b$  и  $\{A_{xC}\}_n$  попарно не пересекаются, то по каждому символу  $C_i$  можно однозначно определить, какому множеству он принадлежит, и, следовательно, какую букву открытого сообщения  $M_i$  он заменяет. В силу этого открытое сообщение восстанавливается из зашифрованного однозначно.

Приведенные утверждения справедливы для следующих типов подстановочных шифров:

- *моноалфавитных* (шифры однозначной замены или простые подстановочные);
- *полиграммных*;
- *омофонических* (однозвучные шифры или шифры многозначной замены);
- *полиалфавитных*.

Кратко поясним особенности указанных шифров.

### 2.1.1. Моналфавитные шифры подстановки

В данных шифрах операция замены производится раздельно над каждым одиночным символом сообщения  $M_i$ . Для наглядной демонстрации шифра простой замены достаточно выписать под заданным алфавитом тот же алфавит, но в другом порядке или,

например, со смещением. Записанный таким образом алфавит называют алфавитом замены.

Максимальное количество ключей для любого шифра этого вида не превышает  $N!$ , где  $N$  – количество символов в алфавите.

Для математического описания криптографического преобразования предполагаем, что зашифрованная буква  $a_y$  ( $a_y \in C_i$ ), соответствующая символу  $a_x$  ( $a_x \in M_i$ ), находится на позиции

$$y \equiv x + k \pmod{N}, \quad (2.1)$$

где  $x, y$  – индекс (порядковый номер, начиная с 0) символа в используемом алфавите;  $k$  – ключ.

Для расшифрования сообщения  $C_i$  необходимо произвести расчеты, обратные выражению (2.1), т. е.

$$x \equiv y - k \pmod{N}. \quad (2.2)$$

Соотношениям (2.1) и (2.2) соответствует классический шифр подстановки – *шифр Цезаря*. Согласно описаниям историка Светония в книге «Жизнь двенадцати цезарей», данный шифр использовался Гаем Юлием Цезарем для секретной переписки со своими генералами (I век до н. э.) [7] (в этой книге любознательный читатель найдет также много исторической информации по криптографии).

**Пример 1.** Имеем открытый текст  $M_i = \langle cba \rangle$ . На основе шифра Цезаря  $C_i = \langle fed \rangle$ .

Здесь  $k = 3$ ,  $N = 26$ . Первый символ открытого текста ( $c$ ) имеет индекс 2 (помним, что начальный символ алфавита ( $a$ ) имеет нулевой индекс). Значит, первый символ шифртекста ( $c$ ) будет иметь индекс  $2 + k = 5$ . А такой индекс в алфавите принадлежит символу  $f$  и т. д.

Известное послание Цезаря *VENI VIDI VICI* (в переводе на русский означает «пришел, увидел, победил»), направленное его другу Аминтию после победы над понтийским царем Фарнаком, выглядело бы в зашифрованном виде так: *YHQL YLGL YLFL*.

Применительно к русскому языку суть его состоит в следующем. Выписывается исходный алфавит (А, Б, ..., Я), затем под ним выписывается тот же алфавит, но с циклическим сдвигом на 3 позиции влево.

Существуют различные модификации шифра Цезаря, в частности, *Атбаш и лозунговый шифр*.

**Атбаш.** В Ветхом Завете существует несколько фрагментов из священных текстов, которые зашифрованы с помощью шифра замены, называемого Атбаш. Этот шифр состоит в замене каждой

буквы другой буквой, которая находится в алфавите на таком же расстоянии от конца алфавита, как оригинальная буква – от начала. Например, в русском алфавите буква А заменяется на Я, буква Б – на Ю и т. д. В оригинальном Ветхом Завете использовались буквы еврейского алфавита. Так, в книге пророка Иеремии слово «Бабель» (Вавилон) зашифровано как «Шешах» [7].



**Одним из существенных недостатков моноалфавитных шифров является их низкая криптостойкость. Зачастую метод криptoанализа базируется на частоте встречаемости букв исходного текста.**

Если в открытом сообщении часто встречается какая-либо буква, то в шифрованном сообщении также часто будет встречаться соответствующий ей символ. Еще в 1412 г. Шихаба ал-Калкашанди в своем труде «Субх ал-Ааша» привел таблицу частоты появления арабских букв в тексте на основе анализа текста Корана. Для разных языков мира существуют подобные таблицы. Так, например, для букв русского алфавита по данным «Национального корпуса русского языка» [8] (корпус – это информационно-справочная система, основанная на собрании текстов на некотором языке в электронной форме; национальный корпус представляет данный язык на определенном этапе (или этапах) его существования и во всем многообразии жанров, стилей, территориальных и социальных вариантов и т. п.). Частота их появления представлена в табл. 2.1.

Существуют подобные таблицы для пар букв (биграмм). Например, часто встречаются биграммами являются «то», «но», «ст», «по», «ен» и т. д. Другой прием взлома шифров основан на исключении возможных сочетаний букв. Например, в текстах (если они написаны без орфографических ошибок) нельзя встретить сочетания «чя», «щы», «ъъ» и т. п. Таблицы с частотами (вероятностями) встречаемости пар и большего числа буквосочетаний существуют для разных алфавитов. Пример использования частотных свойств символов алфавита английского языка для шифроанализа можно найти на страницах 17–19 пособия [9].

**Система шифрования Цезаря с ключевым словом (лозунгом).** Также является *одноалфавитной системой подстановки*. Особенностью этой системы является использование *ключевого слова (лозунга)* для смещения и изменения порядка символов в алфавите подстановки (желательно, чтобы все буквы ключевого слова были различными). Ключевое слово пишется в начале алфавита подстановки.

Таблица 2.1  
Частота появления букв русского языка в текстах

Номер п/п	Буква	Частота, %	Номер п/п	Буква	Частота, %
1	О	10,97	18	Ь	1,74
2	Е	8,45	19	Г	1,70
3	А	8,01	20	З	1,65
4	И	7,35	21	Б	1,59
5	Н	6,70	22	Ч	1,44
6	Т	6,26	23	Й	1,21
7	С	5,47	24	Х	0,97
8	Р	4,73	25	Ж	0,94
9	В	4,54	26	Ш	0,73
10	Л	4,40	27	Ю	0,64
11	К	3,49	28	Ц	0,48
12	М	3,21	29	Щ	0,36
13	Д	2,98	30	Э	0,32
14	П	2,81	31	Ф	0,26
15	У	2,62	32	Ь	0,04
16	Я	2,01	33	Ё	0,04
17	Ы	1,90	-	-	-

**Пример 2.** Для шифра с использованием кодового слова «TABLE» исходный алфавит (первая строка) и алфавит подстановки (вторая строка) выглядят следующим образом:

А В С Д Е Ф Г Н И Ј К Л М Н О Р Q S Т У В В Х Й З  
T A B L E C D F G H I J K M N O P Q R S U V W X Y Z

Если  $M_i = \text{«HELLO»}$ , то  $C_i = \text{«FEJJN»}$ , если же  $M_i = \text{«VENIVIDIVICI»}$ , то  $C_i = \text{«VEMGVGLGVGBG»}$ .

Метод можно видоизменить, если ключевое слово записывать начиная не с первого символа (нулевой индекс) во второй строке, а в соответствии с некоторым числом  $a$ :  $0 \leq a < N$ . Рассмотрим систему на примере.

**Пример 3.** Выберем число  $a = 10$  и слово *information* в качестве ключа.

Ключевое слово записывается под буквами алфавита, начиная с буквы, индекс которой совпадает с выбранным числом  $a$ , как это показано ниже:

0 1 2 3 4 5	10	15	20	25
А В С Д Е F G H И Ј К Л М Н О Р Q S Т У В В X Y Z				
I N F O R M A T				

Как видим, повторяющиеся буквы (I, N и O в конце слова) во второй строке не дублируются. Оставшиеся буквы алфавита подстановки записываются после ключевого слова в алфавитном порядке:

5	10	15
А В С Д Е F G H И Ј К Л М Н О Р Q S Т У В В X Y Z		
L P Q S U V W X Y Z I N F O R M A T B C D E G H J K		

Если  $M_i = \langle\langle VENIVIDIVICI\rangle\rangle$ , то  $C_i = \langle\langle EUOYEYSYEQY\rangle\rangle$ .

Расшифрование сообщения производится по правилу, которое мы рассматривали на примерах, проанализированных выше.

Применяя одновременно операции сложения и умножения по модулю  $n$  над элементами множества (индексами букв алфавита), можно получить *систему подстановок*, которую называют **аффинной системой подстановок Цезаря**. Определим процедуру зашифрования в такой системе:

$$y \equiv ax + b \pmod{N}, \quad (2.3)$$

где  $a$  и  $b$  – целые числа.

При этом взаимно однозначные соответствия между открытым текстом и шифртекстом будут иметь место только при выполнении следующих условий:  $0 \leq a, b < N$ , наибольший общий делитель (НОД) чисел  $a, N$  равен 1, т. е. эти числа являются *взаимно простыми*.

**Пример 4.** Пусть  $N = 26$ ,  $a = 3$ ,  $b = 5$ . Тогда НОД (3, 26) = 1, и мы получаем следующее соответствие между индексами букв:

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$3x + 5$	5	8	11	14	17	20	23	0	3	6	9	12	15	18	21	24	1	4	7	10	13	16	19	22	25	2

Преобразуя числа в буквы английского алфавита, получаем следующее соответствие для букв открытого текста и шифртекста:

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

Для  $M_i = \langle\langle VENIVIDIVICI\rangle\rangle$  получение зашифрованного сообщения в деталях показывает табл. 2.2.

Таблица 2.2

**Иллюстрация получения шифртекста на основе аффинной системы подстановок Цезаря**

$M_i$	V	E	N	I	V	I	D	I	V	I	C	I
x	21	4	13	8	21	8	3	8	21	8	2	8
$y = 3x + 5$	16	17	18	3	16	3	14	3	16	3	11	3
$C_i$	Q	R	S	C	Q	C	O	C	Q	C	L	C

Таким образом, зашифрованное сообщение будет таким:  $C_i = \langle\langle QRSCQCOCQCLC\rangle\rangle$ .

Расшифрование основано на использовании соотношения

$$x \equiv a^{-1}(y + N - b) \bmod N, \quad (2.4)$$

где  $a^{-1}$  – обратное к  $a$  число по модулю  $N$ , т. е. оно удовлетворяет уравнению  $aa^{-1} \equiv 1 \bmod N$ .

### 2.1.2. Полиграммные шифры

В таких шифрах одна подстановка соответствует сразу нескольким символам исходного текста.

Первым известным шифром этого типа является *шифр Порты* [10]. Шифр представляется в виде таблицы. Наверху горизонтально и слева вертикально записывается стандартный алфавит. В ячейках таблицы записываются числа в определенном порядке. Одним

из возможных вариантов такой таблицы для алфавита русского языка будет показанный ниже фрагмент таблицы (табл. 2.3).

Таблица 2.3  
**Фрагмент шифра Порты для алфавита русского языка,  
состоящего из 33 букв**

	А	Б	В	...	Э	Ю	Я
А	001	002	003	...	031	032	033
Б	034	035	036	...	064	065	066
В	067	068	069	...	097	098	099
Г	100	101	102	...	130	131	132
...	...	...	...	...	...	...	...
Ю	1024	1025	1026	...	1054	1055	1056
Я	1057	1058	1059	...	1087	1088	1089

Шифрование выполняется парами букв исходного сообщения. Первая буква пары указывает на строку, вторая – на столбец. В случае нечетного количества букв в сообщении  $M_i$  к нему добавляется вспомогательный символ, например «А».

**Пример 5.** Исходное сообщение  $M_i = \langle\langle АВВА\rangle\rangle$ . Сообщение состоит из двух пар (биграмм): АВ и ВА – и будет зашифровано так: 003 067.

Другими известными полиграфическими шифрами являются **шифр Плейфера** и **шифр Хилла** [10].

С точки зрения криптостойкости рассматриваемый тип шифров имеет преимущества передmonoалфавитными шифрами. Это связано с тем, что, во-первых, распределение частот групп букв значительно более равномерное, чем отдельных символов. Во-вторых, для эффективного частотного анализа требуется больший размер зашифрованного текста, так как число различных групп букв значительно больше, чем мощность алфавита.

### 2.1.3. Омофонические шифры

*Омофонические шифры* (*омофоническая замена*), или *однозвуковые шифры подстановки*, создавались с целью увеличить сложность частотного анализа шифртекстов путем маскировки реальных частот появления символов текста с помощью *омофонии*.

В 1401 г. Симеоне де Крема стал использовать таблицы омофонов для сокрытия частоты появления гласных букв в тексте при помощи более чем одной подстановки. Такие шифры позже стали называться **шифрами многозначной замены**, или **омофонами** (от греч. *homos* – одинаковый и *phone* – звук; слова, которые звучат одинаково, но пишутся по-разному и имеют разное значение; очень много подобных слов содержит английский язык). Они получили развитие в XV в. В книге «Трактат о шифрах» Леона Баттисты Альберти (итальянский ученый, архитектор, теоретик искусства, секретарь папы Климентия XII), опубликованной в 1466 г. [11], приводится описание шифра замены, в котором каждой букве ставится в соответствие несколько эквивалентов, число которых пропорционально частоте встречаемости буквы в открытом тексте  $M_i$ . В этих шифрах буквы исходного алфавита соответствуют более чем одному символу из алфавита замены. Обычно символам исходного текста с наивысшей частотой дают большее количество эквивалентов, чем более редким символам. Таким образом, распределение частоты становится более равномерным, сильно затрудняя частотный анализ.

В табл. 2.4 представлен фрагмент таблицы подстановок для алфавита русского языка [12].

Таблица 2.4  
**Фрагмент таблицы подстановок для системы омофонов**

Номер п/п	А	Б	В	...	М	...	О	...	Р	...	Я
1	311	128	175	...	037	...	248	...	064	...	266
2	357	950	194	...	149	...	267	...	189	...	333
...	...	...	...	...	...	...	...	...	...	...	...
16	495	990	199	...	349	...	303	...	374	...	749
...	...	-	...	...	...	...	...	...	...	...	...
20	519	-	427	...	760	...	306	...	469	...	845
...	...	-	...	...	...	...	...	...	...	-	-
32	637	-	524	...	777	...	432	...	554	-	-
...	...	-	...	-	-	-	...	...	...	-	-
45	678	-	644	-	-	-	824	...	721	-	-
...	...	-	-	-	-	-	...	...	...	-	-

Окончание табл. 2.4

Номер п/п	А	Б	В	...	М	...	О	...	Р	...	Я
47	776	-	-	-	-	-	828	...	954	-	-
...	...	-	-	-	-	-	...	-	-	-	-
80	901	-	-	-	-	-	886	-	-	-	-
...	-	-	-	-	-	-	...	-	-	-	-
110	-	-	-	-	-	-	903	-	-	-	-

При шифровании символ исходного сообщения заменяется на любую подстановку из «своего» столбца. Если символ встречается повторно, то, как правило, используют разные подстановки. Например, исходное сообщение «АБРАМОВ» после зашифрования может выглядеть так: «357 990 374 678 037 828 175» [12].

Заметным вкладом греческого ученого Энея Тактика в криптографию является предложенный им так называемый *книжный шифр* [11, 12]. После Первой мировой войны книжный шифр приобрел иной вид. Шифrozамена для каждой буквы определялась набором цифр, которые указывали на номер страницы, строки и позиции в строке (вспомните пример использования такого шифра известными героями фильма «Семнадцать мгновений весны»). Даже с формальной стороны отсутствие полной электронной базы изданных к настоящему времени книг делает процедуру взлома шифра практически невыполнимой.

#### 2.1.4. Полиалфавитные шифры

*Полиалфавитные* (или *многоалфавитные*) шифры состоят из нескольких шифров однозначной замены. Выбор варианта алфавита для зашифрования одного символа зависит от особенностей метода шифрования.

**Диск Альберти.** В «Трактате о шифрах» [11] Альберти приводит первое точное описание *многоалфавитного шифра* на основе *шифровального диска* (см. рис. 2.1).

Он состоял из двух дисков – внешнего неподвижного и внутреннего подвижного, на которые были нанесены буквы алфавита. Процесс шифрования заключался в нахождении буквы открытого текста на внешнем диске и замене ее на букву с внутреннего диска, стоящую под ней. После этого внутренний диск сдвигался на одну

позицию, и шифрование второй буквы производилось уже по-новому шифралфавиту. Ключом данного шифра являлся порядок расположения букв на дисках и начальное положение внутреннего диска относительно внешнего.



**Рис. 2.1.** Реплика диска Альберти, используемого Конфедерацией во время Гражданской войны в Америке [13]

**Таблица Трисемуса.** В 1518 г. в развитии криптографии был сделан важный шаг благодаря появлению в Германии первой печатной книги по криптографии. Аббат Иоганнес Трисемус, настоятель монастыря в Вюрцбурге, написал книгу «Полиграфия», в которой он описал ряда шифров, один из которых развивает идею многоалфавитной подстановки. Зашифрование осуществляется так: заготавливается **таблица подстановки** (так называемая «таблица Трисемуса» – таблица со стороной, равной  $N$ , где  $N$  – мощность алфавита), где первая строка – это алфавит, вторая – алфавит, сдвинутый на один символ, и т. д. При зашифровании первая буква открытого текста заменяется на букву, стоящую в первой строке, вторая – на букву, стоящую во второй строке, и т. д. После использования последней строки вновь возвращаются к первой.

**Пример 6.** Рассмотрим процесс зашифрования сообщения  $M_i = \text{«БГТУ»}$ , используя таблицу, фрагмент которой показан на рис. 2.2.

Стрелками на приведенном рисунке показан принцип зашифрования каждого символа открытого текста. Из этого следует, что шифртекст имеет вид:  $C_i = \text{«БДФЦ»}$ .

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
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
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
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
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
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

**Рис. 2.2.** Фрагмент таблицы Трисемуса для алфавита русского языка

В указанной книге Трисемус впервые систематически описал применение шифрующих таблиц, заполненных алфавитом в случайном порядке. Для получения такого шифра подстановки обычно использовались таблица для записи букв алфавита и *ключевое слово* (или фраза). Можно найти определенную аналогию с системой шифрования Цезаря с ключевым словом. В таблицу сначала вписывалось по стрелкам ключевое слово, причем повторяющиеся буквы также отбрасывались. Затем эта таблица дополнялась не вошедшими в нее буквами алфавита по порядку.

Таким образом, ключом в таблицах Трисемуса является ключевое слово и размер таблицы. При шифровании буква открытого текста заменяется буквой, расположенной ниже нее в том же столбце. Если буква текста оказывается в нижней строке таблицы, тогда для шифртекста берут самую верхнюю букву из того же столбца.

Указанный размер таблицы для алфавита русского языка может соответствовать  $4 \times 8$  либо  $8 \times 4$ .

**Пример 7.** Пусть  $M_i = \text{«ПРИШЕЛУВИДЕЛПОБЕДИЛ»}$ , а ключевое слово – «ЦЕЗАРЬ». Используем таблицу  $8 \times 4$  (табл. 2.5).

Таблица 2.5

Ц	Е (Ё)	З	А
Р	Ь	Б	В
Г	Д	Ж	И
Й	К	Л	М
Н	О	П	С
Т	У	Ф	Х
Ч	Ш	Щ	Ъ
Ы	Э	Ю	Я

Следуя вышеуказанному принципу подстановки, получим  $C_i = \text{«ФГМЭЛШИМКЬПФУЖЬКМП»}$ .

**Шифр Виженера.** В 1586 г. французский дипломат Блез Виженер представил перед комиссией Генриха III описание простого, но довольно стойкого шифра, в основе которого лежит таблица Трисемуса.

В этом шифре мы имеем дело с последовательностью сдвигов, циклически повторяющейся. Основная идея заключается в следующем. Создается таблица (таблица Виженера) размером  $N \times N$  ( $N$  – число знаков в используемом алфавите). Эти знаки могут включать не только буквы, но и, например, пробел или иные знаки. В первой строке таблицы записывается весь используемый алфавит. Каждая последующая строка получается из предыдущего циклического сдвига последней на 1 символ влево. Таким образом, при мощности алфавита (английского языка), равной 26, необходимо выполнить последовательно 25 сдвигов для формирования всей таблицы. Более подробное описание шифра можно найти в [3, с. 41–43].

Листинг содержит часть кода, реализующего алгоритм шифрования Виженера.

```
/*
 *Главный цикл, который проходит по входной стороне
 */
foreach (char symbol in input)
{
    /* characters – алфавит
       keyword – ключевое слово
       N – мощность алфавита
       keyword_index – индекс текущей буквы ключевого слова
    */
    int c = (Array.IndexOf(characters, symbol) +
    Array.IndexOf(characters, keyword[keyword_index])) % N;

    /* result – результирующая строка */
    result += characters[c];

    keyword_index++;

    /* циклический проход по ключевому слову */
    if ((keyword_index + 1) == keyword.Length)
        keyword_index=0;
}
```

**Листинг.** Фрагмент кода, реализующего алгоритм шифра Виженера

Следует добавить, что в 1863 г. Фридрих Касиски опубликовал алгоритм атаки на этот шифр, хотя известны случаи взлома шифра некоторыми опытными криптоаналитиками и ранее. В частности, в 1854 г. шифр был взломан изобретателем первой аналитической вычислительной машины Чарльзом Бэббиджем. Этот факт стал известен только в XX в., когда группа ученых разбирала вычисления и личные заметки Бэббиджа [6]. Несмотря на это, шифр Виженера имел репутацию исключительно стойкого к «ручному» взлому еще долгое время. Так, известный писатель и математик Чарльз Доджсон (Льюис Кэрролл) в своей статье «Алфавитный шифр», опубликованной в детском журнале в 1868 г., назвал шифр Виженера невзламываемым. В 1917 г. научно-популярный журнал «Scientific American» также отзывался о шифре Виженера как о неподдающемся взлому [14].

**Роторные машины.** Идеи Альберти и Виженера использовались при создании электромеханических роторных машин первой половины XX в. Некоторые из них применялись в разных странах вплоть до 1980-х гг. В большинстве из них использовались роторы (механические колеса), взаимное расположение которых определяло текущий алфавит шифрозамен, используемый для выполнения подстановки. Наиболее известной из роторных машин является немецкая машина времен Второй мировой войны «Энigma». Более детальному изучению и практическому анализу «Энигмы» далее будет посвящена отдельная лабораторная работа.

К полиалфавитным относится также шифр на основе «одноразового блокнота».

Много полезной информации по рассмотренному классу шифров можно найти в [15].

**!** Существует определенное сходство между подстановочными шифрами и шифрами на основе гаммирования. Последние рассматриваются как самостоятельный класс. Такие шифры схожи с подстановочными (и в определенном плане – с перестановочными) тем, что в обоих случаях можно использовать табличное представление выполняемых операций на основе ключей. В шифрах на основе гаммирования и в подстановочных шифрах при зашифровании происходит подмена одних символов другими.

Гаммирование будем рассматривать и изучать более подробно в лабораторной работе № 6.

## 2.2. Общие сведения о криptoанализе

Данная лабораторная работа посвящена анализу одного из разделов практической криптографии. В связи с этим здесь будет уместно охарактеризовать противоположность криптографии – криptoанализ. Данный термин введен американским криптографом Уильямом Ф. Фридманом в 1920 г.

! **Еще раз вспомним, что криptoанализ – это раздел криптологии, занимающийся методами взлома шифров или методами организации криптографических атак на шифры.**

Кратко проанализируем основные криptoатаки [3, 5, 12, 15].

*Атака с известным шифртекстом* (ciphertext only attack).

Предполагается, что противник знает алгоритм шифрования, у него имеется набор перехваченных шифrogramм, но он не знает секретный ключ.

Разновидности такой атаки:

- *полный перебор ключей* (лобовая атака, brute force attack);
- *атака по словарю*, перебор ключей по словарю (dictionary attack); применяется часто для взлома паролей;
- *частотный криptoанализ* – метод взлома шифра, основывающийся на предположении о существовании зависимости между частотой появления символов алфавита в открытых сообщениях и соответствующих шифрозамен в шифrogramмах (этот вопрос с практической точки зрения мы анализировали при выполнении лабораторной работы № 2 из [2]).

*Атака с выбором шифртекста* (chosen cipher text attack).

Криptoаналитик имеет возможность выбрать необходимое количество шифrogramм и получить соответствующие им открытые тексты. Он также может воспользоваться устройством расшифрования один или несколько раз для получения шифртекста в расшифрованном виде. Используя полученные данные, он может попытаться восстановить секретный ключ.

*Адаптивная атака с выбором шифртекста* (adaptive chosen ciphertext attack). Криptoаналитик имеет возможность выбирать новые шифrogramмы для расшифрования с учетом того, что ему известна некоторая информация из предыдущих сообщений. В некоторых криптографических протоколах при получении шифrogramмы, несоответствующей стандарту (содержащей ошибки), отправитель получает ответное сообщение, иногда с детализированным

описанием этапа проверки и причины возникновения ошибки. Криptoаналитик может использовать эту информацию для последовательной посылки и уточнения параметров криптосистемы.

*Атака с известным открытым текстом* (known plaintext attack). То же, что и предыдущая, но противник для некоторых шифрограмм получает в свое распоряжение соответствующие им открытые тексты.

*Атака с выбором открытого текста* (chosen plaintext attack). Криptoаналитик обладает некоторыми открытыми текстами и соответствующими шифртекстами. Кроме того, он имеет возможность зашифровать несколько предварительно выбранных открытых текстов (до начала атаки).

Разновидности:

- атака на основе получения временного неконфиденциального доступа к шифрующему устройству для получения пар открытых и тайных текстов (известны случаи реализации таких атак спецслужбами);

- атака на основе использования информации о структуре сообщений или стандартных фразах – криptoаналитики из Блетчли-Парка (Bletchley Park) могли определить открытый текст сообщений «Энигмы» (см. далее лабораторную работу № 4) в зависимости от того, когда эти сообщения были посланы и как они подписывались;

- перебор ключей по словарю (dictionary attack) – криptoаналитик шифрует слова и фразы, наличие которых предполагается в шифrogramме, с использованием различных ключей; совпадение зашифрованных слов и фраз с частями шифrogramмы может говорить о взломе ключа.

*Адаптивная атака с выбором открытого текста* (adaptive chosen plaintext attack). Криptoаналитик имеет возможность выбирать новые открытые тексты с учетом того, что ему известна некоторая информация из предыдущих сообщений – он может получить пары «открытое сообщение – шифrogramма», в том числе и после начала атаки.

Разновидности:

- провоцирование противника на использование в сообщениях определенных слов или фраз; придуман англичанами: Королевские военно-воздушные силы Великобритании минировали определенные участки Северного моря, этот процесс был назван «садоводством» (англ. *gardening*); практически сразу после этого немцами

посылались зашифрованные сообщения, включающие слово «мины» и названия мест, где они были сброшены;

- *дифференциальный криptoанализ* – метод вскрытия симметричных блочных шифров (и других криптографических примитивов, в частности хеш-функций), предложен в 1990 г. израильскими специалистами Эли Бихамом и Ади Шамиром и основан на изучении разностей между шифруемыми значениями на различных рундах для пары подобранных открытых сообщений при их зашифровании одним и тем же ключом;

- *интегральный криptoанализ* – аналогичен дифференциальному криptoанализу, но в отличие от него рассматривает воздействие алгоритма не на пару, а сразу на множество открытых текстов; предложен в 1997 г. Ларсом Кнудсеном;

- *линейный криptoанализ* – предложен японским криптологом Мицуру Мацуи в 1993 г.; основан на использовании некоторых *линейных приближений*, которые означают, например, следующее: если выполняется операция XOR над некоторыми битами открытого текста, затем – над некоторыми битами шифртекста, а затем над результатами, то получается бит (или биты), который представляет собой XOR некоторых битов ключа; это и есть линейное приближение, которое может быть верным с некоторой вероятностью;

- *использование открытых ключей в асимметричных системах* – криptoаналитик имеет возможность получить шифртекст, соответствующий выбранному сообщению, на основе открытого ключа.

**Атака на основе связанных ключей** (*related key attack*). Криptoаналитик знает не сами ключи, а некоторые различия (соотношения) между ними; реальные криптосистемы используют разные ключи, связанные известным соотношением, например, для каждого нового сообщения предыдущее значение ключа увеличивается на единицу или преобразуется на основе операции сдвига.

**Атака с выбором ключа** (*chosen key attack*). Криptoаналитик задает часть ключа, а на оставшуюся часть ключа выполняет атаку на основе связанных ключей.

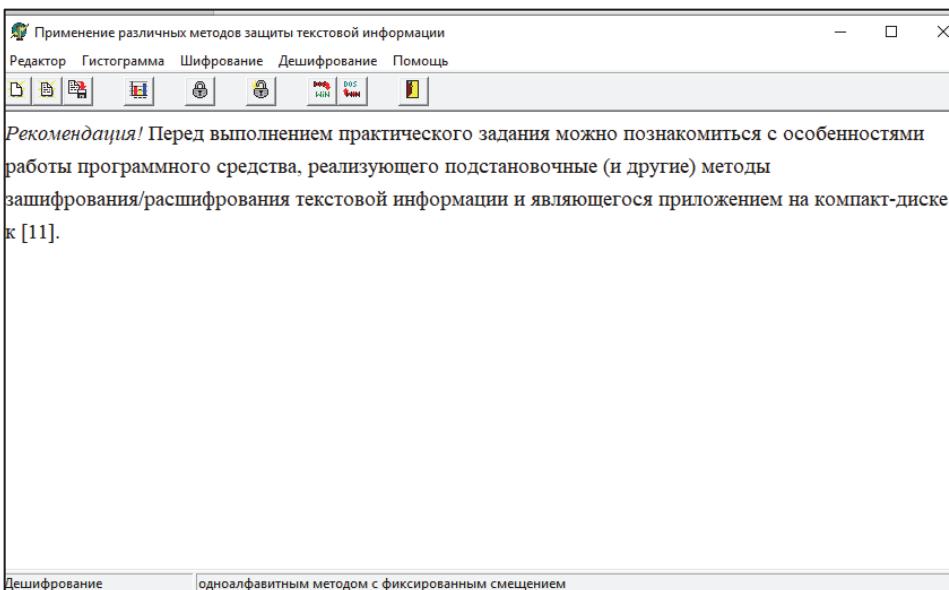
### **2.3. Практическое задание**

**Рекомендация!** Перед выполнением практического задания можно познакомиться с особенностями работы программного

средства *L\_LUX*, реализующего подстановочные (и другие) методы зашифрования/расшифрования текстовой информации и являющегося приложением на компакт-диске к [6].

Программа понятна и проста в использовании с точки зрения интерфейса и функционала. Основная часть окна – текстовый редактор, в котором можно набирать текст либо размещать скопированный фрагмент из другого текстового документа (вставка – Ctrl + V). Здесь же отображается зашифрованный текст, а также сформированные программой распределения частот (в виде гистограмм) для исходного и зашифрованного текстов.

На рис. 2.3 представлено основное диалоговое окно программы после запуска исполнительного файла *L\_LUX.EXE*.



**Рис. 2.3.** Основное диалоговое окно программного средства *L\_LUX*

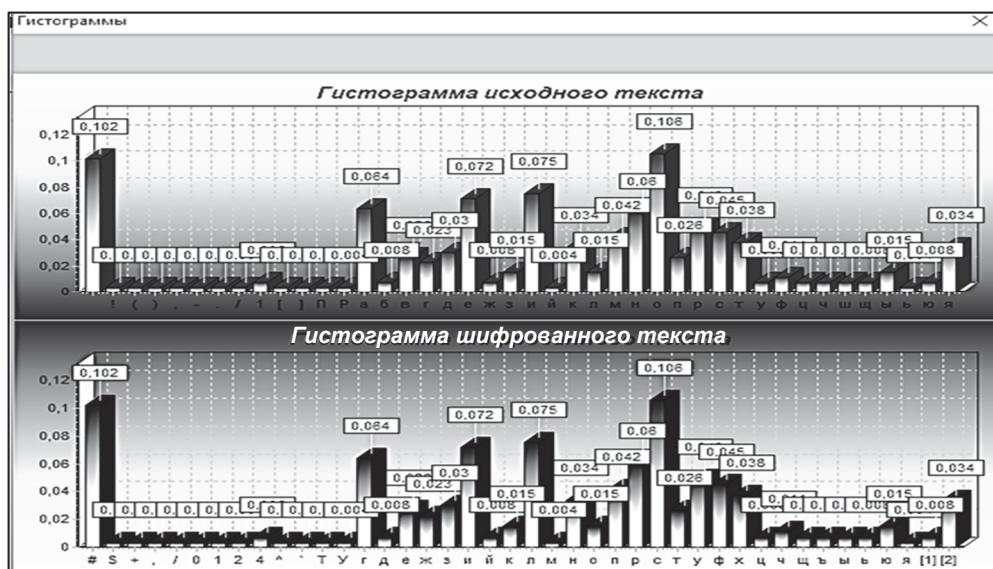
На рис. 2.4 для примера и сравнения приведены гистограммы для исходного (в окне редактора на рис. 2.3) и зашифрованных документов (обратим внимание, что отдельные буквы – строчные и прописные – рассматриваются здесь как разные, что не соответствует традиционному подходу).

### Основное задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

- выполнять зашифрование/расшифрование текстовых документов (объемом не менее 5 тысяч знаков), созданных на основе алфавита

языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы (варианты задания представлены в табл. 2.6);



**Рис. 2.4.** Гистограммы для исходного и зашифрованного текстовых документов

Таблица 2.6

## **Варианты задания**

Вариант	Алфавит	Шифр
1	Белорусский	1. На основе соотношений (2.1) и (2.2); $k = 5$ 2. Виженера, ключевое слово – собственная фамилия
2	Русский	1. На основе аффинной системы подстановок Цезаря; $a = 7, b = 10$ 2. Виженера, ключевое слово – собственная фамилия
3	Английский	1. Шифр Цезаря с ключевым словом, ключевое слово – собственная фамилия 2. Таблица Трисемуса, ключевое слово – собственное имя
4	Немецкий	1. На основе соотношений (2.1) и (2.2); $k = 7$ 2. Таблица Трисемуса, ключевое слово – enigma
5	Польский	1. На основе аффинной системы подстановок Цезаря; $a = 5, b = 7$ 2. Шифр Порты
6	Белорусский	1. На основе соотношений (2.1) и (2.2); $k = 21$ 2. Таблица Трисемуса, ключевое слово – собственное имя

## Окончание табл. 2.6

Вариант	Алфавит	Шифр
7	Русский	1. Шифр Порты 2. Шифр Цезаря с ключевым словом, ключевое слово – собственная фамилия
8	Английский	1. Шифр Цезаря с ключевым словом, ключевое слово – собственная фамилия, $a = 24$ 2. Таблица Трисемуса, ключевое слово – собственное имя
9	Немецкий	1. На основе соотношений (2.1) и (2.2); $k = 7$ 2. Таблица Трисемуса, ключевое слово – enigma
10	Польский	1. На основе соотношений (2.1) и (2.2); $k = 28$ 2. Шифр Порты
11	Белорусский	1. Шифр Цезаря с ключевым словом, ключевое слово – інфарматыка, $a = 2$ 2. Таблица Трисемуса, ключевое слово – собственное имя
12	Русский	1. Шифр Цезаря с ключевым словом, ключевое слово – безопасность 2. Таблица Трисемуса, ключевое слово – безопасность
13	Английский	1. На основе аффинной системы подстановок Цезаря; $a = 6, b = 7$ 2. Таблица Трисемуса, ключевое слово – security
14	Немецкий	1. Виженера, ключевое слово – собственная фамилия 2. Шифр Порты
15	Польский	1. Виженера, ключевое слово – bezpieczeństwo 2. На основе соотношений (2.1) и (2.2); $k = 20$

- сформировать гистограммы частот появления символов для исходного и зашифрованного сообщений;
- оценить время выполнения операций зашифрования/расшифрования (напоминание: *во многих языках программирования есть встроенные методы для замеров времени; при отсутствии такого в используемом языке можно воспользоваться разностью двух дат (например, в миллисекундах: время после выполнения программы – время до начала выполнения преобразования)*).

При анализе полученных гистограмм можно сопоставить полученные данные с аналогичными результатами выполнения лабораторной работы № 2 из [2].

Если указанный в таблице язык исходного текста не известен разработчику программного средства, можно взять документ на требуемом языке и воспользоваться доступным электронным переводчиком (возникающие при этом отдельные семантические неточности не следует считать существенным недостатком выполняемого анализа).

2. Результаты оформить в виде отчета по установленным правилам.

## **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. В чем заключается основная идея криптографических преобразований на основе шифров замены?

2. Привести классификационные признаки и дать сравнительную характеристику разновидностям подстановочных шифров.

3. Сколько разновидностей шифров, подобных шифру Цезаря, можно составить для алфавитов русского и белорусского языков?

4. Найти ключ шифра, с помощью которого получен шифртекст: «*byajhyfwbjyyfzgjcktljdfntkmujsmt*».

5. Расшифровать (с демонстрацией каждого шага алгоритма) текст  $C_i = \text{«qrscqcosqclc»}$ , зашифрованный аффинным шифром Цезаря при  $N = 26$ ,  $a = 3$ ,  $b = 5$ .

6. Зашифровать и расшифровать свою фамилию (на основе кириллицы), используя аффинный шифр Цезаря.

7. Можно ли использовать в качестве ключевого в шифре Виженера слово, равное по длине открытому тексту? Обосновать ответ.

8. По какому признаку можно определить, что текст зашифрован шифром Плейфера?

9. Имеются ли предпочтения в выборе размеров таблицы Три-семуса для виртуального алфавита мощностью  $40: 4 \times 10; 10 \times 4; 5 \times 8; 8 \times 5; 2 \times 20; 20 \times 2$ ?

10. Охарактеризовать основные виды атак на шифры.

11. Сравнить криптостойкость шифра Цезаря и шифра Виженера.

12. Охарактеризовать основные методы взлома подстановочных шифров.

## **Лабораторная работа № 3**

# **ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ ШИФРОВ НА ОСНОВЕ ПЕРЕСТАНОВКИ СИМВОЛОВ**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации перестановочных шифров (работа рассчитана на 4 часа аудиторных занятий).

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости перестановочных шифров.
2. Ознакомиться с особенностями реализации и свойствами различных перестановочных шифров на основе готового программного средства (L\_LUX).
3. Разработать приложение для реализации указанных преподавателем методов перестановочного зашифрования/расшифрования.
4. Выполнить исследование криптостойкости шифров на основе статистических данных о частотах появления символов в исходном и зашифрованном сообщениях.
5. Оценить скорость зашифрования/расшифрования реализованных способов шифров.
6. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### **3.1. Теоретические сведения**

**!** Сущность перестановочного шифрования состоит в том, что исходный текст ( $M$ ) и зашифрованный текст ( $C$ ) основаны на использовании одного и того же алфавита, а тайной или ключевой информацией является алгоритм перестановки.

Шифры перестановки относятся к классу *симметричных*. Элементами текста могут быть отдельные символы (самый распространенный случай), пары, тройки букв и т. д.

Классическими примерами перестановочных шифров являются **анаграммы**. Анаграмма (от греч. *ана* – снова и *gramma* – запись) – литературный прием, состоящий в перестановке букв (или звуков), что в результате дает другое слово или словосочетание, например: проездной – подрезной, листовка – вокалист, апельсин – спаниель.

В классической криптографии шифры перестановки делятся на два подкласса:

- шифры *простой*, или *одинарной*, *перестановки* – при зашифровании символы открытого текста  $M_i$  перемещаются с исходных позиций в новые (в шифртексте  $C_i$ ) один раз;
- шифры *сложной*, или *множественной*, *перестановки* – при зашифровании символы открытого текста  $M_i$  перемещаются с исходных позиций в новые (в шифртексте  $C_i$ ) несколько раз.

### **3.1.1. Шифры одинарной перестановки**

#### **3.1.1.1. Шифры простой перестановки**

Среди шифров рассматриваемого подкласса иногда выделяют *шифры простой перестановки* (или *перестановки без ключа*). Символы открытого текста  $M_i$  перемешиваются по каким-либо правилам. Формально каждое из таких правил может рассматриваться в качестве ключа.

**Пример 1.** Простейшим примером является запись открытого текста в обратной последовательности. Так, если  $M_i$  = «шифр перестановки», то  $C_i$  = «иквонатсереп рфиш». Если переставляются в соответствующем порядке пары букв, то  $C_i$  = «киованстрепе фрши». При более длинных сообщениях можно таким же образом перемещать целые слова или блоки слов.

Подобную перестановку можно трактовать как *транспозицию*.

В общем случае для использования *шифров одинарной перестановки* используется таблица, состоящая из двух строк: в первой строке записываются буквы, во второй – цифры  $J$ . Строки состоят из  $n$  столбцов. Буквы составляют шифруемое сообщение. Цифры  $J = j_1, j_2, \dots, j_n$ , где  $j_1$  – номер позиции в зашифрованном сообщении первого символа открытого текста,  $j_2$  – номер позиции в зашифрованном сообщении второго символа открытого текста и т. д. Таким образом, порядок следования цифр определяется используемым правилом (ключом) перестановки символов открытого текста для получения шифrogramмы.

Если предположить, что некоторое сообщение  $M_i$  состоит из букв от  $m_1$  до  $m_n$ , то рассматриваемую таблицу можно представить как показано ниже (табл. 3.1).

**Таблица 3.1  
Общий вид таблицы для шифра одинарной перестановки**

$m_1$	$m_2$	...	$m_n$
$j_1$	$j_2$	...	$j_n$

В первую строку табл. 3.1 могут записываться также числа в порядке возрастания от 1 до  $n$ . Понятно, что эти числа соответствуют позициям букв в открытом тексте.

Процедура расшифрования также основана на использовании таблиц перестановки. Эти таблицы строятся на основе табл. 3.1.

**Пример 2.** Пусть  $M_i$  = «кибервойны», здесь  $n = 10$ . Далее принимаем правило (ключ) перестановки:  $j_1 = 5, j_2 = 3, j_3 = 1, j_4 = 6, j_5 = 4, j_6 = 2, j_7 = 10, j_8 = 7, j_9 = 8, j_{10} = 9$ .

Составим таблицу для зашифрования сообщения (табл. 3.2) по форме табл. 3.1.

**Таблица 3.2  
Таблица для зашифрования сообщения**

к	и	б	е	р	в	о	й	н	ы
5	3	1	6	4	2	10	7	8	9

Представим эту таблицу только числами (табл. 3.3).

**Таблица 3.3  
Таблица для зашифрования сообщений, представленная  
только цифрами**

1	2	3	4	5	6	7	8	9	10
5	3	1	6	4	2	10	7	8	9

В соответствии с принятым ключом зашифрованное сообщение будет иметь вид:  $C_i$  = «бвиркейнью».

Легко подсчитать, что при отсутствии повторяющихся букв в шифруемом сообщении длиной  $n$  символов всего существует  $n!$  неповторяющихся ключей.

Для расшифрования сообщения, следя логике рассмотренных процедур зашифрования, нам нужно также составить таблицу, первой

строкой которой будет зашифрованный текст (табл. 3.4.). Здесь применяется примерно такой же подход, как и в шифрах подстановки.

**Таблица 3.4**  
**Таблица, где первой строкой является зашифрованный текст**

б	в	и	р	к	е	й	н	ы	о
1	2	3	4	5	6	7	8	9	10

Табл. 3.4 дополним 3-й строкой, числа в столбцах которой соответствуют первой строке табл. 3.3, одновременно составляя неизменную пару: 1 соответствует 3, 2 – 6, и т. д. (см. табл. 3.5).

**Таблица 3.5**  
**Таблица с добавленной 3-й строкой**

б	в	и	р	к	е	й	н	ы	о
1	2	3	4	5	6	7	8	9	10
3	6	2	5	1	4	8	9	10	9

Теперь расшифрованному сообщению «бвиркейнью» будет соответствовать обратная перестановка: символы первой строки табл. 3.5 нужно расположить в порядке в соответствии с 3-й строкой: 1 – «к», 2 – «и» и т. д.

Для использования на практике рассмотренный метод зашифрования/расшифрования не очень удобен. При больших значениях  $n$  приходится работать с таблицами, состоящими из большого числа столбцов. Кроме того, для сообщений разной длины необходимо создавать разные таблицы перестановок.

Следует также отметить сходство рассмотренных алгоритмов зашифрования/расшифрования и алгоритмов перемежения, которые изучались и анализировались в лабораторной работе № 7 из [2].

### **3.1.1.2. Шифры простой блочной перестановки**

Указанные шифры строятся по тем же правилам, что и шифры простой перестановки. Блок должен состоять из 2 или более символов. Если общее число таких символов в сообщении не кратно длине сообщения, то последний блок можно дополнить произвольными знаками.

**Пример 3.** Пусть  $M_i$  = «кибервойны», примем длину блока, равную 2.

Для зашифрования построим таблицу (табл. 3.6).

Таблица 3.6  
Таблица для зашифрования

ки	бе	рв	ой	ны
5	1	4	2	3

В соответствии с табл. 3.6 получим  $C_i = \langle\text{бeойнырвки}\rangle$ . Расшифрование производится по правилам, схожим с правилами для шифров простой перестановки.

### 3.1.1.3. Шифры маршрутной перестановки

Основой современных шифров рассматриваемого типа является геометрическая фигура, обычно прямоугольник или прямоугольная матрица. В ячейки этой фигуры по определенному маршруту (слева направо, сверху вниз или каким-либо иным образом) записывается открытый текст. Для получения шифrogramмы нужно записать символы этого сообщения в иной последовательности, т. е. по иному маршруту (см. аналогию с методами перемежения/деперемежения данных в лабораторной работе № 7 [2]).

**Шифр Скитала (Сцитала).** Известно, что в V в. до н. э. в Спарте существовала хорошо отработанная система секретной военной связи. Для этого использовался специальный жезл «скитала» (греч. σκιτάλη) – первое, вероятно, простейшее криптографическое устройство, реализующее метод перестановки (рис. 3.1).



Рис. 3.1. Скитала [16]

Для зашифрования и расшифрования необходимо было иметь абсолютно одинаковые жезлы. На такой предмет наматывалась пергаментная лента. Далее на эту ленту построчно наносился текст. Для расшифрования ленту с передаваемым сообщением нужно было намотать так же, как и при нанесении открытого текста. Подобным образом работает шифр, который иллюстрирует пример на рис. 3.5 в [3].

Следуя вышеприведенным рассуждениям, можно отождествить Скитала с таблицей размерами  $k \times s$ , где  $k$  – количество столбцов,  $s$  – количество строк. Поскольку при регулярном обмене данными сообщения часто имеют разную длину, то оба этих параметра за неизменяющийся ключ взять неудобно. Поэтому обычно в качестве известного каждой стороне ключа выбирается один из них (часто это  $s$ ), а второй вычисляется на основе известного и длины  $n$  сообщения  $M_i$ :

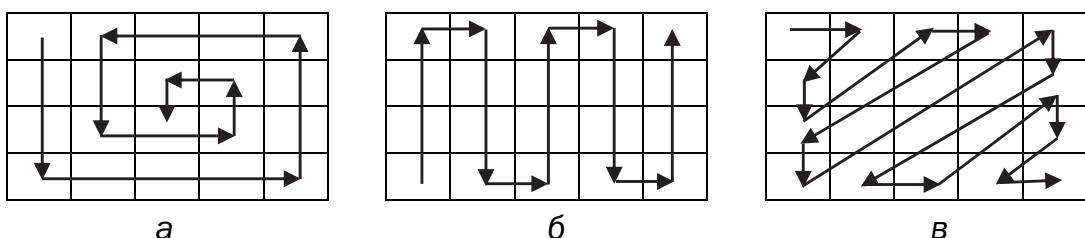
$$k = [(n - 1)/s] + 1.$$

При этом слагаемое в квадратных скобках должно быть целым числом [16].

Нетрудно себе представить аналогию между Скитала и таблицей, которая «намотана» на цилиндр.

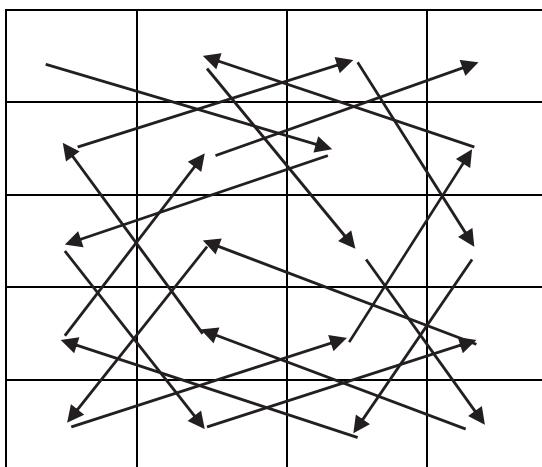
При использовании шифра Скитала для формирования шифртекста сначала выбирается 1-я буква открытого текста, затем  $(k + 1)$ -я буква,  $(2k + 1)$ -я буква и т. д. для некоторого  $k$ , равного числу букв в каждой строке Скиталы. Значение  $k$  является постоянной величиной для данной Скиталы.

**Организация маршрутной перестановки.** Уже упоминавшаяся маршрутная перестановка (записываем сообщение по строкам, считываем – по столбцам матрицы) можно усложнить и считывать не по столбцам, а по спирали (рис. 3.2, *a*), зигзагом (рис. 3.2, *б*), змейкой (рис. 3.2, *в*) или каким-то другим способом. Такие способы шифрования несколько усложняют процесс, однако усиливают криптостойкость шифра.



**Рис. 3.2.** Графическое представление методов маршрутной перестановки:  
а – по спирали; б – зигзагом; в – змейкой

Маршруты могут быть значительно более изощренными. Например, обход конем шахматной доски таким образом, чтобы в каждой клетке конь побывал один раз. Один из таких маршрутов был найден Л. Эйлером в 1759 г. Для примера на рис. 3.3 показан такой маршрут для обхода таблицы размером  $5 \times 4$ .



**Рис. 3.3.** Пример маршрута «обход конем»

Не менее занимательным и не менее сложным является организация маршрутов на основе «магических квадратов» – квадратных матриц со вписанными в каждую клетку неповторяющимися последовательными числами от 1, сумма которых по каждому столбцу, каждой строке и каждой диагонали дает одно и то же число.

Создание новых оригинальных маршрутов приветствуется и поощряется при выполнении данной лабораторной работы.

#### **3.1.1.4. Шифр вертикальной перестановки**

Данный шифр является разновидностью шифра маршрутной перестановки. К особенностям вертикального шифра можно отнести следующие:

- количество столбцов в таблице фиксируется и определяется длиной ключа;
- маршрут вписывания: слева направо, сверху вниз;
- шифrogramма выписывается по столбцам в соответствии с их нумерацией (ключом).

Ключ может задаваться в виде текста (слова или словосочетания). Лексикографическое местоположение символов в ключевом выражении определяет порядок считывания столбцов.

**Пример 4.** Ключом является слово «крипто». Это означает, что количество столбцов  $k$  в таблице должно быть равно длине ключа, т. е. 6. Если вспомним порядок букв из ключевого слова в алфавите, то последовательность считывания столбцов будет следующей: 2, 5, 1, 4, 6, 3.

Необходимо зашифровать сообщение  $M_i$  = «шифр вертикальной перестановки»;  $n = 30$ .

Строим основную таблицу  $5 \times 6$  (табл. 3.7), в которую по строкам будет записано исходное сообщение.

Таблица 3.7

**Таблица с исходным сообщением**

к	р	и	п	т	о
2	5	1	4	6	3
ш	и	ф	р		в
е	р	т	и	к	а
л	ь	н	о	й	
п	е	р	е	с	т
а	н	о	в	к	и

Считывая информацию из таблицы по столбцам в соответствии с ключом, получим шифrogramму  $C_i$  = «фтирошелпава тириоеви-ръен кйск».

### 3.1.2. Шифры множественной перестановки

Особенностью шифров данного подкласса является минимум двукратная перестановка символов шифруемого сообщения. В простейшем случае это может задаваться перемешиванием не только столбцов (как в примере 4), но и строк. Таким образом, этот случай соответствует использованию двух основных ключей: длина одного из них равна числу столбцов, другого – числу строк. К ключевой информации мы можем относить также способы вписывания сообщения и считывания отдельных символов из текущего столбца матрицы.

**Пример 5.** Предположим, что (в продолжение к последнему примеру) вторым ключом будет «слово», или 5, 2, 3, 1, 4 (одинаковым буквам «о» мы присвоили последовательные числа).

Предыдущая таблица несколько видоизменится и примет следующий вид (табл. 3.8).

Таблица 3.8

**Видоизмененная таблица**

Ключи		к	р	и	п	т	о
		2	5	1	4	6	3
с	5	ш	и	ф	р		в
л	2	е	р	т	и	к	а
о	3	л	ь	н	о	й	
в	1	п	е	р	е	с	т
о	4	а	н	о	в	к	и

Для удобства отсортируем последовательно строки в соответствии с ключом (табл. 3.9).

Таблица 3.9

**Отсортированная таблица**

Ключи		к	р	и	п	т	о
		2	5	1	4	6	3
в	1	п	е	р	е	с	т
л	2	е	р	т	и	к	а
о	3	л	ь	н	о	й	
о	4	а	н	о	в	к	и
с	5	ш	и	ф	р		в

И столбцы – в соответствии с ключевым словом «слово» (табл. 3.10).

Таблица 3.10

**Таблица в соответствии с ключевым словом**

Ключи		и	к	о	п	р	т
		1	2	3	4	5	6
в	1	е	т	р	е	с	п
л	2	и	а	т	р	к	е
о	3	о		н	ь	й	л
о	4	в	и	о	н	к	а
с	5	р	в	ф	и		ш

Получим итоговую шифrogramму  $C_i = \text{«еиоврта ивртноферь-ниский пелаш»}$ .

Шифры гаммирования рассматриваются как самостоятельный класс. Такие шифры схожи с перестановочными тем, что в обоих случаях можно использовать табличное представление выполняемых операций на основе ключей. Вместе с тем шифры гаммирования имеют много общего с подстановочными шифрами, поскольку на самом деле при зашифровании происходит подмена одних символов другими.

Полезную информацию о классе рассмотренных шифров можно найти в [17, 18].

### **3.2. Практическое задание**

*Рекомендация!* Перед выполнением практического задания целесообразно освежить практические навыки использования и особенности функционирования программного средства *L\_LUX*, реализующего перестановочные (и другие) методы зашифрования/расшифрования текстовой информации и являющегося приложением на компакт-диске к [6].

