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

60612100 - “ Инжениринг по кибербезопасности”

Ташкент 2022

Авторы: Старший преподаватель кафедры "Криптология" Б.М.Шамшиева, ассистент О.О.Турсунов, Ш.Ж.Хамидов, У.У.Тоджиакбарова методическая пособия по выполнению практических работ по учебной дисциплине «Криптографии 1» для студентов бакалавра специальностей информационной безопасности - Ташкент: ТУИТ. 2022.-160 с.

В данной методической пособии представлен материал, необходимый для введения в теорию криптографических 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 № ___ от "___" "_____" 20__ г.)

Ташкентский университет информационных технологий имени Мухаммада аль-Хорезми, 2022

Практическая работа № 1

Тема: Математические основы криптографии

Цель работы: Является знакомство с криптографией и изучение применения в ней основ математики, таких как простые числа, модулярная арифметика, системе счисления и логические функции.

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: числа вида 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 $b \neq 0$, 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 $b \neq 0$. Если $a \neq 0$, то очевидно, что множество всех делителей числа a конечное.

Утверждение о том, что b делит a обозначают символом $b \mid a$. Если b не делит a, то этот факт обозначают $b \nmid a$.

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=a d:

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

В этом случае число a называют 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±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

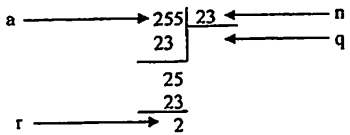
$$a = q \times n + r$$

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 «е», 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 коэфициенты и α и β.

Применяется 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	$0{,}64_2$
1	0	0	1	0	0	0	0	1		× 2 = 1,28
										× 2 = 0,56
										× 2 = 1,12
										× 2 = 0,24
										× 2 = 0,48
										× 2 = 0,96
										× 2 = 1,96

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

289 | 8
288 | 36 | 8
1 | 4

$0{,}64_2$
× 8 = 5,12
× 8 = 0,96
× 2 = 7,69

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

$0{,}501=5*8^{-1}+1*8^{-3}=0{,}625+0{,}002\approx0{,}64$

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

```
_11110 | 110
 110   |-----
 ---   | 101
  110
  ---
  110
  ---
    0
```

```
_30 | 8
 24 |---
 -- | 3
  6
```

```
_36 | 6
 36 |---
 -- | 5
  0
```

$30:6_{(10)}=5$ $\qquad\qquad$ $36_{(8)}:6_{(8)}=5_{(8)}$

$11110:110_{(2)}$

Таблица 1.2 Действия в двоичной системе счисления

Сложение	Вычитание	Умножение
0+0=0	0-0=0	0*0=0
0+1=1	1-0=1	0*1=0
1+0=1	10-1=1	1*0=0
1+1=10		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 1.3 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 1.3 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 1.3 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 сложное 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 выраженная 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 → 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 ↔ B» истинно тогда и только тогда, когда A и B равны.

Обозначение: F = A ↔ B.

Таблица истинности для эквивалентности

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

б) Операция XOR (исключающие или)

«A ⊕ B» истинно тогда, когда истинно A или B, но не оба одновременно.

Эту операцию также называют "сложение по модулю два".

Обозначение: F = A ⊕ 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?	$X_2 -> Y_{10}$	$X_8 -> Y_{10}$	$X_{16} -> Y_{10}$
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
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;	a=-74;	n=47;	11111010	3112	B1D

	c=5; n=8;	n=69;	e=6;			
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, простые числа и приведите примеры?
2. Объясните на примере свойства работы модуля?
3. Целые числа и их использование в криптографии?

Практическая работа №2

Тема: Анализ шифров основанных на одинарной перестановке

Цель работы: Изучить способы шифрования данных различными перестановками и подстановками. Изучить метод частотного анализа шифр текста.

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:

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

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

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 $\{0,1,2,\ldots, N^2\text{-}1\}$. 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
Ш	И	Ф	Р	_

И	К	Л	Ч	Ю
_	Ш	И	Р	Ф

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

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

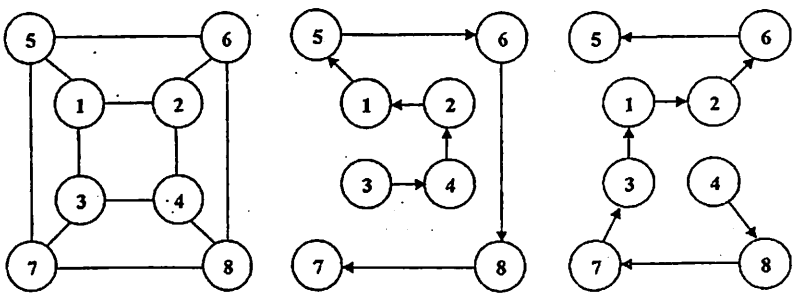
Записать сообщение в прямоугольную таблицу по маршруту: по горизонтали, начиная с верхнего левого угла, поочередно слева направо. Сообщение будем списывать по маршруту: по вертикалям, начиная с верхнего правого угла, поочередно сверху вниз.

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

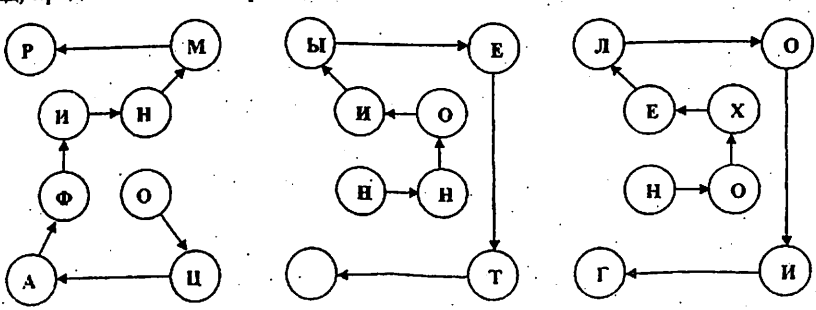
Другим методом перестановки является метод перестановки на основе применения маршрутов Гамильтона.



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. 2.1.



Маршрут № 2 Маршрут № 1 Маршрут № 1

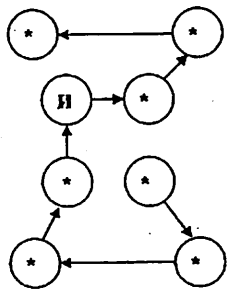


Рис. 2.2. Пример шифрования с помощью маршрутов Гамильтона

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	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Задание

Произвести шифрование имени, фамилии и отчества выше указанными методами шифрования.

Контрольные вопросы

1. Метод простой перестановки с помощью ключа?
2. Метод простой перестановки с помощью ключевыми словами?
3. Что такое Аффинный шифр?
4. Что такое Биграммный шифр?

Практическая работа №3

Тема: Деление и умножение многочленов в поле $GF(2^m)$

Цель работы: Изучить операцию умножения элементов конечного поле Галуа $GF(2^m)$ с использованием операций умножения и деления многочленов.

Теоретическая часть

В мире информационных технологий конечные поля Галуа $GF(2^m)$ имеют огромное практическое значение. В частности, важнейшие алгоритмы обнаружения и исправления искажения информации в системах хранения и сетях передачи данных, использующие коды Рида-Соломона, а также криптографические алгоритмы (например, AES – Advanced Encryption Standard), защищающие информацию от несанкционированного доступа, базируются на арифметике конечных полей Галуа $GF(2^m)$.

Однако для эффективной программной и аппаратной реализации алгоритмов, также необходима быстродействующая аппаратная реализация арифметики поля Галуа $GF(2^m)$. В частности, для ускорения операций умножения и деления элементов необходимы специальные подходы к разработке быстрой параллельной схемы умножения элементов и нахождения мультипликативной инверсии элемента для операции деления.

В рамках научных исследований в области надежности систем хранения, передачи и обработки данных, а также методов информационного резервирования, автором была исследована эффективная аппаратная схема быстрого умножения, и на базе нее была разработана схема арифметического процессора для поля Галуа $GF(2^m)$.

Арифметика поля Галуа $GF(2^m)$

Поле Галуа $GF(2^m)$, по определению являющееся полем многочленов вида $a(x) = a_{m-1}x^{m-1} + \ldots + a_1 x - a_0$, $a_i \in \{0,1\}$, образуется на базе простого поля Галуа GF(2) и нормированного примитивного неприводимого многочлена m-й степени: $p(x) = x^m + p_{m-1}x^{m-1} + p_1 x + p_0$, $p_i \in \{0,1\}$. Особо отметим, что элементы поля можно также рассматривать как m-разрядные двоичные числа

$(a)_2=(a_{m-1}\dots a_1a_0)_2$, и более того, для компактной формы представления записывать двоичные эквиваленты элементов поля в десятичной форме (a)10.

Например, поле Галуа GF(24) образуется при помощи неприводимого многочлена $p(x)=x^4+x+1$, и его элементы можно рассматривать как многочлены, так и соответствующие двоичные и десятичные эквиваленты:

$GF(2^4)$:	$a(x)$:	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
	$(a)_2$:	0000	0001	0010	0011	0100	0101	0110	0111
	$(a)_{10}$:	0	1	2	3	4	5	6	7

$a(x)$:	x^3	x^3+1	x^3+x	x^3+x+1	x^3+x^2	x^3+x^2+1	x^3+x^2+x	x^3+x^2+x+1
$(a)_2$:	1000	1001	1010	1011	1100	1101	1110	1111
$(a)_{10}$:	8	9	10	11	12	13	14	15

Отметим, что в базовом простом поле GF(2) для элемента 0 обратным элементом по сложению является сам элемент 0, также как и для элемента 1 обратным элементом по сложению является сам элемент 1. Соответственно, как сложение, так и вычитание элементов простого поля GF(2) фактически сводятся к одной и той же операции суммирования по модулю 2, и обозначается символом $\ominus$.

Тогда, при сложении и вычитании элементов поля $GF(2^m)$ мы имеем сложение соответствующих коэффициентов многочленов по модулю 2 (при представлении в виде многочленов) или побитовое сложение по модулю 2 соответствующих разрядов двоичных чисел (при представлении в виде двоичных чисел):

$$\underset{GF(2^m)}{a\pm b}=\underbrace{(a(x)\pm b(x))\bmod p(x)}_{GF(2)}=\underbrace{(a_{m-1}\pm b_{m-1})}_{GF(2)}x^{m-1}+\dots+\underbrace{(a_1\pm b_1)}_{GF(2)}x+\underbrace{(a_0\pm b_0)}_{GF(2)}=$$
$$=(a_{m-1}\oplus b_{m-1})x^{m-1}+\dots+(a_1\oplus b_1)x+(a_0\oplus b_0)=((a_{m-1}\oplus b_{m-1})\dots(a_0\oplus b_0))_2. \qquad (1)$$

Пример. Найдем сумму элементов расширенного поля GF(24), представленных в виде соответствующих чисел «13» и «7» в десятичной системе счисления. Имеем,

$$\underset{GF(2^4)}{(13)_{10}+(7)_{10}}=\underbrace{(1101)_2+(0111)_2}_{GF(2^4)}=\begin{array}{|c|c|c|c|}\hline \oplus & 1 & \ominus & 1 \\ \hline 0 & 1 & 1 & 0 \\ \hline \end{array}\ \begin{array}{|c|c|c|c|}\hline \oplus & 0 & \ominus & 1 \\ \hline 1 & 1 & 0 & 1 \\ \hline \end{array}=(1010)_2=(10)_{10}$$

Быстрое умножение и деление элементов поля $GF(2^m)$

Для построения быстрых и компактных умножителей следует использовать классическое определение операции умножения элементов поля Галуа GF(2m), представленных в виде многочленов с коэффициентами из простого поля GF(2):

$$\forall a,b \in GF(2^m) \Rightarrow \underbrace{a \cdot b}_{GF(2^m)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)}$$

Рассмотрим умножение элементов на примере поля Галуа GF(24), образованного при помощи примитивного неприводимого многочлена $p(x) = x^4 + x + 1$. Имеем следующее:

$$\underbrace{a \cdot b}_{GF(2^4)} = \underbrace{((a_3x^3 + a_2x^2 + a_1x + a_0)(b_3x^3 + b_2x^2 + b_1x + b_0)) \bmod (x^4 + x + 1)}_{GF(2)}$$

После перемножения многочленов и вычисления остатка по модулю $p(x) = x^4 + x + 1$ в общем виде получаем:

$$\underbrace{a \cdot b}_{GF(2^4)} = \underbrace{(a(x) \cdot b(x)) \bmod p(x)}_{GF(2)} = c_3x^3 + c_2x^2 + c_1x + c_0;$$

$$\begin{cases} c_3 = a_c \cdot b_3 \oplus a_1 \cdot b_2 \oplus a_2 \cdot b_1 \oplus a_3 \cdot b_0 \oplus a_3 \cdot b_3; \\ c_2 = a_0 \cdot b_2 \oplus a_1 \cdot b_1 \oplus a_2 \cdot b_0 \oplus a_1 \cdot b_3 \oplus a_2 \cdot b_3 \oplus a_3 \cdot b_2; \\ c_1 = a_0 \cdot b_1 \oplus a_1 \cdot b_0 \oplus a_2 \cdot b_3 \oplus a_1 \cdot b_2 \oplus a_2 \cdot b_2 \oplus a_3 \cdot b_3 \oplus a_3 \cdot b_1; \\ c_0 = a_0 \cdot b_0 \oplus a_1 \cdot b_3 \oplus a_2 \cdot b_2 \oplus a_3 \cdot b_1. \end{cases}$$

Таким образом, мы имеем m аддитивных функций для вычисления коэффициентов $c_{m-1} \ldots c_0$. Функции содержат слагаемые в виде произведений коэффициентов ai·bj, где $i,j = 0 \ldots m-1$. Поскольку мы имеем дело с полями $GF(2^m)$, являющиеся расширением базового простого поля GF(2), то произведение коэффициентов эквивалентно логическому умножению (конъюнкции).

Для аппаратной реализации таких функции удобнее использовать специализированные программируемые логические матрицы (ПЛМ), содержащие в себе логические элементы «И» с двумя входами и многовходовые сумматоры по модулю 2.

Ниже на рис. 1 приведена функциональная схема умножителя элементов поля GF(2^4), образованного на базе примитивного неприводимого многочлена $p(x)=x^4+x+1$.

Входы сумматоров в соответствии с аддитивными функциями подключаются к выходам логических элементов «И», формирующих соответствующие произведения коэффициентов ai·bj. Незадействованные входы сумматоров подключаются к «земле».

Теперь обобщим вышеприведенный пример для общего случая умножения элементов поля Галуа GF(2^m), m ≥ 2, образованного на базе заданного примитивного неприводимого многочлена $p(x)=x^m+p_{m-1}x^{m-1}+\ldots+p_1x+p_0$, в виде итерационной процедуры, в которой за m итераций выводятся m формул для расчета всех коэффициентов многочлена-произведения:

$$(a(x)\cdot b(x))\bmod(p(x))=\sum_{k=0}^{m-1}c_k^{(m)}\cdot x^k;$$

$$s=1\ldots m;\quad m\geq 2;\quad c_0^{(0)}=\ldots=c_{m-1}^{(0)}=0;$$

$$\begin{cases} c_0^{(s)}=c_{m-1}^{(s-1)}\cdot p_0\oplus a_0\cdot b_{m-s};\\ c_1^{(s)}=c_0^{(s-1)}\oplus c_{m-1}^{(s-1)}\cdot p_1\oplus a_1\cdot b_{m-s};\\ \vdots\\ c_{m-1}^{(s)}=c_{m-2}^{(s-1)}\oplus c_{m-1}^{(s-1)}\cdot p_{m-1}\oplus a_{m-1}\cdot b_{m-s}. \end{cases} \quad (2)$$

Следует особо отметить, что итерационный вывод прямых формул осуществляется лишь один раз на этапе проектирования аппаратной реализации, и далее формулы аппаратно реализуются в коммутационной матрице соединений между выходами m2 двухвходовых элементов «И» и входы m элементов сумматоров по модулю 2.

Что касается операции деления элемента a на ненулевой элемент b поля, то ее можно свести к операции умножения на обратный элемент b–1.

$$\forall a,b\in GF(2^m):b\neq 0\Rightarrow \underbrace{a:b}_{GF(2^m)}=\underbrace{a\cdot b^{-1}}_{GF(2^m)}. \quad (3)$$

Вычисление обратного элемента по умножению для максимального быстродействия, очевидно, лучше всего опять же осуществлять табличным способом, используя ПЗУ емкостью 2mn бит. Что касается формирования самой таблицы обратных элементов на этапе проектирования, то ее можно подготовить заранее, используя расширенный алгоритм Евклида для многочленов, который сводит решение уравнения

$$\underbrace{(b(x)\cdot b^{-1}(x)) \bmod p(x) = 1}_{GF(2)},$$

где b–1(x) разыскиваемый обратный многочлен по умножению, к нахождению многочленов g(x) и h(x), а также наибольшего общего делителя $НОД(b(x), p(x))$ для многочленов b(x) и p(x) таких, что

$$\underbrace{b(x)\cdot g(x) + p(x)\cdot h(x) = НОД(b(x), p(x))}_{GF(2)}.$$

Поскольку мы имеем дело с полем, то для любого ненулевого многочлена $b(x)$ алгоритм в качестве $НОД(b(x), p(x))$ дает скаляр $\lambda \in GF(2)$ (многочлен нулевой степени), и в нашем «двоичном» случае скаляр будет равен строго $\lambda = 1$, так как в базовом простом поле GF(2) существует только один ненулевой элемент – это единица. Соответственно, многочлен g(x), находимый алгоритмом, и является искомым обратным многочленом по умножению, то есть $b^{-1}(x) = g(x)$.

Приведем для примера таблицу обратных элементов для элементов поля GF(24), представленных для компактности в виде десятичных и двоичных эквивалентов элементов.

$(b)_{10}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(b)_2$	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
$(b^{-1})_{10}$	1	9	14	13	11	7	6	15	2	12	5	10	4	3	8
$(b^{-1})_2$	0001	1001	1110	1101	1011	0111	0110	1111	0010	1100	0101	1010	0100	0011	1000

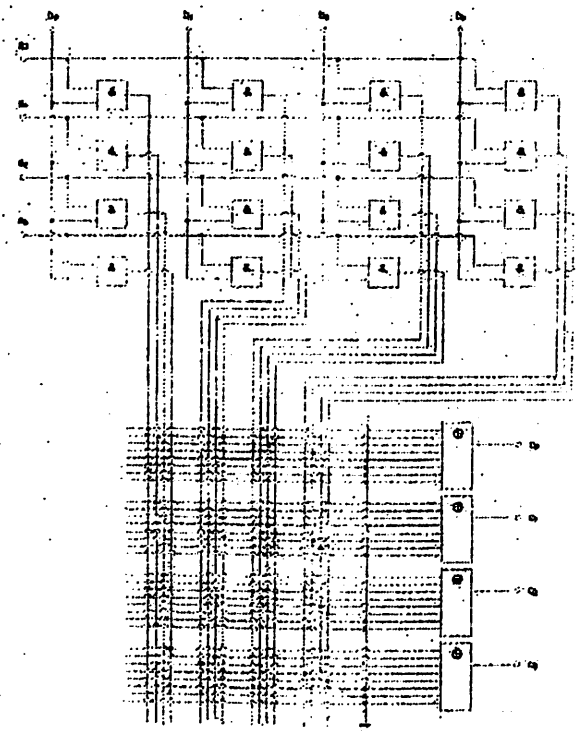


Рис. 3.1. Функциональная схема умножителя элементов поля Галуа GF(24) на базе специализированной логической матрицы

Арифметический процессор для поля Галуа GF(2^m) на базе быстрого умножения и инвертирования элементов. Используя умножитель элементов поля Галуа GF(2^m) на базе специализированной программируемой логической матрицы теперь можно построить арифметический процессор для поля Галуа GF(2^m), сведя операцию деления элемента a на ненулевой элемент b поля к умножению на обратный элемент b^{-1} по умножению. Вычисление обратного элемента по умножению для максимального быстродействия, очевидно, лучше осуществлять табличным способом, используя ПЗУ емкостью 2mm бит.

Ниже на рисунке 2 представлена функциональная схема арифметического процессора для поля Галуа GF(2m), использующего таблицу обратных элементов по умножению, хранящуюся в ПЗУ и умножитель

элементов, который, как было рассмотрено выше, реализуется на базе специализированной программируемой логической матрицы.

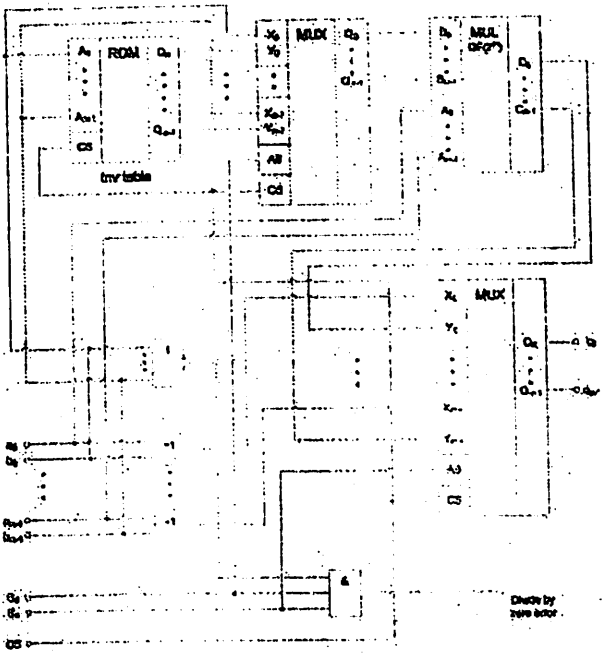


Рис. 3.2. Функциональная схема арифметического процессора для поля Галуа $GF(2^m)$ с использованием умножителя элементов и таблицы обратных элементов по умножению

В схеме процессора используются два m-битных мультиплексора 2 → 1. Нижний мультиплексор коммутирует свои входы с выходами логических элементов XOR при управляющем сигнале S1 = 0 (режим операций сложения / вычитания), и с выходами умножителя при S1 = 1 (режим операций умножения / деления). При S1 = 0, управляющий сигнал S0 не играет никакой роли (сложение и вычитание сводится к одной и той же операции «побитового» XOR). При S1 = 1, сигнал S0 управляет верхним мультиплексором, который при S0 = 0 подключает линии $b_{m-1} \ldots b_0$ напрямую к умножителю элементов, что соответствует операции умножения на операнд b, а при S0 = 1 подключает выходы ПЗУ, преобразующего операнд b в его

обратный элемент по умножению, что соответствует операции деления на операнд b. В схеме процессора также предусмотрена цепь обнаружения нулевого делителя (b = 0) в режиме операции деления (S1 = 1 и S0 = 1).

Арифметический процессор на базе умножителя и таблицы обратных элементов требует одного ПЗУ емкостью 2mm бит, и умножителя, состоящего из m2 двухвходовых логических элементов «И», m многовходовых сумматоров по модулю 2, и коммутационной матрицы размером $(m^2+1)m^2 m \sim m^3$. Если коммутационную матрицу тоже рассматривать как «постоянную память», то, очевидно, что ее размер составляет m5.

Практическая работа №4

Тема: Исследование работы регистров линейной обратной связи

Цель работы: Изучение принципа работы генератора псевдослучайных последовательностей, основанного на регистре сдвига с линейной обратной связью.

Описание лабораторной работы

Общие сведения о регистрах сдвига с линейной обратной связью. последовательности регистров сдвига используются как в криптографии, так и в теории кодирования. их теория прекрасно проработана, потоковые шифры на базе регистров сдвига являлись рабочей лошадкой военной криптографии задолго до появления электроники.

Регистр сдвига с обратной связью (*далее РгСсОС*) *состоит из двух частей: регистра сдвига и функции обратной связи* (рис. 4.1). регистр сдвига 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 одну 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.

