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 модулярной 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 экспериментов 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 Ферма́*, *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, ...}.

Определение 3. *Делимость* – одно из основных понятий теории чисел.

Если для некоторого целого числа a и натурального числа b существует целое число q, такое, что $bq=a$, то говорят, что число a *делится* на b. В этом случае b называется *делителем* числа a, а a называется *кратным* числа b. При этом используются следующие обозначения:

$a \vdots b$ – 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 $a = bq + r$, $0 \le r \le b$. Число q называется *неполным частным*, а число r – *остатком* от деления 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, 2365347734339, $2756839 - 1$. 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:

$$n = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_z,\ z > 1. \qquad (1.1)$$

Пример 3. Целое число 1554985071 = 3·3·4463 ·38713 – произведение четырех простых чисел, два из которых совпадают.

Пример 4. Целое число 39616304 = 2·13·7·2·23·13·2·13·2·7 = = 2·2·2·2·7·7·13·13·13·23.

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 39616304 в виде простых сомножителей. Порядок действий виден из нижеследующей иллюстрации.

144	2	1-й шаг	39616304	2
72	2	2-й шаг	19808152	2
36	2		9904076	2
18	2	…..	4952038	2
9	3		2476019	7
3	3		353717	13
1			27209	7
			3887	13
			299	13
			23	23
			1	

Таким образом, 144 = 2·2·2·2·3·3. 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 $n=qp$ 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 криптостойкость 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 чем 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.

Существует правила, способные заметно сократить время вычислений [3].

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; $\sqrt{287}\approx16{,}94$).

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: 136827658235479371. 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 = 29·3. 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: 2576562845756365782383. Сумма его четных цифр 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 с*войстве 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: 2576562845756365782383. Перепишем его так: 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 2 в. до н. э. и известен сейчас как «*решето Эратосфена*». Его суть в последова-

тельном исключении из списка целых чисел от 1 до n чисел (или из сокращенного диапазона, например, от m до n, $1<m\leq n$), 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, …, n).

Пусть некоторая переменная (положим 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, …).

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/ ln15 ≈ 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, понятно, 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 [4].

1.1.3 Взаимно простые числа и φ-функция

Понятие *делимости чисел* (см. Определение 3) является одним из важных в теории чисел. С этим понятием, а также с его производным – *общим делителем* (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 метод или *алгоритм Евклида* (примеры его использования приведены в [2]. В основе алгоритма лежит Определение 5. В соответствии с этим определением используется цепочка вычислений двумя исходными (начальными) числами: a и b:

$$a_i = b_i q_i + r_i, \quad 0 \le r_i \le b_i. \tag{1.2}$$

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

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

$$1234 = 54 \cdot 22 + 46;$$
$$54 = 46 \cdot 1 + 8;$$
$$46 = 8 \cdot 5 + 6;$$
$$8 = 6 \cdot 1 + \mathbf{2};$$
$$6 = 2 \cdot 3 + 0.$$

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 просты тогда и только тогда, когда существуют такие целые u и v, что выполняется равенство

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

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

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

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 таблица, из которой можно легко понять, как по алгоритму Евклида вычисляются остатки:

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

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:
$2 = 8–(46–8\cdot5)\cdot1 = 8–46+8\cdot5 = 8\cdot6–46 = (54–46)\cdot6–46 = 54\cdot6–46\cdot6–46 = 54\cdot6 – – 46\cdot7 = 54\cdot6–(1234–54\cdot22)\cdot7 = 54\cdot6–1234\cdot7+54\cdot154 =$ **$54\cdot160+(-7)\cdot1234$** $= 8640– –8638$. Из выражения перед последним знаком равенства (выделено) следует, что для нашего примера $u = -7$ и $v = 160$ в соответствии с формой записи в (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

$$n = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_n^{a_n} \tag{1.5}$$

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

$$\varphi(n) = n \times \left(1 - \frac{1}{p_1}\right) \times \left(1 - \frac{1}{p_2}\right) \times \cdots \times \left(1 - \frac{1}{p_n}\right) \tag{1.6}$$

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

Пример 16. 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. $\varphi(12) = 4$ – получено методом «ручного» подсчета.

Теперь подсчитаем $\varphi(12)$ по (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, $\varphi(12)$:

$$\varphi(12) = 12 \cdot (1–1/2) \cdot (1–1/3) = 4.$$

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

$$\varphi(p) = (p – 1)\,(q – 1). \tag{1.7}$$

1.1.4 Модулярная арифметика и обратные числа по модулю

Понятие «*модулярная арифметика*» ввел немецкий ученый 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:

$$a \equiv b \pmod{n} \quad \text{или} \quad a \equiv b \bmod n.$$

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 17. 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 18. Справедливы следующие сравнения чисел:

$–5 \equiv 7 \bmod 4 \equiv 11 \bmod 4 \equiv 23 \bmod 4 \equiv 3 \bmod 4$.

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 ($a^m \bmod n \equiv (a \bmod n)^m$, если $a \equiv b \bmod n$, то $a^m \equiv b^m \pmod n$; другие примеры см. в [2]).

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

$$ax \equiv 1 \bmod n \qquad (1.8)$$

предусматривает поиск таких значений x и k, которые удовлетворяют равенству

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

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

$$1 \equiv ax \bmod n. \qquad (1.10)$$

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:

$$a^{-1} \equiv x \bmod n. \qquad (1.11)$$

Пример 19. *При* $a=5$ и $n=14$ получим $x=3$: $5^{-1} \equiv 3 \bmod 14$, так как $5\cdot 3 \pmod{14} \equiv 1$.

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

Справедливо также: если

$$x^{-1} \equiv y \bmod n, \text{ то } y^{-1} \equiv x \bmod n. \qquad (1.12)$$

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

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

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 [2]).

Пример 20. Решить уравнение $7y \equiv 1 \bmod 40$ или $y^{-1} \equiv 7 \bmod 40$

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

$40 = 7 \cdot 5 + 5,$

$7 = 5 \cdot 1 + 2,$

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

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

$1=5–2\cdot2=5–2(7–5\cdot1)=5\cdot3+7(-2)=(40–7\cdot5)3+7(-2) = 40\cdot3+7(-17) = kn + xy=1 (\bmod n)$, или $7(-17) = 7y$, так как $-17 \bmod 40 = 23$, 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:

$$a^n \equiv 1 \bmod n. \qquad (1.14)$$

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

$$a^{\varphi(n)} \bmod n \equiv 1. \qquad (1.15)$$

Последнее выражение можно переписать в следующем виде:

$$a^{-1} \bmod n \equiv a^{\varphi(n)-1} \bmod n. \qquad (1.16)$$

Пример 21. Найти число, обратное 5 по модулю 7. Число 7 является простым. Поэтому $\varphi(7) = 7 – 1 = 6$. Теперь с помощью (1.16) получаем: $5^{6-1} \bmod 7 \equiv 5^5 \bmod 7 \equiv 3$.

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

1.2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Рекомендация! Перед выполнением практического задания можно познакомиться с функционалом программного средства *L_PROST*, особенностями выполнения с его помощью операций над простыми числами и являющегося приложением на компакт-диске к [5].

На рисунке 1.1 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.

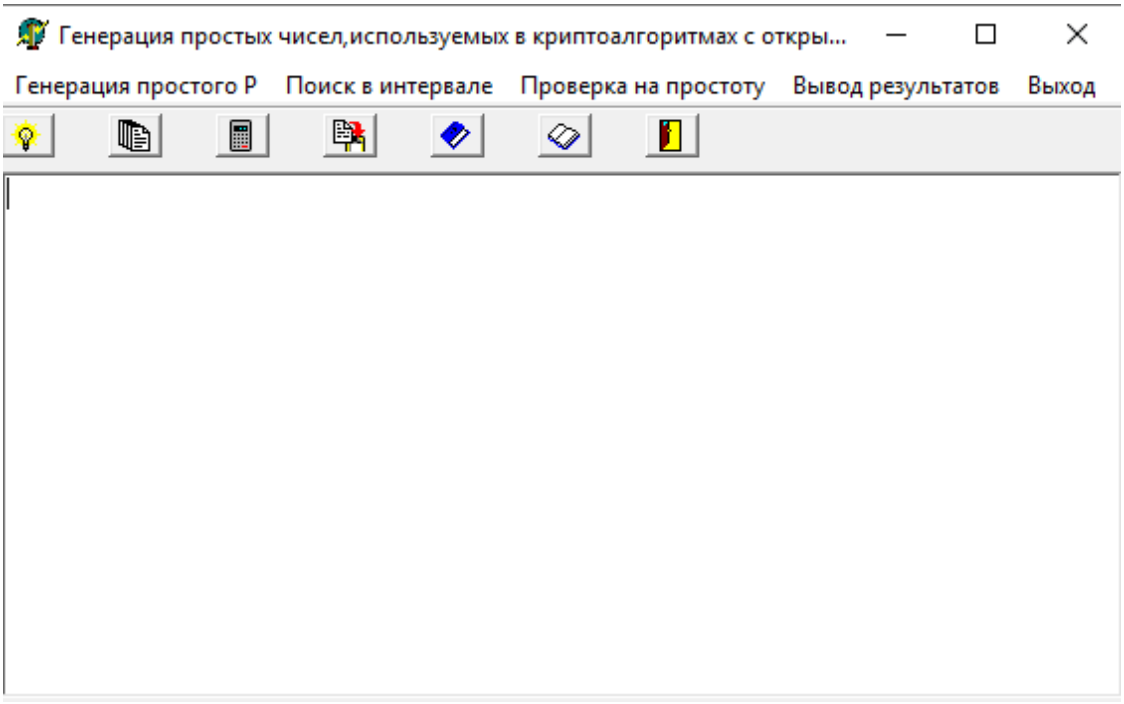


Рисунок 1.1 Основное диалоговое окно программного средства *L_PROST*

1. Используя *L_PROST*, найти все простые числа в интервале [2, *n*]. Значение *n* соответствует варианту из таблицы 1.1, указанному преподавателем.

Таблица 1.1 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

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*||*n* (таблица 1.1), 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.п. 1 и 2.

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

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 [2].