Обратим внимание на использование «горячих» клавиш для реализации некоторых операций:

- Ctrl + F3 – зашифрование на основе простой перестановки;
- Shift + F3 – расшифрование на основе простой перестановки;
- Shift + Ctrl + F1 – вывод гистограмм (частотных параметров символов) исходного и зашифрованного сообщений;
- Shift + Ctrl + F2 – вывод гистограмм (частотных параметров символов) зашифрованного и расшифрованного сообщений.

#### **Основное задание**

1. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

- выполнять зашифрование/расшифрование текстовых документов (объемом не менее 500 знаков), созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы (варианты задания в табл. 3.11);

Таблица 3.11

**Варианты задания**

Вариант	Алфавит	Шифр
1	Белорусский	1. Маршрутная перестановка (маршрут: запись – по строкам, считывание – по столбцам таблицы; параметры таблицы – по указанию преподавателя) 2. Множественная перестановка, ключевые слова – собственные имя и фамилия
2	Русский	1. Маршрутная перестановка (маршрут – по спирали; параметры таблицы – по указанию преподавателя) 2. Множественная перестановка, ключевые слова – собственные имя и фамилия
3	Английский	1. Маршрутная перестановка (маршрут – зигзагом; параметры таблицы – по указанию преподавателя) 2. Множественная перестановка, ключевые слова – собственные имя и фамилия
4	Немецкий	1. Маршрутная перестановка (маршрут – змейкой; параметры таблицы – по указанию преподавателя) 2. Множественная перестановка, ключевые слова – собственные имя и фамилия
5	Польский	1. Маршрутная перестановка (маршрут: запись – по столбцам, считывание – по строкам таблицы; параметры таблицы – по указанию преподавателя) 2. Множественная перестановка, ключевые слова – собственные имя и фамилия

- формировать гистограммы частот появления символов для исходного и зашифрованного сообщений;
- оценивать время выполнения операций зашифрования/расшифрования (напоминание: *во многих языках программирования есть встроенные методы для замеров времени; при отсутствии такого в используемом языке можно воспользоваться разностью двух дат (например, в миллисекундах: время после выполнения программы – время до начала выполнения преобразования)*).

Ниже представлен (листинг) пример кода программы (класса *Encryption*) для зашифрования сообщения на основе табличного представления сообщений.

```
class Encryption{
{
    private int[] key = null;
    public void SetKey(string[] _key)
    {
        key=new int[_key.Length];
        for(int i=0;i<_key.Length;i++)
key[i] = Convert.ToInt32(_key[i]);
    }
    public string Encrypt(string input)
    {
        for(int i=0;i<input.Length % key.Length;i++)
input += input[i];

        string result = "";
        for(int i=0;i<input.Length;i+=key.Length)
        {
            char[] transposition = new
char[key.Length];
            for(int j=0;j<key.Length;j++)
transposition[key[j]-1]=input[i+j];
            for(int j=0;j<key.Length;j++)
result += transposition[j];
        }
        return result;
    }
    public string Decrypt(string input)
    {
        string result = "";
        for(int i=0;i<input.Length;i+=key.Length)
        {
            char[] transposition = new
char[key.Length];
            for(int j=0;j<key.Length;j++)
transposition[j] = input[i + key[j] - 1];
            for(int j=0;j<key.Length;j++) result +=
transposition[j];
        }
        return result;
    }
}
```

**Листинг.** Пример кода программы для зашифрования сообщения на основе табличного представления сообщений

При анализе полученных гистограмм можно сопоставить полученные данные с аналогичными результатами выполнения лабораторной работы № 2 из [2] и лабораторной работы № 2 настоящего пособия.

Если указанный в таблице язык исходного текста не известен разработчику программного средства, можно взять документ на требуемом языке и воспользоваться доступным электронным переводчиком (возникающие при этом отдельные семантические неточности не следует считать существенным недостатком выполняемого анализа).

3. Результаты оформить в виде отчета по установленным правилам.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. В чем заключается основная идея криптографических преобразований на основе шифров перестановки?
2. Привести классификационные признаки и дать сравнительную характеристику разновидностям перестановочных шифров.
3. Сколько разновидностей шифров, подобных шифру Цезаря, можно составить для алфавитов русского и белорусского языков?
4. Охарактеризовать криптостойкость перестановочных и подстановочных шифров.
5. Привести примеры и дать характеристику перестановочным шифрам, не рассмотренным в материалах к данной лабораторной работе.
6. Имеются ли предпочтения в выборе размеров используемой таблицы для перестановочных шифров?
7. Охарактеризовать основные методы взлома перестановочных шифров.

## **Лабораторная работа № 4**

### **ИЗУЧЕНИЕ УСТРОЙСТВА И ФУНКЦИОНАЛЬНЫХ ОСОБЕННОСТЕЙ ШИФРОВАЛЬНОЙ МАШИНЫ «ЭНИГМА»**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации перестановочных шифров (работа рассчитана на 4 часа аудиторных занятий).

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости подстановочно-перестановочных шифров.
2. Изучить структуру, принципы функционирования, реализацию процедур зашифрования сообщений в машинах семейства «Энигма».
3. Изучить и приобрести практические навыки выполнения криптореобразований информации на платформе «Энигма», реализованной в виде симуляторов.
4. Получить практические навыки оценки криптостойкости подстановочных и перестановочных шифров на платформе «Энигма».
5. Результаты выполнения лабораторной работы оформить в виде отчета о проведенных исследованиях, методике выполнения практической части задания и оценке криптостойкости шифров.

#### **4.1. Теоретические сведения**

##### **4.1.1. Краткая историческая информация**

Идея создания шифровального устройства высказана голландцем Гуго Кох де Дельфту (в некоторых источниках – Гуго Александр Кох, Hugo Alexander Koch) еще в 1919 г. В 1920 г. он же изобрел первую роторную шифровальную машинку. Параллельно с этим немец Артур Шербиус (Arthur Scherbius) изучал проблему

криптостойкости (в нашем современном понимании) шифровальных машин. Он же получил патент на такую машину, которую назвали «Энигма» (от греч. *enigma* – загадка). Основная особенность «Энигмы» – все знали в то время алгоритм шифрования, но никто не мог подобрать нужный ключ.

Первая шифровальная машина, «*Enigma A*», появилась на рынке в 1923 г. Это была большая и тяжелая машина со встроенной пишущей машинкой и весом около 50 кг. Вскоре после этого была представлена «*Enigma B*», очень похожая на «*Enigma A*». Вес и размеры этих машин сделали их непривлекательными для использования в военных целях.

По достоинству шифровальную машину оценили в немецкой армии. В 1925 г. ее принял на вооружение сначала военно-морской флот (модель *Funkschlussen C*), а в 1930 г. – Вермахт («*Enigma I*»). Общее количество шифраторов, произведенных до и во время Второй мировой войны, превысило 100 тысяч. Применялись они всеми видами вооруженных сил Германии, а также военной разведкой и службой безопасности.

Идея коллеги А. Шербиуса Вилли Корна (Willi Korn) позволила создать компактную и намного более легкую «*Enigma C*». Особенностью этой модели было наличие ламповой панели. В 1927 г. «*Enigma D*» была представлена и коммерциализирована в нескольких версиях с различными роторами и продана военным и дипломатическим службам многих стран Европы. В «*Enigma D*» было три обычных ротора и один отражатель (рефлектор), которые можно было установить в одном из 26 положений (по числу букв используемого алфавита). Именно эта модель стала основным прототипом многих известных версий машин «Энигма», которые использовала Германия в годы Второй мировой войны.

#### 4.1.2. Конструкция и принцип функционирования «Энигмы»

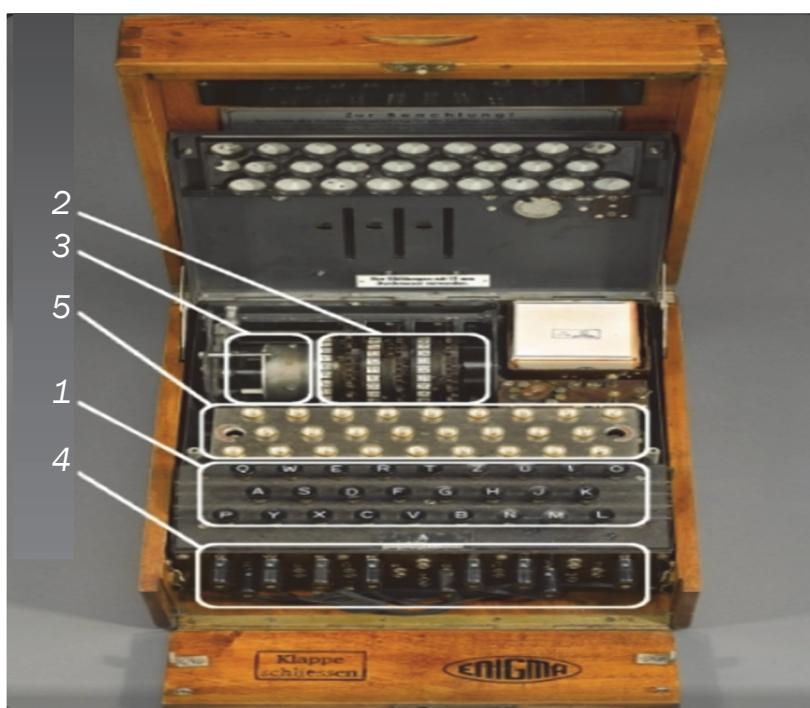
Машина «Энигма» – это электромеханическое устройство. Как и другие роторные машины, «Энигма» состоит из комбинации механических и электрических подсистем.

Механическая часть включает в себя клавиатуру, набор вращающихся дисков – роторов, которые расположены вдоль вала и прилегают к нему, и ступенчатого механизма,двигающего один или несколько роторов при каждом нажатии на клавишу.

Электрическая часть, в свою очередь, состоит из электрической схемы, соединяющей между собой клавиатуру, коммутационную панель, лампочки и роторы (для соединения роторов использовались скользящие контакты).

На рис. 4.1 показана фотография одной из моделей «Энигмы» с указанием месторасположения основных модулей машины. Как видно на этом рисунке, «Энигма» состоит из 5 основных блоков:

- панели механических клавиш 1 (дают сигнал поворота роторных дисков);
- трех (или более) роторных дисков 2, каждый имеет контакты по сторонам, по 26 на каждую, которые коммутируют в случайном порядке; по окружности нанесены буквы латинского алфавита либо числа;
- рефлектора 3 (имеет контакты с крайним слева ротором);
- коммутационной панели 4 (служит для того, чтобы дополнительно менять местами электрические соединения (контакты) двух букв);
- панели в виде электрических лампочек 5; индикационная панель с лампочками служит индикатором выходной буквы в процессе шифрования.

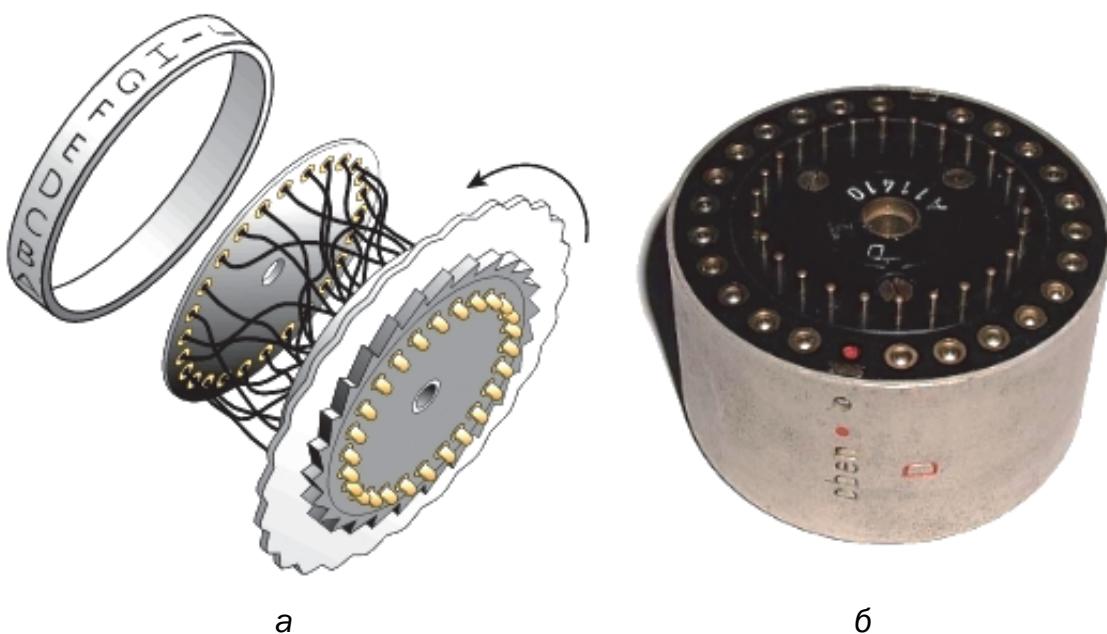


**Рис. 4.1.** Одна из моделей (трехроторная) «Энигмы» [19]:

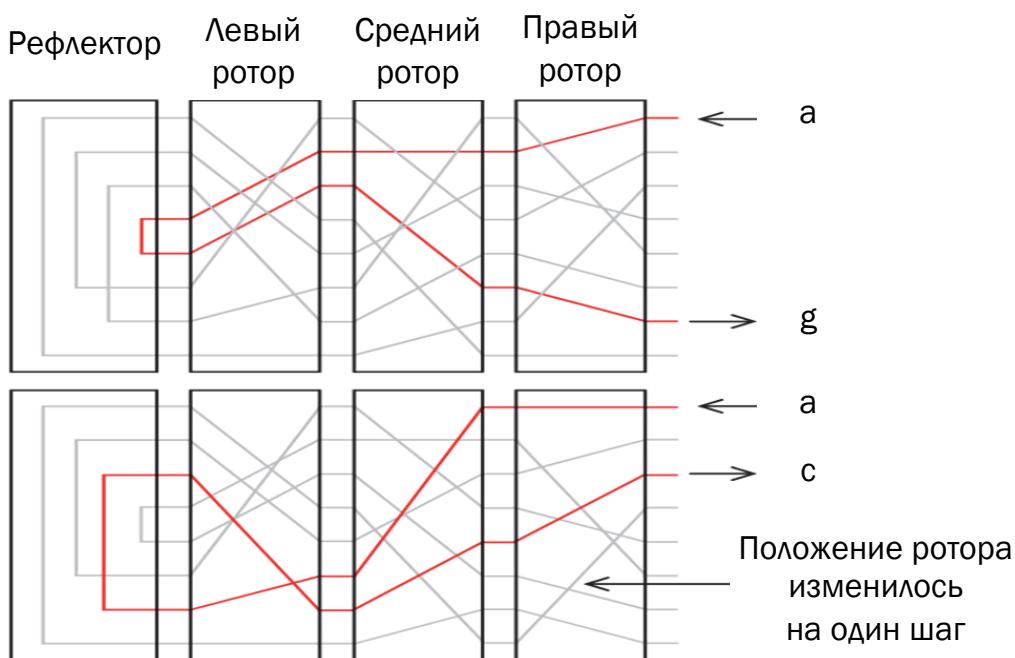
- 1 – панель механических клавиш; 2 – роторные диски; 3 – рефлектор;
- 4 – коммутационная панель; 5 – индикационная панель

Конкретный механизм мог быть разным, но общий принцип был таков: при каждом нажатии на клавишу самый правый ротор сдвигался на одну позицию, а при определенных условиях сдвигались и другие роторы. Движение роторов приводило к различным криптографическим преобразованиям при каждом следующем нажатии на клавишу на клавиатуре, т. е. зашифрование/расшифрование сообщений основано на выполнении ряда замен (подстановок) одного символа другим. Идея А. Шербиуса состояла в том, чтобы добиться этих подстановок электрическими связями.

Механические части двигались и, замыкая контакты, образовывали меняющийся электрический контур. При нажатии на клавишу клавиатуры контур замыкается, ток проходит через созданную (для зашифрования/расшифрования одного конкретного символа сообщения) цепь и в результате включает одну из набора лампочек, отображающую искомую букву шифртекста (или расшифрованного сообщения). На рис. 4.2 показаны упрощенная конструкция ротора (а) и рефлектора (б). Замыкание цепи происходило за счет рефлектора. На рис. 4.3 схематично показано, как некоторая буква (например, «а») будет зашифрована другой буквой (например, «г»), а следующая за ней буква сообщения (также «а») – уже буквой «с».

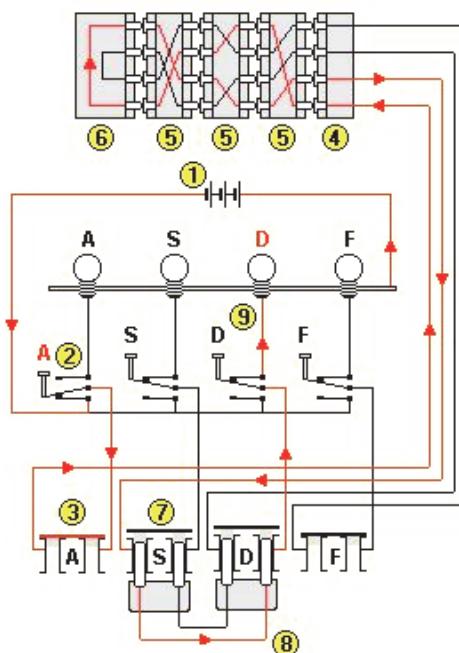


**Рис. 4.2.** Упрощенная конструкция ротора (а) и рефлектора (б) «Энигмы»



**Рис. 4.3.** Пояснение к принципу шифрования путем формирования электрической цепи [20]

Отметим, что на рис. 4.3 электрическая цепь не представлена в виде замкнутой, поскольку не показаны части коммутационной панели и электрическая лампочка. Замкнутые электрические цепи хорошо иллюстрирует рис. 4.4.



**Рис. 4.4.** Пояснение к принципу формирования зашифрованного символа с помощью замкнутой электрической цепи [19]

Замкнутую цепь составляют: батарея 1 (это могут быть и иные источники питания), нажатая двунаправленная буквенная клавиша 2, разъем коммутационной панели 3 (как видим, в одном случае – буква «а» – коммутационного перехода на другую букву нет), входной разъем (входное колесо) роторного модуля 4, роторный модуль 5 (состоит из трех роторов, как в версии «Энигмы» для Вермахта, *Wehrmacht Enigma M3*, или четырех – в версии «Энигмы» для военно-морского флота, *Kriegsmarine Enigma M4*), рефлектор 6. Последний возвращает ток (цепь) по другому пути через те же узлы, «зажигая» на ламповой панели букву «D», к другому полюсу батареи. Обратим внимание, что обратная часть цепи уже проходит с учетом выполненной коммутации (7 и 8).

Отметим также, что клавиатура соответствовала немецкой раскладке *QWERTZ*.

#### 4.1.3. Шифры «Энигмы»

Во время Второй мировой войны немецкие операторы использовали специальную (тайную) шифровальную книгу для установки роторов и настроек колец.

Операторы «Энигмы» (шифровальщики и дешифровальщики) выполняли следующие основные операции.

##### **Пример 1.**

Зашифрование сообщения.

1. Установить начальную стартовую позицию роторов (предположим, их 3) согласно текущей кодовой таблице (коду дня), например *WZA*.

2. Выбрать случайный ключ сообщения, например *SXT*. Затем оператор устанавливал роторы в стартовую позицию *WZA*.

3. Зашифровать ключ сообщения *SXT*. Предположим, что в результате зашифрования ключа получится *UHL*.

4. Далее операторставил ключ сообщения (*SXT*) как начальную позицию роторов и зашифровывал собственно сообщение. После этого он отправлял стартовую позицию (*WZA*) и зашифрованный ключ (*UHL*) вместе с сообщением.

Расшифрование сообщения.

1. Установить стартовые позиции роторов в соответствии с первой трехграммой (*WZA*).

2. Расшифровать вторую треграмму (*UHL*) и извлечь исходный ключ (*SXT*).

3. Далее получатель использовал этот ключ как стартовую позицию для расшифрования шифртекста. Обычно срок действия ключей составлял одни сутки.

### **Пример 2.**

Зашифрование сообщения.

1. Установить стартовую позицию роторов согласно коду дня.

Например, если код был «*HUA*», роторы должны быть инициализированы на «*H*», «*U*» и «*A*» соответственно.

2. Выбрать случайный код с тремя буквами, например *ACF*.

3. Зашифровать текст «*ACFACF*» (повторный код), используя начальную установку роторов шага 1. Например, предположим, что зашифрованный код – «*OPNABT*».

4. Установить стартовые позиции роторов к *OPN* (половина зашифрованного кода).

5. Присоединить зашифрованные шесть букв, полученных на шаге 2 (*OPNABT*), в конец к начальному сообщению.

6. Зашифровать сообщение, включая код с 6 буквами. Передать зашифрованное сообщение.

Расшифрование сообщения.

1. Получить сообщение и отделить первые шесть букв.

2. Установить стартовую позицию роторов согласно коду дня.

3. Расшифровать первые шесть букв сообщения, используя начальную установку шага 2.

4. Установить позиции роторов на первую половину расшифрованного кода.

5. Расшифровать сообщение (без первых шести букв).

Военная модель Энигмы использовала только 26 букв. Прочие символы заменялись редкими комбинациями букв. Пробел пропускался либо заменялся на «*X*». Символ «*X*» также использовался для обозначения точки либо конца сообщения. Некоторые особые символы использовались в отдельных вооруженных частях, например, *Wehrmacht* заменял запятую двумя символами «*ZZ*», вопросительный знак – словом «*FRAGE*» либо буквосочетанием «*FRAQ*», а у *Kriegsmarine M4* запятой соответствовала буква «*Y*».



**Как мы отмечали выше, «Энигма» строится на основе подстановочных шифров, подобных шифру Цезаря, в котором, как известно, ключ сообщения, который должен знать получатель, – это просто смещение между двумя алфавитами. Принято считать, что в основе шифра «Энигмы» лежит динамический шифр Цезаря.**

Более сложная система использует случайный ряд символов для нижнего алфавита. Принцип, положенный в основу этой «случайности», имеет много общего с перестановочными шифрами. Например, ниже показан принцип подстановки, основанный на взаимной перестановке во втором (нижнем) алфавите в 13 парах символов, расположенных случайным образом:

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

Этот принцип случайности использовался и при изготовлении роторов и рефлекторов для «Энигмы». Всего за время Второй мировой войны немцами было изготовлено восемь роторов и четыре рефлектора, но одновременно могло использоваться ровно столько, на сколько была рассчитана машина.

Техническую спецификацию на все произведенные роторы и рефлекторы можно найти в [13, 22]. Ниже на рис. 4.5 и 4.6 представлены спецификации соответственно на роторы и на рефлекторы.

Достаточно подробная информация об основных особенностях и обстоятельствах патентования, разработки и сферах использования практически всех (или большинства) версий «Энигмы» содержится в [23].

INPUT	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
Rotor I	E	K	M	F	L	G	D	Q	V	Z	N	T	O	W	Y	H	X	U	S	P	A	I	B	R	C	J
Rotor II	A	J	D	K	S	I	R	U	X	B	L	H	W	T	M	C	Q	G	Z	N	P	Y	F	V	O	E
Rotor III	B	D	F	H	J	L	C	P	R	T	X	V	Z	N	Y	E	I	W	G	A	K	M	U	S	Q	O
Rotor IV	E	S	O	V	P	Z	J	A	Y	Q	U	I	R	H	X	L	N	F	T	G	K	D	C	M	W	B
Rotor V	V	Z	B	R	G	I	T	Y	U	P	S	D	N	H	L	X	A	W	M	J	Q	O	F	E	C	K
Rotor VI	J	P	G	V	O	U	M	F	Y	Q	B	E	N	H	Z	R	D	K	A	S	X	L	I	C	T	W
Rotor VII	N	Z	J	H	G	R	C	X	M	Y	S	W	B	O	U	F	A	I	V	L	P	E	K	Q	D	T
Rotor VIII	F	K	Q	H	T	L	X	O	C	B	J	S	P	D	Z	R	A	M	E	W	N	I	U	Y	G	V
Beta rotor	L	E	Y	J	V	C	N	I	X	W	P	B	Q	M	D	R	T	A	K	Z	G	F	U	H	O	S
Gamma rotor	F	S	O	K	A	N	U	E	R	H	M	B	T	I	Y	C	W	L	Q	P	Z	X	V	G	J	D

Рис. 4.5. Спецификация на роторы «Энигмы»

reflector B	(AY) (BR) (CU) (DH) (EQ) (FS) (GL) (IP) (JX) (KN) (MO) (TZ) (VW)
reflector C	(AF) (BV) (CP) (DJ) (EI) (GO) (HY) (KR) (LZ) (MX) (NW) (TQ) (SU)
reflector B Dünn	(AE) (BN) (CK) (DQ) (FU) (GY) (HW) (IJ) (LO) (MP) (RX) (SZ) (TV)
reflector C Dünn	(AR) (BD) (CO) (EJ) (FN) (GT) (HK) (IV) (LM) (PW) (QZ) (SX) (UY)

**Рис. 4.6.** Спецификация на рефлекторы «Энигмы»

Рассмотрим пример использования приведенных спецификаций.

**Пример 3.** В этом примере мы рассмотрим процедуру зашифрования только одной буквы («G»).

Предположим, что «Энигма» оснащена роторами I, II, III (см. рис. 4.5). Таким образом, правым ротором (R) является III приведенной спецификации. Предположим также, что каждый ротор находится в своем положении A, когда выполняется шифрование. Если взять информацию из рис. 4.5 и 4.6, указав фактическую разводку ротора, это означает, что правый ротор R производит подстановку в соответствии с переставленными буквами исходного алфавита, т. е. буква «G» будет заменена буквой «C»:

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

Центральный ротор (M) или II заменяет букву «C» на букву «D»:

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

Левый ротор (L) или III – соответственно букву «D» на букву «F»:

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

Предположим далее, что используется рефлектор B (спецификация на рис. 4.6 – первая строка):

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

Обратим внимание на то, что рефлектор имеет только 13 соединений, т. е. имеется 13 пар подстановок: A – Y, B – R и т. д. В нашем примере произошла подстановка F → S.

Ток теперь проходит обратный путь через три ротора в последовательности  $L \rightarrow M \rightarrow R$ .

Эффект преобразования левого ротора (обратный):

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
U	W	Y	G	A	D	F	P	V	Z	Б	Е	С	К	М	Т	Н	Х	S	L	R	I	N	Q	О	Ј

соответственно – среднего ротора (обратный):

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	Ј	Р	С	З	W	Р	Л	Ф	Б	Д	К	О	Т	Ү	У	Q	Г	Е	Н	Х	М	І	В	Ѕ	

и, наконец, правого ротора (обратный):

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
T	A	Г	В	Р	С	Д	Q	Е	У	Ф	В	Н	З	Х	Ү	І	Х	Ј	W	Л	Р	К	О	М	

После всех (в данном случае – 7) подстановок буква «G» будет зашифрована буквой «Р».

Процедура расшифрования шифртекстов предусматривала настройку отражателя, роторов и коммутационной панели машины в соответствии с таблицами (книгами) и использованными при зашифровании паролями. Достаточно подробно информацию об этом можно получить в [24], комментарии к функционалу «Энигмы» и ее симулятору – в [25]. Подробное описание кода симулятора на языке Python содержится в книге [26].

#### 4.1.4. Оценка криптостойкости «Энигмы»

Для получения общего представления об особенностях работы криptoаналитиков над шифрами «Энигмы» полезно ознакомиться с содержанием материалов в [27].

Как мы неоднократно подчеркивали, преобразование «Энигмы» для каждой буквы может быть определено математически как результат подстановок. Рассмотрим трехроторную модель «Энигмы». Положим, что символом  $B$  обозначаются операции с использованием коммутационной панели, соответственно, символы  $Re$  – отражателя, а  $L$ ,  $M$  и  $R$  обозначают действия левых, средних и правых роторов соответственно. Тогда процесс зашифрования символа  $m$  с использованием некоторой ключевой информации  $K$  формально можно записать в следующем виде:

$$E_K = f(m, B, Re, L, M, R).$$

Чтобы оценить криптостойкость шифра, нужно учитывать все возможные настройки машины. Для этого необходимо рассмотреть следующие свойства «Энигмы»:

- выбор и порядок роторов;
- разводку (коммутацию) роторов;
- настройку колец на каждом из роторов;
- начальное положение роторов в начале сообщения;
- отражатель;
- настройки коммутационной панели.

Используются различные варианты подсчета всех возможных состояний перечисленных конструктивных модулей машины [28]. К сожалению для немцев, взломщики шифра союзников знали машину, роторы и внутреннюю разводку этих роторов. Поэтому им нужно было учитывать только возможные способы настройки «Энигмы». Такая априорная информация о конструктивных особенностях устройства для шифрования (вспомним об основных постулатах О. Керкгоффса [3]) в нашем случае снижает уровень (теоретический) криптостойкости (до практического). Немецкие криптологии полагали, что один ротор может быть подключен  $4 \cdot 10^{26}$  различными способами. Сочетание трех роторов и отражателя позволяет получить астрономические цифры возможных вариантов подстановок. Для союзников, которые знали конструкции роторов, число различных вариантов существенно уменьшалось.

Рассмотрим пример для трехроторной «Энигмы» Вермахта с отражателем (по умолчанию – В, см. рис. 4.6) и выбором из 5 роторов. Использовались 10 штекерных кабелей на коммутационной панели (количество кабелей, по умолчанию поставляемых с машиной).

Чтобы выбрать 3 ротора из возможных 5, существует 60 комбинаций ( $5 \cdot 4 \cdot 3$ ). Каждый ротор (его внутренняя проводка) может быть установлен в любом из 26 положений. Следовательно, с 3 роторами имеется 17 576 различных положений ротора ( $26 \cdot 26 \cdot 26$ ). Кольцо на каждом роторе содержит маркировку ротора (что здесь неважно) и выемку, которая влияет на шаг перемещения расположенного левее ротора. Каждое кольцо может быть установлено в любом из 26 положений. Поскольку слева от третьего (наиболее левого) ротора нет ротора, на расчет влияют только кольца самого правого и среднего ротора. Это дает 676 комбинаций колец ( $26 \cdot 26$ ).

Коммутационная панель обеспечивает самый большой набор возможных настроек. Для первого кабеля одна сторона может иметь любое из 26 положений, а другая сторона – любое из 25 оставшихся положений (одна буква коммутируется с другой). Однако поскольку комбинация и ее обратная сторона идентичны (AB такая же, как BA), мы должны игнорировать все двойные числа во всех возможных комбинациях для одного кабеля, представляя  $(26 \cdot 25) / (1! \cdot 2^1)$ , или 325 уникальных способов коммутаций одним кабелем. Для двух кабелей –  $(26 \cdot 25)$  комбинаций для первого кабеля и, поскольку два разъема уже используются, то получается  $(24 \cdot 23)$  комбинаций для второго кабеля. Следуя этой простой логике, получается  $(26 \cdot 25 \cdot 24 \cdot 23) / (2! \cdot 2^2) = = 44\ 850$  уникальных способов коммутаций с использованием двух кабелей. Для трех кабелей –  $(26 \cdot 25 \cdot 24 \cdot 23 \cdot 22 \cdot 21) / (3! \times \times 2^3) = 3\ 453\ 450$  комбинаций и т. д. Таким образом, с использованием 10 кабелей на коммутационной панели получаются 150 738 274 937 250 различных комбинаций. Формула, где  $n$  равно количеству кабелей, равна  $26! / (26 - 2n)! \cdot n! \cdot 2^n$ . Численно это дает:  $60 \cdot 17\ 576 \cdot 676 \cdot 150\ 738 \cdot 274 \cdot 937 \cdot 250 = = 107\ 458\ 687\ 327\ 250\ 619\ 360\ 000$ , или  $1,07 \cdot 10^{23}$ .

Таким образом, практически рассматриваемая версия «Энигмы» (3 ротора с выбором из 5 роторов, отражатель В и 10 штекерных кабелей для коммутационной панели) может быть настроена на  $1,07 \cdot 10^{23}$  различных состояний, что сопоставимо с 77-битным криптографическим ключом.

Добавление четвертого ротора (например, для *Naval Enigma M4*) для повышения его криптостойкости было практически бесполезным: неподвижный четвертый ротор «усложнил машину» только в 26 раз и вместе с тонким отражателем мог рассматриваться как настраиваемый отражатель с 26 положениями. Внедрение общего числа роторов в 8 единиц (на *Kriegsmarine M3*), а затем – на четырехроторной версии (*M4*) было гораздо более эффективным шагом. Они увеличили комбинации роторов с 60 до 336.

Оценим далее практический размер криптографического ключа (или его эквивалент) для четырехроторной версии *Kriegsmarine Enigma M4*. Эта машина использует 3 обычных ротора, выбранных из набора из 8. Это, как мы уже отметили, дает 336 комбинаций подключений роторов ( $8 \cdot 7 \cdot 6$ ). *M4* также имела

специальный четвертый ротор, называемый *Beta* или *Gamma* (без кольца), который дает 2 варианта. Они не совместимы с другими роторами и подходят только как четвертый (самый левый) ротор. Четыре ротора могут быть установлены в любом из 456 976 положений ( $26 \cdot 26 \cdot 26 \cdot 26$ ). Рефлектор не меняется. Четвертый ротор был неподвижным. Версия *M4* была снабжена также 10 кабелями для коммутационной панели.

В сумме это дает:  $336 \cdot 2 \cdot 456\,976 \cdot 676 \cdot 150\,738\,274\,937\,250 = 31\,291\,969\,749\,695\,380\,357\,632\,000$ , или  $3,1 \cdot 10^{25}$ , что сопоставимо с 84-битным ключом.

Проблема криптоанализа шифров «Энигмы» была экстраординарной (с учетом электромеханических конструкций устройств для криптоанализа, применяемых в то время). Исчерпывающий поиск всех возможных  $1,07 \cdot 10^{23}$  настроек (атака *brute force*) был невозможен в 1940-х гг., а его сопоставимый 77-битный ключ огромен даже для современных электронных систем. Чтобы дать представление о размере этого числа, представим, что у нас есть  $1,07 \cdot 10^{23}$  листов бумаги толщиной около 1 мм. Из этих листов можно сложить примерно 70 000 000 стопок бумаги, каждая из которых простирается от Земли до Солнца. Кроме того,  $1,07 \cdot 10^{23}$  дюйма равно 288 500 световых лет.

## 4.2. Практическое задание

1. Ознакомиться с функционалом хотя бы одного (по согласованию с преподавателем) симулятора «Энигмы»:

1) симулятор «Энигмы» *M3* (*M3 Enigma Simulator*):

<https://cryptocellar.org/simula/m3/index.html>

2) симулятор «Энигмы» *M4* (*M4 Enigma Simulator*):

<https://cryptocellar.org/simula/m4/index.html>

3) симулятор «Энигмы» Army/Air Force and the Railway:

<https://cryptocellar.org/simula/enigma/index.html>

4) симулятор «Энигмы» для Абвера (*Abwehr Enigma Simulator*):

<https://cryptocellar.org/simula/abwehr/index.html>

5) симулятор «Энигмы» для Тирпица (*T (Tirpitz) Enigma Simulator*):

<https://cryptocellar.org/simula/tirpitz/index.html>

Произвести зашифрование сообщения (собственные имя, отчество, фамилия) при 8–10 различных настройках машины-симулятора.

Оценить частотные свойства символов в шифртекстах и сравнить этот параметр с частотными свойствами символов для исходного текста.

2. Разработать приложение-симулятор шифровальной машины, состоящей из клавиатуры, трех роторов и отражателя. Типы роторов ( $L - M - R$ ) и отражателя  $Re$  следует выбрать из рис. 4.5 и 4.6 в соответствии со своим вариантом, представленным в таблице. Крайний правый столбец этой таблицы показывает, на какое число шагов (букв,  $i$ ) перемещается соответствующий ротор при зашифровании одного (текущего) символа; число 0 означает перемещение соответствующего ротора на один шаг при условии, что расположенный правее ротор совершил один оборот.

#### Варианты задания

Вариант задания	$L$	$M$	$R$	$Re$	$L_i - M_i - R_i$
1	I	II	III	B	0-2-2
2	II	III	V	C	1-2-2
3	III	VII	I	B Dunn	1-0-1
4	IV	III	II	C Dunn	0-0-4
5	I	Beta	Gamma	B	3-1-3
6	II	Gamma	IV	C	1-1-1
7	Beta	Gamma	V	B Dunn	0-2-2
8	V	VI	VII	C Dunn	1-2-2
9	VIII	II	IV	B	1-0-1
10	Gamma	III	Beta	C	0-0-4
11	Beta	VIII	I	B Dunn	3-1-3
12	III	Gamma	V	C Dunn	1-1-2
13	VI	IV	II	B	1-2-2
14	II	Beta	VIII	C	0-2-2
15	VII	Gamma	II	B Dunn	1-2-2

С помощью разработанного приложения зашифровать сообщение в соответствии с п. 1 практического задания, применив не менее 5 вариантов начальных установок роторов.

Оценить криптостойкость вашего варианта машины.

3. Результаты оформить в виде отчета по установленным правилам.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. Дать пояснение к структуре шифровальных машин «Энigma».
2. На основе каких шифров строится машина «Энigma»?
3. Дать пояснение к принципам зашифрования сообщений.
4. Дать характеристику криптостойкости шифровальной машины Энигма.
5. Дать характеристику (с численными оценками) криптостойкости машины-симулятора на основе разработанного приложения.
6. Пояснить основные принципы расшифрования сообщений «Энгмы».
7. Дать предложения по модификации известных аналогов «Энгмы».

# **Лабораторная работа № 5**

## **|| ИССЛЕДОВАНИЕ БЛОЧНЫХ ШИФРОВ**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации блочных шифров (рассчитана на 4 часа аудиторных занятий).

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости блочных шифров.
2. Разработать приложение для реализации указанных преподавателем методов блочного зашифрования/расшифрования.
3. Выполнить анализ криптостойкости блочных шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### **5.1. Теоретические сведения**

#### **5.1.1. Краткая историческая информация и общая характеристика блочных шифров**

В 1972 г. Национальное бюро стандартов США (ныне – Национальный институт стандартов и технологии, National Institute of Standards & Technology – NIST) инициировало программу защиты каналов связи и компьютерных данных. Одна из целей – разработка единого стандарта криптографического шифрования. Основными критериями оценки алгоритма являлись следующие [5]:

- алгоритм должен обеспечить высокий уровень защиты;
- алгоритм должен быть понятен и детально описан;
- криптостойкость алгоритма должна зависеть только от ключа;
- алгоритм должен допускать адаптацию к различным применением;
- алгоритм должен быть разрешен для экспорта.

В качестве начального варианта нового алгоритма рассматривался Lucifer – разработка компании IBM начала семидесятых годов. В основе указанного алгоритма использовались два запатентованных в 1971 г. Хорстом Фейстелем (Horst Feistel) устройства, реализующие различные алгоритмы шифрования, позже получившие название **шифр (сеть) Фейстеля** (Feistel Cipher, Feistel Network). В первой версии проекта Lucifer сеть Фейстеля не использовалась.

После многочисленных согласований, специальных конференций, где рассматривались в основном вопросы криптостойкости алгоритма, подлежащего утверждению в качестве федерального стандарта, в ноябре 1976 г. был утвержден стандарт DES (Data Encryption Standard – стандарт шифрования данных). Предполагалось, что стандарт будет реализовываться только аппаратно.

В 1981 г. ANSI одобрил DES в качестве стандарта для публичного использования (стандарт ANSI X3.92), назвав его алгоритмом шифрования данных (Data Encryption Algorithm – DEA).

В 1987 г. были разработаны алгоритмы FEAL и RC2. Сети Фейстеля получили широкое распространение в 1990-е гг. – в годы появления таких алгоритмов, как Blowfish (1993), TEA (1994), RC5 (1994), CAST-128 (1996), XTEA (1997), XXTEA (1998), RC6 (1998) и др. На основе сети Фейстеля в 1990 г. в СССР был принят в качестве ГОСТ 28147–89 стандарт шифрования.

Предполагалось, что DES будет сертифицироваться каждые 5 лет. Срок действия последнего сертификата на территории США истек практически к концу XX в. К тому времени DES был вскрыт «лобовой атакой».

В 1998 г. NIST объявил конкурс на новый стандарт, который завершился в 2001 г. принятием AES (Advanced Encryption Standard).

**! Все перечисленные стандарты и алгоритмы блочных шифров (БШ) строятся на основе подстановочных и перестановочных шифров, т. е. являются комбинационными. БШ относятся также к классу симметричных.**

Блочное зашифрование (расшифрование) предполагает разбиение исходного открытого (зашифрованного) текста на равные блоки, к которым применяется однотипная процедура зашифрования (расшифрования). Указанная однотипность характеризуется прежде всего тем, что процедура зашифрования (расшифрования)

состоит из совокупности повторяющихся наборов преобразований, называемых **раундами**.

Основные требования к шифрам рассматриваемого класса можно сформулировать следующим образом:

- даже незначительное изменение исходного сообщения должно приводить к существенному изменению зашифрованного сообщения;
- устойчивость к атакам по выбранному тексту;
- алгоритмы зашифрования/расшифрования должны быть реализуемы на различных платформах;
- алгоритмы должны базироваться на простых операциях;
- алгоритмы должны быть простыми для написания кода, вероятность появления программных ошибок должна быть низкой;
- алгоритмы должны допускать их модификацию при переходе на иные требования по уровню криптостойкости.

### 5.1.2. Сеть Фейстеля

Само название конструкции Фейстеля (сеть) означает ее **ячеистую** топологию [28]. Формально одна ячейка сети соответствует одному раунду зашифрования или расшифрования сообщения.

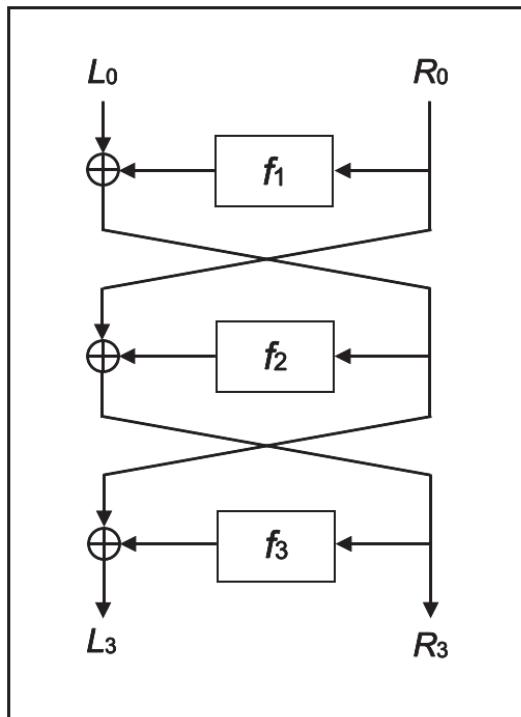
При зашифровании сообщение разбивается на блоки одинаковой (фиксированной) длины (как правило – 64 или 128 битов).

Полученные блоки называются **входными**. В случае если длина входного блока меньше, чем выбранный размер, блок удлиняется установленным способом.

Каждый входной блок шифруемого сообщения изначально делится на два подблока одинакового размера: левый ( $L_0$ ) и правый ( $R_0$ ). Далее в каждом  $i$ -м раунде выполняются преобразования в соответствии с формальным представлением ячейки сети Фейстеля:

$$\left. \begin{aligned} L_i &= R_{i-1}; \\ R_i &= L_{i-1} + f(R_{i-1}, K_i). \end{aligned} \right\}$$

По какому-либо математическому правилу вычисляется раундовый ключ  $K_i$ . В приведенном выражении знак «+» соответствует поразрядному суммированию на основе «XOR». На рис. 5.1 приведено графическое отображение сети Фейстеля.



**Рис. 5.1.** Графическое отображение сети Фейстеля

Расшифрование происходит так же, как и зашифрование, с той лишь разницей, что раундовые ключи будут использоваться в обратном порядке по отношению к зашифрованию.

В своей статье [28] Х. Фейстель описывает два блока преобразований с использованием функции  $f(R_{i-1}, K_i)$ :

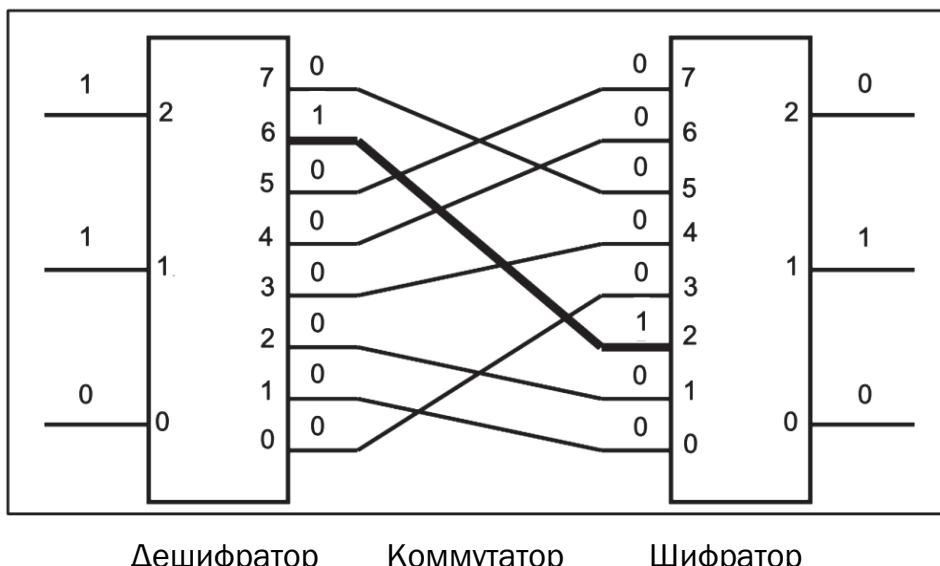
- блок подстановок (*S*-блок, англ. *S-box*);
- блок перестановок (*P*-блок, англ. *P-box*).

Блок подстановок состоит:

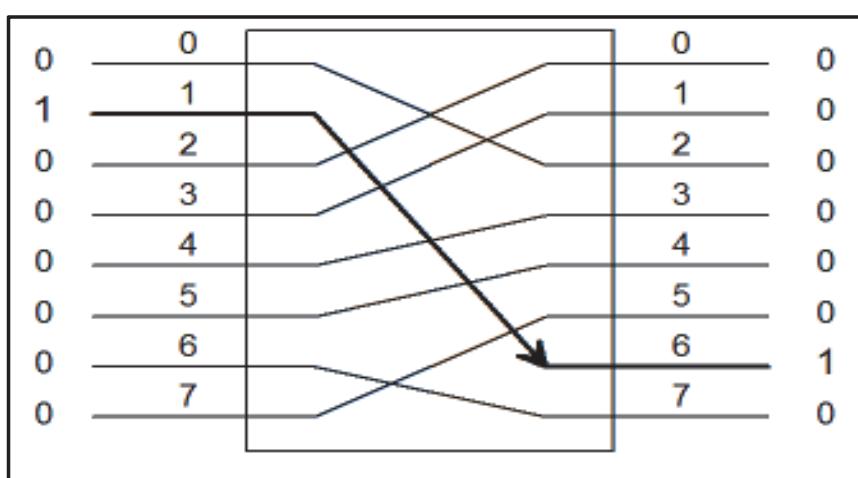
- из дешифратора, преобразующего  $n$ -разрядное двоичное число в одноразрядный сигнал по основанию  $2^n$ ;
- внутреннего коммутатора;
- шифратора, преобразующего сигнал из одноразрядного  $2^n$ -ричного в  $n$ -разрядный двоичный.

Пример реализации трехразрядного *S*-блока показан на рис. 5.2.

Блок перестановок изменяет положение цифр, т. е. является линейным устройством. Этот блок может иметь очень большое количество входов-выходов, однако в силу линейности является слабым местом преобразования с точки зрения криптостойкости. На рис. 5.3 приведен пример реализации 8-разрядного *P*-блока.



**Рис. 5.2.** Пример реализации трехразрядного S-блока

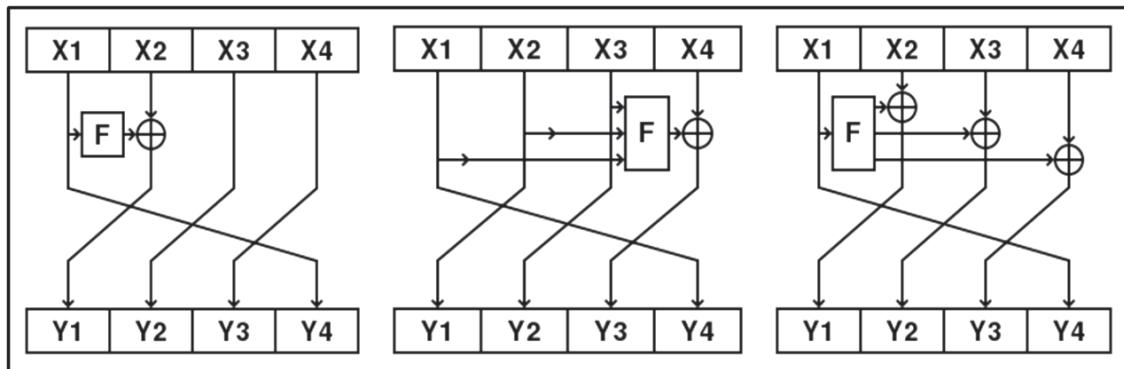


**Рис. 5.3.** Пример реализации 8-разрядного P-блока

Термин «блок» в оригинальной статье [28] используется вместо термина «функция» вследствие того, что речь идет о блочном шифре.

При большом размере блоков шифрования (128 битов и более) реализация такой конструкции Фейстеля на 32-разрядных архитектурах может вызвать затруднения, поэтому применяются модифицированные варианты этой конструкции. Обычно используются сети с четырьмя ветвями. На рис. 5.4 показаны наиболее распространенные модификации.

Такие модификации предусматривают использование не двух ( $L$  и  $R$  на рис. 5.1), а четырех ветвей. На рис. 5.4 показаны некоторые модификации.



**Рис. 5.4.** Примеры модификаций базовой сети Фейстеля



**В основе криптостойкости блочных шифров лежит идея К. Шеннона о представлении составного шифра таким образом, чтобы он обладал двумя важными свойствами: *рассеиванием* и *перемешиванием*. Рассеивание должно скрыть отношения между зашифрованным текстом и исходным текстом.**

*Рассеивание подразумевает, что каждый символ (или бит) в зашифрованном тексте зависит от одного или всех символов в исходном тексте. Другими словами, если единственный символ в исходном тексте изменен, несколько или все символы в зашифрованном тексте будут также изменены.*

**Идея относительно перемешивания заключается в том, что оно должно скрыть отношения между зашифрованным текстом и ключом.**

### 5.1.3. Базовые операции сложения чисел в блочных шифрах

Как было указано выше, в основе сети Фейстеля лежит простейшая операция суммирования двух ( $A + B$ )  $n$ -разрядных чисел – XOR:  $A + B \pmod n$ . Помимо этой операции некоторые алгоритмы (Blowfish, IDEA, ГОСТ и др.) предусматривают выполнение операций сложения чисел по модулю более высоких порядков: XOR:  $A + B \pmod {2^n}$ . Понятно, что числа  $A$  и  $B$  также являются  $n$ -разрядными.

Реализация второй из указанных операций является более сложной. Вспомним основные ее особенности:

1) самое большое слагаемое меньше  $2^n$ . Например, при  $n = 3$  самое большое слагаемое в двоичном виде – это 111 (или 7), а  $2^n = 8$ ;

2) результатом сложения также должно быть  $n$ -разрядное число;

3) побитовое сложение предусматривает известную взаимосвязь между соседними символами (порядками);

4) в силу известных правил модулярной арифметики результат вычисления  $A + B \pmod{2^n}$  – это остаток от деления:  $(A + B) / 2^n$ .

Рассмотрим примеры.

**Пример 1.** Пусть  $n = 4$ ,  $A = 9$ ,  $B = 7$ , т. е.  $2^n = 16$ .

Легко подсчитать,  $A + B = 16$ ;  $9 + 7 \pmod{16} = 0$ .

Представим слагаемые в двоичной форме:  $A = 1001$ ,  $B = 0111$ ;  $A + B = 1001 + 0111 = 10000$ . Для получения нужного результата – вычисления  $9 + 7 \pmod{16}$  – следует взять младшие 4 разряда ( $n = 4$ ) суммы: 0000.

**Пример 2.** Пусть  $n = 4$ ,  $A = 4$ ,  $B = 10$ . Выполним вычисления по аналогии с примером 1.

Получаем  $A + B = 14$ , результирующая сумма меньше 16;  $14 \pmod{16} = 14$ .

Представим слагаемые в двоичной форме:  $A = 0100$ ,  $B = 1010$ ;  $A + B = 0100 + 1010 = 1110$ . Таким образом, итоговое число состоит из требуемых 4 разрядов.

**Пример 3.** Пусть  $n = 4$ ,  $A = 10$ ,  $B = 11$ . Их сумма по модулю 16 дает 5. В двоичной форме:  $1010 + 1011 = 10101$ . Искомое двоичное число – 4 младших разряда.

Если для данных в каждом из рассмотренных примеров мы проведем операцию деления  $(A + B) / 2^4$ , получим те же 4 бита (выделены жирным).

Таким образом, общие правила выполнения рассматриваемой операции формально можно представить следующим образом:

$$A + B \pmod{2^n} \equiv \begin{cases} A + B, & \text{если } A + B < 2^n \\ A + B - 2^n, & \text{если } A + B \geq 2^n \end{cases}.$$

#### 5.1.4. Некоторые блочные алгоритмы

##### 5.1.4.1. Алгоритм DES

В силу причин, перечисленных в п. 5.1.1, данный алгоритм является, пожалуй, наиболее исследованным.

Алгоритм строится на основе сети Фейстеля.

Входной блок данных, состоящий из 64 битов, преобразуется в выходной блок идентичной длины. В алгоритме широко используются **рассеивания** (подстановки) и **перестановки** битов текста, о которых мы упоминали выше. Комбинация двух указанных методов преобразования образует фундаментальный строительный блок DES, называемый **раундом** или циклом.

Один блок данных подвергается преобразованию (и при зашифровании, и при расшифровании) в течение 16 раундов.

