В отчет по лабораторной работе необходимо внести:

* Номер и название работы;
* Цель работы;
* Теоретическую часть;
* Текст программы, таблицы, расчетные формулы, графики и т.д.
* Вывод.

Оглавление

[Лабораторная работа №1. Линейные конгруэнтные генераторы псевдослучайных последовательностей. 2](#_Toc186215912)

[Краткие теоретические сведения 2](#_Toc186215913)

[Выполнение работы 4](#_Toc186215914)

[Контрольные вопросы 4](#_Toc186215915)

[Лабораторная работа №2. Сеть Фейстеля 4](#_Toc186215916)

[Краткие теоретические сведения 4](#_Toc186215917)

[Выполнение работы 5](#_Toc186215918)

[Контрольные вопросы 6](#_Toc186215919)

[Лабораторная работа №3. Алгоритм DES 6](#_Toc186215920)

[Краткие теоретические сведения 6](#_Toc186215921)

[Выполнение работы 14](#_Toc186215922)

[Контрольные вопросы 15](#_Toc186215923)

[Лабораторная работа №4. Алгоритм RC4 15](#_Toc186215924)

[Краткие теоретические сведения 15](#_Toc186215925)

[Выполнение работы 16](#_Toc186215926)

[Контрольные вопросы 16](#_Toc186215927)

[Лабораторная работа №5. Ассиметричные алгоритмы шифрования данных: алгоритм RSA 17](#_Toc186215928)

[Краткие теоретические сведения 17](#_Toc186215929)

[6.3. Выполнение работы 18](#_Toc186215930)

[Контрольные вопросы 18](#_Toc186215931)

[Лабораторная работа №6. Комбинирование симметричных и асимметричных алгоритмов 18](#_Toc186215932)

[Краткие теоретические сведения 18](#_Toc186215933)

[Выполнение работы 19](#_Toc186215934)

[Контрольные вопросы 19](#_Toc186215935)

[Лабораторная работа №7. Алгоритм N-хэш 19](#_Toc186215936)

[Краткие теоретические сведения 19](#_Toc186215937)

[Выполнение работы 23](#_Toc186215938)

[Контрольные вопросы 24](#_Toc186215939)

[Лабораторная работа №8. Алгоритм SHA 24](#_Toc186215940)

[Краткие теоретические сведения 24](#_Toc186215941)

[Выполнение работы 26](#_Toc186215942)

[Контрольные вопросы 27](#_Toc186215943)

[Лабораторная работа №9. Программирование криптографических провайдеров на платформе .NET: Алгоритмы хэширования SHA-1 и MD5 27](#_Toc186215944)

[Краткие теоретические сведения 27](#_Toc186215945)

[Выполнение работы 30](#_Toc186215946)

[Контрольные вопросы 30](#_Toc186215947)

[Лабораторная работа №9. Программирование криптографических провайдеров на платформе .NET: Алгоритм цифровой подписи DSA 30](#_Toc186215948)

[Краткие теоретические сведения 30](#_Toc186215949)

[Выполнение работы 32](#_Toc186215950)

[Контрольные вопросы 32](#_Toc186215951)

# Лабораторная работа №1. Линейные конгруэнтные генераторы псевдослучайных последовательностей.

### Краткие теоретические сведения

Линейными конгруэнтными генераторами псевдослучайных последовательностей называются генераторы следующего вида:

Xn = (a\*Xn-1+b) mod m,

где Xn - n-й элемент последовательности, а Xn-1 - n-1-й элемент последовательности. Параметры a (множитель), b (инкремент) и m (модуль) - константы. Ключом для генератора служит значение X0.

Период такого генератора не превышает m. Если параметры a, b и m подобраны правильно, то генератор будет генератором с максимальным периодом (длиной) и его период будет равен m. Для этого, например, необходимо, что бы значение b было взаимно простым с m. В таблице приведены некоторые "хорошие" константы линейных конгруэнтных генераторов. Все они обеспечивают максимальный период генератора и, что наиболее важно, выполнение для таких генераторов спектрального теста на случайность для размерностей 2, 3, 4, 5 и 6. Таблица организована по максимальному произведению, которое не вызывает переполнения в слове указанной длины.

Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Переполняется при** | **a** | **b** | **m** |
| 220 | 106 | 1283 | 6075 |
| 221 | 211 | 1662 | 7875 |
| 222 | 421 | 1663 | 7875 |
| 223 | 430 | 2531 | 11979 |
|  | 936 | 1399 | 6655 |
|  | 1366 | 1283 | 6075 |
| 224 | 171 | 11213 | 53125 |
|  | 859 | 2531 | 11979 |
|  | 419 | 6173 | 29289 |
|  | 967 | 3041 | 14406 |
| 225 | 141 | 28411 | 134456 |
|  | 625 | 6571 | 31104 |
|  | 1541 | 2957 | 14000 |
|  | 1741 | 2731 | 12960 |
|  | 1291 | 4621 | 21870 |
|  | 205 | 29573 | 139968 |
| 226 | 421 | 17117 | 81000 |
|  | 1255 | 6173 | 29282 |
|  | 281 | 28411 | 134456 |
| 227 | 1093 | 18257 | 86436 |
|  | 421 | 54773 | 259200 |
|  | 1021 | 24631 | 116640 |
|  | 1021 | 25673 | 121500 |

Преимуществом линейных конгруэнтных генераторов является высокое быстродействие за счёт небольшого количества операций на один бит.

Однако, линейных конгруэнтные генераторы нельзя использовать в криптографии, т.к. они предсказуемы. Впервые такой генератор был взломан Джимом Ридсом (Jim Reeds), позднее Джоан Бояр (Joan Boyar) удалось также вскрыть квадратичные и кубические генераторы. Позднее идеи Бояр позволили разработать способы вскрытия любого полиномиального генератора, чем была доказана неприменимость таких генераторов в криптографии.

Тем не менее, линейные конгруэнтные генераторы остаются весьма полезными для других задач, например, в системах моделирования. Они эффективны и в большинстве используемых эмпирических тестах, демонстрируя хорошие статистические характеристики.

#### Объединение линейных конгруэнтных генераторов

Для увеличения длины периода и улучшения характеристик в некоторых статистических тестах были предприняты попытки объединения линейных конгруэнтных генераторов (при этом криптографическая стойкости их не повышается).

Так для 32-битовых компьютеров может быть использован следующий генератор псевдослучайных последовательностей, который объединяет два линейных конгруэнтных генератора с периодами 231-85 и 231-249. Период этого генератора равен произведению периодов объединяемых генераторов.

/\* "long" должно быть 32-битовым целым

static long s1 = 1;

static long s2 = 1;

/\* MODMULT (a, b, c, m, s) рассчитывает s\*b mod m при условии, что

m = a\*b+c и 0<=c<m

#define MODMULT(a, b, c, m, s) q = s/a; s = b\*(s-a\*q)-c\*q; if (s<0) s+=m;

/\* Возвращает действительное псевдослучайное значение в диапазоне (0,1) \*/ double CombLCG (void)

{ long q; long z;

MODMULT (53668, 40014, 12211, 2147483563L, s1) MODMULT (52774, 40692, 3791, 2147483399L, s2)

z = s1 - s2; if(z<1)

z += 2147483562; return z\*4.656613e-10; }

/\* Инициализирует CombLCG, вызывается один раз перед использованием \*/ void InitLCG(long InitS1, long InitS2)

{ s1 = InitS1; s2 = InitS2; }

Этот генератор работает при условии, что компьютер способен представить все целые числа в диапазоне от -231+85 до 231-249. Переменные s1 и s2 глобальные и содержат текущее состояние генератора. Перед вызовом CombLCG() необходимо инициализировать состояние генератора вызовом InitLCG() и передать начальные значения s1 и s2. Для s1 начальное значение берётся из диапазона [1, 2147483562], а для s2 - из диапазона [1, 2147483398].

Период генератора близок к 1018.

### Выполнение работы

Для выполнения работы в личном рабочем пространстве создать проект с именем Lab01 и перенести код примера.

Провести серию экспериментов (не менее 5), изменяя длину последовательности и начальные значения s1, s2. Для каждого эксперимента вычислить математическое ожидание и дисперсию.

#### Отчётные материалы

Преподавателю демонстрируется работающая программа и предоставляется печатный отчёт. Отчёт содержит код программы и таблицу с результатами экспериментов.

# Контрольные вопросы

1. Каковы особенности линейного конгруэнтного генератора?
2. Для чего объединяются линейные конгруэнтные генераторы?
3. Каковы преимущества линейного конгруэнтного генератора?
4. Как вычисляются математическое ожидание и дисперсия?

# Лабораторная работа №2. Сеть Фейстеля

### Краткие теоретические сведения

Рассмотрим случай, когда нужно зашифровать некоторую информацию, представленную в двоичном виде в компьютерной памяти (например, файл) или электронике, как последовательность нулей и единиц.