В этом разделе материалов кротко 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 $\{A_{xC}\}_n$, $b \neq n$, $1 \le b, n, x, y \le N$, N – мощность алфавита), используемые для замены разных букв открытого текста, не пересекались:

$$\{A_{xC}\}_b \cap \{A_{xC}\}_n = 0.$$

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. Моноалфавитные шифры подстановки

В данных шифрах операция замена производится только 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

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

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

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

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

Соотношения (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.) [6] (в этой книге любознательный читатель найдет также много исторической информации по криптографии).

Пример 1. Имеем открытый текст M_i = «*cba*». На основе шифра Цезаря C_i = «*fed*».

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» [6].

Одним из существенных недостатков моноалфавитных шифров является их низкая криптостойкость. Зачастую метод криптоанализа 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» [7] (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 всём многообразии жанров, стилей, территориальных и социальных вариантов и т. п.) такая таблица выглядит следующим образом (таблица 2.1).

Таблица 2.1. Частота появления букв русского языка в текстах

№ п/п	Буква	Частота, %	№ 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	Ъ	0.04
16	Я	2.01	33	Ё	0.04
17	Ы	1.90			

Существуют подобные таблицы для пар букв (биграмм). Например, часто встречаемыми 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 станицах 17-19 пособия [8].

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.

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

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 = '*HELLO* ', то C_i = '*FEJJN*', если же M_i = '*VENIVIDIVICI* ', то C_i = '*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: $0 \le a < N$. 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 в нижеследующей таблице:

0 1 2 3 4 5 10 15 20 25

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

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

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

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 = '*VENIVIDIVICI* ', то C_i = '*EUOYEYSYEYQY*'.

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

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:

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

где a и b – целые числа.

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

Приме 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

Если M_i = '*VENIVIDIVICI*', то получение зашифрованного сообщения в деталях показывает таблица 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 = = '*QRSCQCOCQCLC*'.

Расшифрование основано на использовании соотношения

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

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*** [9]. Шифр представляется в виде таблицы. Наверху горизонтально и слева вертикально записывался стандартный алфавит. В ячейках таблицы записываются числа в определенном порядке. Одним из возможных вариантов такой таблицы для алфавита русского языка будет показанная ниже таблица, точнее – ее фрагмент (таблица 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 = ‘АВВА’. Сообщение состоит из двух пар (биграмм): АВ и ВА и будет зашифровано так: 003067.

Другими известными полиграммными шифрами являются ***шифр Плейфера*** и ***шифр Хилла*** [9].

С точки зрения криптостойкости рассматриваемый тип шифров имеет преимущества перед моноалфавитными шифрами. Это связано с тем, что 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 *однозвучные 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* (омофоны – 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 веке. В книге «Трактат о шифрах» Леона Баттисты Альберти (итальянский ученый, архитектор, теоретик искусства, секретарь папы Климентия XII), опубликованной в 1466 г. [10], 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 таблице 2.4 представлен фрагмент таблицы подстановок для алфавита русского языка [11].

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» [11].

Книжный шифр. Заметным вкладом греческого ученого Энея Тактика в криптографию является предложенный им так называемый книжный шифр [10, 11]. После Первой мировой войны книжный шифр приобрел иной вид. Шифрозамена для каждой буквы определялась набором цифр, которые указывали на номер страницы, строки и позиции в строке (вспомните пример использования такого шифра известными героями фильма «17 мгновений весны»). Даже с формальной стороны отсутствие полной электронной базы изданных к настоящему времени книг делает процедуру взлома шифра практически не выполнимой.

Таблица 2.4 Фрагмент таблицы подстановок для системы омофонов

№ 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		
…	…						…	…	…		
47	776						828	…	954		
…	…						…				
80	901						886				
…							…				
110							903				

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» [10] 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 дисков, 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.

Рисунок 2.1. Реплика диска Альберти, используемого Конфедерацией во время Гражданской войны в Америке [12]

Таблица Трисемуса. В 1518 году 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. Рассмотрим процесс зашифрования сообщения M_i = «*БГТУ*», используя таблицу, фрагмент которой показан на рисунке 2.2.

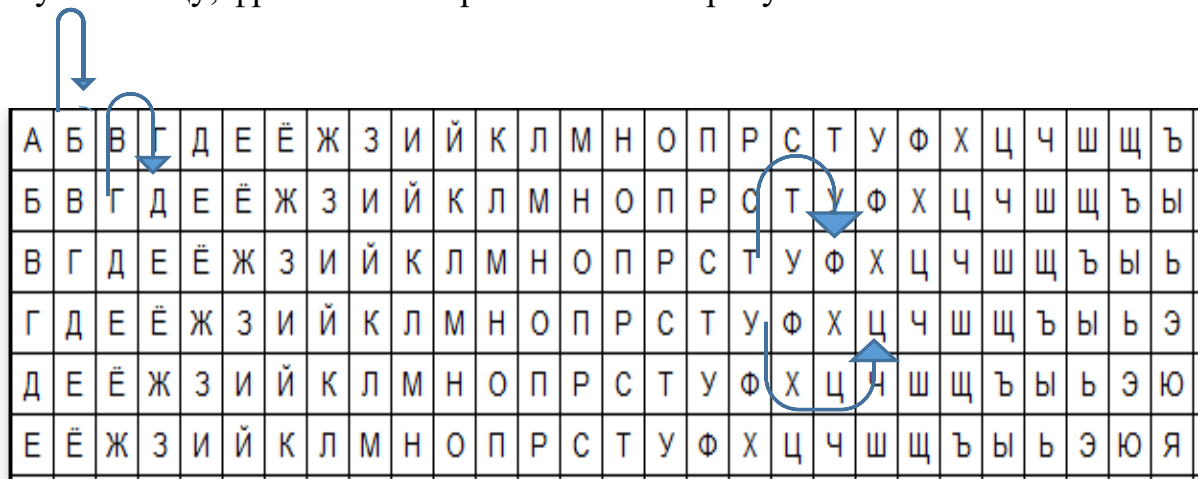


Рисунок 2.2 Фрагмент таблицы Трисемуса для алфавита русского языка

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

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 4х8 либо 8х4.

Пример 6. Пусть M_i = «*ПРИШЕЛУВИДЕЛПОБЕДИЛ*», а ключевое слово – «*ЦЕЗАРЬ*». Используем таблицу 8х4.

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 вышеуказанного принципу подстановки, получим C_i = = «*ФГМЭЫПШИМКЫПФУЖЬКМП*».

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 $N \cdot N$ (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, например, в [2] (с. 41–43).

Листинг 2.1 содержит часть кода, реализующего алгоритм шифрования Виженера.

