МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

ФАКУЛЬТЕТ ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ

Кафедра Криптология

Методическое пособие

по выполнению практических работ учебной дисциплины Криптография 1 для студентов направлении информационной безопасности

Ташкент 2021

Авторы: Ассистенты кафедра “Криптология” Хамидов Ш.Ж, Тоджиакбарова У.У., Каримов А.А. методическая пособия по выполнению практических работ по учебной дисциплине «Криптографии 1» для студентов бакалавра специальностей информационной безопасности - Ташкент: ТУИТ. 2021.-72 с.

Данной методической пособие представлен материал, необходимый для введения в теорию криптографичесих алгоритмов, математическим фундаментом которых является прикладная теория чисел. Это в первую очередь теория групп, теория колец и теория полей. Рассмотрены крипто системы с секретным ключом (симметричные или классические), криптосистемы с открытым ключом (асимметричные), методы криптоанализа, а также хеш-функции. Кроме того, представлены основные положения криптографического протокола "электронная подпись". В каждом разделе приведены примеры на соответствующие темы.

Рекомендовано к печати решением учебно-методического совета ТУИТ имени Мухаммада ал-Хоразмий (протокол № 10/3 от “25” “ мая ” 2021 г.)

Ташкентский университет информационных технологий имени Мухаммада аль-Хорезми, 2021

1-практическая работа

Тема: Математические основы криптографии

Цель работы: приобрести теоретические и практические навыки о свойствах взаимно простых чисел и действии модуля, системе счисления, логических действиях.

Теоретическая часть

Элементы теории чисел: основные понятия и теоремы.

В дальнейшем изложении будем использовать следующие понятия и обозначения:

N – множество натуральных чисел: целые положительные числа вида 1,2,...

Z – множество целых чисел: числа вида n, -n и 0, где n – натуральное число.

Q – множество рациональных чисел: числавида p/q, где p и q – целые числа и 0≠q. Класс рациональных чисел включает в себя все целые числа Z, которые в свою очередь включают в себя все натуральные числа N.

R – множество действительных чисел: этот класс включает все рациональные числа и все числа не являющиеся таковыми т.е. все иррациональные числа. Рассмотрим множество целых чисел, которые обозначим через Z. На множестве целых чисел рассмотрим операции сложения + и умножения ×. Операция деления, обратная операции умножения, выполняется не для всех пар чисел из Z. (Напомним Z – целые числа).

Число a делится на число b ≠ 0, если существует число q такое, что

$$\frac{a}{b} = q \text{ или } a = b * q$$

В этом случае говорят, что число b делит число a. При этом число b - делитель числа a, число a – кратное числа b, число q - частное от деления a на b. Число 0 делится на любое целое b ≠ 0 . Если a ≠ 0 , то очевидно, что множество всех делителей числа a конечно.

Утверждение о том, что b делит a обозначают символом b / a . Если b не

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI

делит a, то этот факт обозначают b $\not\vdots$ a.

Отношение делимости

Понятие делимости является одним из фундаментальных понятий теории чисел.

Для данных целых чисел a и b говорят, что a делит b(или иными словами, что b делится на a), и обозначают a|b если существует такое целое число d, что b=ad:

$$\forall a, b \in Z: a|b \Leftrightarrow \exists d \in Z: b = ad$$

В этом случае число аназывают делителем числа b. Очевидно, что любое целое число 1>b имеет, по крайней мере, два положительных делителя: 1 и b.

Собственным делителем числа b называют любой положительный делитель, не равный b.

Нетривиальным делителем числа b называют любой положительный делитель, не равный 1 или b.

Простым(prime) называют целое число, большее 1, не имеющее нетривиальных делителей.

Составным(complex) называют число, имеющее, по крайней мере, один нетривиальный делитель.

Непосредственно из определения делимости следуют свойства данного отношения.

1. Если a делит b, и c – любое целое число, то a делит произведение bc:

$$\forall a, b \in Z: a|b \,\&\, c \in Z \Rightarrow a|bc$$

2. Если a делит b, a b, в свою очередь, делит c, то a делит c

$$\forall a, b, c \in Z: a|b \,\&\, b|c \Rightarrow a|bc$$

3. Если a делит b, и a делит c, то a делит b$\pm$c

$$\forall a, b, c \in Z: a|b \,\&\, a|c \Rightarrow a|(b \pm c)$$

4. Простое число p делит произведение целых чисел ab, в том и только в том случае, когда p делит a, либо p делит b:

$$\forall x, p: (x = 1 \vee x = p) \forall: a|b \in Z: p|ab \Rightarrow p|a \vee p|b$$

5. Если m делит a и, n делит a, и если m и n не имеют общих делителей больших 1, то произведение m n делит a:

$$\forall m,n \in Z: m|a \& n|a \& \forall d: d|m \& d|n: d = 1 \Rightarrow (m \cdot n)|a$$

Наибольшим общим делителем двух данных чисел a и b, не равных одновременно нулю, называется наибольшее целое число d, делящее одновременно a и b.

a b – есть единственное целое положительное число, которое делит a и b, и делится при этом на любой их общий делитель. Хорошо известен способ нахождения наибольшего общего делителя двух положительных чисел по известному разложению на множители. Для отыскания a,b необходимо взять все простые числа, входящие в оба разложения, и каждое возвести в 35 степень, равную минимуму из двух соответствующих показателей.

Наименьшим общим кратным двух данных чисел a и b, не равных одновременно нулю, называется наименьшее целое положительное число d, делящееся одновременно на a и b.

Два целых числа a и b называются ***взаимно простыми***, если они не имеют общих делителей больших 0.

В арифметике целых чисел, если мы a делим на n, мы можем получить q и r. Отношения между этими четырьмя целыми числами можно показать как

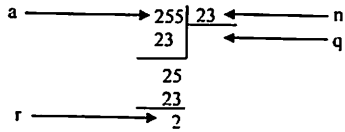
$$a = q \times n + r$$

В этом равенстве a называется делимое ; q — частное ; n — делитель и r — остаток. Обратите внимание, что это — не операция, поскольку результат деления a на n — это два целых числа, q и r. Мы будем называть это уравнением деления.

Пример

Предположим, что a = 255, a n = 23. Мы можем найти q = 11 и r = 2, используя алгоритм деления, мы знаем из элементарной арифметики — оно определяется, как показано снизу.

Нахождение частного и остатка



Большинство компьютерных языков может найти частное и остаток, используя заданные языком операторы. Например, на языке C оператор "/" может найти частное, а оператор "%" — остаток.

Теория делимости

Теперь кратко обсудим теорию делимости — тема, с которой мы часто сталкиваемся в криптографии. Если a не равно нулю, а r = 0, в равенстве деления мы имеем

$$a = q \times n$$

Мы тогда говорим, что a делится на n (или n — делитель a). Мы можем также сказать, что a делится без остатка на n. Когда мы не интересуемся значением q, мы можем записать вышеупомянутые отношения как n|a. Если остаток не является нулевым, то n не делит, и мы можем записать отношения как n†a.

Пример

Целое число 4 делит целое число 32, потому что 32=8×4. Это можно отобразить как 4|32. Число 8 не делит число 42, потому что 42 = 5 ×8 + 2. В этом уравнении число 2 — остаток. Это можно отобразить как 8†42.

Теорема о делении с остатком. Если a и b целые числа, то b>0, то существуют единственные целые числа q и r такие, что a=b*q+r, $0 \le r < b$.

Число q называют неполным частным при делении a на b, число r называют остатком от деления a на b.

Очевидно, что если r=0, то b/a.

Пример. Пусть b=12. Тогда для чисел a= 110, a= -53, a =156 имеем a=b q+r.

110=12 9+2 0<r=2<b=12

-53=12+(-5)+7 0<r=7<b=12

156=12 13 +0 r=0

Любое натуральное составное число n представляется в виде n=ab, 1<a<n, 1<b<n.

Наименьший отличный от единицы делитель натурального числа n>1 есть простое число.

Алгоритм Евклида нахождения наибольшего общего делителя. Выполним следующие деления с остатком:

$$a = bq_1 + r_1, 0 \le r_1 \le b,$$

$$b = r_1 q_1 + r_2, 0 \le r_2 \le r_1,$$

$$r_1 = r_2 q_3 + r_3, 0 \le r_3 \le r_2,$$

...

$$r_{n-2} = r_{n-1} q_n + r_n, 0 \le r_n \le r_{n-1},$$

$$r_{n-1} = r_n q_{n+1}$$

Применим алгоритм Евклида к отысканию (175, 77).

$175 = 77 \cdot 2 + 21,$

$77 = 21 \cdot 3 + 14,$

$21 = 14 \cdot 1 + 7,$

$14 = 7 \cdot 2.$

Последний положительный остаток есть $r_3 = 7$. Значит, (175, 77)=7.

Расширенный алгоритм Евклида

Криптосистема RSA встречает уравнение d*e(mod (n))=1 при нахождении открытого ключа «e», и его решение напрямую связано с проблемой поиска всех решений уравнения ax+by=d, d=НОД(a,b) эквивалентно и позволяет найти обратный элемент mod n для заданного числа a в соответствии с этим алгоритмом.

Теорема: a и b натуральные числа, d=НОД(a,b), тогда находиться такие целые числа α и β.

$$\alpha * a + \beta * b = d$$

Этот алгоритм дает возможность найти не толька НОД двух натуральных чисел, но коэфиценты и α и β.

Приминеятся для нахождения открытых и закрытых ключей в алгоритме RSA.

(e*d) mod n = 1

n=17 e=2 нужна найти число d=?

a=-63; n=6; a mod n = -63 mod 6 = -63 +11*6 = -63+66=3

-256 mod 75 = -256+4*75=44

x * 17 mod 67 = 1

67 = 17*3 + 16

16=67-17*3

17 = 16 +1

1=17-16 = 17 – (67-17*3)=17 – 67 +17*3 = 4*17-67 = 68 – 67

x=4

U={a, 1, 0}, V={b, 0, 1}, T={U[1]modV[1], U[2]-[U[1]/V[1]*V[2], U[3]-[U[1]/V[1]*V[3]}.

Здесь из исходных значений формируются множества U и V, а на их основе вычисляется множество T. Вычисления прекращаются, когда первый элемент множества T равен T[1]=1, и становится равным d=T[3]. В противном случае значения множества V присваиваются множеству U, а значения множества T присваиваются множеству V (U=V, V=T), и на их основе снова вычисляется множество T и снова проверяется равенство T[1]=1. Эта последовательность выполняется до тех пор, пока не будет выполнено равенство T[1]=1, а при равенстве d=T[3], и вычисления прекращаются.

Например,

(d*8)mod23=1; a=23, b=8;

тогда множества: U={23, 1, 0}, V={8, 0, 1} и T={23mod8, 1-[23/8]*0, 0-[23/8]*1}={7, 1, -2} следовательно, условие T[1]=1 не выполнено.

U=V = {8, 0, 1}, V=T={7, 1, -2}, T={8mod7, 0-[8/7]*1, 1-[8/7]*(-2)}={1, -1, 3}. Следовательно, T[1]=1, и d=T[3]=3. Результат:

(3*8) можно доказать равенством mod23=1.

Таблица 1.1 таблица

Системы счисления, используемые в вычислительной технике

Десятичная	Двоичная	Восьмеричная	Шестнадцатеричная
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Дробные числа из десятичного в двоичную

8 5 0 -1 -3 -7

$289{,}64_{10} \rightarrow x_2 = 10010001{,}1010001$

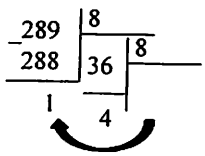
$10010001 = 2^8+2^5+2^0 = 256+32+1 = 289$

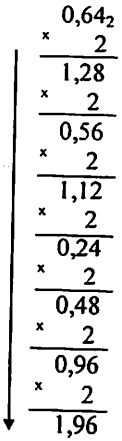
$0{,}1010001 = 2^{-1}+2^{-3}+2^{-8} = 0{,}5+0{,}125+0{,}007 \approx 0{,}64$

256	128	64	32	16	8	4	2	1	
289	33	33	33	1	1	1	1	1	0
1	0	0	1	0	0	0	0	1	

$0{,}64_2$
$\times$ 8
5,12
$\times$ 8
0,96
$\times$ 2
7,69

$289{,}64_{10} \rightarrow x_8 = 441{,}507$





$441 = 4*8^2 + 4*8^1 + 1*8^0$

$0{,}501 = 5*8^{-1} + 1*8^{-3} = 0{,}625 + 0{,}002 \approx 0{,}64$

Умножение чисел

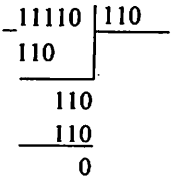
$115*51_{(10)} = 5865$

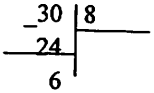
1110011*110011

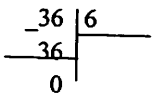
```
      1110011
       110011
      1110011
     1110011
    1110011
   1110011
1011011101001
```

1011011101001 = $2^{12}*1 + 2^{11}*0 + 2^{10}*1 + 2^9*1 + 2^8*0 + 2^7*1 + 2^6*1 + 2^5*1 + 2^4*0 + 2^3*1 + 2^2*0 + 2^1*0 + 2^0*1 = 4096 + 0 + 1024 + 512 + 0 + 128 + 64 + 32 + 0 + 8 + 0 + 0 + 1 = 5865$

Деление чисел







$30:6_{(10)} = 5$

$36_{(8)}:6_{(8)} = 5_{(8)}$

$11110:110_{(2)}$

Таблица 1.2

Действия в двоичной системе счисления

Сложение	Вычитание	Умножение
0+0=0 0+1=1	0-0=0 1-0=1	0*0=0 0*1=0
1+0=1 1+1=10	10-1=1	1*0=0 1*1=1

Таблица 1.3

Восьмеричное сложение.

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

Таблица 1.4

Восьмеричное умножение.

X	0	1	2	3	4	5	6	7
0	0	0	0	3	4	5	6	7
1	0	1	2	4	5	6	7	10
2	0	2	4	5	6	7	10	11
3	0	3	6	6	7	10	11	12
4	0	4	10	7	10	11	12	13
5	0	5	12	10	11	12	13	14
6	0	6	14	11	12	13	14	15
7	0	7	16	12	13	14	15	16

Таблица 1.5

шестнадцатеричное сложение.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15

7	7	8	9	A	B	C	D	E	F	1	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	1	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	1	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	1	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	1	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	1	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	1	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	1	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	1	19	1A	1B	1C	1D	1E

Таблица 1.6

шестнадцатеричное умножение.

x	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	7	E	15	1C	23	2A	31	38	3F	48	4D	54	5B	62	69
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	B	16	21	2C	37	42	4D	58	63	6E	78	84	8F	9A	A5
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	D	1A	27	34	41	4E	5B	88	75	82	8F	90	A9	B6	C3
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	F	1E	2D	2C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

1) Логическое умножение или конъюнкция:

Конъюнкция - это сложное логическое выражение, которое считается истинным в том и только том случае, когда оба простых выражения являются истинными, во всех остальных случаях данное сложеное выражение ложно. Обозначение: F = A & B.

Таблица истинности для конъюнкции

A	B	F
1	1	1
1	0	0
0	1	0

0	0	0

2) Логическое сложение или дизъюнкция:

Дизъюнкция - это сложное логическое выражение, которое истинно, если хотя бы одно из простых логических выражений истинно и ложно тогда и только тогда, когда оба простых логических выраженныя ложны. Обозначение: F = A v B..

Таблица истинности для дизъюнкции

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

3) Логическое отрицание или инверсия:

Инверсия - это сложное логическое выражение, если исходное логическое выражение истинно, то результат отрицания будет ложным, и наоборот, если исходное логическое выражение ложно, то результат отрицания будет истинным. Другими простыми слова, данная операция означает, что к исходному логическому выражению добавляется частица НЕ или слова НЕВЕРНО, ЧТО.

Обозначение: F =¬A.

Таблица истинности для инверсии

A	¬A
1	0
0	1

4) Логическое следование или импликация:

Импликация - это сложное логическое выражение, которое истинно во всех случаях, кроме как из истины следует ложь. То есть данная логическая операция связывает два простых логических выражения, из которых первое является условием (A), а второе (B) является следствием.

«A → B» истинно, если из A может следовать B.

Обозначение: $F = A \rightarrow B$.

Таблица истинности для импликации

A	B	F
1	1	1
1	0	0
0	1	1
0	0	1

5) Логическая равнозначность или эквивалентность:

Эквивалентность - это сложное логическое выражение, которое является истинным тогда и только тогда, когда оба простых логических выражения имеют одинаковую истинность.

«$A \leftrightarrow B$» истинно тогда и только тогда, когда A и B равны.

Обозначение: $F = A \leftrightarrow B$.

Таблица истинности для эквивалентности

A	B	F
1	1	1
1	0	0
0	1	0
0	0	1

6) Операция XOR (исключающие или)

«$A \oplus B$» истинно тогда, когда истинно A или B, но не оба одновременно.

Эту операцию также называют "сложение по модулю два".

Обозначение: $F = A \oplus B$.

A	B	F
1	1	0
1	0	1
0	1	1
0	0	0

Порядок выполнения логических операций в сложном логическом выражении

1. Инверсия;
2. Конъюнкция;
3. Дизъюнкция;
4. Импликация;
5. Эквивалентность.

