# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»

NSTU_Logo_blue

## Кафедра теоретической и прикладной информатики

### Лабораторная работа № 3 по дисциплине «Информационная безопасность»

|  |  |  |
| --- | --- | --- |
| сигма градиент синий1 | Факультет: | ПМИ |
| Группа: | ПМИ-61 |
| Студенты: | Ершов П.К., Мамонова Е.В., Цыденов З.Б. |
| Вариант: | 2 |
| Контактный адрес | Ершов П.К. - [PMI62.Ershov@outlook.com](mailto:PMI62.Ershov@outlook.com)  Мамонова Е.В. - [PMI62.Mamonova@outlook.com](mailto:PMI62.Mamonova@outlook.com)  Цыденов З.Б.- [tsydebov.zanabazar@gmail.com](mailto:tsydebov.zanabazar@gmail.com) |
| Преподаватель: | Авдеенко Т.В. |

Новосибирск

2020

1. **Цель работы**

Ознакомиться с существующими криптографическими библиотеками. Научиться использовать сторонние криптографические библиотеки при разработке собственных приложений.

1. **Задание**
2. Найти криптографическую библиотеку.
3. Реализовать приложение с графическим интерфейсом, позволяющее выполнять следующие действия.
4. Шифровать и дешифровать выбранный пользователем файл с использованием одного или нескольких симметричных алгоритмов шифрования:
5. результаты шифрования и дешифрования должны сохраняться в файлы;
6. требуемые параметры шифрования, такие как ключ, вектор инициализации и пр., должны считываться из файла.
7. Шифровать и дешифровать выбранный пользователем файл с использованием одного асимметричного алгоритма шифрования:
8. реализовать процедуру генерации пары открытый–закрытый ключ;
9. результаты шифрования и дешифрования должны сохраняться в файлы;
10. требуемые параметры шифрования должны считываться из файла.
11. Вычислять и проверять электронную цифровую подпись для выбранного пользователем файла с использованием одного из алгоритмов
12. результаты вычисления подписи должны сохраняться в файлы;
13. требуемые параметры вычисления и проверки подписи должны считываться из файла.
14. Вычислять хэш-значения для выбранного пользователем файла по одному или нескольким алгоритмам хэширования, при этом вычисленное хэш-значение должно сохраняться в файл.
15. Протестировать правильность работы реализованного приложения.

**3.** Описание разработанного программного средства

Была выбрана криптографическая библиотека языка C# System.Security.Cryptography.

**Для запуска программного средства необходимо запустить файл IS\_lab3.exe, расположенный в IS\_lab3\IS\_lab3\bin\Debug\.**

В качестве демонстрационных алгоритмов были выбраны:

Для симметричных алгоритмов шифрования:

1. 3DES.
2. AES.

Для асимметричных алгоритмов шифрования:

1. RSA.

Для создания цифровых подписей:

1. DSA.
2. ECDSA.
3. HMACSHA512 (алгоритм HMAC на основе SHA512).

Для создания хэшей:

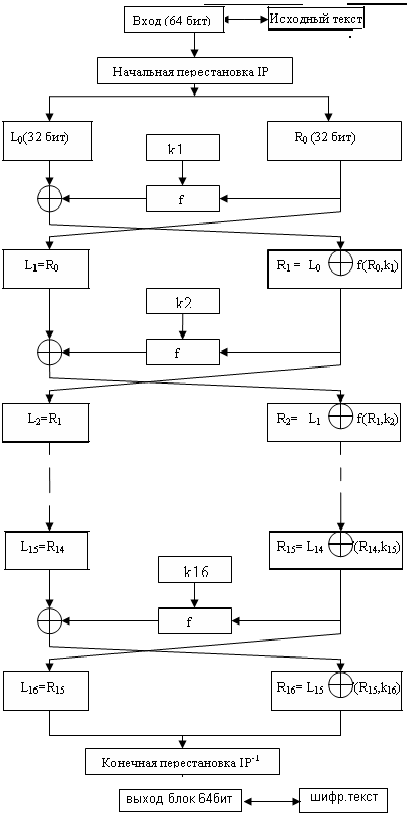
1. SHA512.
2. RIPEMD160.

Описание используемых алгоритмов:

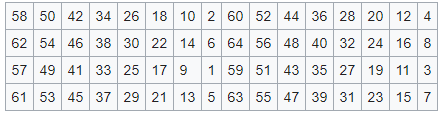
1. **3DES**

Симметричный алгоритм шифрования. Создан на основе DES алгоритма. Основная идея алгоритма заключается в тройном использовании алгоритма DES. Также, вместо 56 битного ключа в DES, в 3DES используется 168 битный ключ, что значительно повышает его надёжность.

В качестве блочного шифра в DES используется сеть Фейстеля:



В ходе работы алгоритма происходит обработка блока, длиной 63 бита. Исходный текст разбивают на блоки по 64 бита. Сначала происходит перестановка битов по следующей схеме:



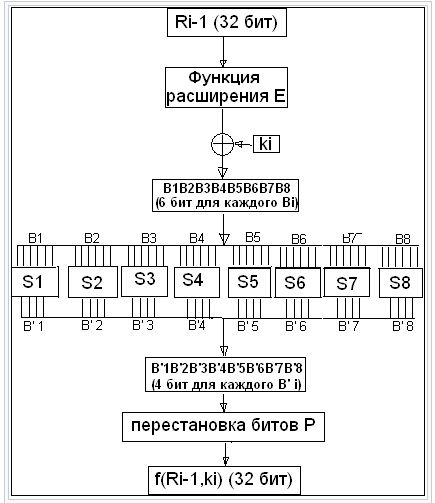
После этого происходит разбиение блока на L – подблок и R-подблок, которые являются младшими и старшими битами. В дальнейшем будут обрабатываться эти подблоки.

В сети Фейстеля f это функция, которой на вход подаётся 32-битный блок Ri и 48-битный ключ ki, которая вычисляется следующим образом:

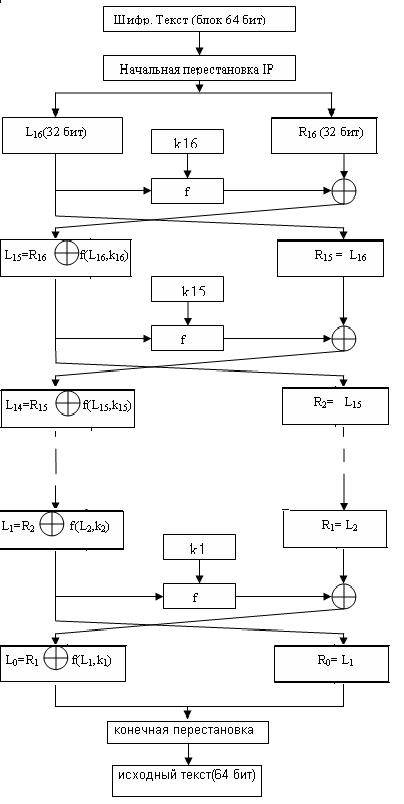
1. Функция расширения {\displaystyle \mathrm {E} }E,
2. Сложение по модулю 2 с ключом {\displaystyle k\_{i}}ki
3. Преобразование {\displaystyle \mathrm {S} }S, состоящее из 8 преобразований {\displaystyle \mathrm {S} }S-блоков {\displaystyle \mathrm {S} \_{1},\mathrm {S} \_{2},\mathrm {S} \_{3}\ldots \ \mathrm {S} \_{8}}S1, S2, …, S8.
4. Перестановка {\displaystyle \mathrm {P} }P.