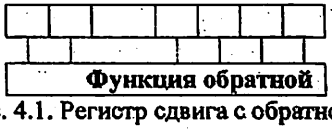
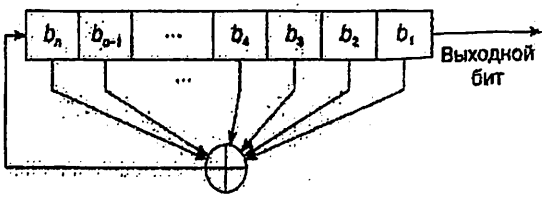


Рис. 4.1. Регистр сдвига с обратной связью

Регистры сдвига очень быстро нашли применение в потоковых шифрах, так как они легко реализовывались с помощью цифровой аппаратуры. В 1965 году Эрнст Селмер (*Ernst Selmer*), главный криптограф норвежского правительства, разработал теорию последовательности регистров сдвига. Соломон Голомб (*Solomon Golomb*), математик NSA, написал книгу, излагающие некоторые свои результаты и результаты Селмера. простейшим

видом регистра сдвига с обратной связью является *регистр сдвига с линейной обратной связью* (*linear feedback shift register*, *далее LFSR или РгССлЛОС*). Обратная связь таких регистров представляет собой просто XOR (сложение по модулю два) некоторых битов регистра, перечень этих битов называется отводной последовательностью (*tap sequence*). иногда такой регистр называется конфигурацией Фиббоначи. из-за простоты последовательности обратной связи для анализа РгССлЛОС можно использовать довольно развитую математическую теорию. проанализировав получаемые выходные последовательности, можно убедиться в том, что эти последовательности достаточно случайны, чтобы быть безопасными. РгССлЛОС чаще других сдвиговых регистров используются в криптографии.



В общем случае «-битовый РгССлЛОС может находиться в одном из $N = 2« — 1$ внутренних состояний. Это означает, что теоретически такой регистр может генерировать псевдослучайную последовательность с периодом $T = 2« — 1$ битов. (Число внутренних состояний и период равны $N = T_{max} = 2« — 1$, потому что заполнение РгССлЛОС нулями приведет к тому, что сдвиговый регистр будет выдавать бесконечную последовательность нулей, что абсолютно бесполезно.) только при определенных отводных последовательностях РгССлЛОС циклически пройдет через все $2« — 1$ внутренних состояний, такие РгССлЛОС являются *РгССлЛОС с максимальным периодом*. получившийся результат называется *М-последовательностью*.

Пример на рисунке 4.2 показан четырехбитовый РгССлЛОС с отводом от первого и четвертого битов. Если его проинициализировать значением 1111, то до повторения регистр будет принимать следующие внутренние состояния (табл. 4.1).

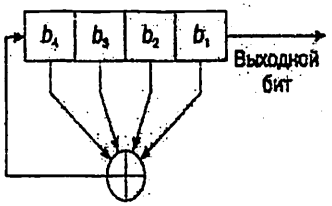


Рис. 4.2. Четырехбитовый РгССлОС с отводом от первого и четвертого битов

Таблица 4.1

номер такта сдвига (внутреннего состояния)	Состояние регистров				Выходной бит
	T1	T2	Тз	T4	
инициальное значение	**1**	**1**	**1**	**1**	—
1	0	1	1	1	1
2	1	0	1	1	1
3	0	1	0	1	1
4	1	0	1	0	0
5	1	1	0	1	1
6	0	1	1	0	0
7	0	0	1	1	1
8	1	0	0	1	1
9	0	1	0	0	0
10	0	0	1	0	0
11	0	0	0	1	1
12	1	0	0	0	0
13	1	1	0	0	0
14	1	1	1	0	0
15 (возврат в инициальное	1	1	1	1	**1**
16 (повтор состояний)	0	1	1	1	**1**

Выходной последовательностью будет строка младших значащих битов: 1 1 1 1 0 1 0 1 1 0 0 1 0 0 0 с периодом $T = 15$, общее число возможных внутренних состояний (кроменулевого) $N = 2^4 - 1 = 16 - 1 = 15 = T_{max}$, следовательно, выходная последовательность — ^-последовательность.

Для того чтобы конкретный РгССлОС имел максимальный период, многочлен, образованный из отводной последовательности и константы 1,

должен быть примитивным по модулю 2. многочлен представляется в виде суммы степеней, например многочлен степени n представляется так:

$$a_n x^n + a_{n\_1} x^{n-1} + \ldots + a_1 x^1 + a_0 x^0 = a_n x^n + a_{n\_1} x^{n-1} + \ldots + ax + a_0,$$

где $a_i = \{0,1\}$ для $i = 1 \ldots n$, а x^i — указывает разряд.

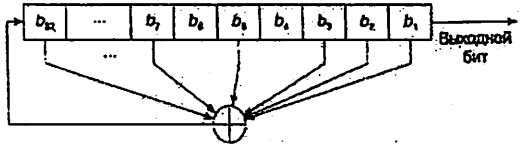
Степень многочлена является длиной сдвигового регистра. примитивный многочлен степени n — это неприводимый многочлен, который является делителем $x^{2n-1} + 1$, но не является делителем $x^d + 1$ для всех d, являющихся делителями $2^n - 1$.

В общем случае не существует простого способа генерировать примитивные многочлены данной степени по модулю 2. проще всего вы- бирать многочлен случайным образом и проверять, не является ли он примитивным. Это нелегко и чем-то похоже на проверку, не является ли простым случайно выбранное число — но многие математические пакеты программ умеют решать такую задачу.

некоторые, но, конечно же не все, многочлены различных степеней, примитивные по модулю 2, приведены далее. например, запись (32, 7, 5, 3, 2, 1, 0) означает, что следующий многочлен примитивен по модулю 2: $x^{32} + x^7 + x^5 + x^3 + x^2 + x + 1$.

Это можно легко обобщить для РгССЛОС с максимальным периодом. первым числом является длина РгССЛОС. последнее число всегда равно 0, и его можно опустить. Все числа, за исключением 0, задают отводную последовательность, отсчитываемую от левого края сдвигового регистра. Другими словами, члены многочлена с меньшей степенью соответствуют позициям ближе к правому краю регистра.

продолжая пример, запись (32, 7, 5, 3, 2, 1, 0) означает, что для взятого 32-битового сдвигового регистра новый бит генерируется с помощью XOR тридцать второго, седьмого, пятого, третьего, второго и первого битов, получающийся РгССЛОС будет иметь максимальную длину, циклически проходя до повторения через $2^{32} - 1$ значений.



Рассмотрим программный код РгССлОС, у которого отводная последовательность характеризуется многочленом (32, 7, 5, 3, 2, 1, 0). На языке *С* это выглядит следующим образом:

```
int LFSR ()
{
static unsigned long ShiftRegister = 1;
/* Все, кроме 0. */
ShiftRegister = ((((ShiftRegister >> 31)
^(ShiftRegister >> 6)
^(ShiftRegister >> 4)
^(ShiftRegister >> 2)
^(ShiftRegister >> 1)
^ShiftRegister)) &0x00000001) <<31)
|(ShiftRegister >> 1);
return ShiftRegister & 0x00000001;
}
```

Если сдвиговый регистр длиннее компьютерного слова, код усложняется, но не намного. В приложении В приведена таблица некоторых примитивных многочленов по модулю 2, будем использовать ее в дальнейшем для выявления некоторых свойств этих многочленов, а также в программной реализации для задания отводной последовательности.

Следует обратить внимание, что у всех элементов таблицы нечетное число коэффициентов. такая длинная таблица приведена для дальнейшей работы с РгССлОС, так как РгССлОС часто используются для работы с потоковыми шифрами и в генераторах псевдослучайных чисел. В нашем случае можно использовать многочлены со старшей степенью не более семи.

Если $p(x)$ примитивен, то примитивен и $x^n p(1 / x)$, поэтому каждый элемент таблицы на самом деле определяет два примитивных многочлена. например, если $(a, b, 0)$ примитивен, то примитивен и $(a, a-b, 0)$. Если примитивен $(a, b, c, d, 0)$, то примитивен и $(a, a-d, a-c, a-b, 0)$. математически:

если примитивен $x^a + x^b + 1$, *то примитивен и* $x^a + x^{a-b} + 1$,

если примитивен $x^a + x^b + x^c + x^d + 1$,

то примитивен и $x^a + x^{a-d} + x^{a-c} + x^{a-b} + 1$.

Наиболее просто программно реализуются примитивные трехчлены, так как для генерации нового бита нужно выполнять XOR только двух битов сдвигового регистра (нулевой член не учитывается, т.е. $x^0 = 1$, см. пример выше). Действительно, все многочлены обратной связи, приведенные в таблице, являются разреженными, т.е., у них немного коэффициентов. разреженность всегда представляет собой источник слабости, которой иногда достаточно для вскрытия алгоритма. Для криптографических алгоритмов гораздо лучше использовать плотные примитивные многочлены, у которых много коэффициентов. применяя плотные многочлены, особенно в качестве части ключа, можно использовать значительно более короткие РгССЛОС.

Генерировать плотные примитивные многочлены по модулю 2 нелегко. В общем случае для генерации примитивных многочленов степени k нужно знать разложение на множители числа $2^k - 1$.

Сами по себе РгССЛОС являются хорошими генераторами псевдослучайных последовательностей, но они обладают некоторыми нежелательными неслучайными (детерминированными) свойствами. последовательные биты линейны, что делает их бесполезными для шифрования. Для РгССЛОС длины n внутреннее состояние представляет собой предыдущие n выходных битов генератора. Даже если схема обратной связи хранится в секрете, она может быть определена по $2n$ выходным битам генератора с помощью высокоэффективного алгоритма *Berlekamp — Massey*.

Кроме того, большие случайные числа, генерируемые с использованием идущих подряд битов этой последовательности, сильно коррелированны и для

некоторых типов приложений вовсе не являются случайными. несмотря на это, РгССлЛОС часто используются в качестве составных частей систем и алгоритмов шифрования.

О потоковых шифрах на базе РгССлЛОС. основной подход при проектировании генератора потока ключей на базе РгССлЛОС прост. Сначала берется один или несколько РгССлЛОС обычно с различными длинами и различными многочленами обратной связи. Если длины взаимно просты, а все многочлены обратной связи примитивны, то у образованного генератора будет максимальная длина. Ключ является начальным состоянием регистров РгССлЛОС. Каждый раз для получения нового бита, достаточно сдвинуть на бит регистры РгССлЛОС (это иногда называют тактированием (*clocking*)). Бит выхода представляет собой функцию, желательно нелинейную, некоторых битов регистров РгССлЛОС. Эта функция называется комбинирующей функцией, а генератор в целом — комбинационным генератором. Если бит выхода является функцией единственного РгССлЛОС, то генератор называется фильтрующим генератором. Большая часть теории подобного рода устройств разработана Селмером (*Selmer*) и нилом Цирлером (*Neal Zierler*). можно ввести ряд усложнений. В некоторых генераторах для различных РгССлЛОС используется различная тактовая частота, иногда частота одного генератора зависит от выхода другого. Все это электронные версии идей шифровальных машин, появившихся до Второй мировой войны, которые называются генераторами с управлением тактовой частотой (*clock-controlled generators*). управление тактовой частотой может быть с прямой связью, когда выход одного РгССлЛОС управляет тактовой частотой другого РгССлЛОС, или с обратной связью, когда выход одного РгССлЛОС управляет его собственной тактовой частотой. Хотя все эти генераторы чувствительны, по крайней мере теоретически, к вскрытиям вложением и вероятной корреляцией, многие из них безопасны до сих пор.

Ян Касселлс (*Ian Cassells*), ранее возглавлявший кафедру чистой математики в Кембридже и работавший криптоаналитиком в Блетчли парк

(*Bletchly Park*), сказал, что «криптография — это смесь математики и путаницы, и без путаницы математика может быть использована против вас». Он имел в виду, что в потоковых шифрах для обеспечения максимальной длины и других свойств необходимы определенные математические структуры, такие как РгСсЛОС, но, чтобы помешать кому-либо получить содержание регистра и вскрыть алгоритм, необходимо внести некоторый сложный нелинейный беспорядок. Этот совет справедлив и для блочных алгоритмов.

Большинство реальных потоковых шифров основаны на РгСсЛОС. Даже в первые дни электроники построить их было несложно. Сдвиговый регистр не представляет собой ничего большего, чем массив битов, а последовательность обратной связи — набор вентилей XOR. Даже при использовании современных интегральных схем потоковый шифр на базе РгСсЛОС обеспечивает немалую безопасность с помощью нескольких логических вентилей. проблема РгСсЛОС состоит в том, что их программная реализация очень неэффективна. Вам приходится избегать разреженных многочленов обратной связи — они облегчают корреляционные вскрытия — а плотные многочлены обратной связи неэффективны.

Эта отрасль криптографии быстро развивается и находится под зорким государственным контролем со стороны NSA. Большинство разработок засекречены — множество используемых сегодня военных систем шифрования основаны на РгСсЛОС. Действительно, у большинства компьютеров *Cray* (*Cray* 1, *Cray X-MP*, *Cray Y-MP*) есть весьма любопытная инструкция, обычно называемая как «счетчик совокупности» (*population count*). Она подсчитывает количество единиц в регистре и может быть использована как для эффективного вычисления расстояния Хэмминга между двумя двоичными словами так и для реализации векторизированной версии РгСсЛОС. Эта инструкция считается канонической инструкцией NSA, обязательно фигурирующей почти во всех контрактах, касающихся компьютеров.

С другой стороны, было взломано удивительно большое число казавшихся сложными генераторов на базе сдвиговых регистров.

О линейной сложности генерируемой последовательности псевдослучайных чисел РгСсЛОС. Анализировать потоковые шифры часто проще, чем блочные. например, важным параметром, используемым для анализа генераторов на базе РгСсЛОС, является линейная сложность (*li«ear complexity*), или линейный интервал. Она определяется как длина « самого короткого РгСсЛОС, который может имитировать выход генератора. Любая последовательность, сгенерированная конечным автоматом над конечным полем, имеет конечную линейную сложность. Линейная сложность важна, потому что с помощью простого алгоритма, называемого алгоритмом *Berlekamp-Massey*, можно определить этот РгСсЛОС, проверив только $2n$ битов потока ключей. Воссоздавая нужный РгСсЛОС, вы взламываете потоковый шифр.

Эта идею можно расширить с полей на кольца и на случаи, когда выходная последовательность рассматривается как числа в поле нечетной характеристики. Дальнейшее расширение приводит к вводу понятия профиля линейной сложности, который определяет линейную сложность последовательности по мере ее удлинения. Существуют также понятия сферической и квадратичной сложности. В любом случае следует помнить, что высокая линейная сложность не обязательно гарантирует безопасность генератора, но низкая линейная сложность указывает на недостаточную безопасность генератора.

О корреляционной независимости генерируемой последовательности псевдослучайных чисел РгСсЛОС. Криптографы пытаются получить высокую линейную сложность, нелинейно объединяя результаты некоторых выходных последовательностей. при этом опасность состоит в том, что одна или несколько внутренних выходных последовательностей — часто просто выходы отдельных РгСсЛОС — могут быть связаны общим ключевым потоком и вскрыты при помощи линейной алгебры. Часто такое вскрытие

называют корреляционным вскрытием, или вскрытием разделяй-и-властвуй. томас Сигенталер (*Thomas Siegenthaler*) показал, что можно точно определить корреляционную независимость и что существует компромисс между корреляционной независимостью и линейной сложностью.

Основной идеей корреляционного вскрытия является обнаружение некоторой корреляции между выходом генератора и выходом одной из его составных частей. тогда, наблюдая выходную последовательность, можно получить информацию об этом промежуточном выходе. используя эту информацию и другие корреляции, можно собирать данные о других промежуточных выходах до тех пор, пока генератор не будет взломан.

Против многих генераторов потоков ключей на базе РгСсЛОС успешно использовались корреляционные вскрытия и их вариации, такие как быстрые корреляционные вскрытия, предлагающие компромисс между вычислительной сложностью и эффективностью.

О других способах вскрытия генерируемой последовательности псевдослучайных чисел РгСсЛОС. Существуют и другие способы вскрытия генераторов потоков ключей. тест на линейную корректность (*linear consistency*) — это попытка найти некоторое подмножество ключа шифрования с помощью матричной техники. Существует и вскрытие корректности «встречей посередине» (*meet-in-the-middle consistency attack*). Алгоритм линейного синдрома (*linear syndrome algorithm*) основан на возможности записать фрагмент выходной последовательности в виде линейного уравнения. Существует вскрытие лучшим аффинным приближением (*best affine approximation attack*) и вскрытие выведенным предложением (*derived sequence attack*). К потоковым шифрам можно применить также методы дифференциального и линейного криптоанализа.

На рисунке 4.4 приведена обобщенная схема алгоритма РгСсЛОС, рассматриваемого в лабораторной работе.

Запуск

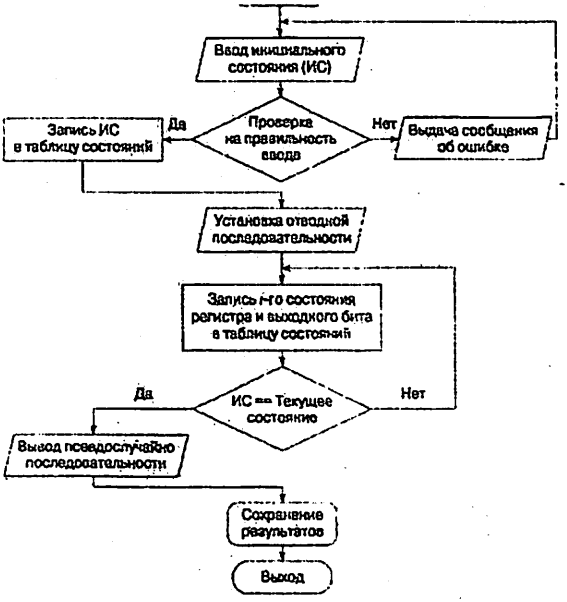


Рис. 4.4. Обобщенная схема алгоритма РгСсЛОС

Описание программного интерфейса. Ниже на рис. 4.5 представлено главное окно программы.

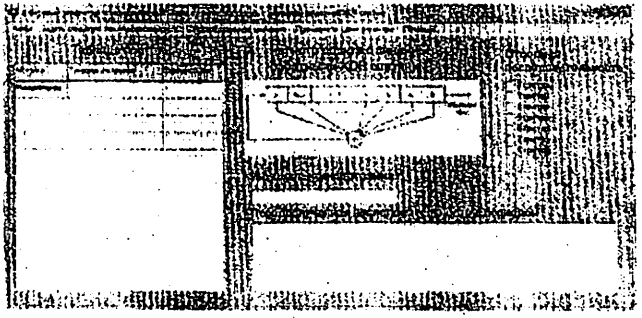


Рис. 4.5. Главное окно программы

В меню есть следующие функции:

- *Файл->Сохранить отчет*

Эта функция осуществляет создание файла отчета, куда записывается инициальное состояние РгСсЛОС, отводная последовательность, период полученной псевдослучайной последовательности бит, сама последовательность и таблица состояний. Если файл успешно был сохранен, то выдается сообщение об успешном сохранении (рис. 4.6). рекомендуемое расширение файла отчета *.txt.

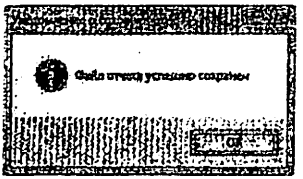


Рис. 4.6. Уведомление о сохранении файла

- *Файл->Выход*

Эта функция обеспечивает закрытие приложения.

- *Задать отводную последовательность*

Эта функция формирует отводную последовательность, считывая значения из клеток, которые пользователь отметил галочкой на экранной форме. после чего она уведомляет пользователя о создании отводной последовательности (рис. 4.7). отводная последовательность определяет, от каких триггеров регистра сдвига будут идти обратные связи XOR для формирования бита смещения. по умолчанию обратная связь от первого триггера стоит всегда, при отсутствии других связей будет осуществляться сдвиг влево с перестановкой младшего бита на позицию старшего.

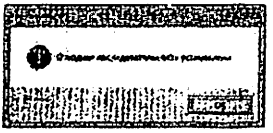


Рис. 4.7. уведомление об установке отводной последовательности

- *Установить инициальное значение*

Эта функция считывает введенное пользователем инициальное значение регистра из окна Edit и осуществляет проверку введенного значения согласно заданным условиям: введенная строка непустая (рис. 4.8), введенная строка

имеет длину равную восьми (8 бит = 1 байт, рис. 4.9), введенная строка содержит только нули и /или единицы (рис. 4.10), введенная строка ненулевая (рис. 4.11). В противном случае выдаются сообщения об ошибке, пользователь должен их исправить и снова нажать на кнопку. В случае успешной проверки инициальное значение будет записано в таблицу состояний в строке «инициальное» и выдано уведомление (рис. 4.12).

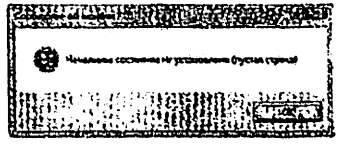


Рис. 4.8. Сообщение о том, что начальное состояние не установлено

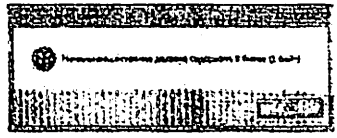


Рис. 4.9. Сообщение о том, что длина начального состояния недопустима

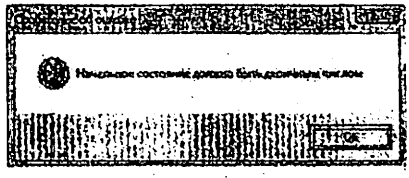


Рис. 4.10. Сообщение о том, что система счисления для указания начального состояния выбрана неверно

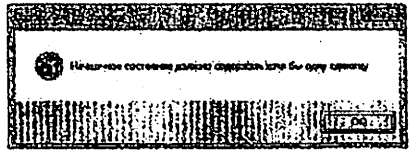


Рис. 4.11. Сообщение о том, что строка начального состояния не содержит единиц

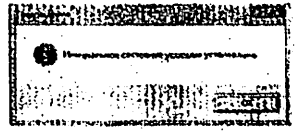


Рис. 4.12. Сообщение об успешном установлении инициального состояния

- *Произвести сдвиг регистра*

Эта функция эмулирует работу регистра сдвига. последовательно производя 256 сдвигов, каждый сдвиг формирует выходной бит псевдослучайной последовательности и новое состояние регистра. Как только находится состояние регистра равное инициальному, вычисляется период и выводит в поле Memo полученную псевдослучайную последовательность.

Помощь->Об авторе

Эта функция выводит на экран информацию об авторе программы (рис. 4.13).

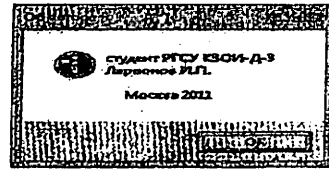


Рис. 4.13. Информация об авторе

- *Помощь->О программе*

Эта функция выводит на экран краткое описание программы и инструкцию (рис. 4.14).

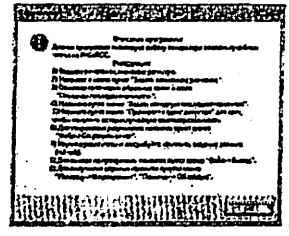


Рис. 4.14. О программе

Задание

Изучить регистр сдвига с линейной обратной связью и отвечать вопросы на контрольных вопросов.

Контрольные вопросы

1. Что такое регистр сдвига с линейной обратной связью?
2. Где используется регистр сдвига с линейной обратной связью?

Практическая работа №5

Тема: Изучение статистических свойств генератора псевдослучайных последовательностей.

Цель работы: Изучение методов построения и тестирования генераторов случайных, псевдослучайных последовательностей

Основная часть

Псевдослучайная последовательность (ПСП) — последовательность чисел, которая была вычислена по некоторому определённому арифметическому правилу, но имеет все свойства случайной последовательности чисел в рамках решаемой задачи.

Хотя псевдослучайная последовательность в этом смысле часто, как может показаться, лишена закономерностей, однако любой псевдослучайный генератор с конечным числом внутренних состояний повторится после очень длинной последовательности чисел. Это может быть доказано с помощью принципа Дирихле.

Генератор псевдослучайных чисел (ГПСЧ, англ. *pseudorandom number generator*, *PRNG*) — алгоритм, порождающий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению (обычно дискретному равномерному).