№	Для ИЛИ, ∨	Для И, &	Примечание
1	$A \vee 0 = A$	$A \& 1 = A$	Ничего не меняется при действии, константы удаляются
2	$A \vee 1 = 1$	$A \& 0 = 0$	Удаляются переменные, так как их оценивание не имеет смысла
3	$A \vee B = B \vee A$	$AB = BA$	Переместительный (коммутативности)
4	$A \vee \neg A = 1$		Один из операторов всегда 1 (закон исключения третьего)
5		$A \& \neg A = 0$	Один из операторов всегда 0 (закон

			не противоречия)
6	$A \vee A = A$	$A \& A = A$	Идемпотентности (NB! В место A можно подставить составное выражение!)
7	$\neg\neg A = A$		Двойное отрицание
8	$(A \vee B) \vee C = A \vee (B \vee C)$	$(A \wedge B) \wedge C = A \wedge (B \wedge C)$	Ассоциативный
9	$(A \vee B)\&C=(A\&C)\vee(B\&C)$	$(A\&B) \vee C = (A \vee C)\&(B \vee C)$	Дистрибутивный
10	$(A \vee B)\&(\neg A \vee B) = B$	$(A\&B) \vee (\neg A\&B) = B$	Склеивания
11	$\neg(A \vee B) = \neg A \ \&\neg B$	$\neg(A\&B) = \neg A \vee \neg B$	Правило де Моргана
12	$A \vee (A\&C) = A$	$A\&(A \vee C) = A$	Поглощение
13	$A \to B = \neg A \vee B$ и $A \to B = \neg B \to \neg A$		Снятие (замена) импликации
14	1) $A \leftrightarrow B = (A\&B) \vee (\neg A\&\neg B)$ 2) $A \leftrightarrow B = (A \vee \neg B)\&(\neg A \vee B)$		Снятие (замена) эквивалентности

Задания к практической работе.

№	Свойства модуля	n>0 va a<0 найти b=a mod n	(e*d)modn=1 задана e и n найдите d?	$X_{2^-} > Y_{10}$	$X_{8^-} > Y_{10}$	$X_{16^-} > Y_{10}$

		n				
1.	a=3; b=-4; c=5; n=8;	a=-45; n=7;	n=17; e=2;	11010110	3261	4BA
2.	a=6; b=-7; c=8; n=8;	a=-54; n=8;	n=19; e=4;	10101011	2512	1AF
3.	a=12; b=-21; c=13; n=8;	a=-5; n=60;	n=17; e=3;	10110110	2674	AA4
4.	a=14; b=-12; c=13; n=8;	a=-55; n=21;	n=23; e=4;	10001010	2713	B1A
5.	a=15; b=-16; c=17; n=8;	a=-56; n=50;	n=23; e=6;	11011011	3054	9A4
6.	a=51; b=-21; c=12; n=8;	a=-52; n=105;	n=23; e=12;	10011101	2751	AA8
7.	a=51; b=-10; c=53; n=8;	a=-55; n=4;	n=23; e=2;	11010111	2517	2BA
8.	a=89; b=-10; c=56; n=8;	a=-63; n=6;	n=29; e=6;	11100110	3012	2DA
9.	a=11; b=-12; c=13; n=8;	a=-36; n=89;	n=29; e=5;	11100111	3232	1AC
10.	a=45; b=-54; c=5; n=8;	a=-65; n=56;	n=31; e=4;	10101010	2677	8BA
11.	a=5; b=-96; c=69; n=8;	a=-89; n=98;	n=31; e=8;	11111111	3134	EAC
12.	a=52; b=-5; c=5; n=8;	a=-89; n=21;	n=17; e=9;	11111110	2513	CA1
13.	a=9; b=-8; c=5; n=8;	a=-100; n=33;	n=50; e=17;	10110110	3171	B2A
14.	a=12; b=-15; c=18; n=8;	a=-102; n=25;	n=37; e=19;	11110000	2739	1BA

MUHAMMAD AL-XORAZMIY NOMIDAGI
TOSHKENT AXBOROT
TEXNOLOGIYALARI UNIVERSITETI

15.	a=40; b=-42; c=30; n=8;	a=-104; n=23;	n=41; e=7;	10111111	2615	AA9
16.	a=50; b=-9; c=61; n=8;	a=-67; n=23;	n=41; e=6;	11111100	3146	31D
17.	a=8; b=-9; c=15; n=8;	a=-47; n=3;	n=43; e=4;	10111011	2730	FA1
18.	a=56; b=-65; c=20; n=8;	a=-89; n=12;	n=43; e=11;	11110011	3025	E2A
19.	a=40;b=-50; c=10; n=8;	a=-45; n=24;	n=47; e=8;	11011111	2519	1AE
20.	a=2; b=-3; c=5; n=8;	a=-74; n=69;	n=47; e=6;	11111010	3112	B1D
21.	a=52;b=-63; c=41; n=8;	a=-105; n=501;	n=61; e=31;	10111000	2740	2AD
22.	a=78; b=-8; c=5; n=8;	a=-91; n=19;	n=67; e=17;	11110101	2614	7BA
23.	a=7; b=-8; c=41; n=8;	a=-58; n=85;	n=67; e=4;	11011000	3024	E2D

Контрольные вопросы

1) Опишите, простые числа и применение в криптографии.
2) Объясните на примере свойства модулярной арифметики.
3) Целые числа и их использование в криптографии.

2-практическая работа

Тема: Анализ шифров основанных на одинарной перестановке

Цель работы: формирование теоретических и практических знаний умений и навыков одно и многозначности, подстановка, перестановка.

Теоретическая часть

Алфавит - законченное множество используемых для кодирования информации символов.

Текст (сообщение, открытое информация) - упорядоченный последовательность из символов алфавита.

Шифрование - процесс преобразования исходного текста (который носит также название открытого текста) в зашифрованный.

Дешифрование - обратный шифрованию процесс. На основе ключа зашифрованный текст преобразуется в исходный.

Кодирование – это перевод информации с одного языка на другой (запись в другой системе символов, в другом алфавите). При этом обычно кодированием называют перевод информации с «человеческого» языка на формальный, например, в двоичный код, а декодированием – обратный переход.

Ключ - это секретная информация, которая используется криптографическим алгоритмом при шифрование и дешифрование сообщений.

Имеются следующие "классические" алгоритмы шифрования:

- **подстановка** (простая – одно алфавитная, многоалфавитная однопетлевая, многоалфавитная многопетлевая);
- **перестановка** (простая, усложненная);
- **гаммирование** (смешивание с короткой, длинной или неограниченной маской).

Устойчивость каждого из перечисленных методов к дешифрованию без знания ключа характеризуется количественно с помощью показателя Sк, представляющего собой минимальный объем зашифрованного текста,

который может быть дешифрован посредством статистического анализа. Классические алгоритмы шифрования данных.

Шифр Цезаря

Также известный, как шифр сдвига, код Цезаря или сдвиг Цезаря — один из самых простых и наиболее широко известных методов шифрования.

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом находящимся на некотором постоянном числе позиций левее или правее него в алфавите. Например, в шифре со сдвигом 3 А была бы заменена на Г, Б станет Д, и так далее.

Математическая модель

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

$$y=(x+k) \bmod n$$

$$x=(y-k+n) \bmod n,$$

где x - символ открытого текста, y - символ шифрованного текста, n - мощность алфавита (число символов), k — ключ.

С точки зрения математики шифр Цезаря является частным случаем аффинного шифра.

Шифр назван в честь римского императора Гая Юлия Цезаря, использовавшего его для секретной переписки со своими генералами.

Шифрование с использованием ключа k = 3. Буква «Е» «сдвигается» на три буквы вперёд и становится буквой «З». Твёрдый знак, перемещённый на три буквы вперёд, становится буквой «Э», и так далее:

Исходный алфавит: АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ

Шифрованный: ГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯАБВ

Исходный текст:

Съешь же ещё этих мягких французских булок, да выпей чаю.

Шифрованный текст получается путём замены каждой буквы исходного текста соответствующей буквой шифрованного алфавита:

Фэзыя йз зьи ахлш пвёнлш чугрщцкфнлш дцосн, жг еютзм ъгб.

Аффинная криптосистема.

Обобщением системы Цезаря является аффинная криптосистема. Она определяется двумя числами a и b, где $0<=a, b<=n-1$, n - как и раньше, является мощностью алфавита. *Числа a и n должны быть взаимно просты.*

Соответствующими заменами являются:

$$E=A_{a,b}(j)=(a\,j+b)(\mathrm{mod}\ n)$$

$$D=A^{-1}{}_{a,b}(\mathrm{j})=(j-b+n)*a^{-1}(\mathrm{mod}\ n)$$

Поиск мультипликативно-обратного (обозначенного как a^{-1}) осуществлять по алгоритму Евклида. Очевидно, что при a=1 аффинная криптосистема вырождается в шифр Цезаря. Взаимная простота a и n необходима для биективности отображения, в противном случае возможны отображения различных символов в один и неоднозначность дешифрирования. Количество ключей аффинной криптосистемы зависит от мощности используемого алфавита. Для алфавита русского языка коэффициент b может принимать 33 значения, коэффициент a – только значения, взаимно простые с числом 33: 1, 2, 4, 5, 7, 8, 10, 13, 14, 16, 17, 19, 20, 23, 25, 26, 28, 29, 31, 32. Итого 20 значений. Все возможные сочетания допустимых b и a дают ключевое пространство 33*20=660-1=659 ключей (1 ключ вычитается, поскольку сочетание a=1 и b=33 преобразует алфавит в самого себя).

Шифр Цезаря и аффинная криптосистема относятся к классу одноалфавитных криптосистем, то есть для выбранного ключа некоторая буква исходного открытого текста всегда будет заменяться одной и той же буквой в шифротексте. Поэтому данные шифры могут быть вскрыты методом *частотного криптоанализа.* Частотный анализ использует то свойство зашифрованного текста, что частота встречаемости символов в нем совпадает с частотой встречаемости соответствующих символов в открытом тексте. Если же учесть, что частоты встречаемости различных символов в текстах соответствующего языка распределены неравномерно (так,

например, относительная частота встречаемости буквы «А» в текстах на русском языке составляет 0.069, а буквы «Ф» 0.003), то, подсчитав относительную частоту встречаемости букв в шифротексте, можно предположить, что символ, наиболее часто встречающийся в шифротексте, соответствует символу, наиболее часто встречающемуся в текстах на соответствующем языке, и найти таким образом ключ k для шифра Цезаря. Для вскрытия параметров a и b аффинной криптосистемы потребуется найти соответствие двух букв – наиболее часто встречающейся в шифротексте и второй по частоте. Эффективность частотного анализа шифра Цезаря с ключевым словом во многом зависит от длины используемой ключевой фразы, общее же количество допустимых отражений алфавита равно, как уже упоминалось, $n!$.

Попробуем зашифровать и расшифровать слово «master», используя при этом Аффинный шифр. Для примера будем использовать следующие ключи: a=5, b=7.

Буква m имеет номер 12, тогда зашифрованный номер будет равен (5*12+7) mod 26 = 15, что соответствует букве p

Буква a имеет номер 0, тогда зашифрованный номер будет равен (5*0+7) mod 26 = 7, что соответствует букве h

Буква s имеет номер 18, тогда зашифрованный номер будет равен (5*18+7) mod 26 = 19, что соответствует букве t

Буква t имеет номер 19, тогда зашифрованный номер будет равен (5*19+7) mod 26 = 24, что соответствует букве y

Буква e имеет номер 4, тогда зашифрованный номер будет равен (5*4+7) mod 26 = 1, что соответствует букве b

Буква r имеет номер 17, тогда зашифрованный номер будет равен (5*17+7) mod 26 = 14, что соответствует букве o

В результате шифрования получилась строка phtybo

Расшифровывать будет строке, полученную в примере (phtybo)

$a*a^{-1} \bmod n = 1$

Для начала, нужно найти . a* по модулю 26 должно давать единицу. Значит, нам подходят результаты 27, 53, 79, 105 и т.д. Т.к. a=5,нам нужно число, заканчивающееся на 5. 105 подходит. 105/5=21, отсюда следует, что =21

Таким образом:

Буква p имеет номер 15, тогда расшифрованный номер будет равен 21*(15-7) mod 26 = 12, что соответствует букве m

Буква h имеет номер 7, тогда расшифрованный номер будет равен 21*(7-7) mod 26 = 0, что соответствует букве a

Буква t имеет номер 19, тогда расшифрованный номер будет равен 21*(19-7) mod 26 = 18, что соответствует букве s

Буква y имеет номер 24, тогда расшифрованный номер будет равен 21*(24-7) mod 26 = 19, что соответствует букве t

Буква b имеет номер 1, тогда расшифрованный номер будет равен 21*(1-7) mod 26 = 4, что соответствует букве e

Буква o имеет номер 14, тогда расшифрованный номер будет равен 21*(14-7) mod 26 = 17, что соответствует букве r

Получилось «master» - Наша начальная строка.

Преобразования биграмм

Элементами открытого и шифрованного текстов являются двухбуквенные блоки, называемые биграммами. Это значит, что открытый текст разбивается на двухбуквенные сегменты. Если открытый текст состоит из нечётного числа букв, то, чтобы получить целое число биграмм, добавим к концу текста ещё одну букву, выбрав её так, чтобы не исказить смысл, например, добавим пробел, если он содержится в нашем алфавите.

Каждой биграмме приписывается далее ее числовой эквивалент. Простейший способ - взять его в виде P = xN + y где x - числовой эквивалент первой буквы биграммы, y - числовой эквивалент второй буквы биграммы, а N - число букв в алфавите, т.е. рассматривать биграмму как запись двузначного целого числа в системе счисления с основанием N. Это дает

взаимно однозначное соответствие между множеством всех биграмм в N-буквенном алфавите и множеством всех неотрицательных целых, меньших N. Выше мы рассмотрели частный случай этой процедуры при N^2.

Следующий шаг - выбор шифрующего преобразования, т.е. перестановки целых чисел $\{0,1,2,..., N^2\text{-}1\}$. Примером простейших шифрующих преобразований служат аффинные преобразования: при шифровании P переходит в неотрицательное целое число, меньшее N^2 и удовлетворяющее сравнению $C \equiv aP + b\ (mod\ N^2)$. Здесь, как и раньше, число a должно не иметь общих множителей с N (что означает отсутствие общих множителей и с N^2) для того, чтобы существовало обратное преобразование, указывающее способ дешифрования.

$P = a^{-1}(C - b) mod\ N^2$.

Открытый текст «crypto» { 2 17 24 15 19 14 } “cr { 2 17 }”

P=am+b, m=26 Здесь a=2, b=17. P=2*26+17=69.

В этом случае a=157, b=580, $N^2 = 676$

$C \equiv aP + b\ (mod\ N^2)$ = 157*69+580 mod 676 = 11413mod676=597

597=22*26+25 22=w 25=z

M=crypto { 2 17 24 15 19 14 }

C { 22 25 6 23 21 22 } = { w z g x v w }

Дешифрования

$P = a^{-1}(C - b) mod\ N^2$

a=157, b=580, N^2=26*26=676

$P = 157^{-1}(597 - 580)$ mod 676 = 521(597-580) mod 676 = 8857 mod 676 = 69

69=2*26+17 2=c 17=r

Шифр перестановки - сущность методов замены (подстановки) - в замене символов исходной информации, записанных в одном алфавите символами из другого алфавита по определенному правилу. Это метод симметричного шифрования, в котором элементы исходного открытого текста меняют местами. Элементами текста могут быть отдельные символы

(самый распространённый случай), пары букв, тройки букв, комбинирование этих случаев и так далее. Типичными примерами перестановки являются анаграммы. В классической криптографии шифры перестановки можно разделить на два класса:

Шифры одинарной (простой) перестановки — при шифровании символы открытого текста перемещаются с исходных позиций в новые один раз.

Шифры множественной (сложной) перестановки — при шифровании символы открытого текста перемещаются с исходных позиций в новые несколько раз.

Шифр табличной маршрутной перестановки

Наибольшее распространение получили маршрутные шифры перестановки, основанные на прямоугольниках (таблицах). Например, можно записать сообщение в прямоугольную таблицу по маршруту: по вертикалям начиная с верхнего левого угла, поочередно сверху вниз. Сообщение будем списывать по маршруту: по горизонтали, начиная с верхнего левого угла, поочередно слева направо.