* Вся информация разбивается на блоки фиксированной длины. В случае, если длина входного блока меньше, чем размер, который шифруется заданным алгоритмом, то блок удлиняется каким-либо способом. Как правило длина блока является степенью двойки, например: 64 бита, 128 бит. Далее будем рассматривать операции происходящие только с одним блоком, так как с другими в процессе шифрования выполняются те же самые операции.
* Выбранный блок делится на два равных подблока — «левый» (L0) и «правый» (R0).
* «Левый блок» L0 видоизменяется функцией f(L0, K1) в зависимости от ключа K1, после чего он складывается по модулю 2 с «правым блоком» R0.
* Результат сложения присваивается новому левому подблоку L1, который будет половиной входных данных для следующего раунда, а «левый блок» L0 присваивается без изменений новому правому подблоку R1 (см. схему), который будет другой половиной.
* После чего операция повторяется N-1 раз, при этом при переходе от i-го к i+1-му этапу могут меняться ключи(Ki на Ki+1) по какому-либо правилу, где N — количество раундов в заданном алгоритме. [8]

**Расшифрование**

Расшифровка информации происходит так же, как и шифрование, с тем лишь исключением, что ключи идут в обратном порядке, т.е не от первого к N-ному, а от N-го к первому.

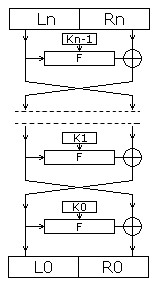
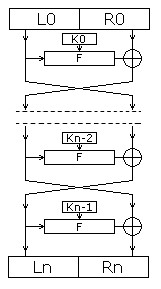


Рис.1. Схема работы сети Фейстеля

Шифрование Дешифрование

**Алгоритмическое описание**

* блок открытого текста делится на 2 равные части (L0,R0)
* в каждом раунде вычисляется ( — номер раунда)

Li=Ri-1 f(Li-1,Ki-1); Ri = Li-1,

где f— некоторая функция, а Ki-1 — ключ i-го раунда. Результатом выполнения n раундов является (Ln,Rn). Но обычно в n-ом раунде перестановка Ln и Rn не производится, что позволяет использовать ту же процедуру и для расшифрования, просто инвертировав порядок использования раундовой ключевой информации:

Li-1=Ri-1 f(Li,Ki-1); Ri-1 = Li.

Небольшим изменением можно добиться и полной идентичности процедур шифрования и дешифрования. Одно из преимуществ такой модели — обратимость алгоритма независимо от используемой функции f, и она может быть сколь угодно сложной.

### Выполнение работы

Реализовать приложение для шифрования на основе сетей Фейстеля, позволяющее выполнять следующие действия:

1. шифруемый текст должен храниться в одном файле, а ключ шифрования – в другом;
2. приложение должно позволять выбирать вид образующей функции: а) функция – единичная, т.е. (f(vi)=vi ) ;

б) функция имеет вид f(vi,x)=vix

3)зашифрованный текст должен сохраняться в файл;

4)расшифрованный текст должен сохраняться в файл.

##### Отчётные материалы

Преподавателю демонстрируется работающая программа и предоставляется печатный отчёт. Отчёт содержит код программы и результаты работы программы.

# Контрольные вопросы

1. Какие образующие функции можно использовать в сети Фейстеля?
2. Как происходит процесс шифрования?
3. Как происходит процесс дешифрования?
4. Как определяется величина блока разбиения?
5. Какое свойство функции XOR позволяет производить шифрование/дешифрование?

# Лабораторная работа №3. Алгоритм DES

### Краткие теоретические сведения

Основные достоинства алгоритма DES**:**

* используется только один ключ длиной 56 битов;
* зашифровав сообщение с помощью одного пакета, для расшифровки вы можете использовать любой другой;
* относительная простота алгоритма обеспечивает высокую скорость обработки информации;
* достаточно высокая стойкость алгоритма.

DES осуществляет шифрование 64-битовых блоков данных с помощью 56-битового ключа. Расшифрование в DES является операцией обратной шифрованию и выполняется путем повторения операций шифрования в обратной последовательности (несмотря на кажущуюся очевидность, так делается далеко не всегда. Позже мы рассмотрим шифры, в которых шифрование и расшифрование осуществляются по разным алгоритмам).

Процесс шифрования заключается в начальной перестановке битов 64битового блока, шестнадцати циклах шифрования и, наконец, обратной перестановки битов (рис.2).



Рис.2. Обобщенная схема шифрования в алгоритме DES

Необходимо сразу же отметить, что ВСЕ таблицы являются стандартными, а следовательно, должны включаться в вашу реализацию алгоритма в неизменном виде. Все перестановки и коды в таблицах подобраны разработчиками таким образом, чтобы максимально затруднить процесс расшифровки путем подбора ключа. Структура алгоритма DES приведена на рис.3.

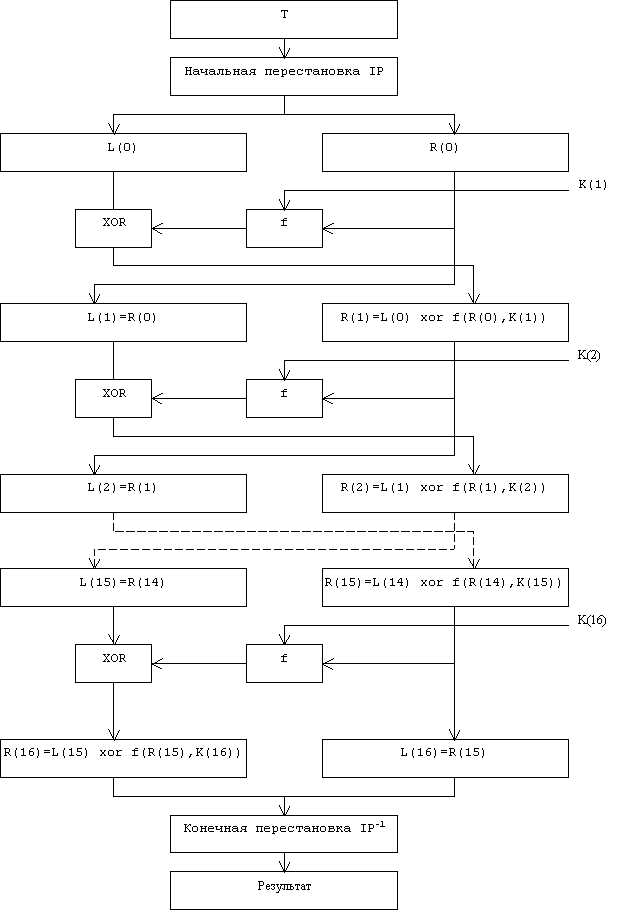


Рис.3. Структура алгоритма шифрования DES

Пусть из файла считан очередной 8-байтовый блок T, который преобразуется с помощью матрицы начальной перестановки IP (табл.1) следующим образом: бит 58 блока T становится битом 1, бит 50 - битом 2 и т.д., что даст в результате: T(0) = IP(T).

Полученная последовательность битов T(0) разделяется на две последовательности по 32 бита каждая: L(0) - левые или старшие биты, R(0) - правые или младшие биты.

Таблица 2. ***Матрица начальной перестановки IP***

58 50 42 34 26 18 10 02

60 52 44 36 28 20 12 04

62 54 46 38 30 22 14 06

64 56 48 40 32 24 16 08

57 49 41 33 25 17 09 01

59 51 43 35 27 19 11 03

61 53 45 37 29 21 13 05

63 55 47 39 31 23 15 07

Затем выполняется шифрование, состоящее из 16 итераций. Результат iй итерации описывается следующими формулами:

L(i) = R(i-1)

R(i) = L(i-1) xor f(R(i-1), K(i)) ,

где xor - операция ИСКЛЮЧАЮЩЕЕ ИЛИ.

Функция f называется функцией шифрования. Ее аргументы - это 32битовая последовательность R(i-1), полученная на (i-1)-ой итерации, и 48битовый ключ K(i), который является результатом преобразования 64битового ключа K. Подробно функция шифрования и алгоритм получения ключей К(i) описаны ниже.

На 16-й итерации получают последовательности R(16) и L(16) (без перестановки), которые конкатенируют в 64-битовую последовательность R(16)L(16).

Затем позиции битов этой последовательности переставляют в соответствии с матрицей IP-1 (табл.3).

Таблица 3. ***Матрица обратной перестановки IP***-1

40 08 48 16 56 24 64 32

39 07 47 15 55 23 63 31

38 06 46 14 54 22 62 30

37 05 45 13 53 21 61 29

36 04 44 12 52 20 60 28

35 03 43 11 51 19 59 27

34 02 42 10 50 18 58 26

33 01 41 09 49 17 57 25

Матрицы IP-1 и IP соотносятся следующим образом: значение 1-го элемента матрицы IP-1 равно 40, а значение 40-го элемента матрицы IP равно 1, значение 2-го элемента матрицы IP-1 равно 8, а значение 8-го элемента матрицы IP равно 2 и т.д.