После первоначальной перестановки и разделения 64-битного блока данных на правую ( $R_0$ ) и левую ( $L_0$ ) половины длиной по 32 бита выполняются 16 раундов одинаковых действий (см. рис. 5.5). Функционал этих действий подробно рассмотрен в п. 5.1 пособия [3].

В табл. 5.1 показан принцип первоначальной перестановки разрядов (*IP*) входного 64-битного слова.

Таблица 5.1  
**Начальная перестановка**

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

Выполненная перестановка означает, например, что первый бит входного блока сообщения будет размещен на 40-й позиции (цифра «1» выделена жирным), а 58-й (выделено жирным с подчеркиванием) – на первой и т. д. Из беглого анализа выполненной перестановки легко понять принцип. Алгоритм перестановки разрабатывался для облегчения загрузки блока входного сообщения в специализированную микросхему. Вместе с тем эта операция придает некоторую «хаотичность» исходному сообщению, снижая возможность использования криptoанализа статистическими методами.

Левая и правая ветви каждого промежуточного значения обрабатываются как отдельные 32-битные значения, обозначенные  $L_i$  и  $R_i$ .

Вначале правая часть блока  $R_i$  расширяется до 48 битов с использованием таблицы, которая определяет перестановку плюс расширение на 16 битов. Эта операция приводит размер правой половины в соответствие с размером ключа для выполнения операции XOR.

Кроме того, за счет выполнения этой операции быстрее возрастает зависимость всех битов результата от битов исходных данных и ключа (это называется «лавинный эффект»).



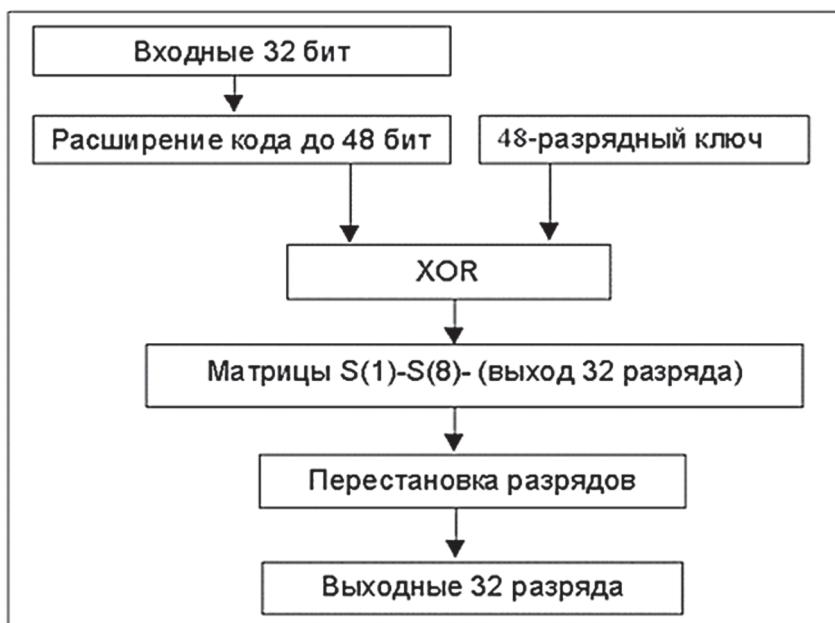
### Рис. 5.5. Общая схема алгоритма DES

После выполнения перестановки с расширением для полученного 48-битного значения выполняется операция XOR с 48-битным подключом  $K_i$ . Затем полученное 48-битное значение подается на вход блока подстановки  $S$  (от англ. *substitution* – подстановка), результатом которой является 32-битное значение. Подстановка выполняется в восьми блоках подстановки или восьми  $S$ -блоках ( $S$ -boxes). При выполнении этой операции 48 битов данных делятся на восемь 6-битных подблоков, каждый из которых по соответствующей *таблице замен* замещается четырьмя битами. Подстановка с помощью  $S$ -блоков является одним из важнейших этапов DES. Таблицы замен для этой операции специально спроектированы так, чтобы обеспечивать максимальную криптостойкость. В результате выполнения этого этапа получаются восемь 4-битных блоков, которые вновь объединяются в единичное 32-битное значение.

Далее полученное 32-битное значение обрабатывается с помощью перестановки  $P$  (от англ. *permutation* – перестановка), которая не зависит от используемого ключа. Целью перестановки является

такое максимальное переупорядочивание битов, чтобы в следующем раунде шифрования каждый бит с большой вероятностью обрабатывался другим  $S$ -блоком.

И наконец, результат перестановки объединяется с помощью операции XOR с левой половиной первоначального 64-битного блока данных. Затем левая и правая половины меняются местами, и начинается следующий раунд (рис. 5.6).



**Рис. 5.6.** Схема реализации функции  $f$  на рис. 5.5

После выполнения 16-раундового зашифрования 64-битного блока данных осуществляется конечная перестановка ( $IP^{-1}$ ). Она является обратной к перестановке  $IP$ . Конечная перестановка определяется табл. 5.2.

Таблица 5.2  
Конечная перестановка

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

Каждый 8-й бит исходного 64-битного ключа отбрасывается. Эти 8 битов, находящихся в позициях 8, 16, 24, 32, 40, 48, 56, 64, изначально добавляются в исходный ключ таким образом, чтобы каждый

байт содержал четное число единиц. Это используется для обнаружения ошибок при обмене и хранении ключей по известным алгоритмам избыточного кодирования (см. лабораторные работы № 4–6 из [2]). Один избыточный бит в ключе DES формируется, как видим, в соответствии с кодом **простой четности**. Этот код позволяет в кодовом слове (в нашем случае – в каждом байте ключа) обнаруживать ошибки, количество которых нечетно.

При расшифровании на вход алгоритма подается зашифрованный текст. Единственное отличие состоит в обратном порядке использования частичных ключей  $K_i$ . Ключ  $K_{16}$  используется в первом раунде,  $K_1$  – в последнем.

После последнего раунда процесса расшифрования две половины выхода меняются местами так, чтобы вход заключительной перестановки был составлен из подблоков  $R_{16}$  и  $L_{16}$ . Выходом этой стадии является расшифрованный текст.

**Слабые и полуслабые ключи.** Из-за того что первоначальный ключ изменяется при получении подключа для каждого раунда алгоритма, определенные первоначальные ключи являются **слабыми** [5]. Вспомним, что первоначальное значение разделяется на две половины, каждая из которых сдвигается независимо. Если все биты каждой половины равны 0 или 1, то для всех раундов алгоритма используется один и тот же ключ. Это может произойти, если ключ состоит из одних 1, из одних 0, или если одна половина ключа состоит из одних 1, а другая – из одних 0.

Четыре слабых ключа показаны в шестнадцатеричном виде в табл. 5.3 (каждый восьмой бит – это бит четности).

Таблица 5.3

**Слабые ключи DES [5]**

0101	0101	0101	0101
1F1F	1F1F	0E0E	0E0E
E0E0	E0E0	F1F1	F1F1
FEFE	FEFE	FEFE	FEFE

Кроме того, некоторые пары ключей при зашифровании переводят открытый текст в идентичный шифртекст. Иными словами, один из ключей пары может расшифровать сообщения, зашифрованные другим ключом пары. Это происходит из-за метода, используемого DES для генерации подключей: вместо 16 различных подключей эти ключи генерируют только два различных подключа.

В алгоритме каждый из этих подключей используется восемь раз. Эти ключи, называемые **полуслабыми**, в шестнадцатеричном виде приведены в табл. 5.4 [5].

Таблица 5.4  
**Полуслабые ключи DES**

01FE	01FE	01FE	01FE	FE01	FE01	FE01	FE01
1FEO	1FEO	0EF1	0EF1	E01F	E01F	F10E	F10E
01EO	01EO	01F1	01F1	E001	E001	F101	F101
1FFE	1EEE	0EFE	0EFE	FE1F	FE1F	FE0E	FE0E
011F	011F	010E	010E	1F01	1F01	0E01	0E01
EOF	EOF	F1FE	F1FE	FEE0	FEE0	FEE1	FEE1

**Криптоанализ DES.** Дифференциальный криптоанализ базируется на таблице неоднородных дифференциальных распределений S-блоков в блочном шифре. Криптоанализ шифртекстов на основе рассматриваемого стандарта «работает» с парами шифртекстов, открытые тексты которых имеют определенные разности, как это отмечалось в материалах к лабораторной работе № 2. Метод анализирует эволюцию этих разностей в процессе прохождения открытых текстов раундов DES при шифровании одним и тем же ключом. Для DES термин «разность» определяется с помощью операции XOR. Затем, используя разности полученных шифртекстов, присваивают различные вероятности различным ключам. В процессе дальнейшего анализа следующих пар шифртекстов один из ключей станет наиболее вероятным. Это и есть правильный ключ (см. более подробно [5], с. 326).

**Линейный криптоанализ:** для того чтобы найти линейное приближение для DES, нужно найти «хорошие» однораундовые линейные приближения и объединить их. Обратим внимание на S-блоки. У них 6 входных битов и 4 выходных. Входные биты можно объединить с помощью операции XOR 63 способами ( $2^6 - 1$ ), а выходные биты – 15 способами. Теперь для каждого S-блока можно оценить вероятность того, что для случайно выбранного входа входная комбинация XOR равна некоторой выходной комбинации XOR, и т. д.

DES давно характеризуется низкой криптостойкостью: в январе 1999 г. закодированное посредством DES сообщение было взломано с помощью связанных через Интернет в единую сеть 100 тыс. персональных компьютеров за 24 часа. Данному алгоритму

присуща проблема так называемых «слабых» и «частично слабых» ключей [5].

Основное достоинство DES – относительно высокая скорость (из-за малой длины ключа); бесплатное распространение по всему миру; общедоступность и отсутствие необходимости лицензионных отчислений.

Модификацией DES является 3DES, он создан У. Диффи, М. Хеллманом, У. Тачманном в 1978 г.

Формальная запись:

$$C_i = f(M_j, (\text{DES}(K_3, (\text{DES}(K_2, (\text{DES}, K_1)))))).$$

Существуют несколько реализаций алгоритма 3DES. Вот некоторые из них:

- DES-EEE3: шифруется 3 раза с 3 разными ключами (операции шифрование-шифрование-шифрование);
- DES-EDE3: 3DES операции шифрование-расшифрование-шифрование с разными ключами;
- DES-EEE2 и DES-EDE2: как и предыдущие, однако на первом и третьем шаге используется одинаковый ключ.

Расшифрование происходит, как и в простом DES, в обратном порядке по отношению к процедуре зашифрования.

3DES с тремя ключами реализован во многих Интернет-приложениях. Например, в PGP (Pretty Good Privacy) он позволяет выполнять операции шифрования и цифровой подписи сообщений, файлов и другой информации, представленной в электронном виде, например, на жестком диске; в S/mime применяется для обеспечения криптографической безопасности электронной почты.

3DES используется при управлении ключами в стандартах ANSI X9.17 (метод генерации 64-битных ключей) и ISO 8732 (управление ключами в банковском деле), а также в PEM (Privacy Enhanced Mail).

#### **5.1.4.2. Стандарт AES**

*AES* (Advanced Encryption Standard) – алгоритм шифрования, действующий в качестве государственного стандарта в США с 2001 г. В основу стандарта положен шифр *Rijndael*. Шифр *Rijndael/AES* (т. е. рекомендуемый стандартом) характеризуется размером блока 128 битов, длиной ключа 128, 192 или 256 битов и количеством раундов 10, 12 или 14 в зависимости от длины ключа.

Основу *Rijndael* составляют так называемые линейно-подстановочные преобразования. В алгоритме широко используются табличные вычисления, причем все необходимые таблицы задаются константно, т. е. не зависят ни от ключа, ни от данных.

#### 5.1.4.3. Стандарт ГОСТ 28147–89

В Советском Союзе в качестве стандарта на блочные алгоритмы шифрования с закрытым ключом в 1989 г. был принят ГОСТ 28147–89 [31].

ГОСТ предусматривает три режима шифрования (*простая замена, гаммирование, гаммирование с обратной связью*) и один режим выработки имитовставки. Первый из режимов шифрования предназначен для шифрования ключевой информации и не может использоваться для шифрования других данных, для этого предусмотрены два других режима. Режим выработки имитовставки (криптографической контрольной комбинации) предназначен для имитозащиты шифруемых данных, т. е. для их защиты от случайных или преднамеренных несанкционированных изменений.

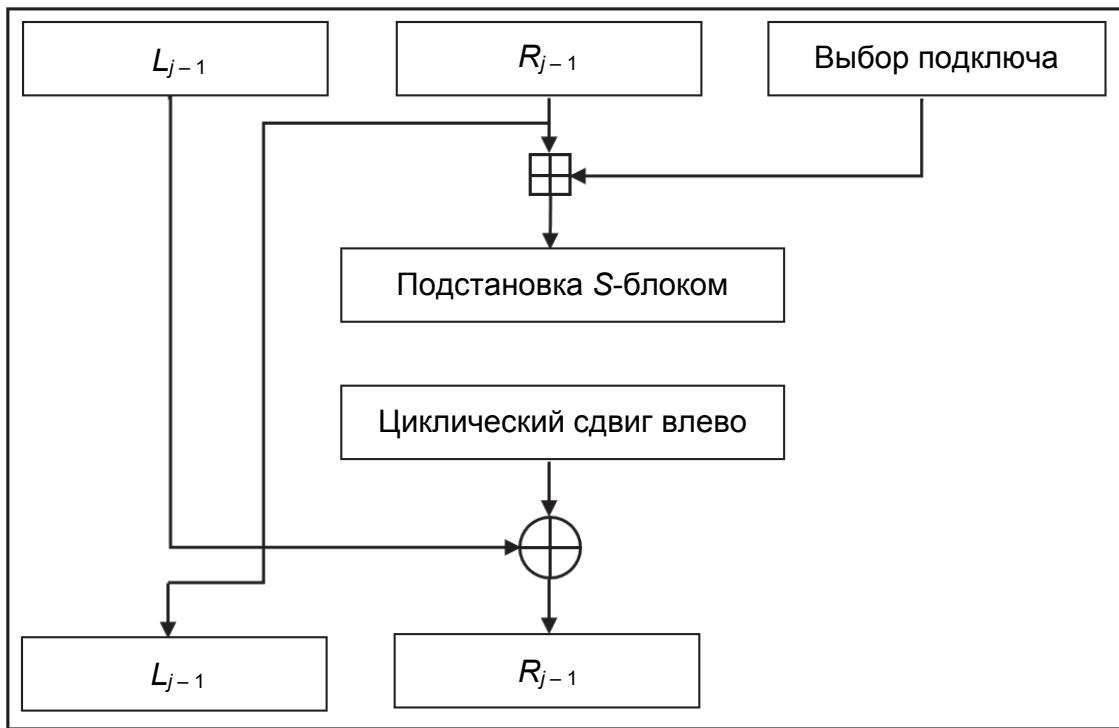
Шифр ГОСТ 28147–89 построен по тем же принципам, что и американский DES, однако по сравнению с DES первый более удобен для программной реализации. В ГОСТ 28147–89 применяется более длинный ключ – 256 битов, здесь используются 32 раунда шифрования.

Таким образом, основные параметры алгоритма криптографического преобразования данных ГОСТ 28147–89: размер блока составляет 64 бита, размер ключа – 256 битов, количество раундов – 32.

Алгоритм представляет собой классическую сеть Фейстеля. Шифруемый блок данных разбивается на две одинаковые части, правую  $R$  и левую  $L$ . Правая часть складывается по модулю  $2^{32}$  с подключом раунда и посредством принятого алгоритма шифрует левую часть. Перед следующим раундом левая и правая части меняются местами. Такая структура позволяет использовать один и тот же алгоритм как для зашифрования, так и для расшифрования блока (рис. 5.7).

Таким образом, в алгоритме используются следующие операции:

- сложение слов по модулю  $2^{32}$ : правый блок ( $R_i$ ) складывается по модулю  $2^{32}$  с текущим подключом ( $K_i$ );
- циклический сдвиг слова влево на указанное число битов;
- побитовое сложение по модулю 2 (XOR);
- замена (подстановка в блоке  $S$ ) по таблице.



**Рис. 5.7.** Структура алгоритма реализации раунда в стандарте ГОСТ 28147-89 (знаком «+» в квадратной рамке обозначена операция сложения по модулю  $2^{32}$ )

Таким образом, в алгоритме используются следующие операции:

- сложение слов по модулю  $2^{32}$ : правый блок ( $R_i$ ) складывается по модулю  $2^{32}$  с текущим подключом ( $K_i$ );
- циклический сдвиг слова влево на указанное число битов;
- побитовое сложение по модулю 2 (XOR);
- замена (подстановка в блоке  $S$ ) по таблице.

На различных шагах алгоритмов ГОСТа данные, которыми они оперируют, интерпретируются и используются различным образом. В некоторых случаях элементы данных обрабатываются как массивы независимых битов, в других случаях – как целое число без знака, в третьих – как имеющий структуру сложный элемент, состоящий из нескольких более простых элементов.

Сначала правый блок складывается по модулю  $2^{32}$  с подключом. Полученное 32-битное сообщение делится на восемь 4-битных чисел. Каждое из этих 4-битных чисел преобразуется соответствующим  $S$ -блоком в другое 4-битное число. Поэтому любой  $S$ -блок определяется некоторой 16-битной перестановкой на множестве из 16 элементов: 0, 1, ..., 15.

Исходный 256-битный ключ делится на восемь 32-битных подключей. Они используются в 32 тактах в следующем порядке: 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 8, 7, 6, 5, 4, 3, 2, 1. При расшифровании порядок использования подключей меняется на противоположный.

Поскольку в ГОСТ 28147–89 используется 256-битный ключ, то объем ключевого пространства составляет  $2^{256}$ . Даже если предположить, что на взлом шифра брошены все силы вычислительного комплекса с возможностью перебора  $10^{12}$  (это примерно равно  $2^{40}$ ) ключей в 1 секунду, то на полный перебор всех  $2^{256}$  ключей потребуется  $2^{216}$  секунд. Это время составляет более миллиарда лет.

К уже отмеченным отличиям между алгоритмами DES и ГОСТ можно добавить следующее. В основном раунде DES применяются нерегулярные перестановки исходного сообщения, в ГОСТ используется 11-битный циклический сдвиг влево. Последняя операция гораздо удобнее для программной реализации. Но перестановка DES увеличивает лавинный эффект. В ГОСТ изменение одного входного бита влияет на один 4-битовый блок при замене в одном раунде, который затем влияет на два 4-битовых блока следующего раунда, три блока следующего и т. д. В ГОСТ требуется 8 раундов прежде, чем изменение одного входного бита повлияет на каждый бит результата; в DES для этого нужно только 5 раундов.

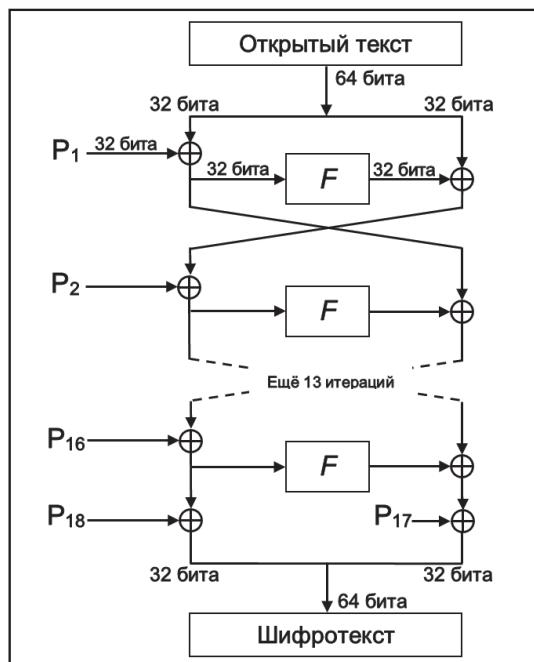
#### 5.1.4.4. Алгоритм Blowfish

Основой алгоритма является сеть Фейстеля с 16 раундами. Длина блока равна 64 битам, ключ может иметь любую длину в пределах 448 битов. Хотя перед началом любого зашифрования выполняется сложная фаза инициализации, само зашифрование данных выполняется достаточно быстро.

Алгоритм предназначен в основном для приложений, в которых ключ меняется нечасто, к тому же существует фаза начального рукопожатия, во время которой происходит аутентификация сторон. Blowfish характеризуется более высокой скоростью обработки блоков, чем DES.

Алгоритм состоит из двух частей: расширение ключа и зашифрование/расшифрование данных. Расширение ключа преобразует ключ длиной, по крайней мере, 448 битов в несколько массивов подключей общей длиной 4168 байтов.

Каждый раунд состоит из перестановки, зависящей от ключа, и подстановки, зависящей от ключа и данных. Операциями являются XOR и сложение по модулю  $2^{32}$  (рис. 5.8).



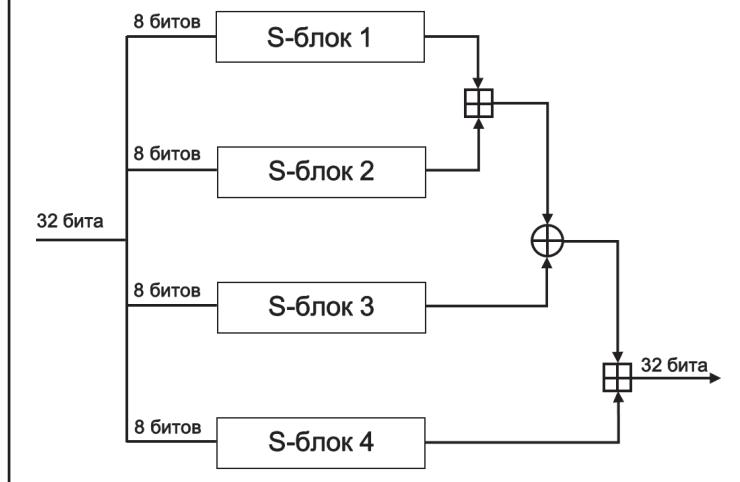
**Рис. 5.8.** Структура алгоритма Blowfish

Blowfish использует большое количество подключей. Эти ключи должны быть вычислены заранее, до начала любого зашифрования/расшифрования данных.

Элементы алгоритма:

- $P$ -массив, состоящий из восемнадцати 32-битных подключей:  $P_1, P_2, \dots, P_{18}$ ;
- $S$ -блоки: каждый из четырех 32-битных  $S$ -блоков содержит 256 элементов;
- функция  $F$ , структура которой показана на рис. 5.9.

Успешные атаки на рассмотренный алгоритм неизвестны.



**Рис. 5.9.** Структура функции  $F$  алгоритма Blowfish

! Следует также обратить внимание на еще одно обстоятельство: файл, который содержит зашифрованные данные, практически нельзя сжать. Это может служить одним из критериев поиска зашифрованных файлов, которые, как и обычные файлы, представляют собой набор случайных битов. Дополнительные сведения по рассмотренным вопросам можно найти в [9].

## 5.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими некоторые блочные алгоритмы, из приложения в [5].

Приложение должно реализовывать следующие операции:

- разделение входного потока данных на блоки требуемой длины с необходимым дополнением последнего блока;
- выполнение требуемых преобразований ключевой информации;
- выполнение операций зашифрования/расшифрования;
- оценка скорости выполнения операций зашифрования/расшифрования;
- пошаговый анализ лавинного эффекта с подсчетом количества изменяющихся символов по отношению к исходному слову.

Исследуемый метод шифрования и ключевая информация – в соответствии с вариантом из табл. 5.5.

Таблица 5.5  
Варианты задания

Вариант	Алгоритм	Ключ
1	DES	Первые 8 символов собственных фамилий имени
2	DES-EDE3	а) <u>Информационная безопаснос*</u> б) <u>лабораторная--работа №5*</u> <small>* каждый из трех ключей выделен шрифтом</small>
3	DES-EDE3	По указанию преподавателя
4	DES-EDE2	По указанию преподавателя
5	DES-EDE2	По указанию преподавателя

По желанию студент может разработать приложение и выполнить связанные исследования для любого другого блочного алгоритма, не указанного в табл. 5.5.

2. Проанализировать влияние слабых ключей (табл. 5.3) и полуслабых ключей (табл. 5.4) на конечный результат зашифрования и на лавинный эффект.
3. Оценить степень сжатия (используя любой доступный архиватор) открытого текста и соответствующего зашифрованного текста. Дать пояснения к полученному результату.
4. Результаты оформить в виде отчета по установленным правилам.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. Какие простейшие операции применяются в блочных алгоритмах шифрования?
2. В чем отличие блочных алгоритмов шифрования от потоковых?
3. Что понимается под «раундом» алгоритма шифрования?
4. Охарактеризовать и привести формальное описание сети Фейстеля.
5. Какие стандартные операции используются в блочных алгоритмах шифрования?
6. В чем состоит особенность сложения чисел по модулю  $2^n$ ?
7. Сложить по модулю  $10^2$  пары чисел: 55 и 14; 76 и 24; 99 и 99.
8. Сложить по модулю  $2^8$ :  
двоичные числа 10101100 и 11001010; 01111111 и 01101101;  
шестнадцатеричные числа 0B5 и 37.
9. Дать пояснение принципам реализации «лавинного» эффекта.
10. Выбрать два произвольных блочных алгоритма. В чем состоят отличия между ними?
11. Представить графически и пояснить функционал одного раунда блочного алгоритма DES (AES, ГОСТ 28147–89, Blowfish).
12. Сколько можно реализовать (теоретически) разновидностей алгоритма 3DES?
13. Какие факторы влияют на стойкость блочного алгоритма шифрования?
14. В чем состоит сущность дифференциального криptoанализа?
15. В чем состоит сущность линейного криptoанализа?
16. Какие ключевые комбинации относятся к слабым (к полуслабым) и почему?
17. Где применяются блочные криптографические алгоритмы?

## **Лабораторная работа № 6**

# **ИССЛЕДОВАНИЕ ПОТОКОВЫХ ШИФРОВ**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации потоковых шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости потоковых шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для потокового зашифрования/расшифрования.
3. Выполнить анализ криптостойкости потоковых шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### **6.1. Теоретические сведения**

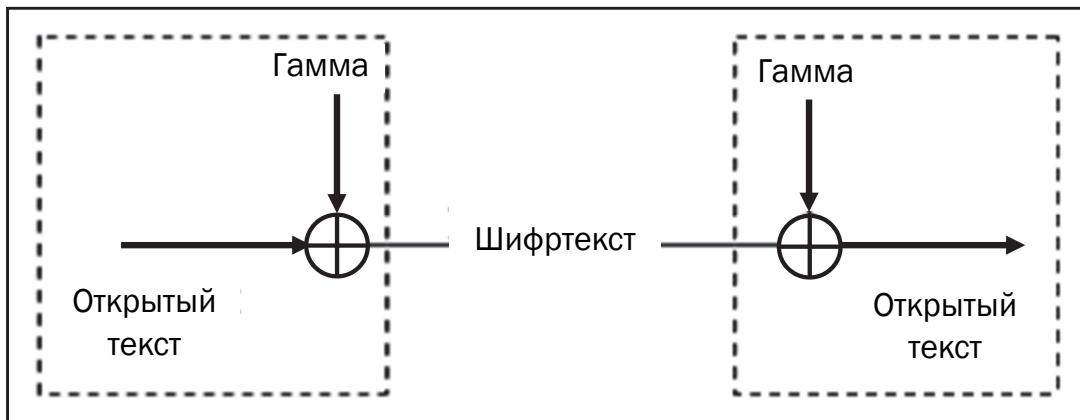
#### **6.1.1. Классификация и общие свойства потоковых шифров**

**Потоковый шифр** (иногда говорят «поточный») – симметричный шифр, преобразующий каждый символ  $t_i$  открытого текста в символ шифрованного  $c_i$ , зависящий от ключа и расположения символа в тексте.

Термин «потоковый шифр» обычно используется в том случае, когда шифруемые символы открытого текста представляются одной буквой, битом или реже – байтом.

Все потоковые шифры делятся на 2 класса: *синхронные* и *асинхронные* (или *самосинхронизирующиеся*). Необходимые начальные сведения об общих характеристиках и свойствах потоковых шифров можно найти в гл. 6 из [3], а также в [5].

Основной задачей потоковых шифров является выработка некоторой последовательности (гаммы) для зашифрования, т. е. выходная гамма является ключевым потоком (ключом) для сообщения. В общем виде схема потокового шифра изображена на рис. 6.1.



**Рис. 6.1.** Схема потокового шифра

*Синхронные потоковые шифры (СПШ)* характеризуются тем, что поток ключей генерируется независимо от открытого текста и шифртекста. Главное свойство СПШ – нераспространение ошибок. Ошибки отсутствуют, пока работают синхронно шифровальное и дешифровальное устройства отправителя и получателя информации. Один из методов борьбы с рассинхронизацией – разбить открытый текст на отрезки, начало и конец которых выделить вставкой контрольных меток (специальных маркеров).

**! Синхронные потоковые шифры уязвимы к атакам на основе изменения отдельных битов шифртекста.**

В самосинхронизирующихся потоковых шифрах символы ключевой гаммы зависят от исходного секретного ключа шифра и от конечного числа последних знаков зашифрованного текста. Основная идея заключается в том, что внутреннее состояние генератора потока ключей является функцией фиксированного числа предыдущих битов шифртекста. Поэтому генератор потока ключей на приемной стороне, приняв фиксированное число битов, автоматически синхронизируется с генератором гаммы.

**! Недостаток этих потоковых шифров – распространение ошибок, так как искажение одного бита в процессе передачи шифртекста приведет к искажению нескольких битов гаммы и, соответственно, расшифрованного сообщения.**

### 6.1.2. Генераторы ключевой информации

Потоковый шифр максимально должен имитировать одноразовый блокнот. В соответствии с этим ключ должен по своим свойствам максимально походить на случайную числовую последовательность.

Ключевые последовательности (*случайные последовательности* (СП), либо *псевдослучайные последовательности* (ПСП))рабатываются специальными блоками систем потокового шифрования – генераторами. В Беларуси в настоящее время действует стандарт СТБ 34.101.47–2017 «Информационные технологии и безопасность. Алгоритмы генерации псевдослучайных чисел» [32].

Стандарт устанавливает криптографические алгоритмы генерации псевдослучайных чисел. Алгоритмы стандарта могут применяться для построения ключей, синхропосылок, одноразовых паролей, других непредсказуемых или уникальных параметров криптографических алгоритмов и протоколов. Стандарт применяется при разработке, испытаниях и эксплуатации средств криптографической защиты информации.

Указанный стандарт определяет базовые понятия в рассматриваемой предметной области:

- *случайные числа* (последовательности) – последовательность элементов, каждый из которых не может быть предсказан (вычислен) только на основе знания предшествующих ему элементов данной последовательности;
- *псевдослучайные числа* – последовательность элементов, полученная в результате выполнения некоторого алгоритма и используемая в конкретном случае вместо последовательности случайных чисел.

#### 6.1.2.1. Линейный конгруэнтный генератор

Часто используемый алгоритм генерирования (программно или аппаратно) ПСП реализуется на основе так называемого *линейного конгруэнтного генератора*, описываемого следующим рекуррентным соотношением:

$$x_{t+1} \equiv (ax_t + c) \bmod n, \quad (6.1)$$

где  $x_t$  и  $x_{t+1}$  – соответственно  $t$ -й (предыдущий) и  $(t+1)$ -й (текущий, вычисляемый) члены числовой последовательности;  $a$ ,  $c$  и  $n$  – константы. Период такого генератора (период ПСП) не превышает  $n$ .

Если параметры  $a$ ,  $b$  и  $c$  выбраны правильно, то генератор будет порождать случайные числа с максимальным периодом, равным  $c$ . При программной реализации значение  $c$  обычно устанавливается равным  $2^{b-1}$  или  $2^b$ , где  $b$  – длина слова в битах.

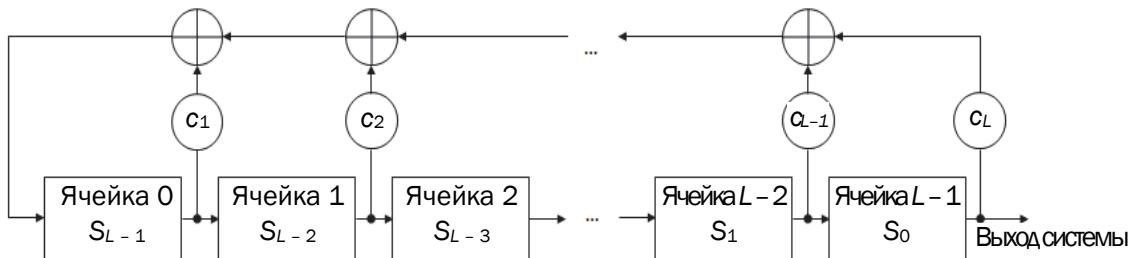
Достоинством линейных конгруэнтных генераторов псевдослучайных чисел является их простота и высокая скорость получения псевдослучайных значений. Линейные конгруэнтные генераторы находят применение при решении задач моделирования и математической статистики, однако в криптографических целях их нельзя рекомендовать к использованию, так как специалисты по криптоанализу научились восстанавливать всю последовательность ПСП по нескольким ее значениям.

Генератор практически не используется в криптографии в силу низкой криптостойкости. Тем не менее он полезен для решения задач моделирования.

Комбинации нескольких (чаще двух) линейных конгруэнтных генераторов позволяют значительно повысить период ПСП. Б. Шнайер, например, приводит данные о том, как на 32-разрядных ПК реализовать генератор в виде комбинации двух, каждый из которых обеспечивает период соответственно  $2^{31} - 85$  и  $2^{31} - 249$ , а комбинированный генератор позволяет достичь периода ПСП, равного произведению указанных чисел [5].

### 6.1.2.2. Генератор ПСП на основе регистров сдвига

Достаточно распространенным является использование *регистров сдвига* (РС) в качестве генераторов ПСП в силу простоты реализации на основе цифровой логики. РС с линейной обратной связью (РСЛОС) состоит из двух частей: собственно РС и функции обратной связи. На рис. 6.2 представлена общая схема РС с линейной обратной связью. Функция обратной связи реализуется с помощью сумматоров сложения по модулю два (элементы XOR; на рис. 6.2 обозначены в виде кружочков со знаком сложения).



**Рис. 6.2.** Общая схема регистра сдвига с линейной обратной связью

РСЛОС строятся на основе *примитивных порождающих полиномов (многочленов)*, которые мы подробно анализировали при изучении циклических помехоустойчивых кодов [2] (см. также [33, 34]). Если многочлен является неприводимым, то период ПСП при ненулевом начальном условии (ненулевом состоянии) регистра будет максимально возможным:  $2^L - 1$ .

### 6.1.2.3. Генератор псевдослучайных чисел на основе алгоритма RSA

Алгоритм RSA разработан для систем асимметричного зашифрования/расшифрования и будет более детально рассмотрен с практической точки зрения ниже.

Генератор же ПСП на основе RSA устроен следующим образом. Последовательность генерируется с использованием соотношения

$$x_t \equiv (x_{t-1})^e \pmod{n}. \quad (6.2)$$

Начальными параметрами служат  $n$ , большие простые числа  $p$  и  $q$  (причем  $n = pq$ ), целое число  $e$ , взаимно простое с произведением  $(p-1)(q-1)$ , а также некоторое случайное начальное значение  $x_0$ .

Выходом генератора на  $t$ -м шаге является младший бит числа  $x_t$ .

**!** **Безопасность генератора опирается на сложность взлома алгоритма RSA, т. е. на разложение числа  $n$  на простые сомножители.**

### 6.1.2.4. Генератор псевдослучайных чисел на основе алгоритма BBS

Широкое распространение получил алгоритм генерации ПСП, называемый алгоритмом BBS (от фамилий авторов: L. Blum, M. Blum, M. Shub) или *генератором на основе квадратичных вычислений*. Для целей криптографии этот метод предложен в 1986 г.

Начальное значение  $x_0$  генератора вычисляется на основе соотношения

$$x_0 \equiv x^2 \pmod{n}, \quad (6.3)$$

где  $n$ , как и в генераторе на основе RSA, является произведением простых чисел  $p$  и  $q$ , однако в нашем случае эти простые числа должны быть сравнимы с числом 3 по модулю 4, т. е. при делении  $p$  и  $q$  на 4 должен получаться одинаковый остаток 3; число  $x$  должно быть взаимно простым с  $n$ ; число  $n$  называют **числом Блюма**.

Выходом генератора на  $t$ -м шаге является младший бит числа  $x_t$ :

$$x_t \equiv (x_{t-1})^2 \pmod{n}. \quad (6.4)$$

Рассмотрим пример.

**Пример 1.** Пусть  $p = 11$ ,  $q = 19$  (убеждаемся, что  $11 \pmod{4} = 3$ ,  $19 \pmod{4} = 3$ ). Тогда  $n = pq = 11 \cdot 19 = 209$ . Выберем  $x$ , взаимно простое с  $n$ : пусть  $x = 3$  [35].

Вычислим стартовое число генератора  $x_0$  в соответствии с (6.3):

$$x_0 = x^2 \pmod{n} = 3^2 \pmod{209} = 9 \pmod{209} = 9.$$

Вычислим первые десять чисел  $x_t$  по алгоритму BBS.

В качестве случайных битов будем брать младший бит в двоичной записи числа  $x_t$  (табл. 6.1; здесь в вычислениях используется обычный знак равенства).

Таблица 6.1  
Пояснение к методу генерации ПСП на основе метода BBS

Число $x_t$	Младший бит двоичного числа $x_t$
$x_1 = 9^2 \pmod{209} = 81 \pmod{209} = 81$	1
$x_2 = 81^2 \pmod{209} = 6561 \pmod{209} = 82$	0
$x_3 = 82^2 \pmod{209} = 6724 \pmod{209} = 36$	0
$x_4 = 36^2 \pmod{209} = 1296 \pmod{209} = 42$	0
$x_5 = 42^2 \pmod{209} = 1764 \pmod{209} = 92$	0
$x_6 = 92^2 \pmod{209} = 8464 \pmod{209} = 104$	0
$x_7 = 104^2 \pmod{209} = 10816 \pmod{209} = 157$	1
$x_8 = 157^2 \pmod{209} = 24649 \pmod{209} = 196$	0
$x_9 = 196^2 \pmod{209} = 38416 \pmod{209} = 169$	1
$x_{10} = 169^2 \pmod{209} = 28561 \pmod{209} = 137$	1

С точки зрения безопасности важным является свойство рассмотренного генератора, заключающееся в том, что при известных  $p$  и  $q$   $x_t$ -й бит легко вычисляется без учета предыдущего ( $x_{t-1}$ ) бита:

$$x_t \equiv (x_0)^a \pmod{n}, \quad (6.4)$$

где  $a \equiv 2^t \pmod{(p-1)(q-1)}$ .

Алгоритм является сравнительно медленным. Для ускорения можно использовать не последний бит числа, а несколько последних битов. Однако понятно, что при этом алгоритм является менее криптостойким.

### 6.1.3. Потоковый шифр RC4

Алгоритм RC4 разработан Р. Ривестом в 1987 г. Он представляет собой потоковый шифр с переменным размером ключа. Здесь гамма не зависит от открытого текста [5].

Алгоритм RC4, как и любой потоковый шифр, строится на основе генератора псевдослучайных битов (генератора ПСП). На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Длина ключа может составлять от 40 до 2048 битов.

Ядро алгоритма состоит из функции генерации ключевого потока. Другая часть алгоритма – функция инициализации, которая использует *ключ переменной длины*  $K_i$  для создания начального состояния генератора ключевого потока.

В основе алгоритма – размер блока или слова, определяемый параметром  $n$ . Обычно  $n = 8$ , но можно использовать и другие значения. Внутренне состояние шифра определяется массивом слов ( $S$ -блоком) размером  $2^n$ . При  $n = 8$  элементы блока представляют собой перестановку чисел от 0 до 255, а сама перестановка зависит от ключа переменной длины. Другими элементами внутреннего состояния являются 2 счетчика (каждый размером в одно слово; обозначим их  $i$  и  $j$ ) с нулевыми начальными значениями. В основе вычислений лежит операция по  $\text{mod } 2^n$ .

Генератор ключевого потока RC4 переставляет значения, хранящиеся в  $S$ , и каждый раз выбирает различное значение из  $S$  в качестве результата. В одном цикле RC4 определяется одно  $n$ -битное слово  $K$  из ключевого потока, которое в последующем суммируется с исходным текстом для получения зашифрованного текста. Эта часть алгоритма называется генератором ПСП. При  $n = 8$  для генерации случайного байта выполняются операции, представленные листингом 6.1.

```
:: i = (i + 1) mod 256;
:: j = (j + Si) mod 256;
поменять местами Si и Sj;
a = (Si + Sj) mod 256;
K = Sa
```

**Листинг 6.1.** Псевдокод для генерации байта ПСП

Байт  $K$  используется в операции XOR с открытым текстом для получения 8-битного шифртекста или для его расшифрования.

Так же достаточно проста и инициализация  $S$ -блока. Этот алгоритм использует ключ, который подается на вход пользователем. Сначала  $S$ -блок заполняется линейно:  $S_0 = 0, S_1 = 1, \dots, S_{255} = 255$ . Затем заполняется секретным ключом другой 256-байтный массив. Если необходимо, ключ повторяется многократно, чтобы заполнить весь массив:  $K_0, K_1, \dots, K_{255}$ . Далее массив  $S$  перемешивается путем перестановок, определяемых ключом. Действия выполняются в соответствии с псевдокодом, представленным листингом 6.2.

- 1.  $j = 0; i = 0;$
- 2.  $j = (j + S_i + K_i) \bmod 256;$
- 3. поменять местами  $S_i$  и  $S_j$ ;
- 4.  $i = i + 1;$
- 5. если  $i < 256$ , то перейти на п.2

**Листинг 6.2.** Псевдокод для начального заполнения таблицы замен  $S$  ( $S$ -блока)

Проиллюстрируем работу алгоритма для случая  $n = 4$ , воспользовавшись примером из [36].

**Пример 2.** Предположим, что секретный ключ состоит из шести 4-битных значений (приведем их в десятичном виде): 1, 2, 3, 4, 5, 6. Для его получения сгенерируем последовательность чисел, для чего заполним таблицу  $S$  линейно (последовательно) 16 ( $2^4$ ) числами от 0 до 15 в соответствии с табл. 6.2.

Таблица 6.2  
Начальное заполнение таблицы  $S$

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

Подготовим таблицу  $K$ , записав в нее ключ необходимое количество раз (табл. 6.3).

Таблица 6.3  
Заполнение таблицы  $K$

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

Перемешаем содержимое таблицы  $S$ . Для этого будем использовать алгоритм в соответствии с листингом 6.2 (для  $n = 4$ ). Процесс выполнения представим в виде трассировочной таблицы (табл. 6.4).

Таблица 6.4  
Подготовительный этап (инициализация таблицы замен) алгоритма RC4

Номер пункта алгоритма	Выполняемое действие (по mod 16)	Новое значение $i$	Новое значение $j$
1	$j = 0; i = 0$	0	
2	$j = j + S_i + K_i = 0 + 0 + 1 = 1$		1
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_0$ и $S_1$		
4	$i = i + 1$	1	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = j + S_i + K_i = 1 + 0 + 2 = 3$		3
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_1$ и $S_3$		
4	$i = i + 1$	2	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (3 + 2 + 3) \bmod 16 = 8$		8
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_2$ и $S_8$		
4	$i = i + 1$	3	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (8 + 0 + 4) \bmod 16 = 12$		12
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_3$ и $S_{12}$		
4	$i = i + 1$	4	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (12 + 4 + 5) \bmod 16 = 5$		5
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_4$ и $S_5$		
4	$i = i + 1$	5	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (5 + 4 + 6) \bmod 16 = 15$		15
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_5$ и $S_{15}$		
4	$i = i + 1$	6	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (15 + 6 + 1) \bmod 16 = 6$		6
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_6$ и $S_6$		
4	$i = i + 1$	7	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (6 + 7 + 2) \bmod 16 = 15$		15
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_7$ и $S_{15}$		
4	$i = i + 1$	8	
5	$I < 16$ , поэтому перейти на п. 2		

## Окончание табл. 6.4

Номер пункта алгоритма	Выполняемое действие (по mod 16)	Новое значение $i$	Новое значение $j$
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (15 + 2 + 3) \bmod 16 = 4$		4
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_8$ и $S_4$		
4	$i = i + 1$	9	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (4 + 9 + 4) \bmod 16 = 1$		1
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_9$ и $S_1$		
4	$i = i + 1$	10	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (1 + 10 + 5) \bmod 16 = 0$		0
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_{10}$ и $S_0$		
4	$i = i + 1$	11	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (0 + 11 + 6) \bmod 16 = 1$		1
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_{11}$ и $S_1$		
4	$i = i + 1$	12	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (1 + 0 + 1) \bmod 16 = 2$		2
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_{12}$ и $S_2$		
4	$i = i + 1$	13	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (2 + 13 + 2) \bmod 16 = 1$		1
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_{13}$ и $S_1$		
4	$i = i + 1$	14	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (1 + 14 + 3) \bmod 16 = 2$		2
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_{14}$ и $S_2$		
4	$i = i + 1$	15	
5	$I < 16$ , поэтому перейти на п. 2		
2	$j = (j + S_i + K_i) \bmod 16 =$ $= (2 + 7 + 4) \bmod 16 = 13$		13
3	Поменять местами $S_i$ и $S_j$ , т. е. $S_{15}$ и $S_{13}$		
4	$i = i + 1$	16	
5	$I < 16$ – неверно, поэтому закончить		

После этого получим инициализированную и подготовленную к основному этапу таблицу  $S$  (табл. 6.5).

Таблица 6.5

Таблица S

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

Далее выполняем генерацию случайных 4-битных слов. Вычислим первые 5 чисел псевдослучайной последовательности, используя алгоритм в листинге 6.1 (для  $n = 4$ ). Результаты вычисления последовательности значений также представим в виде таблицы (табл. 6.6).

Таблица 6.6

Вычисление элементов ПСП ( $K_i$ ) на основе алгоритма RC4

Вычисление $K_i$	Выполняемое действие (по mod 16)	Новое значение $i$	Новое значение $j$	Новое значение $a$
$K_1$	1. $i = (i + 1) = 0 + 1 = 1$	1		
	2. $j = (j + S_i) \text{ mod } 16 = (0 + 13) \text{ mod } 16 = 13$		13	
	3. Поменять местами $S_1$ и $S_{13}$			
	4. $a = (S_i + S_j) \text{ mod } 16 = (7 + 13) \text{ mod } 16 = 4$			4
	5. $K_1 = S_4 = 2$			
$K_2$	1. $i = (i + 1) = 1 + 1 = 2$	2		
	2. $j = (j + S_i) \text{ mod } 16 = (13 + 14) \text{ mod } 16 = 11$		11	
	3. Поменять местами $S_2$ и $S_{11}$			
	4. $a = (S_i + S_j) \text{ mod } 16 = (9 + 14) \text{ mod } 16 = 7$			7
	5. $K_2 = S_7 = 4$			
$K_3$	1. $i = (i + 1) = 2 + 1 = 3$	3		
	2. $j = (j + S_i) \text{ mod } 16 = (11 + 12) \text{ mod } 16 = 7$		7	
	3. Поменять местами $S_3$ и $S_7$			
	4. $a = (S_i + S_j) \text{ mod } 16 = (4 + 12) \text{ mod } 16 = 0$			0
	5. $K_3 = S_0 = 10$			
$K_4$	1. $i = (i + 1) = 3 + 1 = 4$	4		
	2. $j = (j + S_i) \text{ mod } 16 = (7 + 2) \text{ mod } 16 = 9$		9	
	3. Поменять местами $S_4$ и $S_9$			
	4. $a = (S_i + S_j) \text{ mod } 16 = (3 + 2) \text{ mod } 16 = 5$			5
	5. $K_4 = S_5 = 15$			

Окончание табл. 6.6

Вычисление $K_i$	Выполняемое действие (по mod 16)	Новое значение $i$	Новое значение $j$	Новое значение $a$
$K_5$	1. $i = (i + 1) = 4 + 1 = 5$	5		
	2. $j = (j + S_i) \text{ mod } 16 = (9 + 15) \text{ mod } 16 = 8$		8	
	3. Поменять местами $S_5$ и $S_8$			
	4. $a = (S_i + S_j) \text{ mod } 16 = (5 + 15) \text{ mod } 16 = 4$			4
	5. $K_5 = S_4 = 3$			

В результате первые пять значений гаммы получились следующие: 2, 4, 10, 15, 3. При необходимости получения большего количества случайных чисел можно продолжить вычисления дальше. При  $n = 4$  генерируемые числа будут иметь размер 4 бита, т. е. для нашего примера: 0010, 0100, 1010, 1111, 0011.

Примеры программной реализации RC4 можно найти, в частности, в [37].

По утверждению [5], рассмотренный алгоритм устойчив к линейному и дифференциальному криптоанализу. Кроме того, при  $n = 8$  этот алгоритм может находиться примерно в  $2^{1700}$  состояниях ( $256! \cdot 256^2$ ).

При  $n = 16$  алгоритм должен обладать большей в сравнении с  $n = 8$  скоростью, но при этом начальные установки занимают гораздо больше времени – нужно заполнить 65 536-элементный массив.

**!** Шифр RC4 применяется в некоторых широко распространенных стандартах и протоколах шифрования, таких как WEP, WPA и TLS, а также в Kerberos и др.

Реализация алгоритма на Python приведена в листинге 6.3.

```
## rc4.py - stream cipher RC4

def RC4crypt(data, key):
    """RC4 algorithm"""
    x = 0
    box = range(256)
    for i in range(256):
        x = (x + box[i] + ord(key[i % len(key)])) % 256
        box[i], box[x] = box[x], box[i]
    x = y = 0
    out = []
    for char in data:
```

```
x = (x + 1) % 256
y = (y + box[x]) % 256
box[x], box[y] = box[y], box[x]
out.append(chr(ord(char) ^ box[(box[x] + box[y])
% 256]))
return ''.join(out)

def hexRC4crypt(data, key):
    """hex RC4 algorithm"""
    dig = crypt(data, key)
    tempstr = ''
    for d in dig:
        xxx = '%02x' % (ord(d))
        tempstr = tempstr + xxx

    return tempstr

assert hexRC4crypt('The quick brown fox jumps over the
lazy dog', '123456') == \
'54901b4755467cff141740c496492fee8ba7224a99f52cc2e84d019e
1c0cda32b6a96d521d067d00ce6716'
assert
RC4crypt('\xA4\x8F\x9B\xFF\x6D\x3A\xFD\x7D\x56\xCB\xAC\xD
6\x46\x27\x50\x65', '10001') == \
'Wow, you did it!'
```

**Листинг 6.3.** Реализация RC4 на Python

Другими известными потоковыми шифрами являются, например, SEAL, программный код которого приведен в [5], и WAKE.

Наилучшие характеристики будут иметь генераторы случайных чисел, основанные на «естественных случайностях», свойственных, например, процессам в радиоэлектронной аппаратуре, в системах телекоммуникаций, в приемах работы с клавиатурой операторов и др.

## 6.2. Практическое задание

1. Разработать авторские многооконные приложения в соответствии с целью лабораторной работы. При этом можно воспользоваться готовыми библиотеками либо программными кодами, реализующими заданные алгоритмы.

Приложение 1 должно реализовывать генерацию ПСП в соответствии с вариантом из табл. 6.7.

Таблица 6.7  
**Варианты для приложения 1**

Вариант задания	Алгоритм генерации ПСП	Параметры
1	RSA	$p, q, e$ – 512-разрядные числа (обосновать выбор)*
2	RSA	$p, q, e$ – 256-разрядные числа (обосновать выбор)*
3	BBS	$n = 256$ ; $p, q$ – обосновать выбор по согласованию с преподавателем
4	BBS	$n = 512$ ; $p, q$ – обосновать выбор по согласованию с преподавателем
5	Линейный конгруэнтный генератор	$a = 421, c = 1663, n = 7875$
6	Линейный конгруэнтный генератор	$a = 430, c = 2531, n = 11\,979$

\* Можно воспользоваться результатами выполнения лабораторной работы № 1.

Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.8, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.

Таблица 6.8  
**Варианты для приложения 2**

Вариант задания	$n$	Ключ (в виде десятичных чисел)
1	8	121, 14, 89, 15
2	8	76, 111, 85, 54, 211
3	8	43, 45, 100, 21, 1
4	8	12, 13, 90, 91, 240
5	8	123, 125, 41, 84, 203
6	6	1, 11, 21, 31, 41, 51
7	6	10, 11, 12, 13, 14, 15
8	6	20, 21, 22, 23, 60, 61
9	6	61, 60, 23, 22, 21, 20
10	6	15, 14, 13, 12, 11, 10
11	8	13, 19, 90, 92, 240
12	8	122, 125, 48, 84, 201
13	8	61, 60, 23, 22, 21, 20
14	8	20, 21, 22, 23, 60, 61
15	8	1, 11, 21, 31, 41, 51

В качестве шифруемого сообщения может быть выбран произвольный текст.

2. Результаты оформить в виде отчета по установленным правилам.

## **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. В чем состоит особенность потоковых шифров?

2. В чем состоят преимущества и недостатки синхронных и асинхронных потоковых шифров?

3. Какими свойствами должен обладать генератор псевдослучайных чисел для использования в криптографических целях?

4. Дать характеристику линейным конгруэнтным генераторам. Области их применения.

5. Значения  $x_0, x_1, x_2, x_3$ , полученные с помощью линейного конгруэнтного генератора, равны соответственно: 1, 12, 3, 6. Найти параметры  $a, c$  и  $n$  генератора ПСЧ, удовлетворяющие выражению (6.1).

6. Представить общую структурную схему генератора ПСП на основе регистров сдвига с линейной обратной связью. Пояснить особенности его функционирования.

7. Синтезировать структурную схему генератора ПСП на основе регистров сдвига с линейной обратной связью, формально обозначаемого следующим образом: а) 3210; б) 420; в) 5410; г) 520; д) 84 320. Составить таблицу состояний генератора и определить период ПСП.

8. Определить первые 12 бит ПСП, задаваемого формально в виде чисел 5410, если начальные состояния ячеек (слева направо) соответствуют последовательности 10101.

9. Как устроен генератор ПСП на основе RSA? На чем основана криптостойкость реализуемого алгоритма?

10. Вычислить  $x_1, x_5, x_9, x_{11}$  по методу генерации псевдослучайных чисел BBS, если  $p = 11, q = 19, x = 3$ .

11. Пояснить базовый алгоритм, реализованный в шифре RC4.

12. Пояснить принципы формирования истинных случайных последовательностей, основанных на «естественных случайностях».