Здесь, функция E расширяет 32-битный блок R до 48-битного блока E, путём дублирования некоторых битов R. После этого, блок E складывается с ключом ki по модулю и разбивается на 8 6-битных блока B1, …, B8. После этого происходит преобразование в 4-битовые блоки с помощью преобразований S, а затем происходит перестановка P полученных 8 4-битовых блоков.

Схема работы функции f:



Дешифрование происходит по обратной схеме:



Подобна схема шифрования в 3DES производиться трижды, для 3 разных ключей k1, k2, k3, которые являются взаимно независимыми.

Особенности библиотечного 3DES: так как 3DES относиться к симметричным алгоритмам, его реализация в System.Security.Cryptography не позволяет дешифровать данные ключом, отличным от использованного для шифрования. При попытке сделать это, будет выдано исключение.

1. **AES**

Является симметричным алгоритмом блочного шифрования с длиной ключа в 128, 192 или 256 бит.

Псевдокод алгоритма шифрования:

Cipher(byte in[4\*Nb], byte out[4\*Nb], word w[Nb\*(Nr+1)])

begin

byte state[4,Nb]

state = in

AddRoundKey(state, w[0, Nb-1])

for round = 1 step 1 to Nr-1

SubBytes(state)

ShiftRows(state)

MixColumns(state)

AddRoundKey(state, w[round\*Nb, (round+1)\*Nb-1])

end for

SubBytes(state)

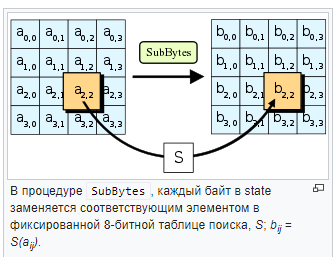
ShiftRows(state)

AddRoundKey(state, w[Nr\*Nb, (Nr+1)\*Nb-1])

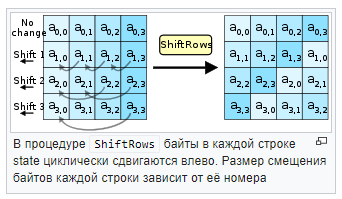
out = state

end

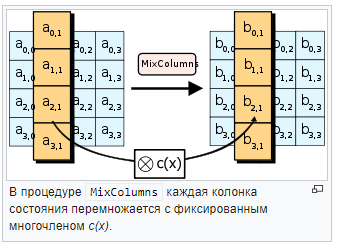
Здесь, функция SubBytes:



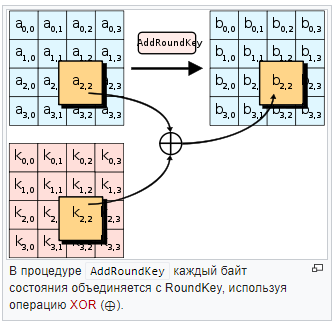
функция ShiftRows:



функция MixColumns:



фукнция AddRoundKey:



Алгоритм обработки ключа состоит из двух процедур:

* Алгоритм генерации раундовых ключей (алгоритм расширения ключа)
* Алгоритм выбора раундового ключа (ключа итерации)

Особенности библиотечного AES: аналогично 3DES, AES относиться к симметричным алгоритмам и реализация в System.Security.Cryptography не позволяет дешифровать данные ключом, отличным от использованного для шифрования. При попытке сделать это, будет выдано исключение.

1. **RSA**

Асимметричный алгоритм шифрования.

RSA-ключи генерируются следующим образом[[16]](https://ru.wikipedia.org/wiki/RSA#cite_note-_0ca773cfe4c06ed0-16):

1) выбираются два различных случайных простых числа p{\displaystyle p} и q{\displaystyle q} заданного размера (например, 1024 бита каждое);

2) вычисляется их произведение n = p \* q {\displaystyle n=p\cdot q}, которое называется *модулем*;

3) вычисляется значение функции Эйлера от числа n{\displaystyle n}:

{\displaystyle \varphi (n)=(p-1)\cdot (q-1)};

4) выбирается целое число e{\displaystyle e} (1 < e < {\displaystyle 1<e<\varphi (n)}), взаимно простое со значением функции {\displaystyle \varphi (n)};

число e{\displaystyle e} называется *открытой экспонентой*;

обычно в качестве e{\displaystyle e} берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые из чисел Ферма: 17, 257 или 65537, так как в этом случае время, необходимое для шифрования с использованием быстрого возведения в степень будет меньше;

слишком малые значения e{\displaystyle e}, например 3, потенциально могут ослабить безопасность схемы RSA;

5) вычисляется число d{\displaystyle d}, мультипликативно обратное к числу e{\displaystyle e} по модулю {\displaystyle \varphi (n)}, то есть число, удовлетворяющее сравнению;{\displaystyle d\cdot e\equiv 1{\pmod {\varphi (n)}}}

(число d {\displaystyle d}называется *секретной экспонентой*; обычно оно вычисляется при помощи расширенного алгоритма Евклида);

6) пара (e; n){\displaystyle (e,n)} публикуется в качестве *открытого ключа RSA*;

7) пара (d; n){\displaystyle (d,n)} играет роль *закрытого ключа RSA* и держится в секрете.

**Алгоритм шифрования**:

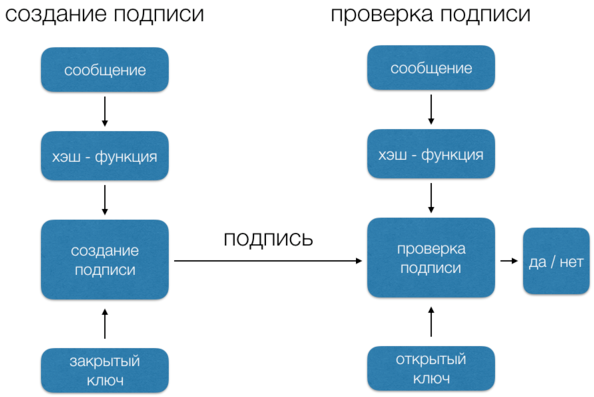
* Взять *открытый ключ* {\displaystyle (e,n)}(e; n){\displaystyle (e,n)}
* Взять *открытый текст* {\displaystyle m}m
* Зашифровать сообщение с использованием открытого ключа:

**Алгоритм расшифрования**:

* Принять зашифрованное сообщение c {\displaystyle c}
* Взять свой *закрытый ключ* {\displaystyle (d,n)}(d; n){\displaystyle (d,n)}
* Применить закрытый ключ для расшифрования сообщения: {\displaystyle c=E(m)=m^{e}\mod n~~~~(1)} m

1. **DSA**

Алгоритм для создания цифровой подписи с открытым ключом. Не используется для шифрования. Созданная подпись может быть проверена публично.

Схема работы алгоритма выглядит следующим образом: 

По сути, DSA подписывает хэш-функция от сообщения.

Для вычисления хэш-функции можно использовать любой алгоритм.

По умолчанию, в библиотечном DSA используется SHA1.

Открытый и секретный ключи:

1. Секретный ключ представляет собой число {\displaystyle x\in (0,q)}
2. Открытый ключ вычисляется по формуле {\displaystyle y=g^{x}\mod p}

Особенности библиотечного DSA: помимо использования SHA1, необходимо дополнять подаваемо сообщение до 8 байт, если оно меньше, так как SHA1 ограничен минимальным размером сообщения в 8 байт.

1. **ECDSA**

Алгоритм для создания цифровой подписи с открытым ключом, похожий на DSA, но определённый в группе эллиптических точек, что повышает его надёжность.