Процесс расшифрования данных является инверсным по отношению к процессу шифрования. Все действия должны быть выполнены в обратном порядке. Это означает, что расшифровываемые данные сначала переставляются в соответствии с матрицей IP-1, а затем над последовательностью бит R(16)L(16) выполняются те же действия, что и в процессе шифрования, но в обратном порядке.

Итеративный процесс расшифрования может быть описан следующими формулами:

R(i-1) = L(i), i = 1, 2, ..., 16;

L(i-1) = R(i) xor f(L(i), K(i)), i = 1, 2, ..., 16 .

На 16-й итерации получают последовательности L(0) и R(0), которые конкатенируют в 64-битовую последовательность L(0)R(0).

Затем позиции битов этой последовательности переставляют в соответствии с матрицей IP. Результат такой перестановки - исходная 64битовая последовательность.

Теперь рассмотрим функцию шифрования f(R(i-1),K(i)). Схематически она показана на рис. 4.

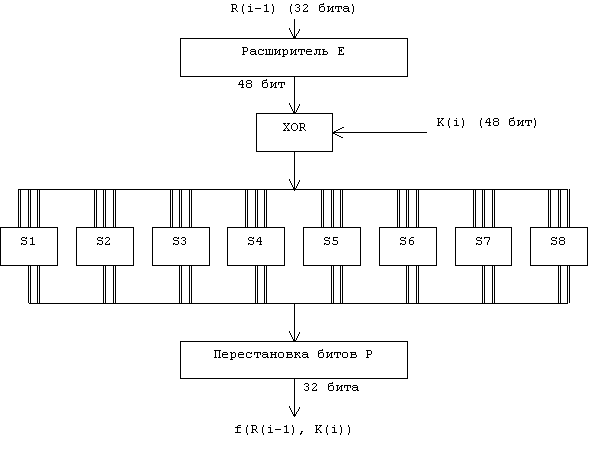


Рис.4. Вычисление функции f(R(i-1), K(i))

Для вычисления значения функции f используются следующие функции-матрицы:

* Е - расширение 32-битовой последовательности до 48-битовой,
* S1, S2, ... , S8 - преобразование 6-битового блока в 4-битовый,
* Р - перестановка бит в 32-битовой последовательности.[3]

Функция расширения Е определяется табл.3. В соответствии с этой таблицей первые 3 бита Е(R(i-1)) - это биты 32, 1 и 2, а последние - 31, 32 и 1.

Таблица 4. ***Функция расширения E***

32 01 02 03 04 05

04 05 06 07 08 09

08 09 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 01

Результат функции Е(R(i-1)) есть 48-битовая последовательность, которая складывается по модулю 2 (операция xor) с 48-битовым ключом К(i).

Получается 48-битовая последовательность, которая разбивается на восемь 6битовых блоков B(1)B(2)B(3)B(4)B(5)B(6)B(7)B(8). То есть:

E(R(i-1)) xor K(i) = B(1)B(2)...B (8) .

Функции S1, S2, ... , S8 определяются табл.5.

Таблица 5. Функции преобразования S1, S2, ..., S8

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  |
| Номер столбца  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 |
|  |  |  | S1 |
| 0  1  2  3 | 14 4 13 1 2 15 11 8 3 10 6 12 5 9 0 7  0 15 7 4 14 2 13 1 10 6 12 11 9 5 3 8  4 1 14 8 13 6 2 11 15 12 9 7 3 10 5 0  15 12 8 2 4 9 1 7 5 11 3 14 10 0 6 13 |
|  |  | S2 |
| 0  1  2  3 | 15 1 8 14 6 11 3 4 9 7 2 13 12 0 5 10  3 13 4 7 15 2 8 14 12 0 1 10 6 9 11 5  0 14 7 11 10 4 13 1 5 8 12 6 9 3 2 15  13 8 10 1 3 15 4 2 11 6 7 12 0 5 14 9 |
|  |  | S3 |
| 0  1  2  3 | 10 0 9 14 6 3 15 5 1 13 12 7 11 4 2 8  13 7 0 9 3 4 6 10 2 8 5 14 12 11 15 1  13 6 4 9 8 15 3 0 11 1 2 12 5 10 14 7  1 10 13 0 6 9 8 7 4 15 14 3 11 5 2 12 |
| Н ом е р с т р о к и |
|  |  | S4 |
| 0  1  2  3 | 7 13 14 3 0 6 9 10 1 2 8 5 11 12 4 15  13 8 11 5 6 15 0 3 4 7 2 12 1 10 14 9  10 6 9 0 12 11 7 13 15 1 3 14 5 2 8 4  3 15 0 6 10 1 13 8 9 4 5 11 12 7 2 14 |
|  |  | S5 |
| 0  1  2  3 | 2 12 4 1 7 10 11 6 8 5 3 15 13 0 14 9  14 11 2 12 4 7 13 1 5 0 15 10 3 9 8 6  4 2 1 11 10 13 7 8 15 9 12 5 6 3 0 14  11 8 12 7 1 14 2 13 6 15 0 9 10 4 5 3 |
|  |  | S6 |
| 0  1  2  3 | 12 1 10 15 9 2 6 8 0 13 3 4 14 7 5 11 10 15 4 2 7 12 9 5 6 1 13 14 0 11 3 8  9 14 15 5 2 8 12 3 7 0 4 10 1 13 11 6  4 3 2 12 9 5 15 10 11 14 1 7 6 0 8 13 |
|  |
|  |  | S7 |
| 0  1  2  3 | 4 11 2 14 15 0 8 13 3 12 9 7 5 10 6 1  13 0 11 7 4 9 1 10 14 3 5 12 2 15 8 6  1 4 11 13 12 3 7 14 10 15 6 8 0 5 9 2  6 11 13 8 1 4 10 7 9 5 0 15 14 2 3 12 |
|  |  | S8 |
| 0  1  2  3 | 13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7  1 15 13 8 10 3 7 4 12 5 6 11 0 14 9 2  7 11 4 1 9 12 14 2 0 6 10 13 15 3 5 8  2 1 14 7 4 10 8 13 15 12 9 0 3 5 6 11 |
|  |  |

К табл.5. требуются дополнительные пояснения. Пусть на вход функции-матрицы Sj поступает 6-битовый блок B(j) = b1b2b3b4b5b6, тогда двухбитовое число b1b6 указывает номер строки матрицы, а b2b3b4b5 - номер столбца. Результатом Sj(B(j)) будет 4-битовый элемент, расположенный на пересечении указанных строки и столбца.

Например, В(1)=011011. Тогда S1(В(1)) расположен на пересечении строки 1 и столбца 13. В столбце 13 строки 1 задано значение 5. Значит, S1(011011)=0101.

Применив операцию выбора к каждому из 6-битовых блоков B(1), B(2),

..., B(8), получаем 32-битовую последовательность

S1(B(1))S2(B(2))S3(B(3))...S8(B(8)).

Наконец, для получения результата функции шифрования надо переставить биты этой последовательности. Для этого применяется функция перестановки P (табл.5). Во входной последовательности биты перестанавливаются так, чтобы бит 16 стал битом 1, а бит 7 - битом 2 и т.д.

Таблица 6. ***Функция перестановки P***

16 07 20 21

29 12 28 17

01 15 23 26

05 18 31 10

02 08 24 14

32 27 03 09

19 13 30 06

22 11 04 25

Таким образом,

f(R(i-1), K(i)) = P(S1(B(1)),...S8(B(8)))

Чтобы завершить описание алгоритма шифрования данных, осталось привести алгоритм получения 48-битовых ключей К(i), i=1...16. На каждой итерации используется новое значение ключа K(i), которое вычисляется из начального ключа K. K представляет собой 64-битовый блок с восемью битами контроля по четности, расположенными в позициях

8,16,24,32,40,48,56,64.

Для удаления контрольных битов и перестановки остальных используется функция G первоначальной подготовки ключа (табл.7).

Таблица 7. Матрица G первоначальной подготовки ключа

57 49 41 33 25 17 09

01 58 50 42 34 26 18

10 02 59 51 43 35 27

19 11 03 60 52 44 36

63 55 47 39 31 23 15

07 62 54 46 38 30 22

14 06 61 53 45 37 29

21 13 05 28 20 12 04

Результат преобразования G(K) разбивается на два 28-битовых блока

C(0) и D(0), причем C(0) будет состоять из битов 57, 49, ..., 44, 36 ключа K, а D(0) будет состоять из битов 63, 55, ..., 12, 4 ключа K. После определения C(0) и D(0) рекурсивно определяются C(i) и D(i), i=1...16. Для этого применяют циклический сдвиг влево на один или два бита в зависимости от номера итерации, как показано в табл.8.