Шифрование ПРОСТОЙ ПЕРЕСТАНОВКИ

М= ШИФР_ПРОСТОЙ_ПЕРЕСТАНОВКИ

Ш	И	Ф	Р	_
П	Р	О	С	Т
О	Й	_	П	Е
Р	Е	С	Т	А
Н	О	В	К	И

С= ШПОРНИРЙЕОФО_СВРСПТК_ТЕАИ

Шифрование ПЕРЕСТАНОВКИ С ПОМОЩЬЮ КЛЮЧА

М= ШИФР_ПРОСТОЙ_ПЕРЕСТАНОВКИ

К=КЛЮЧИ

К	Л	Ю	Ч	И

И	К	Л	Ч	Ю

Ш	И	Ф	Р	_
П	Р	О	С	Т
О	Й	_	П	Е
Р	Е	С	Т	А
Н	О	В	К	И

_	Ш	И	Р	Ф
Т	П	Р	С	О
Е	О	Й	П	_
А	Р	Е	Т	С
И	Н	О	К	В

С= _ТЕАИ ШПОРНИРЙЕОРСПТКФО_СВ

Шифрование ДВОЙНОЙ ПЕРЕСТАНОВКИ

М= ШИФР_ПРОСТОЙ_ПЕРЕСТАНОВКИ

$К_1$= 8, 5, 4, 6, 2 $К_2$=3, 2, 4, 6, 1

К	**3**	**2**	**4**	**6**	**1**
8	Ш	И	Ф	Р	_
5	П	Р	О	С	Т
4	О	Й	_	П	Е
6	Р	Е	С	Т	А
2	Н	О	В	К	И

К	**3**	**2**	**4**	**6**	**1**
2	Н	О	В	К	И
4	О	Й	_	П	Е
5	П	Р	О	С	Т
6	Р	Е	С	Т	А
8	Ш	И	Ф	Р	_

К	**1**	**2**	**3**	**4**	**6**
2	И	О	Н	В	К
4	Е	Й	О	_	П
5	Т	Р	П	О	С
6	А	Е	Р	С	Т
8	_	И	Ш	Ф	Р

С= ИЕТА_ОЙРЕИНОПРШВ_ОСФКПСТР

Шифр табличной маршрутной перестановки

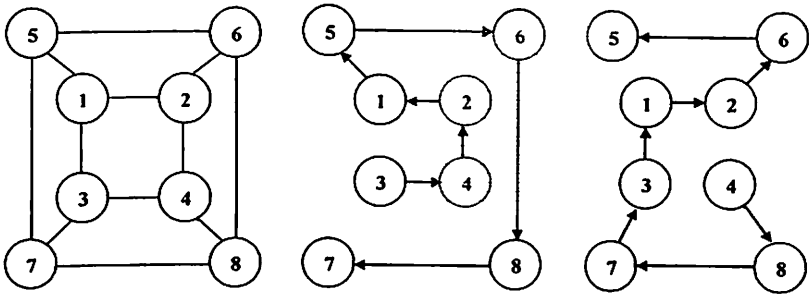
Записать сообщение в прямоугольную таблицу по маршруту: по горизонтали, начиная с верхнего левого угла, поочередно слева направо. Сообщение будем списывать по маршруту: по вертикалям, начиная с верхнего правого угла, поочередно сверху вниз.

М=ПРИМЕР МАРШРУТНОЙ ПЕРЕСТАНОВКИ

П	Р	И	М	Е
Р	М	А	Р	Ш
Р	У	Т	Н	О
Й	П	Е	Р	Е
С	Т	А	Н	О
В	К	И	*	*

С= ЕШОЕОМРНРНИАТЕАИРМУПТКПРРЙСВ

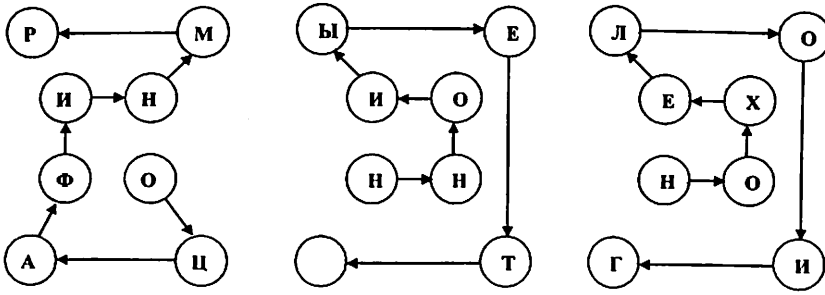
Другим методом перестановки является метод перестановки на основе применения маршрутов Гамильтона.



Маршрут Маршрут № 1 Маршрут № 2

рис. 2.1 - Элементной таблицы и маршрутов Гамильтона

Пусть требуется зашифровать исходный текст: Информационные технологии. В этом случае ключ будет равен: 2, 1, 1, 2. Длина зашифрованных блоков равна: 4. Для шифрования используется таблица и два маршрута. Для заданных условий маршруты с заполненными матрицами имеют следующий вид, представленный на рис. 2.2.



Маршрут № 2 Маршрут № 1 Маршрут № 1

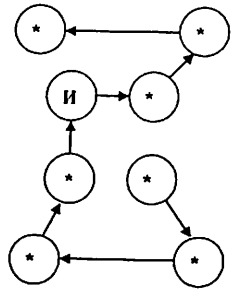


Рис. 2.2- Пример шифрования с помощью маршрутов Гамильтона

Технология метода перестановки Гамильтона такова:

Шаг 1. Исходный текст разбивается на четыре блока:

1: ИНФОРМАЦ 2: ИОННЫЕ_Т 3: ЕХНОЛОГИ 4: И*******

Шаг 2. Заполняются четыре матрицы с маршрутами 2, 1, 1, 2.

Шаг 3. Получение шифртекста путем расстановки символов в соответствии с маршрутами: ОЦАФИНМРННОИИЕТ_НОХЕЛОИГ****И***

Шаг 4. Разбиение на блоки шифртекста: ОЦАФ ИНМР ННОИ ИЕТ_ НОХЕ ЛОИГ **** И***

Буквы	Число		Буквы	Частота
E	21912		E	12.02
T	16587		T	9.10
A	14810		A	8.12
O	14003		O	7.68
I	13318		I	7.31
N	12666		N	6.95
S	11450		S	6.28
R	10977		R	6.02
H	10795		H	5.92
D	7874		D	4.32
L	7253		L	3.98
U	5246		U	2.88

C	4943		C	2.71
M	4761		M	2.61
F	4200		F	2.30
Y	3853		Y	2.11
W	3819		W	2.09
G	3693		G	2.03
P	3316		P	1.82
B	2715		B	1.49
V	2019		V	1.11
K	1257		K	0.69
X	315		X	0.17
Q	205		Q	0.11
J	188		J	0.10
Z	128		Z	0.07

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
0	1	2	3	4	5	6	7	8	9	1 0	1 1	1 2	1 3	1 4	1 5	1 6	1 7	1 8	1 9	2 0	2 1	2 2	2 3	2 4	2 5

Контрольные вопросы

1. Произвести шифрование ими, фамилии и отчества выше указанными методами шифрования.

2. Метод простой перестановки с помощью ключом?

3. Метод простой перестановки с помощью ключевым словам?

4. Аффинный шифр?

5. Биграммный шифр?

3-практическая работа

Тема: Программные реализации классических шифров

Цель работы: получение практических знаний и умений о классических шифрах.

Теоретическая часть

Шифр Цезаря (шифр простого сдвига).

Шифр Цезаря, также известный как шифр сдвига, код Цезаря или сдвиг Цезаря - один из самых простых и наиболее широко известных методов шифрования.

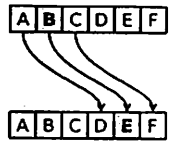


Рис. – Схема шифра Цезаря

Шифр Цезаря - это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется символом, находящимся на некотором постоянном числе позиций левее или правее него в алфавите.

Математическая модель

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

$$y = (x + k) \bmod n$$

$$x = (y - k) \bmod n$$

где x - символ открытого текста, y - символ шифрованного текста, n - мощность алфавита, а k - ключ.

С точки зрения математики шифр Цезаря является частным случаем аффинного шифра.

Алфавит:

Буква	**А**	**Б**	**В**	**Г**	**Д**	**Е**	**Ё**	**Ж**	**З**	**И**	**Й**
Номер	1	2	3	4	5	6	7	8	9	10	11
Буква	**К**	**Л**	**М**	**Н**	**О**	**П**	**Р**	**С**	**Т**	**У**	**Ф**

Номер	12	13	14	15	16	17	18	19	20	21	22
Буква	**Х**	**Ц**	**Ч**	**Ш**	**Щ**	**Ь**	**Ы**	**Ъ**	**Э**	**Ю**	**Я**
Номер	23	24	25	26	27	28	29	30	31	32	33

Необходимо зашифровать сообщение по методу Цезаря.

Исходное сообщение: «Криптография»

Ключ: 3

Решение:

Сообщение	**К**	**Р**	**И**	**П**	**Т**	**О**	**Г**	**Р**	**А**	**Ф**	**И**	**Я**
Номер l	12	18	10	17	20	16	4	18	1	22	10	33
Номер l + 3	15	21	13	20	23	19	7	21	4	25	13	3
Шифр	**Н**	**У**	**Л**	**Т**	**Х**	**С**	**Ё**	**У**	**Г**	**Ч**	**Л**	**В**

Шифр Вежинера

Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов, называемая tabula recta или квадрат (таблица) Виженера. Применительно к латинскому алфавиту таблица Виженера составляется из строк по 26 символов, причём каждая следующая строка сдвигается на несколько позиций. Таким образом, в таблице получается 26 различных шифров Цезаря. На каждом этапе шифрования используются различные алфавиты, выбираемые в зависимости от символа ключевого слова.

Человек, посылающий сообщение, записывает ключевое слово циклически до тех пор, пока его длина не будет соответствовать длине исходного текста:

Расшифровывание производится следующим образом: находим в таблице Виженера строку, соответствующую первому символу ключевого слова; в данной строке находим первый символ зашифрованного текста.

Столбец, в котором находится данный символ, соответствует первому символу исходного текста. Следующие символы зашифрованного текста расшифровываются подобным образом.

Если n - количество букв в алфавите, m_j - буквы открытого текста, k_j - буквы ключа, то шифрование Виженера можно записать следующим образом:

$$c_j=m_j+k_j \ (mod\ n)$$

И расшифровывание:

$$m_j=c_j-k_j \ (mod\ n)$$

Таблица 3.1

Шифра Вежинер

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

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

M – VEJINER

K – KEYKEYK

C – FIHSRCB

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

Алгоритмы шифрования гаммирования

В аддитивных шифрах используется сложение по модулю (mod) исходного сообщения с гаммой, представленных в числовом виде.

Гаммирование — метод симметричного шифрования, заключающийся в «наложении» последовательности, состоящей из случайных чисел, на открытый текст. Последовательность случайных чисел называется гамма-последовательностью и используется для зашифровывания и расшифровывания данных. Суммирование, обычно, выполняется в каком-либо конечном поле.

$$C_i = (P_i + K_i) \bmod N$$

$$P_i = (C_i + N - K_i) \bmod N$$

где P_i, C_i - i-ый символ открытого и шифрованного сообщения;

N - количество символов в алфавите;

K_i - i-ый символ гаммы (ключа). Если длина гаммы меньше, чем длина сообщения, то она используется повторно.

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
20	21	22	23	24	25	26	27	28	29	30	31	32

Таблица 3.2

Пример аддитивного шифрования по модулю N = 33

1	Отурытое сообщение P_i	Г	А	М	М	И	Р
		3	0	13	13	9	17
2	Гамма K_i	М	Е	Т	О	Д	Ы
		13	5	19	15	9	28
3	Шифр текст C_i	16	5	32	28	18	45
		16	5	32	28	18	12
		П	**Е**	**Я**	**Ы**	**С**	**Л**

М – ГАММИР

К – МЕТОДЫ

С – ПКЯЫСЛ

Шифр Вернама (англ. Vernam Cipher) - система симметричного шифрования, был изобретен в 1917 году Мейджором Джозефом Моборном (Major Joseph Mauborn) и Гильбертом Вернамом (Gilbert Vernam) из AT&T (American Telephone & Telegraph). В классическом понимании одноразовый блокнот является большой неповторяющейся последовательностью символов ключа, распределенных случайным образом. Первоначально это была одноразовая лента для телетайпов. Отправитель использовал каждый символ ключа для шифрования только одного символа открытого текста. Шифрование представляет собой сложение по модулю n (мощность

алфавита) символа открытого текста и символа ключа из одноразового блокнота. Каждый символ ключа используется только один раз и для единственного сообщения, иначе даже если использовать блокнот размером в несколько гигабайт, при получении криптоаналитиком нескольких текстов с перекрывающимися ключами он сможет восстановить исходный текст. Он сдвинет каждую пару шифротекстов относительно друг друга и подсчитает число совпадений в каждой позиции. если шифротексты смещены правильно, соотношение совпадений резко возрастет. С этой точки зрения криптоанализ не составит труда. Если же ключ не повторяется и случаен, то криптоаналитик, перехватывает он тексты или нет, всегда имеет одинаковые знания[2].

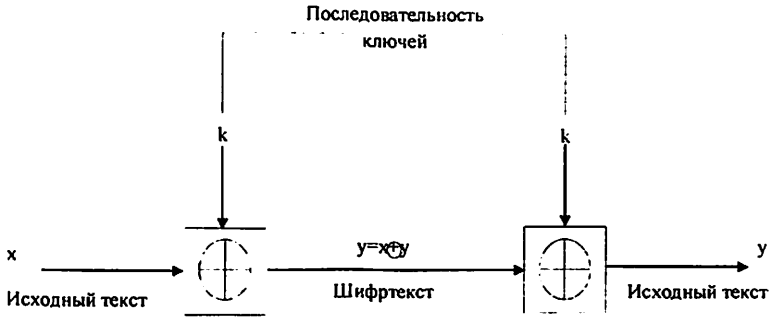


Рис. 3.1- Схема шифрования Вернама

Криптосистема была предложена для шифрования телеграфных сообщений, которые представляли собой бинарные тексты, в которых открытый текст представляется в коде Бодо (в виде пятизначных «импульсных комбинаций»). В этом коде, например, буква «А» имела вид (1 1 0 0 0). На бумажной ленте цифре «1» соответствовало отверстие, а цифре «0» - его отсутствие. Секретный ключ должен был представлять собой хаотичный набор букв того же самого алфавита[5].

$$C = M \oplus K$$

$$M = C \oplus K$$

Для получения шифротекста открытый текст объединяется операцией «исключающее ИЛИ» с секретным ключом. Так, например, при применении ключа (1 1 1 0 1) на букву «А» (1 1 0 0 0) получаем зашифрованное сообщение (0 0 1 0 1): (11000) $\oplus$ (11101) = (00101). Зная, что для принимаемого сообщения имеем ключ (1 1 1 0 1), легко получить исходное сообщение той же операцией: (00101) $\oplus$ (11101) = (11000). Для абсолютной криптографической стойкости ключ должен обладать тремя критически важными свойствами[2]:

1. Иметь случайное равномерное распределение: P_k (k) = $1/2^N$ где k - ключ, а N - количество бинарных символов в ключе;

2. Совпадать по размеру с заданным открытым текстом;

3. Применяться только один раз.

Также хорошо известен так называемый шифр Вернама по модулю m, в котором знаки открытого текста, шифрованного текста и ключа принимают значения из кольца вычетов Z_m. Шифр является обобщением оригинального шифра Вернама, где m = 2.

Без знания ключа такое сообщение не поддаётся анализу. Даже если бы можно было перепробовать все ключи, в качестве результата мы получили бы все возможные сообщения данной длины плюс колоссальное количество бессмысленных дешифровок, выглядящих как беспорядочное нагромождение букв. Но и среди осмысленных дешифровок не было бы никакой возможности выбрать искомую. Когда случайная последовательность (ключ) сочетается с неслучайной (открытым текстом), результат этого (шифротекст) оказывается совершенно случайным и, следовательно, лишённым тех статистических особенностей, которые могли бы быть использованы для анализа шифра.

В настоящее время шифрование Вернама используется достаточно редко. В большой степени из-за существенного размера ключа, длина которого должна совпадать с длиной сообщения. То есть, использование таких шифров требует огромных затрат на производство, хранение,

уничтожение ключевых материалов. Тем не менее, совершенно стойкие шифры типа Вернама всё же нашли практическое применение для защиты особо важных линий связи с относительно небольшим объёмом информации.

Недостатком использования шифра Вернама является отсутствие подтверждения подлинности и целостности сообщения. Получатель не может удостовериться в отсутствии повреждений или в подлинности отправителя.