Следует добавить, что в 1863 г. Фридрих Касиски опубликовал алгоритм атаки на этот шифр, хотя известны случаи его 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 [5]. 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 отозвался о шифре Виженера, как о неподдающемся взлому [13].

```
/*
 * Главный цикл, который проходит по входной строке
 */
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 2.1 Фрагмент кода, реализующего алгоритм шифра Виженера

Роторные машины. Идеи Альберти и Виженера использовались при создании электромеханических роторных машин первой половины XX века. Некоторые из них использовались в разных странах вплоть до 1980-х годов. 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 «Энигма». 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 [14].

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 на другие.

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

2.2 ОБЩИЕ СВЕДЕНИЯ О КРИПТОАНАЛИЗЕ

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

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

Кратко проанализируем основные криптоатаки [2, 4, 11, 29].

Атака с известным шифртекстом (ciphertext only attack). Предполагается, что противник знает алгоритм шифрования, у него имеется набор перехваченных шифрограмм, 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 криптоанализ* – метод взлома шифра, основывающийся на предположении о существовании зависимости между частотой появления символов алфавита в открытых сообщениях и соответствующих шифрозамен в шифрограммах (этот вопрос с практической точки зрения мы анализировали при выполнении лабораторной работы №2 из [1].

Атака с выбором шифртекста (chosen ciphertext attack). Криптоаналитик имеет возможность выбрать необходимое количество шифрограмм 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). Криптоаналитик имеет возможность выбирать новые шифрограммы для расшифрования с учетом того, что ему известна некоторая информация из предыдущих сообщений. Поскольку в некоторых криптографических протоколах при получении шифрограммы, несоответствующей стандарту (содержащей ошибки), отправитель получает ответное сообщение, иногда с детализированным описанием этапа проверки и причины возникновения ошибки. Криптоаналитик 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). Криптоаналитик 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* – криптоаналитики 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) – криптоаналитик шиф-

рует слова и фразы, наличие которых предполагается в шифрограмме, с использованием различных ключей; совпадение зашифрованных слов и фраз с частями шифрограммы может говорить о взломе ключа.

Адаптивная атака с выбором открытого текста (adaptive chosen plaintext attack). Криптоаналитик имеет возможность выбирать новые открытые тексты с учетом того, что ему известна некоторая информация из предыдущих сообщений – он может получить пары «открытое сообщение – шифрограмма», в т.ч. 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 криптоанализ* – 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 раундах для пары подобранных открытых сообщений при их зашифровании одним и тем же ключом;
- *интегральный криптоанализ* – аналогичен дифференциальному криптоанализу, но в отличие от него рассматривает воздействие алгоритма не на пару, а сразу на множество открытых текстов; предложен в 1997 г. Ларсом Кнудсеном;
- *линейный криптоанализ* – 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* – криптоаналитик имеет возможность получить шифртекст, соответствующий выбранному сообщению, на основе открытого ключа.

Атака на основе связанных ключей (related key attack). Криптоаналитик 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). Криптоаналитик 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 [5].

На рисунке 2.3 представлено основное диалоговое окно программы после запуска исполнительного файла *L_LUX.EXE*.

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.

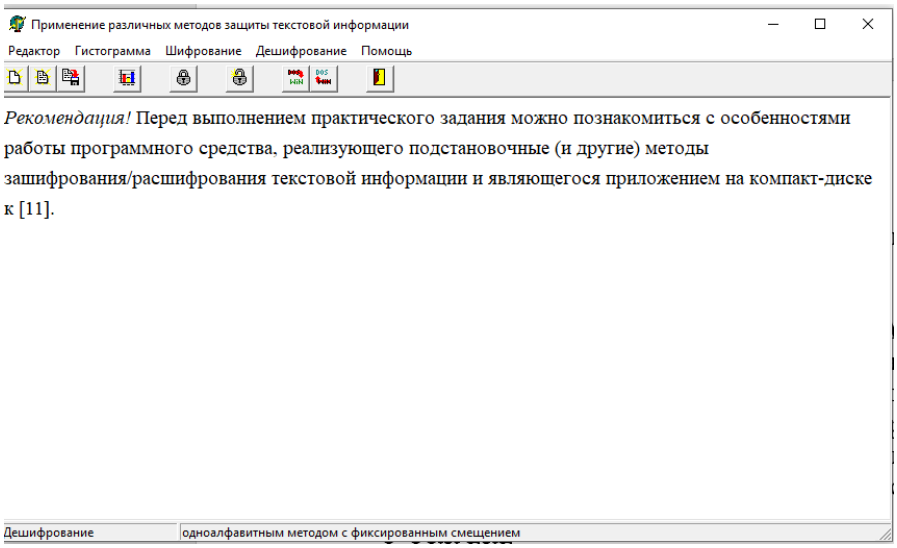


Рисунок 2.3 Основное диалоговое окно программного средства *L_LUX*

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

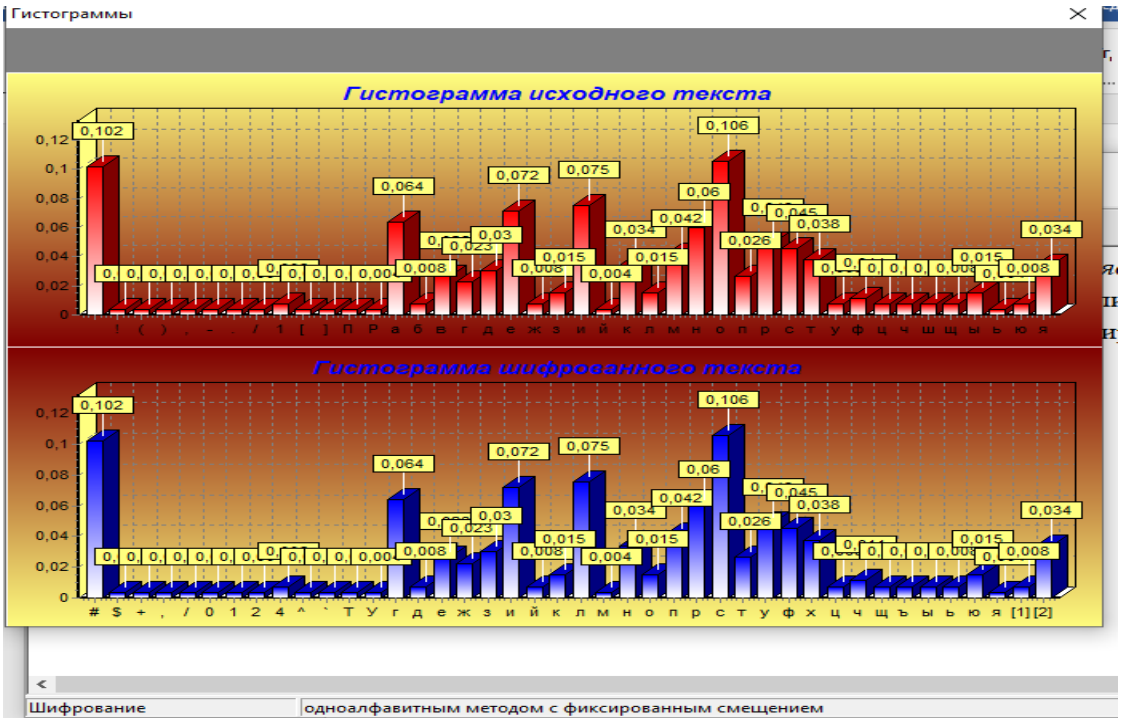


Рисунок 2.4 Гистограммы для исходного и зашифрованного текстовых документов

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

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
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

- 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 такового 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 [1].

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: ‘*byajhvfwbjyyfzgjcktljdfntkmyjcnm*’.

5. Расшифровать (с демонстрацией каждого шага алгоритма) текст C_i = = ‘*qrscqcocqclc*’, 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 Трисемуса для виртуального алфавита мощностью 40: 4x10? 10x4? 5x8? 8x5? 2x20? 20x2?

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 распространённый случай), пары, тройки букв и так далее.

Классическими примерами перестановочных шифров являются *анаграммы*. Анаграмма (от греч. ανα – «снова» и γράμμα – «запись») – литературный приём, 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 $J = j_1, j_2, \ldots, j_n$, где j_1 – номер позиции в зашифрованном сообщении первого символа открытого текста, где 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 шифрограммы.

Если предположить, что некоторое сообщение M_i состоит из букв от m_1 до m_n, то рассматриваемую таблицу можно представить как показано ниже (таблица 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 таблицы 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 таблиц вида 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 в форме табл. 3.1.

Таблица 3.2

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.

Таблица 3.3.

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 = «бвиркейныо».

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

Для расшифрования сообщения, следуя логике рассмотренных процедур зашифрования, нам нужно также составить таблицу, первой строкой которой будет зашифрованный текст (таблица 3.4.). Здесь применяется примерно такой же подход, как и в шифрах подстановки.

Таблица 3.4.

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

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

Таблица 3.5

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 «бвиркейныо» будет соответствовать обратная перестановка: символы первой строки таблицы 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 [1].

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-х 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

ки	бе	рв	ой	ны
5	1	4	2	3

В соответствии с табл. 3.6 получим C_i = «беойнырвки». 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

шифрограммы нужно записать символы этого сообщения в иной последовательности, т.е. по иному маршруту (см. аналогию с методами перемежения/деперемежения данных в лабораторной работе №7 [1]).

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

Рисунок 3.1 – Скитала [15]

Для зашифрования и расшифрования необходимо было иметь абсолютно одинаковые жезлы. На такой предмет наматывалась пергаментная лента. Далее на эту ленто 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 рисунке 3.5 в [2].

Следуя вышеприведенным рассуждениям, может отождествить скитала с таблицей размерами: 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:

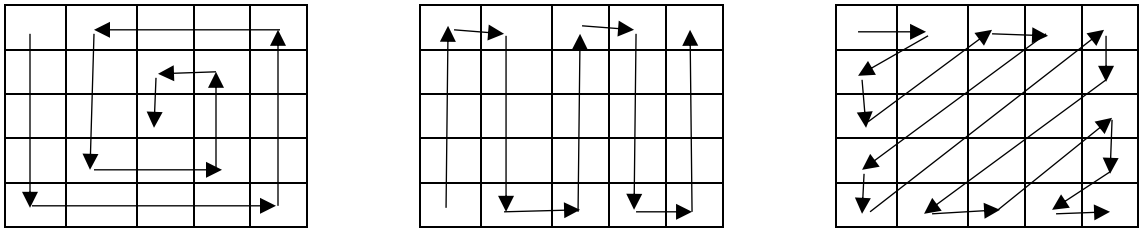
$$k = [(n - 1)/s] + 1. \qquad (3.1)$$

При этом слагаемое в квадратных скобках должно быть целым числом [15].

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

При использовании шифра Скитала для формирования шифртекста сначала выбирается 1-ая буква открытого текста, затем (k+1)-буква, (2k+1)-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,а), зигзагом (рис. 3.2,б), змейкой (рис. 3.2,в) или каким-то другим способом (см. рис. 3.2). Такие способы шифрования несколько усложняют процесс, однако усиливают криптостойкость шифра.



а) б) в)

Рисунок 3.2 – Графическое представление методов маршрутной перестановки

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 х 4.

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;
- шифрограмма выписывается по столбцам в соответствии с их нумерацией (ключом).

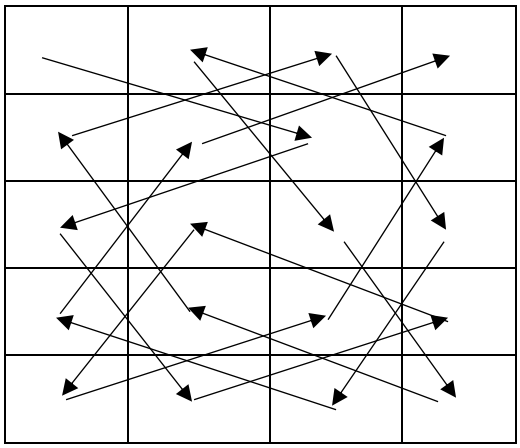


Рисунок 3.3 – Пример маршрута «обход конем»

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

Пример 4. Например, ключом является слово «крипто». Во-первых, 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 следующим: 2, 5, 1, 4, 6, 3.

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

Строим основную таблицу 5x6 (табл. 3.7), в которую по стокам будет записано исходное сообщение.

Считывая информацию из таблицы по столбцам в соответствии с ключом, получим шифрограмму C_i = «фтнрошелпава тириоевирьен кйск».

Таблица 3.7

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

3.1.2. Шифры множественной перестановки

Особенностью шифров данного подкласса является минимум двукратная перестановка символов шифруемого сообщения. В простейшем случае это может задаваться перемешиваем 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 информацию 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

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

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».

Таблица 3.10

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 шифрограмму C_i = «еиоврта ивртноферьнискйк 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 на другие.

Полезную информацию о классе рассмотренных шифров можно найти в [16, 17].

3.2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Рекомендация! Перед выполнением практического задания целесообразно освежить практические навыки использования и особенностями функционирования программного средства *L_LUX*, реализующего перестановочные (и другие) методы зашифрования/расшифрования текстовой информации и являющегося приложением на компакт-диске к [5].

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

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 такового в используемом языке можно воспользоваться разностью двух дат (например, в миллисекундах: время после выполнения программы – время до начала выполнения преобразования).*

Ниже представлен (Листинг 3.1) 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 += transposi-
tion[j];
            }
            return result;
        }
}
```

Листинг 3.1. 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 [1] 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).

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 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 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 (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 криптопреобразований 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 проведенных исследованиях, методики выполнения практической части задания и оценки криптостойкости шифров.