## **Лабораторная работа № 7**

### **ИССЛЕДОВАНИЕ АСИММЕТРИЧНЫХ ШИФРОВ**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров.
2. Разработать приложение для реализации указанных преподавателем методов генерации ключевой информации и ее использования для асимметричного зашифрования/расшифрования.
3. Выполнить анализ криптостойкости асимметричных шифров.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

#### **7.1. Теоретические сведения**

##### **7.1.1. Основные свойства асимметричных крипtosистем**

Две известные нам проблемы, связанные с практическим использованием симметричных крипосистем, стали важными побудительными мотивами для разработки принципиально нового класса методов шифрования: криптографии с *открытым* ключом, или *асимметричной криптографии*.

Концепция нового подхода предложена Уитфилдом Диффи (Whitfield Diffie) и Мартином Хеллманом (Martin Hellman) и, независимо, Ральфом Мерклом (Ralph Merkle).

**! В основу асимметричной криптографии положена идея использовать ключи парами: один – для зашифрования**

(открытый, или публичный, ключ), другой – для расшифрования (тайный ключ). Отметим, что указанная пара ключей принадлежит получателю зашифрованного сообщения. Все алгоритмы шифрования с открытым ключом основаны на использовании односторонних функций, к числу которых, как известно, относится вычисление дискретного логарифма.

**Определение 1. Односторонней функцией** (one-way function) называется математическая функция, которую относительно легко вычислить, но трудно найти по значению функции соответствующее значение аргумента, т. е. зная  $x$ , легко вычислить  $f(x)$ , но по известному  $f(x)$  трудно найти подходящее значение  $x$ .

Практически первой реализацией идеи Диффи – Хеллмана стал алгоритм согласования по открытому каналу тайного ключа между абонентами  $A$  и  $B$  [2, 4].

Алгоритмы шифрования с открытым ключом можно использовать для решения следующих задач:

- зашифрования/расшифрования передаваемых и хранимых данных в целях их защиты от несанкционированного доступа;
- формирования цифровой подписи под электронными документами;
- распределения секретных ключей, используемых далее при шифровании документов симметричными методами.

В данной работе мы будем работать над аспектами решения первой из указанных задач.

По мнению Диффи и Хеллмана, алгоритм шифрования с открытым ключом должен:

- вычислительно легко создавать пару (открытый ключ  $e$  – закрытый ключ  $d$ );
- вычислительно легко зашифровывать сообщение  $M_i$  открытым ключом;
- вычислительно легко расшифровывать сообщение  $C_i$ , используя закрытый ключ;
- обеспечивать непреодолимую вычислительную сложность определения соответствующего закрытого ключа при известном открытом ключе;
- обеспечивать непреодолимую вычислительную сложность восстановления исходного (открытого сообщения  $M_i$ ) зная только открытый ключ и зашифрованное сообщение  $C_i$ .

### 7.1.2. Криптоалгоритм на основе задачи об укладке ранца

#### 7.1.2.1. Общая характеристика алгоритма

Алгоритм разработан Р. Мерклом и М. Хеллманом. Это первый алгоритм шифрования с открытым ключом широкого назначения.

**Определение 2.** *Ранцевый (рюкзачный) вектор  $S = (s_1, \dots, s_z)$  – это упорядоченный набор из  $z$ ,  $z \geq 3$ , различных натуральных чисел  $s_i$ . Входом задачи о ранце (рюкзаке) называем пару  $(S, S)$ , где  $S$  – рюкзачный вектор, а  $S$  – натуральное число.*

Решением для входа  $(S, S)$  будет такое подмножество из  $S$ , сумма элементов которого равняется  $S$ .

В наиболее известном варианте задачи о ранце требуется выяснить, обладает или нет данный вход  $(S, S)$  решением. В варианте, используемом в криптографии, нужно для данного входа  $(S, S)$  построить решение, зная, что такое решение существует. Оба эти варианта являются NP-полными.

Имеются также варианты этой задачи, которые не лежат даже в классе NP.

Как видим, проблема укладки ранца формулируется просто. Дано множество предметов общим числом  $z$  различного веса. Спрашивается, можно ли положить некоторые из этих предметов в ранец так, чтобы его вес стал равен определенному значению  $S$ ? Более формально задача формулируется так: дан набор значений  $k_1, k_2, \dots, k_z$  и суммарное значение  $S$ . Требуется вычислить значения  $b_z$  такие, что

$$S = b_1k_1 + b_2k_2 + \dots + b_zk_z. \quad (7.1)$$

Здесь  $b_i$  может быть либо нулем, либо единицей. Значение  $b_i = 1$  означает, что предмет  $m_i$  кладут в рюкзак, а  $b_i = 0$  – не кладут.



**Суть метода для шифрования состоит в том, что существуют две различные задачи укладки ранца: одна из них решается легко и характеризуется линейным ростом трудоемкости, а другая решается трудно. Легкий для укладки ранец можно трансформировать в трудный.**

**Трудный для укладки ранец применяется в качестве открытого ключа, который легко использовать для зашифрования, но невозможно – для расшифрования. В качестве закрытого ключа применяется легкий для укладки ранец, который предоставляет простой способ расшифрования сообщения.**

В качестве закрытого ключа  $d$  (легкого для укладки ранца) используется *сверхвозрастающая последовательность, состоящая из  $z$  элементов*:  $d_1, d_2, \dots, d_z$ :  $d = \{d_i\}, i = 1, \dots, z$ .

**Определение 3.** *Сверхвозрастающей* называется последовательность, в которой каждый последующий член больше суммы всех предыдущих.

**Пример 1.** Последовательность  $\{2, 3, 6, 13, 27, 52, 105, 210\}$  является сверхвозрастающей, а  $\{1, 3, 4, 9, 15, 25, 48, 76\}$  – нет.

### 7.1.2.2. Алгоритм укладки ранца на основе сверхвозрастающей последовательности

Необходимо по очереди анализировать некоторый «текущий вес»  $S$  предметов, составляющих сверхвозрастающую последовательность; в результате анализа нужно упаковать (доупаковать) ранец.

1. В качестве текущего выбирается число  $S$ , которое сравнивается с «весом» самого тяжелого предмета ( $d_z$ ); если текущий вес меньше веса данного предмета, то его в ранец не кладут (0), в противном случае его укладывают (1) в ранец и переходят к анализу очередного (в общем случае –  $i$ -го предмета).

2. Если на предыдущем ( $i$ -м шаге) предмет пополнил ранец, то текущий вес уменьшают на вес положенного предмета ( $S = S - d_i$ ); переходят к следующему по весу предмету в последовательности:  $d_{i-1}$ .

Шаги повторяются до тех пор, пока процесс не закончится.

Если текущий вес уменьшится до нуля ( $S = 0$ ), то решение найдено. В противном случае – нет.

**Пример 2.** Пусть полный вес ранца равен 270, а последовательность весов предметов равна:  $\{2, 3, 6, 13, 27, 52, 105, 210\}$  ( $d_1 = 2, d_2 = 3, d_3 = 6$  и т. д.).

Шаг 1.  $S = 270$ . Самый большой вес предмета ( $d_z = d_8 = 210$ ) – 210. Он меньше 270, поэтому предмет весом 210 кладут в ранец (1): вычитывают 210 из 270 и получают 60:  $S = S - d_8 = 270 - 210 = 60$ .

Шаг 2. Следующий наибольший вес последовательности равен 105. Он больше 60, поэтому предмет весом 105 в ранец не кладут (0):  $S = S - 0 = 60$ .

Шаг 3. Следующий самый тяжелый предмет имеет вес 52. Он меньше 60, поэтому предмет весом 52 также кладут в рюкзак (1):  $S = S - 52 = 8$ .

На 4-м и на 5-м шагах рюкзак не пополняется. Текущий вес его остается неизменным.

На 6-м шаге в ранец кладут предмет весом 6 и на 8-м шаге – весом 2.

В результате полный вес уменьшится до 0, т. е. получили текущее значение  $S = 0$ .

Если бы этот ранец был использован для расшифрования, то открытый текст, полученный из значения шифр текста 270, был бы равен 10100101.

Открытый ключ  $e$  представляет собой нормальную (не сверхвозрастающую) последовательность. Он формируется на основе закрытого ключа и не позволяет легко решить задачу об укладке ранца.

Для получения открытого ключа  $e$  ( $e = \{e_i\}$ ,  $i = 1, \dots, z$ ) все значения закрытого ключа умножаются на некоторое число  $a$  по модулю  $n$ :

$$e_i \equiv d_i a \pmod{n}. \quad (7.2)$$

Значение модуля  $n$  должно быть больше суммы всех чисел последовательности; кроме того,  $\text{НОД}(a, n) = 1$ .

**Пример 3.** Сумма элементов последовательности  $\{2, 3, 6, 13, 27, 52, 105, 210\}$  равна 418:  $2 + 3 + 6 + 13 + 27 + 52 + 105 + 210 = 418$ . Элементы  $d_i$  этой последовательности являются элементами ключа  $d$ :  $d = \{d_i\}$ . Примем, что  $n = 420$  и  $a = 31$ .

В соответствии с этим при использовании выражения (7.2) результат построения нормальной последовательности (открытого ключа  $e$ ) будет представлен следующим множеством:  $\{62, 93, 186, 403, 417, 352, 315, 210\}$ :  $e_1 = 62$ ,  $e_2 = 93$  и т. д.

### 7.1.2.3. Зашифрование сообщения

В основе операции лежит соотношение (7.1).

Для зашифрования сообщения ( $M$ ) оно сначала разбивается на блоки, по размерам равные числу ( $z$ ) элементов последовательности в ранце. Затем, считая, что 1 указывает на присутствие элемента последовательности в ранце, а 0 – на его отсутствие, вычисляются полные веса рюкзаков ( $S_i$ ,  $i = 1, \dots, z$ ): по одному ранцу для каждого блока сообщения с использованием открытого ключа получателя  $e$ .

**Пример 4.** Возьмем открытое сообщение  $M$ , состоящее из 7 букв ( $m_j$ ), которые представим в бинарном виде (1 символ текста – 1 байт). Бинарное представление символов дано в первом столбце табл. 7.1.

Открытый ключ  $e: \{62, 93, 186, 403, 417, 352, 315, 210\}$ .

Результат зашифрования (упаковки ранца) каждого блока (буквы) сообщения с помощью открытого ключа представлен в правом столбце таблицы.

#### Пояснение к примеру зашифрования сообщения укладкой ранца

Бинарный код символа $m_j$ сообщения	Укладка ранца	Вес ранца
11010000	$62 + 93 + 403$	558
11000010	$62 + 93 + 315$	470
11000000	$62 + 93$	155
11000001	$62 + 93 + 210$	365
11001110	$62 + 93 + 417 + 352 + 315$	1239
11000000	$62 + 93$	155
11001100	$62 + 93 + 417 + 352$	924

Таким образом, зашифрованное сообщение  $C = 558\ 470\ 155\ 365\ 1239\ 155\ 924$ :  $c_1 = 558$ ,  $c_2 = 470$  и т. д.

#### 7.1.2.4. Расшифрование сообщения

Для расшифрования сообщения получатель (используя свой тайный ключ  $d$ : сверхвозрастающую последовательность) должен сначала определить такое обратное к  $a$  число  $a^{-1}$ , что

$$aa^{-1} \bmod n \equiv 1. \quad (7.3)$$

Для вычисления обратных чисел по модулю можно использовать известный нам расширенный алгоритм Евклида.

После определения обратного числа каждое значение шифрограммы ( $c_i$ ) преобразуется в соответствии со следующим соотношением:

$$S_i \equiv c_i a^{-1} \bmod n. \quad (7.4)$$

Полученное на основании последней формулы для каждого блока число далее рассматривается как заданный вес ранца, который следует упаковать по изложенному выше алгоритму, используя сверхвозрастающую последовательность (тайный ключ получателя).

Продолжим рассмотрение *примера 4*.

В нашем примере значение  $a^{-1} = 271 / 31 \cdot 271 \bmod 420 = 1$ .

Вспомним, что сверхвозрастающая последовательность равна  $d$ :  $d = \{2, 3, 6, 13, 27, 52, 105, 210\}$ , а также  $n = 420$ ,  $a = 31$ ; шифртекст:  $C = 155\ 365\ 558\ 155\ 924\ 1239\ 470$ .

Расшифрование первого блока шифртекста. Сначала вычисляем, используя (7.4), вес первого ранца (при  $c_1 = 155$ ):

$$S_1 = c_1 a^{-1} \bmod n = 155 \cdot 271 \bmod 420 = 5.$$

Используем  $S_1 = 5$  и с помощью сверхвозрастающей последовательности ( $\{2, 3, 6, 13, 27, 52, 105, 210\}$ ) и известного алгоритма упаковки ранца получаем  $m_1 = 11000000$ . Понятно, что последней бинарной последовательности должен соответствовать некоторый символ алфавита в используемой таблице кодировки.

Расшифрование остальных блоков шифртекста производится аналогично.

### 7.1.3. Безопасность криptoалгоритма на основе задачи об укладке ранца

Криптостойкость алгоритма во многом определяется скоростью (временем) поиска нужного варианта укладки ранца. Понятно, что для последовательности из шести-десяти или немногим более того элементов нетрудно решить задачу укладки ранца, даже если последовательность не является сверхвозрастающей. При практической же реализации алгоритма ранец должен содержать не менее нескольких сотен элементов. Длина каждого члена сверхвозрастающей последовательности должна быть несколько сотен бит, а длина числа  $n$  – от 100 до 200 бит. Для получения этих значений практические реализации алгоритма используют генераторы ПСП.

С другой стороны, известным способом определения того, какие предметы кладутся в ранец, является проверка возможных решений до получения правильного. Самый быстрый алгоритм, принимая во внимание различную эвристику, имеет экспоненциальную зависимость от числа возможных предметов. Если добавить к последовательности весов еще один член, то найти решение станет вдвое труднее. Это намного труднее сверхвозрастающего ранца, где при добавлении к последовательности одного элемента поиск решения увеличивается на одну операцию.

**! Ранцевые криптосистемы не являются криптостойкими.**  
**А. Шамир и Р. Циппел обнаружили, что зная числа  $a$ ,  $a^{-1}$  и  $n$  («секретную лазейку»), можно восстановить сверхвозрастающую последовательность по нормальной последовательности [5].**

Важно то, что числа  $a$  и  $n$  («секретная пара») не обязательно должны быть теми же, что использовались при создании системы легальным пользователем.

Достаточно подробное и понятное для начинающего криptoаналитика рассмотрение криптостойкости ранцевого алгоритма изложено в [38].

## 7.2. Практическое задание

1. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами.

В основе вычислений – кодировочные таблицы Base64 и ASCII.

Приложение должно реализовывать следующие операции:

- генерация сверхвозрастающей последовательности (тайного ключа); старший член последовательности – 100-битное число; в простейшем случае принимается  $z = 6$  (для кодировки Base64) и  $z = 8$  (для кодировки ASCII);

- вычисление нормальной последовательности (открытого ключа);
- зашифрование сообщения, состоящего из собственных фамилии, имени и отчества;
- расшифрование сообщения;
- оценка времени выполнения операций зашифрования и расшифрования.

2. Проанализировать время выполнения операций зашифрования/расшифрования при увеличении числа членов ключевой последовательности; при использовании разных таблиц кодировки.

3. Результаты оформить в виде отчета по установленным правилам.

## ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Что такое «ранцевый (рюкзачный) вектор»? Дать определение.
2. Сформулировать задачу укладки ранца.
3. Если вектор рюкзака имеет вид (14, 28, 56, 82, 90, 132, 197, 284, 341, 455), то какими следует принять коэффициенты  $b_i$  из выражения (7.1), чтобы получить  $S = 517$ ? Каким будет решение задачи для  $S = 516$ ?

4. Что такое сверхвозрастающая последовательность? Привести примеры.

5. Можно ли последовательности чисел:  $\{89, 3, 11, 2, 45, 6, 22\}$ ,  $\{3, 41, 5, 1, 21, 10\}$ ,  $\{2, 3, 11, 29, 45, 6, 39\}$  преобразовать в сверхвозрастающие?

6. Записать в виде псевдокода алгоритм зашифрования и алгоритм расшифрования сообщения на основе задачи об укладке ранца.

7. Используя некоторый вектор  $S = (103, 107, 211, 430, 863, 1716, 3449, 6907, 13807, 27610)$ , вычислить ключи для зашифрования и расшифрования сообщений.

8. Можно ли, с Вашей точки зрения, одновременно зашифровывать (и, соответственно, одновременно расшифровывать) более, чем по одному символу текста? Обосновать решение. Привести примеры решений.

9. Что такое «секретная лазейка»?

10. Охарактеризовать криптостойкость алгоритма на основе задачи об укладке ранца.

## **Лабораторная работа № 8**

### **Исследование асимметричных шифров RSA и Эль-Гамаля**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации асимметричных шифров RSA и Эль-Гамаля.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости асимметричных шифров RSA и Эль-Гамаля.
2. Разработать приложение для реализации асимметричного зашифрования/расшифрования на основе алгоритмов RSA и Эль-Гамаля.
3. Выполнить анализ криптостойкости асимметричных шифров RSA и Эль-Гамаля.
4. Оценить скорость зашифрования/расшифрования реализованных шифров.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

#### **8.1. Теоретические сведения**

##### **8.1.1. Математические основы асимметричных шифров**



**Как отмечалось выше, асимметричная криптография основана на сложности решения некоторых математических задач. По существу, таких задач две:**

- разложение больших чисел на простые сомножители (задача факторизации);
- вычисление дискретного логарифма в конечном поле, а также вычислительные операции над точками эллиптической кривой.

Эти задачи объединяет то, что они используют операцию получения остатка от целочисленного деления.

В силу этого практически все системы асимметричного зашифрования/расшифрования основаны либо на проблеме факторизации (среди них – RSA), либо на проблеме дискретного логарифмирования (среди них – Эль-Гамаля).

Базовые элементы перечисленных проблем мы рассмотрели с практической точки зрения при выполнении лабораторной работы № 1. Алгебраическая теория рассматриваемого класса крипtosистем подробно изучена в [39]. Мы же здесь остановимся лишь на нескольких основных элементах этой теории.

**Теорема 1:** основная теорема арифметики. Всякое натуральное число  $N$ , кроме 1, можно представить как произведение простых множителей:

$$N = p_1 p_2 p_3 \dots p_z, z > 1. \quad (8.1)$$

**Определение 1.** Задача дискретного логарифмирования формулируется так: для данных целых чисел  $a$  и  $b$ ,  $1 < a, b < n$ , найти логарифм – такое целое число  $x$ , что

$$a^x \equiv b \pmod{n}, \quad (8.2)$$

если такое число существует.

По аналогии с вещественными числами используется обозначение  $x = \log_a b$ .

**Теорема 2:** китайская теорема об остатках. В общем случае если разложение числа  $N$  на простые множители представляет собой  $p_1 p_2 \dots p_t$  (некоторые простые числа могут встречаться несколько раз), то система уравнений

$$(x \bmod p_i) \equiv a_i, \quad (8.3)$$

где  $i = 1, 2, \dots, t$ , имеет единственное решение:  $x$ , меньшее  $N$ .

Иными словами, число (меньшее, чем произведение нескольких простых чисел) однозначно определяется своими вычетами по модулю от этих простых чисел. Китайской теоремой об остатках можно воспользоваться для решения полной системы уравнений в том случае, если известно разложение числа  $N$  на простые множители.

### 8.1.2. Алгоритм RSA

Рассматриваемый алгоритм появился (1977 г.) после алгоритма ранца Меркла. Он стал первым полноценным алгоритмом с открытым

ключом, который впоследствии стал одним из основных для шифрования и для электронных цифровых подписей.

Из всех предложенных алгоритмов с открытыми ключами RSA проще всего понять и реализовать. Он назван в честь трех его создателей: Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Эдлемана (Leonard Adleman).

Как было отмечено, безопасность RSA основана на трудности разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших простых чисел. Предполагается, что восстановление открытого текста по шифртексту и открытому ключу эквивалентно разложению на множители двух больших чисел.

Для генерации двух ключей: тайного и открытого (а по сути – двух взаимосвязанных частей одного ключа, т. е. ключа, принадлежащего одному физическому лицу (или группе лиц), либо одному юридическому лицу), используются два больших случайных простых числа  $p$  и  $q$ . Для максимальной большей криптостойкости нужно выбирать  $p$  и  $q$  равной длины. Рассчитывается произведение:  $n = pq$ . Это есть один из трех компонент ключа, состоящего из чисел  $n, e, d$ .

Затем случайным образом выбирается второй компонент ключа (открытый ключ или ключ зашифрования,  $e$ , такой что  $e$  и  $(p - 1)(q - 1)$  являются взаимно простыми числами; вспомним, что  $(p - 1)(q - 1) = \phi(n)$  – функция Эйлера). Б. Шнайер [5] рекомендует число  $e$  выбирать из ряда: 3, 17,  $2^{16} + 1$ .

Наконец, расширенный алгоритм Евклида используется для вычисления третьего компонента ключа: ключа расшифрования  $d$  такого, что выполняется условие:

$$ed \equiv 1 \pmod{\phi(n)}. \quad (8.4)$$

Другими словами:

$$d^{-1} \equiv e \pmod{\phi(n)}. \quad (8.5)$$

Таким образом, сформирован ключ, состоящий из трех чисел, которые в свою очередь образуют две вышеупомянутые взаимосвязанные части: открытый (публичный) ключ ( $e, n$ ) и тайный ключ ( $d, n$ ; на самом деле, как видим, тайным здесь является лишь первое из пары чисел).

Примеры генерации ключевой информации, как и ее использования, можно найти в [3].

Для зашифрования/расшифрования используется ключ получателя: отправитель шифрует сообщение открытым ключом, а получатель расшифровывает шифртекст своим тайным ключом.

*Зашифрование.* Если шифруется сообщение  $M$ , состоящее из  $r$  блоков:  $m_1, m_2, \dots, m_i, \dots, m_r$ , то шифртекст  $C$  будет состоять из такого же числа ( $r$ ) блоков, представляемых числами:

$$c_i \equiv (m_i)^e \pmod{n}. \quad (8.6)$$

*Расшифрование.* Для расшифрования каждого зашифрованного блока производится вычисление вида:

$$m_i \equiv (c_i)^d \pmod{n}. \quad (8.7)$$

Разработаны несколько версий стандарта рассматриваемого алгоритма. Среди прочего, в этих документах обсуждаются размеры безопасного ключа. Доступна одна из последних версий стандарта RSA: RFC 3447.

Размер ключа в алгоритме RSA связан с размером модуля  $n$ . Два числа  $p$  и  $q$ , произведение которых равно  $n$ , должны иметь приблизительно одинаковую длину, поскольку в этом случае найти сомножители (факторы) сложнее, чем в случае, когда длина чисел значительно различается. Например, если предполагается использовать 768-битный модуль, то каждое число должно иметь длину приблизительно 384 бита. В 1999 г. 512-битный ключ был вскрыт за семь месяцев [5]. Это означает, что 512-битные ключи уже не обеспечивают достаточную криптостойкость. Сейчас в критических системах применяются ключи длиной 1024 и 2048 битов. Статистические представления этих чисел в десятичной системе счисления даны в [9].

### 8.1.3. Алгоритм Эль-Гамала

Предложен Т. Эль-Гамалем (T. El-Gamal) в 1985 г. Он может быть использован для решения трех основных криптографических задач: для зашифрования/расшифрования данных, для формирования цифровой подписи и для согласования общего ключа. Кроме того, возможны модификации алгоритма для схем проверки пароля, доказательства идентичности сообщения и другие варианты.



Как подчеркивалось выше, безопасность алгоритма Эль-Гамала, как и безопасность алгоритма Диффи – Хеллмана, основана на трудности вычисления дискретных логарифмов.

**Алгоритм Эль-Гамаля фактически использует схему Диффи – Хеллмана, чтобы сформировать общий секретный ключ для абонентов, передающих друг другу сообщение, и затем сообщение шифруется путем умножения его на этот ключ.**

И в случае шифрования, и в случае формирования цифровой подписи каждому пользователю необходимо генерировать пару ключей.

Рассматриваемый алгоритм отличается от алгоритма RSA несколькими параметрами и особенностями:

1) генерацией ключевой информации и числом компонент, составляющих ключ;

2) каждому блоку (символу) открытого сообщения в шифртексте на основе алгоритма Эль-Гамаля соответствуют 2 блока (в RSA – один-один);

3) в алгоритме Эль-Гамаля при зашифровании используется число (обозначим его  $k$ ), которое практически никак не связано с ключевой информацией получателя и которое принимает (по определению) различные значения при зашифровании различных блоков сообщения.

**Генерация ключевой информации.** Выбирается простое число  $p$ . Выбирается число  $(g, g < p)$ , являющееся первообразным корнем числа  $p$  – очень важный элемент с точки зрения безопасности алгоритма (см. ниже).

Далее выбирается число  $x$  ( $x < p$ ) и вычисляется последний компонент ключевой информации:

$$y \equiv g^x \pmod{p}. \quad (8.8)$$

Владельцу сформированной ключевой информации, состоящей из 4 чисел, может посыпаться некоторый шифртекст, созданный с использованием открытого ключа получателя:  $p, g, y$ . Расшифрование шифртекста получатель производит своим тайным ключом:  $p, g, x$ .

Как видим, на самом деле тайным является лишь одно число (как и в RSA):  $x$ .

**Определение 2. Первообразный корень** (primary (residual) root) по модулю  $p$  является таким числом, что его степени  $(g^i, 1 \leq i \leq p - 1)$  дают все возможные по модулю  $p$  вычеты (остатки), которые взаимно просты с  $p$ .

**Пример 1.** Следующие остатки по модулю 5 ( $p = 5$ ) от  $2^i$ : 2, 4, 3, 1. Они дают все возможные остатки. Число 2 является первообразным корнем по модулю 5.

**Пример 2.** Следующие остатки по модулю 7 от  $2^i$ : 2, 4, 1, 2, ... (они не дают всех возможных остатков). Число 2 не является первообразным корнем по модулю 7.

**Пример 3.** Для  $p = 17$  и  $g = 3$ : остатки по модулю 17 от  $3^i$ : 3, 9, 10, 13, 5, 15, 11, 16, 14, 8, 7, 4, 12, 2, 6, 1.

Число 3 является первообразным корнем по модулю 17.

Понятно, что для больших значений  $p$  количество всех неповторяющихся остатков ( $p - 1$ ) будет также большим. А поскольку в уравнении (8.8) мы используем модуль  $p$  большого простого числа и находим первообразным корень от  $p$ , который имеет важное свойство: при использовании разных степеней ( $a^i = a^x$ ) решение будет равномерно распределяться от 0 до  $p - 1$ , то нахождение криptoаналитиком нужного  $x$  чрезвычайно затруднено. В этом заключается односторонность функции, задаваемой (8.8). И на этом основывается криптостойкость шифра Эль-Гамаля.

Для схемы вероятностного шифрования само сообщение и ключ не определяют шифротекст однозначно.

**Зшифрование сообщения.** Как ранее, предположим, что сообщение  $M = \{m_i\}$ , где  $m_i$  –  $i$ -й блок сообщения.

Зшифрование отправителем (каждого отдельного блока  $m_i$  исходного сообщения) предусматривает использование, как это особо подчеркивалось выше, некоторого случайного числа  $k$  ( $1 < k < p - 1$ ).

**! В силу использования случайной величины  $k$  шифр Эль-Гамаля называют также шифром многозначной замены, а также схемой вероятностного шифрования.**

Вероятностный характер шифрования является преимуществом для схемы Эль-Гамаля по сравнению, например, с алгоритмом RSA.

Блок шифртекста ( $c_i$ ) состоит из двух чисел –  $a_i$  и  $b_i$ :

$$a_i \equiv g^k \pmod{p}; \quad (8.9)$$

$$b_i \equiv (y^k m_i) \pmod{p}. \quad (8.10)$$

Здесь стал очевидным упомянутый недостаток алгоритма шифрования Эль-Гамаля: удвоение (реально – примерно в 1,5 раза) длины зашифрованного текста по сравнению с начальным текстом.

Случайное число  $k$  должно сразу после вычисления уничтожаться.

**Расшифрование  $c_i$ .** Выполняется по следующей формуле:

$$m_i \equiv (b_i(a_i)^x)^{-1} \pmod{p} \quad (8.11)$$

или

$$m_i \equiv (b_i(a_i)^{p-x-1}) \pmod{p}, \quad (8.12)$$

где  $(a^x)^{-1}$  – обратное значение числа  $a^x$  по модулю  $p$ .

Нетрудно проверить, что  $((a_i)^x)^{-1} \equiv g^{kx} \pmod{p}$ .

Еще раз возвратимся к криптостойкости рассмотренного алгоритма.

Если для зашифрования двух разных блоков ( $m_1$  и  $m_2$ ) некоторого сообщения использовать одинаковые  $k$ , то для соответствующих шифртекстов  $c_1 = (a_1, b_1)$  и  $c_2 = (a_2, b_2)$  выполняется соотношение  $b_1(b_2)^{-1} = m_1(m_2)^{-1}$ . Из этого выражения можно легко вычислить  $m_2$ , если известно  $m_1$ .

При примерно одинаковой размерности ключей рассмотренные алгоритмы обеспечивают примерно одинаковый уровень криптостойкости.

## 8.2. Практическое задание

1. С помощью простого консольного приложения составить табличную или графическую форму зависимости времени вычисления параметра  $y$ , функционально заданного выражением вида:

$$y \equiv a^x \pmod{n},$$

от параметров:  $a$  (десятичные числа от 5 до 35; можно взять 1 или 2 числа),  $x$  (числа, желательно простые, из диапазона от  $10^3$  до  $10^{100}$ ; для примера взять 5–10 чисел, равномерно распределенных в указанном диапазоне),  $n$  (для примера взять числа, в двоичном виде состоящие из 1024 и 2048 битов).

2. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться доступными библиотеками либо программными кодами.

В основе вычислений – кодировочные таблицы Base64 и ASCII.

Приложение должно реализовывать следующие операции:

- зашифрование и расшифрование текстовых документов на основе алгоритмов RSA и Эль-Гамаля;
- определение времени выполнения операций.

Исходный текст для зашифрования – собственные фамилия, имя, отчество. Для численного представления блоков текста можно в том числе пользоваться указанными выше кодировочными таблицами.

Ключевую информацию для обоих алгоритмов можно сгенерировать самостоятельно либо воспользоваться, например, одной из утилит криптографической библиотеки OpenSSL, с помощью которой, в частности, можно сгенерировать ключевую информацию для алгоритма RSA.

3. Используя примерно одинаковый порядок ключевой информации, оценить производительность обоих алгоритмов и относительное изменение объемов криптокстов (по отношению к объемам открытых текстов).

Ниже даны некоторые указания к инсталляции (при желании) библиотеки OpenSSL.

OpenSSL – это система защиты и сертификации данных; SSL (Secure Socket Layer – система безопасных сокетов). Внутри OpenSSL имеются отдельные компоненты (утилиты), отвечающие за то или иное действие. OpenSSL поддерживает в том числе много различных стандартов шифрования. К их числу относится RSA.

Краткую начальную информацию к процедуре инсталляции OpenSSL можно найти по адресу: [https://www.xolphin.com/support/OpenSSL/OpenSSL\\_-\\_Installation\\_under\\_Windows](https://www.xolphin.com/support/OpenSSL/OpenSSL_-_Installation_under_Windows).

Для загрузки OpenSSL можно также воспользоваться источником по адресу: <https://sourceforge.net/projects/openssl/>.

После скачивания выбранной версии OpenSSL запускается *exe*-файл, например, *Win64OpenSSL\_Light-1\_1\_1d.exe* (рис. 8.1).

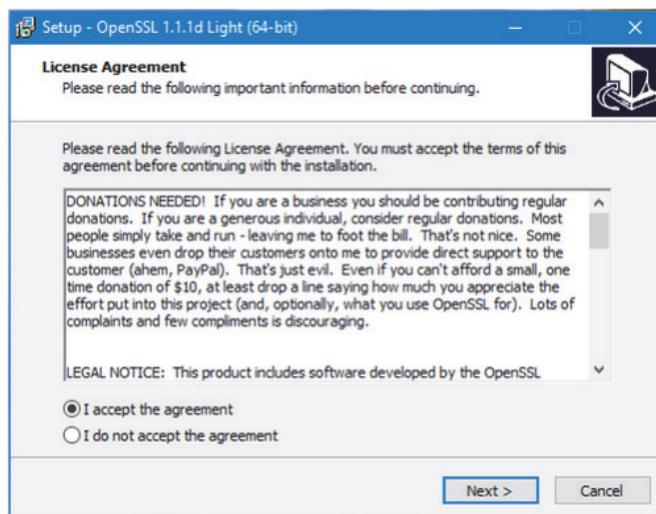


Рис. 8.1. Начальное диалоговое окно инсталляции OpenSSL

Далее выполняем необходимые (достаточно традиционные) требования для установки.

Синтаксис OpenSSL следующий:

`$ openssl <команда> [<опции команды>] [<параметры команды>]`

При вызове `openssl` без команды и параметров будет открыта интерактивная оболочка, из которой можно выполнять все те же команды, что и в неинтерактивном режиме. При вызове `openssl` с несуществующей командой будет выведен перечень команд и поддерживаемых криптоалгоритмов.

Вывод на экран списка доступных команд с неправильным ключом: `OpenSSL > help` или `OpenSSL > help -h`

Для нас в рамках выполнения данной лабораторной работы интерес представляют следующие команды:

`rsa` – обработка ключей RSA;

`speed` – вычисление скорости алгоритмов.

Для создания ключей алгоритма RSA используется команда `genrsa`: `openssl genrsa [-out file] [-des | -des3 | -idea] [-rand file] [bits]`

Команда `genrsa` создает секретный ключ длиной (дается в битах) в формате *PEM* (текстовый формат), зашифровывая его одним из алгоритмов (des, des3, idea). Опция `out` позволяет указать, в какой файл выполняется вывод созданного тайного ключа.

**Пример 4.** Команда `genrsa -out c:\key.txt -des 1024` позволяет сгенерировать тайный ключ длиной 1024 бита, который будет записан в файл `key.txt`.

**Пример 5.** При генерации тайного ключа с записью его содержимого в файл `private.pem` пользователь увидит сообщение, показанное на рис. 8.2.

```
OpenSSL> genrsa -out c:\lab\private.pem 1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
-----+
e is 65537 <0x10001>
-----+
OpenSSL>
```

**Рис. 8.2.** Диалоговое окно OpenSSL при генерации тайного ключа RSA

Для создания открытого ключа на основе созданного секретного ключа используется команда `rsa`, которая имеет следующий синтаксис: `openssl rsa -in filename [-out file] [-des | -3des | -idea] [-check] [-pubout]`

В качестве параметров используются следующие ключи (действы перед ключами обязательны):

- pubout* – указывает на необходимость создания открытого ключа (public key) в файле, указанном в параметре *-out*;
- in* – указывает на файл с секретным ключом;
- pubin* – указание на то, что передается открытый ключ.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. Охарактеризовать алгоритмы RSA и Эль-Гамаля. Для каких целей они могут применяться?

2. На чем основана криптостойкость алгоритмов RSA и Эль-Гамаля?

3. Что такое первообразный корень?

4. Найти первообразные корни (если они существуют) чисел (*p*): 13, 19, 23, 27, 31, 37, 39, 43.

5. Пусть пользователь А хочет передать пользователю В сообщение *M*, которое в некоторой кодировке соответствует числу 17 и зашифровано с помощью алгоритма RSA. Пользователь В имеет следующие ключевые параметры:  $p = 7$ ,  $q = 11$ ,  $d = 47$ . Описать процесс зашифрования сообщения пользователем А.

6. Пользователю системы RSA с ключевыми параметрами  $n = 33$ ,  $d = 3$  передано зашифрованное сообщение *C*, состоящее из блока цифр: 13. Расшифровать это сообщение (взломав систему RSA пользователя).

7. В системе связи, применяющей шифр Эль-Гамаля, пользователь А желает передать сообщение *M* пользователю В. Найти недостающие параметры системы при следующих заданных параметрах:  $p = 19$ ,  $g = 2$ ,  $x = 3$ ,  $k = 5$ ,  $M = 10$ . Описать по шагам зашифрование сообщения и расшифрование шифртекста.

8. Положим, что в системе применяется алгоритм шифрования/расшифрования Эль-Гамаля. Известны некоторые параметры системы:  $p = 167$ ,  $g = 5$ ,  $y = g^{29} \equiv 55 \pmod{p}$ . Используя указанные и недостающие (выбрать самостоятельно) параметры, зашифровать свое имя (в любом языке) в предположении: а) первая буква алфавита соответствует числу 0 и т. д.; б) первая буква алфавита соответствует числу 1. Проанализировать результат.

# Лабораторная работа № 9

## ||| Исследование криптографических хеш-функций

**Цель:** изучение алгоритмов хеширования и приобретение практических навыков их реализации и использования в криптографии.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и алгоритмам реализации операций вычисления односторонних хеш-функций.
2. Освоить методику оценки криптоустойчивости хеш-преобразований на основе «парадокса дня рождения».
3. Разработать приложение для реализации заданного алгоритма хеширования (из семейств MD и SHA).
4. Оценить скорость вычисления кодов хеш-функций.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### 9.1. Теоретические сведения

#### 9.1.1. Определение, свойства, описание и типы хеш-функций

**Определение 1. Хеш-функция** – математическая или иная функция  $h = H(M)$ , которая принимает на входе строку символов  $M$ , называемую также **прообразом**, переменной длины  $n$  и преобразует ее в выходную строку фиксированной (обычно – меньшей) длины  $l$ .

**Определение 2. Хеширование** (или **хэширование**, англ. *hashing*) – это преобразование входного массива данных определенного типа и произвольной длины (практически) в выходную битовую строку фиксированной длины.

Преобразования называются **хеш-функциями**, или **функциями свертки**, а их результаты называют **хешем**, **хеш-кодом**, **хеш-таблицей** или **дайджестом сообщения** (англ. *message digest*).

**Пример 1.** Простейшим примером хеш-преобразования может быть вычисление на основе операции *суммирования по модулю 2*: символы (байты) входной строки  $M$  складываются побитно по модулю 2 и байт-результат есть значение хеш-функции  $h$ . Длина  $l$  значения хеш-функции составит в этом случае 8 битов независимо от размера входного сообщения.

Если, в продолжение примера, представить входное сообщение в виде 16-ричных чисел:  $M = 1f\ 01\ 9a$  ( $n = 24$  бита) и далее выполнить операцию суммирования в соответствии вышеуказанным принципом:

$$\begin{array}{r} 00011111 \\ 00000001 \\ \hline 10011010 \\ 10000100, \end{array}$$

то мы получаем  $h = 10000100$ , или  $h = 84$  (в 16-ричной системе счисления).

Все существующие функции хеширования можно разделить на два больших класса:

- бесключевые хеш-функции, зависящие только от сообщения;
- хеш-функции с секретным ключом, зависящие как от сообщения, так и от секретного ключа.

**Определение 3. Криптографическая хеш-функция** – это специальный класс хеш-функций, который имеет различные свойства, необходимые для решения задач в области криптографии.



#### Основные задачи, решаемые с помощью хеш-функций:

- аутентификация (хранение паролей);
- проверка целостности данных;
- защита файлов;
- обнаружение зловредного ПО;
- криптовалютные технологии.

На рис. 9.1 перечислены некоторые известные хеш-функции, классифицированные в соответствии с используемым внутренним преобразованием.

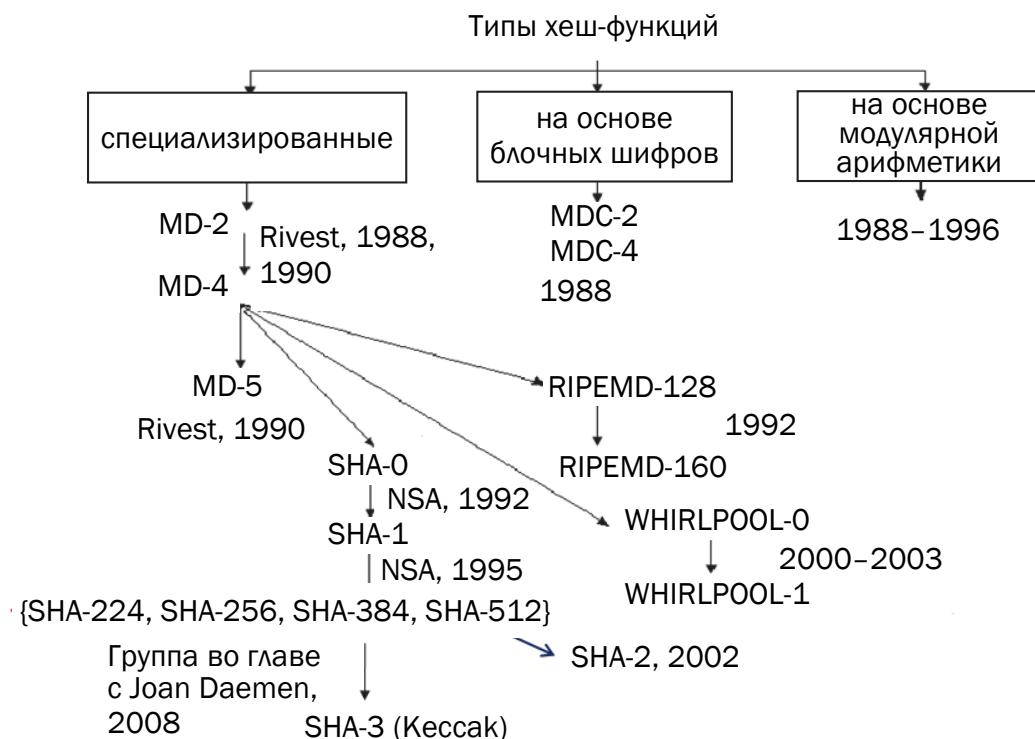
К основным свойствам хеш-функций можно отнести следующие.

*Свойство 1. Детерминированность:* независимо от того, сколько раз вычисляется  $H(M)$ ,  $M - \text{const}$ , при использовании одинакового алгоритма код хеш-преобразования  $h$  всегда должен быть одинаковым.

*Свойство 2. Скорость вычисления хеша  $h$ :* если процесс вычисления  $h$  недостаточно быстрый, система просто не будет эффективной.

*Свойство 3. Сложность обратного вычисления:* для известного  $H(M)$  невозможно (практически) определить  $M$ . Это важнейшее свойство хеш-функции для криптографических применений – *свойство односторонности преобразования*.

Это означает, что по хеш-коду должно быть практически невозможным восстановление входной строки  $M$ .



**Рис. 9.1.** Основные классы хеш-функций в соответствии с используемым внутренним преобразованием

Большинству даже простых пользователей известно, что наиболее распространенная область применения хеширования – хранение паролей. К примеру, если пользователь забыл пароль и пытается воспользоваться доступными функциями-сервисами восстановления пароля, то такой сервис может выдать пароль, как правило, в двух случаях:

- информационная система, в которую входит пользователь с помощью пароля, на самом деле не хеширует этот пароль;
- система восстановления пароля использует некоторую базу данных, содержащую простые, наиболее часто используемые

пароли, например 123456 или qwerty (примером такой системы восстановления пароля является *Online Reverse Hash Lookup*).

**Пример 2.** При длине хеш-кода  $l = 128$  битов в среднем нужно проверить – по методу «грубой силы» –  $2^{128}/2 = 2^{127}$  вариантов входного сообщения для получения фиксированного  $H(M)$ .

В плане односторонности хешей на основе блочных шифров отметим одно обстоятельство. Блочный шифр необратим по ключу шифрования, и если в качестве такого ключа на текущем шаге преобразования используется выход предыдущего шага, а в качестве шифруемого сообщения – очередной блок сообщения (или наоборот), то можно получить хеш-функцию с хорошими криптографическими характеристиками с точки зрения односторонности.

Такой подход использовался, например, в российском стандарте хеширования – ГОСТ Р 34.11–94.

Основным недостатком хеш-функций на основе блочных шифров является сравнительно невысокая производительность.

**Свойство 4.** Даже минимальные изменения в хешируемых данных ( $M \neq M'$ ) должны изменять хеш:  $H(M) \neq H(M')$ .

**Пример 3.** Используются алгоритмы хеширования MD5 и SHA1.

Положим,  $M = \text{«Hash+login2020»}$  и  $M' = \text{«hash+login2020»}$ .

Для MD5 получились следующие хеши (в 16-ричном виде):

$H(M) = c71b846d449901adb3b8308421ef203d$ ,

$H(M') = d228d48d152d929c8ff667dc1a2d663a$ ,

соответственно – для SHA1:

$H(M) = a39ec24cf6a4ab6d15f51c39791211af5743963f$ ,

$H(M') = a31ec6b7710c0e7d4e0c23b261eab9a0ac37177c$ .

**Определение 4. Коллизией хеш-функции  $H$**  называют ситуацию, при которой различным входам (в общем случае –  $x$  и  $y$  или  $M \neq M'$ ) соответствует одинаковый хеш-код:  $H(x) = H(y)$  или  $H(M) = H(M')$ .

**Свойство 5. Коллизионная устойчивость** (стойкость).

Зная  $M$ , трудно найти такое  $M'$  ( $M \neq M'$ ), для которого  $H(M) = H(M')$ .

Если последнее равенство выполняется, то говорят о **коллизии 1-го рода**.

Если **случайным образом** выбраны два сообщения ( $M$  и  $M'$ ), для которых  $H(M) = H(M')$ , говорят о **коллизии 2-го рода**.



**Мерой криптостойкости хеш-функции считается вычислительная сложность нахождения коллизии.**

Для хеш-функций одним из основных средств поиска коллизий является метод, основанный на известной статистической задаче – «парадоксе дня рождения».

В более общем случае: для того чтобы хеш-функция  $H(M)$  считалась криптографически стойкой, она должна удовлетворять трем основным требованиям: необратимостью вычислений (свойство 3), устойчивостью к коллизиям первого рода и устойчивостью к коллизиям второго рода (свойство 5).

### 9.1.2. Парадокс «дней рождения» и его использование в криптографических приложениях

Основной постулат **парадокса «дней рождения»** гласит: в группе минимум из 23 человек с вероятностью более 0,5 день рождения одинаков хотя бы у двух членов группы. Парадоксом является высокая (как кажется на первый взгляд) вероятность наступления указанного события. При этом предполагается, что:

- в этой группе нет близнецов;
- люди рождаются независимо друг от друга, т. е. дата (день) рождения любого человека не влияет на дату рождения другого;
- люди рождаются равномерно и случайно, т. е. люди с равной вероятностью могут рождаться в любой день года; с формальной точки зрения это означает, что вероятность  $p_1$  рождения отдельно выбранного члена группы (как и любого человека) в любой выбранный день равна  $p_1 = 1 / 365$  (хотя известно, что в реальности рождение людей не совсем соответствует такому предположению).

Теперь посмотрим на ситуацию с иной позиции.

Пусть  $A_n$  будет соответствовать ситуации, при которой среди  $n$  членов группы все имеют различные дни рождения, т. е. дни рождения не совпадают. Вероятность такого события обозначим  $P(A_n)$ .

Если предположить, что группа состоит из 2 человек ( $n = 2$ ), то вероятность несовпадения их дней рождения  $P(A_2) = 364 / 365 \approx 0,997$ . С вероятностью  $P(A_3) = (364 / 365) \cdot (363 / 365) \approx 0,991$  в группе из 3 человек будут разные дни рождения.

Вероятность  $P(A_3)$  можно также представить следующим образом:

$$P(A_3) = P(A_2) \cdot ((362 - 2) / 365).$$

Следуя далее принятой логике рассуждений, можно также записать

$$P(A_4) = P(A_3) \cdot ((362 - 3) / 365)$$

и, наконец,

$$P(A_{23}) = P(A_{22}) \cdot ((362 - 22) / 365). \quad (9.1)$$

Выполнив поступательно все вычисления, получим  $P(A_{23}) = 0,493$ . Тогда вероятность совпадения дня рождения для 2 членов группы  $P_c(A_{23}) = 1 - P(A_{23}) = 1 - 0,493 = 0,507$ . Мы получили математическое подтверждение «парадоксу», сформулированному в начале подраздела.

Последний численный вывод мы получили с учетом того, что

$$P_c(A_n) = 1 - P(A_n). \quad (9.2)$$

Теперь несколько обобщим наши рассуждения, предположив, что существуют  $m$  (вместо 365) возможных дней рождения, а группа состоит из  $n$  человек. Тогда выражение (9.1) можно переписать в следующем виде:

$$P(A_n) = P(A_{n-1}) \cdot ((m - (n - 1)) / m). \quad (9.3)$$

Подставляя в правую часть тождества (9.3) все необходимые сомножители, можно записать:

$$P(A_n) = \prod_{i=1}^{n-1} (1 - (i / m)). \quad (9.4)$$

Последнее соотношение можно аппроксимировать, положив  $(1 - x) = \exp(-x)$ :

$$P(A_n) = \prod_{i=1}^{n-1} \exp(-i / m) \approx \exp(-n^2 / 2m). \quad (9.5)$$

Теперь возвратимся к криптографической функции хеширования. Ее также можно определить следующим образом.

**Определение 5. Хеш-функция** – это функция, выполняющая отображение из множества  $M$  в число, находящееся в интервале  $[0, m - 1]$ :  $h: M \rightarrow [0, m - 1]$ .

Мы ранее отмечали, что стойкость хеш-преобразования к коллизии означает, что трудно найти такие  $M_i$  и  $M_j$  ( $M_i, M_j \in M$ ), при которых  $h(M_i) = h(M_j)$ ,  $i \neq j$ ,  $1 \leq i, j \leq n$ .

Для выполнения анализа атаки на основе парадокса «дней рождения» будем использовать те же принципы, которые мы применяли для вероятностной оценки дней рождения.

В атаке «дней рождения»  $m$  соответствует количеству календарных дней в году, а  $M$  – множеству людей, составляющих группу. Люди «хешируются» в их дни рождения, которые могут быть одним из значений  $m$ .

Допустим (переходя в информационную область), нам нужно найти коллизию с вероятностью 0,99 ( $P_c(A_n) = 0,99$ ). Мы хотим определить наименьшее  $n$ , при котором хеш двух значений из  $A_n$  будет «одним днем рождения», что в интересующей нас плоскости означает, что два входных набора данных ( $M_i, M_j \in M$ ) хешируются в одинаковое значение:  $h(M_i) = h(M_j)$ . Допустим далее, что все входные данные хешируются в  $m$  выходных хеш-кодов.

При атаке «дней рождения» злоумышленник будет случайным образом подбирать  $M_i$  и  $M_j$  и сохранять пары их хешей, пока не найдет двух значений, при которых  $h(M_i) = h(M_j)$ . Нам нужно определить, сколько раз атакующему нужно повторить эту операцию, пока не будет обнаружена коллизия.

Иначе говоря, стоит задача отыскания наименьшего  $n$ , при котором хеши двух значений  $m$  будут «одним днем рождения».

Итак, наш случай ограничивается частной задачей поиска условия, при котором выполняется тождество  $P_c(A_n) = 0,99$  или, в соответствии с (9.2),  $P(A_n) = 0,01$ . Воспользуемся выражением (9.5):

$$P(A_n) \approx \exp(-n^2 / 2m) = 0,01. \quad (9.6)$$

Последнее можно записать в виде равенства

$$n^2 / 2m = \ln 100,$$

откуда находим

$$n = (2m \ln 100)^{1/2}. \quad (9.7)$$

Последнее, кстати, согласуется с нашим предыдущим анализом при  $m = 365$ , потому что  $(2 \cdot 365 \ln 100)^{1/2}$  примерно равно 23.

Если хеш имеет длину  $l$  бит, то  $m = 2^l$ . И в соответствии с формулой (9.7) для поиска коллизии с вероятностью 0,99 нужно выполнить  $2^{l/2}$  операций хеширования различных входных сообщений.

В таблице приведены вероятностные оценки появления коллизии для хеш-функций различной длины (в приведенной таблице параметр  $N$  соответствует принятому нами обозначению  $l$ ).

**Вероятностные оценки появления коллизии  
для хеш-кодов различной длины /**

$N$	Количество выходных цепочек ( $2^N$ )	Вероятность хотя бы одной коллизии ( $p$ )								
		$10^{-18}$	$10^{-15}$	$10^{-12}$	$10^{-9}$	$10^{-6}$	$0,1\%$	$1\%$	$25\%$	$50\%$
32	$4,23 \cdot 10^9$	2	2	2,9	93	$2,9 \cdot 10^3$	$9,3 \cdot 10^3$	$5 \cdot 10^4$	$7,7 \cdot 10^4$	$1,1 \cdot 10^5$
64	$1,8 \cdot 10^{19}$	6,1	$1,9 \cdot 10^2$	$6,1 \cdot 10^2$	$1,9 \cdot 10^5$	$6,1 \cdot 10^6$	$1,9 \cdot 10^8$	$6,1 \cdot 10^8$	$3,3 \cdot 10^9$	$5,1 \cdot 10^9$
128	$3,4 \cdot 10^{38}$	$2,6 \cdot 10^{10}$	$8,2 \cdot 10^{11}$	$2,6 \cdot 10^{13}$	$8,2 \cdot 10^{14}$	$2,6 \cdot 10^{16}$	$8,3 \cdot 10^{17}$	$2,6 \cdot 10^{18}$	$1,4 \cdot 10^{19}$	$2,2 \cdot 10^{19}$
256	$1,2 \cdot 10^{77}$	$4,8 \cdot 10^{29}$	$1,5 \cdot 10^{31}$	$4,8 \cdot 10^{32}$	$1,5 \cdot 10^{34}$	$4,8 \cdot 10^{35}$	$1,5 \cdot 10^{37}$	$4,8 \cdot 10^{37}$	$2,6 \cdot 10^{38}$	$5,7 \cdot 10^{38}$
384	$3,9 \cdot 10^{115}$	$8,9 \cdot 10^{48}$	$2,8 \cdot 10^{50}$	$8,9 \cdot 10^{51}$	$2,8 \cdot 10^{53}$	$8,9 \cdot 10^{54}$	$2,8 \cdot 10^{56}$	$8,9 \cdot 10^{56}$	$4,8 \cdot 10^{57}$	$7,4 \cdot 10^{57}$
512	$1,3 \cdot 10^{154}$	$1,6 \cdot 10^{68}$	$5,2 \cdot 10^{69}$	$1,6 \cdot 10^{71}$	$5,2 \cdot 10^{72}$	$1,6 \cdot 10^{74}$	$5,2 \cdot 10^{75}$	$1,6 \cdot 10^{76}$	$8,8 \cdot 10^{76}$	$1,4 \cdot 10^{77}$
										$1,9 \cdot 10^{77}$

Устойчивость хеш-функции к коллизиям имеет первостепенное значение в технологиях электронной цифровой подписи и криптовалютных технологиях. В настоящее время длина хеша  $l = 64$  не относится к числу криптостойких.