Разработка программного модуля шифра Цезаря в среде C++ Builder 6.0

```
void __fastcall TForm2::Button3Click(TObject *Sender)
{
    AnsiString s="";
    int cod=0,key=0;
    if(Memo1 -> Lines -> Text =="")
    {
    ShowMessage(" Did not choice file for encryphtion !");
    return;
    }
    else
    {
     s = Memo1 -> Lines -> Text;
    }

     if((Edit1 ->Text != "") && StrToInt(Edit1 ->Text)>=0)
    {
     key=StrToInt(Edit1 ->Text);
     }
    else
    {
    ShowMessage("You did not enter to key number");
    return;
```

```
    }

    for( int i = 1; i <= s.Length(); i++)
   {
    cod = (int) s[i];
    if(65 <= cod && cod <= 90)
   {
      cod += key;
    if( cod > 90)
    cod = cod % 91 + 65;
   }
    if(97 <= cod && cod <= 122)
    {
       cod += key;
    if(cod > 122)
       cod = cod % 123 +97;
    }
   s[i] = (char) cod;
   }
  Memo1 -> Lines -> Text =s;
}
```

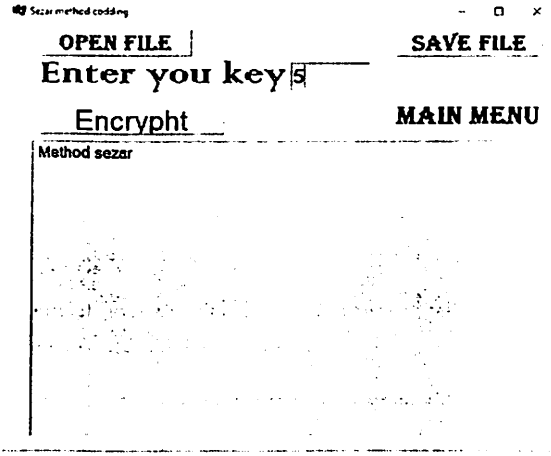


Рис. 3.2– Окно шифрования шифра Цезаря

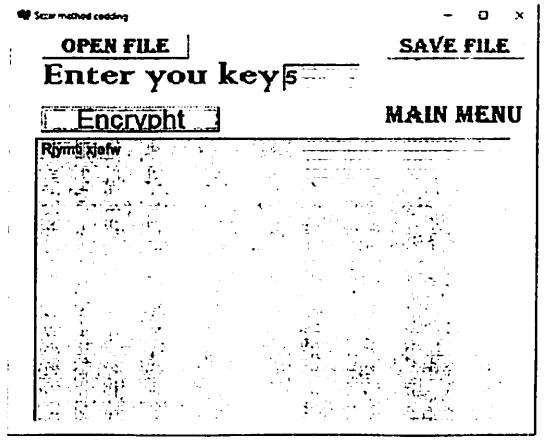


Рис.3.3 – Окно дешифрования шифра Цезаря

Разработка программного модуля шифра Вежинера в среде C++ Builder 6.0

```
void __fastcall TForm2::Button1Click(TObject *Sender)
{
 int d=1,l;
 String A="",C="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
 String s="",k="",p="";
```

```
 s=Edit1->Text;
 k=Edit2->Text;

 StringGrid1->RowCount=k.Length()+1;


 for (int i=1; i<=s.Length(); i++) s[i]=toupper(s[i]);
 for (int i=1; i<=k.Length(); i++) k[i]=toupper(k[i]);

 for (int i=1; i<=C.Length(); i++)
 {StringGrid1->Cells[i-1][0]=C[i];}


 for (int i=1; i<=k.Length(); i++)
 { for (int j=0; j<26; j++)
 {StringGrid1->Cells[j][i]=(char)(((int)k[i]+j-65)%26+65);}
 }

 for (int i=1; i<=s.Length(); i++)
 {
 if (s[i]==' ') p=p+' ';
 else
 {
 l = (int)s[i]-65+(int)k[d++]-65;
 p=p+char(l%26+65);
 if (d==k.Length()+1) d=1;
 }
 }

Edit3->Text=p;
```

}

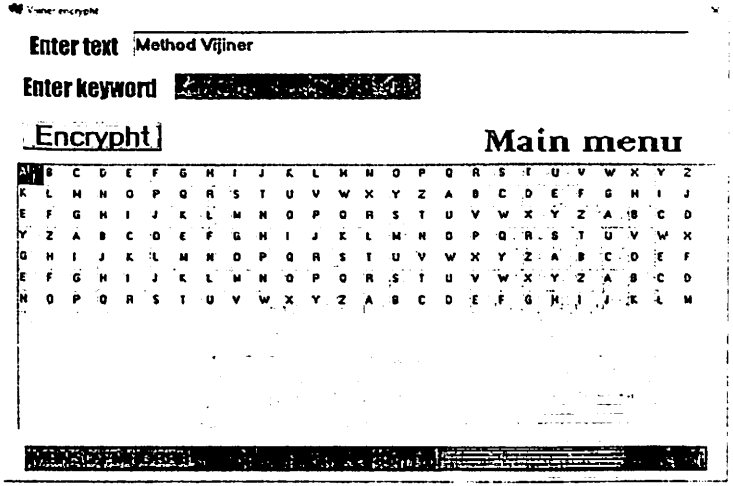


Рис.3.4 – Окно шифрования шифра Виженера

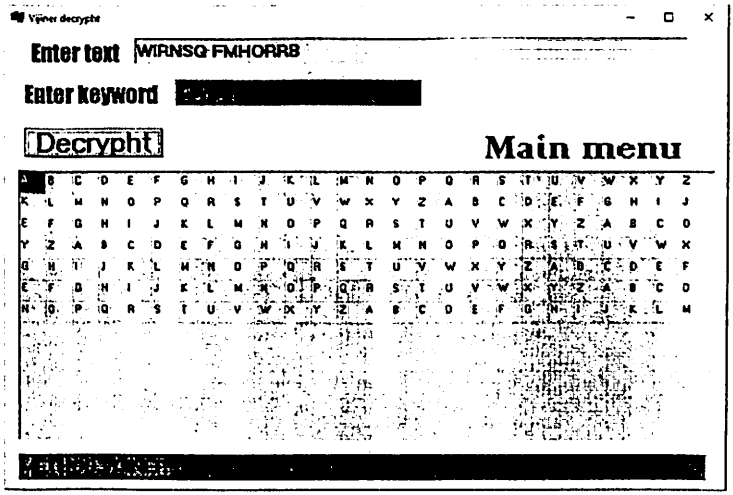


Рис.3.5 – Окно дешифрования шифра Виженера

Разработка программного модуля шифра Вернама в среде C++ Builder 6.0

```
void __fastcall TForm6::Button1Click(TObject *Sender)
{
```

```
        int vd=1,vt=1,vg=1,vl;
        AnsiString
va="ABCDEFGHIJKLMNOPQRSTUVWXYZ#!_@?*",vs="",vk="",vr="";
        char ikkilik[8];

        vs=Edit1->Text;
        vk=Edit2->Text;

        StringGrid1->RowCount=vs.Length();

        for (int i=1; i<=vs.Length(); i++) {if (vs[i]==' ') vs[i]='_';
vs[i]=toupper(vs[i]);}
        for (int i=1; i<=vk.Length(); i++) vk[i]=toupper(vk[i]);

            for (int i=1; i<=vs.Length(); i++)
            {for (int j=1; j<=va.Length(); j++)
            if (vs[i]==va[j]) {vt=j-1; break;}
            itoa(vt,ikkilik,2);
            if (vt==0) vt=1; else vt=vt+1;
            StringGrid1->Cells[0][i-1]=va[vt];
            StringGrid1->Cells[1][i-1]=ikkilik;}

            for (int i=1; i<=vs.Length(); i++)
            {for (int j=1; j<=va.Length(); j++)
             if (vk[vd]==va[j]) {vg=j-1; break;}
            itoa(vg,ikkilik,2);
            if (vg==0) vg=1; else vg=vg+1;
            StringGrid1->Cells[2][i-1]=va[vg];
            StringGrid1->Cells[3][i-1]=ikkilik;
            vd++; if (vd==vk.Length()+1) {vd=1;}
```

```
    }

    vt=1; vg=1; vd=1;

    for (int i=1; i<=vs.Length(); i++)
    {for (int j=1; j<=va.Length(); j++)
     if (vs[i]==va[j]) {vt=j-1; break;}

     for (int j=1; j<=va.Length(); j++)
     if (vk[vd]==va[j]) {vg=j-1; break;}
     vd++; if (vd==vk.Length()+1) {vd=1;}

     vl=vt^vg;
     itoa(vl,ikkilik,2);
     if (vl==0) vl=1; else vl=vl+1;
    StringGrid1->Cells[4][i-1]=va[vl];
    StringGrid1->Cells[5][i-1]=ikkilik;

     vr=vr+va[vl];
     }

     for (int i=1; i<=vr.Length(); i++)
     if (vr[i]=='_') vr[i]=' ';

     Edit3->Text=vr;

}
```

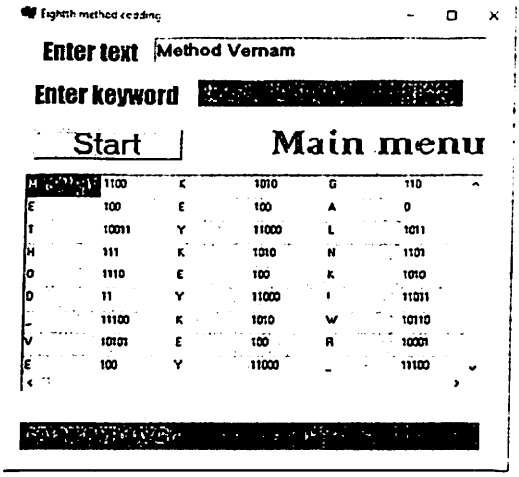


Рис. – 3.6 Окно шифрования шифра Вернама

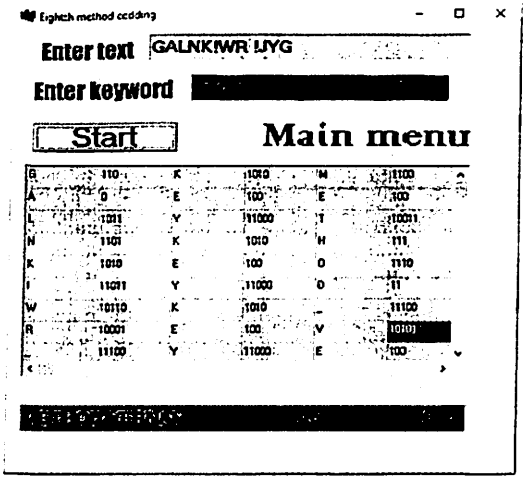


Рис. –3.7 Окно дешифрования шифра Вернама

Задание

1. Разработка программного модуля шифра Цезаря.
2. Разработка программного модуля шифра Вежинера.
3. Разработка программного модуля шифра Вернама.

4-практическая работа

Тема: Построение N битовое скремблер и расчет частоты

Цель работы: получение теоретических и практических знаний по построению скремблера и расчету периода.

Теоретическая часть

Скремблерами называются программные или аппаратные реализации алгоритма, позволяющего шифровать побитно непрерывные потоки информации. Сам скремблер представляет из себя набор бит, изменяющихся на каждом шаге по определенному алгоритму/После выполнения каждого очередного шага на его выходе появляется шифрующий бит — либо 0, либо 1, который накладывается на текущий бит информационного потока операцией XOR.

В последнее время сфера применения скремблирующих алгоритмов значительно сократилась. Это объясняется в первую очередь снижением объемов побитной последовательной передачи информации, для защиты которой были разработаны данные алгоритмы. Практически повсеместно в современных системах применяются сети с коммутацией пакетов, для поддержания конфиденциальности которой используются блочные шифры. А их криптостойкость превосходит, и порой довольно значительно криптостойкость скремблеров.

Суть скремблирования заключается в побитном изменении проходящего через систему потока данных. Практически единственной операцией, используемой в скремблерах является XOR — «побитное исключающее ИЛИ». Параллельно прохождению информационного потока в скремблере по определенному правилу генерируется поток бит — кодирующий поток. Как прямое, так и обратное шифрование осуществляется наложением по XOR кодирующей последовательности на исходную.

Генерация кодирующей последовательности бит производится циклически из небольшого начального объема информации — ключа по следующему алгоритму. Из текущего набора бит выбираются значения

определенных разрядов и складываются по XOR между собой. Все разряды сдвигаются на 1 бит, а только что полученное значение («0» или «1») помещается в освободившийся самый младший разряд. Значение, находившееся в самом старшем разряде до сдвига, добавляется в кодирующую последовательность, становясь очередным ее битом.

x_i	s_i	$y_i \equiv x_i + s_i \bmod 2$
0	0	0
0	1	1
1	0	1
1	1	0

Для построения поточных шифров очень часто используют последовательности на регистрах сдвига. Регистр сдвига с обратной связью состоит из двух частей: регистра сдвига и функции обратной связи.

Сдвиговый регистр представляет собой последовательность битов. (Количество битов определяется длиной сдвигового регистра. Если длина равна n битам, то регистр называется n-битовым сдвиговым регистром.) Всякий раз, когда нужно извлечь бит, все биты сдвигового регистра сдвигаются вправо на 1 позицию. Новый крайний левый бит является функцией всех остальных битов регистра. На выходе сдвигового регистра оказывается один, обычно младший значащий, бит. Периодом сдвигового регистра называется длина получаемой последовательности до начала ее повторения.

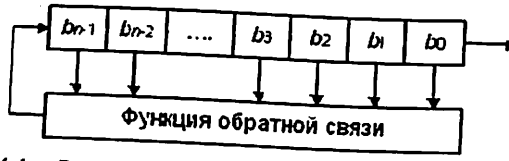


Рис.4.1 – Регистр сдвига с линейной обратной связью

Простейший тип регистров сдвига – регистр сдвига с линейной братной связью (РСЛОС или ЛРС). Обратная связь – простая операция XOR ад некоторыми битами регистра. Перечень этих битов определяется

характеристическим многочленом и называется последовательностью отводов. Иногда такую схему называют конфигурацией Фибоначчи.

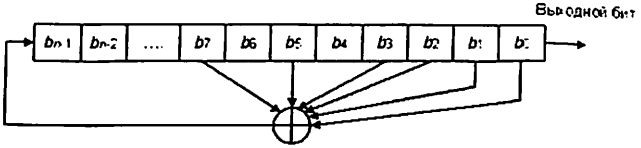


Рис.4.2 – Регистр сдвига с линейной обратной связью

При программной реализации РСЛОС пользуются модифицированной схемой: для генерации нового значащего бита вместо использования битов последовательности отводов над каждым ее битом выполняется операция XOR с выходом генератора, заменяя старый бит последовательности отводов. Такую модификацию иногда называют конфигурацией Галуа.

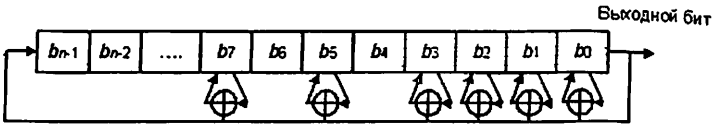


Рис.4.3 – Регистр сдвига с линейной обратной связью

n-битовый РСЛОС может находиться в одном из $2^n - 1$ внутренних состояний. Это значит, что теоретически такой регистр может генерировать псевдослучайную последовательность с периодом $2^n - 1$ битов (заполнение нулями совершенно бесполезно). Прохождение всех $2^n - 1$ внутренних состояний возможно только при определенных последовательностях отводов. Такие регистры называют РСЛОС с максимальным периодом. Для обеспечения максимального периода РСЛОС необходимо, чтобы его характеристический многочлен был примитивным по модулю 2. Степень многочлена является длиной регистра сдвига. Примитивный многочлен степени n – это такой неприводимый многочлен, который является делителем $x^{2^n}+1$, но не является делителем $x^d + 1$ для всех d, являющихся делителями $2^n - 1$. (При обсуждении многочленов термин *простое число* заменяется

термином *неприводимый многочлен*). Характеристический многочлен приведенных на рисунках РСЛОС:

$x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$

примитивен по модулю 2. Период такого регистра будет максимальным, циклически проходя все $2^{32} - 1$ значений до их повторения. Наиболее часто используются прореженные многочлены, т.е. у которых есть только некоторые коэффициенты. наиболее популярны трехчлены.

Важным параметром генератора на базе РСЛОС, является линейная сложность. Она определяется как длина *n* самого короткого РСЛОС, который может имитировать выход генератора. Линейная сложность важна, поскольку при помощи простого алгоритма Берленкемпа-Мэсси можно воссоздать такой РСЛОС, проверив всего 2*n* битов гаммы. С определением нужного РСЛОС поточный шифр фактически взламывается.

Задание

1. Используя n-разрядный скремблер написать функцию генерирующую ключ шифрования

2. Написать функцию шифрования дешифрования текста с помощью сгенерированного ключа.

№	Skrembler
1.	$x^8 + x^4 + x^3 + x^2 + 1$
2.	$x^8 + x^5 + x^3 + x^2 + 1$
3.	$x^9 + x^4 + 1$
4.	$x^9 + x^3 + 1$
5.	$x^{10} + x^3 + 1$
6.	$x^{10} + x^7 + 1$
7.	$x^5 + x^2 + 1$
8.	$x^5 + x^4 + x + 1$
9.	$x^{11} + x^2 + 1$

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

5-практическая работа

Тема: Шифрование данных с помощью блочных шифров с использованием библиотеки OpenSSL (алгоритм 3DES и его использование в библиотеке OpenSSL)

Цель работы: приобретение навыков шифрования данных на основе современных алгоритмов блочного шифрования.

Теоретическая часть

DES (Data Encryption Standart) - Симметричный алгоритм шифрования, в котором один ключ используется, как для шифрования, так и для расшифрования данных. DES разработан фирмой IBM и утвержден правительством США в 1977 году как официальный стандарт (FTPS 46-3). DES имеет блоки по 64 бит и 16 цикловую структуру сети Фейстеля, для шифрования использует ключ с длиной 56 бит. Алгоритм использует комбинацию нелинейных (S-блоки) и линейных (перестановки E, IP, IP-1) преобразований. Для DES рекомендовано несколько режимов:

- режим электронной кодовой книги (ECB — Electronic Code Book),
- режим сцепления блоков (CBC — Cipher Block Chaining),
- режим обратной связи по шифротексту (CFB — Cipher Feed Back),
- режим обратной связи по выходу (OFB — Output Feed Back).

Входными данными для блочного шифра служат:

- блок размером n бит;
- ключ размером k бит.