Современная информатика широко использует псевдослучайные числа в самых разных приложениях — от метода Монте-Карло и имитационного моделирования до криптографии. При этом от качества используемых ГПСЧ напрямую зависит качество получаемых результатов. Это обстоятельство подчёркивает известный афоризм математика ORNL Роберта Кавью (англ.)рус.: «*генерация случайных чисел слишком важна, чтобы оставлять её на волю случая*».

Источники настоящих случайных чисел найти крайне трудно. Физические шумы, такие, как детекторы событий ионизирующей радиации, дробовой шум в резисторе или космическое излучение, могут быть такими источниками. Однако применяются такие

устройства в приложениях сетевой безопасности редко. Сложности также вызывают грубые атаки на подобные устройства.

У физических источников случайных чисел существует ряд недостатков:

- Время и трудозатраты при установке и настройке по сравнению с программными ГПСЧ;
- Дороговизна;
- Генерация случайных чисел происходит медленнее, чем при программной реализации ГПСЧ;
- Невозможность воспроизведения ранее сгенерированной последовательности случайных чисел.

В то же время случайные числа, получаемые из физического источника, могут использоваться в качестве порождающего элемента (англ. seed) для программных ГПСЧ. Такие комбинированные генераторы применяются в криптографии, лотереях, игровых автоматах.

Список генераторов псевдослучайных чисел

Xorshift	G. Marsaglia	2003	Это очень быстрый подтип генераторов LFSR. Марсалья также предложил в качестве улучшения генератор xorwow, в котором выход генератора xorshift суммируется с последовательностью Вейля. Генератор xorwow является генератором по умолчанию в библиотеке CURAND интерфейса прикладного программирования nVidia CUDA для графических процессоров.
Алгоритм Fortuna	Шнайер, Брюс; Нильс Фергюсон	2003	Алгоритм считается криптографически безопасным. CSPRNG, хорошо известный тем, что был внедрен в системы и продукты Apple.

Well equidistributed long-period linear (WELL)	F. Panneton; P. L'Ecuyer; M. Matsumoto	2006	Алгоритм, известный как дополнение к Mersenne Twister (MT), намеренно стремящийся скрыть его слабые стороны.
Усовершенствованная система рандомизации (ARS)	J. Salmon; M. Moraes; R. Dror; D. Shaw	2011	Упрощенная версия блочного шифра AES, обеспечивающая очень высокую производительность на системе, поддерживающей AES-NI.
Threefry	J. Salmon, M. Moraes, R. Dror and D. Shaw	2011	Упрощенная версия блочного шифра Threefish, подходящая для реализации на GPU.
Philox (Филокс)	J. Salmon, M. Moraes, R. Dror and D. Shaw	2011	Упрощение и модификация блочного шифра Threefish с добавлением S-box.
Пермутированный конгруэнциальный генератор (PCG)	M. E. O'Neill	2014	Модель, полученная с помощью линейного конгруэнтного метода.
Генератор битов случайного цикла (RCB)	R. Cookman	2016	RCB описывается как генератор битовых шаблонов, созданный для преодоления некоторых недостатков Вихрь Мерсенна (MT) и ограничения короткого периода/длины бита генераторов сдвигов/модулей.
Middle Square Weyl Sequence RNG	B. Widynski	2017	Разновидность оригинального метода средних квадратов Джона фон Неймана.
Xoroshiro128+	D. Blackman; S. Vigna	2018	Модификация генератора Xorshift Г. Марсальи, одного из самых быстрых генераторов на современных 64-битных процессорах. Родственными генераторами являются xoroshiro128**, xoshiro256+ и xoshiro256***.
64-bit MELG (MELG-64)	S. Harase; T. Kimoto	2018	Реализация 64-битных линейных генераторов F2 с первичным периодом Мерсенна.

Squares RNG	B. Widynski	2020	Основанная на счетчике версия Middle Square Weyl Sequence RNG. По конструкции похож на Philox, но работает значительно быстрее.
Itamaracá (Ita)	D. H. Pereira	2021	Известен как первый алгоритм PRNG, основанный на функции абсолютного значения. Itamaracá также является простой и быстрой моделью, которая генерирует апериодические последовательности случайных чисел.

Таблица значений $\chi 2$ и p-значений

Для любого числа *p* между 0 и 1 определено *p*-значение — вероятность получить для данной вероятностной модели распределения значений случайной величины такое же или более экстремальное значение статистики (среднего арифметического, медианы и др.), по сравнению с наблюдаемым, при условии верности нулевой гипотезы. В данном случае это распределение . Так как значение функции распределения в точке для соответствующих степеней свободы дает вероятность получить значение статистики *менее экстремальное*, чем эта точка, *p*-значение можно получить, если отнять от единицы значение функции распределения. Малое *p*-значение — ниже выбранного уровня значимости — означает статистическую значимость. Этого будет достаточно, чтобы отвергнуть нулевую гипотезу. Чтобы различать значимые и незначимые результаты, обычно используют уровень 0,05.

В таблице даны *p*-значения для соответствующих значений у первых десяти степеней свободы.

Степени свободы (df)	Значение $\chi 2$									
1	0,004	0,02	0,06	0,15	0,46	1,07	1,64	2,71	3,84	6,63	10,83

2	0,10	0,21	0,45	0,71	1,39	2,41	3,22	4,61	5,99	9,21	13,82
3	0,35	0,58	1,01	1,42	2,37	3,66	4,64	6,25	7,81	11,34	16,27
4	0,71	1,06	1,65	2,20	3,36	4,88	5,99	7,78	9,49	13,28	18,47
5	1,14	1,61	2,34	3,00	4,35	6,06	7,29	9,24	11,07	15,09	20,52
6	1,63	2,20	3,07	3,83	5,35	7,23	8,56	10,64	12,59	16,81	22,46
7	2,17	2,83	3,82	4,67	6,35	8,38	9,80	12,02	14,07	18,48	24,32
8	2,73	3,49	4,59	5,53	7,34	9,52	11,03	13,36	15,51	20,09	26,12
9	3,32	4,17	5,38	6,39	8,34	10,66	12,24	14,68	16,92	21,67	27,88
10	3,94	4,87	6,18	7,27	9,34	11,78	13,44	15,99	18,31	23,21	29,59
p-значение	0,95	0,90	0,80	0,70	0,50	0,30	0,20	0,10	0,05	0,01	0,001