Таблица 8. Таблица сдвигов для вычисления ключа

|  |  |
| --- | --- |
| Номер итерации | Сдвиг (бит) |
|  |  |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | 1  1  2  2  2  2  2  2  1  2  2  2  2  2  2  1 |
|  |  |

Полученное значение вновь "перемешивается" в соответствии с матрицей H (табл.9).

Таблица 9. ***Матрица H завершающей обработки ключа***

14 17 11 24 01 05

03 28 15 06 21 10

23 19 12 04 26 08

16 07 27 20 13 02

41 52 31 37 47 55

30 40 51 45 33 48

44 49 39 56 34 53

46 42 50 36 29 32

Ключ K(i) будет состоять из битов 14, 17, ..., 29, 32 последовательности C(i)D(i). Таким образом:

K(i) = H(C(i)D(i))

Блок-схема алгоритма вычисления ключа приведена на рис.5.

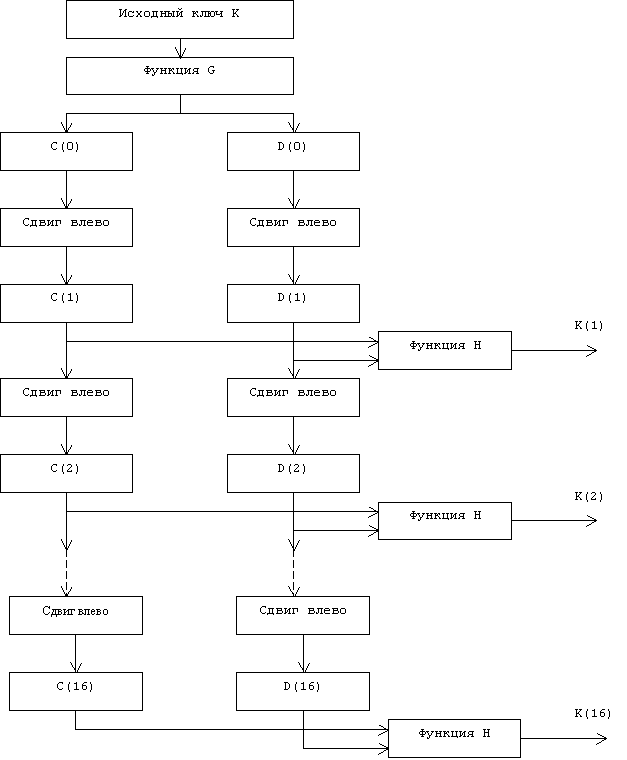


Рис.5. Блок-схема алгоритма вычисления ключа K(i)

Восстановление исходного текста осуществляется по этому алгоритму, но вначале вы используете ключ K(15), затем - K(14) и так далее.

### Выполнение работы

Реализовать приложение для шифрования на основе алгоритма DES, позволяющее выполнять следующие действия:

1) шифруемый текст должен храниться в одном файле, а ключ шифрования – в другом;

2)зашифрованный текст должен сохраняться в файл;

3)расшифрованный текст должен сохраняться в файл.

**Отчётные материалы**

Преподавателю предоставляется работающая программа и отчёт. Отчёт содержит постановку задачи, код программы и несколько результатов работы программы. Студент должен объяснить принцип работы алгоритма при защите лабораторной работы.

## Контрольные вопросы

1. Какова величина блока открытого текста и ключа в алгоритме DES?
2. Каковы основные шаги алгоритма DES?
3. Как осуществляется процесс дешифрования?
4. Как вычисляется ключ?
5. Какие стандартные таблицы используются в процессе шифрования?

# Лабораторная работа №4. Алгоритм RC4

### Краткие теоретические сведения

Алгоритм шифрования **RC4** — широко используемый [потоковый шифр,](http://grishnan.ru/books/secure/stream_cipher.html) который применяется в таких популярных протоколах, как **TLS** (для защиты Интернет-трафика) и **WEP** (для защиты WLAN-сетей).

Ядро алгоритма [поточных шифров](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80) состоит из функции — [генератора псевдо случайных](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) [битов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) [(гаммы)](https://ru.wikipedia.org/wiki/%D0%93%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), который выдаёт поток битов ключа (ключевой поток, гамму, последовательность псевдо случайных битов). [2]

Алгоритм шифрования.

1. Функция генерирует последовательность битов (ki).
2. Затем последовательность битов

посредством операции «суммирование по модулю дв[а»](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) (xor) объединяется с [открытым текстом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) (mi). В результате получается [шифрограмма](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82) (ci): .

Алгоритм расшифровки.

1. Повторно создаётся (регенерируется) поток битов ключа

(ключевой поток) (ki).

1. Поток битов ключа складывается с [шифрограммой](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82) (ci) операцией [«xor»](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2). В силу свойств операции «[xor»](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) на выходе получается исходный (не зашифрованный) текст (mi): .

**RC4** генерирует псевдослучайный поток битов, который называют гаммой или гаммирующей последовательностью (по англ. **keystream**). Как и в любом другом потоковом шифре, шифрование в **RC4** осуществляется с помощью операции **XOR** над гаммой и открытым текстом. Расшифровка происходит с помощью операции **XOR** над гаммой и шифротекстом. Таким образом, самым принципиальным моментом в **RC4** является то, каким образом генерируется гамма.

В самую первую очередь пользователь создаёт секретный ключ, длина которого, обычно, находится в пределах от 5 до 32 байт. Затем происходит процесс под названием **инициализация S-блока** (**Key-scheduling algorithm**).

Сначала берётся массив из 256 байтов (S-блок) и последовательно заполняется целыми числами от 0 до 255. Потом происходит перемешивание этого массива. Причем перемешивание зависит только от ключа в том смысле, что результат перемешивания всегда жёстко детерминирован после выбора ключа.[8]

Следующий этап носит название **генерации псевдослучайной последовательности** (**Pseudo-random generation algorithm**). Это итеративный процесс, который генерирует псевдослучаный поток байтов, играющий роль гаммирующей последовательности. Каждый байт гаммы жестко зависит от S-блока. Впоследствии, как уже упоминалось, байты гаммирующей последовательности будут складываться с помощью операции **XOR** с байтами открытого текста. Это и есть шифрование **RC4**.

В одном цикле RC4 определяется одно *n*-битное слово K из ключевого потока. В дальнейшем ключевое слово будет [сложено по модулю два](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2) с исходным текстом, которое пользователь хочет зашифровать, и получен зашифрованный текст.

Зациклив этот участок кода, можно сгенерировать столько байтов гаммирующей последовательности, сколько требуется для решения задачи шифрования или расшифровки.

### Выполнение работы

Для выполнения работы необходимо:

1. В собственном рабочем пространстве создать проект с именем Lab03.
2. В коде реализовать отдельный модуль (rc4.h, rc4.cpp), содержащий класс для алгоритма RC4 (назвать класс - RC4). Класс должен предусматривать минимум два метода - конструктор, которому передаётся строка байтов ключа и метод получения следующего случайного значения - GetNext(). Класс должен "уметь" поддерживать состояние S-блока межу вызовами.
3. Реализовать консольное приложение Win32, которое требует передачи трёх параметров - входной файл, выходной файл и файл, содержащий строку ключа. Запуск приложения должен осуществляться из командной строки, например: "C:\lab03.exe infile.txt outfile.txt keyfile.txt". Для передачи параметров используются параметры argc и argv функции main() программы. Преобразование шифрования/дешифрования - побитовое XOR для потоков сообщения и случайных байтов.
4. Выполнить шифрование и дешифрование текстового файла с одним и различными ключами.

Примечание: Т.к. в этом случае для шифрования используется побитовое XOR между потоком сообщения и потоком случайных байтов, то шифрование и дешифрование выполняются полностью идентично.

##### Отчётные материалы

Преподавателю демонстрируется работающая программа и предоставляется печатный отчёт. Отчёт содержит оформленный согласно [требованиям](http://192.168.18.18/student/course/isec/lab/index.htm) код программы и несколько результатов работы программы. Студент должен объяснить принцип работы алгоритма при защите лабораторной работы.

# Контрольные вопросы

1. Из каких шагов состоит алгоритм шифрования RC4?
2. Как произвести дешифрование по алгоритму RC4?
3. Что из себя представляет гамма?
4. Какими свойствами обладает [потоковый шифр?](http://grishnan.ru/books/secure/stream_cipher.html)

# Лабораторная работа №5. Ассиметричные алгоритмы шифрования данных: алгоритм RSA

### Краткие теоретические сведения

**RSA (**аббревиатура от фамилий создателей: Rivest, Shamir и Adleman) – один из самых популярных алгоритмов шифрования. Сначала приведем несколько определений:

*mod* – операция взятия остатка от деления,

под простым числом будем понимать такое число, которое делится только на и на само себя

*взаимно простыми* называются такие числа, которые не имеют между собой ни одного общего делителя, кроме единицы.