На выходе (после применения шифрующих преобразований) получается зашифрованный блок размером n бит, причём незначительные различия входных данных, как правило, приводят к существенному изменению результата.

Блочные шифры реализуются путём многократного применения к блокам исходного текста некоторых базовых преобразований.

Базовые преобразования:

- сложное преобразование на одной локальной части блока;
- простое преобразование между частями блока.

Так как преобразования производятся поблочно, требуется разделение исходных данных на блоки необходимого размера. При этом формат исходных данных не имеет значения (будь то текстовые документы, изображения или другие файлы). Данные должны интерпретироваться в двоичном виде (как последовательность нулей и единиц) и только после этого должны разбиваться на блоки. Все вышеперечисленное может осуществляться как программными, так и аппаратными средствами.

Каждый раунд преобразования включает перестановку правой части блока, причем на вход модуля расширения подается 32-разрядное число, а снимается с него 48-разрядное (некоторые биты входного числа копируются на выход дважды). После сложения с материалом ключа осуществляется табличная подстановка, в результате которой по 8 таблицам подстановки происходит замена 48-битного входа на 32-битный выход (каждая S-таблица подстановки заменяет 6-битный вход на 4-битный выход). Выход S-блока поступает на блок перестановки, где биты переставляются по законам, определяемым специальной P-таблицей. Заканчивается раунд традиционным для сетей Фейстеля смешиванием ветвей между собой. За счет использования нелинейных операций преобразований удалось уменьшить количество раундов до 16 при размере ключа 56 бит.

Схема шифрования алгоритма DES указана ниже. Исходный текст - блок 64 бит. Процесс шифрования состоит из начальной перестановки, 16 циклов шифрования и конечной перестановки.

Начальная перестановка

Исходный текст T (блок 64 бит) преобразуется с помощью начальной перестановки IP которая определяется таблицей 5.1:

Таблица 5. 1.

Начальная перестановка IP

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

По таблице первые 3 бита результирующего блока IP(T) после начальной перестановки IP являются битами 58, 50, 42 входного блока T, а его 3 последние бита являются битами 23, 15, 7 входного блока.

Циклы шифрования

Полученный после начальной перестановки 64-битовый блок IP(T) участвует в 16 циклах преобразования Фейстеля.

- 16 циклов преобразования Фейстеля:

Разбить IP(T) на две части L_0 , R_0 где L_0, R_0 - соответственно 32 старших битов и 32 младших битов блока T_0 $IP(T)= L_0R_0$.

Пусть $T_{i-1}= L_{i-1}R_{i-1}$ результат (i-1) итерации, тогда результат i-ой итерации $T_i=L_iR_i$ определяется:

$$\begin{cases} L_i = R_{i-1}, \\ R_i = L_{i-1} \oplus F(R_{i-1}, K_i), i = 1,2,\dots,16; \end{cases}$$

Левая половина L_i равна правой половине предыдущего вектора $L_{i-1}R_{i-1}$. А правая половина R_i - это битовое сложение L_{i-1} и $f(R_{i-1},k_i)$ по модулю 2.

В 16-циклах преобразования Фейстеля функция f играет роль шифрования. Рассмотрим подробно функцию f.

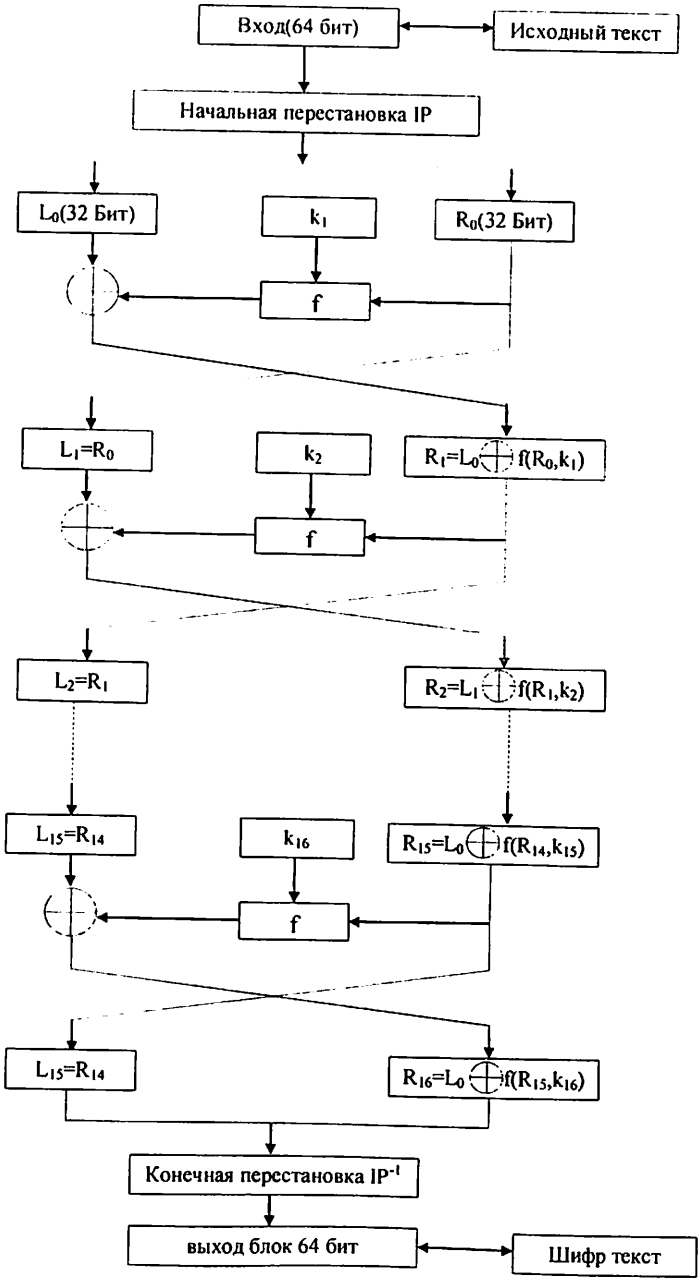


Рис. 5.1-Подробная схема шифрования алгоритма DES

Основная функция шифрования (функция Фейстеля)

Аргументами функции f являются 32-битовый вектор R_{i-1} и 48-битовый ключ k_i, который является результатом преобразования 56-битового исходного ключа шифра k.

Для вычисления функции f последовательно используются

- функция расширения E,
- сложение по модулю 2 с ключом k_i
- преобразование S, состоящее из 8 преобразований S - блоков $S_1,S_2,S_3...S_8$
- перестановка P.

Функция E расширяет 32-битовый вектор R_{i-1} до 48-битового вектора $E(R_{i-1})$ путём дублирования некоторых битов из R_{i-1}; порядок битов вектора $E(R_{i-1})$ указан в таблице 5.2.

Таблица 5.2.

Функция расширения E

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

Первые три бита вектора $E(R_{i-1})$ являются битами 32, 1, 2 вектора R_{i-1}. По таблице 2 видно, что биты 1, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21, 24, 25, 28, 29, 32 дублируются. Последние 3 бита вектора $E(R_{i-1})$ - это биты 31, 32, 1 вектора R_{i-1}. Полученный после перестановки блок $E(R_{i-1})$ складывается по модулю 2 с ключами k_i и затем представляется в виде восьми последовательных блоков $B_1,B_2,...B_8$.

$$E(R_{i-1})\oplus k_i=B_1B_2...B_8$$

Каждый B_j является 6-битовым блоком. Далее каждый из блоков B_j трансформируется в 4-битовый блок B'_j с помощью преобразований S_j. Преобразования S_j определяются таблицей 5.3.

Таблица 5.3

Преобразования i=1…8

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

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

Предположим, что B_3=101111, и мы хотим найти B'_3. Первый и последний разряды B_3 являются двоичной записью числа a, 0<=a<=3, средние 4 разряда представляют число b, 0<=b<=15. Строки таблицы S_3 нумеруются от 0 до 3, столбцы таблицы S_3 нумеруются от 0 до 15. Пара чисел (a, b) определяет число, находящееся в пересечении строки a и столбца b. Двоичное представление этого числа дает B'_3. В нашем случае $a=11_2=3$, $b=0111_2=7$, а число, определяемое парой (3,7), равно 7. Его двоичное представление B'_3=0111.

Значение функции $f(R_{i-1}, k_i)$(32 бит) получается перестановкой P, применяемой к 32-битовому блоку $B'_1 B'_2...B'_8$. Перестановка P задана таблицей 5.4.

Таблица 5.4.

Перестановка P

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

$f(R_{i-1},k_i)=P(B'_1B'_2...B'_8)$

Согласно таблице 4, первые четыре бита результирующего вектора после действия функции f - это биты 16, 7, 20, 21 вектора B'_1 B'_2...B'_8.

Генерирование ключей k_i.

Ключи k_i получаются из начального ключа k (56 бит = 7 байтов или 7 символов в ASCII) следующим образом. Добавляются биты в позиции 8, 16, 24, 32, 40, 48, 56, 64 ключа k таким образом, чтобы каждый байт содержал нечетное число единиц. Это используется для обнаружения ошибок при обмене и хранении ключей. Затем делают перестановку для расширенного ключа (кроме добавляемых битов 8, 16, 24, 32, 40, 48, 56, 64). Такая перестановка определена в таблице 5.6

Таблица 5.6

Объединение ключа

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

Эта перестановка определяется двумя блоками C_0 и D_0 по 28 бит каждый. Первые 3 бита C_0 есть биты 57, 49, 41 расширенного ключа. А первые три бита D_0 есть биты 63, 55, 47 расширенного ключа. C_i,D_i i=1,2,3...получаются из C_{i-1},D_{i-1} одним или двумя левыми циклическими сдвигами согласно таблице 5.6.

Таблица 5.6.

Циклы сдвига

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

Ключ k_i, i=1,...16 состоит из 48 бит, выбранных из битов вектора C_iD_i(56 бит) согласно таблице 5.7. Первый и второй биты k_i есть биты 14, 17 вектора C_iD_i.

Таблица 5.7.

Таблица перестановки

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

Конечная перестановка

Конечная перестановка IP^{-1} действует на T_{16} и является обратной к первоначальной перестановке. Конечная перестановка определяется таблицей 5.8.

Таблица 5.8.

Обратная 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

Схема расшифрования

При расшифровании данных все действия выполняются в обратном порядке. В 16 циклах расшифрования, в отличие от шифрования с помощью прямого преобразования сетью Фейстеля, здесь используется обратное преобразование сетью Фейстеля.

$$\begin{cases} R_i = L_{i-1}, \\ L_i = R_i \oplus F(L_i, K_i), i = 16,15,\dots,1; \end{cases}$$

Ключ k_i, i=16,...,1, функция f, перестановка IP и IP^{-1} такие же, как и в процессе шифрования.

Если в реализации ранее сгенерированные ключи не сохранялись, нужные нам ключи (C_i,D_i i=1,2,3...) получаются из C_{i-1},D_{i-1} правыми циклическими сдвигами согласно таблице 5.9.

Таблица 5.9.

Таблица перестановки

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

Число сдвига	0	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Advanced Encryption Standard – симметричный алгоритм блочного шифрования, принятый правительством США в качестве стандарта в результате конкурса, проведенного между технологическими институтами. Он заменил устаревший Data Encryption Standard, который больше не соответствовал требованиям сетевой безопасности, усложнившимся в XXI веке.

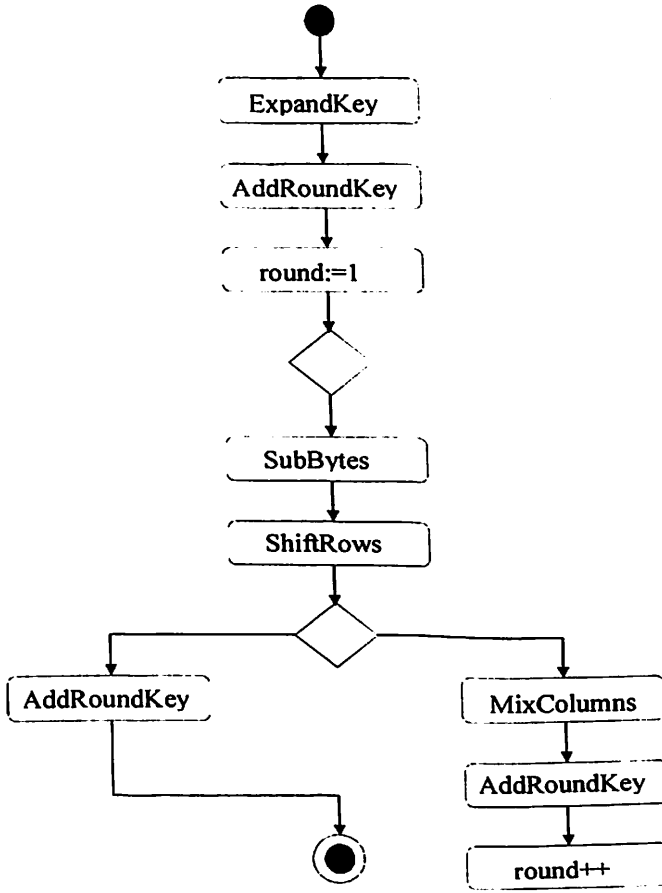


Рис.5.2 - Раунд алгоритма AES

Этот алгоритм, кроме аббревиатуры AES, иногда называют еще Rijndael – это анаграмма из частей имен бельгийских программистов Joan Daemen и Vinent Rijmen, которые разработали AES. Строго говоря, AES и Rijndael – не совсем одно и то же, поскольку AES имеет фиксированный размер блока в 128 бит и размеры ключей в 128, 192 и 256 бит, в то время как для Rijndael могут быть заданы любые размеры блока и ключа, от минимума в 32 бит до максимума в 256 бит.

Основные требования к кандидатам на стандарт AES:

- метод шифрования должен быть блочным;
- длина обрабатываемого блока должна равняться 128 битам;
- размерность ключей должна равняться 128, 192 или 256 битам.

N_r	N_b=4 128 bit	N_b=6 192 bit	N_b=8 256 bit
N_k=4 128 bit	10	12	14
N_k=6 192 bit	12	12	14
N_k=8 256 bit	14	14	14

Для шифрования каждого блока требуется как минимум 10 раундов, в каждый раунд входят следующие функции:

- KeyExpansion.
- AddKey
- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey
- SubBytes()

Преобразование SubBytes – это нелинейная замена байт, проводящаяся над каждым байтом state, используя таблицу замены S-box, см. табл. . Цель

применения таблицы замен затруднить линейный и дифференциальный криптоанализ. Таблица замен в алгоритме AES фиксированная. В табл. числа представлены в шестнадцатеричной системе счисления, в этой системе счисления любое значение байта представимо не более чем двумя шестнадцатеричными разрядами.

Замена байта по таблице S-box производится так: 1. Байт Z преобразуется в шестнадцатеричную систему счисления, например XYh, X — старший разряд, Y — младший разряд. Если старшего разряда нет — он заменяется нулем. 2. В S-box выбирается строка X и столбец Y. 3. Значение Z' на пересечении строки X и столбца Y таблицы S-box используется как замена Z. Например, для значения байта Z = 9Ah, заменой согласно таблицы S-box будет Z' = B8h. Полный процесс SubBytes состоит в замене всех 16 байт матрицы state.

X	Y															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**b**	**C**	**d**	**e**	**F**
0	63	7c	77	7b	12	6b	6f	C5	30	01	67	2b	Fe	d7	ab	76
1	ca	82	c9	7d	Fa	59	47	F0	ad	d4	a2	af	9c	a4	72	c0
2	b7	Fd	93	26	36	3f	F7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	62	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	Fc	b1	5b	6a	Cb	Be	39	4a	4c	58	cf
6	d0	Ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	Da	21	10	ff	f3	d2
8	Cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	18	73
9	60	81	4f	dc	22	2a	90	88	46	Ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	Ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	D5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	Ba	78	25	2e	1c	A6	b4	c6	e8	Dd	74	1f	4d	bd	8b	8a
d	70	3e	b5	66	48	03	F6	0e	61	35	57	b9	86	c1	1d	9e

e	e1	f8	98	11	69	D9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	Bf	E6	42	68	41	99	2d	0f	b0	54	bb	16

Процедура SubBytes() обрабатывает каждый байт состояния, независимо производя нелинейную замену байтов используя таблицу замен (S-box). Такая операция обеспечивает нелинейность алгоритма шифрования.

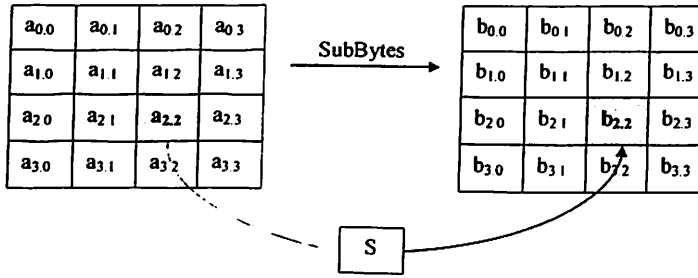


Рис.5.3-Процедура SubBytes()

ShiftRows()

В преобразовании ShiftRows байты в последних трех строках state циклически смещаются влево на различное число байт. Строка 1 (нумерация строк с нуля, см. рис.) смещается на один байт, строка 2 – на два байта, строка 3 – на три байта. На рис. проиллюстрировано применение преобразования ShiftRows к state. ShiftRows работает со строками State. При этой трансформации строки состояния циклически сдвигаются на r байт по горизонтали, в зависимости от номера строки.