4.1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

4.1.1 Краткая историческая информация

Идея создания шифровального устройства высказана голландцем Гуго Кох де Дельфтю (в некоторых источниках – Гуго Александр Кох, Hugo Alexander Koch) еще в 1919 г. В 1920 году 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, которая получила название «Enigma» (от греч. – 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 году. Это была большая и тяжелая машина со встроенной пишущей машинкой и ве-

сом 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 году её принял на вооружение сначала военно-морской флот (модель *Funkschlussen* C), а в 1930-м – и Вермахт (*Enigma* I). Общее количество шифраторов, произведённых 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 году *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, состояла 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.

Рисунок 4.1 Одно из моделей (трехроторная) Энигмы [18]

Конкретный механизм мог быть разным, но общий принцип был таков: при каждом нажатии на клавишу самый правый ротор сдвигается на одну позицию, а при определённых условиях сдвигаются и другие роторы. Движение роторов приводит 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, «a») будет зашифрована другой буквой (например, «g»), а следующая за ней буква сообщения (также «a») – уже буквой «c».

а) б)

Рисунок 4.2 Упрощенная конструкция ротора (а) и рефлектора (б) Энигмы

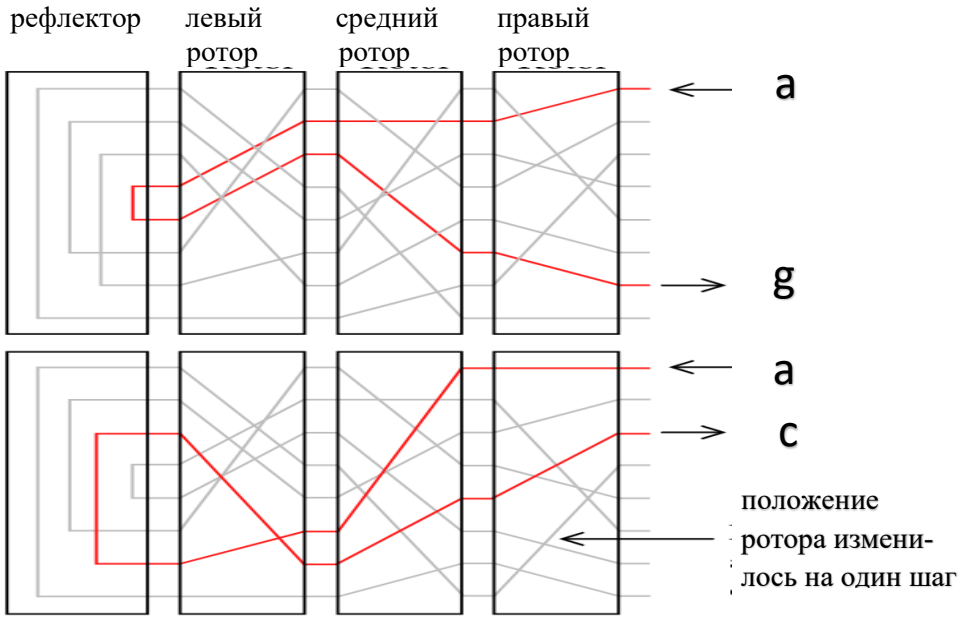


Рисунок 4.3 Пояснение к принципу шифрования путем формирования электрической цепи [19]

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.

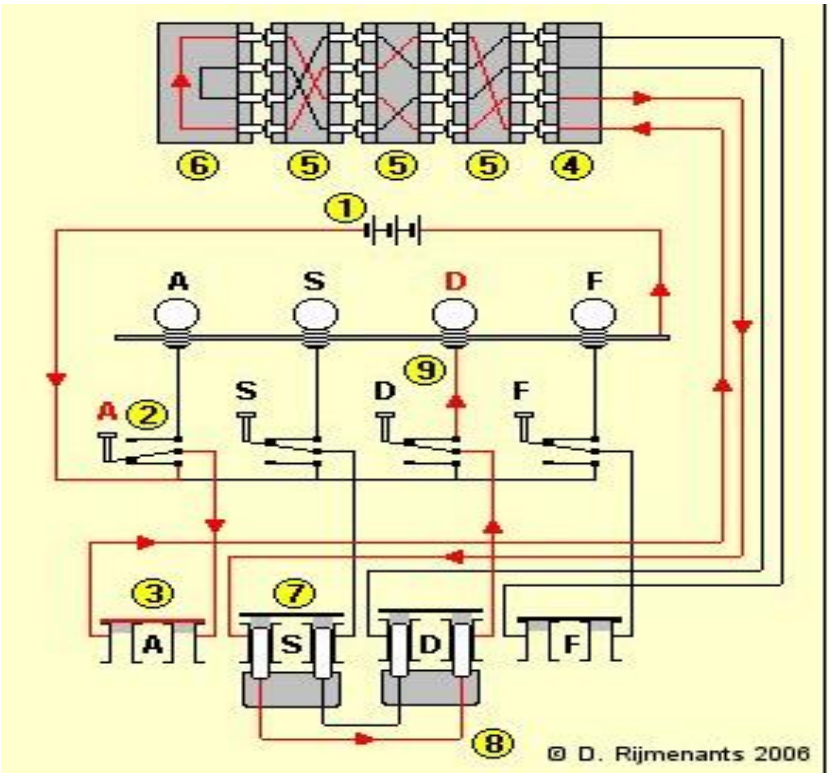


Рисунок 4.4 Пояснение к принципу формирования зашифрованного символа с помощью замкнутой электрической цепи [18]

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) 4 роторного модуля, 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 узлы 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. Зашифровывать ключ сообщения *SXT*. Предположим, что в результате зашифрования ключа получится *UHL*.

4. Далее оператор ставил 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. Расшифровывая 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 -ю 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* и вопросительный знак – словом *FRAGE* либо буквосочетанием *FRAQ*, а *Kriegsmarine M4* запятой соответствовала буква «*У*».

Как мы отмечали выше, Энигма строится на основе подстановочных шифров, подобных на шифр 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 подстановке, 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 [12, 21]. Ниже на рис. 4.5 и 4.6 представлены эти 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 [22].

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

Рисунок 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)

Рисунок 4.6 Спецификация на рефлекторы Энигмы

Рассмотрим пример использования приведенных спецификаций.

Пример 4.1. В этом примере мы процедуру только зашифрования 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 с этой страницы, 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»:

```
ABCDEF G HIJKLMNOPQRSTUVWXYZ
BDFHJL C PRTXVZNYEIWGAKMUSQO
```

,

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

```
AB C DEFGHIJKLMNOPQRSTUVWXYZ
AJ D KSIRUXBLHWTMCQGZNPYFVOE
```

,

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

```
ABC D EFGHIJKLMNOPQRSTUVWXYZ
EKM F LGDQVZNTOWYHXUSPAIBRCJ
```

.

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

```
ABCDE F GHIJKLMNOPQRSTUVWXYZ
YRUHQ S LDPXNGOKMIEBFZCWVJAT
```

.

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-M-R.

Эффект преобразования левого ротора (обратный):

```
ABCDEFGHIJKLMNOPQR S TUVWXYZ
UWYGADFPVZBECKMTHX S LRINQOJ
```

,

соответственно – среднего ротора (обратный):

```
ABCDEFGHIJKLMNOPQR S TUVWXYZ
AJPCZWRLFBDKOTYUQG E NHXMIVS
```

и, наконец, – правого ротора (обратный):

```
ABCD E FGHIJKLMNOPQRSTUVWXYZ
TAGB P CSDQEUFVNZHYIXJWLRKOM
```

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

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 [23], комментариев к функционалу Энигмы и ее симулятору [24]. Подробное описание кода симулятора на языке Pyton содержится в книге [25].

В начале данного параграфа

4.1.4 Оценка криптостойкости Энигмы

Для получения общего представления об особенностях работы криптоаналитиков над шифрами Энигмы полезно ознакомиться с содержанием материалов в [26].

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:

$$E_K = f\,(m, B, Re, L, M, R). \qquad (4.1)$$

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 [27]. 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 [2]) в нашем случае снижает уровень (теоретический) криптостойкости (до практического). Немецкие криптологи полагали, что один ротор может быть подключен 4×10^{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 – B, 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 x 4 x 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 x 26 x 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 x 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 коммутируются 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, предоставляя $(26 \times 25) / (1! \times 2^1)$ или 325 уникальных способов коммутаций одним кабелем. Для двух кабелей: есть (26 x 25) комбинаций – для первого кабеля и, поскольку два разъема уже используются, то получается (24 x 23) комбинаций – для второго кабеля. Следуя этой простой логике, получается $(26 \times 25 \times 24 \times 23) / (2! \times 2^2) = 44\,850$ уникальных способов коммутаций с использованием двух кабелей. Для трех кабелей – $(26 \times 25 \times 24 \times 23 \times 22 \times 21) / (3! \times 2^3) = 3\,453\,450$ комбинаций и так далее. 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 x 17 576 x 676 x 150 738 274 937 250 = 107 458 687 327 250 619 360 000 или $1{,}07 \times 10^{23}$.

Таким образом, практически рассматриваемая версия Энигмы (три ротора с выбором из 5 роторов, отражатель B и 10 штекерных кабелей для коммутационной панели) может быть настроена на $1{,}07 \times 10^{23}$ 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 *Krigsmarine Enigma M4*. Эта машина использует 3 обычных ротора, выбранных из набора из 8. Это, как мы уже отметили, дает 336 комбинаций подключений роторов (8 x 7 x 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 x 26 x 26 x 26). Рефлектор не меняется. Четвертый ротор был неподвижным. Версия *M4* была снабжена также 10 кабелями для коммутационной панели.

В сумме это дает: 336 x 2 x 456 976 x 676 x 150 738 274 937 250 = = 31 291 969 749 695 380 357 632 000 или $3{,}1 \times 10^{25}$, *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 $1{,}07 \times 10^{23}$ настроек (атака *brute force*) был невозможен в 1940-х годах, а его сопоставимый 77-битный ключ огромен даже для современных электронных систем. Чтобы дать представление о размере этого числа, представим, что у нас есть $1{,}07 \times 10^{23}$ 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, $1{,}07 \times 10^{23}$ дюйма равно 288 500 световых лет.