### **9.1.3. Структурные и функциональные особенности некоторых хеш-функций**

Рис. 9.1 (см. с. 123) дает начальные представления об основных хеш-преобразованиях. В [41–48] можно ознакомиться с подробным описанием всех упоминаемых нами алгоритмов (от их создателей). Много полезной информации, а также программные коды некоторых алгоритмов хеширования на С можно найти в [5].

Алгоритмы семейства MD-x (2/4/5/6) являются творениями Р. Ривеста; MD – Message Digest. Алгоритм MD6, в отличие от предыдущих версий алгоритма этого семейства, не стандартизован.

Алгоритмы семейства SHA (SHA – Secure Hash Algorithm) являются в настоящее время широко распространенными. По существу, во многих случаях завершился переход от SHA-1 к стандартам версии SHA-2. SHA-2 – собирательное название алгоритмов SHA-224, SHA-256, SHA-384 и SHA-512. SHA-224 и SHA-384 являются, по сути, аналогами SHA-256 и SHA-512 соответственно.

Известен также алгоритм хеширования, долгое время использовавшийся в качестве национального стандарта России (ГОСТ 34.11–94).

С 5 августа 2015 г. утвержден и опубликован в качестве действующего стандарта (FIPS 202) алгоритм SHA-3, одной из отличительных особенностей которого является использование конструкции «криптографической губки». В этой конструкции реализован итеративный подход для создания функции с произвольной длиной на входе и произвольной длиной на выходе на основе определенного преобразования [49]. На основе технологии «губки» построен ныне действующий в Беларуси стандарт хеширования [50].

Алгоритмы семейства MD входные сообщения максимальной длины  $2^{64} - 1$  битов (в общем случае –  $L$  битов) преобразуют в хеш длиной  $l = 128$  битов. Исключением является последняя (6-я) из версий алгоритма, где длина результирующего хеша может изменяться от 1 до 512 бит.

Максимальный объем хешируемых сообщений для алгоритмов SHA-1, SHA-256, SHA-224 такой же, как и для алгоритмов MD.

Однако длина хешей разная: в SHA-1 – 160 битов; в алгоритмах, относящихся к семейству SHA-2, – соответствует числу, дополняющему через дефис название алгоритма. Максимальная же длина входных сообщений в алгоритмах SHA-512, SHA-384, SHA-512/256, SHA-512/224 составляет  $2^{128}-1$  битов.

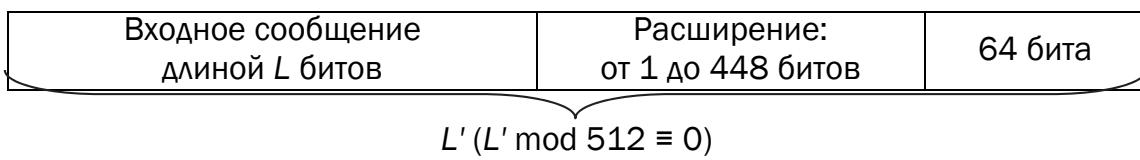
Базовые алгоритмы обоих рассматриваемых семейств (MD и SHA) условно можно разделить на 5 стадий:

- расширение входного сообщения;
- разбивка расширенного сообщения на блоки;
- инициализация начальных констант;
- обработка сообщения поблочно (основная процедура алгоритма хеширования);
- вывод результата.

Процедура расширения хешируемого сообщения достаточно подробно описана в [3] на примере алгоритма MD-4 (см. п. 10.3.1).

Входное сообщение «дополняется» (расширяется) так, чтобы его длина (в битах) была конгруэнтной к 448 по модулю 512. Это значит, что сообщение начальной длиной  $L$  битов расширяется так, что остаются незаполненными всего лишь 64 бита, чтобы итоговая длина  $L'$  была кратной 512. В указанные 64 бита записывается двоичная длина.

Расширение происходит всегда, даже если длина сообщения уже соответствует 448, по модулю 512. Эта операция выполняется следующим образом: один бит «1» добавляется к сообщению, а затем добавляются биты «0», так что длина в битах дополненного сообщения стала конгруэнтной 448 по модулю 512. Добавляется не менее одного бита, но не более 448 битов. Схематично состав и размеры дополненного входного сообщения показаны на рис. 9.2.



**Рис. 9.2.** Состав и размеры дополненного хешируемого сообщения

Основой рассматриваемых базовых алгоритмов является модуль, состоящий из циклических преобразований каждого 512-битного блока, который делится на подблоки длиной 32 либо 64 бита (в алгоритмах SHA-512, SHA-384, SHA-512/256, SHA-512/224).

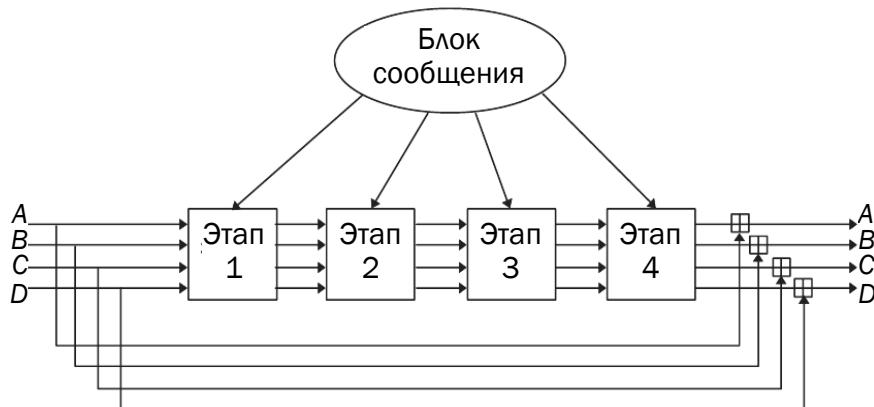
При длине подблока в 16 битов каждый 512-битный блок должен состоять из 32 подблоков.

Рассмотрим пример.

**Пример 3.** Пусть исходное сообщение, или первообразная, будет словом «CRYPTO» ( $M = \text{«CRYPTO»}$ ), или в кодах ASCII – это будут десятичные (67-82-89-80-84-79) и соответствующие двоичные (01000011-01010010-01011001-01010000-01010100-01001111; здесь для отделения чисел используются дефисы) числа. Как видим, длина  $L$  хешируемого сообщения равна 48 битам. Эту длину мы должны расширить до 448 битов, добавив одну «1» и 399 «0». В последнюю часть из 64 битов полученного 512-битного модуля ( $L' = 512$ ) мы запишем справа двоичное представление числа  $L = 48$ : 110000. В остальные 58 разрядов (из 64) мы впишем «0». После этого полученный расширенный блок делим на 16 32-разрядных подблоков:

Как было отмечено выше, основная операция заключается в циклической (пораундовой или поэтапной) обработке 512-битных блоков. Таких циклов может быть 3 (как в MD-4), или 4 (как в MD-4), или более. В каждом цикле используется своя нелинейная функция (обычно обозначаемая по порядку  $F$ ,  $G$ ,  $H$ , ...), зависящая от текущего состояния 4 (в MD), 5 (в SHA-1), 8 (SHA-256) и т. д. переменных, начальные состояния которых известны, а текущие – зависят от выполненных операций над хешируемым сообщением [1, 4].

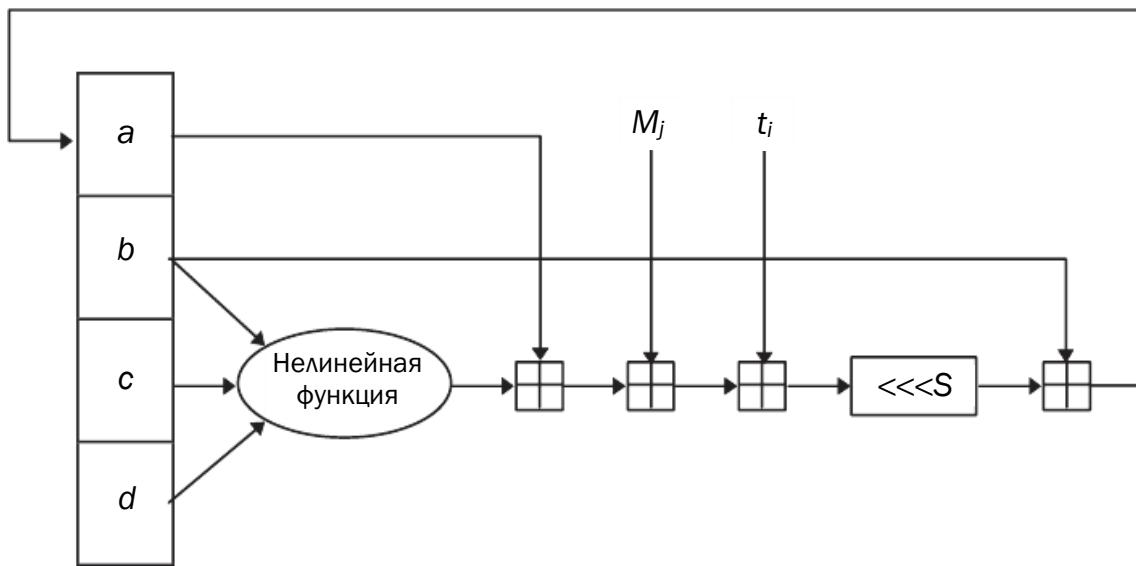
На рис. 9.3 приведена укрупненная структурная схема алгоритма MD-5, на рис. 9.4 – структурная схема одной операции (над одним 32-разрядным подблоком).



**Рис. 9.3.** Укрупненная структурная схема алгоритма MD-5

Здесь знак «+» в квадратной фигуре соответствует операции сложения по модулю  $2^{32}$ . Вспомним (см. лабораторную работу № 5):

$$A + B \pmod{2^{32}} \equiv \begin{cases} A + B, & \text{если } A + B < 2^{32} \\ A + B - 2^{32}, & \text{если } A + B \geq 2^{32} \end{cases}.$$



**Рис. 9.4.** Структурная схема одной операции алгоритма MD-5

Главный модуль (рис. 9.3) состоит из четырех похожих этапов (у MD-4 было только три этапа). На каждом этапе 16 раз используются различные операции. Каждая операция представляет собой нелинейную функцию над тремя из  $a$ ,  $b$ ,  $c$  и  $d$ . Затем она добавляет

этот результат к четвертой переменной, подблоку текста  $M_j$  и константе  $t_i$ . Далее результат циклически сдвигается вправо на переменное число  $s$  битов и добавляется результат к одной из переменных  $a$ ,  $b$ ,  $c$  и  $d$ . Наконец, результат заменяет одну из этих переменных (рис. 9.4).

Результатом хеширования  $h$  является конкатенация последних значений указанных переменных, т. е.  $32 \cdot 4 = 128$  битов.

На рис. 9.5 показана схема выполнения одной операции в алгоритме SHA-1. Цикл состоит из четырех этапов по 20 операций в каждом (в MD5 – 4 этапа по 16 операций в каждом). Каждая операция представляет собой нелинейную функцию над тремя из пяти:  $a$ ,  $b$ ,  $c$ ,  $d$ ,  $e$ . Сдвиг и сложение – аналогично MD5.

В алгоритме используются следующие четыре константы:

$K_t = 0x5a827999$ , при  $t = 0, \dots, 19$ ,

$K_t = 0x6ed9eba1$ , при  $t = 20, \dots, 39$ ,

$K_t = 0x8flbbcdcc$ , при  $t = 40, \dots, 59$ ,

$K_t = 0xca62c1d6$ , при  $t = 60, \dots, 79$ .

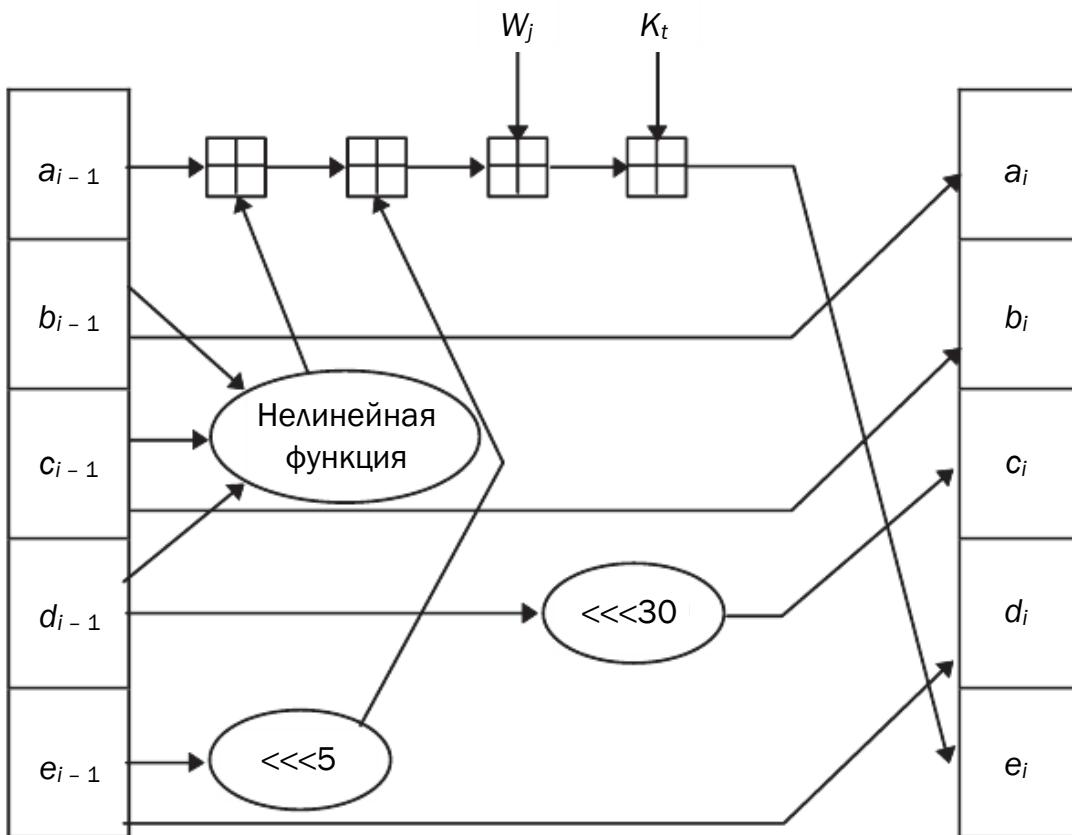


Рис. 9.5. Схема выполнения одной операции в алгоритме SHA-1 [5]

Блок сообщения трансформируется из 16 32-битных слов (от  $M_0$  по  $M_{15}$ ) в 80 32-битных слов ( $W_0, \dots, W_{79}$ ) с помощью следующего алгоритма:

$$W_t = M_t, \text{ при } t = 0, \dots, 15,$$

$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \ll 1$ , при  $t = 16, \dots, 79$ ; здесь  $W_t$  вычисляется как сумма по модулю ( $\oplus$ ).

После обработки всех 512-битных блоков выходом является 160-битный дайджест сообщения в виде конкатенации последних значений переменных  $a, b, c, d, e$ .

Как мы заметили, в алгоритмах MD-5 SHA-1 результат текущего действия прибавляется к результату предыдущего. Это направлено на усиление лавинного эффекта. Этой же цели служит то обстоятельство, что значения циклического сдвига влево на каждом этапе были приближенно оптимизированы: четыре сдвига, используемые на каждом этапе, отличаются от значений, используемых на других этапах.

Как отмечает Б. Шнайер [5], SHA – это MD-4 с добавлением расширяющего преобразования, дополнительного этапа и с улучшенным лавинным эффектом. MD-5 – это MD-4 с улучшенным битовым хешированием, дополнительным этапом и улучшенным лавинным эффектом.

## 9.2. Практическое задание

1. Разработать оконное приложение, реализующее один из алгоритмов хеширования из указанного преподавателем семейства (MD или SHA; или иного). При этом можно воспользоваться доступными готовыми библиотеками. Язык программирования – на свой выбор.

Приложение должно обрабатывать входные сообщения, длина которых определяется спецификацией на реализуемый алгоритм.

2. Оценить быстродействие выбранного алгоритма хеширования.

3. Результаты оформить в виде отчета по установленным правилам.

## ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Дать определение хеш-функции.

2. Что такое «однонаправленность» хеш-функций и какова роль этого свойства хеш-функций в криптографии?

3. Что такое «коллизия»? Типы коллизий хеш-функций.
4. Сформулировать в общем виде парадокс «дней рождений».
5. Как парадокс «дней рождений» используется в криптографии?
6. Сколько попыток нужно сделать, чтобы с вероятностью более 0,5 (0,7; 0,8; 0,9) обнаружить коллизию при длине хеша ( $l$ ) 64 (128; 256; 512) битов?
7. Дать общую характеристику алгоритмам хеширования семейств MD и SHA. Из каких основных стадий состоит алгоритм хеширования сообщения?
8. Рассчитать общую длину ( $L'$ ) хешируемого сообщения после предварительной стадии на основе алгоритма MD, если объем ( $L$ ) исходного сообщения составлял: 0; 484; 512; 1000; 2000; 16 000 битов. Какова в каждом случае будет длина хеша?
9. Входное сообщение (прообраз) состоит:
  - а) из вашего имени;
  - б) из ваших фамилии\_имени\_отчества (алфавит – на свой выбор).
- Используя представление сообщения в кодах ASCII, представить в табличной форме (как выше в примере 5) содержание каждого 32-битного подблока расширенного входного сообщения.
10. Представить и охарактеризовать структурную схему одного раунда алгоритмов хеширования на основе MD4; MD5; SHA-1.
11. На чем основан «лавинный эффект» в алгоритмах хеширования? В чем состоит цель его реализации?
12. В чем состоят основные структурные и функциональные особенности алгоритма хеширования SHA-3?
13. Охарактеризовать структурные, функциональные особенности и криптостойкость белорусского государственного стандарта хеширования (СТБ 34.101.77–2016).

## **Лабораторная работа № 10**

# **ИССЛЕДОВАНИЕ АЛГОРИТМОВ ГЕНЕРАЦИИ И ВЕРИФИКАЦИИ ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ**

**Цель:** изучение алгоритмов генерации и верификации электронной цифровой подписи и приобретение практических навыков их реализации.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и алгоритмам реализации операций генерации и верификации электронной цифровой подписи (ЭЦП).
2. Получить навыки практической реализации методов генерации и верификации ЭЦП на основе хеширования подписываемых сообщений и алгоритмов RSA, Эль-Гамаля и Шнорра, а также DSA.
3. Разработать приложение для реализации заданных алгоритмов генерации и верификации ЭЦП.
4. Оценить скорость генерации и верификации ЭЦП.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### **10.1. Теоретические сведения**

#### **10.1.1. Определение, назначение, основные функции и типы ЭЦП**

Электронная цифровая подпись (ЭЦП) является важным элементом современных информационных систем, использующих методы и технологии криптографического преобразования информации.

Остановимся на важнейших свойствах и иных информационных и фактологических характеристиках ЭЦП. Более подробные сведения из предметной области можно найти в [2, 4, 29, 50].

Понятие «электронная цифровая подпись» было введено в 1976 г. У. Диффи и М. Хеллманом.

После создания RSA разработаны алгоритмы цифровой подписи И. Рабина и Р. Меркле. В 1984 г. Ш. Гольдвассер, С. Микали и Р. Ривест сформулировали требования безопасности к алгоритмам ЭЦП, описали атаки на ЭЦП.

Государственный стандарт Республики Беларусь [51] определяет понятие ЭЦП в следующем виде.

**Определение 1. Электронная цифровая подпись** – контрольная характеристика сообщения, которая *вырабатывается с использованием личного ключа*, проверяется с использованием открытого ключа, служит для контроля целостности и подлинности сообщения и обеспечивает невозможность отказа от авторства.

! Таким образом, ЭЦП выполняет те же функции, что и собственноручная (поставленная «от руки») подпись:

- **автентификация лица, подписавшего сообщение;**
- **контроль целостности подписанного сообщения;**
- **защита сообщения от подделок;**
- **доказательство авторства лица, подписавшего сообщение, если это лицо отрицает свое авторство.**

! Важнейшие отличительные особенности ЭЦП:

- ЭЦП представляет собой бинарную последовательность (в отличие от графического образа, каковым является подпись от руки);
- указанная бинарная последовательность зависит от содержания подписываемого сообщения.

Как следует из определения 1, основным компонентом в технологии ЭЦП является ключ. Принадлежность ключа, в предположении, что он известен только законным пользователям, позволяет решать все «возложенные на ЭЦП», сформированную на основе этого ключа, задачи. В соответствии с этим обстоятельством перечисленные выше функции ЭЦП могут быть реализованы на основе классических методов зашифрования/расшифрования (см. гл. 10 в [3]):

- на основе симметричных систем (с тайным ключом);
- на основе симметричных систем и посредника;
- на основе асимметричных систем (с открытым ключом).

Первый из перечисленных методом ничем не отличается, например, от DES.

Во втором случае создаются две симметричные системы: между отправителем и посредником и между посредником и получателем. Причем посредник выдает двум сторонам различный тайный (для иных субъектов системы) ключ.

В последнем случае сообщение, отправляемое получателю, шифруется тайным ключом отправителя. Отправитель же верифицирует подпись (в данном случае – устанавливает авторство, используя для расшифрования публичный ключ отправителя, и получает гарантию в защищенности переданного сообщения от подделок, если после расшифрования формат и содержание документа имеют логическую стройность) с помощью открытого ключа отправителя.

Таким образом, в этом случае, как и в первых двух случаях, ЭЦП, как отдельный, самостоятельный, присоединенный к исходному документу элемент получаемого сообщения, отсутствует. Кроме того, в отличие от классической асимметричной криптографии, где используется ключевая информация получателя, в нашем случае используется ключевая информация отправителя: открытый ключ – для зашифрования, тайный – для расшифрования.

С учетом изложенного можем сформулировать определение ЭЦП в несколько ином виде.

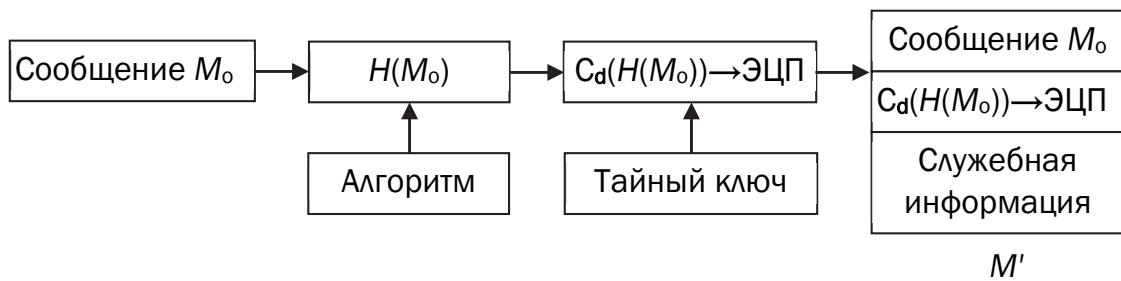
**Определение 2. Электронная цифровая подпись** – бинарная (или в ином виде) последовательность символов, являющаяся реквизитом электронного документа, зависящая от содержания этого документа и предназначенная для подтверждения целостности и подлинности электронного документа.

### 10.1.2. ЭЦП на основе хешей подписываемых сообщений

Классическая технология использования ЭЦП предусматривает подписание не самого сообщения (обозначим его здесь  $M_o$ ), а его хеша,  $H(M_o)$ . Это сокращает время генерации/верификации подписи и снижает вероятность появления случайных ошибок в итоговом документе.

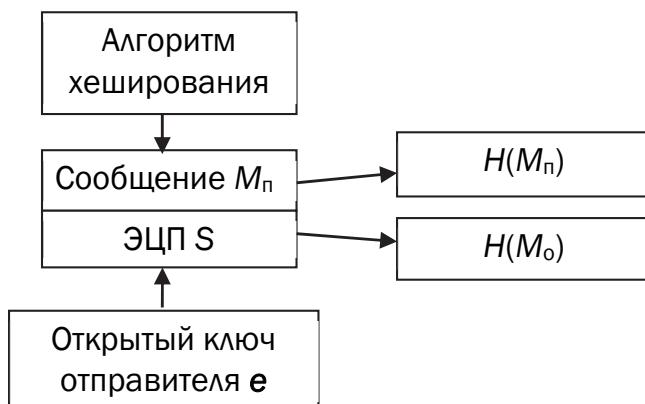
Основу рассматриваемых протоколов составляют методы асимметричной криптографии и эллиптических кривых.

Общая структура подписанного электронного документа –  $M_o - M'$  – представляет собой, как правило, конкатенацию этого документа и ЭЦП  $S$ . Кроме этих двух элементов, интегральный документ может содержать некоторую служебную информацию (дата, время отправки или различные данные об отправителе), как это схематично показано на рис. 10.1.



**Рис. 10.1.** Пояснение к процедуре формирования ЭЦП и структуре подписанного документа

Важное свойство цифровой подписи заключается в том, что ее может проверить (верифицировать) каждый, кто имеет доступ к *открытым ключу* ее автора. На рис. 10.2 показан в общем виде порядок процесса верификации (без учета использования служебной информации). Заметим, что в общем случае версии исходного документа ( $M_o$ ) и полученного ( $M_n$ ) могут отличаться.



**Рис. 10.2.** Пояснение к процедуре верификации ЭЦП

Если в результате устанавливается равенство хешей:  $H(M_n) = H(M_o)$ , то принимается решение о подлинности подписи и целостности документа  $M_n$ , т. е. это также означает, что  $M_n = M_o$ .

Из приведенных на рис. 10.1 и рис. 10.2 последовательных преобразований можно сделать следующие общие выводы:

– при генерации ЭЦП (по классической схеме) для сообщения  $M$  отправитель последовательно выполняет следующие действия:

- вычисляет хеш (хеш-образ) сообщения  $M$ :  $H(M)$ ;
- вычисляет содержание ЭЦП (с собственно ЭЦП  $S$ ) по хешу  $H(M)$  с использованием своего закрытого ключа  $d$ :  $S = C_d(H(M))$ ;

- присоединяет (конкатенирует) ЭЦП к сообщению  $M$  и некоторой служебной информации, создавая таким образом итоговое сообщение  $M'$ ;
- посыпает сообщение  $M'$  получателю;
  - получив сообщение  $M'$ , другая сторона последовательно выполняет следующие действия:
    - отделяет цифровую подпись  $S$  от сообщения  $M$  (для общего случая применим одинаковые символьные обозначения);
    - применяет к сообщению  $M$  операцию хеширования, используя ту же функцию, что и отправитель, и получает *хеш-образ полученного сообщения*;
    - используя открытый ключ отправителя, расшифровывает  $S$ , т. е. извлекает из ЭЦП *хеш-образ отправленного сообщения*;
    - проверяет соответствие (равенство) обоих хеш-образов, и если они совпадают, то отправитель действительно является тем, за кого себя выдает, а сообщение при передаче не подверглось искажению.

При этом стойкость ЭЦП к подделыванию (криптостойкость) определяется теми же факторами, что и криптостойкость алгоритмов зашифрования/расшифрования сообщений: чтобы применение ЭЦП имело смысл, необходимо, чтобы вычисление легитимной подписи без знания закрытого ключа было вычислительно сложным процессом. Решение такой задачи в асимметричных алгоритмах реализации ЭЦП опирается на известные нам вычислительные задачи:

- факторизации, т. е. разложения числа на простые множители;
- дискретного логарифмирования.

На основе первой задачи строится алгоритм RSA, на основе второй – алгоритмы, например, Эль-Гамаля, DSA, Шнорра. Эти алгоритмы достаточно подробно рассмотрены в [3], главе 11. Здесь остановимся на кратком описании математических основ алгоритмов.

### **10.1.2.1. ЭЦП на основе RSA**

Здесь можно рассматривать две ситуации:

- сообщение  $M_0$  подписывается и передается в открытом (незашифрованном) виде;
- сообщение  $M_0$  подписывается и передается в зашифрованном виде.

Первый случай соответствует схеме и операциям, представленным на рис. 10.1 и рис. 10.2. При этом подпись  $S$  вычисляется на основе известного из лабораторной работы № 8 соотношения (8.5):

$$S \equiv (H(M_o))^{d_o} \pmod{n_o}, \quad (10.1)$$

при указанном выше реверсе в отношении ключевой информации; в (10.1)  $d_o$  и  $n_o$  – элементы тайного ключа отправителя. Передаваемое сообщение  $M' = M_o \| S$ .

Соответственно, операция расшифрования на приемной стороне (получатель анализирует  $M_n \| S$ ) будет производиться в соответствии с формулой (8.6) с известной модификацией ключей:

$$H(M_o) \equiv (S)^{e_o} \pmod{n_o}. \quad (10.2)$$

Далее вычисляется  $H(M_n)$ . Если  $H(M_o) = H(M_n)$ , подпись верифицирована.

Если подписываемое сообщение  $M(M')$  также должно передаваться в зашифрованном виде, то обычно  $M'$  шифруется на стороне отправителя стандартным образом: с помощью открытого ключа получателя ( $e_n$  и  $n_n$ ), который перед основным процессом верификации подписи расшифровывает послание своим тайным ключом:  $d_n$  и  $n_n$ . Далее осуществляются вычисления и анализ, как и в первом случае.

### **10.1.2.2. ЭЦП на основе DSA**

Алгоритм DSA (Digital Signature Algorithm – алгоритм цифровой подписи), или DSS (Digital Signature Standard – стандарт цифровой подписи), является одним из известных, нередко и сейчас применяемых. В алгоритме используются следующие параметры:  $p$  – простое число длиной от 64 до 1024 битов (число должно быть кратно 64);  $q$  – 160-битный простой множитель ( $p - 1$ ). Далее вычисляется число  $g$ :

$$g = v^{(p-1)/q} \pmod{p}, \quad (10.3)$$

где  $v$  – любое число, меньшее ( $p - 1$ ), для которого выполняется условие:

$$v^{(p-1)/q} \pmod{p} > 1.$$

Числа  $p$ ,  $q$ ,  $v$  могут использоваться группой лиц. Еще один элемент открытого ключа  $y$  вычисляется в соответствии с выражением

$$y \equiv g^x \pmod{p}, \quad (10.4)$$

где  $x < q$ ;  $x$  – закрытый ключ.

Общая схема генерации и верификации ЭЦП приведена на рис. 10.3. Здесь  $H(m)$  – хеш подписываемого сообщения. ЭЦП

состоит из двух чисел:  $r$  и  $s$ . Число  $k$  здесь играет такую же роль, что и одноименный параметр в шифре Эль-Гамаля.

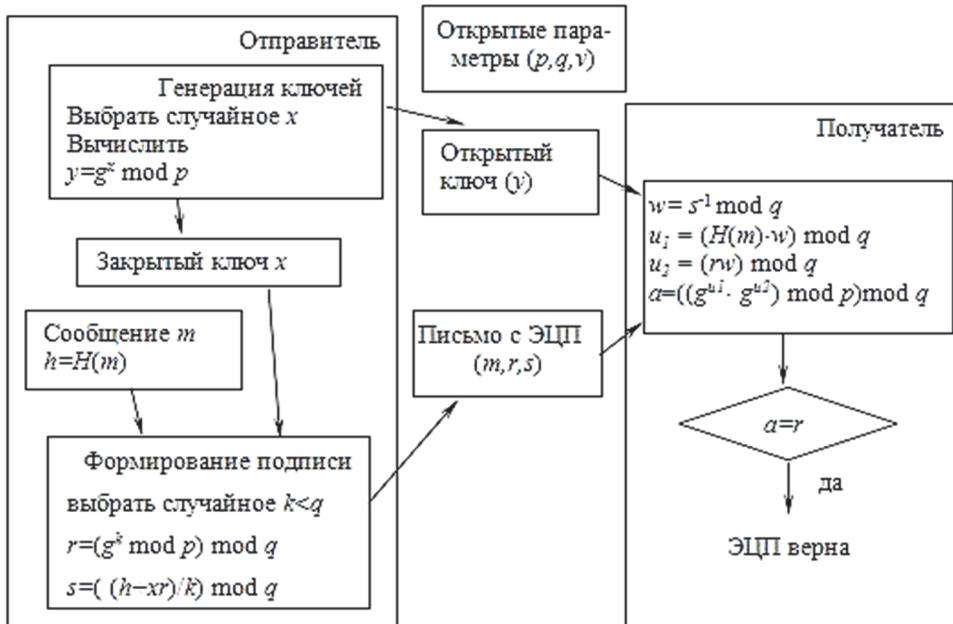


Рис. 10.3. Общая схема генерации и верификации ЭЦП DSA

### 10.1.2.3. ЭЦП Эль-Гамаля

Ключевая информация отправителя для ЭЦП создается точно так же, как это описано в материалах к лабораторной работе № 8. Она состоит из тех же элементов, что и ключи в DSA. Основное отличие в применении расчетов состоит в том, что результатом зашифрования является только одна пара чисел, а не пара для каждого блока исходного сообщения. Причем в рассматриваемом случае таким сообщением является хеш подписываемого документа:  $H(M_o)$ .

Итак, ключевая информация отправителя: открытый ключ:  $y, g$  и  $p$ ; тайный ключ:  $x$ . Чтобы подписать сообщение  $M_o$ , обладатель используемых для ЭЦП ключей должен выбрать, как и в предыдущей схеме, случайное число  $k$ , взаимно простое с  $(p - 1)$ . Затем вычисляются числа  $a$  и  $b$ , являющиеся цифровой подписью ( $S = \{a, b\}$ ):

$$a \equiv g^k \text{ mod } p; \quad (10.5)$$

для вычисления  $b$  с помощью расширенного алгоритма Евклида решается уравнение

$$H(M_o) \equiv (xa + kb) \text{ mod } (p - 1). \quad (10.6)$$

Получателю отправляется сообщение  $M' = M_o \| S$ .

Для верификации подписи вычисляется хеш полученного сообщения  $H(M_{\Pi}) = h$ . Далее нужно убедиться, что выполняется равенство

$$y^a a^b \equiv g^h \pmod{p}. \quad (10.7)$$

Если равенство выполняется, подпись верифицируется.

#### **10.1.2.4. ЭЦП Шнорра**

Рассматриваемая схема является основой стандарта ЭЦП в Беларуси. Алгоритм ЭЦП К. Шнорра (K. Schnorr) является вариантом алгоритма ЭЦП Эль-Гамаля.

Одной из особенностей ЭЦП Эль-Гамаля является то, что число  $p$  должно быть очень большим, чтобы сделать действительно трудной проблему дискретного логарифма. Рекомендуемая длина  $p$  должна составлять по крайней мере 1024 бита. Чтобы уменьшить размер подписи, Шнорр предложил новую схему, но с уменьшенным размером подписи.

Ключевая информация:  $p$  – простое число в диапазоне от 512 до 1024 битов;  $q$  – 160-битное простое число, делитель  $(p - 1)$ ; любое число  $g$  ( $g \neq 1$ ) такое, что

$$g^q \equiv 1 \pmod{p}. \quad (10.8)$$

Числа  $p, g, q$  являются открытыми и могут применяться группой пользователей.

Выбирается число  $x < q$  ( $x$  является тайным ключом) и вычисляется последний элемент открытого ключа:

$$y \equiv g^{-x} \pmod{p}. \quad (10.9)$$

Секретный ключ имеет длину не менее 160 битов.

Для подписи сообщения  $M_0$  выбирается случайное число  $k$  ( $1 < k < q$ ) и вычисляется параметр  $a$ :

$$a \equiv g^k \pmod{p}. \quad (10.10)$$

Далее вычисляется хеш от канкатенации сообщения  $M_0$  и числа  $a$ :  $h = H(M_0 || a)$ . Обратим внимание, что хэш-функция непосредственно не применяется к сообщению. Создается хеш-образ подписываемого сообщения, спереди присоединенного к числу  $a$ . Далее вычисляется значение  $b$ :

$$b \equiv (k + xh) \pmod{q}. \quad (10.11)$$

Получателю отправляются  $M' = M_0 || S$ ;  $S = \{h, b\}$ .

Для проверки подписи получатель вычисляет

$$X \equiv g^b y^h \pmod{p}. \quad (10.12)$$

Затем он проверяет выполнение равенства:  $h = H(M_{\Pi} || X)$ . Подпись достоверна, если равенство выполняется.

Основные вычисления для генерации подписи могут производиться предварительно. Порядок величин  $x$  и  $h$  – около 140 двоичных разрядов, порядок числа  $k$  – около 70–72 разрядов. С учетом этого сложность операций умножения можно считать ничтожно малой по сравнению с модульным умножением в схеме RSA.

## 10.2. Практическое задание

1. Разработать авторское оконное приложение в соответствии с целью лабораторной работы. При этом можно воспользоваться результатами выполнения предыдущих лабораторных работ, а также доступными библиотеками либо программными кодами.

Приложение должно реализовывать следующие операции:

- генерацию и верификацию ЭЦП на основе алгоритмов RSA, Эль-Гамаля и Шнорра;

- оценку времени выполнения указанных процедур при реальных (требуемых) ключевых параметрах.

Для вычисления хешей можно также воспользоваться доступными online-средствами, например *katvin* (<https://katvin.com/tools/hash-generator.html>).

2. Для выполнения необходимых операций передачи (по сети)/верификации информации обменяться открытой ключевой информацией с получателем подписанного сообщения для каждого исследуемого алгоритма (по согласованию с преподавателем).

3. Результаты оформить в виде отчета по установленным правилам.

## ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Дать определение ЭЦП.
2. Охарактеризовать основные функции ЭЦП.

3. В чем заключаются сходства и различия между собственно-ручной и электронной подписью?
4. Охарактеризовать основные способы реализации ЭЦП.
5. Имеется ли различие в использовании ключевой информации при передаче зашифрованных сообщений и при передаче подписанных (ЭЦП) сообщений?
6. Охарактеризовать криптостойкость ЭЦП на основе RSA, схемы Эль-Гамаля, схемы Шнорра, а также на основе DSA.
7. Какие элементы составляют ключевую информацию алгоритмов реализации ЭЦП, перечисленных в вопросе 6?
8. Дать сравнительные характеристики схемам ЭЦП, перечисленным в вопросе 6.
9. Охарактеризовать особенности государственного стандарта ЭЦП в Республике Беларусь.

## **Лабораторная работа № 11**

# **ИССЛЕДОВАНИЕ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ НА ОСНОВЕ ЭЛЛИПТИЧЕСКИХ КРИВЫХ**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации криптографических алгоритмов на основе эллиптических кривых (содержит 3 самостоятельных задания, каждое из которых рассчитано на 2 часа аудиторных занятий).

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и геометрическому представлению операций над эллиптическими кривыми (ЭК):

- по алгоритмам согласования ключевой информации на основе ЭК;
- алгоритмам зашифрования/расшифрования информации на основе асимметричной криптонафии и ЭК;
- алгоритмам генерации и верификации электронной цифровой подписи на основе асимметричной криптографии и ЭК;
- оценке криптостойкости систем на основе ЭК.

2. Разработать приложение для реализации указанных преподавателем методов криптопреобразования на основе ЭК.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### **11.1. Теоретические сведения**

#### **11.1.1. Эллиптические кривые над действительными числами и конечными полями**

##### **11.1.1.1. ЭК над действительными числами**

Мы ранее отмечали, что криптография базируется на задачах факторизации, дискретного логарифмирования и операциях над

точками эллиптической кривой (Elliptic Curve, EC; EC Cryptography, ECC). Последние являются предметом исследования в данной работе.

Перед выполнением лабораторной работы целесообразно ознакомиться с базовыми элементами теории эллиптических кривых (см., например, п. 5.4.4 в [3]).

Здесь вспомним основные определения и кратко проанализируем базовые операции над точками эллиптических кривых (ЭК).

**Определение 1.** *Эллиптические кривые* – математический объект, который может быть определен над любым полем.

**Определение 2.** *Эллиптическая кривая* над вещественными числами – это множество точек, описываемых уравнением

$$y^2 = x^3 + ax + b, \quad (11.1)$$

при этом константы ( $a$  и  $b$  – вещественные числа) должны удовлетворять условию

$$4a^3 + 27b^2 \neq 0. \quad (11.2)$$

Нетрудно понять, что вид ЭК (11.1) также задается парой чисел:  $a$  и  $b$ .

Формула (11.1) называется *уравнением Вейерштрасса*, а условие (11.2) исключает из рассмотрения *кривые с особыми точками* или *особые кривые*.

В зависимости от значений  $a$  и  $b$  ЭК могут принимать на плоскости разные формы (см. также [3]).

**Определение 3.** Частью ЭК является *бесконечно удаленная точка* (также известная как *идеальная точка*), которую мы обозначим символом  $O$ .

**Определение 4.** *Группа* – непустое множество с определенной на нем бинарной операцией, называемой сложением и удовлетворяющей некоторым аксиомам.

На основе последнего определения мы можем определить группу для ЭК.

**Определение 5.** *Группа для ЭК* есть непустое множество, элементы которого являются точками ЭК, обладающими следующими свойствами:

- *единичный элемент* – это бесконечно удаленная точка  $O$ ;
- *обратная величина точки*  $R$  – это точка, симметричная относительно оси  $X$ ;

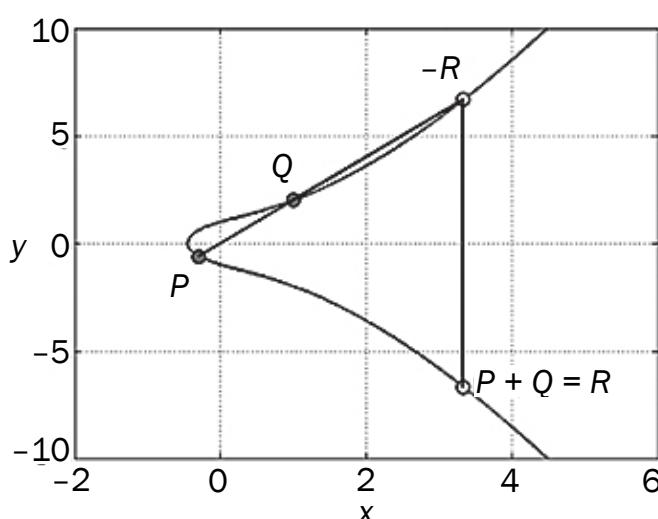
- *сложение* задается следующим правилом: сумма трех ненулевых точек  $P$ ,  $Q$  и  $-R$ , лежащих на одной прямой, будет равна  $P + Q + (-R) = O$ .

В соответствии с этим можем сформулировать *законы сложения точек эллиптической кривой*:

- прямая, проходящая через точки  $R$  и  $-R$ , является вертикальной прямой, которая не пересекает ЭК ни в какой третьей точке; если  $R = (x, -y)$ , то  $R + (x, y) = O$ . Точка  $(x, y)$  является отрицательным значением точки  $R$  и обозначается  $-R$ . Таким образом, по определению  $R + (-R) = O$ ;

- $P + Q = R$ : пусть  $P$  и  $Q$  – две различные точки ЭК (рис. 11.1), и  $P$  не равно  $Q$ ; если проведем через  $P$  и  $Q$  прямую, то она пересечет ЭК еще только в одной точке, называемой  $-R$ ; точка  $-R$  отображается относительно оси  $X$  в точку  $R$ , равную сумме точек  $P$  и  $Q$ :  $P + Q = R$ .

Геометрическая интерпретация операции сложения двух точек показана на рис. 11.1.



**Рис. 11.1.** Пояснение к операции сложения двух точек  $P$  и  $Q$  эллиптической кривой  $y^2 = x^3 + 2x + 1$  ( $a = 2$ ,  $b = 1$ )

Что будет, если  $P = Q$ ? В этом случае мы можем говорить об операции *удвоения точки*:  $P + P = 2P$ . Обобщив (к точке  $2P$  можно прибавить еще раз точку  $P$ :  $2P + P$ ), сформулируем принцип умножения точки  $P$  на целое положительное число  $n$  – это сумма  $n$  точек  $P$ :  $nP = P + P + \dots + P$ .

Скалярное умножение осуществляется посредством нескольких комбинаций сложения и удвоения точек эллиптической кривой.

Например, точка  $25P$  может быть представлена как  $25P = 2(2(2(2P)) + 2(2(2P))) + P$ .

Понятно, что каждая точка на плоскости задается парой координат:  $x$  и  $y$ .

Числа  $x$  и  $y$  являются *рациональными*, а точки  $P, Q, R$  и  $-R$  (как и любые точки ЭК) – *рациональными точками*.

Если  $P = (x_1, y_1)$  и  $Q = (x_2, y_2)$ , то  $P + Q = (x_3, y_3)$  определяется в соответствии с правилами:

$$x_3 = \lambda^2 - x_1 - x_2; \quad (11.3)$$

$$y_3 = \lambda(x_1 - x_3) - y_1, \quad (11.4)$$

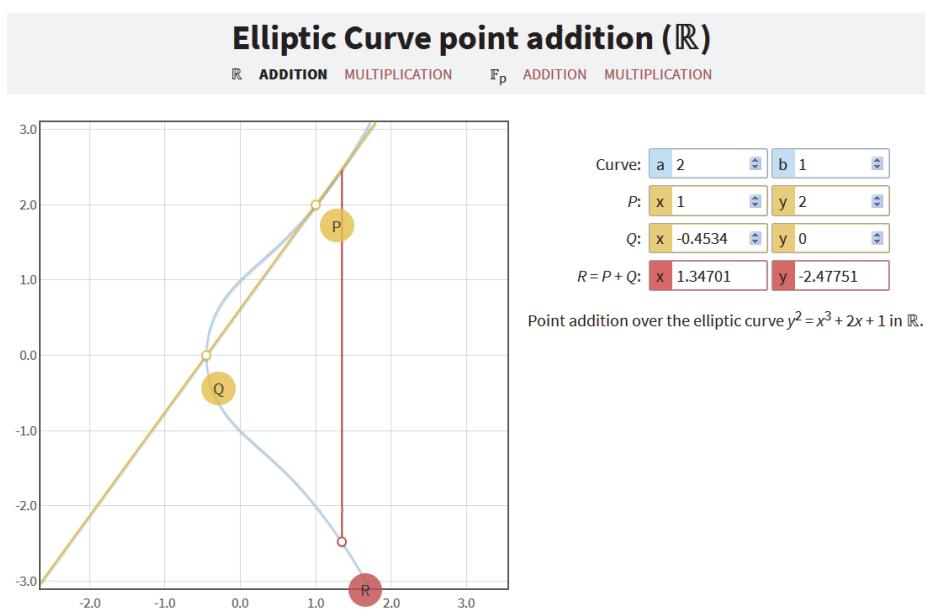
где

$$\lambda = (y_2 - y_1)/(x_2 - x_1) \text{ при } P \neq Q \text{ и } \lambda = (3(x_1)^2 + a)/2y_1 \text{ при } P = Q. \quad (11.5)$$

Из этого следует, что число  $\lambda$  – 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  $\lambda$ .

Хорошее представление об операциях над точками различных ЭК можно получить, воспользовавшись онлайн-приложением, доступным по ссылке: <https://cdn.rawgit.com/andreacorbellini/ecc/920b29a/interactive/reals-add.html>.

Для примера на рис. 11.2 приведено окно этого приложения.



**Рис. 11.2.** Окно приложения  
*Elliptic Curve Points Addition*

На приведенном рисунке как раз представлен пример над точками (координаты в правой части экрана), упоминавшимися выше (см. рис. 11.1) ЭК. Отметим также, что при этом выбраны опции **R** и **Addition** (вторая строка), соответствующие операции сложения над рациональными числами.

Рассмотрим пример.

**Пример 1.** Пусть ЭК задается уравнением с параметрами  $a = -7$ ,  $b = 10$ . Точки  $P(1, 2)$  и  $Q(3, 4)$ . Нужно вычислить сумму точек:  $P + Q = R$ .

Воспользуемся выражениями (11.3)–(11.5):

$$\begin{aligned}\lambda &= (2 - 4) / (1 - 3) = 1, \\ x_R &= x_3 = 1^2 - 1 - 3 = -3, \\ y_R &= y_3 = 2 + 1 \cdot (-3 - 1) = -2.\end{aligned}$$

Тот же результат получаем и при использовании указанного выше приложения.

**Пример 2.** Для той же ЭК при  $P(1, 2) = Q(1, 2)$  получим для  $P + Q = R = 2P$ :

$$\begin{aligned}\lambda &= 3(1^2 - 7) / (2 \cdot 2) = -1, \\ x_R &= x_3 = (-1)^2 - 1 - 1 = -1, \\ y_R &= y_3 = 2 + (-1) \cdot (-1 - 1) = 4.\end{aligned}$$

Таким образом, получили точку  $2P(-1, 4)$ .

С помощью упомянутого выше приложения (опции **R** и **Multiplication**) можно вычислить любые операции *скалярного умножения точки*.

Для заданных  $n$  и  $P$  существуют алгоритмы вычисления  $Q = nP$ . Если же известны  $Q$  и  $P$ , а нам нужно определить  $n$ , то такая задача нам известна как *задача логарифмирования*.

### 11.1.1.2. ЭК над конечными полями

Именно этот тип ЭК будет нас интересовать в плане практического применения.

**Определение 6. Конечное поле** – это множество конечного числа элементов. Примером конечного поля является множество целых чисел по модулю  $p$ , где  $p$  – простое число.

Поле обозначается как  $GF(p)$  или  $F_p$ . Здесь операции сложения и умножения работают как в модулярной арифметике.

Например, поле  $F_{13}$  ( $p = 13$ ) состоит из чисел:  $0, 1, \dots, 12$ .

**Определение 7. Эллиптическая кривая над полем  $F_p$**  задается теми же уравнениями, что и ЭК над действительными числами, только все вычисления производятся по модулю  $p$  ( $\text{mod } p$ ):

$$y^2 \equiv x^3 + ax + b \pmod{p}, \quad (11.6)$$

далее для упрощения используем знак простого неравенства:

$$4a^3 + 27b^2 \neq 0 \pmod{p} \quad (11.7)$$

и т. д.

Формально ЭК над полем задается так:  $E_p(a, b)$ .

Важно отметить, что, как и ранее, существует точка (бесконечно удаленная)  $O$ ;  $a$  и  $b$  – вещественные числа.

Прежде чем приступить к алгебраическим операциям над точками кривой, такими как суммирование двух разных точек на ЭК и удвоение точек, кратко проанализируем операции для расчета точек, принадлежащих ЭК. Должны быть приняты некоторые предположения, такие как площадь, на которой будут рассчитываться точки кривой, и функция кривой.

Рассмотрим конкретный пример.

**Пример 3.** Пусть ЭК формально задается записью  $E_{13}(6, -9)$ . Проверяем выполнение условия (11.7). Исходя из этого, координаты расположения точек должны быть ограничены квадратом некоторых чисел по модулю 13 (левая часть основного уравнения –  $y^2$ ). Здесь стоит отметить известную нам цикличность в вычислениях на основе модулярной арифметики. Это видно для нашего случая из табл. 11.1.

Таблица 11.1  
Цикличность квадратов целых чисел над полем  $F_{13}$

$0^2 \pmod{13} = 0$	$13^2 \pmod{13} = 0$
$1^2 \pmod{13} = 1$	$14^2 \pmod{13} = 1$
$2^2 \pmod{13} = 4$	$15^2 \pmod{13} = 4$
$3^2 \pmod{13} = 9$	$16^2 \pmod{13} = 9$
$4^2 \pmod{13} = 3$	$17^2 \pmod{13} = 3$
$5^2 \pmod{13} = 12$	$18^2 \pmod{13} = 12$
$6^2 \pmod{13} = 10$	$19^2 \pmod{13} = 10$
$7^2 \pmod{13} = 10$	$20^2 \pmod{13} = 10$
$8^2 \pmod{13} = 12$	$21^2 \pmod{13} = 12$
$9^2 \pmod{13} = 3$	$22^2 \pmod{13} = 3$
$10^2 \pmod{13} = 9$	$23^2 \pmod{13} = 9$
$11^2 \pmod{13} = 4$	$24^2 \pmod{13} = 4$
$12^2 \pmod{13} = 1$	$25^2 \pmod{13} = 1$

Числа, приведенные после знаков равенства, являются *квадратичными вычетами* по модулю 13. В данном примере это числа из множества  $\{1, 3, 4, 9, 10, 12\}$  (обычно число 0 не включают в такие множества).

Важным элементом рассматриваемой технологии является определение точек кривой с целочисленными координатами. Эти задачи в общем случае решаются на основе известных алгоритмов, которые мы здесь опустим. Имея приведенные в табл. 11.1 вычисления квадратов чисел по модулю 13, рассмотрим ситуацию для  $x = 0$ . Подставим это значение в правую часть уравнения (11.6), имея в виду ЭК  $E_{13}(6, -9)$ :

$$y^2 = 0^3 + 6 \cdot 0 - 9 \pmod{13},$$

откуда получим  $y^2 = -9 \pmod{13}$ ,  $y^2 = 4$  и  $y = \pm 2$ . Таким образом, пользуясь данными из табл. 11 (смотрим строки с числами 4 справа от знака равенства), определяем, что точками нашей ЭК будут:  $(0, 2)$  и  $(0, 11)$ ; здесь мы приняли во внимание то, что значение некоторого целого отрицательного числа  $(-k)$  по модулю  $(p)$  вычисляется следующим образом:

$$(-k) \pmod{p} \equiv -(k \pmod{p}) + p.$$

Следуя приведенной логике рассуждений, определим, например, точки при  $x = 3$ :  $y^2 = 3^3 + 6 \cdot 3 - 9 \pmod{13} = 36 \pmod{13} = 10$ . Обращаем внимание на 7-ю и 8-ю строки левого столбца табл. 11.1 и устанавливаем координаты еще 2 точек ЭК:  $(3, 6), (3, 7)$ .

Теперь вернемся к  $x = 1$ :  $y^2 = 1^3 + 6 \cdot 1 - 9 \pmod{13} = -2 \pmod{13} = 11$ . В табл. 11.1 не найдено ни одного соответствия. Это означает, что на рассматриваемой ЭК нет ни одной точки, координата  $x$  которой равна 3.

На рис. 11.3 представлены все точки для ЭК  $E_{13}(6, -9)$ .