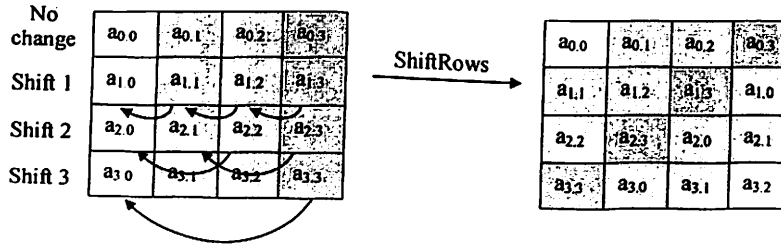


Рис.5.4- Процедура MixColumns

MixColumns()

В преобразовании MixColumns — перемешивание столбца — столбцы состояния (state) рассматриваются как полиномы над полем F(28) и умножаются по модулю x 4 + 1 на постоянный полином:

$$a(x) = 3x^3 + 1x^2 + 1x^2 + 2$$

Процесс умножения полиномов эквивалентен матричному умножению

$$\begin{bmatrix} S'_{0c} \\ S'_{1c} \\ S'_{2c} \\ S'_{3c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \bullet \begin{bmatrix} S_{0c} \\ S_{1c} \\ S_{2c} \\ S_{3c} \end{bmatrix}$$

В процедуре MixColumns четыре байта каждой колонки State смешиваются, используя для этого обратимую линейную трансформацию. MixColumns обрабатывает состояния по колонкам, трактуя каждую из них как полином третьей степени.

где c – номер столбца массива state и $0 \le c \le 3$. Операция сложения и умножения чисел выполняется по правилам, описанным в пунктах и.

В результате такого умножения, байты столбца с {S0c, S1c, S2c, S3c} заменяются, соответственно, на байты

$$S'_{0c} = (2 \bullet S_{0c}) \oplus (3 \bullet S_{1c}) \oplus S_{2c} \oplus S_{3c};$$

$$S'_{1c} = S_{0c} \oplus (2 \bullet S_{1c}) \oplus (3 \bullet S_{2c}) \oplus S_{3c};$$

$$S'_{2c} = S_{0c} \oplus S_{1c} \oplus (2 \bullet S_{2c}) \oplus (3 \bullet S_{3c});$$

$$S'_{3c} = (3 \bullet S_{0c}) \oplus S_{1c} \oplus S_{2c} \oplus (2 \bullet S_{3c}).$$

Преобразование применяется к каждому из четырех столбцов state.

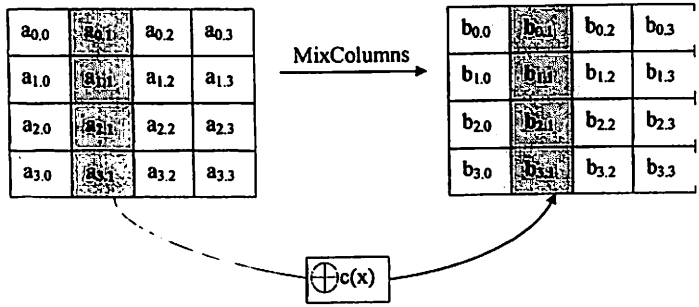


Рис.5.5- Процедура MixColumns

AddRoundKey()

В процедуре AddRoundKey, RoundKey каждого раунда объединяется со State. Для каждого раунда Roundkey получается из CipherKey с помощью процедуры KeyExpansion; каждый RoundKey такого же размера, что и State.

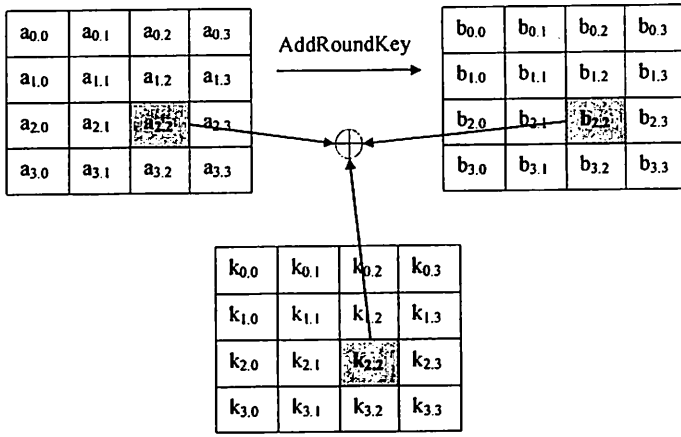


Рис.5.6- Процедура AddRoundKey()

OpenSSL — это криптографический инструментарий, реализующий сетевые протоколы Secure Sockets Layer (SSL v2/v3) и Transport Layer Security (TLS v1) и соответствующие им стандарты криптографии.

Программа **openssl** — это инструмент командной строки для использования различных криптографических функций криптографической библиотеки OpenSSL в консоли. Основны возможности:

- Создание и управление закрытыми ключами, открытыми ключами и параметрами.
- Криптографические операции с открытым ключом
- Создание сертификатов X.509, CSR и CRL
- Расчёт дайджестов сообщений
- Шифрование и дешифрование с помощью шифров
- Клиентские и серверные тесты SSL/TLS
- Обработка подписанной или зашифрованной почты S/MIME
- Запросы отметок времени, генерация и проверка

Приведу несколько примеров:

зашифруем файл, используя алгоритм des3

```
# openssl des3 -in file -out file.des3
```

расшифруем полученный файл

```
# openssl des3 -d -in file.des3 -out file
```

зашифруем файл, используя алгоритм blowfish(bf), и закодируем base64

```
# openssl bf -a -in file -out file.bf64
```

теперь расшифруем его и обработаем сразу же base64

```
# openssl bf -a -d -in file.bf64 -out file
```

Задание

1. Шифрование инициалов с помощью блочных шифров с использованием библиотеки OpenSSL (алгоритм 3DES)

число превосходит n, то его можно отбросить и сгенерировать еще одну последовательность бит.

Поэтому далее мы будем использовать термин генератор случайных чисел наравне с термином генератор случайных бит.

Генераторы случайных чисел по способу получения чисел делятся на:

- аппаратные;
- табличные;
- алгоритмические.

Табличные генераторы в качестве источника случайных чисел используют заранее подготовленные таблицы, содержащие проверенные некоррелированные числа и не являются генераторами в строгом понимании этого понятия. Недостатки такого способа очевидны: использование внешнего ресурса для хранения чисел, ограниченность последовательности, предопределенность значений. В качестве примера табличного метода можно привести книгу.

Аппаратные генераторы (истинно) случайных последовательностей должны обладать источником энтропии. Разработка генераторов, использующих источники энтропии, генерирующих не коррелированные и статистически независимые числа – достаточно сложная задача. Кроме того, для большинства криптографических приложений такой ГПСЧ не должен быть предметом изучения и воздействий противной стороны. О криптографически стойких генераторах случайных чисел речь пойдет в четвертом разделе.

Алгоритмический генератор является комбинацией физического генератора и детерминированного алгоритма. Такой генератор использует ограниченный набор данных, полученный с выхода физического генератора для создания длинной последовательности чисел преобразованиями исходных чисел. Данный вид генераторов представляет наибольший интерес в силу его очевидных преимуществ над генераторами случайных чисел других видов. Разбор свойств, достоинств и недостатков алгоритмических

ГПСЧ является основной темой данной книги.

```
#include <iostream>
unsigned int PRNG()
{
    // Наше стартовое число - 4 541
    static unsigned int seed = 4541;
    // Берем стартовое число и, с его помощью, генерируем новое значение.
    // Из-за использования очень больших чисел (и переполнения) угадать
следующее число исходя из предыдущего - очень сложно
    seed = (8253729 * seed + 2396403);
    // Берем стартовое число и возвращаем значение в диапазоне от 0 до 32767
    return seed % 32768;
}
int main()
{
    // Выводим 100 случайных чисел
    for (int count=0; count < 100; ++count)
    {
        std::cout << PRNG() << "\t";
        // Если вывели 5 чисел, то вставляем символ новой строки
        if ((count+1) % 5 == 0)
            std::cout << "\n";
}
}
```

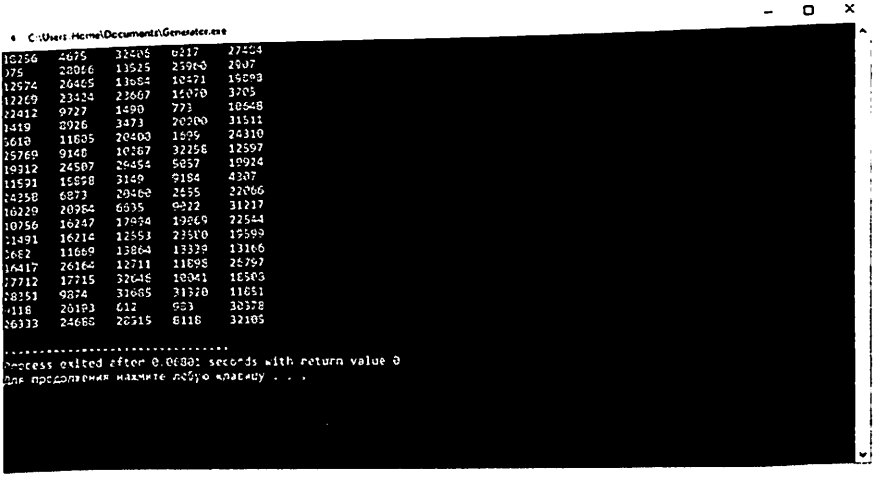


Рис. 6.1– Окно генерации последовательностей

№	Generator parametrlari
1.	a=4, b=7, c=9, d=5, m=13
2.	a=3, b=5, c=9, d=7, m=11
3.	a=7, b=7, c=4, d=3, m=31
4.	a=4, b=7, c=9, d=5, m=17
5.	a=7, b=5, c=9, d=8, m=13
6.	a=4, b=7, c=9, d=5, m=13
7.	a=11, b=6, c=9, d=5, m=17
8.	a=8, b=7, c=11, d=4, m=19
9.	a=6, b=3, c=9, d=8, m=19
10.	a=4, b=7, c=9, d=5, m=13
11.	a=8, b=7, c=9, d=5, m=11
12.	a=4, b=7, c=9, d=5, m=23
13.	a=4, b=5, c=9, d=5, m=13
14.	a=6, b=7, c=9, d=11, m=17
15.	a=6, b=8, c=4, d=3, m=19
16.	a=11, b=7, c=8, d=5, m=23
17.	a=11, b=7, c=9, d=8, m=29

18.	a=7, b=7, c=7, d=5, m=29
19.	a=9, b=7, c=9, d=5, m=23
20.	a=5, b=7, c=11, d=5, m=31
21.	a=8, b=7, c=9, d=11, m=17
22.	a=12, b=7, c=8, d=10, m=19
23.	a=12, b=7, c=3, d=10, m=19
24.	a=7, b=11, c=5, d=9, m=23
25.	a=12, b=5, c=9, d=7, m=17
26.	a=7, b=4, c=8, d=11, m=31
27.	a=10, b=6, c=12, d=11, m=29

Задание

1. Создайте последовательности на основе конгруэнтных генераторов, используя приведенные выше параметры, и вычисляйте период.

Контрольные вопросы

1. Каковы требования к генераторам, генерирующим псевдослучайные последовательности?

2. Перечислите криптографические алгоритмы, генерирующие псевдослучайные последовательности.

3. Можно ли использовать линейные конгруэнтные генераторы в криптографии для формирования псевдослучайной последовательности.

4. Могут ли алгоритмы блочного шифрования использоваться в качестве генератора для генерации псевдослучайной последовательности.

7-практическая работа

Тема: Разработка программу шифрования и дешифрования данных на основе алгоритма шифрования RC4

Цель работы: Теоретические и практические знания алгоритма шифрования RC4 приобретение навыков и умений.

Теоретическая часть

Шифрование есть процесс преобразования оригинального сообщения (часто именуемого открытым текстом) в шифрованный текст – строку, которая представляется случайным набором символов. Прочитать зашифрованное сообщение может только тот, кто знает, как привести шифрованный текст в исходное состояние (т.е. расшифровать его).

RC4 – это потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях (например, в протоколе SSL и для шифрования паролей в Windows NT). Шифр разработан компанией RSA Security Inc. и для его использования требуется лицензия. Автором RC4 является Рональд Ривест (Ronald Rivest). RC расшифровывается как Ron's Code или Rivest's Cipher. До 1995 года программный код RC4 нигде не публиковался.

Алгоритм RC4 строится как и любой потоковый шифр на основе параметризованного ключом генератора псевдослучайных битов с равномерным распределением. Основные преимущества шифра – высокая скорость работы и переменный размер ключа.

RC4 – это потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях (например, в протоколе SSL и для шифрования паролей в Windows NT). Шифр разработан компанией RSA Security Inc. и для его использования требуется лицензия. Автором RC4 является Рональд Ривест (Ronald Rivest). RC расшифровывается как Ron's Code или Rivest's Cipher. До 1995 года программный код RC4 нигде не публиковался.

Алгоритм RC4 строится как и любой потоковый шифр на основе параметризованного ключом генератора псевдослучайных битов с равномерным распределением. Основные преимущества шифра – высокая скорость работы и переменный размер ключа.

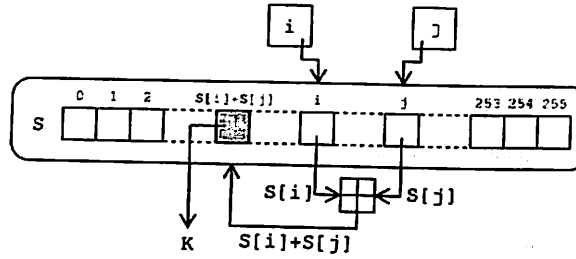


Рисунок 7.1 – Генератор ключевого потока RC4

Ядро алгоритма состоит из функции генерации ключевого потока. Эта функция генерирует последовательность битов, которая затем объединяется с открытым текстом посредством суммирования по модулю два. Расшифровка состоит из регенерации этого ключевого потока и суммирования его с шифрограммой по модулю два, восстанавливая исходный текст. Другая главная часть алгоритма – функция инициализации, которая использует ключ переменной длины для создания начального состояния генератора ключевого потока.

RC4 – фактически класс алгоритмов, определяемых размером его блока. Этот параметр n является размером слова для алгоритма. Обычно, n = 8, но в целях анализа можно уменьшить его. Однако для повышения безопасности необходимо увеличить эту величину. Внутреннее состояние RC4 состоит из массива размером 2n слов и двух счетчиков, каждый размером в одно слово. Массив известен как S-бокс, и далее будет обозначаться как S. Он всегда содержит перестановку 2n возможных значений слова. Два счетчика обозначены через i и j.

Алгоритм инициализации RC4 приведен ниже. Этот алгоритм использует ключ, сохраненный в Key, и имеющий длину l байт. Инициализация начинается с заполнения массива S, далее этот массив

перемешивается путем перестановок определяемых ключом. Так как только одно действие выполняется над S, то должно выполняться утверждение, что S всегда содержит все значения кодового слова.

Начальное заполнение массива:

for i = 0 to 2n − 1

S[i] = i

Скремблирование:

j = 0

for i = 0 to 2n − 1

j=(j+S[i]+Key[imodl]) mod2n

Перестановка (S[i], S[j])

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.

Инициализация:

i = 0

j = 0

Цикл генерации:

i = (i + 1) mod 2n

j = (j + S[i]) mod 2n

Перестановка (S[i], S[j])

Результат: K = S[(S[i] + S[j]) mod 2n]

Начальная установка генератора

	j=0	S=(0, 1, 2, 3, 4, 5, 6, 7)
i=0;	j=(0+0+0) mod8=0	
	$S_0 \leftrightarrow S_0$	S=(0, 1, 2, 3, 4, 5, 6, 7)
i=1;	j=(0+1+4) mod8=5	
	$S_5 \leftrightarrow S_1$	S=(0, 5, 2, 3, 4, 1, 6, 7)

i=2;	j=(5+2+0) mod8=7	
	$S_7 \leftrightarrow S_2$	S=(0, 5, 7, 3, 4, 1, 6, 2)
i=3;	j=(7+3+4) mod8=6	
	$S_6 \leftrightarrow S_3$	S=(0, 5, 7, 6, 4, 1, 3, 2)
i=4;	j=(6+4+0) mod8=2	
	$S_2 \leftrightarrow S_4$	S=(0, 5, 4, 6, 7, 1, 3, 2)
i=5;	j=(2+1+4) mod8=7	
	$S_7 \leftrightarrow S_5$	S=(0, 5, 4, 6, 7, 2, 3, 1)
i=6;	j=(7+3+0) mod8=2	
	$S_2 \leftrightarrow S_6$	S=(0, 5, 3, 6, 7, 2, 4, 1)
i=7;	j=(2+1+4) mod8=7	
	$S_7 \leftrightarrow S_7$	S=(0, 5, 3, 6, 7, 2, 4, 1)

Выработка случайных чисел псевдослучайной последовательности

i=1;	j=(0+5) mod8=5			
	$S_5 \leftrightarrow S_1$	S=(0, 2, 3, 6, 7, 5, 4, 1)	t=(5+2) mod8=1	z=2
i=2;	j=(5+3) mod8=0			
	$S_0 \leftrightarrow S_2$	S=(3, 2, 0, 6, 7, 5, 4, 1)	t=(0+3) mod8=3	z=6
i=3;	j=(0+6) mod8=6			
	$S_6 \leftrightarrow S_3$	S=(3, 2, 0, 4, 7, 5, 6, 1)	t=(6+4) mod8=2	z=0
i=4;	j=(6+7) mod8=5			
	$S_5 \leftrightarrow S_4$	S=(3, 2, 0, 4, 5, 7, 6, 1)	t=(7+5) mod8=4	z=5
i=5;	j=(5+7) mod8=4			
	$S_4 \leftrightarrow S_5$	S=(3, 2, 0, 4, 7, 5, 6, 1)	t=(5+7) mod8=4	z=7
i=6;	j=(4+6) mod8=2			
	$S_2 \leftrightarrow S_6$	S=(3, 2, 6, 4, 7, 5, 0, 1)	t=(0+6) mod8=6	z=0

i=7;	j=(2+1) mod8=3			
	$S_3 \leftrightarrow S_7$	S=(3, 2, 6, 1, 7, 5, 0, 4)	t=(4+1) mod8=5	z=5

z = (2, 6, 0, 5, 7, 0, 5)

В двоичном коде:

z = (010 110 000 101 111 000 101)

Контрольные вопросы

1. Общая классификация симметричных алгоритмов потока.