4.2 ПРАКТИЧЕСКОЕ ЗАДАНИЕ

1. Ознакомиться с функционалом хотя бы одного (по согласованию с преподавателем) симулятора Энигмы:
1.1 Симулятор Энигмы M3 (*M3* Enigma Simulator)
https://cryptocellar.org/simula/m3/index.html
1.2 Симулятор Энигмы M4 (*M4* Enigma Simulator)
https://cryptocellar.org/simula/m4/index.html
1.3 Симулятор Энигмы Army/Air Force and the Railway
https://cryptocellar.org/simula/enigma/index.html
1.5 Симулятор Энигмы для Абвера (*Abwehr* Enigma Simulator)
https://cryptocellar.org/simula/abwehr/index.html
1.5 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 таблиц на рис. 4.5 и 4.6 в соответствии со своим вариантом, представленным в таблице 4.1. 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 совершит один оборот.

Таблица 4.1

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 Энигма.
2. На основе каких шифров строится машина Энигма?
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 Энигмы.
7. Ваши предложения по модификации известных аналогов Энигмы?

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 Standarts & Technology – NIST) инициировал программу защиты каналов связи и компьютерных данных. Одна из целей – разработка единого стандарта криптографического шифрования. Основными критериями оценки алгоритма являлись [4]:

- 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 применениям,
- 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 1971 г. Хорстом Фейстелем (Horst Feistel) устройства, реализующие различные алгоритмы шифрования,

позже получившие *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 году были разработаны алгоритмы FEAL и RC2. Сети Фейстеля получили широкое распространение в 1990-е годы — в годы появления таких алгоритмов, как *Blowfish* (1993), TEA (1994), RC5 (1994), CAST-128 (1996), XTEA (1997), XXTEA (1998), RC6 (1998) и других. На основе сети Фейстеля в 1990 году 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 20 века. 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, 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 реализуемыми 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 бит).

Полученные блоки называются *входными*. В случае, если длина входного блока меньше, чем выбранный размер, то 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-ом раунде выполняются преобразования в соответствии с формальным представлением ячейки сети Фейстеля:

$$\begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} + f(R_{i-1},\ K_i), \end{aligned} \qquad (5.1)$$

По какому-либо математическому правилу вычисляется раундовый ключ K_i. В приведенном выражение знак «+» соответствует поразрядному суммированию на основе «XOR». На рис. 5.1 приведено 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 одноразрядное сигнал по основанию 2^n;
- внутреннего коммутатора;

- шифратора, преобразующего сигнала из одноразрядного 2^n-ричного в n-разрядный двоичный.
  Пример реализации трехразрядного S-блок показан на рис. 5.2.

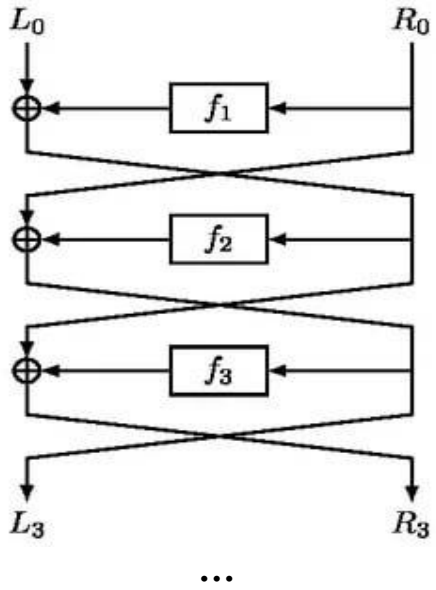


Рисунок 5.1 Графическое отображение сети Фейстеля

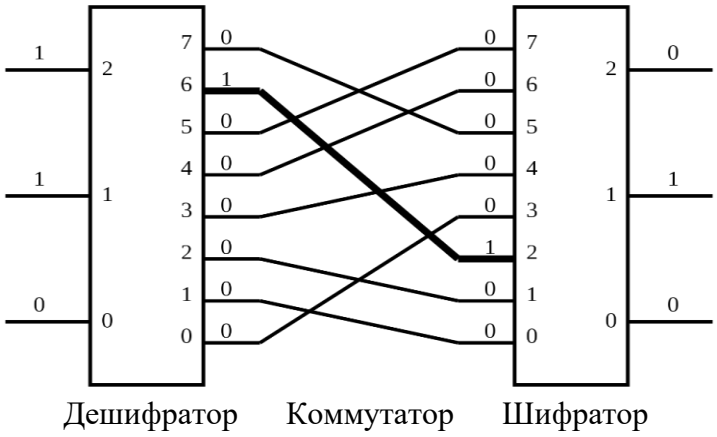


Рисунок 5.2 Пример реализации трехразрядного S-блока

Блок перестановок изменяет положение цифр, т. е. является линейным устройством. Этот блок может иметь очень большое количество входов-выходов, однако в силу линейность является слабым местом преобразования с

точки зрения криптостойкости. На рис. 5.3 приведен пример реализации 8-миразрядного P-блока.

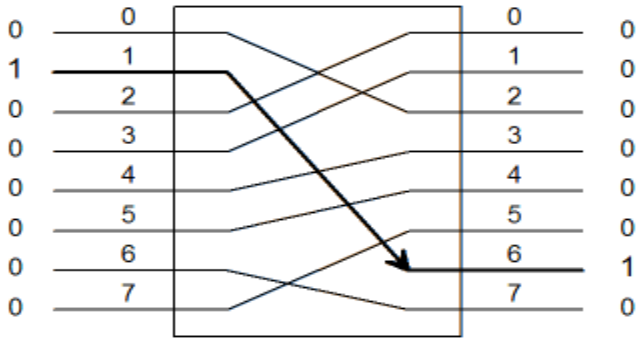


Рисунок 5.3 Пример реализации 8-миразрядного P-блока

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

При большом размере блоков (128 бит и более) реализация сети Фейстеля на 32-разрядных архитектурах может вызвать затруднения, поэтому применяются модифицированные варианты сети. Такие модификации предусматривают использование не 2-х (*L* и *R* на рис. 5.1), а 4-х ветвей. На рис. 5.4 показаны некоторые модификации.

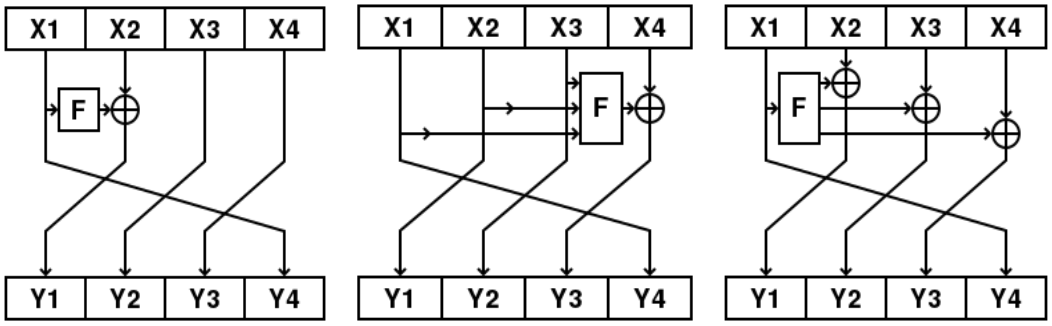


Рисунок 5.4 Примеры модификаций базовой сети Фейстеля

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

Рассеивание подразумевает, что каждый символ (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 2-х ($A + B$) 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-разрядных.

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

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

Во-вторых. Результатом сложения также должно быть n-разрядное число.

В-третьих. Побитовое сложение предусматривает известную взаимосвязь между соседними символами (порядками).

В-пятых. В силу известных правил модулярной1 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 (mod 16) = 0.

Представим слагаемые в двоичной форме: A = 1001, B = 0111; $A + B$ =1001 + 0111 = 1**0000**. Для получения нужного результата – вычисления 9 + 7 (mod 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 (mod 16) = 14.

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

Пример 3. Пусть n=4, A = 10, B = 11. Их сумма по модулю 16 дает 5. В двоичной форме: 1010 + 1011 = 1**0101**. 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:

$$A + B \pmod{2^n} = \left\{ \begin{array}{ll} A + B, & \text{если } A + B < 2^n \\ A + B - 2^n, & \text{если } A + B \geq 2^n \end{array} \right\}.$$

5.1.4 Некоторые блочные алгоритмы

5.1.3.1 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 бит, 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 [2].

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

Таблица 5.1

Начальная перестановка

58	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	**1**	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 криптоанализа статистическими методами.

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

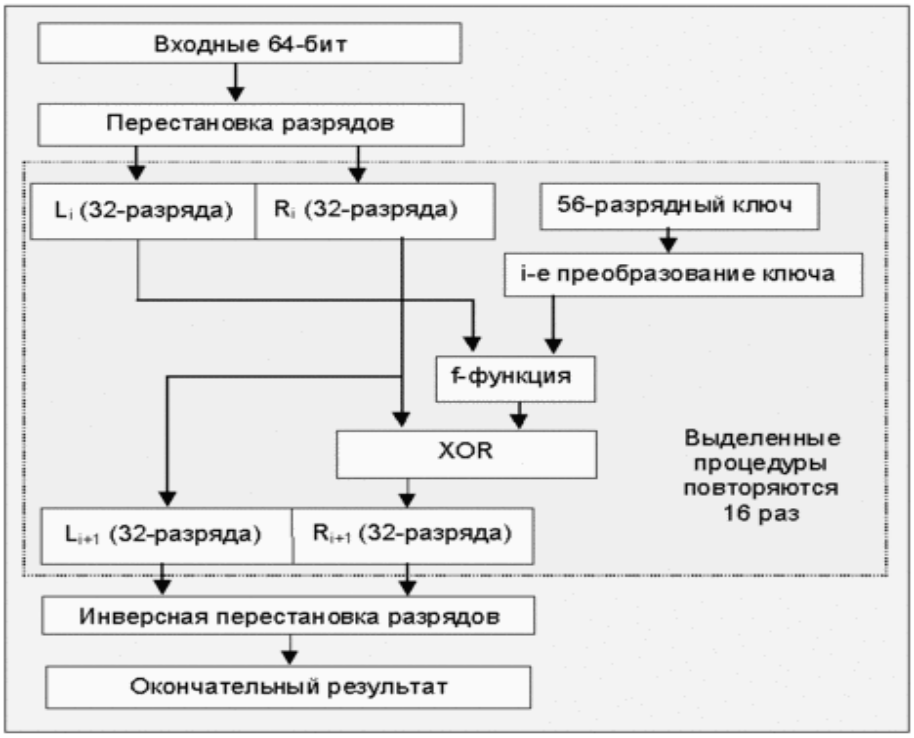


Рисунок 5.5 Общая схема алгоритма DES

Вначале правая часть блока R_i расширяется до 48 битов, используя таблицу, 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 «*лавинным эффектом*»).