Параметры алгоритма

1. Выбор хеш-функции {\displaystyle H(x)}H(x). Для использования алгоритма необходимо, чтобы подписываемое сообщение являлось числом. Хеш-функция должна преобразовать любое сообщение в последовательность битов, которые можно потом преобразовать в число.
2. Выбор большого простого числа {\displaystyle q}q — порядок одной из циклических подгрупп группы точек эллиптической кривой.

*Замечание*: Если размерность этого числа в битах меньше размерности в битах значений хеш-функции H(x){\displaystyle H(x)} то используются только левые биты значения хеш-функции.

1. Простым числом {\displaystyle p}p обозначается характеристика поля координат {\displaystyle F\_{p}}Fp.

Создание ключей:

Для простоты будем рассматривать эллиптические кривые над полем Fp{\displaystyle F\_{p}}, где Fp{\displaystyle F\_{p}} —конечное простое поле. Причем, если необходимо, конструкцию можно легко адаптировать для эллиптических кривых над другим полем.

Пусть E{\displaystyle E} — эллиптическая кривая, определенная над Fp {\displaystyle F\_{p}}, и{\displaystyle P} P — точка простого порядка q{\displaystyle q} кривой E(Fp){\displaystyle E(F\_{p})}. Кривая E{\displaystyle E} и точка P{\displaystyle P} являются системными параметрами. Число p{\displaystyle p} — простое. Каждый пользователь — условно назовём его A — конструирует свой ключ посредством следующих действий:

1. Выбирает случайное или псевдослучайное целое число {\displaystyle x}x из интервала [1; q - 1] {\displaystyle [1,q-1]}.
2. Вычисляет произведение (кратное) {\displaystyle Q} = {\displaystyle x}{\displaystyle P}Q = x \* P.

Открытым ключом пользователя A{\displaystyle A} является точка Q{\displaystyle Q}, а закрытым — x{\displaystyle x}.

Вычисление цифровой подписи:

Для того, чтобы подписать какое-либо сообщение, для которого подсчитано значение h{\displaystyle h} хеш-функции H{\displaystyle H}, пользователь A{\displaystyle A} должен сделать следующее:

1. Выбрать случайное целое число {\displaystyle k\in [1,q-1]} .
2. Вычислить {\displaystyle k\cdot P=(x\_{1},y\_{1})} и положить в {\displaystyle r=x\_{1}\mod q}

.

1. Вычислить {\displaystyle k^{-1}{\pmod {q}}} и положить {\displaystyle s=k^{-1}(h+x\cdot r)\mod q}, где *h* — значение хеш-функции подписываемого сообщения.

Подписью для сообщения является пара целых чисел (r, s){\displaystyle (r,s)}.

Проверка цифровой подписи:

Для того, чтобы проверить подпись пользователя А (r, s){\displaystyle (r,s)} на сообщение, пользователь Б {\displaystyle B}должен сделать следующее:

1. Получить подтвержденную копию открытого ключа {\displaystyle Q}Q пользователя А;
2. Проверить, что числа {\displaystyle r}r и {\displaystyle s}s являются целыми числами из интервала {\displaystyle [1,q-1]}[1, q - 1], и вычислить значение хеш-функции {\displaystyle h}h от сообщения;
3. Вычислить {\displaystyle u\_{1}=s^{-1}h\mod q} и {\displaystyle u\_{2}=s^{-1}r\mod q};
4. Вычислить {\displaystyle u\_{1}\cdot P+u\_{2}\cdot Q=(x\_{0},y\_{0})}, и относительно {\displaystyle x\_{0}}, как целого числа между {\displaystyle 0}0 и {\displaystyle (p-1)}(p - 1), положить {\displaystyle v=x\_{0}\mod q}
5. Принять подпись, тогда и только тогда, когда {\displaystyle v=r}𝑣 = r.

Заметим, что, если пользователь А вычислил свою подпись правильно, то {\displaystyle u\_{1}P+u\_{2}Q=(u\_{1}+xu\_{2})P=(s^{-1}\cdot h\mod q+x\cdot s^{-1}\cdot r\mod q)\cdot P=k\cdot P}, так как {\displaystyle k=s^{-1}(h+xr){\pmod {q}}}, и поэтому {\displaystyle v=r}.

Для подтверждения публичного ключа Q нужно проделать следующее ({\displaystyle O} здесь обозначает бесконечно удалённую точку):

1. Проверить, что {\displaystyle Q}Q не равно {\displaystyle O}O и координаты верны;
2. Проверить, что {\displaystyle Q}Q лежит на кривой;
3. Проверить, что qQ = O{\displaystyle qQ=O};
4. **HMACSHA512**

Алгоритм, позволяющий проверить, изменилось ли сообщение. Использует в качестве цифровой подписи хэш-функцию от сообщения. В данной версии, выбран алгоритм, основанный на SHA512.

Алгоритм HMAC можно записать в виде одной формулы:

H – какая-то хэш-функция (в данном случае, SHA512),

*|| -* конкитенация строк.

1. **SHA512**

Представитель семейства криптографически безопасных хэш-функций. Отличается от SHA256 следующим:

* слова имеют длину 64 бита,
* используется 80 раундов вместо 64,
* сообщение разбито на чанки по 1024 бит,
* начальные значения переменных и константы расширены до 64 бит,
* постоянные для каждого из 80 раундов — 80 первых простых чисел,
* 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

*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 := 0x6a09e667f3bcc908,

h1 := 0xbb67ae8584caa73b,

h2 := 0x3c6ef372fe94f82b,

h3 := 0xa54ff53a5f1d36f1,

h4 := 0x510e527fade682d1,

h5 := 0x9b05688c2b3e6c1f,

h6 := 0x1f83d9abfb41bd6b,

h7 := 0x5be0cd19137e2179

*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 ǁ [*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 c 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 ǁ h1 ǁ h2 ǁ h3 ǁ h4 ǁ h5 ǁ h6 ǁ h7

1. **RIPEMD160**

Криптографическая хэш-функция. Генерирует 160-разрядное хэш-значение – дайджест.

*Шаг 1. Добавление недостающих битов*

Сообщение расширяется так, чтобы его длина в битах по модулю 512 равнялась 448. Таким образом, в результате расширения, сообщению недостает 64 бита до длины, кратной 512 битам. Расширение производится всегда, даже если сообщение изначально имеет нужную длину.

Расширение производится следующим образом: один бит, равный 1, добавляется к сообщению, а затем добавляются биты, равные 0, до тех пор, пока длина сообщения не станет равной 448 по модулю 512. В итоге, к сообщению добавляется, как минимум, 1 бит, и, как максимум, 512.

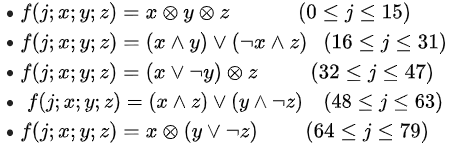
*Шаг 2. Добавление длины сообщения*

64-битное представление b{\displaystyle b} (длины сообщения перед добавлением набивочных битов) добавляется к результату предыдущего шага. В маловероятном случае, когда b{\displaystyle b} больше, чем 264{\displaystyle 2^{64}}, используются только 64 младших бита. Эти биты добавляются в виде двух 32-битных слов, и первым добавляется слово, содержащее младшие разряды.

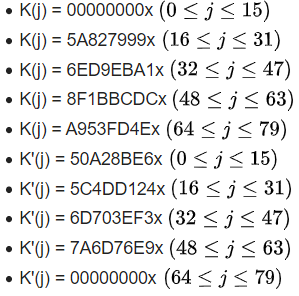
На этом этапе (после добавления битов и длины сообщения) мы получаем сообщение длиной кратной 512 битам. Это эквивалентно тому, что это сообщение имеет длину, кратную 16-ти 32-битным словам. Каждое 32-битное слово содержит четыре 8-битных, но следуют они не подряд, а наоборот (например, из восьми 8-битных слов (a b c d e f g h) мы получаем два 32-битных слова (dcba hgfe)).