2. Алгоритмы, которые являются частью алгоритмов потокового шифрования?

8-практическая работа

Тема: Применение набор статистических тестов NIST для проверки случайности последовательностей

Цель работы: приобретение знаний и навыков о наборе статистических тестов NIST при проверке последовательностей на случайность.

Теоретическая часть

Тестирование генераторов случайных и псевдослучайных чисел (ГСЧ и ГПСЧ), используемых в криптографических приложениях, является актуальной задачей как в практическом, так и в теоретическом плане. Несмотря на значительные наработки в данной области, разработчики, тем не менее, нуждаются в удобном инструментарии, способном предоставить приемлемую метрику, которая позволит достаточно ясно исследовать степень случайности последовательностей, порождаемых ГСЧ (ГПСЧ), и обеспечить разработчиков достаточным объемом информации для принятия решения относительно "качества" генератора.

На сегодняшний день разработано достаточно большое количество различных типов ГСЧ (ГПСЧ). Однако для демонстрации их статистических свойств использовались различные подходы к статистическому тестированию. Чаще всего набор и методику тестирования предлагал сам разработчик генератора. Таким образом, сложилась ситуация, которая характеризуется тем, что невозможно объективно сравнить различные генераторы с единых позиций. Выходом из этого положения является использование некоторого стандартного набора статистических тестов, объединенных единой методикой расчета необходимых показателей эффективности ГПСЧ и принятия решения о случайности формируемых последовательностей.

Пакет NIST STS включает в себя 16 статистических тестов, которые разработаны для проверки гипотезы о случайности двоичных последовательностей произвольной длины, порождаемых ГСЧ или ГПСЧ. Все тесты направлены на выявление различных дефектов случайности.

Основным принципом тестирования является проверка нулевой гипотезы HR 0R, заключающейся в том, что тестируемая последовательность является случайной. Альтернативной гипотезой HR aR является гипотеза о том, что тестируемая последовательность не случайна. По результатам применения каждого теста нулевая гипотеза либо принимается, либо отвергается. Решение о том, будет ли заданная последовательность нулей и единиц случайной или нет, принимается по совокупности результатов всех тестов. Порядок тестирования отдельной двоичной последовательности S выглядит следующим образом.

1. Выдвигается нулевая гипотеза H_0 – предположение о том, что данная двоичная последовательность S случайна.

2. По последовательности S вычисляется статистика теста c(S).

3. С использованием специальной функции и статистики теста вычисляется значение вероятности $P = f(c(S))$, $P \in [0, 1]$.

4. Значение вероятности P сравнивается с уровнем значимости α, $\alpha \in [0{,}001, 0{,}01]$. Если $P \geq \alpha$, то гипотеза H_0 принимается. В противном случае принимается альтернативная гипотеза.

Как уже было сказано, пакет включает в себя 16 статистических тестов. Но фактически, в зависимости от входных параметров, вычисляется 189 значений вероятности P, которые можно рассматривать как результат работы отдельных тестов. В таблице 8.1 приводятся сводные данные по всем тестам с указанием количества вычисляемых значений вероятности P, физического смысла статистики теста и дефекта, на выявление которого направлен тест (в скобках дан порядковый номер теста, который используется на диаграммах).

Таблица 8.1

№	Статистический тест	Количество значений вероятности P	Статистика теста c(S)	Выявляемый дефект
1	Частотный	1 (1)	Нормализованная	Слишком много

	(монобитный) тест		абсолютная сумма значений элементов последовательности	нулей или единиц в последовательности
2	Частотный тест внутри блока	1 (2)	Мера согласования наблюдаемого количества единиц внутри блока с теоретически ожидаемым.	Локализованные отклонения частоты появления единиц в блоке от идеального значения ½.
3	Проверка накопленных сумм	2 (3-4)	Максимальное отклонение значения накопленной суммы элементов последовательности от начальной точки отсчета (точка 0)	Большое значение единиц или нулей в начале или в конце двоичной последовательности.
4	Проверка серий	1 (5)	Общее количество серий на всей длине последовательности.	Большое значение единиц или нулей в начале или в конце двоичной последовательности.
5	Проверка максимальной длины серии в блоке	1 (6)	Мера согласования наблюдаемого значения максимальной длины единичной серии с теоретически ожидаемым значением.	Отклонение от теоретического закона распределения максимальных длин серий единиц.
6	Проверка ранга двоичной матрицы	1 (7)	Мера согласования наблюдаемого значения рангов различного порядка с теоретически ожидаемым.	Отклонение эмпирического закона распределения значений рангов матрицы от теоретического, что указывает на

				зависимость символов в последовательности.
7	Проверка ранга двоичной матрицы	1 (8)	Нормализованная разница между наблюдаемым и ожидаемым количеством частотных компонент, которые превышают 95 % пороговый уровень.	Выявление периодических составляющих (трендов) в двоичной последовательности.
8	Проверка перекрывающихся шаблонов	1 (9)	Мера согласования наблюдаемого количества перекрывающихся шаблонов в последовательности с теоретическим значением.	Большое количество mбитных серий из единиц в последовательности.
9	Универсальный тест Маурера	1 (10)	Сумма логарифма расстояния между l-битными шаблонами.	Сжимаемость последовательности.
10	Универсальный тест Маурера	1 (11)	Мера согласования наблюдаемого значения энтропии источника с теоретически ожидаемым для случайного источника.	Неравномерность распределения m-битных слов в последовательности (регулярность свойств источника).
11	Проверка случайных отклонений	8 (12-19)	Мера согласования наблюдаемого количества визитов при случайном блуждании в заданное состояние внутри цикла с теоретически ожидаемым.	Отклонение от теоретического закона распределения визитов в конкретное состояние при случайном

				блуждании.
12	Проверка случайных отклонений (вариант)	18 (20-37)	Общее количество визитов в заданное состояние при случайном блуждании	Мера согласования наблюдаемого количества событий, заключающихся в появлении фиксированной длины эквивалентного ЛРР для заданного блока, с теоретически ожидаемым.
13	Последовательн ый тест	2 (38-39)	Мера согласования наблюдаемого количества всех встретившихся вариантов mбитных шаблонов с теоретически ожидаемым.	Неравномерность распределения m-битных слов в последовательност и.
14	Последовательн ый тест	1 (40)	Мера согласования наблюдаемого количества всех встретившихся вариантов mбитных шаблонов с теоретически ожидаемым.	Большая степень сжатия тестируемой последовательност и по сравнению с ожидаемой степенью сжатия для случайной последовательност и.
15	Проверка неперекрываю щихся шаблонов	148 (41-188)	Мера согласования наблюдаемого количества непериодических шаблонов в последовательности с	Большое количество заданных непериодических шаблонов в

			теоретическим значением.	последовательности.
16	Проверка линейной сложности	1 (189)	Мера согласования наблюдаемого количества событий, заключающихся в появлении фиксированной длины эквивалентного ЛРР для заданного блока, с теоретически ожидаемым.	Отклонение эмпирического распределения длин эквивалентных ЛРР для последовательности фиксированной длины от теоретического закона распределения для случайной последовательности, что указывает на недостаточную сложность тестируемой последовательности.

Рассматриваемый пакет статистических тестов может использоваться для решения следующих задач: – идентификация ГСЧ (ГПСЧ), которые формируют "плохие" двоичные последовательности; – разработка новых ГСЧ (ГПСЧ); – проверка корректности реализации ГСЧ (ГПСЧ); – изучение генераторов, описанных в стандартах; – исследование степени случайности реально используемых ГСЧ (ГПСЧ). При решении перечисленных задач применяют следующую методику тестирования генераторов.

1. Из множества аппаратных или программных генераторов выбирают генератор G, который необходимо оценить и принять решение о том, что он формирует случайные двоичные последовательности. Генератор должен порождать двоичную последовательность $S = \{s_1, s_2, \ldots, s_n\}$, $s_i \in \{0,1\}$, произвольной длины n.

2. Для фиксированного значения n формируют множество из m двоичных последовательностей:

$S_1 = \{s_1, s_2, \ldots, s_n\};$

$S_2 = \{s_1, s_2, \ldots, s_n\};$

.....

$S_m = \{s_1, s_2, \ldots, s_n\}.$

Таким образом, для тестирования необходимо сформировать выборку объемом N = m × n.

3. Каждую последовательность подвергают тестированию с использованием пакета NIST STS. В результате формируется статистический портрет генератора следующего вида

№ теста j / № пос-ти i	1	2	…	q
S_1	P_{11}	P_{12}		P_{1q}
S_2	P_{21}	P_{22}		P_{2q}
:	:			
S_m	P_{m1}	P_{m2}		P_{mq}

$$\begin{pmatrix} P_{11} & P_{12} & \Lambda & P_{1q} \\ P_{21} & P_{22} & \Lambda & P_{1q} \\ M & M & O & M \\ P_{m1} & P_{m2} & \Lambda & P_{mq} \end{pmatrix}$$

Контрольные вопросы

1. Классификация критерия "Xi-квадрат" проверки распределения на случайность.

2. Виды тестов, определяющих степень случайности 3. Классификация генераторов псевдослучайных последовательностей.

9-практическая работа

Тема: Вычислить хеш-значение данных с помощью библиотеки Openssl

Цель работы: получение практических знаний о сферах применения и принципах работы алгоритма хэширования и навыков.

Теоретическая часть

Хэширование (иногда хеширование, англ. hashing) – преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины. Такие преобразования также называются хэш-функциями или функциями свёртки, а их результаты называют хэшем, хэш-кодом или дайджестом сообщения (англ. message digest).

Хэширование применяется для сравнения данных: если у двух массивов хэш-коды разные, массивы гарантированно различаются; если одинаковые – массивы, скорее всего, одинаковы. В общем случае, однозначного соответствия между исходными данными и хэш-кодом нет в силу того, что количество значений хэш-функций меньше, чем вариантов входного массива. Также существует множество массивов, дающих одинаковые хэш-коды – так называемые коллизии. Вероятность возникновения коллизий играет немаловажную роль в оценке качества хэш-функций. Существует множество алгоритмов хэширования с различными характеристиками (разрядность, вычислительная сложность, криптостойкость и т.п.). Выбор той или иной хэш-функции определяется спецификой решаемой задачи. Простейшими примерами хэш-функций могут служить контрольные суммы.

Контрольные суммы – это несложные, крайне быстрые и легко реализуемые аппаратные алгоритмы, используемые для защиты от непреднамеренных искажений, в том числе ошибок аппаратуры. По скорости вычисления они в десятки и сотни раз быстрее, чем криптографические хэшфункции, и значительно проще в аппаратной реализации. Платой за столь высокую скорость является отсутствие криптостойкости – очень просто

подобрать сообщение под заранее известную сумму. Разрядность контрольных сумм (обычно 32 бита) ниже, чем криптографических хэшей (типичные значения – 128, 160 и 256 бит), что означает возможность возникновения непреднамеренных коллизий.

MD5 (англ. Message Digest 5) – 128-битный алгоритм хэширования, разработанный Р. Ривестом из Массачусетского технологического института (MIT) в 1991 году [1, 5]. Предназначен для создания «отпечатков» или «дайджестов» сообщений произвольной длины. Является улучшенной в плане безопасности версией алгоритма MD4. Используется для проверки подлинности опубликованных сообщений посредством сравнения дайджеста сообщения с опубликованным хэшем. Эту операцию называют «проверкой хэша». Алгоритм MD5 разработан таким образом, чтобы быть достаточно быстрым для выполнения на 32-разрядном процессоре. Алгоритм не требует больших таблиц подстановок и может быть закодирован весьма компактно. При описании алгоритма под термином слово понимается 32-битная последовательность, а под термином байт – 8-битная последовательность. Последовательность бит может быть интерпретирована естественным образом как последовательность байт, где каждая последовательная группа из 8 бит представляет собой 1 байт. Внутри байта биты располагаются следующим образом: сначала (слева) перечисляются более значимые биты (старшие биты, соответствующие более высокой степени двойки:2^7 ,2^8 , ...), а в конце (справа) оказываются наименее значимые биты (младшие, соответствующие 2^2,2^1,2^0). Такой порядок расположения бит (или байт) называется big-endian (порядок от старшего к младшему). Последовательность байт может быть интерпретирована как последовательность 32-битных слов, где каждая последовательная группа из 4 байт представляет собой 1 слово. Внутри слова байты располагаются следующим образом: сначала идут наименее значимые байты, затем – наиболее. Такой порядок расположения бит (или байт) называется little-endian (порядок от младшего к старшему).

Например, пусть есть последовательность бит (выделена полужирным шрифтом):

|0 0 0 1 0 0 0 1||0 0 1 0 0 0 1 0||0 0 1 1 0 0 1 1||0 1 0 0 0 1 0 0|...

|7 6 5 4 3 2 1 0||7 6 5 4 3 2 1 0||7 6 5 4 3 2 1 0||7 6 5 4 3 2 1 0

Тогда она может быть интерпретирована как 4 подряд расположенных байта (0x11,0x22,0x33,0x44 – шестнадцатеричные числа) или как одно слово 0x44332211.

Пусть символ '+' обозначает сложение по модулю 2^{32}, т.е. $a+b\equiv(a+b)(\mathrm{mod}2^{32})$ Пусть выражение 'X<<<s' означает циклический сдвиг бит числа на позиций влево.

Пусть имеется сообщение M длиной в b бит, хэш-функцию которого нужно вычислить. Здесь b может быть любым неотрицательным целым числом, не обязательно кратным 8. Тогда сообщение может быть представлено в виде последовательности бит: $m_0m_1m_2\ldots m_{b-2}m_{b-1}$.

Шаг 1. Добавление дополнительных бит.

К концу сообщения добавляется бит 1, вслед за которым добавляются нулевые биты (0) до тех пор, пока длина дополненного сообщения не станет удовлетворять условию: $L\equiv 448(\mathrm{mod}512)$. Таким образом, может быть добавлено от 1 до 512 бит.

После этого нужно преобразовать сообщение из последовательности бит, сгруппированных в байты, в последовательность слов согласно порядку расположения бит little-endian.

Шаг 2. Добавление исходной длины сообщения.

К результату предыдущего шага добавляются младшие 64 бита, взятые из побитного представления числа b (исходная длина сообщения). При этом сначала дописываются младшие 4 байта, а затем – старшие. В итоге получается сообщение, длина которого кратна 512 битам, т.е. полученное сообщение можно разбить на блоки, каждый из которых представляется в виде шестнадцати 32-битных слов: $M_0,M_1,M_2,\ldots,M_{N-1}$, где кратно 16.

Шаг 3. Инициализация буфера.

Буфер состоит из четырёх 32-битных слов (A,B,C,D) и используется для вычисления хэшзначения. В начале работы алгоритма буфер инициализируется следующими значениями:

A: 01 23 45 67, т.е. A = 0x67452301

B: 89 AB CD EF, т.е. B = 0xEFCDAB89

C: FE DC BA 98, т.е. C = 0x98BADCFE

D: 76 54 32 10, т.е. D = 0x10325476

Также определяются четыре функции, которые будут использоваться в дальнейшем (здесь '∧','∨','⊕' и '¬' – побитные операции):

F (x, y, z) = (x∧y) ∨ (¬x∧z)

G (x, y, z) = (x∧z) ∨ (¬z∧y)

H (x, y, z) = x⊕y⊕z

I (x, y, z) = y⊕ (¬z∧x)

Наконец, определяется таблица констант T[1...64], i-й элемент которой задается следующим образом:

T[i]=int(4 294 967 296|$sin(i)$|), где i – в радианах, а 4 294 967 296 = 2^{32}.