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";
}
}
```

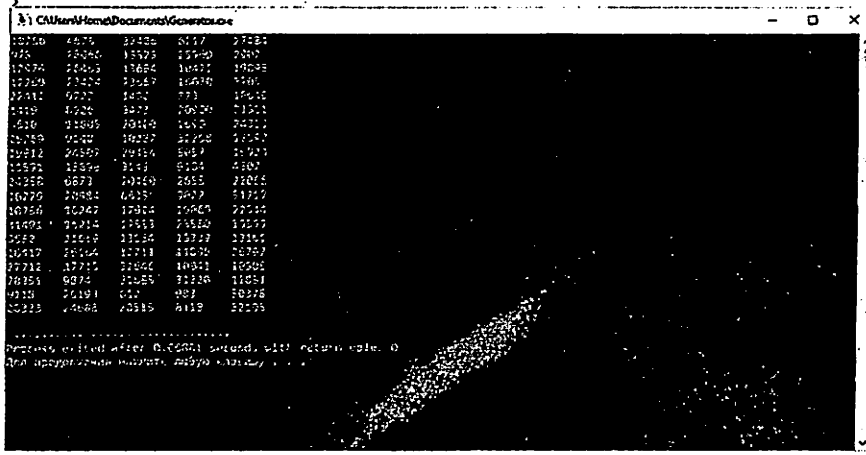


Рис. 5.1. Окно генерации последовательностей

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.

Практическая работа № 6

Тема: Реализация потокового шифра на основе обратного регистра

Цель работы: Формирование знаний о методах построения и работы потоковых шифров. Изучить принцип работы алгоритма A5/1.

Основная часть

Алгоритм A5/1

Шифр A5/1 - эго поточный шифр, используемый для шифрования связи GSM (Group Special Mobile - мобильная групповая специальная связь). Это европейский стандарт для мобильных цифровых сотовых телефонов. Он используется для шифрования канала «телефон/базовая станция».

С шифром A5/1 не все так просто, как с описанием остальных стандартов. Долгое время этот шифр был засекречен и общедоступным стал после утечки информации. Однако до сих пор отсутствует официальное описание схемы работы шифра A5/1. Изучая различные источники, можно запутаться. Потому как разные ученые по-разному изображают и сам шифр, и полиномы, лежащие в его основе, так же по-разному отображают используемые точки синхронизации. Проведя широкий анализ различных публикаций, мы убедились в том, что в основе всех описаний лежит одна и та же схема работы, однако описывают сс по-разному. Попробуем прояснить картину и разберем, как же работает алгоритм шифрования A5/1.

Генератор A5/1 состоит из трех РСЛОС длиной 19, 22 и 23, все многочлены обратной связи у него являются неприводимыми по модулю 2 (см. подразд. 6.1). Выходом является результат операций XOR над тремя РСЛОС. В A5/1 используется изменяемое управление тактированием. Каждый регистр тактируется в зависимости от своего среднего бита, затем над регистром выполняется операция XOR с обратной пороговой функцией средних битов всех грех регистров.

Каждый кадр шифруется с помощью секретного ключа шифрования *Кс* и сквозного порядкового номера очередного кадра. Генератор ПСП A5/1 состоит из трех коротких РСОЛС, обозначаемых как РСОЛС 1,

РСОЛС 2 и РСОЛС 3. Выходные биты снимаются с самых старших разрядов регистров, после чего с помощью операции XOR над битами с выходов всех трех регистров формируется выходной бит у шифра. Общий вид работы алгоритма шифрования A5/1 представлен на рис. 91.

Итак, согласно Б. Шнайеру, РСЛОС обычно производят сдвиг в сторону младших разрядов. Однако в случае с шифром A5/1 дело обстоит иначе, сдвиг происходит в сторону старших разрядов. На рис. 91 изображены три РСЛОС. Слева в них находятся старшие значащие биты, справа - младшие. Для того чтобы понять соответствие различных описаний шифра A5/1, мы представили на рис. 6.1 нумерацию битов как слева направо, начиная от нуля, так и справа налево, также от нуля.

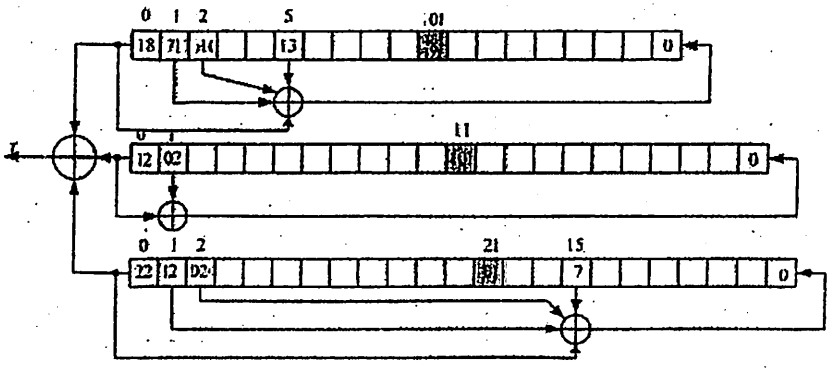


Рис. 6.1. Алгоритм шифрования A5/1

Из рис. 6.1 видно, что многочлены, лежащие в основе шифра A5/1, можно описать двумя разными способами. Если следовать логике, предложенной Б. Шнайером, то, используя нумерацию слева направо, начиная от нуля, получится следующая картина:

РСОЛС 1: $x^{19} + x^5 + x^2 + x + 1$;
РСОЛС 2: $x^{22} + x + 1$;
РСОЛС 3: $x^{23} + x^{15} + x^2 + x + 1$.

При этом первые два многочлена можно найти в таблице неприводимых полиномов в книге Б. Шнайера. Третий регистр также является неприводимым

по модулю 2. Согласно, полином, который в данном поле является образующим, называется примитивным, или базовым, если все его коэффициенты являются взаимно простыми. Для многочлена, описывающего РСЛОС3, это условие выполняется.

Регистры шифра A5/1 работают по принципу stop-and-go, что обеспечивается с помощью применения специальной функции majority, на вход которой подаются значения битов регистров: бит C1 для РСОЛС 1, бит C2 для РСОЛС 2 и C3 для РСОЛС 3. В некоторых источниках указывается следующее соответствие битов: C1 = 8, C2 = 10, C3 = 10. Это соответствует нумерации справа налево, начиная от нуля, при этом порядковые номера битов будут соответственно 9, 11 и 11. В других же источниках можно встретить указание на C1 = 11, C2 = 12, C3 = 13, что соответствует тем же самым битам, что и в первом случае, но при нумерации слева направо и начиная не от нуля, а от единицы.

Функция majority имеет следующий вид:

$$\text{Majority}(x1,x2,x3) = x1x2+x1x3+x2x3.$$

Результатом работы этой функции является один бит, который определяет, какие регистры будут тактироваться (сдвигаться), а какие нет. То есть, если бит с выхода функции совпадает со значением бита регистра на определенной позиции, то регистр сдвигается, иначе нет. Фактически эта функция является функцией большинства, она принимает то значение, которое преобладает на ее входах. В любом случае в режиме работы stop-and-go минимум два регистра будут работать на сдвиг (те, для которых определяющий бит оказался в большинстве). А в случае, когда определяющий бит совпадает у всех регистров, тактироваться будут все три регистра. Работа регистров по правилу stop-and-go обеспечивает нелинейность работы алгоритма в целом, схема работы этого режима схематично показана на рис. 6.2. Функцию определения, какой из регистров необходимо тактировать, а какой нет, выполняет блок управления синхронизацией регистров (Clock Control).

На каждом шаге работы шифра два или три регистра сдвигаются. Таким образом, каждый регистр сдвигается в одном такте работы алгоритма с вероятностью % и нс сдвигается с вероятностью *'А.*

Процесс формирования гаммы, необходимой для шифрования одного кадра, состоит из следующих шагов.

1. Все три регистра сбрасываются в ноль, а затем тактируются 64 раза без учета режима stop-and-go (все три регистра тактируются). Во время этого этапа каждый бит ключа *Кс* последовательно записывается в самый младший бит каждого регистра после операции XOR с сигналом обратной связи, как показано на рис. 6.3.

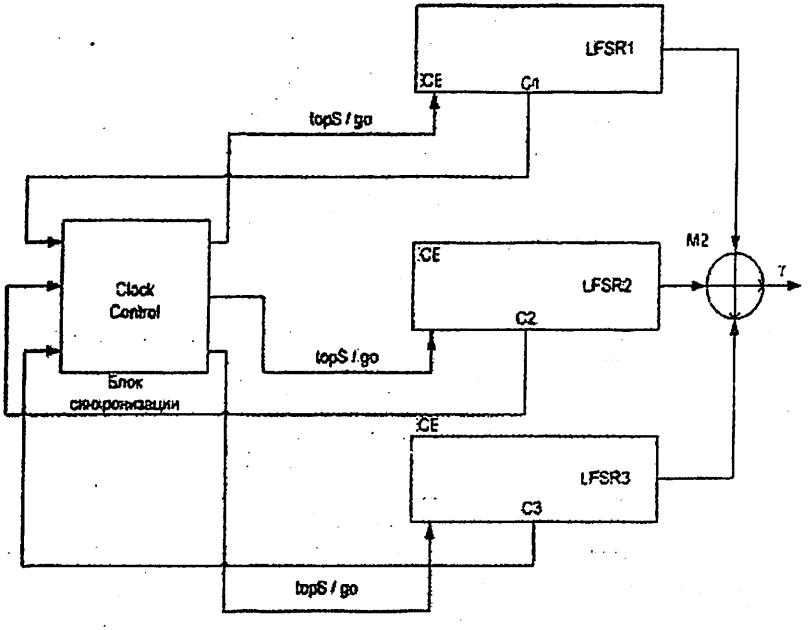


Рис. 6.2. Схема работы регистров по правилу stop-and-go в алгоритме A5/1

2. Все три регистра тактируются 22 раза без учета режима stop-and-go. Во время этого этапа биты номера кадра (так же как и биты ключа шифрования Кс в предыдущем пункте) с помощью операции XOR с сигналом обратной связи последовательно записываются в самый младший разряд каждого регистра.

3. Осуществляется 100 тактов работы алгоритма с использованием режима stop-and-go, что обеспечивает перемешивание ключевой информации.

4. Осуществляется 228 тактов работы алгоритма с использованием режима stop-and-go для формирования гаммы. Затем осуществляется шифрование, когда с помощью операции XOR сформированная последовательность накладывается на информацию, содержащуюся в кадре.

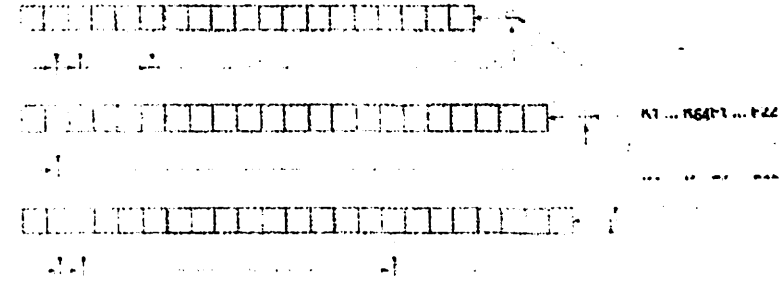


Рис. 6.3. Инициализация ключа K и номера кажра F

Структура алгоритма A5 выглядит следующим образом:

- Три регистра(X, Y, Z) имеют длины 19, 23 и 22 бита,
- Многочлены обратных связей:
  - $X^{19} + X^{18} + X^{17} + X^{14}$ для X,
  - $Y^{23} + Y^{22}$ для Y и
  - $Z^{22} + Z^{21} + Z^{20}$ для Z,
- Управление тактированием осуществляется специальным механизмом:
  - в каждом регистре есть биты синхронизации: 8 (X), 10 (Y), 10 (Z),
  - вычисляется функция F = Maj(X_8; Y_{10}; Z_{10}) = x&y|x&z|y&z, где & — булево AND, | - булево OR, а x, y и z — биты синхронизации X, Y и Z соответственно,
  - сдвигаются только те регистры, у которых бит синхронизации равен F,
  - фактически, сдвигаются регистры, синхробит которых принадлежит большинству.
- Выходной бит системы — результат операции XOR над выходными битами регистров K=X_{19}; Y_{23}; Z_{22}.

- C=M XOR K

Расшифровка тоже так же. Дешифрование на месте M ставят C.

Задание

Зашифровать случайный 20-битный текст. Преобразуйте свое имя и фамилию в двоичное, и его первые 64 бита будут использоваться в качестве регистр.

Контрольные вопросы

1. Общая классификация симметричных поточных алгоритмов?
2. Генераторы с нелинейными преобразованиями?
3. Генераторы с различным тактированием регистров сдвига?

Практическая работа № 7

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

Цель работы: Формирование зананий шифрования данных на основе современных алгоритмов блочного шифрования. Шифрование файлов с использованием пакета программ OpenSSL.

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 7.1:

Таблица 7. 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.

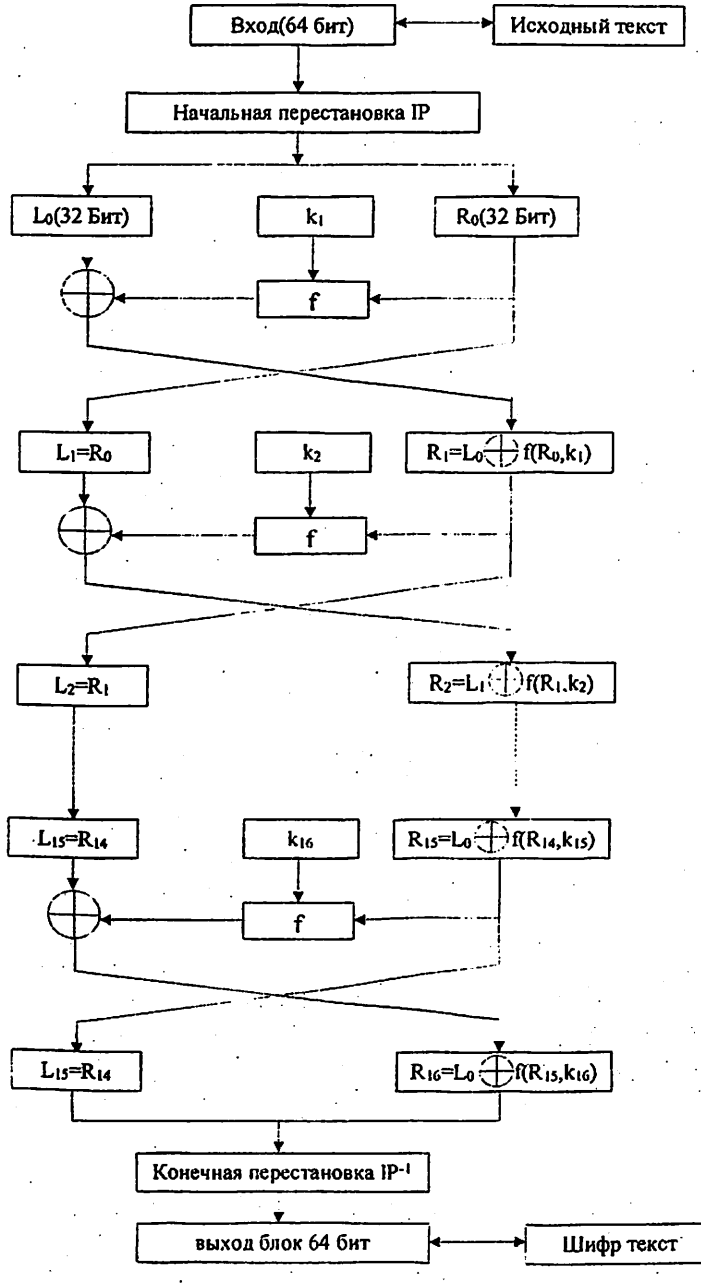


Рис. 7.1. Подробная схема шифрования алгоритма DES

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 7.2.

Таблица 7.2. 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 7.3.

Таблица 7.3 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 7.4.

Таблица 7.4.

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': 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 7.6

Таблица 7.6. 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 7.6.

Таблица 7.6. 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 7.7. Первый и второй биты k_i есть биты 14, 17 вектора C_iD_i.

Таблица 7.7. 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 7.8.

Таблица 7.8. 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 7.9. 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.

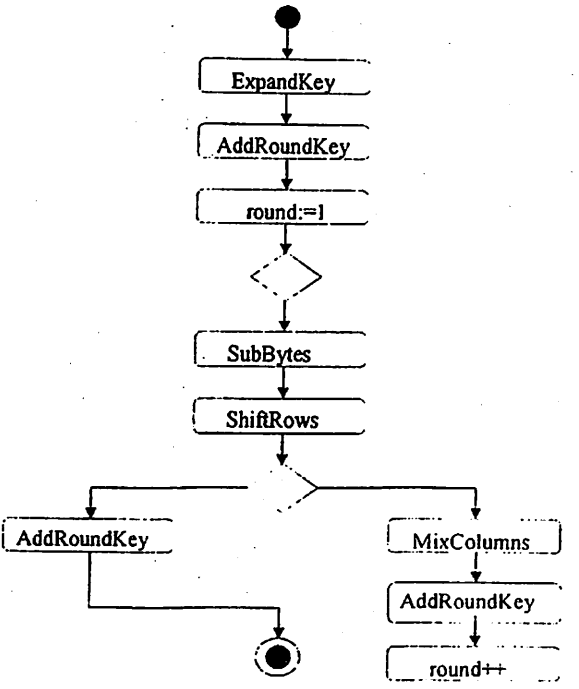


Рис. 7.3. Раунд алгоритма AES

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.

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

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.

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

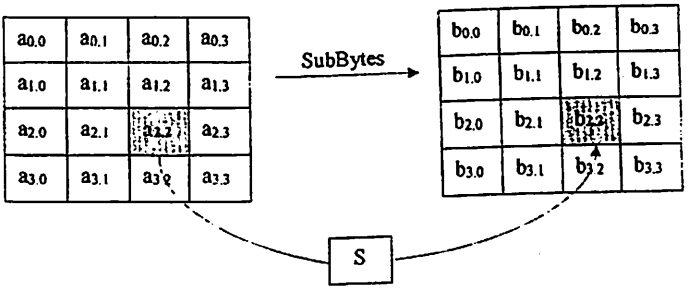


Рис. 7.4. Процедура SubBytes()

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.

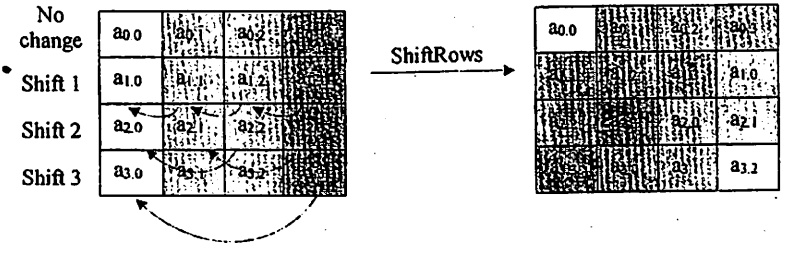


Рис.7.5. Процедура MixColumns

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 с – 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.

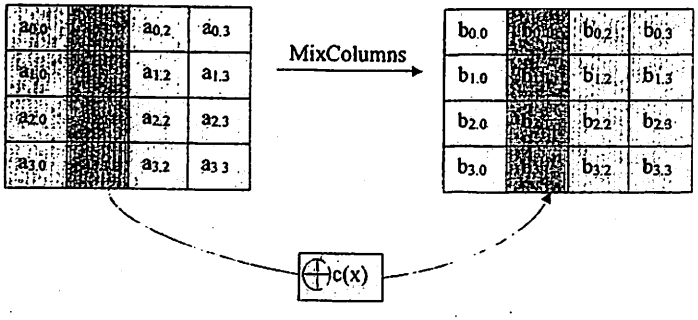


Рис.7.6. Процедура MixColumns

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.

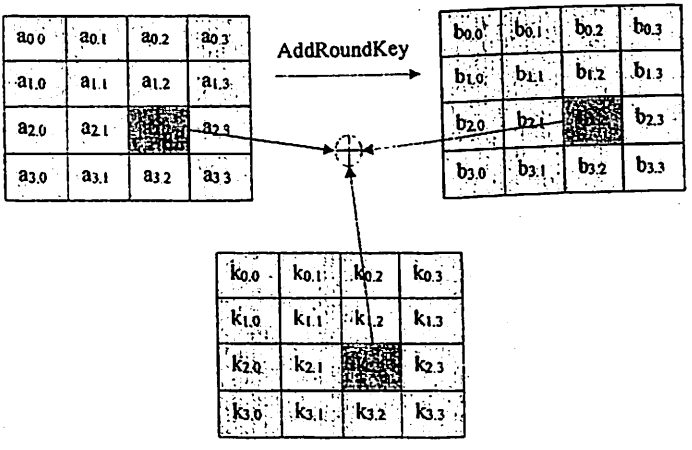


Рис.7.7. Процедура AddRoundKey()

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)

Контрольные вопросы

1. Шифрование более одного блока?
2. Какие сети существует для построения блочных алгоритмов?
3. Объясните пожалуйста принцип работы сеть Фейстеля.

Практическая работа №8

Тема: Хеширование по алгоритмам MD5, SHA-1, SHA-256, SHA-512. Оценка криптостойкости

Цель работы:

Изучить существующие алгоритмы хэш-функций для вычисления дайджестов сообщений и данных. Оценка криптостойкости современных алгоритмов хэширования данных.

Теоретическая часть

MD5 (англ. Message Digest 5) — 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института (Massachusetts Institute of Technology, MIT) в 1991 году. Предназначен для создания «отпечатков» или «дайджестов» сообщений произвольной длины и последующей проверки их подлинности. Является улучшенной в плане безопасности версией MD4.

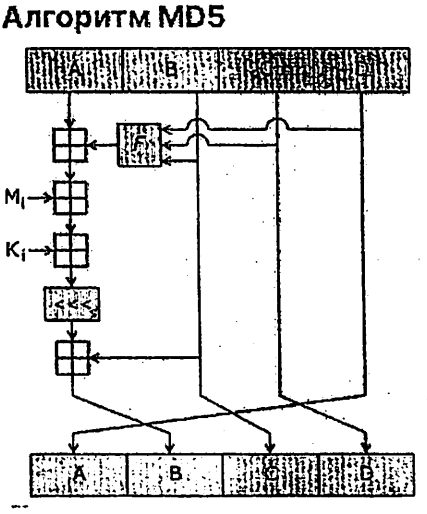


Рис. 8.1. Схема работы алгоритма MD5

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Ниже приведены 5 шагов алгоритма:

Шаг 1. Выравнивание потока

Сначала дописывают единичный бит в конец потока (байт 0x80), затем необходимое число нулевых бит. Входные данные выравниваются так, чтобы их новый размер L' был сравним с 448 по модулю 512 ($L' = 512 \times N + 448$). Выравнивание происходит, даже если длина уже сравнима с 448.

Шаг 2. Добавление длины сообщения.

В оставшиеся 64 бита дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта. Если длина превосходит 264– 1, то дописывают только младшие биты. После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

Шаг 3. Инициализация буфера.

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (шестнадцатеричное представление, сначала младший байт):

A = 01 23 45 67;

B = 89 AB CD EF;

C = FE DC BA 98;

D = 76 54 32 10.

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим

вектором. Определим ещё функции и константы, которые нам понадобятся для вычислений.

Потребуются 4 функции для четырёх раундов. Введём функции от трёх параметров— слов, результатом также будет слово.

1 раунд $FunF(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$.

2 раунд $FunG(X, Y, Z) = (X \wedge Z) \vee (\neg Z \wedge Y)$.

3 раунд $FunH(X, Y, Z) = X \oplus Y \oplus Z$.

4 раунд $FunI(X, Y, Z) = Y \oplus (\neg Z \vee X)$.

Определим таблицу констант T[1..64]— 64-элементная таблица данных, построенная следующим образом: T[i] =int(4294967296 * |sin(i) |), где 4294967296 = 232.

Выровненные данные разбиваются на блоки (слова) по 32 бита, и каждый блок проходит 4 раунда из 16 операторов. Все операторы однотипны и имеют вид: [abcd k s i], определяемый как a=b+ ((a+Fun(b,c,d) + X[k] +T[i]) < < <s), где X— блок данных. X[k] = M [n * 16 + k], где k— номер 32-битного слова из n-го 512-битного блока сообщения, и s— циклический сдвиг влево на s бит полученого 32-битного аргумента.

Шаг 4. Вычисление в цикле.

Заносим в блок данных элемент n из массива. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

Раунд 1

/*[abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]

[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]

[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]

[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

Раунд 2

/*[abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]

[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]

[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]

[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

Раунд 3

/*[abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]

[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]

[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]

[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

Раунд 4

/*[abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]

[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]

[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]

[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

A = AA + A

B = BB + B

C = CC + C

D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то изменяем номер элемента массива (n++) и переходим в начало цикла.

SHA-2(Secure Hash Algorithm Version 2 — безопасный алгоритм хеширования, версия 2)— семейство криптографических алгоритмов — однонаправленных хеш-функций, включающее в себя алгоритмы SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/256 и SHA-512/224. Хеш-функции предназначены для создания «отпечатков» или «дайджестов» для сообщений произвольной длины. Применяются в различных приложениях или компонентах, связанных с защитой информации. Хеш-функции семейства SHA-2 построены на основе структуры Меркла — Дамгора. Исходное сообщение после дополнения разбивается на блоки, каждый блок— на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хеш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя. Подробнее— см. В следующей таблице показаны некоторые технические характеристики различных вариантов SHA-2. «Внутреннее состояние» обозначает промежуточную хеш-сумму после обработки очередного блока данных:

Хеш-функция	Длина дайджеста сообщения (бит)	Длина внутреннего состояния (бит)	Длина блока (бит)	Максимальная длина сообщения (бит)	Длина слова (бит)	Количество итераций в цикле	Скорость (MiB/s)
SHA-256, SHA-224	256/224	256 (8×32)	512	264− 1	32	64	139
SHA-512, SHA-384, SHA-512/256, SHA-512/224	512/384/256/224	512 (8×64)	1024	2128− 1	64	80	154

Псевдокод использует следующие битовые операции:

I—конкатенация,

+—сложение,

and—побитовое «И»,

xor—исключающее «ИЛИ»,

shr (shift right)—логический сдвиг вправо,

rotr (rotate right)—циклический сдвиг вправо.

SHA-256 править править код.

Пояснения:

Все переменные беззнаковые, имеют размер 32 бита и при вычислениях суммируются по модулю 232 message— исходное двоичное сообщение m— преобразованное сообщение

Инициализация переменных.

(первые 32 бита дробных частей квадратных корней первых восьми простых чисел [от 2 до 19]):

h0:= 0x6A09E667

h1:= 0xBB67AE85

h2:= 0x3C6EF372

h3:= 0xA54FF53A

h4:= 0x510E527F

h5:= 0x9B05688C

h6:= 0x1F83D9AB

h7:= 0x5BE0CD19

Таблица констант.

(первые 32 бита дробных частей кубических корней первых 64 простых чисел [от 2 до 311]):

k[0..63]:=

0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5, 0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5,

0xD807AA98, 0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74, 0x80DEB1FE, 0x9BDC06A7, 0xC19BF174,

0xE49B69C1, 0xEFBE4786, 0x0FC19DC6, 0x240CA1CC, 0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,

0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3, 0xD5A79147, 0x06CA6351, 0x14292967,

0x27B70A85, 0x2E1B2138, 0x4D2C6DFC, 0x53380D13, 0x650A7354, 0x766A0ABB, 0x81C2C92E, 0x92722C85,

0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3, 0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070,

0x19A4C116, 0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3, 0x4ED8AA4A, 0x5B9CCA4F, 0x682E6FF3,

0x748F82EE, 0x78A5636F, 0x84C87814, 0x8CC70208, 0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2

Предварительная обработка:

m:= message **|** [единичный бит]

m:= m ǁ [k нулевых бит], где k— наименьшее неотрицательное число, такое, что

(L + 1 + K) mod 512 = 448, где L — число бит в сообщении (сравнима по модулю 512 с 448)

m:= m ǁ Длина(message) — длина исходного сообщения в битах в виде 64-битного числа с порядком байтов от старшего к младшему.

Далее сообщение обрабатывается последовательными порциями по 512 бит: разбить сообщение на куски по 512 бит для каждого куска разбить кусок на 16 слов длиной 32 бита (с порядком байтов от старшего к младшему внутри слова): w[0..15].

Сгенерировать дополнительные 48 слов: для i от 16 до 63

s0:= (w[i-15] rotr 7) xor (w[i-15] rotr 18) xor (w[i-15] shr 3)

s1:= (w[i-2] rotr 17) xor (w[i-2] rotr 19) xor (w[i-2] shr 10)

w[i]:= w[i-16] + s0 + w[i-7] + s1

Инициализация вспомогательных переменных:

a:= h0

b:= h1

c:= h2

d:= h3

e:= h4

f:= h5

g:= h6

h:= h7

Основной цикл:

для i от 0 до 63

Σ0:= (a rotr 2) xor (a rotr 13) xor (a rotr 22)

Ma:= (a and b) xor (a and c) xor (b and c)

t2:= Σ0 + Ma

Σ1 := (e rotr 6) xor (e rotr 11) xor (e rotr 25)

Ch:= (e and f) xor ((not e) and g)

t1:= h + Σ1 + Ch + k[i] + w[i]

h:= g

g:= f

f:= e

e:= d + t1

d:= c

c:= b

b:= a

a:= t1 + t2

Добавить полученные значения к ранее вычисленному результату:

h0:= h0 + a

h1:= h1 + b

h2:= h2 + c

h3:= h3 + d

h4:= h4 + e

h5:= h5 + f

h6:= h6 + g

h7:= h7 + h

Получить итоговое значение хеша:

digest = hash = h0 ǀ h1 ǀ h2 ǀ h3 ǀ h4 ǀ h5 ǀ h6 ǀ h7

SHA-224 идентичен SHA-256, за исключением:

для инициализации переменных h0—h7 используются другие начальные значения, в итоговом хеше опускается значение h7.

Начальные значения переменных h0—h7 в SHA-224:

h0:= 0xC1059ED8

h1:= 0x367CD507

h2:= 0x3070DD17

h3:= 0xF70E5939

h4:= 0xFFC00B31

h5:= 0x68581511

h6:= 0x64F98FA7

h7:= 0xBEFA4FA4

SHA-512 имеет идентичную структуру, но:

1. слова имеют длину 64 бита,
2. используется 80 раундов вместо 64,
3. сообщение разбито на чанки по 1024 бит,
4. начальные значения переменных и константы расширены до 64 бит,
5. постоянные для каждого из 80 раундов— 80 первых простых чисел,
6. сдвиг в операциях rotr и shr производится на другое число позиций.

Начальные значения переменных h0—h7 в SHA-512:

h0:= 0x6a09e667f3bcc908,

h1:= 0xbb67ae8584caa73b,

h2:= 0x3c6ef372fe94f82b,

h3:= 0xa54ff53a5f1d36f1,

h4:= 0x510e527fade682d1,

h5:= 0x9b05688c2b3e6c1f,

h6:= 0x1f83d9abfb41bd6b,

h7:= 0x5be0cd19137e2179

SHA-384 идентичен SHA-512, за исключением:

Переменные h0—h7 имеют другие начальные значения, в итоговом хеше опускаются значения h6 и h7.

Начальные значения переменных h0—h7 в SHA-384

(первые 64 бита дробных частей квадратных корней простых чисел с 9-го по 16-е [от 23 до 53]):

h0:= CBBB9D5DC1059ED8

h1:= 629A292A367CD507

h2:= 9159015A3070DD17

h3:= 152FECD8F70E5939

h4:= 67332667FFC00B31

h5:= 8EB44A8768581511

h6:= DB0C2E0D64F98FA7

h7:= 47B5481DBEFA4FA4

SHA-512/256 идентичен SHA-512, за исключением:

Переменные h0—h7 имеют другие начальные значения, итоговый хеш обрезается до левых 256 бит.

Начальные значения переменных h0—h7 в SHA-512/256:

h0:= 22312194FC2BF72C

h1:= 9F555FA3C84C64C2

h2:= 2393B86B6F53B151

h3:= 963877195940EABD

h4:= 96283EE2A88EFFE3

h5:= BE5E1E2553863992

h6:= 2B0199FC2C85B8AA

h7:= 0EB72DDC81C52CA2

SHA-512/224 идентичен SHA-512, за исключением:

Переменные h0—h7 имеют другие начальные значения, итоговый хеш обрезается до левых 224 бит.

Начальные значения переменных h0—h7 в SHA-512/224:

h0:= 8C3D37C819544DA2

h1:= 73E1996689DCD4D6

h2:= 1DFAB7AE32FF9C82

h3:= 679DD514582F9FCF

h4:= 0F6D2B697BD44DA8

h5:= 77E36F7304C48942

h6:= 3F9D85A86A1D36C8

h7:= 1112E6AD91D692A1

Хеш-функции SHA-2 используются для проверки целостности данных и в различных криптографических схемах. На 2008 год семейство хеш-функций SHA-2 не имеет такого широкого распространения, как MD5 и SHA-1[17], несмотря на обнаруженные у последних недостатки.

Некоторые примеры применения SHA-2 указаны в таблице:

Область применения	Детали
S/MIME	SHA-224, SHA-256, SHA-384 или SHA-512 дайджесты сообщений
OpenLDAP	SHA-256, SHA-384 или SHA-512 хеши паролей
DNSSEC	SHA-256 дайджесты DNSKEY в протоколе DNSSEC
X.509	SHA-224, SHA-256,SHA-384 и SHA-512 используются для создания электронной цифровой подписи сертификата
PGP	SHA-256, SHA-384, SHA-512 используются для создания электронной цифровой подписи
IPSec	Некоторые реализации поддерживают SHA-256 в протоколах ESP и IKE
DSA	Семейство SHA-2 используется для создания электронной цифровой подписи

SHACAL-2	Блочный алгоритм шифрования SHACAL-2 построен на основе хеш-функции SHA-256
Биткойн	Нахождение комбинации данных, SHA-256-хеш которых удовлетворяет оговоренному условию, является доказательством выполнения работы при эмиссии криптовалюты

Задание

Вычислите хеш значения свои фамилия имя на основе алгоритма MD5.

Контрольные вопросы

1. Что такое хэш-функции и их функции.

2. Опишите хэш-функцию MD5.

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

Практическая работа №9

Тема: Шифрование данных с помощью криптоалгоритмов с открытым ключом с использованием библиотеки OpenSSL

Цель работы: Изучение метод криптографической защиты данных основанные на криптосистемы с открытыми ключоми. Шифрование файлов с использованием пакета программ OpenSSL..

Полно ситуаций когда нужно зашифровать определённый файл или папку. Например, если данные передаются по открытым каналам либо сохраняются на внешнем носителе. Многие (в том числе и я) используют truecrypt, однако основное предназначение этой программы — работа с зашифрованными разделами, поэтому она не очень хороша в этом случае. Для подобных задач вполне подходит OpenSSL — надёжное кросплатформенное решение. OpenSSL поддерживает различные алгоритмы шифрования, плюс он по умолчанию установлен во многих операционных системах, а установка на остальные не составит труда. Под хабракатом — основы использования симметричного и асимметричного шифрования в OpenSSL, а таке пара скриптов упрощающих асимметричное шифрование с одноразовым ключом. Простейший способ защиты данных с помощью OpenSSL — симметричное шифрование. Следующие команды шифруют и расшифровывают файл documents.zip, используя алгоритм AES с длиной ключа 256 бит: openssl enc -aes-256-cbc -salt -in documents.zip -out documents.enc openssl enc -d -aes-256-cbc -in documents.enc -out documents.zip. Проблема этих команд может заключаться в том что они требуют ввода пароля. Есть ситуации когда это нежелательно. Например, автоматическое резервное копирование/шифрование данных по расписанию, либо если данные шифруются одним человеком а расшифровываются другим. Как раз для таких случаев было придумано шифрование с открытым ключом. В общем случае вам понадобится создать открытый и закрытый ключи. Первая команда сгенерирует закрытый ключ private.pem, вторая создаст открытый ключ public.pem: openssl genrsa -out private.pem -aes256 2048 openssl rsa -in

private.pem -pubout -out public.pem В результате вы получаете пару RSA ключей длиной 2048 бит. К сожалению, в системе RSA размер шифруемых данных ограничен размером ключа, поэтому зашифровать более 2Кб данных не получится. Есть способ обойти это — информация сначала шифруется симметричным алгоритмом (подобно использованному выше) с использованием одноразового ключа. Затем этот одноразовый ключ шифруется публичным ключом. При расшифровке одноразовый ключ расшифровывается закрытым. Подробнее об этом уже было очень хорошо написано в статье на Хабре. Автоматизировать шифрование поможет следующий скрипт, на выходе которого вы получите одноразовый ключ и данные (encrypt.sh) в зашифрованном виде: # !/bin/bash

FILENAME="$1"

PUBLICKEY="$2"

SESSIONKEY="$3"

RESULT="$4"

```
# Generate the random symmetric-key
PASSIZE=30
if [ -c /dev/urandom ] ; then
KEY=`head -c 30 /dev/urandom | openssl enc -base64`
else
KEY=`openssl rand -base64 30`
fi
export KEY

# Encrypt the symmetric key using the public key
openssl rsautl -encrypt -inkey "$PUBLICKEY" -out "$SESSIONKEY" -pubin <<EOF
$KEY
EOF

# Encrypt the file
openssl enc -aes-256-cbc -pass env:KEY -in "$FILENAME" -out "$RESULT"
```

Следующая команда использует открытый ключ public.pem чтобы зашифровать файл documents.zip. Она сгенерирует зашифрованный

одноразовый ключ session.key и зашифрованные данные documents.enc: ./encrypt.sh documents.zip public.pem session.key documents.enc.

```
./encrypt.sh documents.zip public.pem session.key documents.enc
```

Скрипт для дешифрования (decrypt.sh):

```
# !/bin/bash

PRIVATEKEY="$1"
SESSIONKEY="$2"
ENCRYPTED="$3"
DECRYPTED="$4"

# Decrypt the symmetric key using the private key
KEY=`openssl rsautl -decrypt -inkey "$PRIVATEKEY" -in "$SESSIONKEY"`
export KEY

# Decrypt the file
openssl enc -aes-256-cbc -d -pass env:KEY -in "$ENCRYPTED" -out "$DECRYPTED"
```

Команда для дешифрования использует закрытый ключ private.pem и одноразовый ключ session.key чтобы расшифровать файл documents.enc. Она сгенерирует файл documents.zip:

```
./decrypt.sh private.pem session.key documents.enc documents.zip
```

Как видите, шифрование с открытым ключом может быть почти таким же простым как и симметричное. Но есть ещё более простой путь. На написание этого поста меня побудил блог SbF□. Его автор (несомненно более искушённый в bash чем я) написал скрипт, который архивирует папку, шифрует её открытым ключом и генерирует другой скрипт, содержащий в себе всё необходимое: одноразовый ключ, данные и собственно команды для расшифровывания. Кроме того, скрипт может сгенерировать для вас пару RSA ключей:

```
./encrypt-file.sh -keys public.pem private.pem
./encrypt-file.sh folder public.pem > decrypt-folder.sh
chmod +x decrypt-folder.sh
./decrypt-folder.sh private.pem > folder.tar
```

В этом примере мы сначала сгенерировали пару ключей. После этого папка folder была зашифрована в скрипт decrypt-folder.sh а затем расшифрована в архив folder.tar. Возможный минус этого способа — то что данные в decrypt-folder.sh хранятся в формате BASE64, а следовательно их размер увеличивается.

Задание

Зашифруйте свою фамилию на основе алгоритма RSA с использованием OpenSSL.

Контрольные вопросы

1. Объясните возможности и функциональности Open SSL?
2. Объясните, как работает криптосистема с открытым ключом?

Практическая работа № 10

Тема: Применение электронной цифровой подписи для проверки авторства и неизменности файла.

Цель работы: Рассмотреть понятие и принципы работы электронной подписи; ознакомиться с классификацией методов реализации электронной подписи; изучить методы электронной подписи.

Теоретическая часть

В соответствии со статьей 160 п. 2 ГК РФ ЭП является аналогом рукописной подписи. То есть при помощи цифрового продукта можно заверить любой документ. Особенность ЭЦП в том, что закон ее приравнял к росчерку чернилами без дополнительных условий. Поэтому такие файлы будут приняты всеми учреждениями, даже судами. Разница между двумя способами заверения только в том, как выглядит электронная подпись. Это не привычное сокращение от фамилии, а уникальная комбинация символов, которая не только указывает на автора сообщения, но и подтверждает достоверность информации. Визуализация ЭЦП не предусмотрена, поэтому при открытии документа ее не увидеть, кроме особых случаев. Например, при выдаче выписки из Росреестра в цифровой форме можно просмотреть подпись в самом файле, открыв его в специальном сервисе. Этот вариант ЭП может встраиваться в электронный файл или располагается в отдельном документе с расширением .sig. Усиленная подпись получается в итоге криптографических операций. Поэтому посмотреть на нее, как на обычную, не получится. Ее не видно ни в самом файле, ни на физическом носителе. Поэтому ее нельзя взять за образец, как рукописную, и подделать. Можно только понять, что файл подписан усиленной квалифицированной ЭП.

Для проверки нужно выполнить следующие действия:

1. открыть файл, заверенный ЭЦП, с расширением .sig при помощи программы-криптопровайдера. Это первый шаг к тому, чтобы посмотреть, как выглядит электронная подпись;

2. нажать на раздел «Подписанные данные» (его имя может быть другим, его определяет применяемое ПО);

3. в окне появится информация о подписи и сертификате: факт простановки ЭП, время формирования, алгоритм и логин для использования ресурса.

Такие же данные легко посмотреть в документах с расширениями, входящими в офисный комплект. Для этого требуется вызвать контекстное меню и раскрыть раздел «Свойства». Поэтому электронная подпись становится неотъемлемой частью документа. Но в ЭЦП налоговой службы, которая показывается на выписке из Росреестра юридических лиц и индивидуальных предпринимателей, в зависимости от того, усиленная подпись используется или нет, можно рассмотреть произвольный комплект символов или изображение. На заверение электронной подписью указывает специальный значок. Отредактировать такой файл не получится, поэтому можно быть уверенным в достоверности информации. Злоумышленники не смогут перехватить и изменить данные. После отправки на печать документы не будут выглядеть подписанными, поэтому они становятся недействительными.

Несмотря на то что нельзя на практике выяснить, как выглядит цифровая подпись, легко насладиться видом ее носителя. Это может быть:

1. токен— специализированное ПО, которое запрашивает пароль для работы с ним, выглядит как флешка, нужен, чтобы усилить защиту информации;

2. Smart-карта— похожа на стандартную банковскую карту;

3. Flash— небольшое устройство с USB-разъемом и картой памяти для сохранения информации, простой вариант без дополнительной защиты;

4. SIM-карта— устройство от мобильного оператора с предустановленным Java-приложением.

Выбрать можно любой вариант, но образцовый уровень защиты пока может обеспечить только токен. А еще им проще всего пользоваться, потому

что он не требует дополнительного ПО и оборудования, совместим с распространенными приложениями и операционными системами. Если токен украдут или он будет потерян, другой человек не сможет с его помощью подписать ни одного сообщения. Код активации есть только у владельца, а без него доступа к ЭЦП нет. Такой способ может усилить уже имеющуюся защиту от подмены данных, которые пересылаются через интернет. Теоретически электронный продукт все-таки можно взломать, но для этого нужен квалифицированный специалист и время. Со вторым пунктом точно будут проблемы, потому что владелец может быстро завить о пропаже, что сделает старую версию недействительной. А вот получить доступ к электронной подписи на обычной флешке или телефоне с установленной сим-картой гораздо проще. Достаточно их украсть, и можно заверять любые сообщения.

Поэтому сертификат очень важен, за правильность его составления УЦ несет ответственность. Если по вине организации с этой частью возникнут проблемы, может быть наложен штраф.

В законодательстве указаны следующие виды сертификатов:

1. обычный— выпускается для неквалифицированной ЭЦП, заказывается в любом центре;

2. квалифицированный — создается для аналогичной подписи, выдавать его разрешено только организациям, прошедшим аккредитацию.

Выглядят эти документы как несвязанные наборы байтов. Но, кроме технической стороны, есть и организационные моменты, которые решают работники удостоверяющего центра. Компания ответственна за подтверждение открытых частей ключа, которые используются владельцами для выполнения юридических, банковских и иных операций. Кроме сертификата, пользователю выдается ключевая пара и устройство выполнения криптозащитных операций. В качестве последнего средства может выступать ПО для подключения к облаку, токен с криптопровайдером или дополнительное приложение, которое нужно установить на компьютер. Ключевая пара — два комплекта байтов, которые формируются средством

электронной подписи. В первом наборе содержится закрытый ключ, который применяется для создания самой подписи. Вторая часть, или открытый сегмент, нужна для подтверждения достоверности ЭЦП. Открытый сегмент может быть увиден всеми, но обязательно прикрепляется к секретной части. Но чтобы подписать электронный документ и усилить его защиту, не обязательно вникать в технические подробности. Пользователь сможет сам без проблем применять ЭЦП без этой информации.

В соответствии с законом в ЭП должна входить форма с отметкой о цифровой подписи. Этот атрибут должен содержать любой электронный документ при его просмотре, печати или сканировании.

Он находится в том месте, где обычно на документе ставится рукописная подпись. В состав штампа ЭП входят следующие реквизиты:

1. номер сертификата;
2. ФИО того, кому принадлежит подпись;
3. дата окончания действия сертификата;
4. данные о заверении файла ЭП.

- *Для этого есть сервис на портале госуслуг.*

1. Чтобы проверить, нужно выполнить следующие действия.
2. Выбрать подходящий раздел — интегрированная или отсоединяемая подпись.
3. В появившуюся форму загрузить документ для проверки.
4. Загрузить ЭП, обычно она имеет формат .p7s или .sig;
5. Внизу страницы появится код, который нужно вписать в соответствующее поле. После этого можно нажать кнопку «Проверить».

В результате появятся данные о документе, если в нем не выполнялись изменения. Электронная подпись позволяет выйти за рамки обычного делопроизводства. Она все глубже входит в жизнь рядовых пользователей, помогая экономить много времени. Физлица могут скачивать через аккаунт налогоплательщика сертификат, чтобы потом включить ее в свою

электронную систему. ЭЦП для юридических лиц выдают в удостоверяющих центрах. Единого вида ЭП на документе не существует, потому что каждый вариант обладает своими особенностями. ЭЦП может складываться из набора знаков, быть совершенно невидимой или иметь вид отметки с печатью и подписью. Бывает несколько типов ЭП, у каждого из которых свой уровень защиты данных. Усиленный квалифицированный вариант позволяет передавать секретные данные, кроме составляющих гостайну.

Задание

1. Формируйте электронно цифровой подпись с использованием Open SSL.

Контрольные вопросы

1. Объясните роль и принцип работы ЭЦП?
2. Модели атак и их возможные результаты?
3. Получение двух документов с одинаковой подписью (коллизия второго рода)?

Практическая работа №11

Тема: Создание сертификата X.509 с использованием библиотеки OpenSSL.

Цель работы: Формирование навыка применения программного продукта OpenSSL для создания сертификатов X.509 и их преобразования, изучить структуру сертификата X.509 и форматы DER и PEM.

Теоретическая часть

X.509 — стандарт ITU-T (англ. International Telecommunication Union — Telecommunication sector) для инфраструктуры открытого ключа (англ. Public key infrastructure, PKI) и инфраструктуры управления привилегиями (англ. Privilege Management Infrastructure).

X.509 определяет стандартные форматы данных и процедуры распределения открытых ключей с помощью соответствующих сертификатов с цифровыми подписями. Эти сертификаты предоставляются удостоверяющими центрами (англ. Certificate Authority). Кроме того, X.509 определяет формат списка аннулированных сертификатов, формат сертификатов атрибутов и алгоритм проверки подписи путём построения пути сертификации. X.509 предполагает наличие иерархической системы удостоверяющих центров для выдачи сертификатов.

Структура сертификата X.509

- Сертификат
- Версия
- Серийный номер
- Идентификатор алгоритма подписи
- Имя издателя
- Период действия:
    - Не ранее
    - Не позднее
- Имя субъекта
- Информация об открытом ключе субъекта:

- Алгоритм открытого ключа
- Открытый ключ субъекта

- Уникальный идентификатор издателя (обязательно только для v2 и v3)
- Уникальный идентификатор субъекта (обязательно только для v2 и v3)
- Дополнения (для v2 и v3)
  - Возможные дополнительные детали
- Алгоритм подписи сертификата (обязательно только для v3)
- Подпись сертификата (обязательно для всех версий)

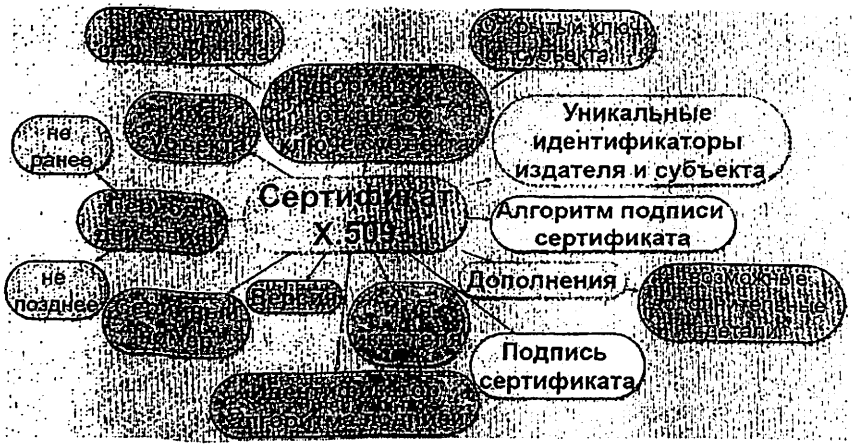


Рис.11.1 Общий стандарт для интернета, использующий формат X.509

В 4 разделе RFC 2459 описана структура сертификата X.509 v3. Обновленная структура сертификата X.509 v3 описана в документе RFC 5280 в 4 разделе. Она специализирована под интернет-приложения.

Для описания внутренней структуры сертификатов X.509 используется ASN.1. Хранятся, как правило, в виде DER или PEM-файлов. Общепринятое расширение .cer или .crt. Может быть и другое расширение.

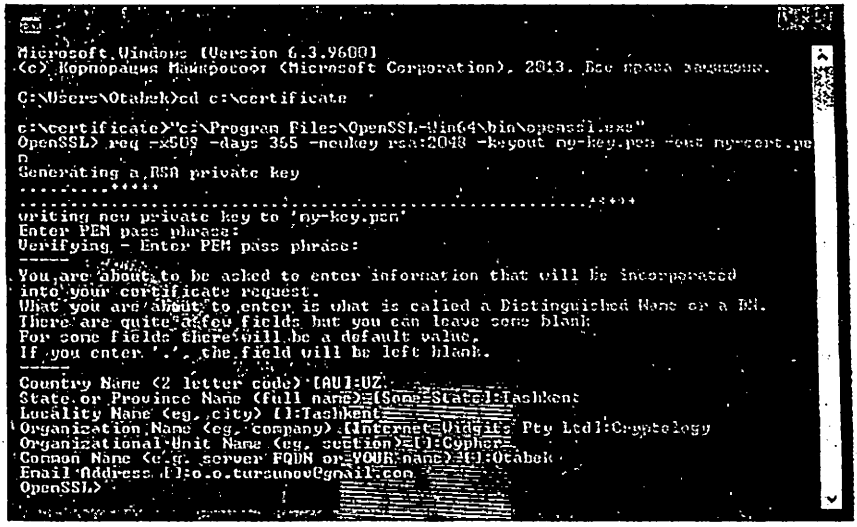


Рис 11.1. Создание закрытого ключа и подписание сертификата X.509

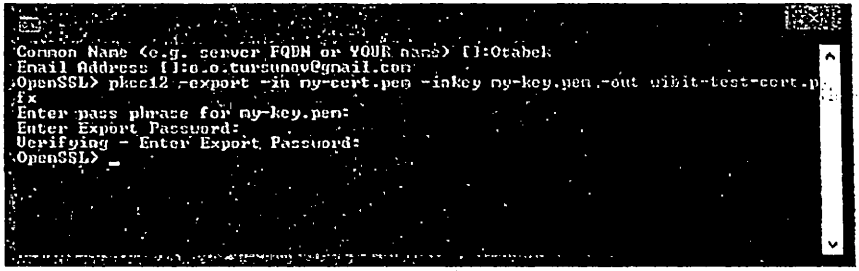


Рис 11.2. Преобразование сертификата в файл pfx.

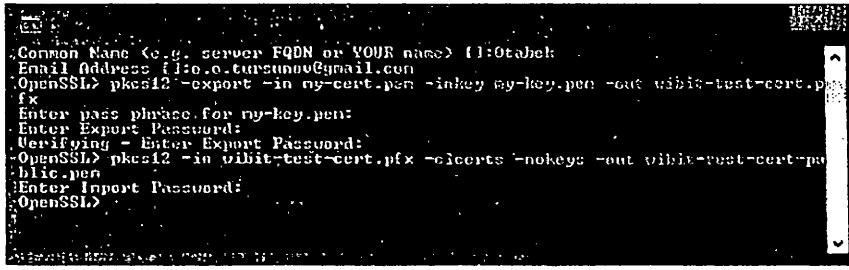


Рис 11.3. Генерация открытого ключа из файла pfx

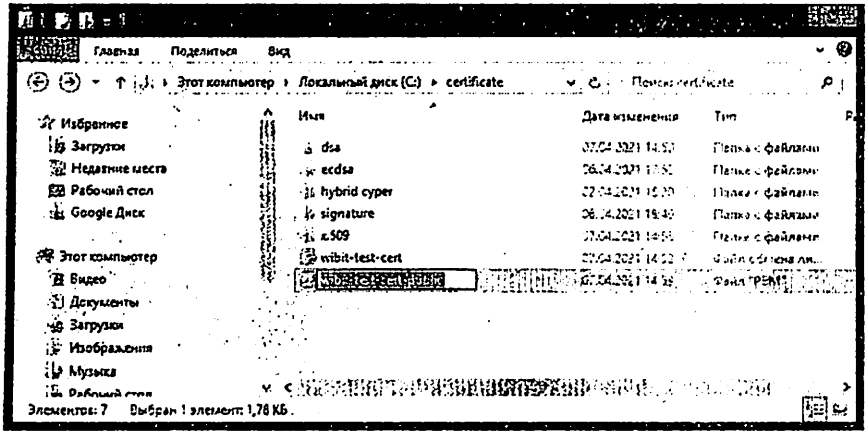


Рис 11.4. Файл открытого ключа

Задание

1. Создать сертификат X.509, сделать снимки экрана процесса создания сертификата, подготовить отчёт с этими снимками.

Контрольные вопросы

1. Объясните роль и цель сертификата X.509?
2. Какие средства существует для создание сертификата X.509?
3. Какие сертификаты используется для обеспечения конфиденциальности передаваемых данных в сети?

Практическая работа № 12

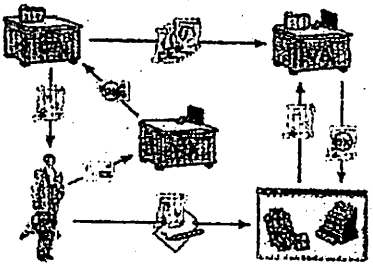
Тема: Создание центра сертификации, поддерживающего список отозванных сертификатов, с использованием библиотеки OpenSSL.

Цель работы: Приобрести теоретические и практические навыки о создание центра сертификации поддерживающеги списка отозванных сертификатов с применением библиотека OpenSSL.

Теоретическая часть

Инфраструктура открытых ключей (ИОК, англ. PKI — Public Key Infrastructure) — набор средств (технических, материальных, людских и т. д.), распределённых служб и компонентов, в совокупности используемых для поддержки криптозадач на основе закрытого и открытого ключей. В основе PKI лежит использование криптографической системы с открытым ключом и несколько основных принципов:

- закрытый ключ (private key) известен только его владельцу;
- удостоверяющий центр (УЦ или CA - Certificate Authority) создает электронный документ — сертификат открытого ключа, таким образом удостоверяя факт того, что закрытый (секретный) ключ известен эксклюзивно владельцу этого сертификата, открытый ключ (public key) свободно передается;
- никто не доверяет друг другу, но все доверяют удостоверяющему центру;
- удостоверяющий центр подтверждает или опровергает принадлежность открытого ключа заданному лицу, которое владеет соответствующим закрытым ключом.



Инфраструктура открытых ключей

Основные задачи системы информационной безопасности, которые решает инфраструктура управления открытыми ключами:

- обеспечение конфиденциальности информации;
- обеспечение целостности информации;
- обеспечение аутентификации пользователей и ресурсов, к которым обращаются пользователи;
- обеспечение возможности подтверждения совершенных пользователями действий с информацией (неотказуемость/неотрекаемость — англ. non-repudiation).

Архитектуры PKI

В основном выделяют 5 видов архитектур PKI, это:

1. простая PKI (одиночный УЦ)
2. иерархическая PKI
3. сетевая PKI
4. кросс-сертифицированные корпоративные PKI
5. архитектура мостового УЦ

В основном PKI делятся на разные архитектуры по следующим признакам:

- количество УЦ (а также количество УЦ, которые доверяют друг-другу)
- сложность проверки пути сертификации

- последствия выдачи злоумышленником себя за УЦ

OpenSSL — это криптографическая библиотека со свободным и открытым исходным кодом, которая предоставляет несколько инструментов командной строки для работы с цифровыми сертификатами. Некоторые из этих инструментов могут быть использованы в качестве центра сертификации.

Центр сертификации (CA) является объектом, который подписывает цифровые сертификаты. Многие веб-сайты нуждаются в том, чтобы их клиенты знали, что соединение является безопасным, поэтому они платят международным доверенным центрам сертификации (например, VeriSign, DigiCert) за подпись сертификата для своего домена.

В некоторых случаях имеет больше смысла выступать в качестве вашего собственного центра сертифкации, чем платить центрам сертификации, таким как DigiCert. Например, обеспечение безопасности вебсайта в интранете или для выдачи собственных сертификатов клиентам, чтобы позволить им проверять подлинность сервера (Apache, OpenVPN).

Практическая часть

Создание корневой пары ключей

Выступая в качестве центра сертификации (CA) означает иметь дело с криптографическими парами приватных (частных) ключей и публичными сертификатами. Самой первой криптографической парой создадим корневую пару. Она состоит из корневого ключа (ca.key.pem) и корневого сертификата (ca.cert.pem). Эта пара образует идентичность нашего центра сертификации. Как правило, корневой центр сертификации не подписывает напрямую серверные или клиентские сертификаты. Корневой центр сертификации используется только для создания одного или нескольких промежуточных центров сертификации, которые являются доверенными корневому центру сертификации чтоб подписывать сертификаты от их имени. Это является лучшей практикой. Таким образом, корневой ключ может хранится «оффлайн» и по-максимому не использоваться, так как любая компрометация губительна для ключа. В качестве примера лучшей практики является

создание корневой пары в максимально защищенной среде. В идеале это будет полностью зашифрованный, физически изолированный компьютер, который постоянно отключен от интернета. Следует вынуть беспроводную сетевую карту, а порт обычной сетевой карты залить клеем.

Подготовка каталогов

Выберите каталог для хранения всех ключей и сертификатов. К примеру, /root/ca (все производится на Ubuntu 14.04.3 LTS)

```
mkdir /root/ca
```

Создадим структуру каталогов. Файлы index.txt и serial будут выступать в качестве плоской базы для отслеживания подписанных сертификатов.

```
# cd /root/ca
# mkdir certs crl newcerts private
# chmod 700 private
# touch index.txt
# echo 1000 > serial
```

Подготовка конфигурационных файлов

Необходимо создать конфигурационный файл для OpenSSL. Создадим файл /root/ca/openssl.cnf, скопируем в него следующее содержимое root-config.txt.

Раздел [ca] является обязательным. Здесь мы говорим OpenSSL использовать параметры из раздела [CA_default] :

```
[ ca ]
#man ca
default_ca = CA_default
```

Раздел [CA_default] содержит ряд значений по умолчанию:

```
[ CA_default ]
# Directory and file locations.
dir             = /root/ca
```

```
certs           = $dir/certs
crl_dir         = $dir/crl
new_certs_dir   = $dir/newcerts
database        = $dir/index.txt
serial          = $dir/serial
RANDFILE        = $dir/private/.rand
# The root key and root certificate.
private_key     = $dir/private/ca.key.pem
certificate     = $dir/certs/ca.cert.pem
# For certificate revocation lists.
crlnumber       = $dir/crlnumber
crl             = $dir/crl/ca.crl.pem
crl_extensions  = crl_ext
default_crl_days  = 30
# SHA-1 is deprecated, so use SHA-2 instead.
default_md      = sha256
name_opt        = ca_default
cert_opt        = ca_default
default_days    = 375
preserve        = no
policy          = policy_strict
```

Policy_strict будет применяться для всех подписей корневого центра сертификации, и корневой центр сертификации будет использоваться только для создания промежуточных центров сертификации.

```
[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of man ca.
countryName             = match
stateOrProvinceName     = match
organizationName        = match
```

```
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional
```

Применим policy_loose для всех подписей промежуточных центров серфтификации, так как промежуточные центры сертификации это подписывающие серверы, и клиентские сертификаты могут приходить от различных третьих лиц.

```
[ policy_loose ]
# Allow the intermediate CA to sign a more diverse range of certificates.
# See the POLICY FORMAT section of the `ca` man page.
countryName            = optional
stateOrProvinceName    = optional
localityName           = optional
organizationName       = optional
organizationalUnitName = optional
commonName             = supplied
emailAddress           = optional
```

Параметры из секции [req] применяются когда создаются сертификаты или запросы на подписывание сертификатов.

```
[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
```

x509_extensions = v3_ca

Секция [req_distinguished_name] определяет информацию, которая обычно требуется при запросе на подписывание сертификата. Можно указать некоторые значения по умолчанию.

```
[ req_distinguished_name ]
# See https://en.wikipedia.org/wiki/Certificate_signing_request.
countryName         = Country Name (2 letter code)
stateOrProvinceName = State or Province Name
localityName        ocality Name
organizationName    = Organization Name
organizationalUnitName = Organizational Unit Name
commonName          = Common Name
emailAddress        = Email Address
# Optionally, specify some defaults.
countryName_default   = GB
stateOrProvinceName_default = England
localityName_default      = England
organizationName_default   = Alice Ltd
#organizationalUnitName_default =
#emailAddress_default        =
```

Следующие несколько секций являются расширениями, которые могут применять при подписывании сертификатов. Например, при указании аргумента командной строки -extensions v3_ca будут применены расширения из секции [v3_ca]. Эти расширения будут применяться при создании корневого сертификата.

```
[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
```

basicConstraints = critical, CA:true

keyUsage = critical, digitalSignature, cRLSign, keyCertSign

При создании сертификата промежуточного центра сертификации будут применяться расширения из v3_ca_intermediate.

Параметр pathlen:0 указывает, что не может быть никаких дальнейших центров сертификации ниже промежуточного центра сертификации.

[v3_intermediate_ca]

Extensions for a typical intermediate CA (`man x509v3_config`).

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid:always,issuer

basicConstraints = critical, CA:true, pathlen:0

keyUsage = critical, digitalSignature, cRLSign, keyCertSign

Расширение usr_cert будет применяться при подписывании клиентских сертификатов, таких, которые используются для аутентификации удаленных пользователей.

[usr_cert]

Extensions for client certificates (`man x509v3_config`).

basicConstraints = CA:FALSE

nsCertType = client, email

nsComment = "OpenSSL Generated Client Certificate"

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid,issuer

keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment

extendedKeyUsage = clientAuth, emailProtection

Расширение server_cert будет применяться при подписывании серверных сертификатов, таких, которые используются для веб-серверов.

[server_cert]

Extensions for server certificates (`man x509v3_config`).

basicConstraints = CA:FALSE

nsCertType = server

nsComment = "OpenSSL Generated Server Certificate"

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid,issuer:always

keyUsage = critical, digitalSignature, keyEncipherment

extendedKeyUsage = serverAuth

Расширение crl_ext будет автоматически применяться при создании списков отзыва сертификатов (CRL — certificate revocation lists).

[crl_ext]

Extension for CRLs (`man x509v3_config`).

authorityKeyIdentifier=keyid:always

Расширение ocsp будет применяться при подписывании сертификата OCSP (Online Certificate Status Protocol, онлайн протокол статуса сертификатов).

[ocsp]

Extension for OCSP signing certificates (`man ocsp`).

basicConstraints = CA:FALSE

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid,issuer

keyUsage = critical, digitalSignature

extendedKeyUsage = critical, OCSPSigning

Создание корневого ключа

Создадим корневой ключ (ca.key.pem), который следует хранить защищенно. Любой, владеющий корневым ключом, может выдавать доверенные сертификаты. Зашифруем ключ стойким алгоритмом AES-256 и надежным паролем. Следует использовать 4096 битное шифрование для ключей корневого и промежуточных центров сертификации. Серверные и клиентские сертификаты возможно подписывать шифрованием с меньшей битностью.

```
# cd /root/ca
# openssl genrsa -aes256 -out private/ca.key.pem 4096
Enter pass phrase for ca.key.pem: secretpassword
Verifying - Enter pass phrase for ca.key.pem: secretpassword
# chmod 400 private/ca.key.pem
```

Создание корневого сертификата

Для создания корневого сертификата (ca.cert.pem) используется корневой ключ (ca.key.pem). Следует указать длинный срок годности сертификата, например, 20 лет. После того, как истечет корневой сертификат, все сертификаты, подписанные центром сертификации становятся недействительными. Всякий раз при использовании утилиты req следует указывать. файл конфигурации для использования с опцией –config, в противном случае OpenSSL будет использовать по умолчанию конфигурационный файл /etc/pki/tls/openssl.cnf!