*Шаг 3. Определение действующих функций и констант*

I. Нелинейные побитовые функции:



II. Добавляемые шестнадцатеричные константы:



III. Выбор 32-битных слов из сообщения:

* r(j) = j при {\displaystyle (0\leq j\leq 15)}
* r(16..31) = 7; 4; 13; 1; 10; 6; 15; 3; 12; 0; 9; 5; 2; 14; 11; 8
* r(32..47) = 3; 10; 14; 4; 9; 15; 8; 1; 2; 7; 0; 6; 13; 11; 5; 12
* r(48..63) = 1; 9; 11; 10; 0; 8; 12; 4; 13; 3; 7; 15; 14; 5; 6; 2
* r(64..79) = 4; 0; 5; 9; 7; 12; 2; 10; 14; 1; 3; 8; 11; 6; 15; 13
* r'(0..15) = 5; 14; 7; 0; 9; 2; 11; 4; 13; 6; 15; 8; 1; 10; 3; 12
* r'(16..31) = 6; 11; 3; 7; 0; 13; 5; 10; 14; 15; 8; 12; 4; 9; 1; 2
* r'(32..47) = 15; 5; 1; 3; 7; 14; 6; 9; 11; 8; 12; 2; 10; 0; 4; 13
* r'(48..63) = 8; 6; 4; 1; 3; 11; 15; 0; 5; 12; 2; 13; 9; 7; 10; 14
* r'(64..79) = 12; 15; 10; 4; 1; 5; 8; 7; 6; 2; 13; 14; 0; 3; 9; 11

VI. Набор для битового поворота влево (операция rol):

* s(0..15) = 11; 14; 15; 12; 5; 8; 7; 9; 11; 13; 14; 15; 6; 7; 9; 8
* s(16..31) = 7; 6; 8; 13; 11; 9; 7; 15; 7; 12; 15; 9; 11; 7; 13; 12
* s(32..47) = 11; 13; 6; 7; 14; 9; 13; 15; 14; 8; 13; 6; 5; 12; 7; 5
* s(48..63) = 11; 12; 14; 15; 14; 15; 9; 8; 9; 14; 5; 6; 8; 6; 5; 12
* s(64..79) = 9; 15; 5; 11; 6; 8; 13; 12; 5; 12; 13; 14; 11; 8; 5; 6
* s'(0..15) = 8; 9; 9; 11; 13; 15; 15; 5; 7; 7; 8; 11; 14; 14; 12; 6
* s'(16..31) = 9; 13; 15; 7; 12; 8; 9; 11; 7; 7; 12; 7; 6; 15; 13; 11
* s'(32..47) = 9; 7; 15; 11; 8; 6; 6; 14; 12; 13; 5; 14; 13; 13; 7; 5
* s'(48..63) = 15; 5; 8; 11; 14; 14; 6; 14; 6; 9; 12; 9; 12; 5; 15; 8
* s'(64..79) = 8; 5; 12; 9; 12; 5; 14; 6; 8; 13; 6; 5; 15; 13; 11; 11

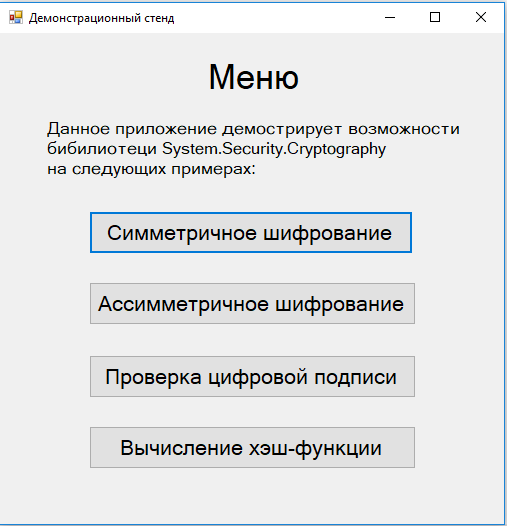
V. Исходные значения слов дайджеста:

* h0 = 67452301x;
* h1 = EFCDAB89x;
* h2 = 98BADCFEx;
* h3 = 10325476x;
* h4 = C3D2E1F0x;

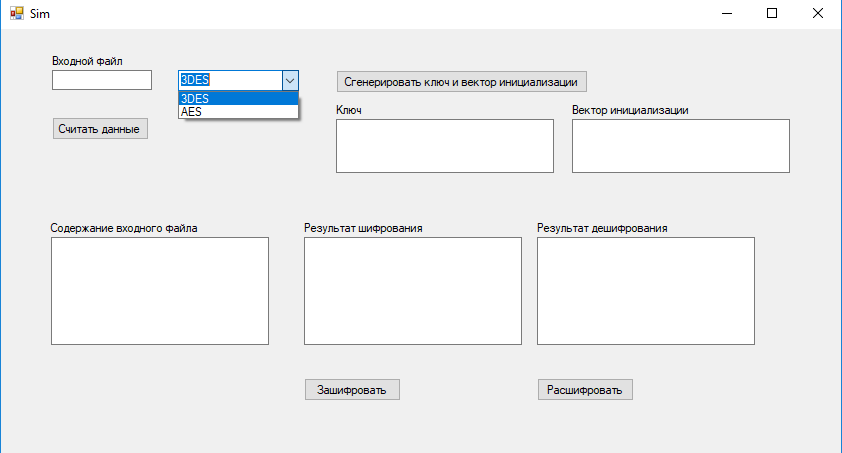
*Шаг 4. Выполнение алгоритма хеширования*

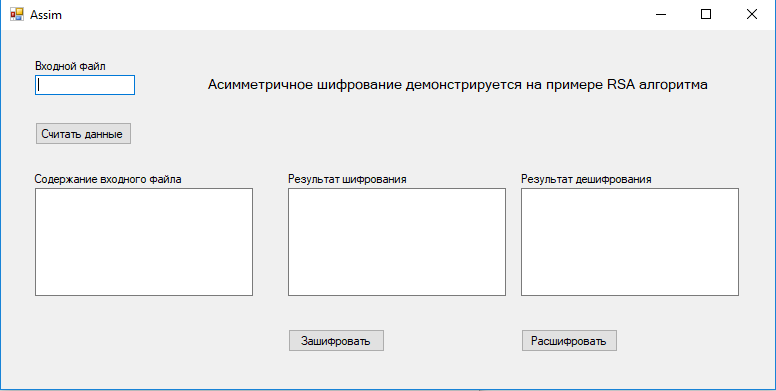
После задания всех исходных функций, констант и начальных значений слов хеш-суммы можно переходить к выполнению алгоритма. Выполнение алгоритма происходит по двум параллельным путям. Обработка сообщения происходит блоками по 16 слов в 32 бита.

Разработанное программное средство способно считывать необходимые данные из файлов и записывать результаты в выходные файлы. Так же разработанное приложение способно учитывать некорректные данные.