Алгоритм RSA включает в себя следующие шаги:

*Генерация ключей*

* 1. Выбрать два больших простых числа p и q; вычисляется их произведение n = p • q, называемое модулем.
  2. Вычисляется функция Эйлера Ф(n) = m=(p – 1) · (q – 1)
  3. Выбирается произвольное число e (e < n), такое, что 1 < e < m и не имеет общих делителей, кроме 1 (взаимно простое) с числом (p – 1) · (q – 1).
  4. Вычисляется d методом Евклида таким образом, что (e • d – 1) делится на (p – 1) • (q – 1).
  5. Два числа (*е*, *n*) публикуются как открытый ключ.
  6. Числа (*d,n)* хранятся в секрете как закрытый ключ.

Открытым ключом зашифровывают сообщение, а закрытым – расшифровывают.

1. *Шифрование*

Шифрование с помощью пары чисел производится следующим образом:

2.1. Отправитель разбивает своё сообщение *M* на блоки *mi*. Значение *mi < n*, поэтому длина блока *mi* в битах не больше *k* = [log2(*n*)] бит, где квадратные скобки обозначают, взятие целой части от дробного числа.

Например, если *n* = 21, то максимальная длина блока *k* = [log2(21)] = [4.39…]= = 4 бита.

Обычно блок берут равным одному символу и представляют этот символ в виду числа – его номера в алфавите или кода в таблице символов (например ASCII или Unicode).

2.1. Для каждого такого числа *mi* вычисляется выражение (*ci* – зашифрованное сообщение): *ci* = ((*mi*)*e*) mod *n*.

*Дешифрование*

Чтобы получить открытый текст, необходимо каждый блок дешифровать отдельно: mi = ((*ci*)*d*) mod *n*. [9] *Пример:*

Выбрать два простых числа: *р* = 7, *q* = 17.

Вычислить *n* = *p* *· q* = 7 *·* 17 = 119.

Вычислить *Ф*(*n*) = (*p* – 1) *·* (*q* – 1) = 96.

Выбрать *е* так, чтобы *е* было взаимнопростым с *Ф*(*n*) = 96 и меньше, чем

*Ф*(*n*): *е* = 5.

Определить *d* так, чтобы *d · e* ≡ 1 mod 96 и *d* < 96, *d* = 77, так как

77 *·* 5 = 385 = 4 *·* 96 + 1.

Результирующие ключи открытый {5, 119} и закрытый ключ {77, 119}. Например, требуется зашифровать сообщение *М* = 19: 195 = 66 (mod 119),

*С* = = 66. Для дешифрования вычисляется 6677 (mod 119) = 19.

### 6.3. Выполнение работы

Реализовать программу для шифрования / дешифрования текстов, работающую по алгоритму RSA.

Реализовать приложение для шифрования/ дешифрования, позволяющее выполнять следующие действия:

1. Вычислять открытый и закрытый ключи для алгоритма RSA:

1. числа p и q генерируются программой или задаются из файла;
2. числа p и q должны быть больше, чем 2128 ;
3. сгенерированные ключи сохраняются в файлы: открытый ключ – в один файл, закрытый – в другой.
4. исходный и зашифрованный тексты хранятся в файлах

###### Отчётные материалы

Преподавателю демонстрируется работающая программа и предоставляется печатный отчёт. Отчёт содержит оформленный согласно [требованиям](http://192.168.18.18/student/course/isec/lab/index.htm) код программы и несколько результатов работы программы. Студент должен объяснить принцип работы алгоритма при защите лабораторной работы.

# Контрольные вопросы

1. Дайте определение алгоритма с открытым ключом.
2. Сколько этапов содержит алгоритм RSA?
3. В чем заключается вычисление ключей алгоритма RSA?
4. Как происходит шифрование в алгоритме RSA?
5. Как происходит дешифрование в алгоритме RSA?

# Лабораторная работа №6. Комбинирование симметричных и асимметричных алгоритмов

### Краткие теоретические сведения

Симметричные алгоритмы и, в частности, DES – быстрые, поэтому ими удобно шифровать большие объёмы информации. Однако для передачи ключа симметричного алгоритма требуется надёжный канал передачи, который очень часто отсутствует. Таким образом, преимущества таких алгоритмов сводятся на нет. С другой стороны, асимметричные алгоритмы не требуют секретного канала для передачи ключа, но на практике криптосистемы с открытым ключом используются для шифрования не сообщений, а ключей. На это есть две основные причины:

1. Алгоритмы шифрования с открытым ключом в среднем работают в тысячи раз медленнее, чем симметричные алгоритмы, а также они требовательны к памяти и вычислительной мощности компьютера, поэтому большие тексты кодировать этими алгоритмами нецелесообразно.
2. Алгоритмы шифрования с открытым ключом уязвимы по отношению к криптоаналитическим атакам со знанием открытого текста. Пусть С=E(P) где C обозначает шифртекст, P –открытый текст,E – функцию шифрования. Тогда, если P принимает значения из некоторого конечного множества, состоящего из n открытых текстов, криптоаналитику достаточно зашифровать все эти тексты, используя известный ему открытый ключ, и сравнить результаты c C . Ключ таким способом ему вскрыть не удастся, однако открытый текст будет успешно определён.

Возможно следующее решение: сообщение шифруется симметричным алгоритмом, что позволяет выиграть в скорости, т.к. сообщение может быть сколь угодно большим, а ключ симметричного алгоритма (обычно маленький, для DES – 56 бит) шифруется асимметричным алгоритмом.

### Выполнение работы

Результатом данной лабораторной работы должны стать приложения, совмещающие в себе достоинства симметричных и асимметричных методов шифрования.

Реализовать приложение для шифрования, позволяющее выполнять следующие действия:

1. Шифровать симметричным алгоритмом DES открытый текст, а асимметричным RSA – ключ симметричного алгоритма;
   * шифруемый текст должен храниться в одном файле, открытый ключ для алгоритма RSA – в другом;
   * ключ для симметричного алгоритма DES должен генерироваться случайным образом;
   * зашифрованный текст должен сохраняться в одном файле, а зашифрованный асимметричным алгоритмом ключ симметричного алгоритма – в другом;
   * программа должна уметь работать с текстом произвольной длины.
2. Реализовать приложение для дешифрования.

 Зашифрованный текст должен храниться в одном файле, зашифрованный ключ симметричного алгоритма – в другом, а секретный ключ для алгоритма RSA – в третьем.

# Контрольные вопросы

1. Для чего и почему используют комбинированные криптоалгоритмы?
2. В чём заключаются достоинства и недостатки асимметричных алгоритмов?
3. В чём заключаются достоинства и недостатки симметричных алгоритмов?
4. Найти алгоритмом Евклида элемент d такой, что e\*d=1(mod n), если e=15 n=82.

# Лабораторная работа №7. Алгоритм N-хэш

### Краткие теоретические сведения

Алгоритм N-хэш был предложен в 1990-м году исследователями компании Nippon Telephone and Telegraph. Он использует 128-разрядные блоки сообщения, сложную рандомизирующую функцию, а его результатом является 128-разрядное хэш-значение для исходного сообщения.

В основе [алгоритма](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) N-Hash лежит [блочный алгоритм шифрования](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80) FEAL.

###### Алгоритм

Алгоритм N-Hash основан на циклическом повторении (12 или 15 раз — число раундов) операций. На входе имеется хеш-код h0 и он может быть произвольным, на выходе получается хеш-код *h* сообщения *M*, которое необходимо хешировать. При этом размер выходящего хеш-кода фиксирован и равен 128 бит, тогда как размер *M* произволен. [8]

Основные обозначения

* M — сообщение, которое необходимо хешировать;
* Mi — блок сообщения длиной 128 бит. Для того, чтобы хешировать сообщение M необходимо поделить его на блоки Mi;
* hi — [хеш-](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%81%D1%83%D0%BC%D0%BC%D0%B0)код i-го шага;
* v=1010…1010 — константа, длиной 128 бит;
* || — [конкатенация;](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%BA%D0%B0%D1%82%D0%B5%D0%BD%D0%B0%D1%86%D0%B8%D1%8F)
* Vj=δ||Aj1|| δ||Aj2|| δ||Aj3|| δ||Aj4||, где Ajk=4(j-1)+k, где k=1, 2, 3, 4; δ=00..00

, длиной 24 бит;

* EXG — функция, которая меняет местами старшие и младшие разряды

(64 младших и 64 старших);

* PS — преобразующая функция;

###### Описание алгоритма

* Покоординатное (попарное) суммирование означает [сложение по модулю 2;](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_2)
* Если *x* поступает на вход функции *f*, то на выходе получается *f(x)*.