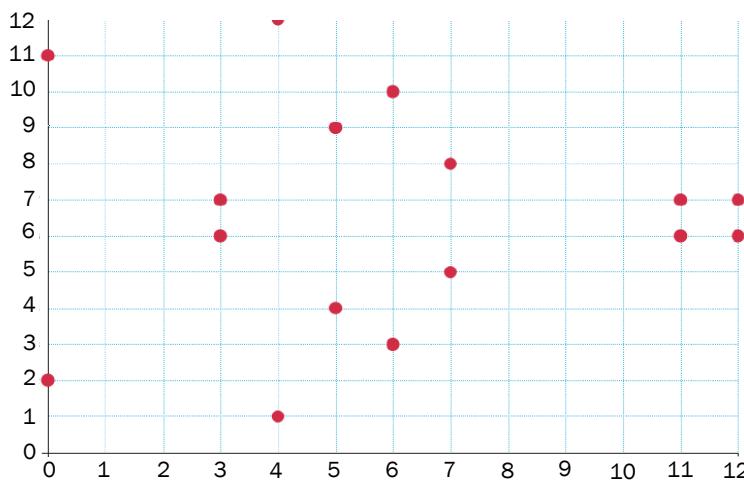
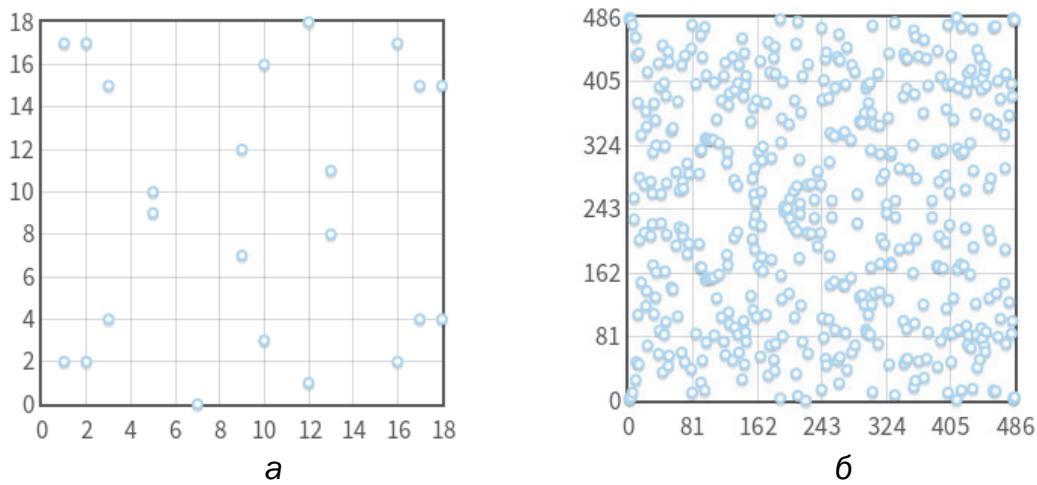


Рис. 11.3. Точки ЭК  $E_{13}(6, -9)$

На рис. 11.4 показаны точки эллиптической кривой  $(7, 10)$  из примера 1 для  $p = 19$  (a) и для  $p = 487$  (б).

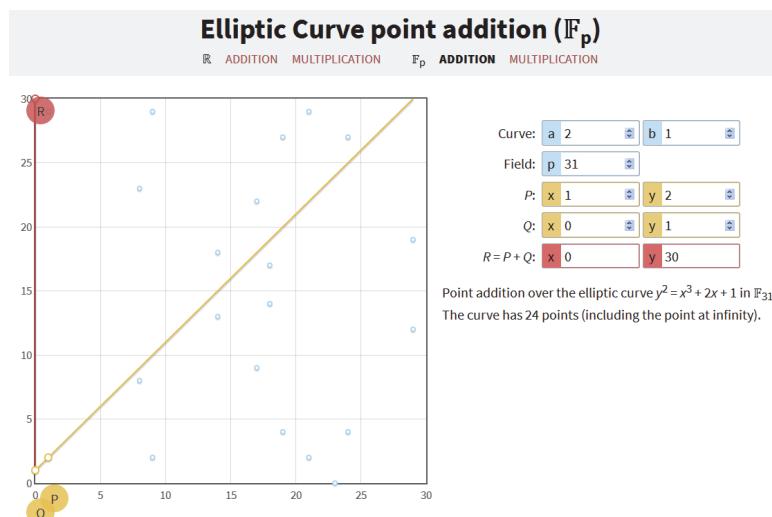


**Рис. 11.4.** Отображение точек ЭК  $y^2 = x^3 - 7x + 10 \pmod{p}$   
(источник: <https://habr.com/ru/post/335906/>)

Из приведенных примеров можно заметить, что для каждого  $x$  существует максимум две точки. Отметим также симметрию в расположении точек относительно  $y = p/2$ . То, что раньше было непрерывной кривой, теперь стало множеством отдельных точек на плоскости  $XY$ , координаты которых ( $x$  и  $y$ ) являются целыми числами.

Можно также сказать, что три точки находятся на одной прямой, если существует прямая, соединяющая их. Рис. 11.3 и 11.4 дают более полное представление о числовом пространстве точек ЭК над конечным полем.

На рис. 11.5 показано геометрическое отображение операции суммирования двух точек с параметрами, примерно соответствующими рис. 11.2 (не для любых параметров выполняется операция).



**Рис. 11.5.** Геометрическое отображение операции суммирования двух точек ЭК с параметрами, расположенными в правой части экрана

Рассмотрим пример.

**Пример 4.** Пусть  $p = 23$ . Рассмотрим ЭК  $y^2 = x^3 + x + 1 \pmod{23}$ :  $E_{23}(1, 1)$ .

Кривая состоит из следующих точек:  $(0, 1); (0, 22); (1, 7); (1, 16); (3, 10); (3, 13); (4, 0); (5, 4); (5, 19); (6, 4); (6, 19); (7, 11); (7, 12); (9, 7); (9, 16); (11, 3); (11, 20); (12, 4); (12, 19); (13, 7); (13, 16); (17, 3); (17, 20); (18, 3); (18, 20); (19, 5); (19, 18)$ , т. е. всего 27 точек.

Не забываем, что во всех случаях эллиптической кривой принадлежит также точка  $O$ .

Пусть  $P = (3, 10)$  и  $Q = (9, 7)$ . Найдем  $P + Q$  и  $2P$ .

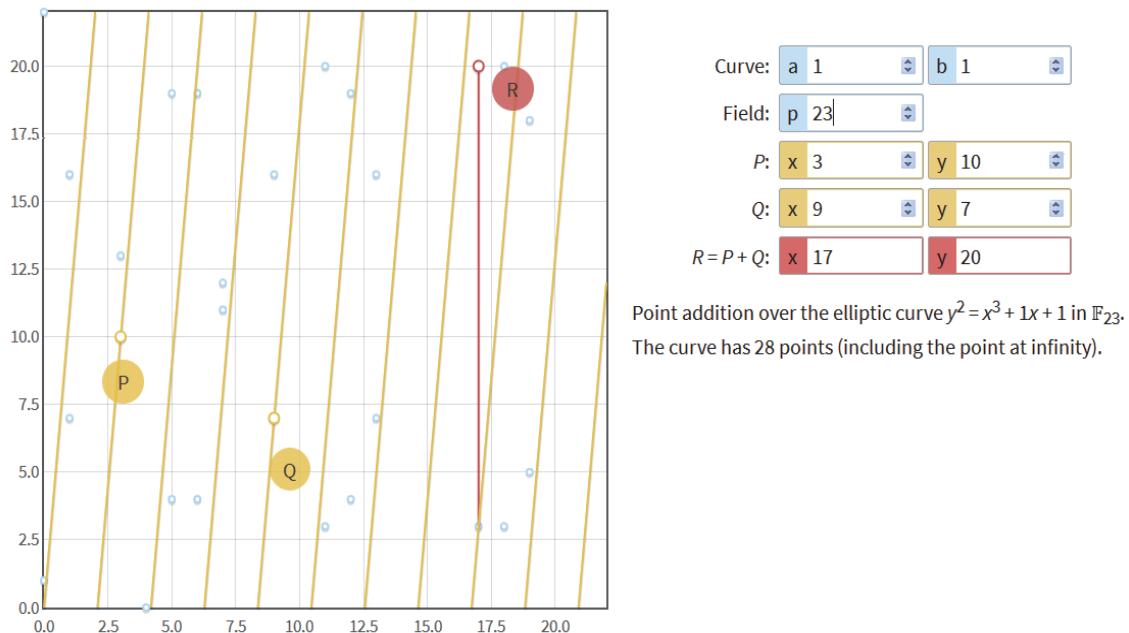
Пусть  $P + Q = (x_3, y_3)$ , тогда при  $\lambda = (7 - 9) / (9 - 3) \pmod{23} = 11 \pmod{23}$  имеем:

$$x_3 = 121 - 3 - 9 \pmod{23} = 109 \pmod{23} = 17,$$

$$y_3 = 11(3 + 6) - 10 \pmod{23} = 89 \pmod{23} = 20.$$

Таким образом,  $P + Q = (17, 20)$ .

Такой же результат получен и с использованием упомянутого онлайн-ресурса (рис. 11.6).



**Рис. 11.6.** Окно приложения с результатами выполнения операции сложения двух точек из примера 2

Найдем теперь точку  $2P = P + P = (x_3, y_3)$  – с формальной точки зрения это также будет третья точка.

Для этого случая

$$\lambda = (3 \cdot 9 + 1) \pmod{23} = 6 \pmod{23}$$

и с учетом последнего для вычисления координаты  $y_3$ :

$$x_3 = 36 - 6 \pmod{23} = 30 \pmod{23} = 7 \pmod{23};$$

$$y_3 = 6 \cdot (3 - 7) - 10 \pmod{23} = -34 \pmod{23} = 12 \pmod{23}.$$

Таким образом,  $2P = (7, 12)$ .

При последовательном выполнении сложения  $nP = P + P + P + \dots + P$  на каждом шаге будет получаться точка, которая также должна принадлежать  $E_p(a, b)$ . В силу того что эллиптическая группа содержит конечное множество точек, наступает такой момент, что для некоторых результатов вычислений будет выполняться равенство  $qP = O$  (см. пример 5.12 из [3], где  $5A = O$ , т. е. здесь  $q = 5$ ).

**Пример 5.** Для точки  $P(4, 2)$  ЭК вида  $E_7(4, 1)$  справедливы следующие соотношения:

$$2P = (4, 2) + (4, 2) = (0, 1),$$

$$3P = (0, 1) + (4, 2) = (0, 6),$$

$$4P = 2(0, 1) = (4, 5),$$

$$5P = (0, 1) + (0, 6) = O.$$

Для данного случая также  $q = 5$ .

Если взять разные точки на одной и той же ЭК, получим разные  $q$ .

В табл. 11.2 показаны умножения точки для ЭК  $E_5(0,1)$ .

Таблица 11.2  
Результаты выполнения операции  $P + \dots + P$  для кривой  $E_5(0,1)$

+	(0, 1)	(0, 4)	(2, 2)	(2, 3)	(4, 0)
$P$	(0, 1)	(0, 4)	(2, 2)	(2, 3)	(4, 0)
$2P$	(0, 4)	(0, 1)	(0, 4)	(0, 1)	O
$3P$	O	O	(4, 0)	(4, 0)	(4, 0)
$4P$	(0, 1)	(0, 4)	(0, 1)	(0, 4)	O
$5P$	(0, 4)	(0, 1)	(2, 3)	(2, 2)	(0, 4)
$6P$	O	O	O	O	O

Данные приведенной таблицы подтверждают наши выводы.

Если требуется, например, точку  $P$  сложить саму с собой  $z$  раз, то это означает, что нужно выполнить вычисление  $zP$ . Для реализации этой операции существует простой метод на основе операции сложения точек. Число  $z$  представляется в двоичном виде. И далее

вычисляются необходимые составляющие общей суммы на основе весовых (единичных) разрядов двоичного числа  $z$ . Рассмотрим это на примере.

**Пример 6.** Пусть  $z = 171$ . Это число в двоичном виде выглядит так: 10101011. В соответствии с весом «1» мы должны сложить следующие составляющие (слагаемые) общей суммы:  $171P = P + 2P + + 8P + 32P + 128P$ .

Первое из приведенных слагаемых известно. Второе слагаемое:  $2P = P + P$ , промежуточное вычисление:  $4P = 2P + 2P$ , третье слагаемое:  $8P = 4P + 4P$ , промежуточное вычисление:  $16P = 8P + 8P$ , промежуточное вычисление:  $16P = 8P + 8P$ , четвертое слагаемое:  $32P = 16P + 16P$ , промежуточное вычисление:  $64P = 32P + 32P$ , последнее слагаемое:  $128P = 64P + 64P$ .

**Определение 8.** Если мы складываем два значения, кратных  $P$ , то получаем значение, кратное  $P$  (т. е. значения, кратные  $P$ , замкнуты относительно операции сложения). Это означает, что *множество кратных  $P$  значений – это циклическая подгруппа* группы, образованной эллиптической кривой.

**Определение 9.** Наименьшее значение числа  $q$ , для которого выполняется равенство  $qP = O$ , называется **порядком точки  $P$** .

**Определение 10. Порядок группы точек эллиптической кривой** равен числу различных точек ЭК, включая точку  $O$ .

**Определение 11.** Точка  $P$  называется **генератором** или **базовой точкой** циклической подгруппы (такую точку во многих документах обозначают символом  $G$ ).

Порядок точки  $P$  связан с порядком  $m$  ЭК **теоремой Лагранжа**, согласно которой *порядок подгруппы – это делитель порядка исходной группы*. Иными словами, если ЭК содержит  $m$  точек, а одна из подгрупп содержит  $q$ , то  $q$  является делителем  $m$ .

Для ЭК  $E_p(a, b)$  порядок  $m$  группы точек должен удовлетворять неравенству

$$p + 1 - 2p^{1/2} \leq m \leq p + 1 + 2p^{1/2}.$$



Как и в случае с непрерывными ЭК, теперь важным является вычисление некоторого числа  $d$ , если мы знаем  $P$  и  $Q$  для  $Q = dP$ . Это и есть **задача дискретного логарифмирования** для эллиптических кривых.

Эта задача аналогична задаче дискретного логарифмирования, используемой в других криптосистемах, таких как алгоритм DSA, протокол Диффи – Хеллмана и схема Эль-Гамала.



**В криптографии на основе ЭК тайный ключ – это случайное целое  $d$ , выбранное из множества  $\{1, 2, \dots, q - 1\}$ , где  $q$  – порядок подгруппы; открытый ключ – это точка  $Q$ , такая, что  $Q = dG$ , где  $G$  – базовая точка подгруппы.**

Криптостойкость алгоритмов на основе ЭК определяется, например, для алгоритма ЭЦП в стандарте Республики Беларусь [51] параметром  $l$ , называемым *уровнем стойкости* и принимающим значения (рекомендуется) из  $\{128, 192, 256\}$ . При этом для взлома ключа злоумышленнику нужно выполнить  $2^l$  операций.

### **11.1.1.3. Основные этапы генерации ключевой информации на основе ЭК**

*Первый этап: выбор (генерация) ЭК.* Обычно он основан на выполнении следующих условий и операций.

1.1. Входными параметрами являются: число  $l$ , число  $p$ , удовлетворяющее условию  $2^{2l-1} < p < 2^{2l}$ ,  $p = 3 \bmod 4$ ,  $0 < a < p$ . Можно использовать некоторое простое число  $p = 2^{2l} - c$ , где  $c$  – небольшое натуральное число.

1.2. Выбирается число  $b$  такое, что  $0 < b < p$ .

Таким образом, задана ЭК:  $E_p(a, b)$ .

1.3. Выбираются порядок  $q$  (простое число) и генерирующая точка  $G$ , которая задается двумя координатами, например  $G = (0, y_G)$ .

Дополнительно к рассмотренным действиям стандарт [51] предусматривает использование вспомогательного параметра  $(s, seed)$  – произвольное 64-битное число.

Для примера в таблицах ниже (табл. 11.3 и 11.4) [51] приведены параметры ЭК для двух значений  $l$ . Здесь нижние индекс в левом столбце обозначают битовую длину числа.

*Второй этап: генерация ключевой информации.*

2.1. Входными параметрами являются:  $p, a, b, q$  и  $G$ .

2.2. Генерируется тайный ключ – число  $d$ , выбранное из множества  $\{1, 2, \dots, q - 1\}$ .

2.3. Вычисляется открытый ключ – точка  $Q$ :

$$Q = dG, \quad (11.8)$$

к открытому ключу также относятся  $p, a, b, q$ .

Отметим также, что можно сгенерировать ключевую информацию на основе ЭК, воспользовавшись известной нам библиотекой *OpenSSL*. Например, если воспользоваться версией *OpenSSL 1.1.1L*

в системе *Debian 9* (с помощью команды с двумя разными псевдонимами (выделены жирным)):

*openssl ecparam -name secp192k1 -genkey -out secp192k1,*

*openssl ecparam -name secp521r1 -genkey -out secp521r1,*

то получим тайные ключи соответственно следующего содержания:

*MFwCAQEEGLDsGwgZq/Kq4suR74ftkipbKMRmoWDtlqAHBgUr  
gQQAH6E0AzIABPfKAzFU+QKsh+I7a6K5taNUe3TZAdLMp92RpYo  
T0PIrmGD3QVRcqAmqZSba6kanKg==*

*MIHcAgEBBEIAvv7P//IWx3QQis5Hb25eN/UY5isVJk+s56ZDSTle  
Ucrqj2mNH4Y3xWLXGMtpmDJRiHalCv3MDt/T5h67daHaViagBwYF  
K4EEACOhgYkDgYYABABgOPla5ygHB/j79g0R2N12/tv4YlIj6ZA+t2F  
htvEMPvj9QHMg5sN45yjGKmLlIwEMP2YWxjPj3YL0Z0uLO9BBYwB  
UGVCPEWKylC8x5qGL1ypG6shCPTUcXQxLuFMmKv+AaDH24TCd  
Bvl9nYANhlxZKv96Pb/lari3OKZkmO5zgVWKCw==*

Таблица 11.3

#### Стандартные параметры ЭК (*I* = 128)

<i>p</i> $\langle p \rangle_{256}$	2 <sup>256</sup> – 189 43FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF <sub>16</sub>
<i>a</i> $\langle a \rangle_{256}$	2 <sup>256</sup> – 192 40FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF <sub>16</sub>
<i>b</i> $\langle b \rangle_{256}$	F1039CD6 6BD2EB2 53928B97 6950F54C BEFBD8E4 AB3AC1D2 EDA8F315 156CCE77 <sub>16</sub>
seed	5E380100 00000000 <sub>16</sub>
<i>q</i> $\langle q \rangle_{256}$	2 <sup>256</sup> – 51 359303463 308904523 350978545 619999225 07663D26 99BF5A7E FC4DFB0D D68E5CD9 FFFFFFFF FFFFFFFF FFFFFF FFFFFFFF <sub>16</sub>
<i>y<sub>G</sub></i> $\langle y_G \rangle_{256}$	936A5104 18CF291E 52F608C4 66399178 5D83D651 A3C9E45C 9FD616FB 3CFCF76B <sub>16</sub>

Таблица 11.4

#### Стандартные параметры ЭК (*I* = 192)

<i>p</i> $\langle p \rangle_{384}$	2 <sup>384</sup> – 317 C3FFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF <sub>16</sub>
<i>a</i> $\langle a \rangle_{384}$	2 <sup>384</sup> – 320 COFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF <sub>16</sub>
<i>b</i> $\langle b \rangle_{384}$	64BF7368 23FCA7BC 7CBDCEF3 F0E2BD14 3A2E71E9 F96A21A6 96B1FB0F BB482771 D2345D65 AB5A0733 20EF9C95 E1DF75C <sub>16</sub>
seed	23AF0000 00000000 <sub>16</sub>

Окончание табл. 11.4

$q$	$2^{384} - 9886\ 438520659\ 958522437\ 788006980\ 66095037$ $549058207\ 958390857$ $B7A70CF3\ 3FDCCB73D\ 0AFFA4A6\ E7DA4680\ C3C4CC6C\ FFFFFFFF$ $FFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF_{16}$
$\langle g \rangle_{384}$	$51C433F7\ 31CB5EEA\ F9422A6B\ 273E4084\ 55D3B166\ 9EE74905$ $A0FF86DC\ 119A723A\ 89BF2D43\ 7E113063\ 9E9E2EA8\ 2482435D_{16}$

Полезные рекомендации по выбору параметров ЭК можно найти, например, в [52].

### 11.1.2. Использование ЭК в криптографии

Отметим еще раз, что ЭК в криптографических приложениях обычно используется на этапе генерации либо согласования ключевой информации. Таким образом, можно отметить 3 направления использования ЭК в криптографии:

- в алгоритмах согласования (передачи) ключевой информации (на основе идеи Диффи – Хеллмана);
- в алгоритмах асимметричного шифрования/декшифрования сообщений;
- в алгоритмах генерации/верификации ЭЦП.

#### 11.1.2.1. Реализация алгоритма Диффи – Хеллмана на основе ЭК

Рассмотрим наиболее общий случай. Предположим, что  $E_p$  – это ЭК над  $F_p$ , а  $Q$  – заранее определенная и согласованная сторонами **А** и **В** точка на  $E$ .

Отправитель **А** выбирает тайное случайное число  $k_A$ , вычисляет точку  $P_A = k_A Q$  и отправляет ее получателю **В**. **В** действует аналогично: он случайным образом выбирает число  $k_B$ , вычисляет случайное число  $k_A$ , вычисляет точку  $P_B = k_B Q$  и отправляет результат стороне **А**.

Общий ключ  $P = k_A k_B Q$ . Отправитель **А** вычисляет  $P$  путем умножения числа  $P_B$ , полученного от получателя **В**, на его секретное число  $k_A$ . Похожим образом действует другая сторона.

#### 11.1.2.2. Реализация алгоритма зашифрования/расшифрования на основе ЭК

Вспомним, что процедура предусматривает использование ключей получателя (стороны **В**). Рассмотрим это на примере алгоритма Эль-Гамаля.

Принимаем также во внимание, что зашифрованное сообщение  $M$  или каждый зашифрованный блок ( $m_i$ ) этого сообщения состоят из двух чисел. Обратимся к лабораторной работе № 8, где блок шифртекста ( $c_i$ ) в соответствии с выражениями (8.9) и (8.10) мы обозначали двумя символами  $a_i$  и  $b_i$  и вычисляли как

$$a_i \equiv g^k \pmod{p}, b_i \equiv (y^k m_i) \pmod{p}.$$

Поскольку символы  $a$  и  $b$  мы зарезервировали в текущей работе для обозначения параметров ЭК, то блок шифртекста сейчас будем обозначать соответственно символами  $C_{i1}$  и  $C_{i2}$ .

При использовании ЭК зашифрование предполагает представление сообщения в виде точки  $P$  (или представления каждого блока сообщения в виде разных точек  $P_i$ ) ЭК с известной точкой  $G$  и известным  $Q$ . Соответственно шифртекст – это две точки на той же ЭК:  $C_1$  и  $C_2$  или  $C_{i1}$  и  $C_{i2}$ .

Предположим, что шифруемое сообщение  $M$  – это точка  $P$  на ЭК.

Страна А выбирает некоторое случайное число  $k$  и далее выполняет вычисления с использованием открытого ключа стороны В:

$$C_1 = kG, C_2 = P + kQ. \quad (11.9)$$

Получатель для расшифрования сообщения вычисляет:

$$P = C_2 - dC_1. \quad (11.10)$$

Знак « $-$ » в (11.10) означает сложение с инверсией: инверсией по отношению к точке  $(x, y)$  является точка  $(x, -y)$  на ЭК.

Рассмотрим пример.

**Пример 7.** Пусть страна В использует ЭК вида  $E_{67}(2, 3)$ ,  $G = (2, 22)$  и  $d = 4$ . Тогда  $Q = dG = 4G = (13, 45)$ ; здесь расчеты, которые проводились на основе (11.3)–(11.5), опускаются.

Полагаем далее, что шифруемое сообщение  $M$  соответствует точке  $P = (24, 26)$ , а  $k = 2$ . Тогда в соответствии с (11.9) получен шифртекст:

$$C_1 = 2G = 2 \cdot (2, 22) = (35, 1), C_2 = P + kQ = (24, 26) + 2 \cdot (13, 45) = = (21, 44).$$

Таким образом, сообщению соответствует шифртекст из двух точек:  $C_1 = (35, 1)$ ,  $C_2 = (21, 44)$ .

Для расшифрования страна В вычисляет последовательно:

$$dC_1 = 4 \cdot (35, 1) = (23, 25),$$

далее инвертирует точку  $(23, 25)$ :  $(23, 42)$ , поскольку  $-25 \pmod{67} = 42$ , и, наконец, выполняется сложение в соответствии с (11.10):

$C_2 + (23, 42) = (24, 26)$ , что соответствует исходной точке  $P$ , т. е. сообщению  $M$ .

Сравнительная оценка влияния размера ключа (в битах) для классической асимметричной системы шифрования (RSA) и асимметричной системы на основе ЭК дана американским институтом стандартов NIST, которую мы приводим ниже в виде табл. 11.5.

Таблица 11.5

**Размер ключей, обеспечивающих  
примерно одинаковый уровень криптостойкости**

Классический RSA	На основе ЭК
102	160
2048	224
3072	256
3680	384

### 11.1.2.3. Реализация ЭЦП на основе ЭК

Рассмотрим генерацию и верификацию ЭЦП на основе алгоритма DSA и ЭК (EC) – ECDSA. Обращаем внимание на то, что используется ключевая информация отправителя (стороны А). Генерация ключей происходит так же, как и в последнем примере. Однако в анализируемом здесь случае во внимание должен приниматься еще один известный параметр ЭК: порядок точки  $G$ , т. е. число  $q$ .

Краткая характеристика алгоритма генерации и верификации ЭЦП состоит в следующем. Полагаем, что отправитель подписывает хеш  $H(M)$  сообщения  $M$ .

#### Генерация ЭЦП.

1. Выбрать число  $k$  ( $1 < k < q$ ),  $q$  – порядок точки  $G$ .
2. Вычислить точку  $kG = (x, y)$ , вычислить  $r \equiv x \pmod{q}$ ; при  $r = 0$  изменить  $k$  и повторить шаг 2.
3. Вычислить  $t \equiv k^{-1} \pmod{q}$  (например, на основе расширенного алгоритма Евклида).
4. Вычислить  $s = (t(H(M) + dr)) \pmod{q}$ ; при  $s = 0$  изменить  $k$  и повторить алгоритм.

Стороне В отсылаются сообщение  $M$  и ЭЦП (числа  $r$  и  $s$ ).

*Верификация ЭЦП.* Получатель знает алгоритм хеширования, который использовался отправителем, открытый ключ отправителя, с помощью чего выполняет следующие операции над  $M$  и полученной ЭЦП (обозначения чисел оставим без изменений).

1. Проверить выполнение условия:  $1 < r, s < q$ ; если условие не выполняется, то легитимность подписи не подтверждается, в противном случае – выполняются дальнейшие шаги.

2. Вычисляются  $H(M)$  и  $w \equiv s^{-1} \pmod{q}$ .

3. Вычисляются  $u_1 \equiv w H(M) \pmod{q}$ ,  $u_2 \equiv wr \pmod{q}$ .

4. Вычисляются  $Gu_1 + Qu_2 = (x', y')$ ,  $v \equiv x' \pmod{q}$ .

5. Сравниваются  $v$  и  $r$ ; если равенство выполняется, подтверждается легитимность подписи и целостность полученного сообщения.

**Пример 8.** Полагаем, что  $H(M) = 12$ . Используется ЭК  $E_{751}(-1, 1)$  с генерирующей точкой  $G = (384, 475)$ ,  $q = 13$  и тайным ключом  $d = 12$ ;  $Q = dG = 12(384, 475) = (384, 276)$ .

*Генерация ЭЦП.*

1. Выбирается число  $k = 3$  ( $1 < 3 < 13$ ),  $q$  – порядок точки  $G$ .

2. Вычисляется точка  $kG = 3(384, 475) = (596, 318)$ , т. е.  $x = 596$ , вычисляется  $r = x \pmod{q} = 596 \pmod{13} = 11$ .

3. Вычисляется  $t = k^{-1} \pmod{q} = 3^{-1} \pmod{13} = 9$ ,  $((9 \cdot 3) \pmod{13} = 1)$ .

4. Вычисляется  $s = (t(H(M) + dr)) \pmod{q} = (9 \cdot (12 + 12 \cdot 11)) \pmod{13} = 9$ .

Стороне **B** отсылается сообщение  $M$  и ЭЦП (числа  $r = 11$  и  $s = 9$ ).

*Верификация ЭЦП.* Получатель знает алгоритм хеширования, который использовался отправителем, открытый ключ отправителя, с помощью чего выполняет следующие операции над  $M$  и полученной ЭЦП (числа  $r = 11$  и  $s = 9$ ).

1. Подтверждается выполнение условия  $1 < r, s < q$ .

2. Вычисляется  $H(M)$  – положим, что в результате хеширования полученного сообщения  $M$  его хеш не изменился:  $H(M) = 12$ ; далее вычисляется  $w = s^{-1} \pmod{q} = 9^{-1} \pmod{13} = 3$ .

3. Вычисляются  $u_1 = w H(M) \pmod{q} = 3 \cdot 12 \pmod{13} = 10$  и  $u_2 = wr \pmod{q} = 3 \cdot 11 \pmod{13} = 7$ .

4. Вычисляются  $Gu_1 + Qu_2 = 10(384, 475) + 7(384, 276) = (596, 318) = (x', y')$ ;  $v = x' \pmod{q} = 596 \pmod{13} = 11$ .

5. Сравниваются  $v = 11$  и  $r = 11$ : равенство выполняется – подтверждается легитимность подписи и целостность полученного сообщения  $M$ .

## 11.2. Практическое задание

В основе задания – ЭК вида  $y^2 = x^3 - x + 1 \pmod{751}$ :  $a = -1$ ,  $b = 1$ ,  $p = 751$ , т. е.  $E_{751}(-1, 1)$ .

**Задание 1** (расчитано на 2 часа аудиторных занятий).

1.1. Найти точки ЭК для значений  $x$ , указанных в табл. 11.6

Таблица 11.6

**Диапазоны изменения координаты  $x$  для поиска точек ЭК**

Параметр	Вариант														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x_{\min}$	0	36	71	106	141	176	201	481	516	551	586	621	656	691	716
$x_{\max}$	35	70	105	140	175	200	235	515	550	585	620	655	690	715	750

1.2. Разработать приложение для выполнения операций над точками кривой:

а)  $kP$ ; б)  $P + Q$ ; в)  $kP + lQ - R$ ; г)  $P - Q + R$ .

Варианты коэффициентов приведены в табл. 11.7.

Таблица 11.7

**Числовые значения коэффициентов для операций над точками**

Параметр	Вариант														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$k$	8	6	7	9	11	9	11	12	8	11	7	6	9	10	11
$l$	11	10	8	7	5	7	4	5	5	3	7	7	5	3	5

В табл. 11.6 указаны координаты точек, над которыми выполняются операции.

Результаты выполнения операций представить в табличной форме.

**Задание 2** (расчитано на 2 часа аудиторных занятий).

2.1. Создать оконное приложение для зашифрования/расшифрования собственной фамилии (или имени – по выбору) на основе ЭК, указанной в задании 1, для генерирующей точки  $G = (0, 1)$ . Тайный ключ – в соответствии с вариантом из табл. 11.8.

2.2. Вычислить самостоятельно значение открытого ключа  $Q$ . При этом следует воспользоваться основной формулой (11.9), а также соотношениями (11.3)–(11.5) для случая  $P = Q$ ; не следует также забывать, что все вычисления производятся по  $\text{mod } 751$ ; см. также пример 5 (вычисление  $2P$ ) и пример 7.

Принять, что шифруемым блоком является один символ сообщения, координаты которого на ЭК соответствуют табл. 11.9 (может быть принята за основу и иная таблица).

Параметры  $k$  – по собственному усмотрению.

Таблица 11.8

## Координаты точек ЭК

Вариант	Координаты точек		
	P	Q	R
1	(58, 139)	(67, 667)	(82, 481)
2	(61, 129)	(59, 365)	(105, 369)
3	(62, 372)	(70, 195)	(67, 84)
4	(56, 332)	(69, 241)	(83, 373)
5	(59, 386)	(70, 195)	(72, 254)
6	(72, 497)	(61, 622)	(70, 556)
7	(74, 170)	(53, 277)	(86, 25)
8	(48, 702)	(69, 241)	(98, 338)
9	(59, 386)	(61, 129)	(100, 364)
10	(72, 497)	(56, 474)	(90, 730)
11	(59, 365)	(59, 386)	(105, 382)
12	(61, 622)	(61, 622)	(90, 730)
13	(61, 129)	(69, 510)	(72, 497)
14	(70, 556)	(56, 419)	(86, 726)
15	(67, 84)	(69, 241)	(66, 199)

Таблица 11.9

## Варианты численных значений тайного ключа

Параметр	Вариант														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
d	41	27	25	12	29	44	32	34	16	18	19	51	20	43	50

**Задание 3** (рассчитано на 2 часа аудиторных занятий).

3.1. Создать оконное приложение для генерации/верификации ЭЦП на основе алгоритма ECDSA: ЭК  $E_{751}(-1, 1)$  с генерирующей точкой  $G = (416, 55)$ ; порядок точки  $q = 13$ . Дополнительные параметры – в соответствии с вариантом из табл. 11.8 и 11.10.

3.2. Вычислить самостоятельно значение открытого ключа  $Q$ . При этом следует воспользоваться основной формулой (11.8), а также соотношениями (11.3)–(11.5) для случая  $P = Q$ ; не следует также забывать, что все вычисления производятся по  $\text{mod } 751$ ; см. также пример 5 (вычисление  $2P$ ) и пример 7.

Параметры  $k$  – по собственному усмотрению.

3.3. Хешем подписываемого сообщения ( $H(M)$ ) является модуль по основанию 13 координаты  $x$  точки ЭК, соответствующей первому символу собственной фамилии из табл. 11.10. Например, фамилия начинается на букву «Я»:  $x = 227$ , тогда  $227 \text{ mod } 13 = 6$ , значит, в данном конкретном случае  $H(M) = 6$ .

Таблица 11.10  
Координаты точек ЭК, соответствующие символам алфавита

А	(189, 297)	Р	(206, 106)
Б	(189, 454)	С	(206, 645)
В	(192, 32)	Т	(209, 82)
Г	(192, 719)	У	(209, 669)
Д	(194, 205)	Ф	(210, 31)
Е	(194, 546)	Х	(210, 720)
Ж	(197, 145)	Ц	(215, 247)
З	(197, 606)	Ч	(215, 504)
И	(198, 224)	Ш	(218, 150)
Й	(198, 527)	Щ	(218, 601)
К	(200, 30)	ъ	(221, 138)
Л	(200, 721)	ы	(221, 613)
М	(203, 324)	ь	(226, 9)
Н	(203, 427)	э	(226, 742)
О	(205, 372)	ю	(227, 299)
П	(205, 379)	я	(227, 452)

Таблица 11.11  
Варианты численных значений тайного ключа

Параметр	Вариант														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
d	3	12	7	4	10	5	6	9	4	10	12	11	5	9	7

Отчет по каждой части задания выполняется отдельно по установленной форме.

Отчет содержит краткие теоретические сведения, описание разработанного приложения, результаты использования приложения в соответствии с целью работы, анализ результатов.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ ЗНАНИЙ**

1. Дать определение эллиптической кривой.
2. Записать уравнение ЭК над вещественными числами (ЭК в криптографии, ECC).
3. Объяснить и показать на примере правила выполнения основных операций над точками ЭК.
4. Что такое «рациональная точка»?

5. Как производится умножение точки ЭК?
6. Как производится умножение точки  $P$  на число  $k$ , если  $k$  принимает значение: 2, 5, 11, 20, 32, 100, 256, 751, 1024?
7. Составить алгоритм многократного сложения точки ЭК (умножения точки на число) на основе примера 7.
8. Привести расчеты для точки  $Q$  при известных  $d$  и  $G$  из примера 7.
9. Есть ли отличия в применении операций над точками ЭК над конечными полями и над действительными числами?
10. Записать уравнение ЭК при формальном ее представлении в следующем виде:  $E_p(a, b)$ .
11. Из какого числа точек состоит ЭК  $E_{11}(6, -9)$ ? Дать их координаты.
12. Найти все точки ЭК  $E_{11}(1, 2)$ .
13. На чем основана криптостойкость систем на основе ЭК? Области применения ЭК в криптографии.
14. Что такое «порядок точки» ЭК? Показать на примере. Какую роль этот параметр играет в криптографии на основе ЭК?
15. Что такое «базовая точка» ЭК? Какую роль этот параметр играет в криптографии на основе ЭК?
16. Объяснить порядок формирования ключевой информации на основе ЭК.
17. Сгенерировать ключевую информацию на основе кривой  $E_{11}(1, 2)$ .

## **Лабораторная работа № 12**

### **ИССЛЕДОВАНИЕ СТЕГАНОГРАФИЧЕСКОГО МЕТОДА НА ОСНОВЕ ПРЕОБРАЗОВАНИЯ НАИМЕНЕЕ ЗНАЧАЩИХ БИТОВ**

**Цель:** изучение стеганографического метода встраивания<sup>\*</sup>/извлечения тайной информации с использованием электронного файла-контейнера на основе преобразования наименее значащих битов (НЗБ), приобретение практических навыков программной реализации данного метода.

**Задачи:**

1. Закрепить теоретические знания из области стеганографического преобразования информации, моделирования стеганосистем, классификации и сущности методов цифровой стеганографии.
2. Изучить алгоритм встраивания/извлечения тайной информации на основе метода НЗБ (LSB – Least Significant Bit), получить опыт практической реализации метода.
3. Разработать приложение для реализации алгоритма встраивания/извлечения тайной информации с использованием электронного файла-контейнера на основе метода НЗБ.
4. Познакомиться с методиками оценки стеганографической стойкости метода НЗБ.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

#### **12.1. Теоретические сведения**

##### **12.1.1. Основные определения, классификация и сущность стеганографических методов**

Сведения, необходимые для понимания сущности вопросов, относящихся к предметной области, в достаточном объеме изложены в [3] (гл. 7); полезно также ознакомиться с содержанием книги [53].

---

<sup>\*</sup> Встраивание (англ. embedding – встраивание, осаждение, внедрение).

Здесь мы остановимся на основных понятиях, которыми будем далее оперировать.

**Определение 1.** Стеганографическая система (*stegosystem*, **стегосистема** или **стеганосистема** – в русскоязычной тематической литературе используются оба сокращения) – совокупность средств и методов, которые используются для формирования скрытого канала передачи (или хранения) информации.

При этом скрытый канал организуется на базе и внутри открытого канала с использованием особенностей восприятия информации. «Скрытость» канала передачи тайной информации отличает стеганографию от криптографии: в первом случае тайной является сам факт наличия канала (передачи информации).

**Определение 2.** Абстрактно стеганографическая система обычно определяется как некоторое множество отображений одного пространства (множества возможных сообщений  $M$ ) в другое пространство (множество возможных стеганосообщений  $S$ ), и наоборот.

Основные компоненты стеганосистемы:

- контейнер  $C$  (файл-контейнер или электронный документ произвольного формата), в котором размещается (осаждается, скрывается) тайное сообщение  $M$ ; именно контейнер является упомянутым скрытым каналом;
- тайное сообщение  $M$ , осаждающееся в контейнер для передачи или хранения (например, с целью доказательства или защиты авторских прав на документ-контейнер [2, 53–56]; здесь речь может идти о невидимых цифровых водяных знаках (ЦВЗ));
- ключи, или ключевая информация,  $K$  системы, выполняющие ту же функцию, что и криптографические ключи; ключей может быть несколько, в соответствии с этим современные стеганосистемы характеризуют как **многоключевые**: один ключ отождествляется с методом встраивания/извлечения тайной информации, другой – с выбором элементов (например, битов) контейнера для его модификации при осаждении тайной информации, третий – для предварительного (перед встраиванием) преобразования тайной информации (например, на основе помехоустойчивого кодирования, сжатия или зашифрования) и т. д. [2, 54, 57];
- контейнер со встроенным сообщением, или стеганоконтейнер,  $S$ , который передается по открытому каналу, также являющемуся важным компонентом анализируемой системы; стеганоконтейнер будем именовать также **стеганосообщением**;

- для полноты упомянем также субъектов системы: *отправителя и получателя*.

В зависимости от формата документа-контейнера *цифровую* (или *компьютерную*) стеганографию подразделяют на классы [2, 52, 58–63]:

- аудиостеганография;
- видеостеганография;
- графическая стеганография;
- текстовая стеганография;
- и др.

**Определение 3.** Стеганографической системой  $\Sigma$  будем называть совокупность сообщений  $M$ , контейнеров  $C$ , ключей  $K$ , стеганосообщений (заполненных контейнеров)  $S$  и преобразований (прямого  $F$  и обратного  $F^{-1}$ ), которые их связывают:

$$\Sigma = (M, C, K, S, F, F^{-1}).$$

Как видим, сущностью рассматриваемой системы является тайное хранение или передача одной информации в другой информации, которая является открытой.

Таким образом, при построении стеганосистемы должны учитываться следующие основные положения:

- свойства контейнера должны быть модифицированы так, чтобы изменение невозможно было выявить при визуальном контроле; это требование определяет качество сокрытия внедряемого сообщения: для обеспечения беспрепятственного прохождения стеганосообщения по каналу связи оно никоим образом не должно привлечь внимание атакующего;

- противник (интруз) имеет полное представление о стеганографической системе и деталях ее реализации; единственной информацией, которая остается ему неизвестной, является ключ, с помощью которого только его держатель может установить факт присутствия и содержание скрытого сообщения;

- если противник каким-то образом узнает о факте существования скрытого сообщения, это не должно позволить ему извлечь подобные сообщения до тех пор, пока ключ хранится в тайне;

- потенциальный противник должен быть лишен каких-либо технических и иных преимуществ в распознавании или раскрытии содержания тайных сообщений.

Информацию об основных видах атак на стеганосистемы можно найти, например, в [3].

### 12.1.2. Метод НЗБ и особенности его реализации

Большинство исследований в предметной области посвящено использованию в качестве стеганоконтейнеров изображений (текст также можно рассматривать как изображение). Это обусловлено следующими причинами:

- относительно большим объемом цифрового представления изображений, что позволяет внедрять большой объем данных;
- заранее известным размером контейнера, отсутствием ограничений, накладываемых требованиями реального времени;
- наличием в большинстве реальных изображений текстурных областей, имеющих шумовую структуру и хорошо подходящих для встраивания информации;
- слабой чувствительностью человеческого глаза к незначительным изменениям цветов изображения, его яркости, контрастности, содержанию в нем шума, искажениям вблизи контуров;
- хорошо разработанными в последнее время методами цифровой обработки изображений.

Метод НЗБ основывается на ограниченных способностях зрения или слуха человека, вследствие чего людям тяжело различать незначительные вариации цвета или звука. Рассмотрим это на примере 24-битного растрового RGB-изображения. Как известно, каждая точка кодируется тремя байтами. Каждый байт определяет интенсивность красного (Red), зеленого (Green) и синего (Blue) цветов. Совокупность интенсивностей цвета в каждом из трех каналов определяет оттенок пикселя.

Представим пиксель тремя байтами в битовом виде, как это показано на рис. 12.1.

1	1	0	0	0	0	1	1	R — 195
0	0	1	0	0	0	0	0	G — 32
0	1	1	0	0	1	1	0	B — 102
1	1	0	0	0	0	1	0	R — 194
0	0	1	0	0	0	1	1	G — 35
0	1	1	0	0	1	0	0	B — 100

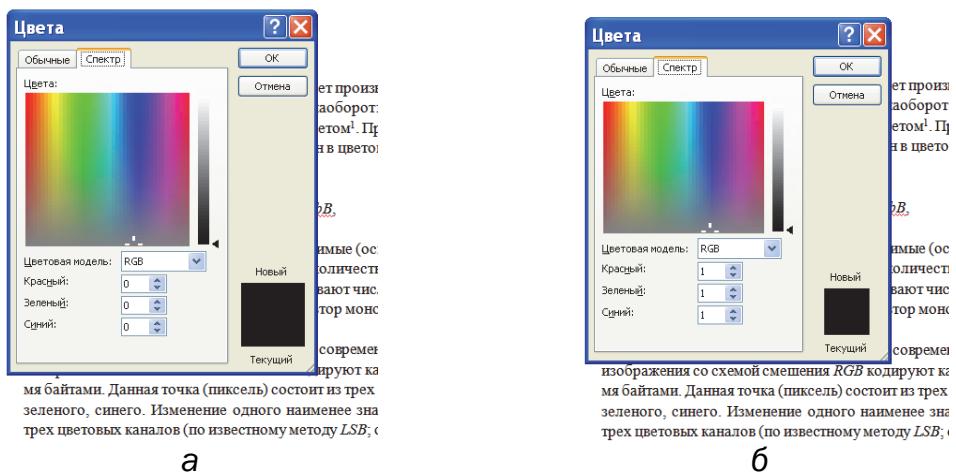
**Рис. 12.1.** Пример, показывающий принцип реализации метода LSB

Младшие биты (выделены бледным, справа) дают незначительный «вклад» в изображение по сравнению со старшими.

Замена одного или даже нескольких младших битов для человеческого глаза будет почти незаметна, поскольку реально человек может различать около полторы сотни цветовых оттенков.

Рассмотрим простейший пример.

**Пример 1.** Контейнером  $C$  выступает обычная буква «А». Цвет текста-контейнера «А» – черный: данный цвет представлен в MS Office Word как (00000000, 00000000, 00000000), т. е. (0, 0, 0) (см. рис. 12.2, а).



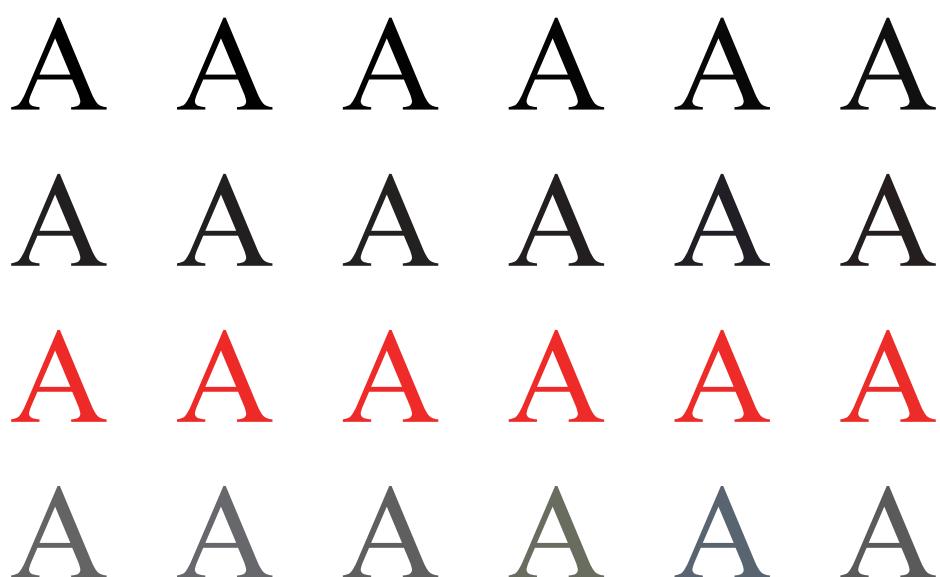
**Рис. 12.2.** Диалоговое окно MS Word  
с указанием цветовых координат символов текста:  
а – (0, 0, 0); б – (1, 1, 1)

Необходимо внедрить секретное сообщение  $M = 111$  в текст-контейнер  $C$ , используя текстовый процессор MS Word. Мы решаем задачу чисто механически: изменяем младший из символов цветового кода в каждом канале (рис. 12.2), т. е. в десятичном виде это можно представить как (1, 1, 1), а в двоичном – (00000001, 00000001, 00000001). Результат осаждения секретного сообщения (111) в текст-контейнер «А» показан на рис. 12.3: «пустой» контейнер ( $C$ ) никак визуально не отличается от стеганоконтейнера ( $S$ ).

На рис. 12.4 приведены четыре строки символов, цветовые координаты которых соответствуют различным числам с изменением вплоть до пятого (справа налево) бита цветового кода (см. таблицу). В таблице номера столбцов с кодами соответствуют позиции символа в строке на рис. 12.4. Цветовые оттенки символов (в строках) на рис. 12.4 едва различимы, притом только в четвертой строке.



**Рис. 12.3.** Пример практической реализации метода НЗБ:  
а – не модифицированный символ-контейнер; б – модифицированный  
символ-контейнер (со встроенным секретным сообщением «111»)



**Рис. 12.4.** Одинаковый символ (A) с различной кодировкой цвета

Подчеркнем, что именно визуальный анализ графического объекта является основой наиболее часто используемой (прежде всего в силу трудозатрат) методики стеганографического анализа.

#### Цветовая кодировка символов на рис. 12.4

Строка	Столбец					
	1	2	3	4	5	6
1	0, 0, 0	1, 1, 1	2, 2, 2	4, 4, 4	8, 8, 8	16, 16, 16
2	0, 1, 2	0, 2, 1	0, 2, 0	4, 2, 1	4, 8, 16	16, 8, 4
3	255, 0, 0	255, 1, 1	254, 1, 1	253, 2, 2	252, 4, 4	251, 8, 8
4	100, 100, 100	99, 100, 101	95, 95, 95	105, 110, 90	90, 100, 110	90, 90, 90

При этом проявляется еще одно важное обстоятельство: примерно в 50% случаев бит, который мы хотим записать, и бит в изображении-контейнере будут совпадать, и изменять ничего не нужно.

Понятно, что графические контейнеры в реальной стеганографии намного сложнее рассмотренных примеров.

Одним из простейших и понятных для решения наших задач является формат BMP (BitMaP) – одна из форм представления растровой графики. Изображение представляется в виде матрицы пикселей, где каждая точка характеризуется тремя параметрами:  $x$ -координатой,  $y$ -координатой и цветовым кодом на основе RGB-модели. Все операции графического ввода-вывода на экран монитора (принтер и на некоторые другие устройства) в конечном итоге осуществляются в этом формате. Для работы с этим форматом в ОС Windows предусмотрено много специальных функций и структур API, которые помогают производить все необходимые операции на достаточно высоком логическом уровне.

Контейнеры на основе BMP-формата разделяют на два класса: чистые и зашумленные. В первых прослеживается связь между младшими и остальными битами элементов цвета, а также видна зависимость самих младших битов между собой. Осаждение сообщения в такой контейнер нарушает такие зависимости, что легко выявляется аналитиком. Если же картинка зашумлена (например, получена со сканера или фотокамеры), то определить встроенное сообщение сложнее. Таким образом, в качестве файлов-контейнеров для метода НЗБ рекомендуется использовать файлы, которые не были созданы на компьютере изначально.

Другим из растровых форматов используемых в стеганографии контейнеров является формат PNG (Portable Network Graphics). По качеству цветового отображения данный формат превосходит JPEG (Joint Photographic Experts Group) и GIF (Graphics Interchange Format), но размер файла будет на 30–40% больше.

Вышеприведенные табличные и иллюстративные данные, а также опыт специалистов показывают, что при модификации даже 3–4 младших разрядов состояние графического стеганоконтейнера у экспертов подозрений не вызывает при визуальном его контроле.

Исходя из такой оценки, следует соотносить объем встроенного сообщения  $V_M$  с объемом  $V_C$  используемого контейнера. Например, если размер изображения  $500 \times 500 = 250\,000$  пикселей, то с учетом используемой 3-цветовой модели имеем  $750\,000$  единиц цветовых

координат. Если мы планируем модифицировать только самые младшие биты всех цветовых каналов матрицы, то максимальный объем осаждаемого сообщения ( $V_{M\max}$ ) не должен превышать 750 000 битов.

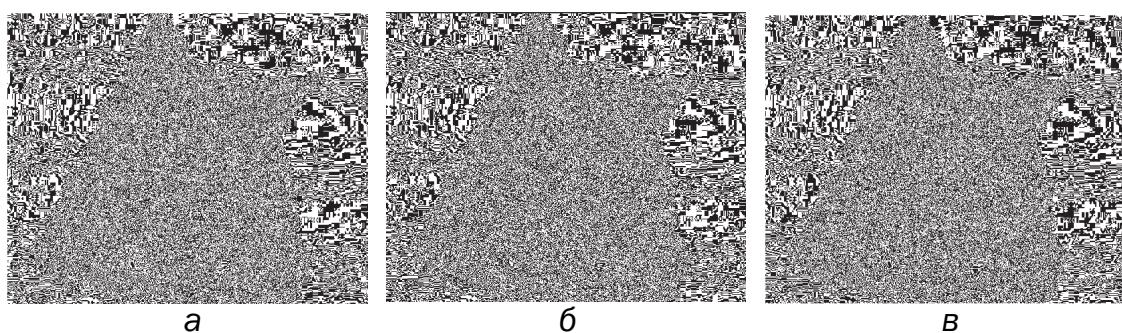
Посмотрим далее на некоторые технические детали и особенности реализации метода НЗБ при использовании в качестве контейнера изображения в формате PNG.

Незаполненный контейнер имеет вид, показанный на рис. 12.5.

Далее возьмем самый младший бит в каждом цветовом канале и отобразим на этом основании «самый нижний слой» исходного изображения в черно-белых (иначе нельзя) оттенках: нулевое значение младшего бита соответствует белому цвету, единичное – черному. Полученные мозаики показаны на рис. 12.6: а) соответствует красному цветовому каналу; б) зеленому; в) синему. Даже внимательный сравнительный анализ трех картинок показывает практически полную их идентичность.



**Рис. 12.5.** Вид «пустого» контейнера  
(источник: <https://habr.com/ru/post/422593/>)



**Рис. 12.6.** Черно-белое отображение младших разрядов «пустого» контейнера в красном канале (а), в зеленом канале (б), в синем канале (в)

Теперь наложим изображения на рис. 12.6 со следующими кодовыми параметрами пикселей в каждом канале: 1 – 255 (11111111 – в бинарном коде; т. е. классический красный, либо классический зеленый, либо классический синий), 0 – 0 (00000000 – в бинарном коде). Результат показан на рис. 12.7.