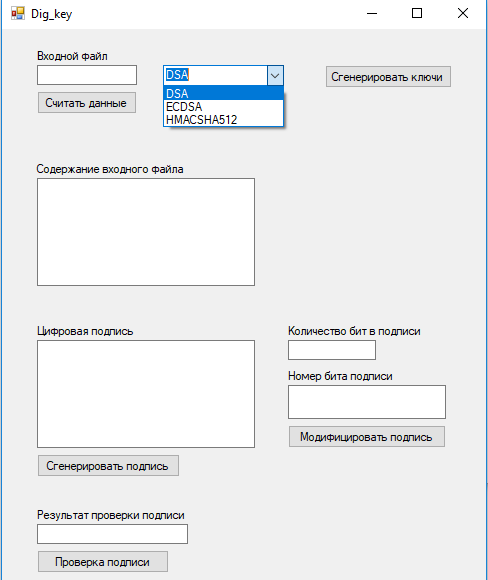
Основное меню:  


Симметричное шифрование:

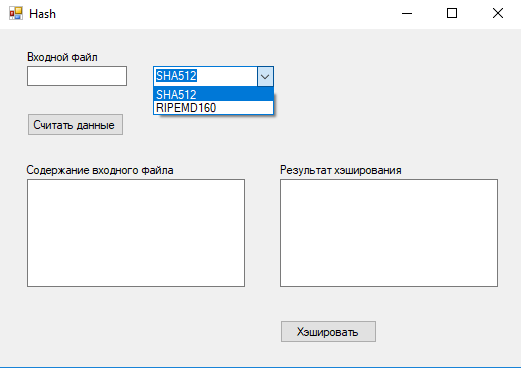


Асимметричное шифрование:  


Цифровые подписи:

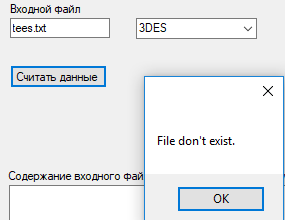


Хэш-функции:

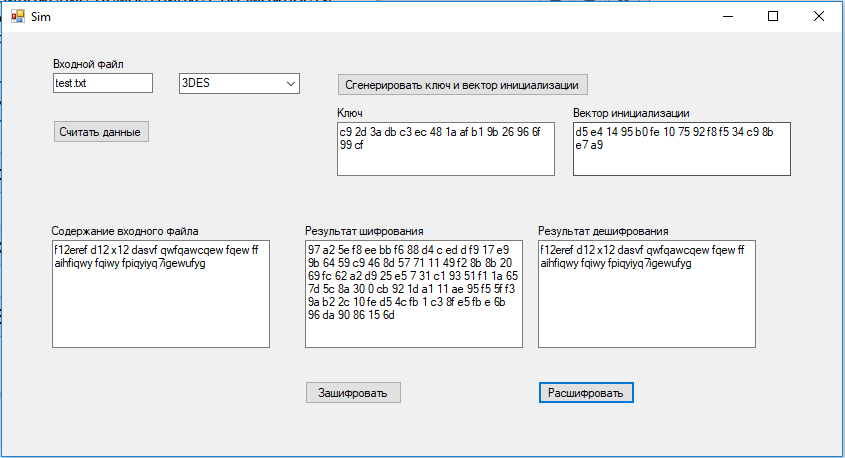


1. Исследования
   1. Симметричное шифрование.

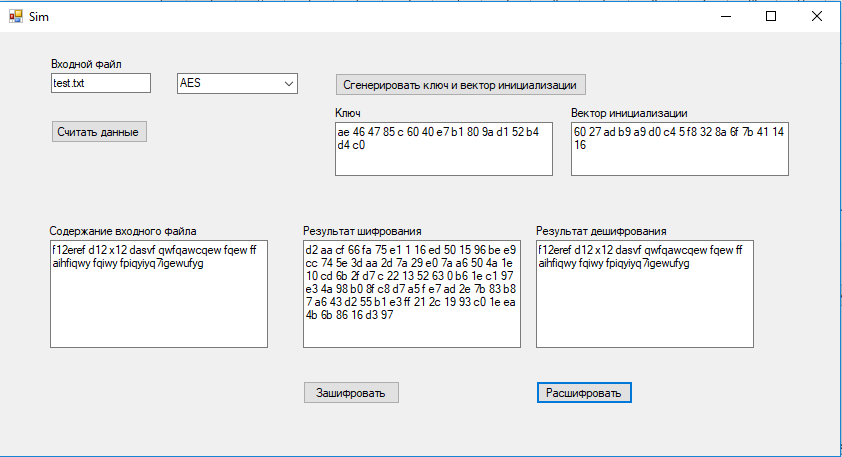
Пример некорректных данных:



Демонстрация 3DES:

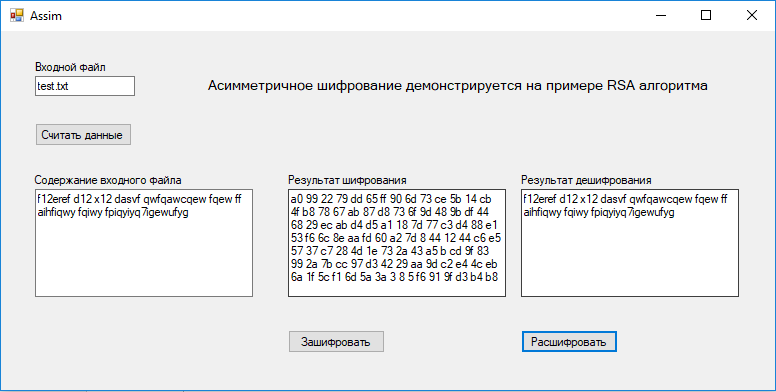


Демонстрация AES:



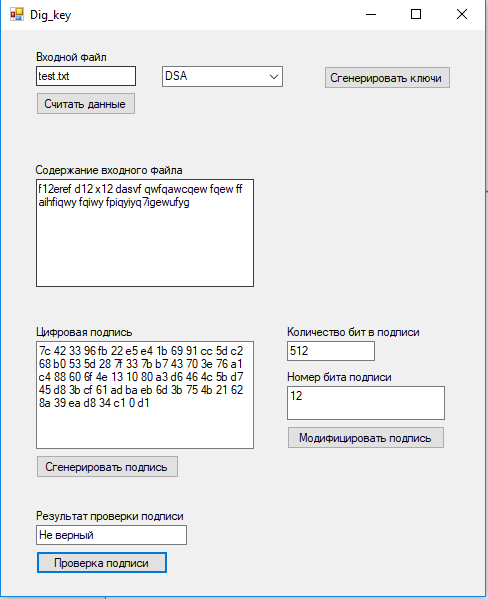
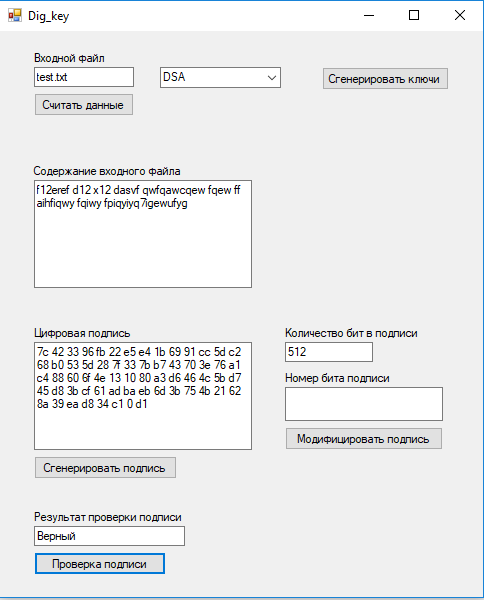
* 1. Асимметричное шифрование.