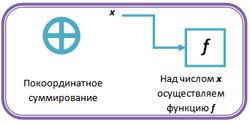


Рис. 7. Покоординатное (попарное) суммирование

###### Один цикл работы N-Hash

Ниже представлен один цикл работы алгоритма N-Hash.

* На вход функции g подается хеш-код (i-1)-го шага hi-1 и i-й блок сообщения Mi. При этом h0 выбирается произвольно: например, он может быть нулевым. А также hi-1 подается на выход на операцию сложения по модулю 2, то есть результат (хеш-код следующего шага) будет выглядеть так: hi-1 (*нечто пока неизвестное*).
* Из схемы видно, что Mi подается не только на *XOR*, но и на выход на операцию сложения по модулю 2. То есть теперь в соответствии с первым пунктом результат выглядит таким образом: hi-1  (*оставшееся пока неизвестным нечто*).

*Оставшееся пока неизвестным нечто* находится после прохождения каскада из восьми преобразующих функций. Его получение может быть описано таким образом:

* Функция EXG меняет местами старшие и младшие разряды hi-1 и прибавляет к результату v, после чего результат складывает по модулю 2 с Mi.
* Как видно из схемы, результат подается последовательно на входы j преобразующих функций, вторым аргументом которых является сумма hi-1 Vj, где j=1, … , 8.
* В результате получается хеш-код i-го шага hi: hi=Mi g(Mi, hi-1)hi-1.

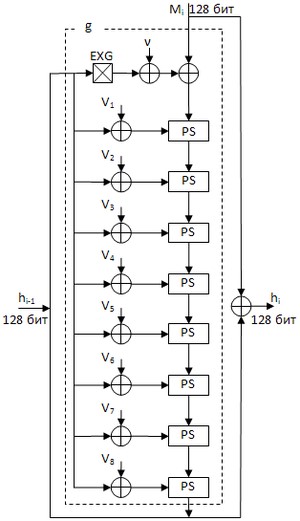


Рис.8. Один цикл работы N-Hash

###### Преобразующая функция

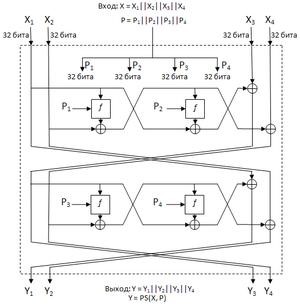


Рис. 9. Схема преобразующей функции. Каждый из аргументов разбивается на 4 блока по 32 бит каждый.

Возникает вопрос, как действует преобразующая функция PS(X,P).

Рассмотрим верхнюю часть схемы до перекрестья.

Исходное сообщение Xразбивается на блоки по 128/4=32 бита.

Будем считать *промежуточными выходами* входы в нижнюю часть схемы. X1 и X2 подаются на *промежуточные выходы*, а на два других выхода подаются операции f(X1, P1) X2X4 и f(f(X1, P1) X2,P2) X1 X3.

Теперь можно результаты на промежуточных выходах переобозначить и через них, аналогично верхней части, найти результаты на выходе нижней части, то есть и всей схемы в целом.[9]

Сделав все необходимые вычисления, получим, что при подаче на вход X= X1|| X2|| X3|| X4 сообщение на выходе Y= Y1|| Y2|| Y3|| Y4 можно представить как [конкатенацию](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%BA%D0%B0%D1%82%D0%B5%D0%BD%D0%B0%D1%86%D0%B8%D1%8F) сообщений

* Y4 = X2 X4  f(X1, P1);
* Y3 =f(f(X1, P1) X2,P2) X1 X3;
* Y2= X2 Y4  f(Y3, P3);
* Y1 =f(f(Y3, P3) Y4,P4) X1 Y3;



###### Поиск функции f(x, P)

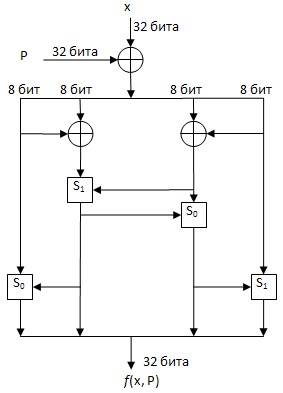


Рис. 10. Схема поиска функции f(x ,P)

Так как функция f работает с аргументами, длина которых составляет 32 бит, то из схемы поиска функции f(x, P) имеем:

* Величину xP разбиваем на части по 8 бит.
* Запишем эти части как xi Pi, i=1,…,4 и введёт новые обозначения:
  + Z1= x1 P1; o Z2= x2 P2; o Z3= x3 P3; o Z4= x4 P4;

Аргументами функции S0 (первая стрелка слева) являются Z1 и S1(Z1 Z2, Z3 Z4). Аргументами функции S1 (вторая стрелка слева) являются Z1 Z2,

Z3 Z4.

То есть две составляющие части из сообщения на выходе уже известны и равны

* + A1= S0(Z1, S1(Z1 Z2, Z3 Z4)); o A2= S1(Z1 Z2, Z3 Z4);

Далее будем пользоваться уже полученными оставляющими частями сообщения на выходе для удобства записи:

* + A3= S0(S0(Z3 Z4), A2); o A4= S1(Z4, A3 );
* Тогда сообщение на выходе можно представить в виде A= A1|| A2|| A3|| A4.
* Причём известно, что o S0=(*левый циклический сдвиг на 2 бита*)(a+b) mod 256 o S1=(*левый циклический сдвиг на 2 бита*)(a+b+1) mod 256

Разработчика N-хэш советуют использовать не менее 8 раундов. [9]

### Выполнение работы

Для выполнения работы необходимо:

1. В собственном рабочем пространстве создать проект с именем Lab04.
2. В коде реализовать отдельный модуль (nhash.h, nhash.cpp), содержащий класс для алгоритма N-хэш (назвать класс - NHash). Класс должен предусматривать ряд методов для работы. В частности, (1) должен быть предусмотрен механизм чтения исходного сообщения из потока (файла, памяти и пр.), (2) методы вычисления промежуточных функций должны быть скрытыми, (3) интерфейс класса должен быть максимально простым.
3. \*Рассмотреть возможность настройки класса на использование произвольного числа этапов.
4. Реализовать консольное приложение Win32, которое требует передачи параметра - входной файл, и выводит на экран соответствующее значение ХЭШ в шестнадцатеричном формате. Запуск приложения должен осуществляться из командной строки, например: "C:\lab04.exe infile.txt". Для передачи параметров используются параметры argc и argv функции main() программы.
5. Выполнить проверку работы алгоритма для нескольких входных файлов.

\*Не является обязательным требованием.

**Примечание.** Если исходное сообщение и имеет длину не кратную 128 разрядам (битам), то : (1) первоначально сообщение дополняется, что бы его длина стала меньше на 64 разряда, чем ближайшее число кратное 128 - для этого в конец добавляется единица и столько нулей, сколько необходимо, (2) последние 64 разряда заполняются 64 разрядным числом, равным длине исходного (до дополнения) сообщения.

###### Отчётные материалы

Преподавателю демонстрируется работающая программа и предоставляется печатный отчёт. Отчёт содержит оформленный согласно [требованиям](http://192.168.18.18/student/course/isec/lab/index.htm) код программы и несколько результатов работы программы. Студент должен объяснить принцип работы алгоритма при защите лабораторной работы.

# Контрольные вопросы

1. Что из себя представляет хэш функция?
2. Пояснить один цикл работы алгоритма N-Hash
3. Как формируется образующая функция для алгоритма N-Hash?
4. Какое минимальное количество раундов рекомендуется использовать в алгоритме N-Hash?
5. На какие блоки делится открытое сообщение во время N-Hash шифрования?

# Лабораторная работа №8. Алгоритм SHA

### Краткие теоретические сведения

SHA-1 реализует [хеш-функцию,](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) построенную на идее функции сжатия. Входами функции сжатия являются блок сообщения длиной 512 бит и выход предыдущего блока сообщения. Выход представляет собой значение всех хеш-блоков до этого момента. Иными словами хеш блока Mi равен hi=f(Mi, hi-

1,). Хеш-значением всего сообщения является выход последнего блока

Описание алгоритма

Для сообщения произвольной длины *l*, не превышающей 264 бит, алгоритм

SHA-1 формирует 160-битный хеш-образ.

Процедура формирования хеш-образа состоит из следующих шагов.