Шаг 4. Основной цикл. Каждая итерация внешнего цикла (по переменной) соответствует обработке одного 512-битного блока сообщения. Алгоритм обработки сообщения на псевдокоде имеет вид:

For i = 0 to N/16-1 do

// Копирование блока сообщения номер i в массив X

For j = 0 to 15 do

X[j] = M[i*16 + j]. end

Шаг 5. Результат. Результатом вычисления хэш-функции являются биты слов A,B,C,D, т.е. биты результата начинаются с младшего байта слова и заканчиваются старшим битом слова D.

Семейство алгоритмов SHA (Secure hash standard) включает в себя 5 алгоритмов вычисления хэш-функции: SHA–1, SHA–224, SHA–256, SHA–384, SHA–512. Четыре последние хэш-функции объединяются в подсемейство SHA–2. Алгоритм SHA–1 разработан Агентством

национальной безопасности США (NSA) в 1995 году. Алгоритмы подсемейства SHA–2 также разработаны Агентством национальной безопасности США и опубликованы Национальным институтом стандартов и технологий в федеральном стандарте обработки информации FIPS PUB 180–2 в августе 2002 года. Эти алгоритмы используются в SSL, SSH, S/MIME, DNSSEC, X.509, PGP, IPSec, при передаче файлов по сети (BitTorrent) [1, 2, 3, 6, 7].

Между собой алгоритмы отличаются криптостойкостью, которая обеспечивается для хэшируемых данных, а также размерами блоков и слов данных, используемых при хэшировании. Основные отличия алгоритмов можно представить в виде таблицы.

Алгоритм	Длина дайджеста сообщения (бит)	Длина внутреннего состояния (бит	Длина блока (бит)	Длина сообщения (бит)	Длина слова (бит)	Количество итераций в цикле
SHA–1	160	160	512	$<2^{64}$	32	80
SHA–224	224	224	512	$<2^{64}$	32	64
SHA–256	256	256	512	$<2^{64}$	32	64
SHA–384	384	384	1024	$<2^{64}$	64	80
SHA–512	512	512	1024	$<2^{64}$	64	80

Алгоритм SHA обладает тем свойством, что каждый бит хэш-кода зависит от всех битов хэшируемых данных. Сложное многократное использование базовых функций в результате дает хорошее перемешивание, это означает, что практически невероятно, чтобы два набора входных данных породили один и тот же хэш-код, несмотря на то, что они оказываются подобными по структуре [Шнайер].

Основные характеристики алгоритма SHA приведены в таблице.

Основные характеристики SHA

Длина хэш-кода	160 бит

Длина обрабатываемых блоков	512 бит
Число шагов алгоритма	80 (4 раунда по 20 шагов)
Максимальная длина хэшируемых данных	2^{64}-1
Число базовых функций	4
Число аддитивных констант	4

Вычисление значения хеш-функции в соответствии с алгоритмом SHA–1 происходит следующим образом (схему алгоритма в рисунках см. в презентации к лекции).

1. На вход поступает k-битовый блок данных, где $k < 2^{64}$.

2. k-битовый блок дополняется так, чтобы его длина стала кратной 512 разрядам (данные обрабатываются 512-битовыми блоками). Структура дополнения следующая: 100...0 (от 1 до 512 бит).

3. К полученному результату добавляется 64-битовое представление длины исходного блока данных.

4. Инициализируются пять 32-разрядных переменных:

$$A = 0x67452301$$

$$B = 0xefcdab89$$

$$C = 0x98badcfe$$

$$D = 0x10325476$$

$$E = 0xc3d2e1f0$$

5. Производится обработка 512-битовых блоков данных в 4 раунда по 20 операций каждый.

На рисунке (см. в презентации к лекции) представлена схема одной операции SHA. Циклический сдвиг влево на s разрядов обозначен « s; W_t – подблок дополненного сообщения такой, что:

$$W_t = M_t \ (0 \le t \le 15), \text{ где } M_t - 32\text{-битовый блок данных}$$

$$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \text{ « } 1 \quad (16 \le t \le 79).$$

Соответствие аддитивных констант K_t и нелинейных функций F_t номеру операции представлено в таблице (см. презентацию к лекции).

6. Значения переменных a, b, c, d, e складываются, соответственно, с A, B, C, D, E.

7. Обрабатывается следующий блок данных.

8. Окончательный результат получается конкатенацией значений A, B, C, D, E.

На выходе получается 160-битовый хэш-код.

Контрольные вопросы

1. Что такое хэш-функции и их функции.

2. Опишите хэш-функцию MD5.

3. Перечислите математические операции, используемые в хэш-функции SHA1.

10-практическая работа

Тема: Определение хеш-значений с использованием метода полного перебора ключа.

Цель работы: приобретение знаний и умений по методу полного подбора ключей к хэш-значениям.

Теоретическая часть

Эго означает, что независимо от того, сколько времени потратит противник на расшифровку, ему не удастся расшифровать шифрованный текст просто потому, что в шифрованном тексте нет информации, требуемой для восстановления открытого текста. Среди алгоритмов шифрования абсолютно стойких нет. Таким образом, максимум, чего может ожидать пользователь от того или иного алгоритма шифрования, это выполнение хотя бы одного из двух следующих критериев защищенности.

1. Стоимость взлома шифра превышает стоимость расшифрованной информации.

2. Время, которое требуется для того, чтобы взломать шифр, превышает время, в течение которого информация актуальна.

Алгоритм шифрования называется защищенным по вычислениям, если он соответствует обоим вышеуказанным требованиям.

Если известен алгоритм шифрования и есть хотя бы одна пара открытый - шифрованный текст, то самым естественным способом анализа, который сразу приходит в голову, является последовательное опробование всех возможных вариантов ключа, которые могли быть использованы. Опробование производят до тех пор, пока зашифрование открытого текста на очередном ключе не приведет к получению имеющегося шифрованного сообщения. Такой способ анализа в разных источниках литературы имеет разные названия, например «метод атаки в лоб», или «метод полного перебора», или «метод грубой силы» или «Brut-force атака». У этого метода есть одно неоспоримое преимущество: рано или поздно искомый ключ будет найден и для этого будет необходим минимальный набор данных. Быстрота нахождения ключа будет зависеть от длины используемого секретного ключа

и от вычислительной мощи, которая есть в наличии у аналитика, а также от доли везения. Ведь может случиться так, что искомый ключ встретится одним из первых. Рассмотрим уравнение относительно ключа k∈K при известной паре (x, y), где x∈X и y∈Y:

$$T(x,k) = y$$

Основной характеристикой криптографической стойкости шифра относительно того или иного метода анализа является трудоемкость E(r) этого метода. В качестве меры трудоемкости раскрытия шифров обычно используется количество элементарных операций, необходимых для расшифрования сообщения или определения ключа. Под элементарной операцией понимают операцию, выполняемую на конкретной аппаратуре за один шаг ее работы. Трудоемкость расшифрования определяется объемом и характером информации, доступной криптоаналитику.

Пусть для простоты для любой пары (x, y) существует единственное значение k, удовлетворяющее. Упорядочим множество K в соответствии с заданным порядком и будем последовательно проверять ключи из K на предмет равенства в уравнении. Если считать проверку одного варианта ключа k∈K в уравнении за одну операцию, то полный перебор ключей потребует |K| операций. Пусть ключ в схеме шифрования выбирается случайно и равновероятно из множества K. Тогда с вероятностью - трудоемкость метода полного перебора равна 1. Это происходит в том случае, когда случайно выбран ключ, расположенный в нашем порядке на первом месте. Поэтому естественно в качестве оценки трудоемкости метода взять математическое ожидание числа шагов в переборе до попадания на использованный ключ. Найдем среднее число шагов в методе полного перебора, когда порядок фиксирован, а выбор ключа случаен и равновероятен. Пусть случайная величина т - число опробований включительно до момента обнаружения использованного ключа. При i = 1, ..., |K| случайные величины n^ = 1, если использованный ключ находится в порядке на месте i и ^ = 0 в противном случае. Тогда

$$E_r = \sum_{i=1}^{|K|} iP(\xi_i = 1)$$

Если считать, что все ключи расположены в установленном порядке, то процедуру равновероятного выбора ключа можно представлять как равновероятный выбор числа i в последовательности натуральных чисел 1, |K|.

Пример продолжительности подбора паролей

В таблице представлено оценочное время полного перебора паролей в зависимости от их длины. Предполагается, что в пароле могут использоваться 36 различных символов (латинские буквы одного регистра + цифры), а скорость перебора составляет 100 000 паролей в секунду (класс атаки B, типичный для восстановления пароля из кэша Windows (.PWL файлов) на Pentium 100).

Количество символов	Количество паролей	Стойкость	Время подбора
1	36	5 бит	менее секунды
2	1296	10 бит	менее секунды
3	46 656	15 бит	менее секунды
4	1 679 616	21 бит	17 секунд
5	60 466 176	26 бит	10 минут
6	2 176 782 336	31 бит	6 часов
7	78 364 164 096	36 бит	9 дней
8	$2{,}821\ 109\ 9 \times 10^{12}$	41 бит	11 месяцев
9	$1{,}015\ 599\ 5 \times 10^{14}$	46 бит	32 года
10	$3{,}656\ 158\ 4 \times 10^{15}$	52 бита	1 162 года
11	$1{,}316\ 217\ 0 \times 10^{17}$	58 бит	41 823 года
12	$4{,}738\ 381\ 3 \times 10^{18}$	62 бита	1 505 615 лет

RainbowCrack — компьютерная программа для быстрого взлома хешей. Является реализацией техники Филиппа Окслина *faster time-memory trade-off*. Она позволяет создать базу предсгенерированных LanManager хешей, с помощью которой можно почти мгновенно взломать практически любой алфавитно-цифровой пароль.

Программное обеспечение RainbowCrack состоит из следующих частей:

- rtgen — программа для генерации радужных таблиц.
- rtsort- программа для сортировки радужных таблиц, порождённых rtgen.
- rcrack — программа для поиска радужных таблиц, упорядоченных по rtsort.

Ighashgpu: взлом с помощью GPU

Но хватит теории. Давай перейдем к делу и поговорим непосредственно о взломе нашего любимого алгоритма. Предположим, что нам в руки попал хеш какого-то пароля: d8578edf8458ce06fbc5bb76a58c5ca4. Для взлома этого хеша я предлагаю воспользоваться программой Ighashgpu, которую можно скачать на сайте www.golubev.com или найти на нашем диске. Утилита распространяется совершенно бесплатно и спокойно работает под виндой. Чтобы ускорить процесс взлома хеша, Ighashgpu использует GPU, поэтому тебе необходима как минимум одна видеокарта nVidia или ATI с поддержкой CUDA/ATI Stream. Современные графические процессоры построены на несколько иной архитектуре, нежели обычные CPU, поэтому они гораздо эффективнее обрабатывают графическую информацию. Хотя GPU предназначены для обработки трехмерной графики, в последние несколько лет появилась тенденция к их применению и для обычных вычислений. Начать работать с программой не просто, а очень просто: распакуй архив в любое место на диске и приступай к взлому с помощью командной строки Windows:

```
ighashgpu.exe -t:md5 \
```

```
ighashgpu.exe -t:md5 \
  -h:d8578edf8458ce06fbc5bb76a58c5ca4 -max:7
```

Мы используем вышеприведенный способ для взлома одного определенного хеша, сгенерированного при помощи алгоритма MD5. Максимальная длина возможного пароля составляет семь символов. Через какое-то время пароль будет найден (qwerty). Теперь давай попробуем взломать еще один хеш, но с немного другими условиями. Пусть наш хеш имеет вид d11fd4559815b2c3de1b685bb78a6283, а включает в себя буквы, цифры, знак подчеркивания и имеет суффикс «_admin». В данном случае мы можем использовать перебор пароля по маске, чтобы упростить программе задачу:

```
ighashgpu.exe -h:d11fd4559815b2c3de1b685bb78a6283 -t:md5
-u:[abcdefghijklmnopqrstuwvxyz1234567890_] -m:??????_admin
```

Здесь параметр '-u' позволяет указать набор символов, используемых при переборе, а параметр '-m' задает маску пароля. В нашем случае маска состоит из шести произвольных символов, после которых идет сочетание «_admin». Подбор пароля также не составит никакого труда.

Коллизии

Коллизией в криптографии называют два разных входных блока данных, которые для одной и той же хеш-функции дают один и тот же хеш. Каждая функция на выходе дает последовательность битов определенной длины, которая не зависит от размера первоначальных данных. Отсюда следует, что коллизии существуют для любого алгоритма хеширования. Однако вероятность того, что ты сможешь найти коллизию в «хорошем» алгоритме, практически стремится к нулю. К сожалению или к счастью, алгоритмы хеширования могут содержать ошибки, как и любые программы. Многие хеш-функции либо уже были сломаны, либо скоро будут. В данном случае «сломать» — значит найти коллизию за время, которое много меньше заявленной бесконечности.

Теперь давай попробуем взломать сразу несколько паролей одновременно. Предположим, что к нам в руки попала база данных хешей паролей. При этом известно, что каждый пароль оканчивается символами c00l:

```
f0b46ac8494b7761adb7203aa7776c2a
f2da202a5a215b66995de1f9327dbaa6
c7f7a34bbe8f385faa89a04a9d94dacf
cb1cb9a40708a151e6c92702342f0ac5
00a931d3facaad384169ebc31d38775c
4966d8547cce099ae6f666f09f68458e
```

Сохрани хеши в файле encrypted.dat и запусти Ighashgpu как указано ниже:

```
ighashgpu.exe -t:md5 -u:[abcdefghijklmnopqrstuwvxyz1234567890_]
 -m:??????c00l encrypted.dat
```

После завершения работы программы в папке Ighashgpu появится файл ighashgpu_results.txt со взломанными паролями:

```
f0b46ac8494b7761adb7203aa7776c2a:1rootxc00l
f2da202a5a215b66995de1f9327dbaa6:pwd12xc00l
c7f7a34bbe8f385faa89a04a9d94dacf:pwd34yc00l
cb1cb9a40708a151e6c92702342f0ac5:pwd56yc00l
4966d8547cce099ae6f666f09f68458e:pwd98zc00l
00a931d3facaad384169ebc31d38775c:pwd78zc00l
```

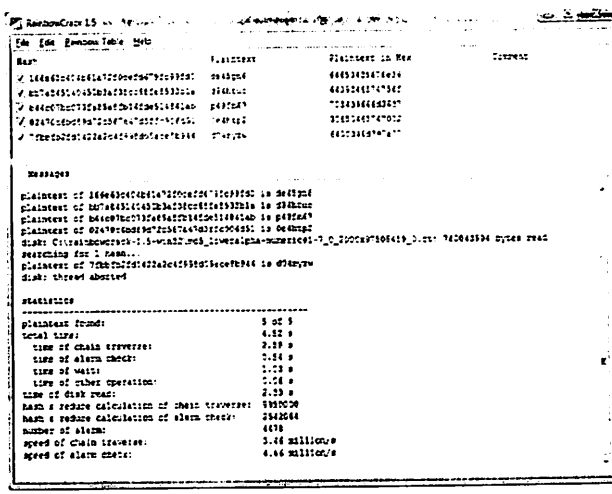


Рис.10.1 – Окно перебора хэш-значений

Задание

	Хеш-значение	Открытые тексты
1.	f97c5d29941bfb1b2fdab0874906ab82	
2.	c4988dbf8716d62d17180dd114f15c71	
3.	66058ec26cd62e957604145e726b0f0b	
4.	b8a9f715dbb64fd5c56e7783c6820a61	
5.	35d6d33467aae9a2e3dccb4b6b027878	
6.	7f46165474d11ee5836777d85df2cdab	
7.	0800fc577294c34e0b28ad2839435945	
8.	7f5f2244206f1ebac53dc0a333176b68	
9.	aa2d6e4f578eb0cfaba23beef76c2194	
10.	755f85c2723bb39381c7379a604160d8	
11.	bae60998ffe4923b131e3d6e4c19993e	
12.	4af3318a57ca352daaac11aa778ff083	
13.	341be97d9aff90c9978347f66f945b77	
14.	639bae9ac6b3e1a84cebb7b403297b79	
15.	ab86a1e1ef70dff97959067b723c5c24	

16.	0d149b90e7394297301c90191ae775f0	
17.	d850f04cdb48312a9be171e214c0b4ee	
18.	6057f13c496ecf7fd777ceb9e79ae285	
19.	444bcb3a3fcf8389296c49467f27e1d6	
20.	1a1dc91c907325c69271ddf0c944bc72	
21.	c47d187067c6cf953245f128b5fde62a	
22.	63a9f0ea7bb98050796b649e85481845	
23.	98bd1c45684cf587ac2347a92dd7bb51	

Контрольные вопросы

1. Перечислите преимущества и недостатки метода полного подбора ключей.
2. Какие шаги необходимо предпринять для защиты от этой угрозы

Список литературы

1. Akbarov D.E. Axborot xavfsizligini ta'minlashning kriptografik usullari va ularning qo'llanilishi. Toshkent, 2008 – 394 bet.

2. G'aniev S. K., Karimov M. M., Tashev K. A. Axborot xavfsizligi. Axborot- kommunikatsiya tizimlar xavfsizligi. Oliy o'quv yurt talabalari uchun mo'ljallangan. "Aloqachi", 2008.

3. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си //М.: Триумф. – 2002. – Т. 816. – С. 3.

СОДЕРЖАНИЕ