Демонстрация RSA:

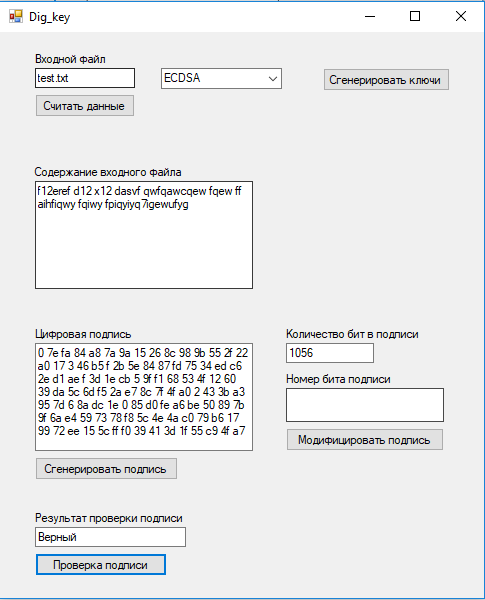
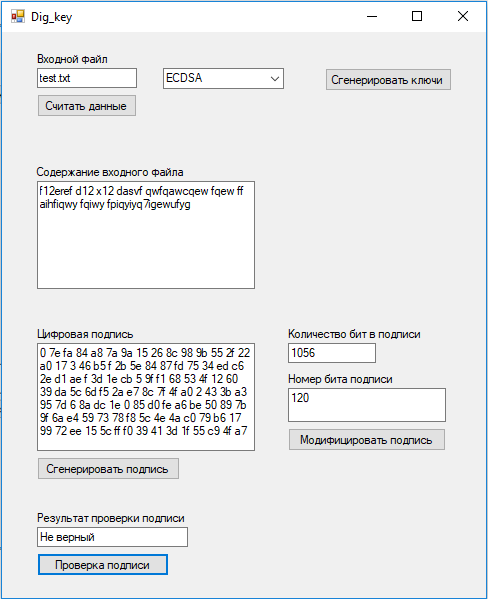


* 1. Цифровые подписи.

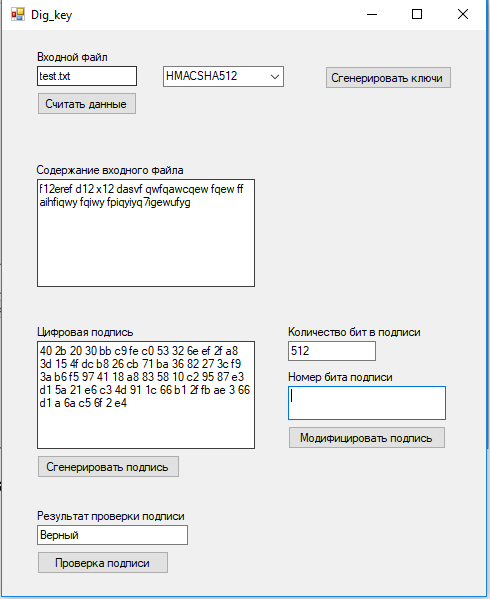
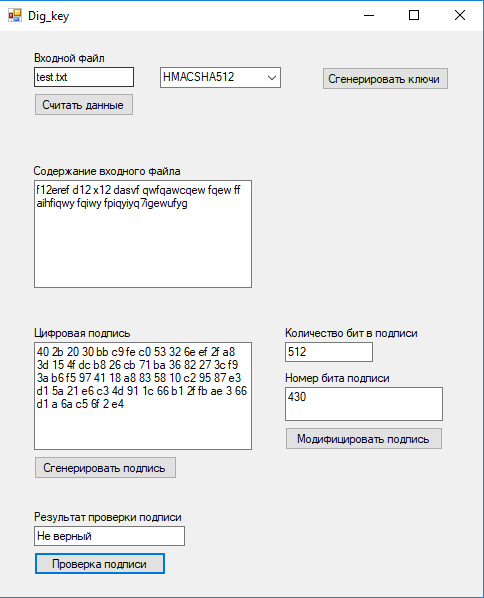
Демонстрация DSA c модификацией подписи:



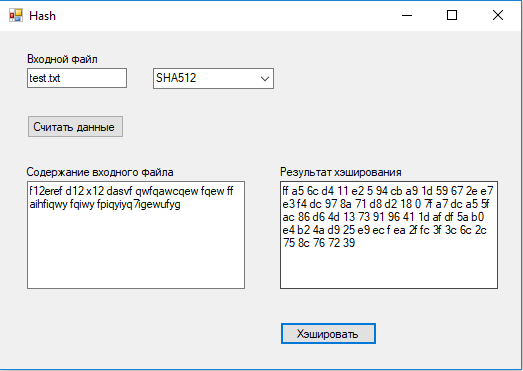
Демонстрация ECDSA c модификацией подписи:

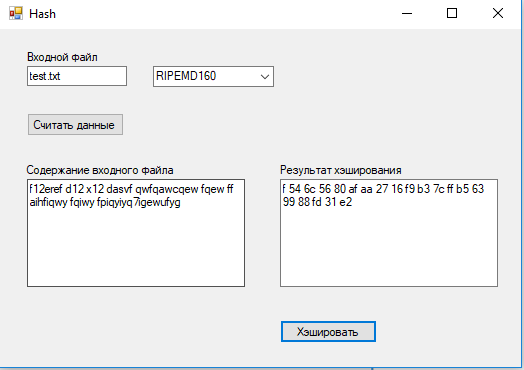
Демонстрация HMACSHA512 c модификацией подписи:

* 1. Хэш-функции.

SHA512:  


RIPEMD160:



1. **Код программы**

Sym\_alg.cs

class Sym\_alg

{

public byte[] key\_gen() // генератор 128-битного ключа

{

byte[] key = new byte[128]; // основной ключ

byte[] b1 = new byte[64]; // 64-битная часть ключа

byte[] b2 = new byte[64]; // 64-битная часть ключа

byte[] xor = new byte[64]; // 64-битная блок случайных бит для равномерного преобразования ключа

byte[] bit = new byte[1]; // случайный бит для выбора совмещения двух частей ключа (1 - !b1 + b2, 0 - b2 + !b1)

Thread.Sleep(20); // тормозим генерацию ключа, чтобы новый ключ был случайным

var rand = new Random((int)DateTime.Now.Ticks & 0x0000FFFF);

rand.NextBytes(b1); // создаём случайную половину ключа

rand.NextBytes(b2); // создаём случайную половину ключа

rand.NextBytes(xor); // создаём случайную последовательность для оперции xor с половинами ключа

ulong k1 = BitConverter.ToUInt64(b1, 0); // переводим в ulong формат

ulong k2 = BitConverter.ToUInt64(b2, 0); // переводим в ulong формат

ulong xr = BitConverter.ToUInt64(xor, 0); // переводим в ulong формат

k1 ^= xr; // приводим к равномерному виду

xr = BitConverter.ToUInt64(xor, 0);

k2 ^= xr; // приводим к равномерному виду

rand.NextBytes(bit); // создаём случайное число (0 или 1), чтобы выбрать, какую половину ключа поставить первой и какую инвертировать

if (bit[0] == 1)

{

b1 = BitConverter.GetBytes(~k1); // обрачаем число

b2 = BitConverter.GetBytes(k2);

key = b1.Concat(b2).ToArray(); // соединяем половины 128-битного ключа

}

else

{

b1 = BitConverter.GetBytes(k1);

b2 = BitConverter.GetBytes(~k2); // обрачаем число

key = b2.Concat(b1).ToArray(); // соединяем половины 128-битного ключа

}

return key;

}

public byte[] TriDES\_Encrypt(byte[] Data, byte[] Key, byte[] IV) // 3DES шифратор

{

MemoryStream mStream = new MemoryStream(); // создаём поток памяти

TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider(); // объект класса 3DES

//tdes.Padding = PaddingMode.None; //отключаем дополнение выходного шифротекства справа

tdes.Mode = CipherMode.CFB; // включаепм режим CFB, чтобы установить длину шифротекста в 64 бита