**Рис. 12.7.** Цветовое отображение младших разрядов «пустого» контейнера в красном

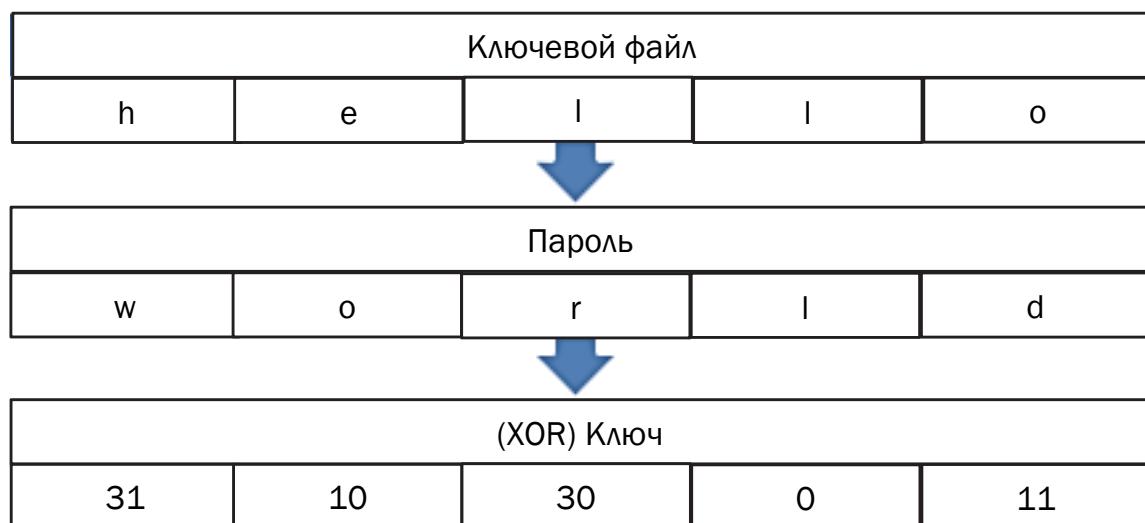
Именно последняя картинка может отождествляться с самым младшим или наименее значащим битом исходного, т. е. пустого контейнера. Такое изображение для стеганоконтейнера может служить основой для выполнения операций стеганоанализа. Для того чтобы этот процесс затруднить, сообщение осаждается в цветовые каналы пикселов не в регулярной, а в псевдослучайной последовательности. Такая последовательность должна рассматриваться как один из элементов ключа стеганосистемы.

Алгоритм реализации, как мы видим из примеров, достаточно прост. Прежде всего нужно определиться с содержанием сообщения  $M$ , а далее выбрать контейнер с учетом наших вышеприведенных оценок. В задачах по защите прав интеллектуальной собственности нужно идти от обратного, т. е. прежде всего иметь в виду готовый контейнер. В обоих случаях нужно сопоставлять объемы контейнера и осаждаемого сообщения.

Для затруднения стеганоанализа порядок и количество встроенных битов в различные цветовые каналы можно подчинить различным правилам. В качестве аналога можно использовать подход,

реализованный в [65]. Для операций «размазывания» сообщения по контейнеру могут применяться ключевой файл и пароль в виде текстов, которые символы которых заменяются числами и в совокупности определяют местоположение пикселя для записи в него части секретного сообщения. Простейшая реализация этого подхода показана на рис. 12.8.

Обычно для выполнения операций стеганографического анализа применяется метод « $\chi$ -квадрат». Особенности практического применения такого и других методов (атак) можно найти в [66].



**Рис. 12.8.** Пояснение к расчету местоположения пикселей для встраивания/извлечения сообщения

## 12.2. Практическое задание

1. Перед выполнением основного задания целесообразно познакомиться со структурой, интерфейсом и функциональными особенностями доступных и заслуживающих внимания приложений, в которых реализован метод НЗБ. К ним можно отнести следующие:

- *openstego* (<https://github.com/syvaidya/openstego>) – может применяться не только для осаждения данных, но и для ЦВЗ; использует *RandomLSB* – псевдослучайный принцип осаждения; поддерживает шифрование (дополнительный ключ), имеет также GUI; осаждение реализуется командой *openstegoembed-mfsecret.txt*

*-cfccover.png -ppassword -sfstego.png*, извлечение – *openstegoextract -sfopenstego.png -abcd -xfoutput.txt*; как видим, работает с PNG-контейнерами;

- *Stegano* (<https://github.com/cedricbonhomme/Stegano>) – работает не только с классическим LSB; имеет гибкую настройку, может также использоваться как модуль Python; осаждение реализуется командой *stegano-lsbhide --inputcover.jpg-fsecret.txt -eUTF-8 --outputstego.png*, извлечение – *stegano-lsbreveal -istego.png -eUTF-8 -ooutput.txt*; работает также с PNG-контейнерами;

- *LSB-Steganography* (<https://github.com/RobinDavid/LSB-Steganography>) – приложение написано на Python; работает с PNG- и BMP-контейнерами.

2. Разработать собственное приложение, в котором должен быть реализован метод НЗБ. При этом:

- выбор файла-контейнера – по согласованию с преподавателем;

- реализовать два варианта осаждаемого/извлекаемого сообщения:

- собственные фамилия, имя и отчество;

- текстовая часть отчета по одной из выполненных лабораторных работ;

- реализовать два метода (на собственный выбор) размещения битового потока осаждаемого сообщения по содержимому контейнера;

- сформировать цветовые матрицы (по аналогии с рис. 12.7), отображающие каждый задействованный для осаждения уровень младших значащих битов контейнера;

- выполнить визуальный анализ (с привлечением коллег в качестве экспертов) стеганоконтейнеров с различным внутренним содержанием; сделать выводы на основе выполненного анализа.

3. Результаты выполнения работы оформить в виде отчета по установленным правилам.

## **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. Охарактеризовать цели, задачи и области применения стеганографии.

2. В чем состоят сходства и различия между стеганографией и криптографией?

3. Дать определение стеганографической системы. Охарактеризовать составные части стеганосистемы и их взаимосвязь.
4. Основные классификационные критерии методов стеганографии.
5. Пояснить сущность основных атак на стеганосистемы.
6. Изобразить структурную схему стеганографической системы.
7. Сущность метода НЗБ. Области его применения.
8. Изобразить алгоритмы встраивания и извлечения сообщений на основе метода НЗБ при передаче этих сообщений.
9. Изобразить алгоритмы встраивания и извлечения сообщений на основе метода НЗБ при решении задачи защиты прав интеллектуальной собственности на электронный контент.

## **Лабораторная работа № 13**

### **ИССЛЕДОВАНИЕ МЕТОДОВ ТЕКСТОВОЙ СТЕГАНОГРАФИИ**

**Цель:** изучение стеганографических методов встраивания/извлечения тайной информации с использованием электронного файла-контейнера текстового формата, приобретение практических навыков программной реализации методов (рассчитана на 4 часа аудиторных занятий: 2 часа – часть 1, 2 часа – часть 2).

**Задачи:**

1. Закрепить теоретические знания из области текстовой стеганографии, классификации, моделирования стеганосистем подобного вида и сущности основных методов.
2. Изучить основные алгоритмы встраивания/извлечения тайной информации на основе методов текстовой стеганографии, получить опыт практической реализации методов.
3. Разработать приложение для реализации алгоритмов встраивания/извлечения тайной информации на основе методов текстовой стеганографии.
4. Познакомиться с методиками оценки стеганографической стойкости методов.
5. Результаты выполнения лабораторной работы (отдельно по каждой из 2 частей) оформить в виде описания разработанного приложения (для части 2), методики выполнения экспериментов с использованием приложений и результатов экспериментов.

#### **13.1. Теоретические сведения**

##### **13.1.1. Классификация, сущность и основные особенности базовых методов текстовой стеганографии**

Мы отмечали, что к текстовой стеганографии относятся методы, предусматривающие использование в качестве контейнера файла-документа текстового типа.

Многообразие методов текстовой стеганографии подразделяется на *синтаксические* методы, которые не затрагивают семантику

текстового сообщения, и *лингвистические*, которые основаны на эквивалентной трансформации текстовых файлов-контейнеров, сохраняющей смысловое содержание текста, его семантику (см. [2, 52–56]).

Для понимания сущности некоторых из методов полезно познакомиться с важнейшими особенностями и параметрами использования стилей (в том числе пространственно-геометрическими параметрами шрифтов), на основе которых строится текстовый файл-контейнер. На рис. 13.1 показаны основные из параметров шрифта.



**Рис. 13.1.** Параметры шрифта (источник: <http://indians.ru/a-font-sizes.htm>)

К синтаксическим методам компьютерной стеганографии, которые характеризуются *сравнительно невысокой эффективностью* (с точки зрения объема встраиваемой информации), относятся следующие (такие методы мы отнесем к числу *базовых* синтаксических методов):

- изменение расстояния между строками электронного текста (*Line-Shift Coding*); называется методом *изменения межстрочных интервалов*; сущность заключается в том, что используется текст с различными межстрочными расстояниями: выделяется максимальное и минимальное расстояния между строками, позволяющее кодировать соответственно символы «1» и «0» осаждаемого сообщения;

- изменение расстояния между словами в одной строке электронного текста (*Word-Shift Coding*); суть метода состоит в том, что осаждение информации основано на модификации расстояния между словами текста-контейнера;
- изменение количества пробелов между словами (частный случай метода *Word-Shift Coding*); основан он на том, что, например, чередование одинарного пробела и двойного (xx\_xx\_xx) кодирует «1», переход же с двойного пробела на одинарный кодирует «0» (xx\_xx\_xx);
- на основе внесения специфических изменений в шрифты, т. е. начертания отдельных букв (*Feature Coding*); заключается в изменении написания отдельных букв используемого стандартного шрифта: визуально заметны различные образы, соответствующие буквам с верхними (например, *l*, *t*, *d*) или нижними (например, *a*, *g*) выносными элементами (см. рис. 13.1); например, букву «А» можно модифицировать, незначительно укорачивая длинную нижнюю часть буквы (рис. 13.2);



**Рис. 13.2.** Пример применения метода *Feature Coding*:  
а – пустой контейнер; б – заполненный контейнер

- изменение интервала табуляции; аналогичен вышеописанному методу изменения количества пробелов, только в этом случае меняется не количество пробелов, а соответственно расстояние между строками и интервал табуляции;
- *Null Chipper* (дословно – *несуществующий, нулевой лепет*); предполагает размещение тайной информации на установленных позициях слов или в определенных словах текста-контейнера, который, как правило, лишен логического смысла (как видно, действительно лепет);
- увеличение длины строки; предусматривает искусственное увеличение длины каждой строки за счет пробелов: например, нет пробела (определяется положением знака перехода на новую строку) – «0», один пробел – «1» (рис. 13.3);

- *использование регистра букв*; для обозначения бита секретного сообщения, представленного единицей, используется символ нижнего регистра, а нулем – верхнего (или наоборот);
- *использование невидимых символов*; знак «пробел» кодируется символом с кодом 32, но в тексте его можно заменить также символом, имеющим код 255 (или 0), который является «невидимым» и отображается как пробел.

```
как·видно,·действительно·лепет);¶  
как·видно,·действительно·лепет);¶
```

**Рис. 13.3.** Пример реализации метода увеличения длины строки

Рассмотренные базовые методы могут применяться независимо и совместно, сохраняют исходный смысл текста, а обеспечиваемые ими показатели плотности кодирования при совмещении складываются.

Еще одна важная особенность. Перечисленные методы работают успешно до тех пор, пока тексты представлены в коде ASCII.

**! Методы также легко применяются к любому тексту, независимо от его содержания, назначения и языка. Синтаксические системы стеганографии легко реализуются в программном коде, так как они полностью автоматические и не требуют вмешательства оператора. Однако синтаксические методы неустойчивы к форматированию текста (вспомним рабочность систем на основе ЦВЗ), и поэтому информация может быть потеряна при простом применении иного стиля форматирования текста-контейнера, скрывающего в себе стегособщение. К тому же с помощью синтаксических методов можно передать незначительное количество информации.**

Существуют также стеганографические методы, которые интерпретируют текст как двоичное изображение. Необходимо отметить, что данные методы нечувствительны к изменению масштаба документа, что обеспечивает им хорошую устойчивость к большинству искажений, которые могут иметь место при активных атаках.

К числу основных лингвистических методов относятся [2, 52]:

- *метод синонимов*; в качестве примера приведем подмножество синонимов: {«тайный», «секретный», «конфиденциальный», «доверительный»}. В приведенном подмножестве каждое слово имеет единственное одинаковое смысловое значение, что позволяет закодировать каждое слово своим уникальным кодом

(т. е. выполнить операцию осаждения), например, «доверительный» – 00, «конфиденциальный» – 01, «секретный» – 10, «тайный» – 11. Подобное кодирование позволяет выбирать одно из четырех слов (как видим, они для удобства расположены по алфавиту) в зависимости от двух битов секретного сообщения. Отметим, что при этом, независимо какое из четырех слов будет выбрано, семантика сообщения не изменится. Очевидно, что при этом количество символов, соответствующих одному из синонимов используемого подмножества, зависит от общего числа элементов в подмножестве. Кроме того, обеим сторонам стеганосистемы должен быть известен общий алгоритм кодирования, т. е. один из ключей системы. Следует отметить, что в каждом подмножестве синонимов их упорядочивание должно выполняться по одному и тому же алгоритму и у отправителя сообщения, и у его получателя. В случае наличия слов с несколькими смысловыми значениями подобное кодирование оказывается невозможным. Также невозможно кодирование, если один из синонимов состоит из двух (или более) разделенных пробелом слов;

- *метод переменной длины слова*; основан на том, что длина слов в сообщении зависит от содержания секретного сообщения и способа кодирования слов: обычно одно слово текста-контейнера определенной длины кодирует два бита информации из стеганособщения; например, слова текста длиной в 4 и 8 символов могут означать комбинацию битов «00», длиной в 5 и 9 – «01», 6 и 10 – «10», 7 и 11 букв – «11»; слова короче 4 и длиннее 11 букв можно вставлять где угодно для лексической и грамматической связки слов в предложении – программное приложение, которое декодирует принятое сообщение (извлекает сообщение из стеганоконтейнера), будет просто игнорировать их;

- *метод первой буквы* – программа-помощник в этом методе накладывает ограничение уже не на длину слова, а на первую (можно на вторую) букву; обычно одну и ту же комбинацию могут кодировать несколько букв, например, комбинацию «101» означают слова, начинающиеся с «А», «Г» или «Т»;

- *мимикрия*; мимикрия генерирует осмысленный текст, используя синтаксис, описанный в *Context Free Grammar* (CFG), и встраивает информацию, выбирая из CFG определенные фразы и слова; грамматика CFG – это один из способов описания языка, который состоит из статических слов и фраз языка, а также узлов.

### **13.1.2. Методы текстовой стеганографии на основе модификации цветовых и пространственно-геометрических параметров символов текста-контейнера**

Поддерживаемые форматы документов-контейнеров, которые мы хотим рассматривать как объекты для скрытия тайной информации – любые, способные хранить цвет и иные указанные параметры символов и которые можно открыть с помощью процессора MS Office Word: (\*.doc, \*.docx), \*.rtf (межплатформенный формат хранения размеченных текстовых документов), \*.odt (открытый формат документов для офисных приложений).

Что касается реализации метода на основе модификации цвета символов текста-контейнера [55, 56, 67, 68], по сути своей он схож с классическим методом наименее значащих битов (см. лабораторную работу № 12) и опирается на использовании цветовой модели RGB (см. рис. 12.2). Подробный алгоритм реализации метода можно найти, например, в [67].

К основным пространственно-геометрическим параметрам символов (и текста в целом), которые могут быть использованы как инструменты для стеганографического преобразования, относятся апрош (обозначен цифрами 7 и 8 на рис. 13.1) и кернинг.

#### **13.1.2.1. Метод на основе апроша**

Апрош определяет расстояние между соседними символами текста. Фактически апрош состоит из двух таких расстояний – полуапрошей, являющихся как бы пространством, прилегающим к каждому из символов-соседей (см. рис. 13.1 и 13.4). Мы далее будем обращаться только к апрошу. Согласно существующим техническим правилам набора нормальный апрош должен быть равен половине кегля (размера) шрифта.

Идея метода [69, 70] заключается в следующем. Встраивание сообщения в контейнер может быть основано на модификации базового (устанавливаемого текстовым процессором по умолчанию) значения апроша  $a_o$ , его изменением от базового до некоторого максимального  $a_{max}$  (или минимального  $a_{min}$ ), которое зрительно не должно отличаться от стандартного. Такое изменение производится с определенным шагом (дискретно)  $\Delta a_i$ , каждому значению которого присваивается определенный бит или определенная комбинация битов.



**Рис. 13.4.** Измерение апрова

Изменение величины апрова между двумя определенными символами текста относительно базового значения  $a_o$  на небольшое расстояние (пункты (пт) или доли пункта) формально можно представить в следующем виде:

$$a_t = a_o + \Delta a_t. \quad (13.1)$$

Такое изменение не должно вызывать визуально заметного уплотнения ( $\Delta a_t < 0$ ) или разрежения ( $\Delta a_t > 0$ ) групп символов. В текстовом процессоре MS Word апраш может принимать значения в диапазоне от 0 до 1584 пунктов.

Для примера и визуального представления об особенностях установки данного пространственно-геометрического параметра шрифта на рис. 13.5 приведена строка текста с различными параметрами апрова.

При использовании метода осаждение секретного сообщения  
 При использовании метода осаждение секретного сообщения

**Рис. 13.5.** Примеры использования различного апрова

На этом рисунке вторая строка оформлена с использованием стандартного (обычного) апрова. В первой строке применено уплотнение во всех словах, кроме первого: во втором слове – на 0,1 пт, в третьем – на 0,2 пт, в четвертом – на 0,3 пт, в пятом – на 0,4 пт, в шестом – на 0,5 пт; в третьей строке слова со второго по шестое оформлены с изменением  $\Delta a$  в противоположную сторону, т. е. с разрежением. В четвертой строке применялось уплотнение (отрицательный  $\Delta a$ ) или разрежение (положительный  $\Delta a$ ) лишь к отдельным символам первого («При») и второго («использовании») слов: для «П» –  $\Delta a = -0,1$  пт; «ри» –  $\Delta a = -0,2$ ; «и» –  $\Delta a = 0$ ;

«с» –  $\Delta a = 0,1$ ; «п» –  $\Delta a = 0$ ; «о» –  $\Delta a = -0,1$ ; «л» –  $\Delta a = -0,2$ ; «ъ» –  $\Delta a = -0,3$ ; «з» –  $\Delta a = 0,4$ ; «ов» –  $\Delta a = -0,3$ ; «ан» –  $\Delta a = 0,2$ ; «ии» –  $\Delta a = 0$ . Пятая строка целиком отформатирована при  $\Delta a = 1$  пт, а шестая – при  $\Delta a = -1$  пт.

Особенностью рассматриваемого метода является возможность одноразового размещения (в апроше одного символа) числа битов, определяемого дискретной разницей между минимальным и максимальным значениями  $\Delta a$ . Например, если отсчет вести от  $\Delta a_{\min}$  до установленного интервала  $\Delta a_t$  в виде параметра  $0,1n_d$  (пт), то количество условных дискретных единиц  $n_d$ , представленное в бинарном виде, определяет число битов, которые можно таким образом разместить; например,  $\Delta a_{\min} = -0,5$  пт, а  $\Delta a_t = 0,3$  пт. Разница между этими величинами составляет 0,8 пт:  $8 \cdot 0,1$  или  $n_d = 8$  (в двоичном виде – 1000; в первом приближении именно такую бинарную комбинацию можно разместить (осадить) путем модификации конкретного апроша). На этой основе могут быть разработаны различные варианты кодировки осаждаемых комбинаций.

### 13.1.2.2. Метод на основе кернинга

В текстовых документах встречаются такие сочетания знаков, которые образовывают визуальные «дыры» либо «сгущения». Например, в текстах на основе кириллицы – это такие сочетания: «ГА», «ТА», «АТА», «ЬТ» и т. п., на основе латиницы – «AY», «AV», «T;», «ff», а на основе греческого алфавита – «ΘΑ», «ΔΟ», «λκ» и др. Такие сочетания называются *кернинговыми парами*. Особенности «кернингования» приведены на рис. 13.6.

Под *кернингом* обычно понимается процесс изменения межсимвольного расстояния между отдельными парами символов или кернинговыми парами (именно фактор парности отличает кернинг от апроша).



**Рис. 13.6.** Пояснение к понятию кернинга



**Таким образом, технология кернинга, появившаяся в полиграфии после внедрения фотонабора (а затем и компьютерного набора), включает подбор межбуквенных интервалов для конкретных пар букв с целью улучшения внешнего вида и удобочитаемости текста. Такой избирательный подбор позволяет компенсировать неравномерности визуальной плотности текста, получаемой при использовании стандартных апрошей для каждой буквы.**

Очевидно, что промежуток между «А» и «В» в первой строке на рис. 13.6 гораздо больше, чем во втором, хотя формально они одинаковы. В данном случае сочетание «AV» как раз и является кернинговой парой. После применения кернинга визуальное восприятие текста улучшилось.

С появлением цифрового фотонабора стало возможным хранить такие критические сочетания знаков (кернинговые пары) для некоторого условного шрифта, общее число которых мы обозначим  $N_k$ , в памяти компьютера с указанием величины ( $\sigma$ ), на которую необходимо сдвинуть символы, чтобы визуально выровнять буквенные просветы.

Как правило, текстовые редакторы или процессоры содержат встроенные средства настройки кернинга, который определяет стандартный межбуквенный интервал для того или иного шрифта. При этом  $\sigma$  устанавливается в соответствии со значениями из таблицы кернинговых пар, встроенной в вышеуказанный файл со шрифтом. Такая настройка позволяет выровнять шрифт и является стандартной. В некоторых шрифтах сейчас количество пар доходит до нескольких тысяч.

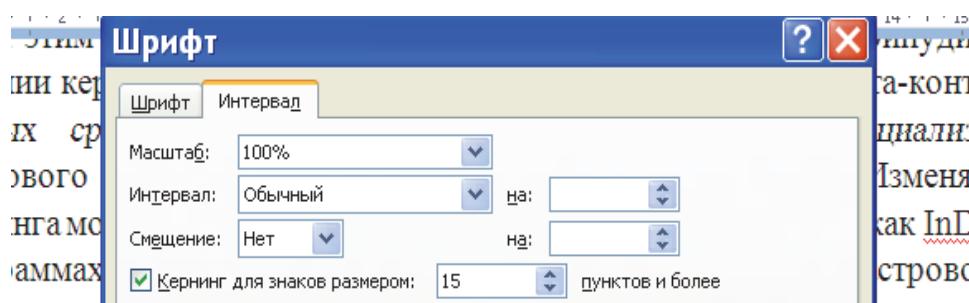
Значение кернинга может быть как положительным (когда знаки раздвигаются,  $\sigma > 0$ ), так и отрицательным (когда сдвигаются,  $\sigma < 0$ ). Эта величина в программах верстки устанавливается в процентах от ширины символа пробела используемого шрифта.

Следуя общепринятой методике, кернинг, как параметр, будем измерять в тысячных долях круглой шпации ( $Em$ ) – единицы изменения, которая определяется относительно текущего размера шрифта и равна ширине символа «М». Например, для шрифта размером 6 пунктов 1 круглая шпация равна 6 пунктам, а для шрифта размером 10 пунктов – 10 пунктам. Таким образом, размер кернинга  $\sigma$  строго пропорционален текущему размеру шрифта. Сдвиги букв относительно автоматически установленного межсимвольного расстояния (измеряемого, например, апрошем) можно производить

с различным шагом: от 0,01 до 0,04 величины  $E_m$ , в зависимости от нужной точности.

Как следует из анализа общих принципов задания размера кернинга и управления этим размером, предлагаемый метод основывается на принудительном применении кернинга, не зависящем от установок параметров текста-контейнера, созданных средствами текстового процессора или иного специализированного текстового редактора. Здесь есть одна важная особенность. Изменять значения кернинга можно лишь в программах верстки (например, InDesign) или в программах, предназначенных для работы с векторной или растровой графикой (например, CorelDraw, Photoshop). В текстовом процессоре MS Word значения кернинга автоматически установлены в таблицах кернинговых пар каждого шрифта, доступа к которым нет.

Здесь возможности пользователя практически связаны лишь с указанием минимального размера шрифта, для которого можно применять кернинг. Это означает, что если текст набран на основе шрифта размером, например, 14 пт, а мы в специальной опции (*Главная/Шрифт*) установили минимальный размер в 15 пт, при котором будет выполняться кернинг (см. поясняющую иллюстрацию на рис. 13.7), то для нашего текста эта процедура (иногда ее называют «кернингованием») не будет выполнена.



**Рис. 13.7.** Пример установки параметров для применения кернинга

Таким образом, рассматриваемый метод может быть интерпретирован так, что само значение кернинга мы программно не изменяем, а изменяем лишь размер символов, к которым будет применен кернинг в результате осаждения секретной информации.

Следует принять во внимание также следующую важную особенность. Текстовый процессор MS Word позволяет изменять параметры кернинга для знаков определенного размера. Размер символов, к которым может быть применим кернинг, должен быть

в диапазоне между 1 и 1638 пт; в частном случае этот размер может быть указан в последней строке окна на рис. 13.7 (а сам размер шрифта можно устанавливать с точностью до 0,5 пт).

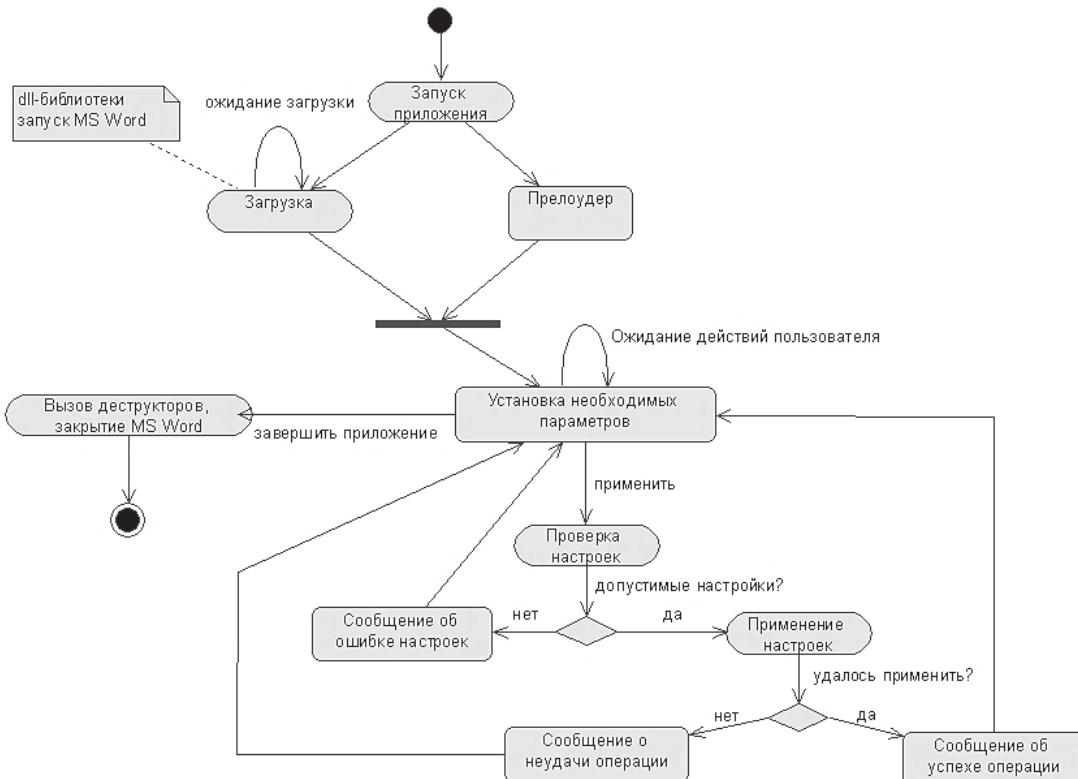
В [69] в общем виде даны алгоритмы встраивания/извлечения сообщений на основе кернинга.

Заметим, что выбор символов текста для встраивания сообщения на основе модификации цветовых и пространственно-геометрических параметров может выполняться на основе ранее описанных принципов в соответствии с одним из элементов ключевой информации:

- локальный разброс;
- случайный (псевдослучайный) разброс;
- случайный (псевдослучайный) разброс с памятью;
- подряд.

### 13.1.3. Особенности программной реализации методов

Общий принцип работы некоторого условного приложения проиллюстрирован на диаграмме деятельности (рис. 13.8).

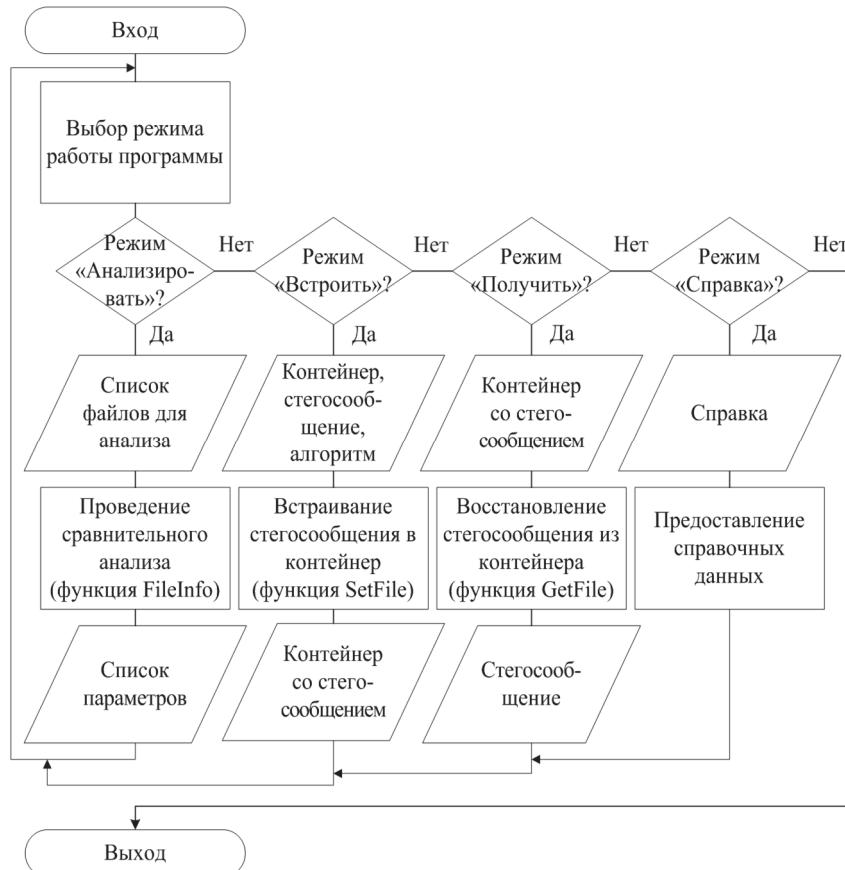


**Рис. 13.8.** Диаграмма деятельности программного средства, реализующего методы текстовой стеганографии на основе MS Word

Некоторые особенности программной реализации рассмотренных методов проанализируем на конкретном примере.

Пусть интерфейс программного средства включает кнопки «Анализировать», «Встроить», «Получить» и «Справка».

Пример общей блок-схемы алгоритма функционирования приложения показан на рис. 13.9.



**Рис. 13.9.** Пример общей блок-схемы функционирования программного средства, реализующего методы текстовой стеганографии

Вкладка «Анализировать» позволяет получить количественные характеристики методов изменения регистра буквы, изменения цвета символов, изменения масштаба символов, изменения апроша, изменения кернигга, а также методов *Line-Shift Coding*, *Word-Shift Coding*, *Feature Coding* при использовании в качестве контейнера выбранных текстовых файлов. По полученным результатам можно произвести сравнение эффективности использования конкретного метода по сравнению с другими.

Вкладка «Встроить» позволяет осуществить встраивание (осаждение) сообщения в контейнер по выбранному алгоритму.

По умолчанию каждый метод производит встраивание 1 бита файла-сообщения в соответствующем параметре каждого символа файла-контейнера. В [71] приведены коды основных функций анализируемого приложения, реализованные в C#.

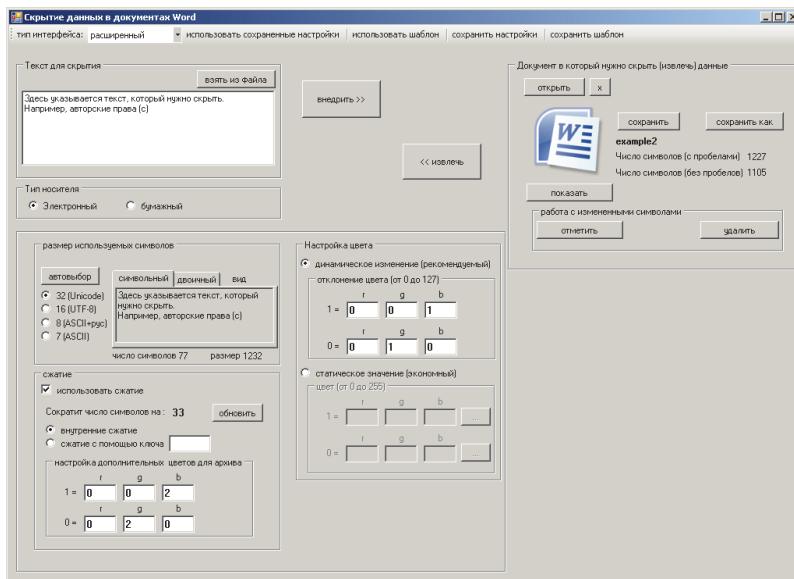
### 13.1.4. Краткое описание специализированных программных средств

Для выполнения основного задания в данной лабораторной работе целесообразно ознакомиться (в качестве примера) со структурой, интерфейсом, особенностями функционирования и программной реализации доступных инструментальных средств, реализующих методы текстовой стеганографии.

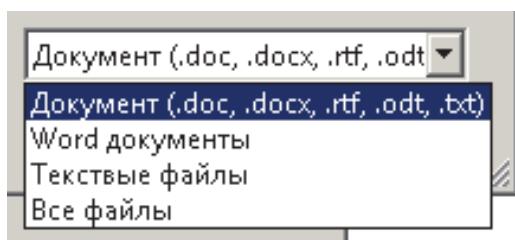
#### 13.1.4.1. Программное средство «Sword»

Функционал программы построен на основе диаграммы, показанной на рис. 13.8. Основное окно средства показано на рис. 13.10.

Как видно из рис. 13.10, интерфейс программы имеет несколько блоков установки. В блоке «Текст для скрытия» пользователь может ввести вручную с клавиатуры или вставить из буфера обмена секретное сообщение, которое необходимо скрыть в документе-контейнере. Имеется возможность использовать в качестве сообщения уже существующий электронный текстовый документ (кнопка «Взять из файла»). В блоке «Настройка шифра» можно выбрать способ сжатия и параметры шифрования. В правом блоке отображаются параметры текущего документа, включая количество символов в нем.



**Рис. 13.10.** Основное диалоговое окно программного средства «Sword»



**Рис. 13.11.** Выбор формата используемого документа

В области «*Тип носителя*» можно выбрать вид текстового документа-контейнера – электронный или бумажный. Разница заключается в том, что при выборе типа «бумажный» информация будет встраиваться лишь в символы, тогда как в типе «электронный» осаждение будет осуществляться и в пробельные элементы.

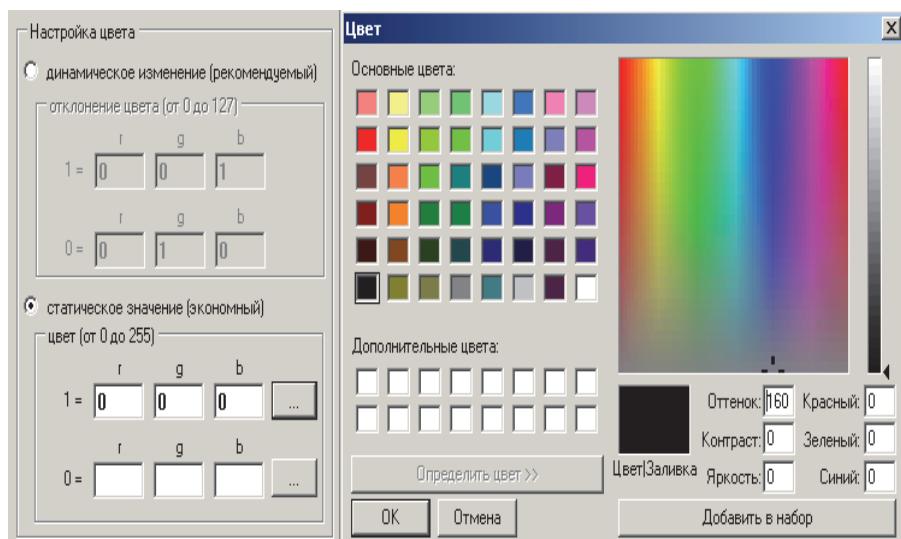
С помощью переключателей в блоке «*Размер используемых символов*» можно посмотреть, как выбранный текст будет выглядеть в двоичном виде в выбранной кодировке. В программе используется четыре основных вида кодировки: ASCII, русская таблица кодов ASCII, UTF-8 и Unicode. При смене кодировки исходный текст не теряется. Поэтому, если в одной кодировке символы поменялись, можно выбрать другую – и они восстановятся. Также в данном блоке отображаются сведения о количестве встраиваемых символов (параметр необходимо учитывать при выборе документа-контейнера), о необходимом минимальном размере файла, в который будет происходить осаждение сообщения.

Использование 16-ричной и 32-ричной систем представления обусловлено тем, что при такой кодировке стегосообщение можно скрыть в меньшем объеме текста. Например, нам необходимо зашифровать секретное сообщение «Слово» в текстовом документе. При этом число символов в документе, необходимых для скрытия, в двоичной системе составляет 79, а в 32-ричной – 15.

С помощью кнопки «*Настойки цвета*» можно выбрать значение отклонения исходного цвета символа, т. е., иначе говоря, в данной области выбирается значение соответствующей ключевой информации. Используемый ключ можно сохранить для последующего его использования при извлечении секретной информации. Эти файлы являются обычными текстовыми документами, в которых хранятся настройки, структурированные специальным способом. Для того чтобы отличить файлы с такими настройками от других текстовых документов, первые сохраняются с расширением \*.sword.

В данной области предусмотрено два типа изменения цвета: динамическое и статическое. При динамическом встраивании за основу берется один какой-либо символ (символ-образец с индексом  $t$ ), считывается его цвет. Необходимое секретное сообщение преобразуется в двоичный вид. Программа «распределяет» секретное сообщение внутри текста-контейнера в соответствии с ключом.

Если выбрать «Статическое изменение цвета», то для «0» и «1» необходимо указать конкретные значения трех цветовых каналов ( $R, G, B$ ) либо выбрать конкретный цвет на вкладке (рис. 13.12). Программа создает переменные, которым присваивает значения для цветовых каналов. Однако наиболее эффективным и наименее заметным является динамическое изменение цвета.



**Рис. 13.12.** Интерфейс для настройки цвета скрываемых символов в приложении *Sword*

И, наконец, в правой части главного окна можно выбрать электронный текстовый документ, в который необходимо произвести скрытие (контейнер). После осаждения сообщения в документ-контейнер можно просмотреть позиции, в которых скрыта информация. Соответствующие символы и пробельные элементы будут выделены синим маркером (рис. 13.13).

Особенности: атака может производить  
которых сами являются жертвами idsf;

**Рис. 13.13.** Фрагмент документа-контейнера с выделением модифицированных символов

После инициализации документа с его содержимым можно производить любые действия. При внедрении исследуется именно текущее содержание документа, а не то, которое было на момент открытия. После извлечения текст будет помещен в поле «*Текст для скрытия*».

С программным кодом средства можно познакомиться по ссылке [72].

#### 13.1.4.2. Программное средство «QuatesEmbed»

Здесь реализованы стеганографические методы для контейнеров DOCX-формата на основе модификации такого символа, как кавычка, которая может быть одинарной или двойной. Использование кода (0 или 1), соответствующего виду кавычки, позволяет осаждать информацию.

Программный код приложения можно найти по ссылке [73]. Для работы с приложением необходимо, чтобы на компьютере пользователя был установлен .NETFramework.

Для хранения информации о документе-контейнере при выполнении осаждения создан вспомогательный класс *XMLFile*, являющийся абстракцией над реальными документами *XML*, описание которого представлено в листинге 13.1.

```
class XMLFile
{
    public StringBuilder File { get; set; }

    public decimal GetContainerCapacity(int bitsPerSymbol)
    {
        int containerCapacity = 0;
        try
        {
            for (int i = 0; i < File.Length; i++)
            {
                if (File[i] == '\'' || File[i] == '\"')
                {
                    containerCapacity++;

                    if (File[i] == '\'')
                    {
                        while (File[++i] != '\'');
                    }
                    else if (File[i] == '\"")
                    {

```

```
                while (File[++i] != '\\'
            );
        }
    }
}
catch (IndexOutOfRangeException e)
{
    return containerCapacity;
}
return Math.Floor((decimal) containerCapacity / bitsPerSymbol;
}
```

**Листинг 13.1.** Описание класса *XMLFile*

Метод *GetContainerCapacity* позволяет получить информацию о размере загруженного в приложение контейнера. На основании подсчета количества пар двойных и одинарных кавычек в документе, учитывая язык внедряемого сообщения, он возвращает максимальное число символов, из которых может состоять внедряемое сообщение. Класс *XMLFile* имеет свойство *File*, содержащее текст считываемого XML-документа. Данное свойство имеет тип *StringBuilder*, позволяющее изменять содержимое свойства, не создавая при этом лишних объектов. Это позволяет обрабатывать большие файлы, при этом эффективно используя оперативную память.

Для манипуляции файлами создан класс *FileManager*, содержащий набор методов для перемещения, а также изменения содержимого документов. В листинге 13.2 представлены методы *ReadXMLFile* и *WriteXMLFile*, позволяющие считывать/изменять содержимое реального XML-файла и сохранять его состояние в объект класса *XMLFile*.

```
public static string ReadXMLFile(string path)
{
    return File.ReadAllText(path);
}

public static void WriteXMLFile(XMLFile XMLFile, string path)
{
    File.WriteAllText(path, XMLFile.File.ToString());
}
```

**Листинг 13.2.** Методы, изменяющие содержимое XML-файлов

Для того чтобы извлечь XML-документ из DOCX-файла, необходимо изменить расширение с DOCX на ZIP. Методы, реализующие смену расширения файла, представлены в листинге 13.3.

```
public static void WriteXMLFile(XMLFile XMLFile, string path)
{
    File.WriteAllText(path, XMLFile.File.ToString());
}

public static void CopyFileAndChangeExtentionToZip(string file)
{
    DeleteTempArchive(tempArchiveStorage);
    File.Copy(file, Path.ChangeExtention(tempArchive
        Storage, ".zip"));
}
```

**Листинг 13.3.** Методы, реализующие смену расширения файлов

Перед внедрением сообщения в XML-документ его необходимо конвертировать в бинарный вид при помощи метода *MakeBinaryString*.

После стеганографического преобразования измененный XML-документ помещается в архив при помощи метода *AddStegoContainerToArchive* класса *Embedder*. Для извлечения содержимого XML-документа из ZIP-архива реализован метод *ReadDocumentFromZipFile*. Метод принимает на вход путь, по которому можно обратиться к XML-документу, и, используя метод *Open* класса *ZipFile* из пространства имен *System.IO.Compression*, считывает текст XML-файла.

Для внедрения сообщения в XML-документ используется метод *EmbedMessage*. Представленный метод заменяет в документе двойные и одинарные кавычки в соответствии с бинарной последовательностью, полученной из осаждаемого сообщения. Данный метод модифицирует принимаемый XML-документ, который при помощи метода *AddStegoContainerToArchive* размещается в ZIP-архиве.

Извлечение бинарной последовательности производится при помощи метода *ExtractMessage*. На вход метод принимает объект класса *XMLFile*, в котором находится строковый объект, содержащий разметку XML-документа. Для оптимизации скорости работы алгоритма в качестве переменной, в которую последовательно записывается извлекаемое сообщение, используется класс *StringBuilder*.

После завершения считывания объект данного класса конвертируется в строку, с которой производятся дальнейшие операции согласно алгоритму извлечения. Представленный метод проходится последовательно по передаваемому на вход документу и считывает все кавычки. В процессе считывания одинарной кавычке ставится в соответствие двоичный 0, а двойной – двоичная 1. Считываемая бинарная последовательность сохраняется как переменная *ExtractMessage*.

Главное окно приложения показано на рис. 13.14. Назначение элементов интерфейса не требует пояснений.

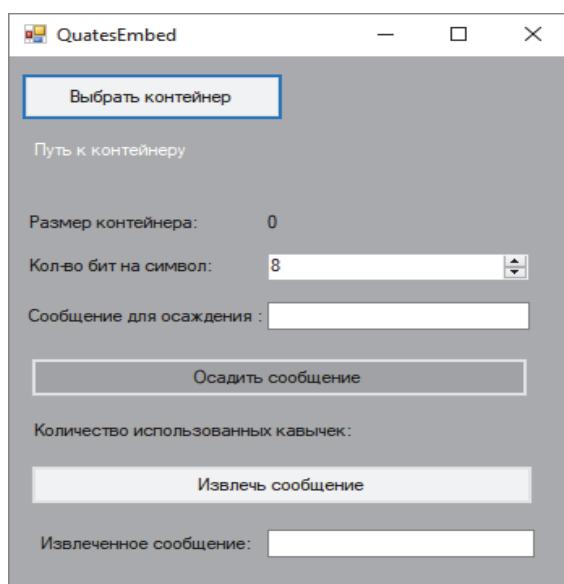


Рис. 13.14. Главное окно приложения «QuatesEmbed»

Как видно из данных на рис. 13.15, для размещения сообщения (information) «использовались» 88 пар кавычек.

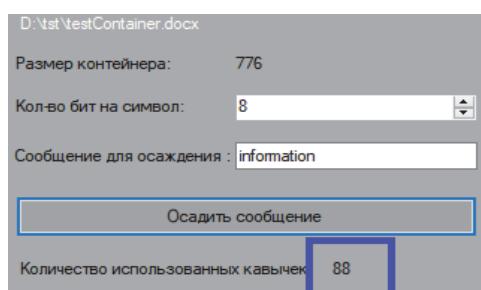


Рис. 13.15. Одно из информативных окон приложения

Мы рассмотрели некоторые особенности программной реализации стеганометодов и использования готовых средств для лучшего понимания сущности основного лабораторного задания и более осознанного его выполнения.

### 13.2. Практическое задание

1. Перед выполнением основного задания целесообразно познакомиться со структурой, интерфейсом и функциональными особенностями приложений «*Sword*» и «*QuatesEmbed*». Для этого необходимо: инициировать приложения, выбрав в указанном преподавателем каталоге соответствующий исполнительный (.exe) файл.

1.1. Используя программное средство «*Sword*», встроить (секретное) сообщение в текстовый документ-контейнер, который необходимо выбрать, и произвести операцию извлечения осажденного сообщения:

- выбрать ключ («*Настройка цвета*»); ключ определяет, на сколько единиц будет изменено значение цвета символа, т. е. его отклонение от исходного; значение отклонения задается для трех базовых цветов: красного (r), зеленого (g), синего (b);

- выбрать встраиваемое сообщение («*Текст для скрытия*»); сообщение можно вводить с клавиатуры либо загружать из файла; стеганосообщение (тайное сообщение) в различных системах счисления можно просмотреть в области «*Размер используемых символов*»);

- выполнить операцию встраивания (нажать кнопку «*Внедрить*»); после осаждения информации можно просмотреть используемые для этого символы контейнера, выделив их специальным маркером (кнопка «*Отметить*»));

- последовательно выделить каждый из модифицированных символов и проанализировать его цветовые характеристики (используйте кнопку «*Цвет текста*»); определить визуально, какое максимальное отклонение цвета от исходного является незаметным;

- извлечь секретное сообщение из стеганоконтейнера, используя известный ключ.

Указанные в п. 1.1 операции выполнить:

- для различных типов кодировки;
- для различных типов носителя;
- для различных режимов настройки цвета.

Оценить эффективность различных режимов встраивания (отношение максимального объема осаждаемой информации к объему контейнера, при которых визуальные изменения контейнера практически отсутствуют).

1.2. Используя программное средство «*QuatesEmbed*», встроить (секретное) сообщение в документ-контейнер, который необходимо выбрать, и произвести операцию извлечения осажденного сообщения:

- для выбора контейнера необходимо нажать кнопку «*Выбрать контейнер*» на главной странице приложения – появится окно, позволяющее указать файл, хранящийся на жестком диске пользователя (рис. 13.16);

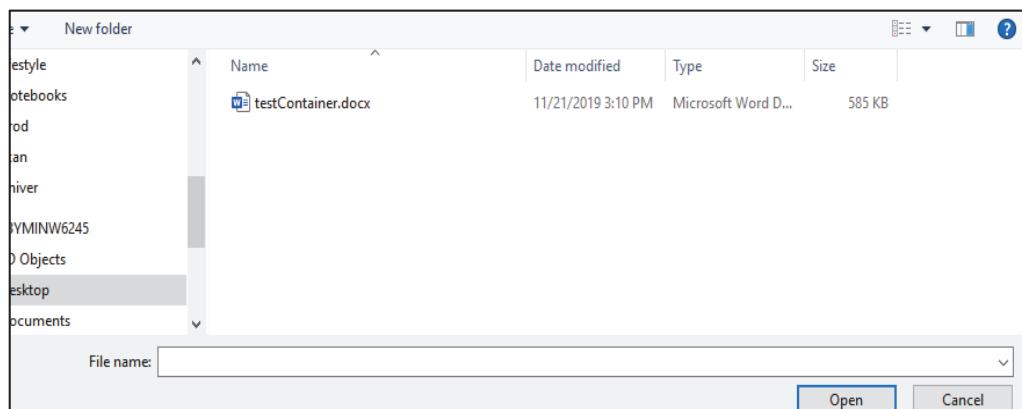


Рис. 13.16. Модальное окно выбора контейнера

- нажать на кнопку «*Open*», после чего файл откроется в приложении; приложение анализирует файл-контейнер на предмет возможности внедрения информации (путем подсчета кавычек в XML-документе); после этого информация о размере контейнера отображается в окне приложения (пример на рис. 13.17);

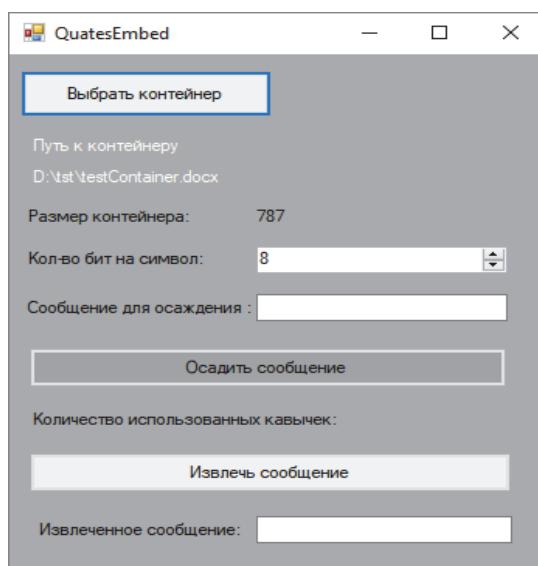


Рис. 13.17. Результаты анализа пустого контейнера

- указать количество битов, отводимое для осаждения одного символа сообщения (на рис. 13.17 указано 8 битов); отметим, что если внедряемое сообщение – на русском языке, то на один символ сообщения будет отводиться 11 битов в контейнере, в то время как для сообщения на английском языке можно использовать всего лишь 8 битов;
- указать внедряемое сообщение;
- выполнить операцию внедрения; после выполнения всех операций при помощи всплывающих окон (после сохранения стеганоконтейнера) пользователю будет доступной информация о том, сколько пар кавычек было использовано в процессе внедрения сообщения.

Для осуществления процедуры извлечения информации из стеганоконтейнера необходимо:

- установить количество битов в поле «*Кол-во бит на символ*», равное значению, которое использовалось для внедрения сообщения;
- нажать на кнопку «*Извлечь сообщение*» и выбрать стеганоконтейнер.

Выполнить указанные в п. 1.2 операции над контейнерами различного объема, содержащими разнотипную информацию. Оценить достигаемый в каждом случае эффект.

Сравнить эффективность методов, реализованных в обоих программных средствах, которые исследовались в части 1 практического задания.

2. Разработать авторское приложение, реализующее один из методов текстовой стеганографии на основе модификации пространственно-геометрических параметров текста-контейнера. Варианты заданий приведены в таблице. Дополнительные параметры согласуются с преподавателем.

### **Варианты заданий**

Вариант	Реализуемые методы
1	Модификация расстояния между строками; модификация апроша
2	Модификация расстояния между словами; модификация цвета
3	Модификация числа пробелов; модификация кернинга
4	Изменение длины строки; модификация апроша
5	Модификация расстояния между строками; модификация цвета
6	Модификация расстояния между словами; модификация апроша
7	Модификация числа пробелов; модификация апроша
8	Модификация числа пробелов; модификация цвета

Окончание таблицы

Вариант	Реализуемые методы
9	Модификация числа пробелов; модификация кернинга
10	Модификация расстояния между строками; модификация кернинга
11	Изменение длины строки; модификация кернинга
12	Изменение длины строки; модификация цвета
13	Модификация расстояния между словами; модификация кернинга
14	Метод переменной длины слов; модификация апроша
15	Метод переменной длины слов; модификация цвета
16	Метод переменной длины слов; модификация кернинга

Результаты выполнения каждой части практического задания оформить в виде отчета по установленной форме.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. В чем состоит сущность методов текстовой стеганографии?
2. Охарактеризовать методы синтаксической текстовой стеганографии. Привести примеры конкретной реализации методов.
3. Охарактеризовать методы лингвистической текстовой стеганографии. Привести примеры конкретной реализации методов.
4. Дать оценку стеганографической стойкости методов текстовой стеганографии при конвертации текста-контейнера в иной текстовый формат.
5. Дать оценку стеганографической стойкости методов текстовой стеганографии при визуальном стеганоанализе текста-контейнера.
6. Дать общую характеристику стеганоанализу в области текстовой стеганографии на основе метода «χ-квадрат».
7. Что такое апрош? В чем состоит сущность стеганометода на основе модификации апроша?
8. Что такое кернинг? В чем состоит сущность стеганометода на основе модификации кернинга?
9. Дать сравнительную оценку методов на основе модификации пространственно-геометрических и цветовых параметров символов текста-контейнера (критерий: отношение оправданного объема осаждающей информации к объему контейнера).
10. Какие новые методы текстовой стеганографии вы можете предложить?