```
# cd /root/ca
# openssl req -config openssl.cnf \
-key private/ca.key.pem \
-new -x509 -days 7300 -sha256 -extensions v3_ca \
-out certs/ca.cert.pem
Enter pass phrase for ca.key.pem: secretpassword
You are about to be asked to enter information that will be incorporated
into your certificate request.
-----
Country Name (2 letter code) [XX]:GB
State or Province Name []:England
Locality Name []:
Organization Name []:Alice Ltd
Organizational Unit Name []:Alice Ltd Certificate Authority
Common Name []:Alice Ltd Root CA
```

Email Address []:

chmod 444 certs/ca.cert.pem

Проверка корневого сертификата

openssl x509 -noout -text -in certs/ca.cert.pem

Вывод показывает:

- Используемый алгоритм Signature Algorithm;
- Даты периода действия сертификата Validity;
- Длину публичного ключа Public-Key;
- Эмитент сертификата Issuer, который является объектом, который подписал сертификат;
- Предмет Subject, который относится к самому сертификату.

Эмитент (Issuer) и предмет (Subject) идентичны для самоподписанных сертификатов. Все корневые сертификаты являются самоподписанными.

Signature Algorithm: sha256WithRSAEncryption

Issuer: C=GB, ST=England,

O=Alice Ltd, OU=Alice Ltd Certificate Authority,

CN=Alice Ltd Root CA

Validity

Not Before: Apr 11 12:22:58 2015 GMT

Not After : Apr 6 12:22:58 2035 GMT

Subject: C=GB, ST=England,

O=Alice Ltd, OU=Alice Ltd Certificate Authority,

CN=Alice Ltd Root CA

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

Public-Key: (4096 bit)

Вывод также показывает расширения X509v3, т.к. было применено расширение v3_ca, и опции из секции [v3_ca] отражены в выводе.

X509v3 extensions:

X509v3 Subject Key Identifier:

38:58:29:2F:6B:57:79:4F:39:FD:32:35:60:74:92:60:6E:E8:2A:31

X509v3 Authority Key Identifier:

keyid:38:58:29:2F:6B:57:79:4F:39:FD:32:35:60:74:92:60:6E:E8:2A:31

X509v3 Basic Constraints: critical

CA:TRUE

X509v3 Key Usage: critical

Digital Signature, Certificate Sign, CRL Sign

Создание промежуточной пары ключей

Промежуточным центомр сертификации является объект, который может подписывать сертификаты от имени корневого центра сертификации. Корневой центр сертификации подписывает промежуточный сертификат, образуя цепочку доверия.

Цель использования промежуточного центра сертификации, прежде всего, для обеспечения безопасности. Корневой ключ должен храниться «оффлайн» и использоваться как можно реже. Если промежуточный ключ будет скомпрометирован, корневой центр сертификации может отозвать промежуточный сертификат и создать новую промежуточную криптографическую пару.

Подготовка каталогов

Файлы корневого центра сертификации хранятся в /root/ca. Выберем каталог (/root/ca/intermediate) для хранения файлов промежуточного центра сертификации.

```
# mkdir /root/ca/intermediate
```

Создадим точно такую же структуру каталогов, которая используется для корневого центра сертификации. Также удобно создать каталог csr для хранения запросов на подпись сертификатов.

```
# cd /root/ca/intermediate
# mkdir certs crl csr newcerts private
# chmod 700 private
# touch index.txt
```

```
# echo 1000 > serial
```

Добавим файл crlnumber к дереву каталогов промежуточного центра сертификации. Crlnumber будет использоваться для отслеживания списков отзывов сертификатов.

```
# echo 1000 > /root/ca/intermediate/crlnumber
```

Создадим конфигурационный файл /root/ca/intermediate/ openssl.cnf, скопируем в него следующее содержимое intermediate-config.txt. В нем изменены пять опций, по сравнению с конфигурационным файлов корневого центра сертификации:

```
[ CA_default ]
dir           = /root/ca/intermediate
private_key   = $dir/private/intermediate.key.pem
certificate   = $dir/certs/intermediate.cert.pem
crl           = $dir/crl/intermediate.crl.pem
policy        = policy_loose
```

Создание промежуточного ключа

Создадим промежуточный ключ (intermediate.key.pem). Зашифруем ключ стойким алгоритмом AES-256 и надежным паролем.

```
# cd /root/ca
# openssl genrsa -aes256 \
-out intermediate/private/intermediate.key.pem 4096
Enter pass phrase for intermediate.key.pem: secretpassword
Verifying - Enter pass phrase for intermediate.key.pem: secretpassword
# chmod 400 intermediate/private/intermediate.key.pem
```

Создание промежуточного сертификата

Используя промежуточный ключ создадим запрос на подписывание сертификата (CSR — certificate signing request). Сведения должны, как правило, соответствовать корневому центру сертификации. Общие имена (Common Name), однако, должны быть разными. И убедитесь, что используете

конфигурационный файл промежуточного центра сертификаци (intermediate/openssl.cnf).

```
# cd /root/ca
# openssl req -config intermediate/openssl.cnf -new -sha256 \
-key intermediate/private/intermediate.key.pem \
-out intermediate/csr/intermediate.csr.pem
Enter pass phrase for intermediate.key.pem: secretpassword
You are about to be asked to enter information that will be incorporated
into your certificate request.
-----
Country Name (2 letter code) [XX]:GB
State or Province Name []:England
Locality Name []:
Organization Name []:Alice Ltd
Organizational Unit Name []:Alice Ltd Certificate Authority
Common Name []:Alice Ltd Intermediate CA
Email Address []:
```

Чтобы создать промежуточный сертификат используем корневой центр сертификации с расширением v3_intermediate_ca для подписывания промежуточного запроса. Промежуточный сертификат должен быть действителен на меньший период, чем корневой. К примеру, на 10 лет. В этот раз следует использовать конфигурационный файл корневого центра сертификации (/root/ca/openssl.cnf).

```
# cd /root/ca
# openssl ca -config openssl.cnf -extensions v3_intermediate_ca \
-days 3650 -notext -md sha256 \
-in intermediate/csr/intermediate.csr.pem \
-out intermediate/certs/intermediate.cert.pem
Enter pass phrase for ca.key.pem: secretpassword
```

Sign the certificate? [y/n]: y

chmod 444 intermediate/certs/intermediate.cert.pem

Для хранения базы данных сертификатов, выданных утилитой ca OpenSSL, используется файл index.txt. Не следует удалять или править вручную этот файл. Сейчас он должен содержать одну строку, которая относится к промежуточному сертификату.

V 250408122707Z 1000 unknown ... /CN=Alice Ltd Intermediate CA

Проверка промежуточного сертификата

Как мы делали с корневым сертификатом, убедимся, что детали промежуточного сертификата корректны.

openssl x509 -noout -text \

-in intermediate/certs/intermediate.cert.pem

Сверим промежуточный сертификат с корневым сертификатом. OK показывает, что цепочка доверия не повреждена.

openssl verify -CAfile certs/ca.cert.pem \

intermediate/certs/intermediate.cert.pem

intermediate.cert.pem: OK

Создание файла цепочки сертификатов

Когда приложение (например, веб-браузер) пытается проверить сертификат, подписанный промежуточным центром сертификации, оно также должно проверить промежуточный сертификат от корневого центра сертификации. Для выполнения проверки цепочки доверия создадим цепочку сертификатов центра сертификации для предоставления приложениям. Чтобы создать цепочку сертификатов требуется объеденить промежуточные и корневые сертификаты вмете. Позже мы будем использовать этот файл для проверки сертификатов подписанных промежуточным центром сертификации.

cat intermediate/certs/intermediate.cert.pem \

certs/ca.cert.pem > intermediate/certs/ca-chain.cert.pem

chmod 444 intermediate/certs/ca-chain.cert.pem

Наш файл цепочки сертификатов должен содержать корневой сертификат, потому что клиентские приложения ничего не знают о нем. Лучшим вариантом, особенно, если вы администратор сети, является установка корневого сертификата на каждом клиенте, которому требуется подключаться к приложению.

Подписывание серверного и клиентского сертификатов

Подпишем сертификаты используя наш промежуточный центр сертификации. Эти подписанные сертификаты можно использовать в различных ситуациях, таких как защищать соединения к веб серверу или аутентифицировать клиентов, присоединяющихся к сервису.

Указанные ниже шаги с вашей точки сзрения, где вы выступаете в качестве центра сертификации. Стороннее лицо, однако, вместо этого может создать свой собственный частный ключ и запрос на подпись сертификата (CSR) без раскрытия вам своего частного ключа. Они дают вам собственный CSR, а вы отдаете им подписанный сертификат. В этом случае, пропустите команды genrsa и req.

Создание ключа

Наши корневая и промежуточная пары 4096 бит. Серверный и клиентский сертификат обычно истекают после одного года, поэтому мы можем безопасно использовать 2048 бит.

Хотя 4096 бит немного больше безопаснее, чем 2048 бит, он замедляет TLS хэндшейки и значительно увеличивает нагрузку на процессор во время хэндшейков. По этой причине, большинство веб-сайтов используют 2048-битные пары ключей.

Если вы создаете криптографическую пару для использования веб-сервером (к примеру, Apache), вам потребуется вводить пароль каждый раз, когда вы перезапускаете веб-сервер. Вы можете указать опцию –aes256 для создания ключа без пароля.

```
# cd /root/ca
# openssl genrsa -aes256 \
```

-out intermediate/private/www.example.com.key.pem 2048

chmod 400 intermediate/private/www.example.com.key.pem

Создание сертификата

Используем частный ключ для создания запроса на подписывание сертификата (CSR). Детали в CSR не обязательно должны совпадать с промежуточным центром сертификации. Для серверных сертификатов, Общее Имя (Common Name) должно быть полным доменным именем (fully qualified domain name — FQDN) (к примеру, www.example.com), в то время как для клиентского сертификата оно должно быть любым уникальным идентификатором (к примеру, e-mail адресом). Common Name не может быть таким же, как любой корневой или промежуточный сертификат.

```
# cd /root/ca
# openssl req -config intermediate/openssl.cnf \
-key intermediate/private/www.example.com.key.pem \
-new -sha256 -out intermediate/csr/www.example.com.csr.pem
Enter pass phrase for www.example.com.key.pem: secretpassword
You are about to be asked to enter information that will be incorporated
into your certificate request.
---
Country Name (2 letter code) [XX]:US
State or Province Name []:California
Locality Name []:Mountain View
Organization Name []:Alice Ltd
Organizational Unit Name []:Alice Ltd Web Services
Common Name []:www.example.com
Email Address []:
```

Для создания сертификата воспользуемся промежуточным центром сертификации для подписи CSR. Если сертификат будет использоваться на сервере, используйте расширение server_cert. Если сертификат будет использоваться для аутентификации клиентов, то используйте расширение

usr_cert. Сертификаты обычно даются сроком на один год, на центре сертификации обычно дают несколько дней для удобства.

```
# cd /root/ca
# openssl ca -config intermediate/openssl.cnf \
-extensions server_cert -days 375 -notext -md sha256 \
-in intermediate/csr/www.example.com.csr.pem \
-out intermediate/certs/www.example.com.cert.pem
# chmod 444 intermediate/certs/www.example.com.cert.pem
```

Файл intermediate/index.txt должен содержать строку, которая относится к этому сертификату

```
V 160420124233Z 1000 unknown ... /CN=www.example.com
```

Проверка сертификата

```
# openssl x509 -noout -text \
-in intermediate/certs/www.example.com.cert.pem
```

Issuer сертификата наш промежуточный центр сертификации. Subject относится к самому сертификату.

```
Signature Algorithm: sha256WithRSAEncryption
Issuer: C=GB, ST=England,
O=Alice Ltd, OU=Alice Ltd Certificate Authority,
CN=Alice Ltd Intermediate CA
Validity
Not Before: Apr 11 12:42:33 2015 GMT
Not After : Apr 20 12:42:33 2016 GMT
Subject: C=US, ST=California, L=Mountain View,
O=Alice Ltd, OU=Alice Ltd Web Services,
CN=www.example.com
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
Public-Key: (2048 bit)
```

Вывод также покажет расширения X509V3. При создании сертификата можно использовать расширения server_cert или usr_cert. Варианты о соответствующем разделе конфигурации будут отражаться в выводе.

```
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Cert Type:
SSL Server
Netscape Comment:
OpenSSL Generated Server Certificate
X509v3 Subject Key Identifier:
B1:B8:88:48:64:B7:45:52:21:CC:35:37:9E:24:50:EE:AD:58:02:B5
X509v3 Authority Key Identifier:
keyid:69:E8:EC:54:7F:25:23:60:E5:B6:E7:72:61:F1:D4:B9:21:D4:45:E9
DirName:/C=GB/ST=England/O=Alice Ltd/OU=Alice Ltd Certificate Authority/CN=Alice Ltd Root CA
serial:10:00
X509v3 Key Usage: critical
Digital Signature, Key Encipherment
X509v3 Extended Key Usage:
TLS Web Server Authentication
```

Используя файл цепочки сертификатов центра сертификации, который мы создали ранее (ca-chain.cert.pem) для проверки того, что новый сертификат имеет действительную цепочку доверия.

```
# openssl verify -CAfile intermediate/certs/ca-chain.cert.pem \
intermediate/certs/www.example.com.cert.pem
www.example.com.cert.pem: OK
```

Установка сертификата

Теперь можно установить новый сертификат на сервере, или распространить сертификат на клиентов. При установке на серверное приложение (к примеру, Apache), понадобятся следующие файлы:

- ca-chain.cert.pem
- example.com.key.pem
- example.com.cert.pem

Если вы подписывали CSR от стороннего лица, у вас нет доступа до их частного ключа, и вы должны им дать только файл цепочки ca-chain.cert.pem и сертификат www.example.com.cert.pem.

Задание

Создание центра сертификации, поддерживающего список отозванных сертификатов, с использованием библиотеки OpenSSL.

Контрольные вопросы

1. Центра сертификации, поддерживающего список отозванных сертификатов?
2. Как создать сертификаты SSL (TLS) для сайтов?
3. Где хранятся корневые сертификаты Центров Сертификации (CA) в операционной системе?
4. Как создать Корневой Центр Сертификации (Root CA)?

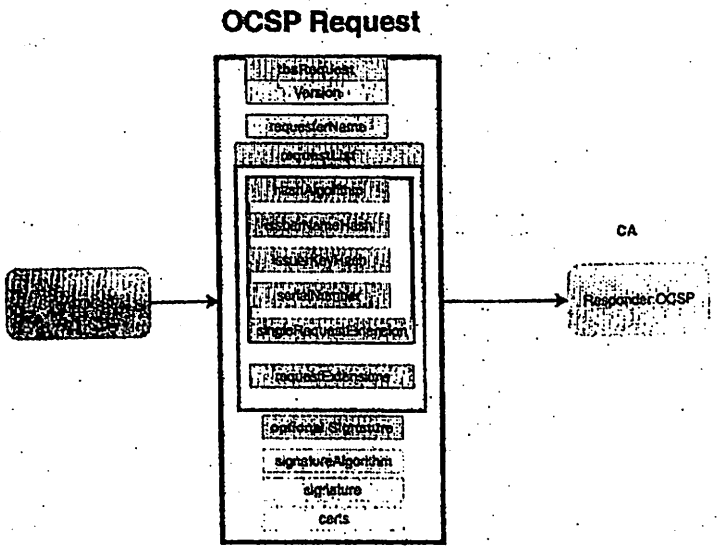
Практическая работа №13

Тема: Создание центра сертификации с поддержкой протокол OCSP применением библиотека OpenSSL.

Цель работы: Приобрести теоретические и практические навыки о создание центра сертификации с поддержкой протокол OCSP применением библиотека OpenSSL.

Теоретическая часть

Протокол состояния сетевого сертификата (Online Certificate Status Protocol - OCSP) - это интернет-протокол, используемый для получения статуса отзыва цифрового сертификата X.509. Механизм протокола описан в RFC 6960 и является одним из стандартов de-facto Интернета. Он был создан в качестве альтернативы спискам отзыва сертификатов (CRL), в частности для решения некоторых проблем, связанных с использованием CRL в инфраструктуре открытых ключей (PKI). Сообщения OCSP кодируются в ASN.1 и обычно передаются по HTTP. Диалоги «запрос/ответ» этих сообщений позволяют называть сервера OCSP ответчиками OCSP.



Сравнение с CRL

- Ответ OCSP содержит меньше данных, чем типичный список отзыва сертификатов (CRL), это снижает нагрузку на сетевые и клиентские ресурсы.

- Так как ответ OCSP содержит меньше данных для анализа и обработки, клиентские библиотеки для обработки менее сложны, чем те, которые обрабатывают CRL.

- OCSP отвечает на запрос, что определенный сетевой узел использует определенный сертификат в определенный период времени.

- Эта информация является открытой, следовательно, OCSP не требует шифрования, поэтому если другие стороны перехватят эту информацию, то это безопасно. Однако возможная подмена ответа является уязвимостью.

Ответчик OCSP (обычно сервер Центра сертификации) возвращает подписанный ответ со статусом сертификата, указанного в запросе: *годный*, *отозван* или *статус неизвестен*. Если он не может обработать запрос, он может вернуть код ошибки.

Формат запроса OCSP поддерживает дополнительные расширения. Это позволяет выполнить дополнительную настройку для конкретной схемы PKI.

OCSP может быть уязвим для атак повторного воспроизведения - replay-атак, где подписанный, валидный ответ захвачен злонамеренным посредником и воспроизведен клиенту позже после того, как предметный сертификат, возможно, был отозван. OCSP позволяет включить одноразовый код в запрос - nonce, который может быть включен в соответствующий ответ. Из-за высокой нагрузки большинство респондентов OCSP не используют расширение nonce для создания различных ответов для каждого запроса, вместо этого используя предварительно подписанные ответы с периодом действия в несколько дней. Таким образом, повторная атака представляет собой серьезную угрозу для систем валидации.

OCSP может поддерживать более одного уровня CA. Запросы OCSP могут быть связаны между одноранговыми ответчиками для запроса выдающего ЦС, подходящего для сертификата субъекта, с ответчиками,

проверяющими ответы друг друга на корневой ЦС, используя свои собственные запросы OCSP.

Ответчик OCSP может запрашивать информацию об отзыве с помощью серверов проверки делегированного пути (DPV). OCSP сама по себе не выполняет DPV поставляемых сертификатов.

Ключ, подписывающий ответ, не обязательно должен быть тем же ключом, который подписал сертификат. Эмитент сертификата может делегировать другой орган, чтобы быть ответчиком OCSP. В данном случае, ответчик сертификата (тот, который используется для подписи ответ) должен выдать эмитенту сертификата в ответ на запрос, определенное расширение, в котором содержится указание на него в качестве подписи сервиса OCSP (точнее, расширенное использование ключа продления с идентификаторами:

OID {iso(1) identified-organization(3) dod(6) internet(1) security(5) mechanisms(5) pkix(7) keyPurpose(3) ocspSigning(9)})

Практическая часть

Тестирование OCSP с Openssl

Я работал над реализацией, которая использует этот сшитый ответ OCSP. Вариант использования заключался в том, что подключенное устройство отправляет запрос на сервер через TLS. Устройство предоставляет клиентский сертификат для аутентификации на сервере. Сервер проверяет, а затем возвращает свой сертификат и скрепленный ответ OCSP для аутентификации клиента.

Мы столкнулись с проблемами при сшивании, и нам пришлось проверить результат. Для этой цели я показываю запрос/ответ, который не включает клиентские сертификаты. Это просто делает дискуссию немного проще.

Чтобы работать над этим аспектом, я начал использовать Openssl, и вот шаги для его достижения:

Шаг 1: Получите сертификат сервера

Сначала сделайте запрос на получение сертификата сервера. При использовании openssl s_client -connectкоманды это материал между -----BEGIN CERTIFICATE-----и -----END CERTIFICATE-----. Я использую www.akamai.com в качестве сервера.

```
openssl s_client -connect www.akamai.com:443 < /dev/null 2>&1 | sed -n '/-----BEGIN/,/-----END/p' > certificate.pem
```

Сертификат сервера сохраняется как *certificate.pem* .

Шаг 2: Получите промежуточный сертификат

Обычно ЦС не подписывает сертификат напрямую. Они используют посредников, и нам нужно, чтобы команда openssl работала. Итак, сделайте запрос, чтобы получить всех посредников.

Чтобы просмотреть список промежуточных сертификатов, используйте следующую команду.

```
openssl s_client -showcerts -connect www.akamai.com:443 < /dev/null 2>&1 | sed -n '/-----BEGIN/,/-----END/p'
```

Самый первый сертификат — это сертификат сервера, который мы сохранили на шаге 2. Все сертификаты ниже него скопируйте и сохраните в файл с именем *chain.pem* .

Шаг 3. Получите ответчик OCSP для сертификата сервера

Следующим шагом является получение информации об ответчике OCSP. Есть два способа сделать это:

Ответчик OCSP с командой

Мы можем использовать сертификат сервера certificate.pem *и* запустить команду, чтобы извлечь только поле респондента OCSP:

```
$ openssl x509 -noout -ocsp_uri -in certificate.pem
http://ss.symcd.com
```

Итак, здесь http://ss.symcd.com — ответчик OCSP.

Ответчик OCSP при проверке

Мы можем использовать команду openssl для печати всей информации о сертификате сервера с помощью этой команды:

```
openssl x509 -text -noout -in certificate.pem
```

В ответе найдите раздел с названием « *Доступ к авторитетной информации* » . Это будет содержать URL-адрес ответчика OCSP. В данном случае вот что я вижу:

```
Authority Information Access:
        OCSP - URI:http://ss.symcd.com
        CA Issuers - URI:http://ss.symcb.com/ss.crt
```

Шаг 4. Сделайте запрос OCSP

Теперь, когда у нас есть сертификат сервера, цепочка сертификатов ЦС и URL-адрес ответчика OCSP, мы можем сделать фактический проверочный вызов.

```
openssl ocsp -issuer chain.pem -cert certificate.pem -text -url http://ss.symcd.com
```

Вот соответствующая часть ответа.