CryptoStream cStream = new CryptoStream(mStream,

tdes.CreateEncryptor(Key, IV),

CryptoStreamMode.Write); // создаём поток для шифрования

byte[] toEncrypt = Data;

cStream.Write(toEncrypt, 0, toEncrypt.Length); // помещаем данные в поток шифрования

cStream.FlushFinalBlock();

byte[] ret = mStream.ToArray(); // извлекаем шифротекст

cStream.Close(); // закрываем потоки

mStream.Close();

return ret;

}

public byte[] TriDES\_Decrypt(byte[] Data, byte[] Key, byte[] IV) // 3DES дешифратор

{

MemoryStream mStream = new MemoryStream(); // создаём поток памяти

TripleDESCryptoServiceProvider tdes = new TripleDESCryptoServiceProvider(); // объект класса 3DES

//tdes.Padding = PaddingMode.None; //отключаем дополнение выходного шифротекства справа

tdes.Mode = CipherMode.CFB; // включаепм режим CFB, чтобы установить длину шифротекста в 64 бита

CryptoStream cStream = new CryptoStream(mStream,

tdes.CreateDecryptor(Key, IV),

CryptoStreamMode.Write); // создаём поток для дешифрования

byte[] toEncrypt = Data;

cStream.Write(toEncrypt, 0, toEncrypt.Length); // помещаем данные в поток шифрования

cStream.FlushFinalBlock();

byte[] ret = mStream.ToArray(); // извлекаем текст

cStream.Close(); // закрываем потоки

mStream.Close();

return ret;

}

public byte[] AES\_Encrypt(byte[] Data, byte[] Key, byte[] IV) // AES шифратор

{

MemoryStream mStream = new MemoryStream(); // создаём поток памяти

Aes AES = Aes.Create(); // объект класса AES

AES.Mode = CipherMode.CFB; // включаепм режим CFB, чтобы установить длину шифротекста в 64 бита

CryptoStream cStream = new CryptoStream(mStream,

AES.CreateEncryptor(Key, IV),

CryptoStreamMode.Write); // создаём поток для шифрования

byte[] toEncrypt = Data;

cStream.Write(toEncrypt, 0, toEncrypt.Length); // помещаем данные в поток шифрования

cStream.FlushFinalBlock();

byte[] ret = mStream.ToArray(); // извлекаем шифротекст

cStream.Close(); // закрываем потоки

mStream.Close();

return ret;

}

public byte[] AES\_Decrypt(byte[] Data, byte[] Key, byte[] IV) // AES дешифратор

{

MemoryStream mStream = new MemoryStream(); // создаём поток памяти

Aes AES = Aes.Create(); // объект класса 3DES

AES.Mode = CipherMode.CFB; // включаепм режим CFB, чтобы установить длину шифротекста в 64 бита

CryptoStream cStream = new CryptoStream(mStream,

AES.CreateDecryptor(Key, IV),

CryptoStreamMode.Write); // создаём поток для дешифрования

byte[] toEncrypt = Data;

cStream.Write(toEncrypt, 0, toEncrypt.Length); // помещаем данные в поток дешифрования

cStream.FlushFinalBlock();

byte[] ret = mStream.ToArray(); // извлекаем текст

cStream.Close(); // закрываем потоки

mStream.Close();

return ret;

}

}

Sym.cs

public partial class Sym : Form

{

public Sym()

{

InitializeComponent();

comboBox1.Items.Add("3DES");

comboBox1.Items.Add("AES");

comboBox1.SelectedIndex = 0;

}

byte[] b\_mess = new byte[1]; // сообщение в байтах

string[] mess = new string[1]; // сообщение в строках

string alg\_flag = ""; // название алгоритма

byte[] encypt = new byte[1]; // шифротекст

byte[] decypt = new byte[1]; // дешифрованное сообщение

byte[] KEY = new byte[1]; // ключ

byte[] IV = new byte[1]; // вектор инициализации

private void Button3\_Click(object sender, EventArgs e)

{

File\_f F = new File\_f();

string file\_name = textBox1.Text;

if (file\_name == "") // проверка на ввод имени файла

MessageBox.Show("Enter file name.");

else

{

bool fl = File.Exists(file\_name); // проверка на существование файла

if (fl)

{

b\_mess = File.ReadAllBytes(file\_name);

mess = F.Read(file\_name);

string ms = "";

for (int i = 0; i < mess.Length; i++)

ms += mess[i];

textBox2.Text = ms;

}

else

MessageBox.Show("File don't exist.");

}

}

private void Button1\_Click(object sender, EventArgs e) // шифруем сообщения

{

textBox3.Clear();

string file\_name = "";

alg\_flag = comboBox1.SelectedItem.ToString();

Sym\_alg SM = new Sym\_alg();

KEY = File.ReadAllBytes("key.bin");

IV = File.ReadAllBytes("iv.bin");

if (alg\_flag == "3DES")

{

encypt = SM.TriDES\_Encrypt(b\_mess, KEY, IV);

file\_name = "enc\_3DES.bin";

}

if (alg\_flag == "AES")

{

encypt = SM.AES\_Encrypt(b\_mess, KEY, IV);

file\_name = "enc\_AES.bin";

}

File.WriteAllBytes(file\_name, encypt);

for (int i = 0; i < encypt.Length; i++)

textBox3.Text += encypt[i].ToString("x") + " ";

}

private void Button2\_Click(object sender, EventArgs e) // дешируем сообщения

{

textBox4.Clear();

Sym\_alg SM = new Sym\_alg();

string file\_name = "";

if (alg\_flag == "3DES")

{

decypt = SM.TriDES\_Decrypt(encypt, KEY, IV);

file\_name = "dec\_3DES.txt";

}

if (alg\_flag == "AES")

{

decypt = SM.AES\_Decrypt(encypt, KEY, IV);

file\_name = "dec\_AES.txt";

}

string dec\_res = Encoding.UTF8.GetString(decypt, 0, decypt.Length);

File.WriteAllText(file\_name, dec\_res);

textBox4.Text = dec\_res;

}

private void Button4\_Click(object sender, EventArgs e) // генератор ключей

{

string file\_key = "key.bin";

string file\_IV = "IV.bin";

textBox5.Clear();

textBox6.Clear();

Sym\_alg SM = new Sym\_alg();

byte[] k = SM.key\_gen();

byte[] iv = SM.key\_gen();

for (int i = 0; i < k.Length; i++)

textBox5.Text += k[i].ToString("x") + " ";

for (int i = 0; i < iv.Length; i++)

textBox6.Text += iv[i].ToString("x") + " ";

File.WriteAllBytes(file\_key, k);

File.WriteAllBytes(file\_IV, iv);

}

}

Asym\_alg.cs

class Asym\_alg

{

public RSAParameters RSA\_KEY\_GEN() // генератор ключей для RSA алкоритма

{

RSACryptoServiceProvider RSA\_key = new RSACryptoServiceProvider();

return RSA\_key.ExportParameters(true);

}

public byte[] RSA\_Encrypt(byte[] data, RSAParameters RSA\_key) // шифратор сообщения

{

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

byte[] res = new byte[1];

RSA.ImportParameters(RSA\_key); // получаем ключи

res = RSA.Encrypt(data, true); // шифруем сообщение

return res;

}

public byte[] RSA\_Decrypt(byte[] data, RSAParameters RSA\_key) // дешифратор сообщения

{

RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();

byte[] res = new byte[1];

RSA.ImportParameters(RSA\_key); // получаем ключ

res = RSA.Decrypt(data, true); // дешифруем сообщение

return res;

}

}

Asym.cs

public partial class Asym : Form

{

public Asym()

{

InitializeComponent();