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. *S*ubstitution – 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 этапом 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 единое 32-битное значение.

Далее полученное 32-битное значение обрабатывается с помощью перестановки *P* (от англ. *P*ermutation – перестановка), которая не зависит от используемого ключа. Целью перестановки является максимальное переупорядочивание битов такое, 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-битового блока данных. Затем левая и правая половины меняются местами, и начинается следующий раунд (см. рис. 5.6).

После выполнения 16-раундового зашифрования 64-битного блока данных осуществляется конечная перестановка (IP^{-1}). Она является обратной к перестановке *IP*. Конечная перестановка определяется таблицей 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	**1**	41	9	49	17	57	25

Рисунок 5.6. Схема реализации функции f на рис. 5.5

Каждый 8-й бит исходного 64-битного ключа отбрасывается. Эти 8 битов, находящих 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 [1]). 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* [4]. 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 [4]

Слабые ключи DES

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.

Таблица 5.4 [4]

Полуслабые ключи DES

01FE	01FE	01FE	01FE	и	FE01	FE01	FE01	FE01
1FE0	1FE0	0EF1	0EF1	и	E01F	E01F	F10E	F10E
01E0	01E0	01F1	01F1	и	E001	E001	F101	F101
1FFE	1EEE	0EFE	0EFE	и	FE1F	FE1F	FE0E	FE0E
011F	011F	010E	010E	и	1F01	1F01	0E01	0E01
E0FE	E0FE	F1FE	F1FE	и	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 [4], 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 Internet в единую сеть 100 тыс. персональных компьютеров за 24 часа. Данному алгоритму присуща проблема так называемых «слабых» и «частично слабых» ключей [4].

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. Создан У. Диффи, М. Хеллманом, У. Тачманном в 1978 г.

Формальная запись:

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

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) – позволяет выполнять операции шифрования и цифровой подписи сообщений, файлов и другой информации, представленной в электронном виде, например, на жёстком диске); в S/mime 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).

5.1.3.2 Стандарт AES

AES (Advanced Encryption Standard) – алгоритм шифрования, действующий в качестве государственного стандарта в США с 2001 года. В основу стандарта положен шифр *Rijndael*. Шифр *Rijndael/AES* (то есть рекомендуемый стандартом) характеризуется размером блока 128 бит, длиной ключа 128, 192 или 256 бит 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.

5.1.3.3 Стандарт ГОСТ 28147-89

В Советском Союзе, в том числе – в Беларуси и в России в качестве стандарта на блочные алгоритмы шифрования с закрытым ключом в 1989 г. был принят ГОСТ 28147-89 [30].

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 бит, здесь используются 32 раунда шифрования

Таким образом, основные параметры алгоритма криптографического преобразования данных ГОСТ 28147-89: размер блока составляет 64 бита, размер ключа – 256 бит, 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).

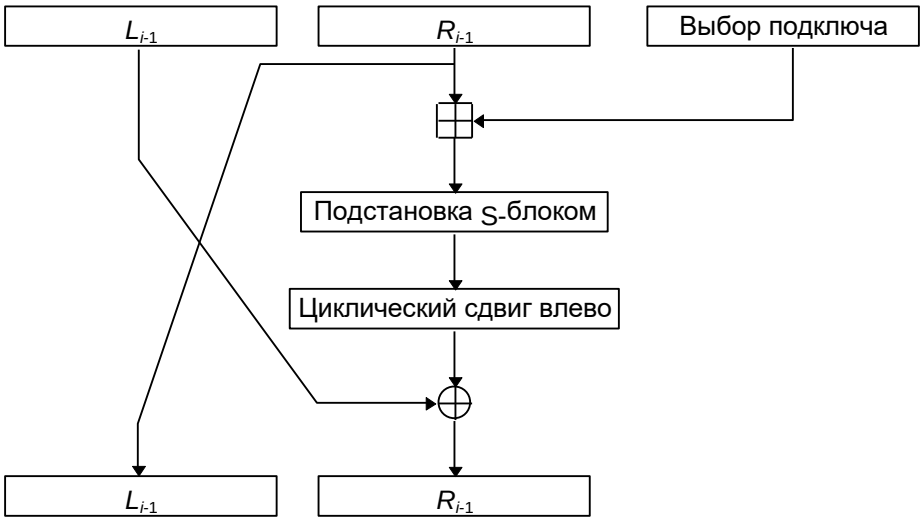


Рисунок 5.7 Структура алгоритма реализации раунда в стандарте ГОСТ 28147-89 (знаком «+» в квадратной рамке обозначена операция сложения по модулю 2^{32})

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 бит;
- 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-битовый 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 также 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. Однако 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; DES для этого нужно только 5 раундов.

5.1.3.4 Алгоритм Blowfish

Основой алгоритма является сеть Фейстеля с 16 раундами. Длина блока равна 64 битам, ключ может иметь любую длину в пределах 448 бит. 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 боков, чем DES.

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

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

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

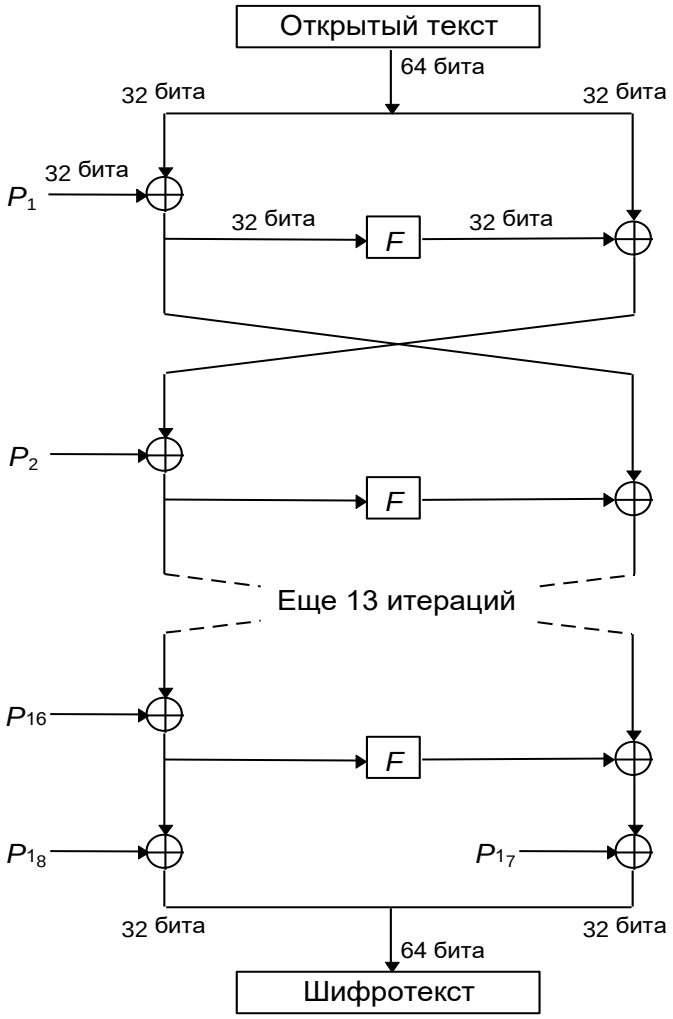


Рисунок 5.8 Структура алгоритма Blowfish

Элементы алгоритма:

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

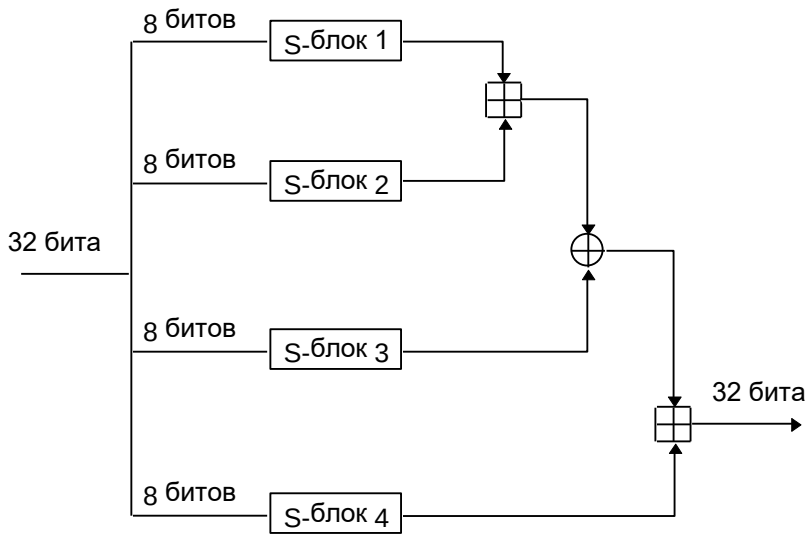


Рисунок 5.9 Структура функции *F* алгоритма Blowfish

Успешные атаки на рассмотренный алгоритм не известны.

Дополнительные сведения по рассмотренным вопросам можно найти в [8].

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.

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 [4].

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 нижеследующей табл. 5.5.

Таблица 5.5

Вариант задания	Алгоритм	Ключ
1	DES	Первые 8 символов собственных *фамилииимени*
2	DES-EEE3	а) Информац*ионнаябе*зопаснос* б) лаборато*рная--ра*бота__№5* *каждый из трех ключей выделен шрифтом
3	DES-EDE3	По указанию преподавателя
4	DES-EEE2	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 криптоанализа?