```
OCSP Request Data:
  Version: 1 (0x0)
  Requestor List:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash:
D1B1648B8C9F0DD16BA38ACD2B5017D5F9CFC064
      Issuer Key Hash:
5F60CF619055DF8443148A602AB2F57AF44318EF
      Serial Number: 7E34F9C3D2D21D999B668CC6C80EA4A8
  Request Extensions:
```

```
        OCSP Nonce:
            0410AE9A77D9FBDE8F7EC1F0C3305183BAEC
    OCSP Response Data:
        OCSP Response Status: successful (0x0)
        Response Type: Basic OCSP Response
        Version: 1 (0x0)
        Responder Id: A2117BB82AC97305A9CCC170F80F82442BBF509A
        Produced At: Sep 12 16:58:10 2017 GMT
        Responses:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash:
D1B1648B8C9F0DD16BA38ACD2B5017D5F9CFC064
          Issuer Key Hash: 5F60CF61905CDF8443148A602AB2F57AF44318EF
          Serial Number: 7E34F9C3D2D21D999B668CC6C80EA4A8
        Cert Status: good
        This Update: Sep 12 16:58:10 2017 GMT
        Next Update: Sep 19 16:58:10 2017 GMT
```

Самая важная часть — **Cert Status: good** line. Это указывает на то, что все кошерно и клиент может доверять сертификату. Другая интересная часть — это подробности ***Next Update*** . Это указывает на то, что ответ сшивания OCSP может быть кэширован до определенного времени, чтобы мы не перегружали ответчик OCSP.

Обработка новых валидаторов OCSP

Многие новые конечные точки проверки OCPS, например ocsp2.globalsign.com, не поддерживают запросы HTTP 1.0. OCSP по умолчанию использует HTTP 1.0, который не требует Hostзаголовка. Когда это произойдет, вы можете увидеть такую 400 ошибку:

```
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
      Certificate ID:
        Hash Algorithm: sha1
        Issuer Name Hash:
12EADF46CC0880387360B65A691601CC0CB5E9E2
        Issuer Key Hash:
A92B87E1CE24473B1BBFCF853702559D0D9458E6
        Serial Number: 5625521AFA513B6D970FFAC1
    Request Extensions:
      OCSP Nonce:
        0410C43AA490D8782F80C2D0751C417B6486
Error querying OCSP responder
4732960364:error:27FFF072:OCSP routines:CRYPTO_internal:server
response
error:/AppleInternal/BuildRoot/Library/Caches/com.apple.xbs/Sources/libressl/libr
essl-47.120.1/libressl-2.8/crypto/ocsp/ocsp_ht.c:251:Code=400,Reason=Bad
Request
```

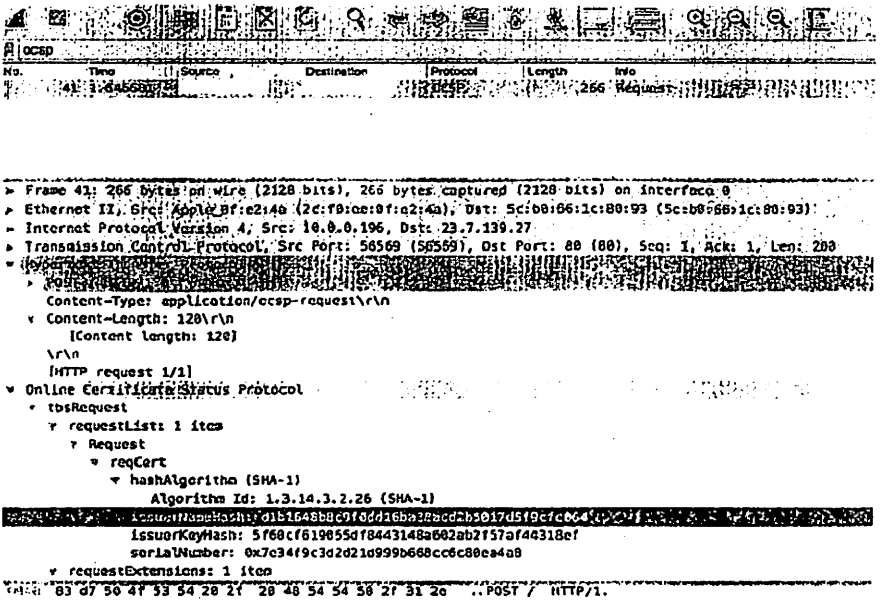
Чтобы исправить это несоответствие, нам нужно добавить заголовок Host. Этого можно добиться, добавив опцию -header, за которой следует информация заголовка.

Вот пример команды:

```
openssl ocsp -issuer chain.pem -cert certificate.pem -text -url
http://ocsp2.globalsign.com/cloudsslsha2g3 -header "HOST"
"ocsp2.globalsign.com"
```

Благодаря объяснению Джима Картера за помощь в выявлении и устранении проблемы.

Мне было любопытно посмотреть, что на самом деле происходит во время запроса OCSP, и я провел трассировку wireshark. Когда мы делаем запрос OCSP, он отправляется как HTTP POST. В этом случае заголовки выглядели так:



POST / HTTP/1.0

Content-Type: application/ocsp-request

Content-Length: 120

0v0t0M0K0I0 ..+.....d...

.k...+P.....d.._`.a.U..C..`*..z.C....~4.....f.....#0!0.. +.....0.....w....~...0Q...

HTTP/1.0 200 OK

Server: nginx/1.10.2

Content-Type: application/ocsp-response

Content-Length: 1609

content-transfer-encoding: binary

Cache-Control: max-age=329052, public, no-transform, must-revalidate

Last-Modified: Tue, 12 Sep 2017 16:58:10 GMT

Expires: Tue, 19 Sep 2017 16:58:10 GMT

Date: Fri, 15 Sep 2017 21:33:58 GMT

Connection: close

Если вы заметили, «Cache-Control: max-age=329052» будет равнозначно указанию клиенту кэшировать ответ до периода времени « *Следующее обновление* » . Таким образом, запрос POST может быть закэширован.

Задание

Создание центра сертификации с поддержкой протокол OCSP применением библиотека OpenSSL

Контрольные вопросы

1. Объясните принцип работа OCSP?
2. Объясните принцип работа центра сертификации?

Практическая работа №14

Тема: Использование сетевого доверия для распространения сертификатов

Цель задачи: Рассмотреть понятие и принципы работы схема сети доверия. Рассмотреть подхода: PGP (Pretty Good Privacy) и GnuPG

Сеть доверия - Web of trust

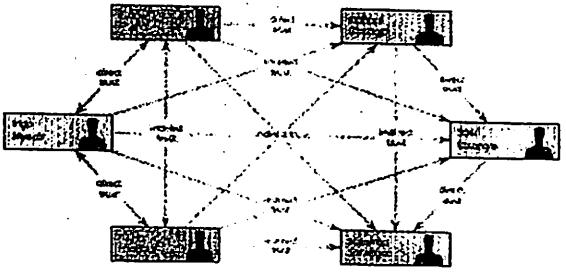


Рис. 14.1-Механизм аутентификации криптографических ключей сети доверия

В криптографии, сеть доверия - это концепция, используемая в PGP, GnuPG и других OpenPGP -совместимых системах для установления подлинность привязки между открытым ключом и его владельцем. Его децентрализованная модель доверия является альтернативой централизованной модели доверия инфраструктуры открытого ключа (PKI), которая опирается исключительно на центр сертификации (или иерархию таких). Как и в компьютерных сетях, существует множество независимых сетей доверия, и любой пользователь (через свой сертификат идентификации) может быть частью и связующим звеном между несколькими сетями.

Концепция сети доверия была впервые предложена создателем PGP Филом Циммерманном в 1992 году в руководстве для PGP версии 2.0:

Со временем вы будете накапливать ключи от других людей, которых вы, возможно, захотите назначить доверенными представителями. Все остальные выберут своих доверенных представителей. И каждый будет постепенно накапливать и распространять со своим ключом коллекцию

удостоверяющих подписей от других людей, ожидая, что любой, кто ее получит, будет доверять хотя бы одной или двум подписям. Это приведет к появлению децентрализованной отказоустойчивой сети доверия для всех открытых ключей.

Работа в сети доверия

Все реализации, совместимые с OpenPGP, включают схему проверки сертификатов , чтобы помочь в этом; его деятельность была названа сетью доверия. Сертификаты удостоверения OpenPGP (которые включают один или несколько открытых ключей вместе с информацией о владельце) могут быть подписаны цифровой подписью другими пользователями, которые этим действием подтверждают связь этого открытого ключа с лицом или организацией, указанными в сертификате. Это обычно делается на сторонах, подписывающих ключи..

Реализации, совместимые с OpenPGP, также включают схему подсчета голосов, которая может использоваться для определения того, какой ассоциации открытого ключа и владельца будет доверять пользователь при использовании PGP. Например, если три частично доверенных индоссатора поручились за сертификат (и, следовательно, включенный в него открытый ключ - владелец привязка), или если это сделал один полностью доверенный индоссант, связь между владельцем и открытым ключом в этом сертификат будет считаться правильным. Параметры настраиваются пользователем (например, без партиалов вообще или, возможно, с шестью частями), и при желании их можно полностью обойти.

Схема является гибкой, в отличие от большинства проектов инфраструктуры открытого ключа, и оставляет решение о доверии в руках отдельных пользователей. Он несовершенен и требует как осторожности, так и разумного контроля со стороны пользователей. По сути, все конструкции PKI менее гибки и требуют от пользователей подтверждения доверия сгенерированных PKI сертификатов, подписанных центром сертификации (CA).

Есть два ключа, относящихся к человеку: открытый ключ, которым открыто делятся, и закрытый ключ, который удерживает владелец. Закрытый ключ владельца расшифрует любую информацию, зашифрованную его открытым ключом. В сети доверия у каждого пользователя есть кольцо с открытыми ключами группы людей.

Пользователи шифруют свою информацию с помощью открытого ключа получателя, и только закрытый ключ получателя расшифрует ее. Затем каждый пользователь подписывает информацию цифровой подписью с помощью своего закрытого ключа, поэтому, когда получатель проверяет ее по собственному открытому ключу пользователя, он может подтвердить, что это именно тот пользователь. Это гарантирует, что информация поступила от конкретного пользователя и не была изменена, и только предполагаемый получатель может прочитать информацию (потому что только он знает свой закрытый ключ).

В отличие от типичного PKI

В отличие от WOT, типичный X.509 PKI позволяет подписывать каждый сертификат одной стороной: центр сертификации (CA). Сертификат CA может быть подписан другим CA, вплоть до «самоподписанного» корневого сертификата. Корневые сертификаты должны быть доступны тем, кто использует сертификат CA более низкого уровня, и поэтому обычно широко распространяются. Например, они распространяются с такими приложениями, как браузеры и почтовые клиенты. Таким образом, веб-страницы, защищенные SSL / TLS, сообщения электронной почты и т. Д. Могут быть аутентифицированы без необходимости вручную устанавливать корневые сертификаты. Приложения обычно включают в себя более ста корневых сертификатов от десятков PKI, таким образом по умолчанию обеспечивая доверие по всей иерархии сертификатов, которые ведут к ним.

WOT поддерживает децентрализацию якорей доверия, чтобы предотвратить нарушение единой точки отказа в иерархии CA. Заметным проектом, который использует WOT против PKI для обеспечения основы для

аутентификации в других областях Интернета, являются утилиты Monkeysphere.

WOT (Web of Trust) — бесплатная надстройка к браузеру, которая предупреждает интернет-пользователя во время поиска информации или совершения покупок о сайтах с низкой репутацией. WOT совместим с такими браузерами как Internet Explorer, Mozilla Firefox, Opera (начиная с 11 версии при помощи расширения), Google Chrome и Safari.

WOT создан на основе сообщества и показывает уровень доверия к тому или иному сайту. Репутация сайта зависит от оценок, выставленных его посетителями — пользователями WOT. Рейтинги постоянно обновляются миллионами пользователей WOT-сообщества, а также многочисленными проверенными ресурсами (например, списки фишинговых сайтов).

После загрузки надстройки на панели инструментов браузера появится логотип WOT. При каждом переходе Интернет-пользователя на новый сайт цвет логотипа будет изменяться в зависимости от репутации данного сайта.

Задание

Проверьте сертификат используя сетевого доверия.

Контрольные вопросы

1. Настройка проверки подлинности на основе сертификатов в лесах без доверия для веб-сервера как происходит?
2. Объясните процесс работа в сети доверия?

Практическая работа №15

Тема: Перехват и расшифровка трафика. Определение типа шифрования

Цель задачи: Научиться собирать сетевой трафик с помощью программы Wireshark. Научиться фильтровать собранный трафик, находить и просматривать соединения. Извлекать передаваемые без шифрования файлы и определить использованные алгоритмы шифрования.

Определение уязвимых устройств и веб-сервисов на вашем целевом беспроводном маршрутизаторе может стать сложной задачей, так как хакеры способны провести атаку на вашу беспроводную сеть, не оставив следов в ваших лог-файлах, а также других признаков своей активности. В этом материале мы расскажем про способ организовать скрытый перехват, дешифровку и анализ WiFi-активности без какого-либо подключения к беспроводной сети!

Хакеры, в попытках взломать защиту на беспроводных маршрутизаторах, могут использовать самые разнообразные методы для получения доступа к вашей конфиденциальной информации: от ресурсоемких атак грубой силы, осуществляемых с помощью различных программ по подбору простых «словарных» паролей, до изящных схем социальной инженерии, таких как фишинг Wi-Fi-паролей путем блокировки соединения и создания поддельных точек доступа, против которых бессильны даже самые надежные пароли. Как только учетные данные Wi-Fi будут получены, у злоумышленников появляются огромные возможности по скрытому захвату и анализу сетевого трафика скомпрометированной беспроводной сети, с применением различных методологий и специализированного инструментария для выявления и дальнейшего использования личной информации пользователей, передаваемой по этой сети.

Сканеры портов будут создавать огромное количество шума в беспроводных сетях. Атаки «человек посередине» могут быть слишком агрессивными, что может насторожить пользователей и предупредить

сетевого администратора о присутствии хакера. Кроме того, маршрутизаторы сохраняют в лог-файлы информацию о каждом устройстве, подключаемом к сети. Таким образом, каждое злонамеренное действие, выполняемое при подключении к скомпрометированному Wi-Fi-роутеру, рано или поздно может быть обнаружено, и защита сети восстановлена.

Однако, злоумышленникам, после получения доступа к учетным данным Wi-Fi, вовсе не обязательно подключаться к беспроводной сети (а, значит, оставлять следы своего присутствия), чтобы продолжить развивать свою атаку.

Основные принципы реализации скрытой атаки на WiFi сеть

В рамках данного практического руководства мы детально разберемся с тем, как злоумышленники перехватывают сетевые пакеты при их беспроводной передаче с и на Wi-Fi-роутер с помощью утилиты Airodump-ng, а затем практически в режиме реального времени дешифруют трафик WPA2 с помощью сниффера Wireshark.

Wireshark — самый популярный в мире и наиболее часто используемый анализатор сетевого трафика. Он позволяет пользователям буквально на микроскопическом уровне видеть то, что происходит в их сетях, и является де-факто основным инструментом сетевого аудита для коммерческих и некоммерческих организаций, а также государственных и образовательных учреждений.

Однако, прекрасная функциональность, встроенная в Wireshark, привлекает и злоумышленников, которые с помощью этого сетевого анализатора могут расшифровывать и просматривать в виде простого текста всю передаваемую по воздуху сетевую активность скомпрометированного беспроводного маршрутизатора.

Утилита Airodump-ng — это набор программ для обнаружения беспроводных сетей и перехвата передаваемого по этим сетям трафика, доступная для всех популярных операционных систем, включая UNIX, Linux, Mac OS X и Windows, которая, в том числе, способна работать на виртуальных

машинах и одноплатных компьютерах Raspberry Pi. Airodump-ng также, как и Wireshark, широко применяется для сетевого аудита, как, впрочем, и используется злоумышленниками.

Итак, давайте детально разберемся на основе пошагового практического примера, как злоумышленники могут реализовать скрытый взлом Wi-Fi-сети, чтобы в дальнейшем мы могли детально оценить, насколько уязвимы наши беспроводные сети. В рамках данного практического руководства мы будем использовать систему на базе Kali Linux для сбора, дешифровки и анализа беспроводных сетевых данных, генерируемых Wi-Fi-роутером, которым мы сами же и управляем.

Шаг 1 Определение целевого Wi-Fi-роутера.

Чтобы с помощью утилиты Airodump-ng включить режим мониторинга на беспроводном адаптере, используйте следующую команду:

«airmon-ng start wlan0»

Скриншот визуального представления запуска данной команды представлен на рисунке 15.1.

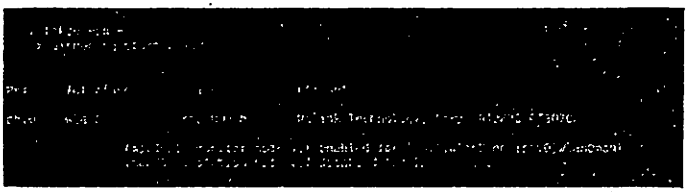


Рис 15.1. Демонстрация запуска команды «airmon-ng start wlan0» для включения режима мониторинга на беспроводном адаптере с помощью утилиты Airodump-ng.

Затем следует найти целевую беспроводную сеть (для нашего практического примера был использован маршрутизатор «Null Byte»). Для просмотра работающих в зоне вашей доступности сетей Wi-Fi используйте следующую команду:

«airodump-ng wlan0mon»

Скриншот визуального представления запуска данной команды представлен на рисунке 15.2.

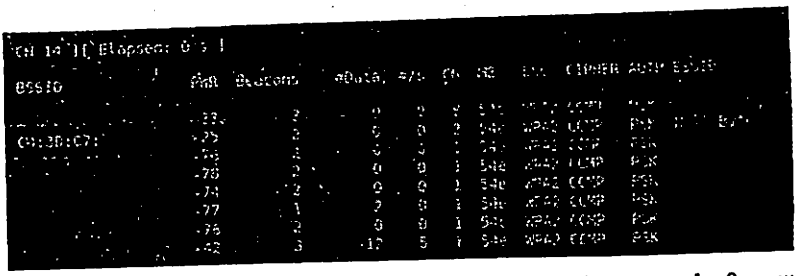


Рис. 15.2. Демонстрация запуска команды «airodump-ng wlan0mon» для просмотра доступных в окрестности сетей Wi-Fi с помощью утилиты Airodump-ng.

Обратите пристальное внимание на такие характеристики выбранной вами Wi-Fi-сети, как «BSSID», «CH», и «ESSID». Эта информация понадобиться для идентификации данных, передаваемых на беспроводный маршрутизатор, чтобы успешно реализовать процесс по их захвату и сбору.

Шаг 2. Захвата WiFi трафика

Чтобы начать сбор данных, относящихся к целевой WiFi сети, используйте следующую команду (обратите внимание, что в данной команде вам следует заменить выделенные курсивом и подчеркнутые части на те, которые актуальны для вашего конкретного случая):

«airodump-ng --bssid *TargetMACaddressHere* --essid *RouterNameHere* -c *ChannelNumber* -w *SaveDestination* wlan0mon»

где

- «TargetMACaddressHere» — это MAC-адрес целевого Wi-Fi-роутера (или значение «BSSID» из примера выше);
- «RouterNameHere» — это имя целевого Wi-Fi-роутера (или значение «ESSID» из примера выше);
- «ChannelNumber» — это номер канала целевого Wi-Fi-роутера (или значение «CH» из примера выше);
- «SaveDestination» — это выбранный для сохранения каталог и имя создаваемого файла.

Скриншот визуального представления с примером ввода данной команды представлен на рисунке 15.3.

Рис. 15.3. Пример ввода команды из утилиты Airodump-ng для старта сбора данных, относящихся к целевому Wi-Fi-роутеру «Null Byte».

Как вы можете видеть, в нашем примере мы указываем папку для сохранения собранных данных «/tmp» и имя файла «null_byte», используя аргумент «-w». Программа Airodump-ng автоматически добавит число в конец имени файла, поэтому на самом деле собранные данные в нашем практическом примере были сохранены в файле с именем «null_byte-01.cap». На рисунке 15.4 проиллюстрировано, что можно увидеть на запущенном терминале утилиты Airodump-ng.

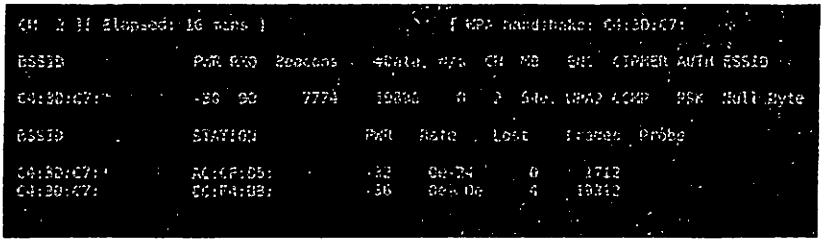


Рис. 15.4. Скриншот запущенного терминала утилиты Airodump-ng

Самой важной вещью, которую вы можете наблюдать на запущенном терминале утилиты Aireplay-ng, является рукопожатие WPA («WPA handshake»), расположенное в правом верхнем углу экрана (смотрите рисунок 4). «Рукопожатие» должно произойти, чтобы Wireshark позже смог расшифровать захваченный трафик Wi-Fi. Другими словами, после того, как вы начали сбор данных с помощью Aireplay-ng, вам необходимо переподключить подсоединенные к целевому Wi-Fi-роутеру устройства, трафик которых вы собираетесь проанализировать. К слову, для этих целей можно использовать встроенные возможности Aireplay-ng по принудительному отключению устройств от сети, чтобы инициировать

переподключение и, соответственно, запуск процесса рукопожатия WPA. Но этот шаг может причинить неудобство и вызвать подозрения у пользователей, подключенных к сети.

Пока терминал Airodump-ng запущен, данные будут накапливаться. Терминал Airodump-ng может работать несколько часов или даже дней. В нашем практическом примере сеанс Airodump-ng по сбору пакетов длился чуть более 15 минут. Длительность сеанса можно увидеть в верхнем левом углу терминала (смотрите рисунок 4).

Обратите внимание также на столбец «#Data» (смотрите рисунок 4). Это число собранных пакетов данных. Чем выше это число, тем больше вероятность того, что в них можно обнаружить конфиденциальные данные, которые злоумышленники могут использовать для дальнейшей компрометации цели или последующего разворачивания атаки внутрь корпоративной локальной сети (например, с помощью pivoting-техник).

Когда будет собрано достаточное количество данных, сеанс Airodump-ng можно остановить, нажав «Ctrl + C». Теперь в каталоге «/tmp» будет файл «null_byte-01.cap» (или как вы его назвали). Этот файл с расширением «.cap» можно будет открыть с помощью Wireshark.

Шаг 3. Установка последней версии сниффера Wireshark

По умолчанию Wireshark включен почти во все версии Kali Linux (https://www.kali.org/downloads/). Однако, есть несколько версий, которые не включают Wireshark, поэтому мы быстро расскажем, как установить Wireshark в Kali Linux.

Прежде всего, необходимо запустить из консоли команду обновления «apt-get», чтобы получить доступ к загрузке последней, протестированной и поддерживаемой (разработчиками Kali Linux) версии Wireshark. Для этого откройте терминал и введите следующую команду:

«sudo apt-get update»

Затем используйте следующую команду для установки Wireshark:

«sudo apt-get install wireshark»

К слову, вы можете использовать символ двойного амперсанда «&&» между двумя этими командами. Это даст указание терминалу сначала синхронизировать индекс пакета с его источниками (репозиториями Kali Linux). А затем (и только при условии успешного обновления) будет установлен Wireshark. Скриншот выполнения этих команд представлен на рисунке 15.5.

Рис. 15.5. Установка сниффера Wireshark в Kali Linux.

Шаг 4. Запуск инструментария Wireshark

Wireshark, после успешной установки, можно будет найти в категории «Sniffing & Spoofing» в меню «Applications». Чтобы запустить Wireshark, просто кликните на значок (смотрите рисунок 15.6).

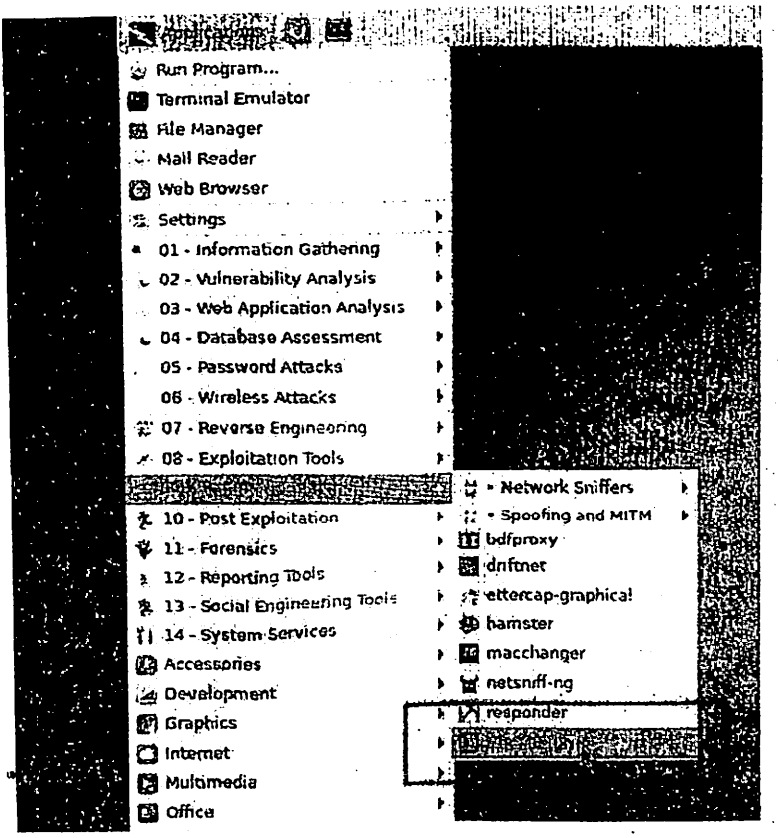


Рис. 15.6. Запуск инструментария Wireshark в Kali Linux.

Шаг 5. Настройка Wireshark для дешифровки трафика WiFi

Чтобы использовать Wireshark для дешифрования данных, находящихся в файле с расширением «.cap», нажмите кнопку «Edit» в верхней панели, затем выберите «Preferences», и разверните раскрывающееся меню «Protocols». Визуальный скриншот представлен на рисунке 15.7.

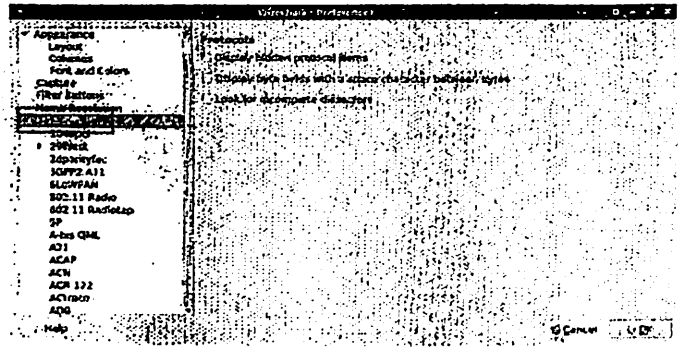


Рис. 15.7. Раскрывающееся меню «Protocols» в разделе «Preferences» в Wireshark.

После этого прокрутите вниз и выберите опцию «IEEE 802.11». Поле добавления, расположенное рядом со значением «Enable decryption», должно быть отмечено галочкой. Затем кликните на кнопке «Edit», чтобы добавить ключи дешифрования («Decryption keys») для конкретной сети Wi-Fi. Визуальный скриншот представлен на рисунке 15.8.

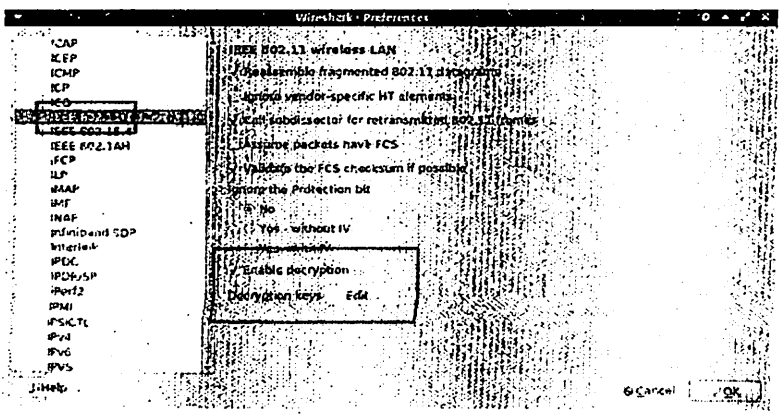


Рис. 15.8. Выбор ключей дешифрования для конкретной сети Wi-Fi в Wireshark

Откроется новое окно, в котором необходимо указать пароль и имя Wi-Fi-роутера. Для этого, прежде всего, выберите для значения «Key type» параметр «wpa-pwd». Этот тип ключа необходим для добавления пароля WPA

в виде обычного текста. При вводе учетных данных пароль и имя беспроводного маршрутизатора разделяется двоеточием (например, так — «password:router_name»). В нашем практическом примере для Wi-Fi-сети «Null Byte» был использован сверхнадежный пароль в виде длинной закодированной стоки, поэтому текст для значения «Key» получился следующим: «bWN2a25yMmNuM2N6amszbS5vbmlvbg==:Null Byte» (визуальный скриншот представлен на рисунке 9). Если же у вас, к примеру, пароль «Wonderfulboat555» к Wi-Fi-роутеру «NETGEAR72», то у вас должна получиться следующая строка: «Wonderfulboat555:NETGEAR72».

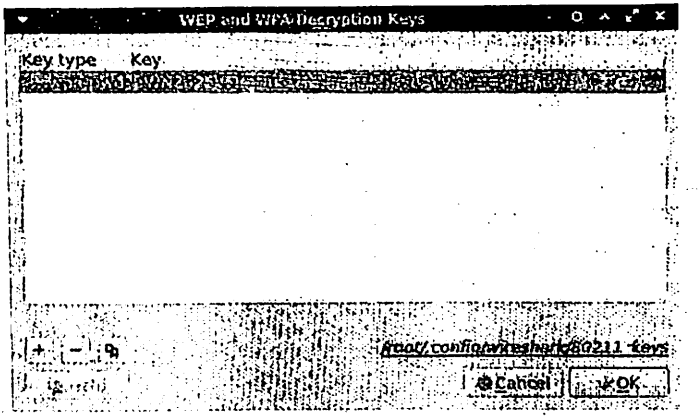


Рис. 15.9. Установленные ключи дешифрования для конкретной сети Wi-Fi в Wireshark

Нажмите «ОК», чтобы сохранить учетные данные. Теперь, после импортирования файла с расширением «.cap», в котором находятся данные перехваченного беспроводного трафика, Wireshark сможет автоматически расшифровывать данные, относящиеся к Wi-Fi-сети «Null Byte».

Шаг 6. Проведение подробного анализа пакетов (Perform Deep Packet Inspection, DPI)

Чтобы импортировать файл с захваченными и сохранными пакетами данных, нажмите на кнопку «File» на верхней панели, а затем выберете «Open». В нашем случае файл с расширением «.cap» был сохранен в каталоге

«/tmp»; выбираем его и нажимаем «Open». В зависимости от того, как долго утилита Airodump-ng собирала данные, Wireshark может потребоваться несколько минут для импорта и дешифрования всех данных. Визуальный скриншот этого действия представлен на рисунке 15.10.

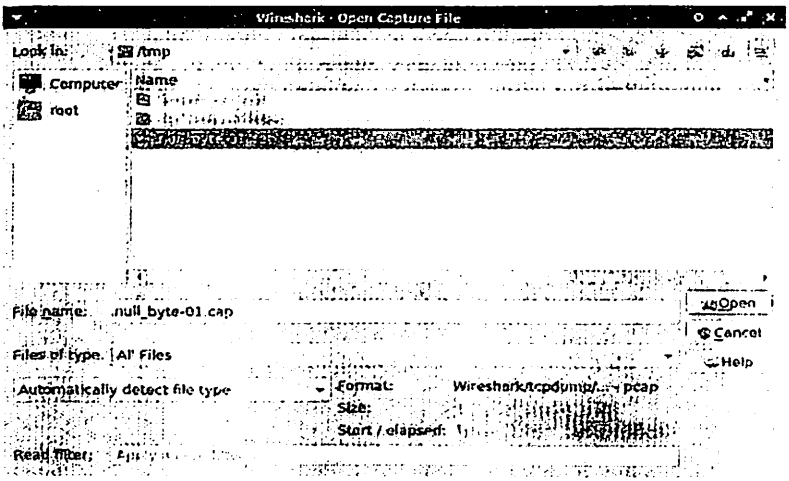


Рис. 15.10. Импортирование файла с расширением «.cap» в Wireshark.

После открытия файла с расширением «.cap» в Wireshark вы можете обнаружить тысячи строк необработанного веб-трафика. На первый взгляд это может выглядеть пугающим. К счастью, в Wireshark есть фильтры отображения («Display Filters»), которые можно использовать для сортировки и фильтрации ненужных пакетов. В сети вы можете найти огромное количество фильтров отображения, которые помогают пользователям провести тонкую настройку Wireshark, чтобы точечно идентифицировать актуальную и деликатную информацию. Далее, в рамках данного практического руководства, мы расскажем о нескольких наиболее эффективных фильтрах отображения, которых злоумышленники очень часто используют для проверки активности, происходящей в сети.

1. Поиск данных, содержащих запросы метода POST

Метод запроса POST, который поддерживается протоколом HTTP, часто используется при загрузке файла или отправке имен пользователей и паролей

на веб-сайт. Другими словами, когда кто-то, например, входит в Facebook или оставляет комментарий под статьей на Интернет-портале, это скорее всего делается с помощью запроса POST.

Поэтому велика вероятность того, что данные POST в файлах с расширением «.cap» будут содержать персональную и компрометирующую информацию. Так, злоумышленники могут найти имена пользователей, пароли, настоящие имена людей и их домашние адреса, адреса электронной почты, логи чата и многое другое.

Итак, чтобы использовать фильтр для получения данных, содержащих POST-запросы, введите приведенную ниже строку в поле ввода фильтра отображения Wireshark:

«http.request.method == "POST"»

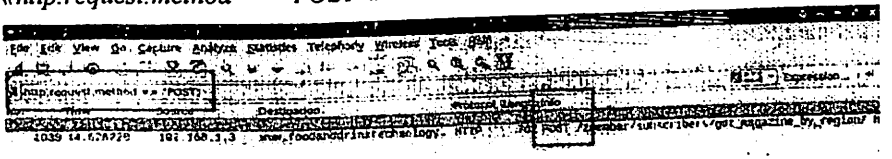


Рис. 15.11. Фильтр отображения для идентификации данных POST-запросов в Wireshark.

Для нашего практического руководства мы использовали имитацию пользовательской активности в виде подписки на электронную рассылку публикаций технологического сайта («http://www.foodanddrinktechnology.com/zmember/subscribers/register?lasturl= http://www.foodanddrinktechnology.com/»), случайно найденного на просторах Всемирной сети. Ведь запрос на получение уведомлений по электронной почте от своих любимых сайтов — распространенная в наши дни практика.

Если в файле с расширением «.cap» Wireshark найдет POST-запросы, то он отобразит их в информационном в столбце «Info» в виде строк, содержащие данные POST. Двойной щелчок по одной из этих строк приведет к появлению нового окна Wireshark, содержащего дополнительную информацию.

Прокрутите вниз и разверните раскрывающийся список «HTML Form», чтобы просмотреть данные.

В нашем случае при проверке данных, собранных только из этого единственного запроса POST, было обнаружено много информации, которая в реальных условиях могла бы принадлежать кому-то в сети. Собранные данные включали в себя имя, фамилию, место работы, а также пароль для регистрации на сервисе и адрес электронной почты, которые впоследствии могут быть использованы для фишинга или целевых хакерских атак.

Более детальная информация представлена на рисунке 15.12.

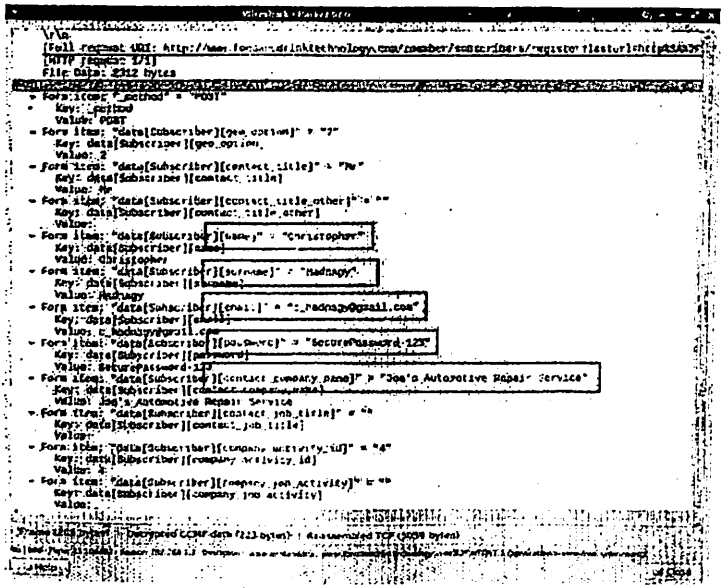


Рис. 15.12. Личная и конфиденциальная информация, найденная сниффером Wireshark при использовании фильтра отображения данных, содержащих POST-запросы

Как вы можете видеть, веб-сайт запросил введение формы пароля, который при обнаружении злоумышленники могут добавить в списки паролей и использовать его при проведении дальнейших атак грубой силы на сеть (с использованием метода перебора). Кроме того, люди часто используют

идентичные пароли для нескольких учетных записей. Вполне возможно, что найденный пароль может предоставить злоумышленникам доступ к электронному адресу Gmail, также указанному в данных POST.

Кроме того, в перехваченных данных было обнаружено название компании, где, предположительно, работает Кристофер Хаднаги. Эта информация может быть использована злоумышленником для дальнейшей атаки на этого человека с использованием техник социальной инженерии.

Еще немного пролистав вниз перехваченные и дешифрованные данные POST, можно найти еще больше информации (скриншот представлен на рисунке 13). К примеру, полный домашний адрес, почтовый индекс и номер телефона, которые также были включены в этот же единственный запрос POST. Эта информация сообщит злоумышленнику, где живет пользователь, и он сможет использовать телефонный номер в мошеннических целях, применяя техники социальной инженерии. Например, отправляя поддельные SMS-сообщения или использовать телефонный звонок для получения дополнительной информации о человеке и/или вымогательства.

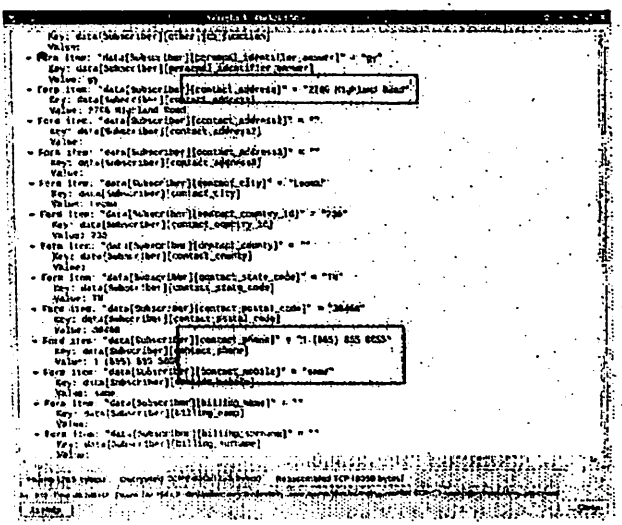


Рис. 15.13. Личная информация, найденная Wireshark при использовании фильтра отображения данных, содержащих POST-запросы.

Поиск данных, содержащих запросы метода GET

Метод запроса GET, который также поддерживается протоколом HTTP, часто используется для извлечения или загрузки данных с веб-серверов. Например, если кто-то просматривает чью-то учетную запись в Twitter, его браузер будет использовать GET-запрос для извлечения необходимой информации с веб-серверов twitter.com. Пристальная проверка файлов с расширением «.cap» (с перехваченным беспроводным трафиком) для GET-запросов не выявит имен пользователей или адресов электронной почты, но позволит злоумышленникам разработать полный профиль поведения пользователей в Интернете.

Чтобы использовать фильтр для получения данных, содержащих GET-запросы, введите приведенную ниже строку в поле ввода фильтра отображения Wireshark:

«http.request.method == "GET"»

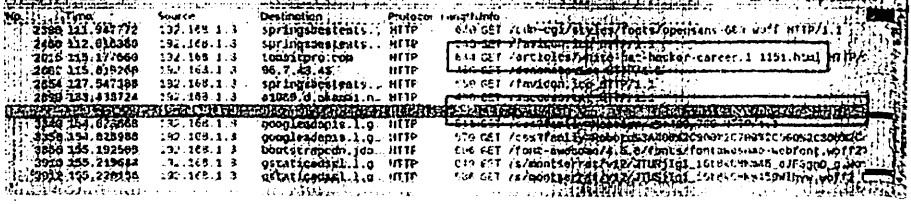


Рис. 15.14. Скриншот примера использования фильтра отображения для идентификации данных GET-запросов в Wireshark

Многие веб-сайты добавляют окончания «.html» или «.php» к концу своих URL-адресов (смотрите рисунок 15.14). Эта закономерность может быть использована в качестве индикатора веб-сайтов, просматриваемых кем-то в сети Wi-Fi.

Кроме того, для оптимизации поиска будет не лишним отсеять все GET-запросы, связанные с CSS (Cascading Style Sheets, каскадные таблицы стилей), для описания стилей и шрифтов, так как эти виды запросов не задаются

пользователями напрямую, а происходят в фоновом режиме, когда кто-то просматривает веб-страницы в Интернете. Чтобы использовать фильтр для получения данных, содержащих GET-запросы и одновременно отфильтровать содержимое CSS, введите приведенную ниже строку в поле ввода фильтра отображения Wireshark:

«http.request.method == "GET" && !(http.request.line matches "css")»

Как вы можете видеть, символ двойного амперсанда «&&» здесь буквально означает «и». Символ «!» (восклицательный знак) в данном случае используется как «нет». Таким образом, мы инструктируем Wireshark отображать только запросы GET, игнорируя при этом все строки HTTP-запросов, которые каким-либо образом соответствуют «css». Используя этот фильтр, вы отсеете часть бесполезной для вас информации, связанной со служебной информацией, используемой веб-ресурсами в фоновом режиме.

Кликнув на одну из найденных строк, мы сможем развернуть и просмотреть более детальные данные «Hypertext Transfer Protocol» («протокола передачи гипертекста» или просто HTTP) с более подробной информацией об активности пользователя.

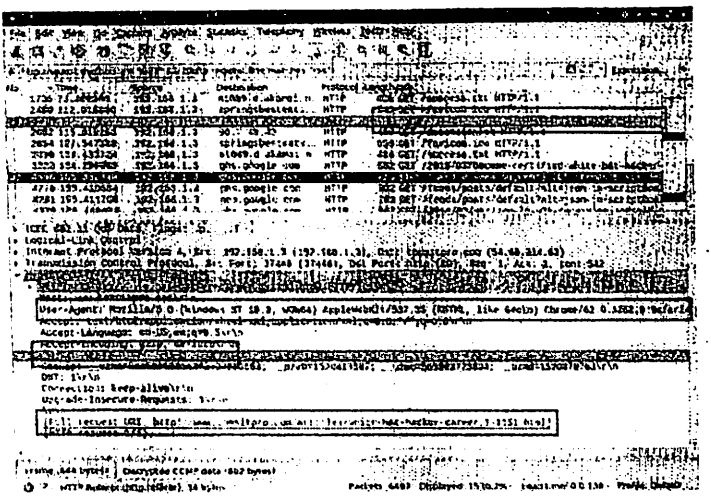


Рис. 15.15. Отображение в Wireshark подробной информацией об активности пользователя через данные «Hypertext Transfer Protocol»

Из информации, которую пересылает о себе клиентское приложение User-agent, мы можем идентифицировать, что наша жертва использует компьютер под управлением Windows и браузер Chrome. Эта информация очень ценна для злоумышленников, так они теперь смогут более целенаправленно выбирать «полезные нагрузки» (payload) при осуществлении атаки эксплоита для получения доступа к системе этого пользователя, учитывая специфические свойства и уязвимости используемой им версии операционной системы Windows.

Строка «Referer» укажет нам на веб-сайт, который пользователь, поведенческую модель которого мы анализируем в данный момент, просматривал непосредственно перед тем, как перешел по ссылке на статью на веб-сайте «tomsitpro.com». Так как это поисковый веб-сайт «duckduckgo.com», то практически наверняка это означает, что статью о карьере этичного хакера («white hat hacker career») пользователь нашел с помощью некого запроса в этой поисковой системе.

Что эта тривиальная и на первый взгляд «неопасная» информация может сказать злоумышленнику? Много. Прежде всего, использование поисковой системы DuckDuckGo вместо стандартной для большинства Google, может указывать на человека, который заботится о своей конфиденциальности, поскольку Google известен своими агрессивными политиками в области собирания и использования личных данных своих пользователей. А тот, кто заботится о конфиденциальности, может также интересоваться защитным программным обеспечением, таким как антивирусные программы. И это то, что хакеры также будут учитывать при создании «полезной нагрузки» для этого пользователя.

3. Поиск данных DNS-запросов

Для передачи зашифрованного интернет-трафика (HTTPS) по умолчанию используется TCP-порт 443. Поэтому для того, чтобы понять, какие веб-сайты просматривает выбранный пользователь, на первый взгляд было бы логично воспользоваться фильтром отображения «tcp.port == 443».

Но это обычно приводит к получению большого списка необработанных IP-адресов (в цифровом виде) в столбце назначения, что не очень удобно для быстрой идентификации доменов. Фактически, более эффективным способом для идентификации веб-сайтов, отправляющих и получающих зашифрованные данные, является фильтрация DNS-запросов.

Система доменных имен (Domain Name System, DNS) используется для преобразования имен веб-сайтов в машиночитаемые IP-адреса, такие как «https://144.76.198.94». Таким образом, когда мы посещаем, к примеру, такой домен, как «https://networkguru.ru», наш компьютер преобразует понятное человеку доменное имя в IP-адрес. Это происходит каждый раз, когда мы используем доменное имя для просмотра веб-сайтов, отправки электронной почты или общения онлайн.

Поиск DNS-запросов в файле с разрешением «.cap» (с перехваченным беспроводным трафиком) поможет злоумышленникам понять, какие веб-сайты часто посещают люди, подключенные к целевому маршрутизатору. Злоумышленники также смогут увидеть доменные имена, принадлежащие веб-сайтам, которые получают и отправляют зашифрованные данные на такие веб-сайты, как Facebook, Twitter и Google.

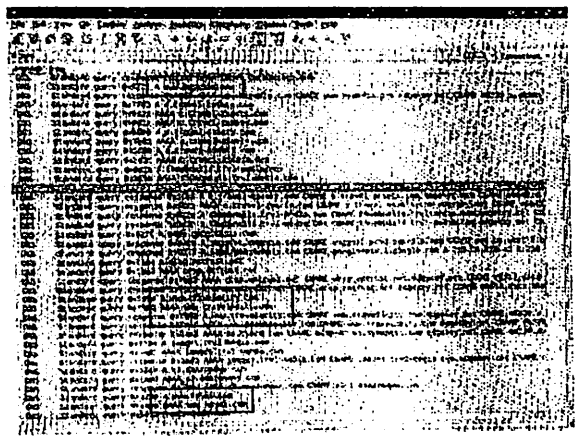


Рис. 15.16. Скриншот примера использования фильтра отображения для идентификации данных DNS-запросов в Wireshark

Чтобы использовать фильтр для получения данных, содержащих DNS-запросы, введите приведенную ниже строку в поле ввода фильтра отображения Wireshark:

«dns»

Анализ DNS-запросов может дать некоторую интересную информацию. К примеру, если вернуться к нашему практическому примеру, то мы можем увидеть (смотрите рисунок 15.16), что пользователь целевого маршрутизатора часто просматривает туристические сайты, такие как «expedia.com» и «kayak.com». И с большой долей вероятности это может означать, что пользователь скоро будет отсутствовать дома в течение длительного периода времени.

Сами данные зашифрованы, поэтому таким образом злоумышленники не смогут узнать ни время отъезда, ни место назначения. Но злоумышленники могут использовать информацию, полученную благодаря анализу DNS-запросов, для оттачивания методов социальной инженерии, направленных на этого пользователя, с целью получения от него личной и/или финансовой информации.

Например, если среди DNS-запросов был обнаружен домен веб-сайта одного из банков, то хакеры могли бы попытаться подделать электронное письмо от этого банка, в котором бы сообщалось, что только что произошла крупная транзакция по кредитной карте Expedia.

Задание

Установить wireshark, отследить и анализировать протокол SSL / TLS.

Контрольные вопросы

1. Что такое фильтрация трафика и зачем она нужна?

2. Какое программное обеспечение существует для фильтрации сетевого трафика?

3. Для фильтрации сетевого трафика какие свойства файлов считаются важными?

Содержание

Методические указания по разработке
и подготовке к изданию
учебных литератур

Рассмотрено и рекомендовано к печати
на заседании учебно-методического совета
ТУИТ им.Мухаммада ал-Хоразмий
202__ год ____ _____
протокол № ____

Составители: Б.М.Шамшиева
О.О.Турсунов
Ш.Ж.Хамидов
У.У.Тоджиакбарова

Рецензенты: О.П. Ахмедова
О.М. Алланов

Главный редактор: _____

Корректировщики: _____

Формат 60x84 1/16. Печ.лист 10.
Заказ №13. Тираж 10.
Отпечатано в «Редакционно издательском»
отделе при ТУИТ.
Ташкент ул. Амир Темур, 108.