1. Исходное сообщение разбивается на блоки по 512 бит в каждом. Последний блок дополняется до длины, кратной 512 бит. Сначала добавляется 1 (бит), а потом нули, чтобы длина блока стала равной (512 - 64 = 448) бит. В оставшиеся 64 бита записывается длина исходного сообщения в битах (в big-endian формате). Если последний блок имеет длину более 448, но менее 512 бит, то дополнение выполняется следующим образом: сначала добавляется 1 (бит), затем нули вплоть до конца 512-битного блока; после этого создается ещё один 512-битный блок, который заполняется вплоть до 448 бит нулями, после чего в оставшиеся 64 бита записывается длина исходного сообщения в битах (в big-endian формате). Дополнение последнего блока осуществляется всегда, даже если сообщение уже имеет нужную длину.
2. Инициализируются пять 32-битовых переменных.

* A = a = 0x67452301
* B = b = 0xEFCDAB89
* C = c = 0x98BADCFE
* D = d = 0x10325476
* E = e = 0xC3D2E1F0 o Выполняется обработка очередных 512 бит исходного текста. Для этого значения переменных *A*, *B*, *C*, *D*, *E* копируются в переменные *a*, *b*, *c*, *d*, *e* и далее для *t* от 1 до 80 выполняется преобразование значений данных переменных по схеме, изображенной на рис. 11.

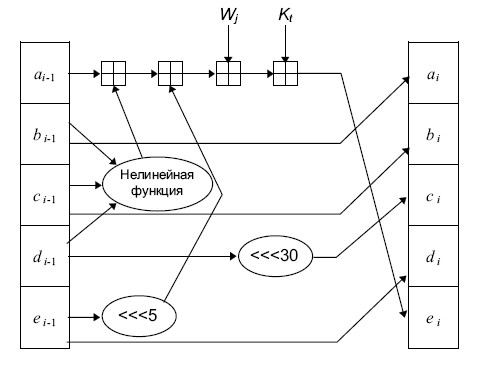


Рис. 11 . Схема итерации алгоритма SHA-1

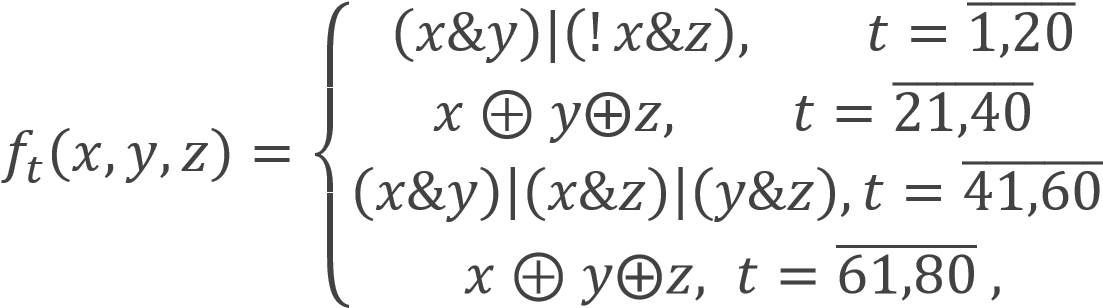
Если t - номер операции (от 1 до 80), Wt представляет собой t-й подблок расширенного сообщения, а <<<s - циклический сдвиг влево на s битов, то главный цикл выглядит следующим образом:

**для** t **от** 0 **до** 79

temp = (a<<5) + ft (b,c,d) + e +Kt e = d d = c

c = b<<30 b = a a = temp

где «+» – операция сложения по модулю 2, *ft*(*X*, *Y*, *Z*) – нелинейная функция, имеющая следующий вид:



где «&» – побитовая операция «И», «|» – побитовая операция «ИЛИ», «!» – операция побитового инвертирования, «» – операция побитового сложения по модулю 2. Параметр *Kt*принимает четыре различных значения в зависимости от номера текущей итерации:

*Kt*= 5A82799916, ;

*Kt*= 6ED9EBA116, ; *Kt* = 8F1BBCDC16, ; *Kt* = CA62C1D616, .

«<<<» – операция циклического сдвига на 30 либо 5 бит влево, *Wt*– одно из шестнадцати 32-битных слов 512-битного блока сообщения при

, либо значение, определяемое в соответствии со следующим выражением при :

.

1. Значения переменных *a*, *b*, *c*, *d*, *e* независимо друг от друга складываются по модулю 2 со значениями переменных *A*, *B*, *C*, *D*, *E*, в которые затем и помещаются полученные результаты.
2. Шаги 3–4 выполняются до тех пор, пока не будет обработан весь текст. После обработки последнего блока текста значение хеш-образа формируется как *ABCDE*.[8]

### Выполнение работы

Для выполнения работы необходимо:

1. В собственном рабочем пространстве создать проект с именем Lab04.
2. В коде реализовать отдельный модуль (SHA.h, SHA.cpp), содержащий класс для алгоритма SHA (назвать класс - SHA). Класс должен предусматривать ряд методов для работы. В частности, (1) должен быть предусмотрен механизм чтения исходного сообщения из потока (файла, памяти и пр.), (2) методы вычисления промежуточных функций должны быть скрытыми, (3) интерфейс класса должен быть максимально простым.
3. \*Рассмотреть возможность настройки класса на использование произвольного числа этапов.
4. Реализовать консольное приложение Win32, которое требует передачи параметра - входной файл, и выводит на экран соответствующее значение SHA в шестнадцатеричном формате. Запуск приложения должен осуществляться из командной строки, например: "C:\lab04.exe infile.txt". Для передачи параметров используются параметры argc и argv функции main() программы.
5. Выполнить проверку работы алгоритма для нескольких входных файлов.

\*Не является обязательным требованием.

**Примечание.** Если исходное сообщение и имеет длину не кратную 128разрядам (битам), то : (1) первоначально сообщение дополняется, что бы его длина стала меньше на 64 разряда, чем ближайшее число кратное 128 - для этого в конец добавляется единица и столько нулей, сколько необходимо, (2) последние 64 разряда заполняются 64 разрядным числом, равным длине исходного (до дополнения) сообщения.

###### Отчётные материалы

Преподавателю демонстрируется работающая программа и предоставляется печатный отчёт. Отчёт содержит оформленный согласно [требованиям](http://192.168.18.18/student/course/isec/lab/index.htm) код программы и несколько результатов работы программы. Студент должен объяснить принцип работы алгоритма при защите лабораторной работы.

# Контрольные вопросы

1. Каков размер блока для алгоритма SHA?
2. Пояснить процедуру формирования хеш-образа?
3. Пояснить схему итерации алгоритма SHA-1
4. На чем основан алгоритм SHA-1?
5. Какая образующая функция применяется в SHA-1?

# Лабораторная работа №9. Программирование криптографических провайдеров на платформе .NET: Алгоритмы хэширования SHA-1 и MD5

### Краткие теоретические сведения

Пространство *System.Security.Cryptography* предоставляет набор примитивов, реализующих криптографические примитивы, включая шифры, хэш-функции, генераторы случайных чисел и др.

На самом высоком уровне пространство имен Cryptography можно разбить на четыре основные части (табл. 1). Главное предназначение этого пространства — предоставлять классы с алгоритмами таких операций, как шифрование и создание хэшей. Эти [алгоритмы](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B) реализуются на основе расширяемого шаблона (pattern), включающего два уровня наследования.

На вершине иерархии располагается абстрактный базовый класс (вроде AsymmetricAlgorithm или HashAlgorithm), имя которого соответствует типу алгоритма. От такого класса наследует [абстрактный класс](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%81) второго уровня, предоставляющий открытый [интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) для использования данного алгоритма. Например, SHA1 (Secure Hash Algorithm) представляет собой производный от HashAlgorithm [класс](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81) и содержит методы и свойства, специфичные для алгоритма SHA1. Наконец, сама реализация алгоритма является производной от класса второго уровня; именно её [экземпляр](https://ru.wikipedia.org/wiki/%D0%AD%D0%BA%D0%B7%D0%B5%D0%BC%D0%BF%D0%BB%D1%8F%D1%80) создается и используется клиентским приложением. На этом уровне реализация может быть управляемой, неуправляемой или и той и другой.