15. В чем состоит сущность линейного криптоанализа?

16. Какие ключевые комбинации относятся к слабым (к полуслабым) и почему?

17. Где применяются блочные криптоалгоритмы?

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 m_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 [2], а также в [4].

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.

Синхронные потоковые шифры (СПШ) характеризуются тем 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 отрытый текст на отрезки, начало и конец которых выделить вставкой контрольных меток (специальных маркеров).

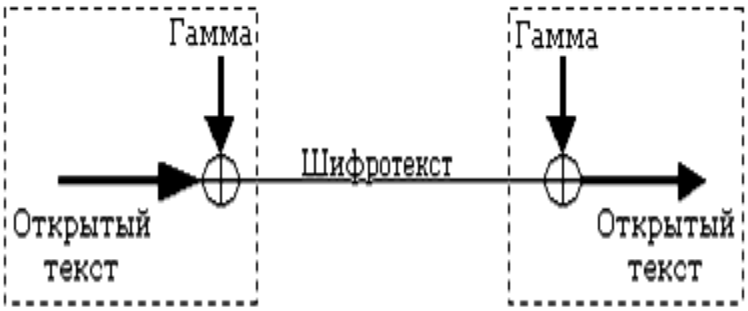


Рисунок 6.1 Схема потокового шифра

Синхронные потоковые шифры уязвимы к *атакам на основе изменения отдельных бит 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) вырабатываются специальными блоками систем потокового шифрования – генераторами. В РБ в настоящее время действует стандарт СТБ 34.101.47-2017 "Информационные технологии и безопасность. Алгоритмы генерации псевдослучайных чисел" [31].

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:

$$x_{t+1} = (a*x_t + c) \bmod n, \qquad (6.1)$$

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 ПСЧ по нескольким ее значениям.

Генератор практически не используются в криптографии в силу низкой криптостойкости. Тем не менее, полезны 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 [4].

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).

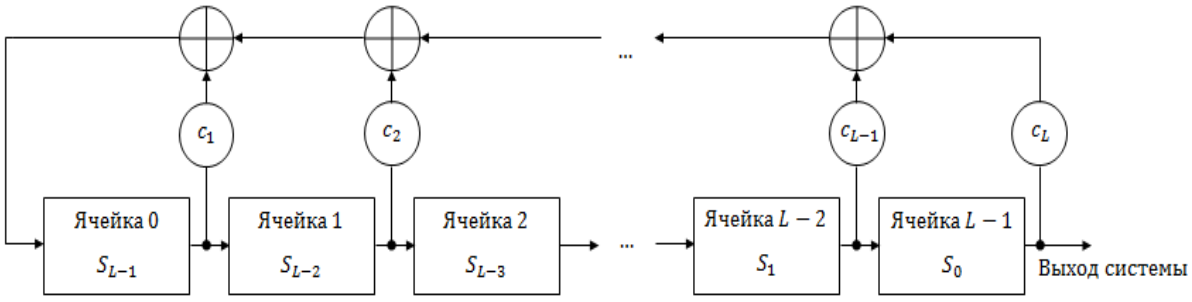


Рисунок 6.2 Общая схема регистра сдвига с линейной обратной связью

РСЛОС строятся на основе *примитивных порождающих полиномов* (*многочленов*), которые мы подробно анализировали при изучении циклических помехоустойчивых кодов [1] (см. также [32, 33]). 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

Собственно 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

$$x_t = (x_{t-1})^e \mod n. \qquad (6.2)$$

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

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

Безопасность генератора опирается на сложности взлома алгоритма RSA, т. е. на разложении 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 вычетов*. Для целей криптографии этот метод предложен в 1986 г.

Начальное значение x_0 генератора вычисляется на основе соотношения

$$x_0 = x^2 \mod n, \qquad (6.3)$$

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:

$$x_t = (x_{t-1})^2 \mod n. \qquad (6.4)$$

Рассмотрим пример.

Пример. Пусть $p = 11$, $q = 19$ (убеждаемся, что 11 mod 4 = 3, 19 mod 4 = 3). Тогда $n = p* q = 11*19=209$. Выберем x, взаимно простое с n: пусть $x = 3$ [34].

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

$$x_0 = x^2 \bmod n = 3^2 \bmod 209 = 9 \bmod 209 = 9.$$

Вычислим первые десять чисел x_t по алгоритму BBS.

В качестве случайных бит будем брать младший бит в двоичной записи числа x_t (табл. 6.1).

Таблица 6.1

Пояснение к методу генерации ПСП на основе метода BBS

Число x_t	Младший бит двоичного числа x_t
$x_1 = 9^2 \bmod 209 = 81 \bmod 209 = 81$	1
$x_2 = 81^2 \bmod 209 = 6561 \bmod 209 = 82$	0
$x_3 = 822 \bmod 209 = 6724 \bmod 209 = 36$	0
$x_4 = 36^2 \bmod 209 = 1296 \bmod 209 = 42$	0
$x_5 = 42^2 \bmod 209 = 1764 \bmod 209 = 92$	0
$x_6 = 92^2 \bmod 209 = 8464 \bmod 209 = 104$	0
$x_7 = 104^2 \bmod 209 = 10816 \bmod 209 = 157$	1
$x_8 = 157^2 \bmod 209 = 24649 \bmod 209 = 196$	0
$x_9 = 196^2 \bmod 209 = 38416 \bmod 209 = 169$	1
$x_{10} = 169^2 \bmod 209 = 28561 \bmod 209 = 137$	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:

$$x_t = (x_0)^a \bmod n, \qquad (6.4)$$

где $a = 2^t \bmod ((p-1)(q-1))$.

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

6.1.3 Потоковый шифр RC4

Алгоритм RC4 разработан Р. Ривестом в 1987 г. Представляет собой потоковый шифр с переменным размером ключа. Здесь гамма не зависит от открытого текста [4].

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 бит.

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 *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: $S_0 = 0$, $S_1 = 1$, …, $S_{255} = 255$. Затем заполняется секретным ключом другой 256-байтный массив. Если необходимо, ключ повторяется многократно, чтобы заполнить весь массив: K_0, K_1, …, 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 + Si + Ki) mod 256;
3. поменять местами Si и Sj;
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 [35].

Пример. 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

№ 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

№ 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 алг.	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) mod 16= (3 + 2 + 3) mod 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) mod 16= (8 + 0 + 4) mod 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) mod 16= (12 + 4 + 5) mod 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) mod 16= (5 + 4 + 6) mod 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) mod 16= (15 + 6 + 1) mod 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) mod 16= (6 + 7 + 2) mod 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		
2	j = (j + S_i + K_i) mod 16= (15 + 2 + 3) mod 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) mod 16= (4 + 9 + 4) mod 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) mod 16= (1 + 10 + 5) mod 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) mod 16= (0 + 11 + 6) mod 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) mod 16= (1 + 0 + 1) mod 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) mod 16= (2 + 13 + 2) mod 16 = 1		1
3	Поменять местами S_i и S_j, т. е. S13 и S1		
4	i = i +1	14	
5	i < 16, поэтому перейти на п.2		
2	j = (j + S_i + K_i) mod 16= (1 + 14 + 3) mod 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) mod 16= (2 + 7 + 4) mod 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

№ 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 знач. i	Новое знач. j	Новое знач. a
K_1	1. i = (i + 1) =0+1=1	1		
	2. j = (j + S_i) mod 16=(0+13) mod 16=13		13	
	3. Поменять местами S_1 и S_{13}			

	4. a = (S_i + S_j)mod 16=(7+13)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) mod 16=(13+14) mod 16=11		11	
	3. Поменять местами S_2 и S_{11}			
	4. a = (S_i + S_j)mod 16=(9+14)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) mod 16=(11+12) mod 16=7		7	
	3. Поменять местами S_3 и S_7			
	4. a = (S_i + S_j)mod 16=(4+12)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) mod 16=(7+2) mod 16=9		9	
	3. Поменять местами S_4 и S_9			
	4. a = (S_i+ S_j)mod 16=(3+2)mod 16=5			5
	5. **K_4= S_5=15**			
K_5	1. i = (i + 1) =4+1=5	5		
	2. j = (j + S_i) mod 16=(9+15) mod 16=8		8	
	3. Поменять местами S_5 и S_8			
	4. a = (S_i+ S_j)mod 16=(5+15)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 [36].

По утверждению [4] рассмотренный алгоритм устойчив к линейному и дифференциальному криптоанализу. Кроме того, при n = 8 этот алгоритм может находиться примерно в 2^{1700} состояниях (256!*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 65536-элементный массив.

Шифр RC4 применяется в некоторых широко распространённых 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') == \

'54901b4755467cff141740c496492fee8ba7224a99f52cc2e84d019e1c0cda32b6a
96d521d067d00ce6716'
assert
RC4crypt('\xA4\x8F\x9B\xFF\x6D\x3A\xFD\x7D\x56\xCB\xAC\xD6\x46\x27\x
50\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 [4], 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. 6.6.

Таблица 6.6

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 = 11979

* – можно воспользоваться результатами выполнения ЛР №1

Приложение 2 должно реализовывать алгоритм RC4 в соответствии с вариантом из табл. 6.7, а также дополнительно выполнять оценку скорости выполнения операций генерации ПСП.

Таблица 6.7

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 (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) 84320. 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 криптосистем

Две известные нам проблемы, связанные с практическим использованием симметричных криптосистем, 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. Стал первым алгоритмом шифрования с открытым ключом широкого назначения.

Определение 2. *Ранцевый* (*рюкзачный*) *вектор* $\boldsymbol{S} = (s_1, \ldots, s_z)$ – это упорядоченный набор из z, $z \geq 3$, различных натуральных чисел s_i. Входом задачи о

ранце (рюкзаке) называем пару ($\boldsymbol{S}$, S), где $\boldsymbol{S}$ – рюкзачный вектор, а S – натуральное число.