## **Лабораторная работа № 14**

# **СОГЛАСОВАНИЕ КРИПТОГРАФИЧЕСКИХ КЛЮЧЕЙ НА ОСНОВЕ ТЕХНОЛОГИЙ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ**

**Цель:** изучение основ построения и функционирования искусственных нейронных сетей (ИНС), а также использования ИНС в криптографии; приобретение практических навыков программной реализации алгоритма согласования ключевой информации на основе технологии ИНС.

**Задачи:**

1. Закрепить теоретические знания по основам построения и функционирования ИНС.
2. Усвоить особенности построения, основные алгоритмы взаимного обучения двух связанных нейронных сетей на основе модели TPM.
3. Разработать приложение для реализации модели TPM с целью согласования двумя сторонами совместного тайного ключа.
4. Познакомиться с методиками оценки криптостойкости алгоритма на основе TPM.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

### **14.1. Теоретические сведения**

#### **14.1.1. Основные принципы построения и основы моделирования ИНС**

Современные суперкомпьютеры превосходно справляются с задачами математических вычислений. Однако существует группа задач, решение которых не является для компьютеров простым. К основным из таких общих функциональных задач относятся: распознавание образов (букв, форм, сигналов), классификация и идентификация объектов, ассоциации.

Нейронные сети, в которых были реализованы указанные основные функции, нашли применение в очень многих областях науки и техники:

- проектирование космических кораблей;
- распознавание речи;
- нечеткая логика;
- компрессия (сжатие) изображений;
- генетические алгоритмы;
- ряды Фурье;
- анализ и прогнозирование в финансовой сфере;
- безопасность информационных систем;
- системы идентификации;
- бизнес-системы принятия решений;
- переработка изображений;
- прогнозирование временных рядов;
- распознавание текста;
- интеллектуальный анализ данных;
- анализ медицинских исследований
- и др.,

а также:

- криptoанализ RSA;
- вычисление хеш-функций;
- протокол обмена ключами.

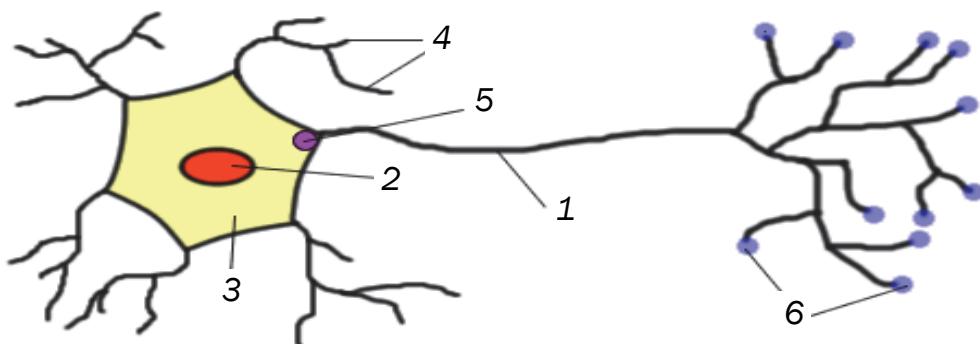
Именно последнее из указанных применений ИНС и является предметом нашего исследования.

#### **14.1.2. Основные принципы построения и основы моделирования ИНС**

Как известно, мозг человека состоит, главным образом, из большого числа соединенных друг с другом элементарных нервных клеток, называемых **нейронами**.

Основная способность нейронов – это возможность управления и выработки нервных импульсов. Каждый нейрон (рис. 14.1) состоит из коротких ветвистых дендритов, тела клетки, а также длинного волокна, называемого аксоном. Дендриты имеют древовидную структуру, каждая ветвь которой соединена с одним из нейронов. Полученные сигналы передаются посредством дендритов к телу клетки. Аксон – это длинный отросток нейрона, который руководит импульсами от тела клетки к другим нейронам.

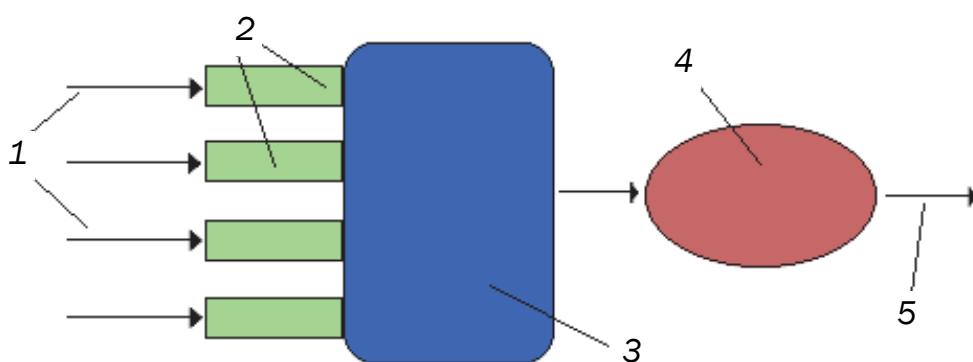
Он соединен с дендритами других нейронов через биохимические стыки, называемые синапсами. Благодаря дендритам тело клетки получает и суммирует сигналы, приходящие от всех его входов. Если общий сигнал, который достиг тела клетки, превысит пороговой уровень, то возбуждающий сигнал с выхода нейрона пересыпается посредством аксона к другим нейронам. К нейрону приходят сигналы, которые могут быть модифицированы через синапсы.



**Рис. 14.1.** Биологическая модель нейрона:  
1 – аксон; 2 – ядро; 3 – тело клетки; 4 – дендриты;  
5 – основание аксона; 6 – синапсы

На основании вышеуказанного описания спроектирован искусственный нейрон, называемый также **персептроном** или **перцептроном** (англ. *perceptron*), симулирующий работу биологического нейрона (рис. 14.2). Впервые модель искусственного нейрона была представлена У. МакКаллоком и У. Питтсом в 1943 г. в исторической работе [74].

Эта модель содержит главные элементы биологического нейрона. Вот их эквиваленты: входные значения – дендриты, весовые коэффициенты – синапсы, суммирующий блок – ядро, функция активации – основание аксона, а выходное значение – это аксон.



**Рис. 14.2.** Модель искусственного нейрона – персептрана:  
1 – входы; 2 – веса; 3 – суммирующий блок; 4 – блок активации; 5 – выход

Математически работу персептрана можно описать так [75]:

$$S = \sum_{i=1}^n w_i x_i, \quad (14.1)$$

где  $x_i$  и  $w_i$  – соответственно  $i$ -е входной сигнал и весовой коэффициент нейрона. Соотношение (14.1) в векторной форме выглядит так:

$$S = W^T X. \quad (14.2)$$

В соответствии с вышеизложенным нейронная сеть (соединение отдельных нейронов), как элементарный эквивалент человеческого мозга, должна состоять из нейронов. Такую самую простую сеть с прямой связью составляет одиночный нейронный слой. В таком слое каждый нейрон получает одинаковый набор входных сигналов  $X = \langle x_1, x_2, \dots, x_n \rangle^T$ , и каждый из них имеет свой собственный вектор весов  $W^{(m)} = \langle w_1^{(m)}, w_2^{(m)}, \dots, w_n^{(m)} \rangle^T$  (где  $m$  – это номер нейрона;  $m = 1, 2, \dots, k$ ). Следовательно, выходной сигнал  $m$ -го нейрона может быть подсчитан следующим образом:

$$S^{(m)} = W^{(m)T} X = \sum_{i=1}^n w_i^{(m)} x_i. \quad (14.3)$$

Таким образом, значение выхода нейрона равно скалярному произведению входных значений на векторы весовых коэффициентов (нейрон с линейной функцией активации). Действие такой сети основывается на вычислении выходов каждого из нейронов на основе общего входного вектора  $X$ .

Представленная выше модель нейронной сети в принципе бесполезна, так как она не способна решать поставленные задачи до тех пор, пока не будет соответствующим способом «натренирована». Этот процесс называется **обучением** и основан на соответствующем подборе коэффициентов вектора весов в контексте решаемой задачи.

Нас далее будет интересовать алгоритм обучения «без учителя», разработанный Д. Хэббом и несколько модифицированный К. Саттоном. Он известен как «правило Хэбба», или просто «обучение без учителя» (англ. *Unsupervised Learning* или *Hebbian Learning*). Процесс обучения данным методом заключается в следующем. Элементы вектора весовых коэффициентов  $w_i^{(m)}$  на каждом

$(n + 1)$ -м шаге равны сумме значения элемента вектора весовых коэффициентов на шаге  $n$  и произведения элементов входного сигнала  $x_i(n)$  на значения выходного сигнала  $y(m)(n)$  обучаемого нейрона. Математически это можно записать в общем виде следующим образом:

$$w_i^m(n+1) = w_i^m(n) + cx_i(n)y^m(n). \quad (14.4)$$

Взаимное обучение двух ИНС касается простейшей одноправленной модели, в которой участвуют два персептрона. Роли таких нейронных сетей заранее не определены, каждая может выполнять функции как учителя, так и ученика (в зависимости от этапа обучения). Это значит, что сети учатся друг у друга, используя для этого полученные результаты, и, стремясь к «общей цели», находят общие элементы в результате своих вычислений.

Эта модель обучения, благодаря своим свойствам, может быть использована в криптографии, а именно в определении и согласовании двумя сторонами (двумя ИНС) криптографического ключа (аналогично известному нам протоколу Диффи – Хеллмана).

Пусть даны два персептрана, каждый из которых получает на вход случайно выбранный вектор входных значений  $x$ . Оба преобразовывают свои внутренние векторы весов в соответствии с принятым правилом обучения. При этом также учитываются собственные выходные величины  $s$ . Вычисляется эта величина следующим образом:

$$s = sign(wx). \quad (14.5)$$

В формуле (14.5)  $w$  – это нормированный  $N$ -размерный входной вектор.

Начальное состояние векторов весов  $w^A$  и  $w^B$  в обеих сетях случайное. Затем на каждом шаге обучения на вход обеих сетей подается один (случайно сгенерированный) входной вектор  $x$ , с помощью которого обе сети вычисляют состояния своих выходов ( $s^A$  и  $s^B$ ). На каждом шаге обучения весовые коэффициенты подвергаются следующему преобразованию:

$$\left. \begin{aligned} w^A(t+1) &= w^A(t) + \frac{\eta}{N} xs^B \Theta(-s^A s^B); \\ w^B(t+1) &= w^B(t) + \frac{\eta}{N} xs^A \Theta(-s^A s^B). \end{aligned} \right\} \quad (14.6)$$

В соответствии с вышеуказанными формулами функция  $\Theta$  определяется следующим образом:

$$\Theta(a) = \begin{cases} 1, & a \geq 0; \\ 0, & a < 0. \end{cases} \quad (14.7)$$

Из этой формулы следует, что функция возвращает ненулевую величину для неотрицательных входных величин. Таким образом, веса активизируются только тогда, когда выходы обеих сетей противоположны. В заключение после каждого шага обучения производится нормализация активизированных векторов весовых коэффициентов.

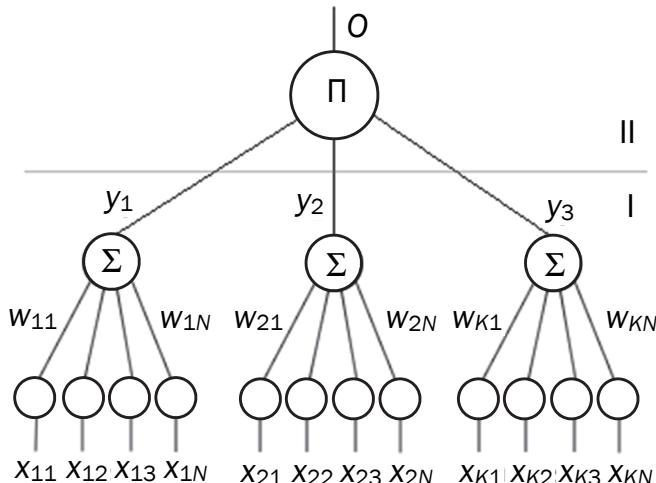
#### **14.1.3. Синхронизация дискретных персепtronов. Модель TPM**

Рассмотренный выше процесс взаимного обучения двух персепtronов может применяться в криптографии. Общую идею такого подхода изложили И. Кантер и В. Кинцель [76].

Именно общее значение векторов весов обоих персепtronов, отличающееся только знаком, может быть использовано в качестве секретного ключа. Протокол этого типа выполняет ту же роль, что и хорошо известный протокол обмена ключами Диффи – Хеллмана. Однако следует обратить внимание на два обстоятельства. Первый из них касается безопасности всей системы. Поскольку если у третьей стороны, наблюдающей за процессом обмена, появится возможность получить внутреннее состояние вектора весов одного из персепtronов, участвующих в процессе обмена ключами, то она сможет синхронизироваться с ним. Вторая проблема касается особенностей практического применения всей системы. Обмен информацией в компьютерных системах базируется на пространстве битов, или на *дискретных* величинах. Поэтому обоснованным является вопрос: будет ли правильным (или сходящимся к общему вектору) процесс обмена ключами для векторов весов с дискретными (битовыми) величинами?

Задача решается с помощью *многослойной* ИНС. Это сеть, называемая **машиной четности** (англ. *parity machine*) или, более точно, **древовидной машиной четности** (англ. *Tree Parity Machine*, TPM). Нейроны представляют собой персептроны с дискретными векторами весов.

Архитектуру сети составляют  $K$  классических персепtronов, принадлежащих внутреннему уровню; внешний уровень (обозначен римской цифрой II) объединяет выходы персепtronов внутреннего уровня (I) (рис. 14.3, здесь  $K = 3$ ).

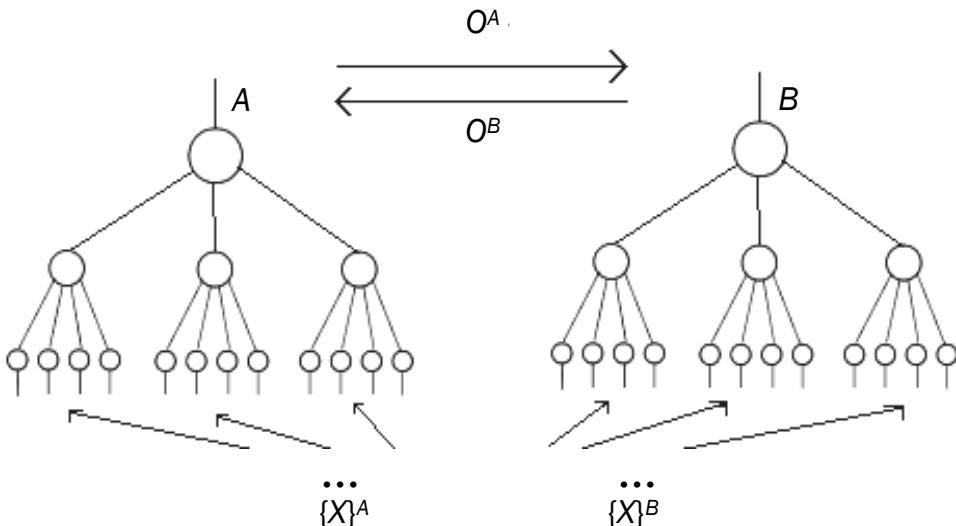


**Рис. 14.3.** Схема двух уровней модели TPM

Каждый из персепtronов имеет  $N$  входов. Следовательно, вектор входных значений всей сети равен  $KN$ . В этой архитектуре приняты следующие обозначения:  $x_{kj}$  –  $j$ -ый вход  $k$ -того персептрона,  $y_k$  – выход  $k$ -того персептрона, где  $k = 1, 2, \dots, K$  и  $j = 1, 2, \dots, N$ . Для упрощения принято, что величины входного вектора бинарны:  $x_{kj} = \{-1, +1\}$ . Каждый из персепtronов имеет веса, которые получают  $j$ -ую величину входа в  $k$ -том персептроне. Следовательно, величины векторов весов обозначаются как  $w_{kj}$ . Выход всей архитектуры  $O$  – это произведение всех выходных величин внутренних персепtronов. Для простоты в дальнейшем будем говорить о рассматриваемой архитектуре как о TPM-архитектуре с тремя внутренними элементами. На величины вектора весов налагаются следующие ограничения:  $w_{kj} = \{-L, -L + 1, \dots, 0, \dots, L - 1, L\}$  [77–81].

Представленная выше архитектура, как и все ее параметры, идентичны для обеих сетей, участвующих в процессе синхронизации. Все структурные элементы (как и в других криптографических системах) известны, однако это не представляет угрозы для всей системы. Секретные элементы, на которых основывается конструкция системы, – это начальные состояния векторов весов. Естественно, сети, участвующие в процессе синхронизации, также взаимно «не знают» векторов весов.

Общая схема процесса синхронизации двух сетей в графическом виде представлена на рис. 14.4. Кратко проанализируем особенности процесса синхронизации сетей на основе архитектуры TPM.



**Рис. 14.4.** Иллюстрация процесса синхронизации двух сетей

Процесс взаимного обучения, или синхронизации, начинается с инициализации весовых параметров обеих сетей подачей на вход соответствующих бинарных последовательностей (для упрощения на рис. 14.4 обозначены соответственно  $\{X\}^A$  и  $\{X\}^B$ ). Их начальное состояние, выбираемое случайным образом, остается секретным на протяжении всего процесса обучения. Каждый следующий шаг ( $t + 1$ ) начинается с подачи на входы обеих сетей выбранного случайному образом вектора  $X$ . Затем вычисляется выходная величина  $O$  для каждой из архитектур следующим способом:

$$O^{A/B} = \prod_{k=1}^K y_k^{A/B} = \prod_{k=1}^K \sigma(\alpha_k^{A/B}) = \prod_{k=1}^K \sigma \left( \sum_{j=1}^N w_{kj}^{A/B} x_{kj} \right). \quad (14.8)$$

Индекс  $A/B$  обозначает, что данная операция касается обеих сетей  $A$  и  $B$ , а единичный индекс  $A$  или  $B$  обозначает, что операция касается соответственно только одной из сетей:  $A$  или  $B$ .

Кроме того, в формуле присутствует модифицированная функция знака  $\sigma$ , определяемая следующим образом:

$$\sigma(\alpha_k^{A/B}) = \begin{cases} 1, & \alpha_k^{A/B} \geq 0; \\ -1, & \alpha_k^{A/B} < 0. \end{cases} \quad (14.9)$$

Выходная величина  $O$  обеих сетей, вычисляемая на основании рассмотренных формул и общего входного вектора  $x$ , передается взаимно каждому из партнеров. Следовательно, обе сети «знают» полученные выходные величины друг друга. Естественно, передача этой информации вызывает ее раскрытие, что приводит к тому, что третьи лица будут иметь доступ к ней. Формула (14.8) – это не классический метод вычисления значения выхода нейронных сетей, так как выходное значение целой архитектуры вычисляется как произведение выходных величин каждого скрытого нейрона.

На основании обеих полученных выходных величин реализован процесс обучения. Активизация векторов весов обеих сетей происходит только тогда, когда обе выходные величины равны друг другу, т. е.  $O^A = O^B$ . Кроме того, внутри данной сети модифицируются веса только тех персепtronов, выходная величина которых  $y_k$  равна выходной величине  $O$  всей сети. Процесс обучения происходит в соответствии, как это было описано выше, с правилом Хэбба, описываемым в виде следующей формулы:

$$w_{kj}^{A/B} = \begin{cases} w_{kj}^{A/B} + O^{A/B} x_{kj}, & O^A = O^B \wedge O^{A/B} = y_k^{A/B}; \\ w_{kj}^{A/B}, & \text{в противном случае.} \end{cases} \quad (14.10)$$

Векторы весов на каждом шаге обучения должны быть подвержены процессу нормализации. Для персепtronов, у которых величины весов являются целыми числами, процесс нормализации связан с необходимостью наложения ограничений на них.

Как было уже сказано, величина каждого весового коэффициента не может выйти за границы промежутка  $[-L, L]$ . Следовательно, каждый этап процесса обучения (активизации векторов весов) требует выполнения следующей операции:

$$w_{kj}^{A/B} = \begin{cases} sign(w_{kj}^{A/B})L, & |w_{kj}^{A/B}| > L; \\ w_{kj}^{A/B}, & \text{в противном случае.} \end{cases} \quad (14.11)$$

Таким образом, процесс обучения начинается с инициализации случным образом генерированного вектора весов каждой из сетей. Как в начальный момент времени, так и на каждом последующем шаге обучения величина обоих векторов весов остается известной только каждой со сторон. Затем каждая из сетей вычисляет выходную величину на основании формулы (14.8), причем обе

архитектуры изменяют свои собственные выходные величины. Соответственно полученным значениям выходов происходит активизация векторов весов согласно формулам (14.10) и (14.11).

Две ИНС, обученные на основании вышеуказанной схемы, впоследствии достигают состояния синхронизации. Это означает, что их векторы весов имеют идентичные величины ( $w^A = w^B$ ).

Число шагов, за которое векторы весов достигают одинаковых значений, зависит от начальных состояний векторов весов, входных значений и от таких параметров системы, как  $K$  и  $N$ . Вся процедура взаимного обучения никак не влияет на два первых параметра, так как выбор начальных состояний векторов весов и значение входных величин генерируются случайным способом. Однако параметры  $K$  и  $N$  пропорционально влияют на увеличение времени синхронизации [79, 82]. Основные особенности программной реализации рассмотренного алгоритма можно найти в [83].

Самый простой способ атаки на синхронизирующуюся сети  $A$  и  $B$  (со стороны  $C$ ) основывается на том, что оппонент может знать архитектуру обеих сетей, а также иметь всю информацию, которой обмениваются между собой сети.

## 14.2. Практическое задание

1. Разработать приложение, реализующее модель TCP, – эмулятор процесса синхронизации весовых коэффициентов двух ИНС.

Рекомендуется выполнить следующие шаги:

1.1. Реализовать класс, описывающий схему персептрона:

- $N$  входов;
- $N$  весовых коэффициентов;
- выход  $y$ ;

• метод расчета значения выхода  $y$  или, более точно, выхода  $k$ -го персептрона  $y_k$ ; используем для этого выражения (14.5) и (14.8):

$$y_k = \text{sign}\left(\sum_{k=1}^N w_{kj}x_{kj}\right),$$

здесь  $\text{sign}(x) = \begin{cases} -1, & \text{если } x \leq 0; \\ \pm 1, & \text{если } x > 0; \end{cases}$

- реализацию одного из алгоритмов обучения (синхронизации) сетей:

– алгоритм Хэбба; при этом соотношение (14.6) представим в несколько ином виде (для сети  $A$ ):

$$(w_i)^A(t+1) = (w_i)^A(t) + (y_i x_i)^A \Theta((y_i)^A(O^B)) \Theta((-O)^A(O^B)); \quad (14.12)$$

обучение другой сети по Хэббу основано на использовании последнего математического выражения при замене  $A$  на  $B$ ;

– алгоритм анти-Хэбба; основано также на использовании (14.12), оператор суммирования в обоих случаях следует поменять на оператор вычитания;

– алгоритм «случайного блуждания»; используется несколько укороченный вариант (14.12):

$$(w_i)^A(t+1) = (w_i)^A(t) + (x_i)^A \Theta((y_i)^A(O^B)) \Theta((-O)^A(O^B)); \quad (14.13)$$

- реализацию метода, налагающего ограничения на весовые коэффициенты в соответствии с выражением (14.11).

1.2. Реализовать класс, описывающий архитектуру нейронной сети TPM:

- $K$  персепtronов;
- выход  $O$  сети;

• метод расчета  $O$  на основе соотношения (14.8); с учетом того, что основная часть вычислений, входящих в данную формулу, нами уже выполнена, достаточно здесь выполнить лишь операцию умножения, т. е. реализовать первый оператор после первого слева знака равенства;

• метод обучения нейронной сети (вызывает метод обучения конкретных – выходы которых совпадают с выходом нейронной сети – персепtronов).

1.3. Реализовать процесс синхронизации двух нейронных сетей, построенных на архитектуре TPM, который в общем виде можно сформулировать так:

1) задаем случайные значения весовых коэффициентов сетей  $A$  и  $B$ :  $w_{ij} \in \{-L, -L+1, \dots, 0, \dots, L\}$ ;

2) выполняем следующие шаги, пока не наступит синхронизация:

2.1) генерируем случайный входной вектор  $X$ :  $x_{ij} = \{-1, +1\}$ ;

2.2) вычисляем  $y_k - e$  для обеих сетей;

2.3) вычисляем  $O^A$  и  $O^B$ ,

2.4) сравниваем выходы двух TPM:

а)  $O^A \neq O^B$  – переход к п. 2.1,

б)  $O^A = O^B$  – применяем выбранное правило обучения.

1.4. Не допустить процесса «бесконечной синхронизации».

1.5. Выводить значения весовых коэффициентов нейронных сетей на экран после наступления синхронизации.

Упрощенный вариант псевдокода рассмотренного алгоритма можно представить следующим образом:

### **Класс Конфигурация Нейронной Сети**

Поля

- Количество Входов Персептрана [N]

- Количество Персептранов [K]

- Ограничения, Налагаемые На Весовые Коэффициенты [L]

Методы

- Преобразование в Конфигурацию Персептрана

Интерфейс Конфигурация Персептрана

Поля

- Количество Входов Персептрана

- Ограничения, Налагаемые На Весовые Коэффициенты

### **Класс Персептрон**

Поля

- Конфигурация Персептрана

- Список Весовых Коэффициентов [1..N]

- Список Входов Персептрана [1..N]

Методы

- Расчет Выходного Значения Сети

вход: список значений входных нейронов [1..N]

выход: сумма произведений значения весовых коэффициентов на значения входных нейронов

- Обучение Персептрана

вход: значение выхода нейронной сети

тело:

- применение к весовым коэффициентам правила обучения (алгоритм Хэбба, алгоритм анти-Хэбба, алгоритм «случайного блуждания»)

- наложение ограничений в соответствии с параметром Конфигурации [-L..L]

### **Класс Нейронная Сеть**

Поля

- Конфигурация Нейронной Сети

- Список Персептранов

*- Выход Нейронной Сети*

Методы

*- Расчет Выхода Нейронной Сети*

*вход: список входных значений [1..N·K]*

*выход: произведение рассчитанных выходов всех персепtronов*

*- Обучение*

*тело: для всех персепtronов вызов метода обучения с передачей в качестве параметра текущего значения выхода нейронной сети*

*- Получения матрицы весовых коэффициентов*

*выход: матрица [N·K], содержащая весовые коэффициенты всех персепtronов*

2. С использованием разработанного приложения произвести не менее 500 реализаций алгоритма синхронизации сетей *A* и *B*, параметры которых соответствуют варианту задания (см. таблицу).

### Варианты заданий

Вариант	Алгоритм обучения	K	N	$\pm L$
1	Хэбба	3	10	3
2	Анти-Хэбба	4	12	2
3	Случайного блуждания	5	8	4
4	Хэбба	5	8	3
5	Анти-Хэбба	4	10	3
6	Случайного блуждания	3	9	5
7	Хэбба	4	10	4
8	Анти-Хэбба	4	7	4
9	Случайного блуждания	4	8	4
10	Хэбба	5	7	5
11	Анти-Хэбба	5	8	5
12	Случайного блуждания	5	9	4
13	Хэбба	3	8	6
14	Анти-Хэбба	3	9	6
15	Случайного блуждания	3	12	5
16	Хэбба	5	8	2

2.1. В каждом из выполненных опытов подсчитать количество шагов, после реализации которых наступает синхронизация.

2.2. Составить распределение: число синхронизаций – число шагов.

2.3. В каждом из опытов выполнять контроль промежутка времени наступления синхронизации после запуска эмулятора. Подсчитать среднее время процесса синхронизации. Сравнить это показатель, а также данные по п. 2.2 с соответствующими результатами, полученными коллегами.

3. Оформить результаты выполнения задания по установленной форме.

### **ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

1. Изобразить схематично структуру персептрана и пояснить аналогии между его компонентами и частями биологического нейрона.

2. Охарактеризовать (и показать на примерах) области использования ИНС.

3. Как в простейшем виде записывается формальное представление персептрана?

4. Охарактеризовать (и показать на примерах) области использования ИНС в криптографии.

5. Дать пояснение к структуре и функционалу информационной системы на основе ИНС, предназначеннной для согласования ключевой информации.

6. Дать характеристику известным алгоритмам обучения ИНС.

7. Какие алгоритмы используются для обучения ИНС, предназначенных для согласования ключевой информации между двумя сторонами?

8. Могут ли легитимно участвовать в процессе синхронизации более трех сетей? Мотивируйте ответ.

9. Дать характеристику криптостойкости системы на основе двух взаимодействующих ИНС.

10. Какие виды атак на нейрокриптографические системы вам известны? В чем заключается их сущность?

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Колесников, В. Л. Компьютерное моделирование в химической технологии. Курсовое и дипломное проектирование: учеб. пособие для студентов вузов / В. Л. Колесников, И. М. Жарский, П. П. Урбанович. – Минск: БГТУ, 2008. – 333 с. (<https://elib.belstu.by/handle/123456789/26015>)
2. Урбанович, П. П. Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 1. Кодирование информации: учеб.-метод. пособие для студентов учреждений высшего образования / П. П. Урбанович, Д. В. Шиман, Н. П. Шутъко. – Минск: БГТУ, 2019. – 116 с. (<https://elib.belstu.by/handle/123456789/29372>)
3. Урбанович, П. П. Защита информации методами криптографии, стеганографии и обfuscации: учеб.-метод. пособие для студентов специальности 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем», направления специальности 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», специальности 1-40 01 01 «Программное обеспечение информационных технологий» специализации 1-40 01 01 10 «Программирование Интернет-приложений» / П. П. Урбанович. – Минск: БГТУ, 2016. – 220 с. (<https://elib.belstu.by/handle/123456789/23763>)
4. Как определить простое число [Электронный ресурс] // КакПросто!: сайт. – 2020. – Режим доступа: <https://www.kakprosto.ru/kak-66290-kak-opredelit-prostoe-chislo#ixzz60MRJ4m5m>. – Дата доступа: 13.01.2020.
5. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – М.: Издательство Триумф, 2003. – 816 с.
6. Бабаш, А. В. Информационная безопасность. Лабораторный практикум: учеб. пособие / А. В. Бабаш, Е. К. Баранова, Ю. Н. Мельникова. – 2-е изд., стер. – М.: КНОРУС, 2013. – 136 с.
7. Гомес Ж. Мир математики. В 2 т. Т. 2: Математики, шпионы и хакеры. Кодирование и криптография / Ж. Гомес. – М.: Де Агостини, 2014. – 144 с. (<https://www.litmir.me/br/?b=247091&p=1>)

8. Национальный корпус русского языка: сайт [Электронный ресурс]. – 2020. – Режим доступа: <http://www.ruscorpora.ru/new/corpora-intro.html>. – Дата доступа: 15.01.2020.
9. Urbanovich, P. Information Protection, Part 7: Basic cryptographic algorithms and standards [Электронный ресурс] / P. Urbanovich. – Minsk: BSTU, 2019. – 58 р. (<https://elib.belstu.by/handle/123456789/29342>)
10. Риксон, Ф. Б. Коды, шифры, сигналы и тайная передача информации / Ф. Б. Риксон – М.: АСТ: Астрель, 2011. – 656 с.
11. Leone Baptista Alberti. De componendis cyfris [Электронный ресурс] // Apprendre enligne: сайт. – 2020. – Режим доступа: <http://www.ap-prendre-en-ligne.net/crypto/alberti/decifris.pdf>. – Дата доступа: 25.01.2020.
12. Шифры замены [Электронный ресурс] // Сайты Google. – 2020. – Режим доступа: <https://www.sites.google.com/site/anisimovkhv/learning/cripto/lecturetema4#tabl41>. – Дата доступа: 05.02.2020.
13. Бабаш, А. В. Криптография / А. В. Бабаш, Г. П. Шанкин. – М.: СОЛОН-ПРЕСС, 2007. – 512 с.
14. Кан, Д. Взломщики кодов / Д. Кан. – М.: Центрполиграф, 2000. – 594 с.
15. Гатченко, Н. А. Криптографическая защита информации: учеб. пособие / Н. А . Гатченко, А. С. Исаев, А. Д. Яковлев. – СПб.: НИУ ИТМО, 2012. – 142 с.
16. Скитала [Электронный ресурс] // Википедия: свободная энциклопедия: сайт. – 2020. – Режим доступа: <https://ru.wikipedia.org/wiki/Скитала>. – Дата доступа: 18.02.2020.
17. Шифры перестановки [Электронный ресурс] // Сайты Google. – 2020. – Режим доступа: <https://www.sites.google.com/site/anisimovkhv/learning/cripto/lecture/tema5>. – Дата доступа: 18.02.2020.
18. Перестановочные шифры [Электронный ресурс] // NoZDR: сайт. – 2020. – Режим доступа: <http://nozdr.ru/games/quest/crypt/cipher/perestanovka>. – Дата доступа: 20.02.2020.
19. Шифровальная машина Энигма [Электронный ресурс] // ADSL: сайт. – 2020. – Режим доступа: <http://www.adsl.kirov.ru/projects/articles/2018/06/24/enigma/>. – Дата доступа: 20.02.2020.
20. Алгоритм Энигмы [Электронный ресурс] // Хабр: сайт. – 2020. – Режим доступа: <https://habr.com/ru/post/217331/>. – Дата доступа: 21.02.2020.
21. Technical Details of the Enigma Machine [Электронный ресурс] // Telenet.be: сайт. – 2020. – Режим доступа: <http://users.telenet.be/d.rijmenants/en/enigmatech.htm#rotors>. – Дата доступа: 22.02.2020.

22. Technical Specification of the Enigma [Электронный ресурс] // WW II Codesand Ciphers: сайт. – 2020. – Режим доступа: <http://www.code-sandciphers.org.uk/enigma/rotorspec.htm>. – Дата доступа: 24.02.2020.
23. Enigma Timeline. Underconstruction [Электронный ресурс] // Crypto Museum: сайт. – 2020. – Режим доступа: <https://www.cryptomuseum.com/crypto/enigma/timeline.htm>. – Дата доступа: 26.02.2020.
24. Enigma Mashinen [Электронный ресурс] // Яндекс. – 2020. – Режим доступа: <https://yandex.by/video/prview/?filmId=4334530940425969248&noreask=1&path=wiard&text=симуляторы+Энигмы>. – Дата доступа: 28.02.2020.
25. Jorge Luis Orejel. Enigma [Электронный ресурс] // CodeProject: сайт. – 2020. – Режим доступа: <https://www.codeproject.com/Articles/831015/ENIGMA>. – Дата доступа: 02.03.2020.
26. Neal, B. Py-Enigma Documentation [Электронный ресурс] / B. Neal // Read The Docs: сайт. – 2020. – Режим доступа: <https://readthedocs.org/projects/py-enigma/downloads/pdf/latest/>. – Дата доступа: 02.03.2020.
27. Криптоанализ «Энигмы» [Электронный ресурс] // Википедия: свободная энциклопедия: сайт. – 2020. – Режим доступа: [https://ru.wikipedia.org/wiki/Криптоанализ\\_«Энигмы»](https://ru.wikipedia.org/wiki/Криптоанализ_«Энигмы»). – Дата доступа: 05.03.2020.
28. Miller, A. R. The Cryptographic Mathematics of Enigma, Center for Cryptologic History National Security Agency [Электронный ресурс] / A. R. Miller // Google Диск. – 2019. – Режим доступа: [https://drive.google.com/file/d/1By1nea1BhIiNwCfykdmQAawkyh5QT\\_hr/view](https://drive.google.com/file/d/1By1nea1BhIiNwCfykdmQAawkyh5QT_hr/view). – Дата доступа: 20.02.2020.
29. Feistel, H. Cryptography and Computer Privacy / H. Feistel // Scientific American. – May 1973. – Vol. 228, No. 5. – P. 15–23. (<http://www.apprendre-en-ligne.net/crypto/bibliotheque/feistel/>)
30. Петров, А. А. Компьютерная безопасность. Криптографические методы защиты / А. А. Петров. – М.: ДМК, 2000. – 448 с.
31. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования: ГОСТ 28147–89. – М.: Издательство стандартов, 1989. – 26 с.
32. Информационные технологии и безопасность. Криптографические алгоритмы генерации псевдослучайных чисел: СТБ 34.101.47–2017. – Минск: Госстандарт, 2017. – 22 с. (<http://apmi.bsu.by/assets/files/std/brng-spec24.pdf>)

33. Урбанович, П. П. Защита информации и надежность информационных систем: пособие / П. П. Урбанович, Д. В. Шиман. – Минск: БГТУ, 2014. – 90 с. (<https://elib.belstu.by/handle/123456789/23761>)
34. Urbanovich, P. Information Protection, Part 5: Error Correcting Codes / P. Urbanovich. – Minsk: BSTU, 2019. – 34 p. (<https://elib.belstu.by/handle/123456789/29340>)
35. Генератор псевдослучайных чисел на основе алгоритма BBS [Электронный ресурс] // НОУ «ИНТУИТ». – 2020. – Режим доступа: <https://www.intuit.ru/studies/courses/691/547/lecture/12383?page=3>. – Дата доступа: 17.03.2020.
36. Алгоритм RC4 [Электронный ресурс] // Сайт НОУ «ИНТУИТ». – 2020. – Режим доступа: <https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=2>. – Дата доступа: 20.03.2020.
37. Алгоритм RC4 [Электронный ресурс] // Викиучебник: сайт. – 2020. – Режим доступа: [https://ru.wikibooks.org/wiki/Реализации\\_алгоритмов/RC4](https://ru.wikibooks.org/wiki/Реализации_алгоритмов/RC4). – Дата доступа: 20.03.2020.
38. Саломаа, А. Криптография с открытым ключом / А. Саломаа. – М.: Мир, 1995. – 318 с. ([https://web.archive.org/web/20111119210028/http://www.ssl.stu.neva.ru/psw/crypto/open\\_key\\_crypt.pdf](https://web.archive.org/web/20111119210028/http://www.ssl.stu.neva.ru/psw/crypto/open_key_crypt.pdf))
39. Харин, Ю. С. Математические основы криптологии: учеб. пособие / Ю. С. Харин, В. И. Берник, Г. В. Матвеев. – Минск: БГУ, 1999. – 319 с.
40. Коркинъ, А. Н. Таблица первообразныхъ корней и характеристовъ, къ нимъ относящихся, для простыхъ чиселъ, меньшихъ 4000 / А. Н. Коркинъ. // Матем. сб. – 1909. – Т. 27, № 1. – С. 121–137. (<http://www.mathet.ru/links/10b5c3ec64e112fd63987586e9f47e93/sm6551.pdf>)
41. Rivest, R. RFC 1320: The MD4 Message Digest Algorithm / R. Rivest. – MIT Laboratory for Computer Science & RSA Data Security, 1990. – 20 p. (<https://tools.ietf.org/html/rfc1320>)
42. Rivest, R. RFC 1321: The MD5 Message-Digest Algorithm The MD5 Message-Digest Algorithm / R. Rivest. – Internet Engineering Task Force, 1992. – 21 p. (<https://tools.ietf.org/html/rfc1321>)
43. RFC 3174: US Secure Hash Algorithm 1 (SHA1). – Network Working Group, 2001. – 22 p. (<https://tools.ietf.org/html/rfc3174>)
44. Housley, R. RFC 3174: A 224-bit One-way Hash Function: SHA-224 / R. Housley. – Network Working Group, 2004. – 6 p. (<https://tools.ietf.org/html/rfc3874>)

45. RFC 4634: US Secure Hash Algorithms (SHA and HMAC-SHA). – Network Working Group, 2006. – 108 p. (<https://tools.ietf.org/html/rfc4634>)
46. FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. – National Institute of Standards and Technology, 2015. – 29 p. (<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf>)
47. MD6 website [Электронный ресурс] // MIT CSAIL: сайт. – 2020. – Режим доступа: <http://groups.csail.mit.edu/cis/md6/>. – Дата доступа: 15.04.2020.
48. Информационная технология (ИТ). Криптографическая защита информации. Функция хэширования: ГОСТ Р 34.11–94. – М.: Издательство стандартов, 1994. – 16 с. (<http://docs.cntd.ru/document/1200004857>)
49. Cryptographic sponge functions [Электронный ресурс] / G. Bertoni [et al.]. – 2011. – Режим доступа: <http://sponge.nokeon.org/CSF-0.1.pdf>. – Дата доступа: 15.04.2020.
50. Информационные технологии и безопасность. Алгоритмы хэширования: СТБ 34.101.77–2016. – Минск: Госстандарт, 2016. – 13 с. (<http://www.apmi.bsu.by/assets/files/std/bash-spec10.pdf>)
51. Информационные технологии и безопасность. Алгоритмы электронной цифровой подписи и транспорта ключа на основе эллиптических кривых: СТБ 34.101.45–2013. – Минск: Госстандарт, 2013. – 29 с. (<http://apmi.bsu.by/assets/files/std/bign-spec29.pdf>)
52. Standards for Efficient Cryptography. SEC 2: Recommended Elliptic Curve Domain Parameters [Электронный ресурс] // Standards for Efficient Cryptography: сайт. – 2020. – Режим доступа: <https://www.secg.org/SEC2-Ver-1.0.pdf>. – Дата доступа: 06.05.2020.
53. Грибунин, В. Г. Цифровая стеганография. Аспекты защиты / В. Г. Грибунин, И. Н. Оков, И. В. Туринцев. – М.: Солон-Пресс, 2002. – 272 с.
54. Text steganography application for protection and transfer of the information / P. Urbanovich [et al.] // Przeglad elektrotechniczny. – 2012. – № 8. – Р. 342–344. (<https://elib.belstu.by/handle/123456789/24783>)
55. Шутько, Н. П. Защита авторских прав на электронные текстовые документы методами стеганографии / Н. П. Шутько // Труды БГТУ. – 2013. – № 6 (162): Физико-математические науки и информатика. – С. 131–134. (<https://elib.belstu.by/handle/123456789/9708>)
56. Шутько, Н. П. Защита авторских прав на текстовые документы на основе стеганографической модификации цвета символов

текста / Н. П. Шутько, П. П. Урбанович // Информационные технологии: материалы 83-й научно-технической конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием), Минск, 4–15 февраля 2019 г. / отв. за изд. И. В. Войтов; БГТУ. – Минск: БГТУ, 2019. – С. 41–43. (<https://elib.belstu.by/handle/123456789/28369>)

57. Urbanovich, P. Theoretical Model of a Multi-Key Steganography System / P. Urbanovich, N. Shutko // Recent Developments in Mathematics and Informatics. Contemporary Mathematics and Computer Science. Vol. 2, Chapter 11. – Lublin: KUL, 2016. – P. 181–202. (<https://elib.belstu.by/handle/123456789/24543>)

58. Шутько, Н. П. Моделирование стеганографической системы в задачах по охране авторских прав / Н. П. Шутько, Н. И. Листопад, П. П. Урбанович // Восьмая Междунар. научно-техн. конф. «Информационные технологии в промышленности» (ITI'2015): тезисы докладов. – Минск: ОИПИ НАН Беларуси, 2015. – С. 30–31. (<https://elib.belstu.by/handle/123456789/25880>)

59. Блинова, Е. А. Сравнительные особенности использования стеганографических методов в электронных картах / Е. А. Блинова, П. П. Урбанович // X Международная научно-техническая конференция «Информационные технологии в промышленности, логистике и социальной сфере» (ITI\*2019), Минск, 23–24 мая 2019 г.: тезисы докладов. – Минск: ОИПИ НАН Беларуси, 2019. – С. 22–25. (<https://elib.belstu.by/handle/123456789/29368>)

60. Блинова, Е. А. Стеганографический метод на основе встраивания дополнительных значений координат в изображения формата SVG / Е. А. Блинова, П. П. Урбанович // Труды БГТУ. Сер. 3, Физико-математические науки и информатика. – 2018. – № 1 (206). – С. 104–109. (<https://elib.belstu.by/handle/123456789/25323>)

61. Блинова, Е. А. Стеганографический метод на основе встраивания дополнительных значений координат в пространственные данные, хранящиеся в базе данных / Е. А. Блинова, П. П. Урбанович // Информационные технологии: тезисы докладов 82-й научно-технической конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием), Минск, 1–14 февраля 2018 г. / Белорусский государственный технологический университет. – Минск: БГТУ, 2018. – С. 8–9. (<https://elib.belstu.by/handle/123456789/24678>)

62. Сущеня, А. А. Применение форматов электронных книг при передаче конфиденциальной информации методами компьютерной

стеганографии / А. А. Сущеня, П. П. Урбанович // Информационные технологии: материалы 83-й научно-технической конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием), Минск, 4–15 февраля 2019 г. / отв. за изд. И. В. Войтов; БГТУ. – Минск: БГТУ, 2019. – С. 39–40. (<https://elib.belstu.by/handle/123456789/28378>)

63. Колмаков, М. В. Особенности применения стеганографических методов в альтернативных потоках файловой системы NTFS / М. В. Колмаков, Е. А. Блинова // Информационные технологии: тезисы докладов 82-й научно-технической конференции профессорско-преподавательского состава, научных сотрудников и аспирантов (с международным участием), Минск, 1–14 февраля 2018 г. / Белорусский государственный технологический университет. – Минск: БГТУ, 2018. – С. 23–24. (<https://elib.belstu.by/handle/123456789/24667>)

64. Сущеня, А. А. Программное средство стеганографического преобразования текстов-контейнеров на основе языка разметки XML / А. А. Сущеня // 69-я научно-техническая конференция учащихся, студентов и магистрантов, Минск, 2–13 апреля 2018 г.: сборник научных работ: в 4 ч. Ч. 4 / Белорусский государственный технологический университет. – Минск: БГТУ, 2018. – С. 81–84. (<https://elib.belstu.by/handle/123456789/27087>)

65. Плаковицкий, В. А. Шифрование кодов программ на основе ключа, задаваемого рекуррентными математическими соотношениями / В. А. Плаковицкий, П. П. Урбанович // Труды БГТУ. – 2012. – № 6 (153): Физико-математические науки и информатика. – С. 146–148. (<https://elib.belstu.by/handle/123456789/3234>)

66. Andreas Westfeld and Andreas Pfitzmann. Attacks on Steganographic Systems [Электронный ресурс] // CiteSeerX: сайт. – 2020. – Режим доступа: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.5975&rep=rep1&type=pdf>. – Дата доступа: 06.05.2020.

67. Шутько, Н. П. Алгоритмы реализации методов текстовой стеганографии на основе модификации пространственно-геометрических и цветовых параметров текста / Н. П. Шутько // Труды БГТУ. – 2016. – № 6 (188): Физико-математические науки и информатика. – С. 160–165. (<https://elib.belstu.by/handle/123456789/20131>)

68. Software tool for the analysis of the text steganography method based on the modification of the characters color / P. Urbanovich [et al.] //

Proc. of 11th Intern. Conf. NEET'2019, Zakopane, Poland, June 25–28, 2019. – Lublin University of Techn., 2019. – P. 33. (<https://elib.belstu.by/handle/123456789/31442>)

69. Shutko, N. The use of aprosh and kerning in text steganography / N. Shutko // Przegląd elektrotechniczny. – 2016. – № 10. – P. 222–225. (<https://elib.belstu.by/handle/123456789/24542>)

70. Shutko, N. A method of syntactic text steganography based on modification of the document-container aprosh / N. Shutko, P. Urbanovich, P. Zukowski // Przegląd elektrotechniczny. – 2018. – R. 94, № 6. – P. 82–85. (<https://elib.belstu.by/handle/123456789/25877>)

71. Риморев, А. В. Программные коды некоторых функций для реализации методов текстовой стеганографии / А. В. Риморев, П. П. Урбанович. – Минск: БГТУ, 2020. – 13 с. (<https://elib.belstu.by/handle/123456789/33973>)

72. Пласковицкий, В. А. Программный код средства текстовой стеганографии «Sword» / В. А. Пласковицкий, П. П. Урбанович, Н. П. Шутъко. – Минск: БГТУ, 2020. – 36 с. (<https://elib.belstu.by/handle/123456789/33971>)

73. Сущеня, А. А. Исходный текст приложения QuatesEmbed / А. А. Сущеня, П. П. Урбанович, А. О. Берников. – Минск: БГТУ, 2020. – 12 с. (<https://elib.belstu.by/handle/123456789/33972>)

74. McCulloch, W. A logical calculus of the ideas immanent in nervous activity / W. McCulloch, W. Pitts // Bulletin of Mathematical Biophysics. – 1943. – No. 7. – P. 115–133.

75. Головко, В. А. Нейросетевые технологии обработки данных: учеб. пособие / В. А. Головко, В. В. Краснопрошин. – Минск: БГУ, 2017. – 263 с. (<http://elib.bsu.by/bitstream/123456789/193558/1/Golovko.pdf>)

76. Kanter, I. Secure exchange of information by synchronization of neural networks / I. Kanter, W. Kinzel, E. Kanter // Europhys. Lett. – 2002. – P. 141–147. ([https://link.springer.com/chapter/10.1007%2F3-540-45618-X\\_30](https://link.springer.com/chapter/10.1007%2F3-540-45618-X_30))

77. Плонковски, М. Использование нейронных сетей в системах криптографического преобразования информации / М. Плонковски, П. П. Урбанович // Известия Белорусской инженерной академии. – 2004. – № 1 (17). – С. 13–15. (<https://elib.belstu.by/handle/123456789/26748>)

78. Плонковски, М. Криптографическое преобразование информации на основе нейросетевых технологий / М. Плонковски,

П. П. Урбанович // Труды БГТУ. Сер. VI, Физико-математические науки и информатика. – 2005. – Вып. XIII. – С. 161–164. (<https://elib.belstu.by/handle/123456789/26746>)

79. Урбанович, П. П. Анализ синхронизации нейронных сетей в прикладной криптографии / П. П. Урбанович, И. А. Бирюк, М. Д. Плонковски // Информационные технологии и системы 2019 (ИТС 2019): материалы международной научной конференции, Минск, 30 октября 2019 г. = Information Technologies and Systems 2019 (ITS 2019): Proceeding of the International Conference, Minsk, 30<sup>th</sup> October 2019 / редкол.: Л. Ю. Шилин [и др.]. – Минск: БГУИР, 2019. – С. 278–279. (<https://elib.belstu.by/handle/123456789/33008>)

80. Урбанович, П. П. Сравнительный анализ методов взаимообучения нейронных сетей в задачах обмена конфиденциальной информацией / П. П. Урбанович, К. В. Чуриков // Труды БГТУ. Сер. VI, Физико-математические науки и информатика. – 2010. – Вып. XVIII. – С. 163–166. (<https://elib.belstu.by/handle/123456789/24402>)

81. Urbanovich, P. Probabilistic measure of space for neurocryptographic system solutions / P. Urbanovich, D. Karczmarski, M. Plonkowski // Proc. of 11th Intern. Conf. NEET'2019, Zakopane, Poland, June 25–28, 2019. – Lublin University of Techn., 2019. – Р. 32. (<https://elib.belstu.by/handle/123456789/31444>)

82. Урбанович, П. П. Моделирование и анализ процесса синхронизации нейронных сетей для обмена критической информацией / П. П. Урбанович, М. Долецки // Материалы XVII Международной научно-технической конференции «Комплексная защита информации. Безопасность информационных технологий», Сузdal', 18 мая 2012 г. – Сузdal', 2012. – С. 255–257. (<https://elib.belstu.by/handle/123456789/26744>)

83. Плонковски, М. Д. Листинг программного кода метода TPM / М. Д. Плонковски, П. П. Урбанович. – Минск: БГТУ, 2020. – 3 с.

# ОГЛАВЛЕНИЕ

Предисловие .....	3
Лабораторная работа № 1. Основы теории чисел и их использование в криптографии .....	4
Лабораторная работа № 2. Исследование криптографических шифров на основе подстановки (замены) символов ....	19
Лабораторная работа № 3. Исследование криптографических шифров на основе перестановки символов.....	40
Лабораторная работа № 4. Изучение устройства и функциональных особенностей шифровальной машины «Энигма» ...	53
Лабораторная работа № 5. Исследование блочных шифров...	68
Лабораторная работа № 6. Исследование потоковых шифров...	87
Лабораторная работа № 7. Исследование асимметричных шифров .....	102
Лабораторная работа № 8. Исследование асимметричных шифров RSA и Эль-Гамаля .....	111
Лабораторная работа № 9. Исследование криптографических хеш-функций.....	121
Лабораторная работа № 10. Исследование алгоритмов генерации и верификации электронной цифровой подписи .....	136
Лабораторная работа № 11. Исследование криптографических алгоритмов на основе эллиптических кривых.....	146
Лабораторная работа № 12. Исследование стеганографического метода на основе преобразования наименее значащих битов .....	167
Лабораторная работа № 13. Исследование методов текстовой стеганографии.....	179
Лабораторная работа № 14. Согласование криптографических ключей на основе технологий искусственных нейронных сетей.....	202
Список использованных источников .....	216

Учебное издание

**Урбанович Павел Павлович  
Шутько Надежда Павловна**

**ЛАБОРАТОРНЫЙ ПРАКТИКУМ  
ПО ДИСЦИПЛИНАМ «ЗАЩИТА ИНФОРМАЦИИ  
И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ»  
И «КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ  
ЗАЩИТЫ ИНФОРМАЦИИ»**

**В 2-х частях**

**Часть 2. Криптографические и стеганографические  
методы защиты информации**

Учебно-методическое пособие

Редактор *О. П. Приходько*  
Компьютерная верстка *Д. С. Жих*  
Дизайн обложки *П. П. Падалец*  
Корректор *О. П. Приходько*

Подписано в печать 17.12.2020. Формат 60×84<sup>1</sup>/<sub>16</sub>.  
Бумага офсетная. Гарнитура Таймс. Печать ризографическая.  
Усл. печ. л. 13,1. Уч.-изд. л. 13,6.  
Тираж 150 экз. Заказ .

Издатель и полиграфическое исполнение  
УО «Белорусский государственный технологический университет».  
Свидетельство о государственной регистрации издателя,  
изготовителя, распространителя печатных изданий  
№ 1/227 от 20.03.2014.  
Ул. Свердлова, 13а, 220006, г. Минск.