Таблица 10. Основные элементы пространства имен Cryptography

|  |  |
| --- | --- |
| **Элемент** | **Описание** |
| Алгоритмы шифрования | Набор классов, применяемых для реализации алгоритмов симметричного и асимметричного шифрования, а также хэширования |
| Вспомогательные классы | Классы, обеспечивающие генерацию случайных чисел, выполнение преобразований, взаимодействие с хранилищем [CryptoAPI](https://ru.wikipedia.org/wiki/CryptoAPI) и само шифрование на основе потоковой модели |
| Сертификат ы Х.509 | Классы, определенные в пространстве имен  System.Security.Сryptographу. Х509Certificates и представляющие цифровые сертификаты |
| Цифровые подписи [XML](https://ru.wikipedia.org/wiki/XML) | [Классы,](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81%D1%8B) определенные в пространстве имен  System.Cryptography.Xml и представляющие цифровые подписи в [XML-](https://ru.wikipedia.org/wiki/XML)документах |

Пространство имен Cryptography содержит базовый класс HashAlgorithm и производные классы, поддерживающие алгоритмы [MD5,](https://ru.wikipedia.org/wiki/MD5) [SHA1,](https://ru.wikipedia.org/wiki/SHA1) [SHA256,](https://ru.wikipedia.org/wiki/SHA256) [SHA384](https://ru.wikipedia.org/w/index.php?title=SHA384&action=edit&redlink=1) и [SHA512.](https://ru.wikipedia.org/w/index.php?title=SHA512&action=edit&redlink=1) [Алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [MD5](https://ru.wikipedia.org/wiki/MD5) дает 128 битный [хэш,](https://ru.wikipedia.org/wiki/%D0%A5%D1%8D%D1%88) a [SHA1](https://ru.wikipedia.org/wiki/SHA1) — 160 битный. Числа в названиях других версий SHA-алгоритмов соответствуют длине создаваемых ими хэшей. Чем больше [хэш,](https://ru.wikipedia.org/wiki/%D0%A5%D1%8D%D1%88) тем надежнее алгоритм и тем труднее его взломать. Все эти [алгоритмы](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B) реализованы в двух версиях: на основе управляемого и неуправляемого кода.

HashAlqorithm

|— KeyedHashAlgorithm

| |— HMACSHA1

| |— MACTripleDES

|— MD5

| |— MD5CryptoServiceProvider

|— SHA1

| |— SHA1CryptoServiceProvider

| |— SHA1Managed

|— SHA256

| |— SHA256Managed

|— SHA384

| |— SHA384Managed

|— SHA512

| |— SHA512Managed

Рис. 12. Иерархия хэширующих алгоритмов

Чтобы вычислить [дайджест,](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%B9%D0%B4%D0%B6%D0%B5%D1%81%D1%82) нужно просто создать экземпляр класса алгоритма хэширования и вызвать его перегруженный метод ComputeHash, наследуемый от HashAlgorithm:

FileStream fsData = new FileStream(«mydata.txt»,FileHode.Open, FileAccess.Read); Byte[] digest;

SHA512Managed oSHA = new SHA512Managed(digest — oSHA.ComputeHash(fsData)); fsKey.Close()

Здесь методу ComputeHash передается объект Stream, но он принимает и байтовый массив. В пространстве имен Cryptography также имеется [абстрактный класс](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%81) KeyedHashAlgorithm. [Алгоритмы,](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B) реализованные в классах HMACSHA1 и MACTripleDES, производных от KeyedHashAlgorithm, позволяют генерировать Message Authentication Codes ([MAC)](https://ru.wikipedia.org/w/index.php?title=Message_Authentication_Codes&action=edit&redlink=1). С помощью MAC можно определить, были ли модифицированы данные, переданные по незащищенному каналу связи, — при условии, что и отправитель, и получатель используют общий секретный ключ. [6]

.NET Framework предоставляет следующие классы для вычисления хэш-значений:

* MD5CryptoServiceProvider
* SHA1CryptoServiceProvider и др. [7]

Следующий код демонстрирует использование данных провайдеров для вычисления хэш-значений для файла:

using System.IO; using System.Security.Cryptography;

...

// вычисление MD5

FileStream fs = new FileStream("filename.ext", FileMode.Open, FileAccess.Read);

MD5 md5 = new MD5CryptoServiceProvider(); byte[] md5res = md5.ComputeHash(fs); Console.WriteLine(ByteArrayToString(md5res)); fs.Close();

// вычисление SHA1

FileStream fs = new FileStream("filename.ext", FileMode.Open, FileAccess.Read);

SHA1 sha1 = new SHA1CryptoServiceProvider(); byte[] sha1res = sha1.ComputeHash(fs); textBoxSHA1.Text = ByteArrayToString(sha1res); fs.Close();

Следующий код реализует ByteArrayToString():

using System.Text;

...

// вспомогательная функция перевода байтового массива в шестнадцатеричную строку static string ByteArrayToString(byte[] arrInput)

{

int i;

StringBuilder sOutput = new StringBuilder(arrInput.Length); for (i=0;i < arrInput.Length -1; i++)

{

sOutput.Append(arrInput[i].ToString("X2"));

}

return sOutput.ToString();

}

### Выполнение работы

Реализовать консольное приложение с меню, в котором пользователь выбирает функцию и файл для вычисления хэш-значения.

Исследоватьпространство имён *System.Security.Cryptography.*

# Контрольные вопросы

1. Какие классы для вычисления хэш-значений предоставляет .NET Framework ?
2. Перечислить основные элементы пространства имен

Cryptography.

1. Как вычислить [дайджест?](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%B9%D0%B4%D0%B6%D0%B5%D1%81%D1%82)
2. Пояснить иерархию хэширующих алгоритмов?

# Лабораторная работа №9. Программирование криптографических провайдеров на платформе .NET: Алгоритм цифровой подписи DSA

### Краткие теоретические сведения

Электронная цифровая подпись (ЭЦП), как и обычная, позволяет установить некую отметку, указывающую на принадлежность электронного сообщения конкретному автору. Алгоритмы цифровой подписи тесно связаны с асимметричными шифрами. Например, алгоритм цифровой подписи RSA – это практически шифр RSA, но шифруется не само сообщение, а его дайджест, и шифрование производится не на открытом ключе, а на закрытом. В этом случае любой получатель, имеющий открытый ключ автора, может расшифровать дайджест и проверить его правильность.

[7]

Следующий код демонстрирует использование

DSACryptoServiceProvider:

using System;

using System.Collections.Generic; using System.Text; using System.IO;

using System.Security.Cryptography;

namespace DSAApp

{

class Program

{

static void Main(string[] args) {

// Экземпляр DSACryptoServiceProvider будет использоваться для

// начальной генерации и как контейнер ключей подписи и проверки

DSACryptoServiceProvider key = new DSACryptoServiceProvider();

// Массив с данными для подписи

byte[] dataToSign =

{ 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, 26, 27, 28, 29, 30, 31 };

Console.WriteLine("Подписывается массив:");

Console.WriteLine(ByteArrayToString(dataToSign));

// Вызов функции Sign() для получения подписи

// dataToSign - массив байт, для которого вычисляется подпись

// key.ExportParameters(true) - извлекает структуру DSAParameters c

// включением информации секретного ключа подписи byte[] signature = Sign(dataToSign, key.ExportParameters(true));

Console.WriteLine("Подпись:");

Console.WriteLine(ByteArrayToString(signature));

Console.Write("Проверка подписи ... ");

// Вызов функции VerifySignature для проверки подписи

// dataToSign - массив байт, для которого проверяется подпись

// signature - массив байт, содержащий подпись

// key.ExportParameters(false) - извлекает структуру DSAParameters с

// включением ТОЛЬКО информации открытого ключа проверки подписи

bool acceptSignature = VerifySignature(dataToSign, signature, key.ExportParameters(false)); if (acceptSignature) {

Console.WriteLine("УСПЕШНО!");

} else {

Console.WriteLine("ОШИБКА.");

}

Console.ReadLine();

}

//

// Функция вычисляет цифровую подпись DSA для массива байт data c ключом privateKey

//

static byte[] Sign(byte[] data, DSAParameters privateKey) {

// Экземпляр провайдера DSA

DSACryptoServiceProvider dsa = new DSACryptoServiceProvider();

// Импорт ключа для вычисления подписи dsa.ImportParameters(privateKey);

// Вычисление и возврат массива байт подписи

return dsa.SignData(data);

}

//

// Функция проверяет цифровую подпись signature для data с ключом publicKey

//

static bool VerifySignature(byte[] data, byte[] signature, DSAParameters publicKey) {

// Экземпляр провайдера DSA

DSACryptoServiceProvider dsa = new DSACryptoServiceProvider();

// Импорт ключа для проверки подписи dsa.ImportParameters(publicKey);

// Возврат статуса проверки подписи return dsa.VerifyData(data, signature);

}

//

// Функция преобразует байтовый массив в шестнадцатеричную строку

//

static string ByteArrayToString(byte[] arrInput)

{

int i;

StringBuilder sOutput = new StringBuilder(arrInput.Length);

for (i = 0; i < arrInput.Length - 1; i++)

{

sOutput.Append(arrInput[i].ToString("X2"));

}

return sOutput.ToString();

}

}

}

### Выполнение работы

Изучить представленный код и запустить программу.

**Отчётные материалы** скриншоты запуска

# Контрольные вопросы

1 . Что такое ЭЦП?

1. Какие существуют алгоритмы формирования ЭЦП?
2. Как может использоваться экземпляр DSACryptoServiceProvider?
3. Как проверяется подпись?

+ <https://xn--80aqobguv5e.xn--p1ai/%D0%BE%D0%B8%D0%B1/index.html>