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

В наиболее известном варианте задачи о ранце требуется выяснить, обладает или нет данный вход ($\boldsymbol{S}$, S) решением. В варианте, используемом в криптографии, нужно для данного входа ($\boldsymbol{S}$, 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 $s_1, s_2, \ldots, s_z$ и суммарное значение S. Требуется вычислить значения s_i такие, что

$$S = b_1 s_1 + b_2 s_2 + \ldots + b_z s_z. \qquad (7.1)$$

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 $\boldsymbol{e}$, который легко использовать для зашифрования, но невозможно – для расшифрования. В качестве закрытого ключа $\boldsymbol{d}$ применяется легкий для укладки ранец, который предоставляет простой способ расшифрования сообщения.

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

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} ($z = 8$) 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 ($b_z = 1$) 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. Он меньше 270, поэтому предмет весом 210 кладут в ранец (1): вычитают 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 бы использован для расшифрования, то открытый текст, полученный из значения шифртекста 270, был бы равен 10100101.

Открытый ключ $\boldsymbol{e}$ 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 $\boldsymbol{e}$ ($\boldsymbol{e} = \{e_i\}$, $i = 1,\ldots, z$) все значения закрытого ключа умножаются на некоторое число a по модулю n:

$$e_i = d_i\, a \pmod{n}. \qquad (7.2)$$

Значение модуля n должно быть больше суммы всех чисел последовательности; кроме того, 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 $\boldsymbol{d}$: $\boldsymbol{d} = \{d_i\}$. Примем, что $n = 420$ и a=31.

В соответствии с этими при использовании (7.2) результат построения нормальной последовательности (открытого ключа, $\boldsymbol{e}$) 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, $i = 1, \ldots, z$): по одному ранцу для каждого блока сообщения с использованием открытого ключа получателя, $\boldsymbol{e}$.

*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 нижеследующей таблицы (табл. 7.1).

Открытый ключ, $\boldsymbol{e}$: {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 7.1.

Таблица 7.1

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 (использует свой тайный ключ, ***d***: сверхвозрастающую последовательность) должен сначала определить обратное к a число: a^{-1}, такое что

$$a\ a^{-1} (\text{mod}\ n) = 1. \qquad (7.3)$$

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:

$$S_i = c_i\ a^{-1} \text{ mod } n. \qquad (7.4)$$

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*271 mod 420 = 1.

Вспомним, что сверхвозрастающая последовательность равна ***d***: ***d***= {2, 3, 6, 13, 27, 52, 105, 210}, а также n = 420, a = 31; шифртекст: 155 365 558 155 924 1239 470

Расшифрование первого блока шифртекста. Сначала вычисляем, используя (7.4), вес первого ранца (при c_1=155):

$$S_1 = c_1 * a^{-1} (\text{mod}\ n) = 155*271\ (\text{mod}\ 420) = 5.$$

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 криптоалгоритма 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, известный способ определения, 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 [4]. 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 криптоаналитика рассмотрение криптостойкости ранцевого алгоритма изложено в [37].

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 колировки.

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 (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 $\boldsymbol{S}$ = (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.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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: учебно-методическое пособие для студентов учреждений высшего образования / П. П. Урбанович, Д. В. Шиман, Н. П. Шутько. – Минск: БГТУ, 2019. – 116 с.
(Ресурс удаленного доступа: https://elib.belstu.by/handle/123456789/29372)

2. Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации : учеб.-метод. пособие для студ. спец. 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем», направления спец. 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», спец. 1-40 01 01 «Программное обеспечение информационных технологий» спец. 1-40 01 01 10 «Программирование Интернет-приложений». - Минск: БГТУ, 2016. - 220 с. (Ресурс удаленного доступа: https://elib.belstu.by/handle/123456789/23763)

3. Как определить простое число, Ресурс удаленного доступа:https://www.kakprosto.ru/kak-66290-kak-opredelit-prostoe-chislo#ixzz60MRJ4m5m

4. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си/ Б. Шнайер. – М.: Издательство Триумф, 2003. – 816 с.

5. Бабаш, А. В. Информационная безопасность. Лабораторный практикум: уч. пособие/ А. В. Бабаш, Е. К. Баранова, Ю. Н. Мельникова. – 2-е изд., стер. – М.: КНОРУС, 2013. – 136 с.

6. Жуан Гомес. Мир математики. Т.2: Математики, шпионы и хакеры. Кодирование и криптография. – М.: Де Агостини, 2014. – 144 с.
(Ресурс удаленного доступа: https://www.litmir.me/br/?b=247091&p=1)

7. Национальный корпус русского языка,Ресурс удаленного доступа:http://www.ruscorpora.ru/new/corpora-intro.html

8. Урбанович, П. П. Защита информации: конспект-лекция. Ч. 7 = Information Protection, Part 7: Basic cryptographic algorithms and standards / П. П. Урбанович. - Минск: БГТУ, 2019. – 58 с. –наангл. яз.
(Ресурс удаленного доступа: https://elib.belstu.by/handle/123456789/29342)

9. Риксон, Фред Б. Коды, шифры, сигналы и тайная передача информации / Фред Б. Риксон - М.: АСТ: Астрель, 2011. – 656 с.

10. LeoneBaptistaAlberti. Decomponendiscyfris, Ресурс удаленного доступа:http://www.apprendre-en-ligne.net/crypto/alberti/decifris.pdf

11.Шифры замены, Ресурс удаленного доступа:
https://www.sites.google.com/site/anisimovkhv/learning/kripto/lecture/tema4#tabl41

12. Бабаш, А. В. Криптография. Под. ред. В. П. Шерстюка, Э. А. Применко / А. В. Бабаш, Г. П. Шанкин. – М.: СОЛОН-ПРЕСС, 2007. – 512 с.
13. Кан, Д. Взломщики кодов / Д. Кан – М.: Центрполиграф, 2000. – 594 с.
14. Н. А. Гатченко, А. С. Исаев, А. Д. Яковлев – Криптографическая защита информации, уч. пособие, Санкт-Петербург – 2012.
15. Скитала, Ресурс удаленного доступа:https://ru.wikipedia.org/wiki/Скитала
16. Шифры перестановки, Ресурс удаленного доступа: https://www.sites.google.com/site/anisimovkhv/learning/kripto/lecture/tema5
17. Перестановочные шифры, Ресурс удаленного доступа: http://nozdr.ru/games/quest/crypt/cipher/perestanovka
18. Шифровальная машина Энигма, Ресурс удаленного доступа: http://www.adsl.kirov.ru/projects/articles/2018/06/24/enigma/
19. Алгоритм Энигмы, Ресурс удаленного доступа: https://habr.com/ru/post/217331/
20. Technical Details of the Enigma Machine, Ресурс удаленного доступа: http://users.telenet.be/d.rijmenants/en/enigmatech.htm#rotors
21. Technical Specification of the Enigma, Ресурс удаленного доступа: http://www.codesandciphers.org.uk/enigma/rotorspec.htm
22. Enigma Timeline. Under construction, Ресурс удаленного доступа: https://www.cryptomuseum.com/crypto/enigma/timeline.htm
23. EnigmaMashinen, Ресурс удаленного доступа: https://yandex.by/video/prview/?filmId=4334530940425969248&noreask=1&path=wiard&text=симуляторы+Энигмы
24. JorgeLuisOrejel. Enigma, Ресурс удаленного доступа: https://www.codeproject.com/Articles/831015/ENIGMA
25. BrianNeal. Py-Enigma Documentation. Release 0.1, 2017. – 29 p. (Ресурсудаленногодоступа:https://readthedocs.org/projects/py-enigma/downloads/pdf/latest/)
26. Криптоанализ «Энигмы», Ресурс удаленного доступа: https://ru.wikipedia.org/wiki/Криптоанализ_«Энигмы»
27. A. Ray Miller. The Cryptographic Mathematics of Enigma, Center for Cryptologic History National Security Agency, 2019
(Ресурс удаленного доступа: https://drive.google.com/file/d/1By1nea1BhIiNwCfykdmQAawkyh5QT_hr/view)
28. Horst Feistel, Cryptography and Computer Privacy// Scientific American, May 1973, Volume 228, No 5, pp. 15-23. (Ресурс удаленного доступа: http://www.apprendre-en-ligne.net/crypto/bibliotheque/feistel/)
29. Петров, А.А. Компьютерная безопасность. Криптографические методы защиты / А.А. Петров.– М.: ДМК, 2000. – 448 с.

30. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования ГОСТ 28147-89. – М.: Издательство стандартов, 1989 – 26 с.
31. Информационные технологии и безопасность. Криптографические алгоритмы генерациипсевдослучайных чисел. Государственный стандарт Республики Беларусь СТБ 34.101.47-2017. – Минск: Госстандарт, 2017. – 22 с. (Ресурс удаленного доступа: http://apmi.bsu.by/assets/files/std/brng-spec24.pdf).
32. Урбанович, П.П. Защита информации и надежность информационных систем: пособие для студентов / П.П. Урбанович, Д.В. Шиман. – Минск:БГТУ, 2014. – 90 с. (Ресурс удаленного доступа: https://elib.belstu.by/handle/123456789/23761).
33. Урбанович, П. П. Защита информации: конспект-лекция. Ч. 5 = Information Protection, Part 5: Error Correcting Codes / П. П. Урбанович. –Минск: БГТУ, 2019. – 34 с. (Ресурсудаленногодоступа: https://elib.belstu.by/handle/123456789/29340).
34.Генератор псевдослучайных чисел на основе алгоритма BBS, Ресурс удаленного доступа:https://www.intuit.ru/studies/courses/691/547/lecture/12383?page=3.
35. Алгоритм RC4, Ресурс удаленного доступа:https://www.intuit.ru/studies/courses/691/547/lecture/12385?page=2.
36. Алгоритм RC4, Ресурс удаленного доступа: https://ru.wikibooks.org/wiki/Реализации_алгоритмов/RC4
37. Саломаа, А. Криптография с открытым ключом/ А. Саломаа: Пер. с англ. – М.: Мир, 1995. – 318 с. (Ресурс удаленного доступа: https://web.archive.org/web/20111119210028/http://www.ssl.stu.neva.ru/psw/crypto/open_key_crypt.pdfhttps://web.archive.org/web/20111119210028/http://www.ssl.stu.neva.ru/psw/crypto/open_key_crypt.pdf)