}

RSAParameters RSA\_key = new RSAParameters();

byte[] b\_mess = new byte[1]; // сообщение в байтах

string[] mess = new string[1]; // сообщение в строках

byte[] enc = new byte[1]; // шифротекст

private void button3\_Click(object sender, EventArgs e)

{

textBox2.Clear();

File\_f F = new File\_f();

string file\_name = textBox1.Text;

if (file\_name == "") // проверка на ввод имени файла

MessageBox.Show("Enter file name.");

else

{

bool fl = File.Exists(file\_name); // проверка на существование файла

if (fl)

{

b\_mess = File.ReadAllBytes(file\_name);

mess = F.Read(file\_name);

string ms = "";

for (int i = 0; i < mess.Length; i++)

ms += mess[i];

textBox2.Text = ms;

}

else

MessageBox.Show("File don't exist.");

}

}

private void Button1\_Click(object sender, EventArgs e) // шифруем сообщение

{

textBox3.Clear();

Asym\_alg AG = new Asym\_alg();

RSA\_key = AG.RSA\_KEY\_GEN();

enc = AG.RSA\_Encrypt(b\_mess, RSA\_key);

for (int i = 0; i < enc.Length; i++)

textBox3.Text += enc[i].ToString("x") + " ";

}

private void Button2\_Click(object sender, EventArgs e) // дешифруем сообщение

{

textBox4.Clear();

Asym\_alg AG = new Asym\_alg();

byte[] dec = AG.RSA\_Decrypt(enc, RSA\_key);

string dec\_res = Encoding.UTF8.GetString(dec, 0, dec.Length);

textBox4.Text = dec\_res;

}

Dig\_key\_alg.cs

class Dig\_key\_alg // демонстрационный класс для создания цифровых подписей

{

public void Key\_gen(string Alg\_name, string key\_file) // функция генерации ключей (ТОЛЬКО ДЛЯ DSA И HMACSHA512)

{

if(Alg\_name == "DSA")

{

DSACng DS = new DSACng(); // создаём объект класса DSACng

string param = DS.ToXmlString(true); // экспортируем открытый и закрытый ключи в строку XML формата

using (StreamWriter writer = new StreamWriter(File.Open(key\_file, FileMode.Create))) // записываем строку с файл key\_file

writer.Write(param);

}

if (Alg\_name == "HMACSHA512")

{

byte[] key = new byte[64];

RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider(); // создаём объект специального класса для создания ключей

rng.GetBytes(key); // создаём 64 случайных байта ключа

File.WriteAllBytes(key\_file, key); // записываем ключ в файл

}

}

public byte[] DSA\_Sig\_Create(byte[] Data, string para\_f) // функция для создания DSA подписи

{

byte[] Sig = new byte[1]; // подпись

string param = "";

using (StreamReader reader = new StreamReader(File.Open(para\_f, FileMode.Open))) // считываем параметры из файла

param = reader.ReadToEnd();

DSACng DS = new DSACng(); // объект класса DSACng

DS.FromXmlString(param); // импортируем параметры

Sig = DS.CreateSignature(Data); // создаём подпись

return Sig;

}

public bool DSA\_Sig\_Ver(byte[] Data, byte[] Signature, string para\_f) // фукнция для проверки DSA подписи

{

string param = "";

using (StreamReader reader = new StreamReader(File.Open(para\_f, FileMode.Open))) // считываем параметры из файла

param = reader.ReadToEnd();

DSACng DS = new DSACng();

DS.FromXmlString(param); // импортируем параметры

if (DS.VerifySignature(Data, Signature)) // проверяем подпись

return true;

else

return false;

}

public byte[] bit\_mod(string n\_bit, byte[] Sig) // функция модификации подписи (меняет указанный номер бита в подписи)

{

int i = Convert.ToInt32(n\_bit); // извлекаем номер бита

BitArray sig\_m = new BitArray(Sig); // преобразуем массив байтов в массив битов

if (sig\_m[i] == true) // инвертируем бит

sig\_m[i] = false;

else

sig\_m[i] = true;

byte[] ou = new byte[(sig\_m.Length - 1) / 8 + 1]; // преобразуем из битов в байты

sig\_m.CopyTo(ou, 0);

return ou;

}

public byte[] ECDSA\_Sig\_Create(byte[] Data, string para\_f) // функция создания ECDSA подписи

{

byte[] Sig = new byte[1];

ECDsaCng ECD = new ECDsaCng(); // создаём объект основного класса

ECD.HashAlgorithm = CngAlgorithm.Sha256; // указываем алгоритм, на основе которого будем создавать подпись

File.WriteAllBytes(para\_f, ECD.Key.Export(CngKeyBlobFormat.EccPublicBlob)); // экспортируем ключи в файл (необходимо, для правильной проверки подписи)

Sig = ECD.SignData(Data); // создаём подпись

return Sig;

}

public bool ECDSA\_Sig\_Ver(byte[] Data, byte[] Signature, string para\_f) // функция проверки ECDSA подписи

{

byte[] key = File.ReadAllBytes(para\_f); // получаем ключ из файла

ECDsaCng ecsdKey = new ECDsaCng(CngKey.Import(key, CngKeyBlobFormat.EccPublicBlob)); // создаём объект с заданным ключом

if (ecsdKey.VerifyData(Data, Signature)) // проверяем подпись

return true;

else

return false;

}

public byte[] HMAC\_Sig\_Create(byte[] Data, string para\_f) // функция для создания HMAC подписи

{

byte[] Sig = new byte[1];

HMACSHA512 hmac = new HMACSHA512(File.ReadAllBytes(para\_f)); // считываем параметры из файла и создаём на их основе объект класса HMACSHA512 (HMAC на основе SHA512)

Sig = hmac.ComputeHash(Data); // создаём подпись (подписью является хэш)

return Sig;

}

public bool HMAC\_Sig\_Ver(byte[] Data, byte[] Signature, string para\_f) // функция проверки HMAC подписи

{

bool ver = true;

HMACSHA512 hmac = new HMACSHA512(File.ReadAllBytes(para\_f)); // создаём объект класса на основе параметров из файла

byte[] ver\_sig = hmac.ComputeHash(Data); // создаём подпись (хэш)

for (int i = 0; i < ver\_sig.Length; i++) // сравниваем хэши (должны быть равны)

if (Signature[i] != ver\_sig[i])

ver = false;

return ver;

}

}

Dig\_key.cs

public partial class Dig\_key : Form

{

public Dig\_key()

{

InitializeComponent();

comboBox1.Items.Add("DSA");

comboBox1.Items.Add("ECDSA");

comboBox1.Items.Add("HMACSHA512");

comboBox1.SelectedIndex = 0;

}

byte[] b\_mess = new byte[1]; // байты сообщения

string[] mess = new string[1]; // строка с сообщением

string alg\_flag = ""; // название алгоритма

string para\_file\_name = "\_Paramet.txt"; // левая часть названия файла с параметрами (правой является название алгоритма)

string out\_file\_name = "\_out.txt"; // левая часть названия файла с подписью (правой является название алгоритма)

byte[] sig = new byte[1]; // подпись

private void button1\_Click(object sender, EventArgs e)

{

File\_f F = new File\_f();

textBox2.Clear();

string file\_name = textBox1.Text;

if (file\_name == "") // проверка на ввод имени файла

MessageBox.Show("Enter file name.");

else

{

bool fl = File.Exists(file\_name); // проверка на существование файла

if (fl)

{

b\_mess = File.ReadAllBytes(file\_name);

mess = F.Read(file\_name);

string ms = "";

for (int i = 0; i < mess.Length; i++)

ms += mess[i];

textBox2.Text = ms;

}

else

MessageBox.Show("File don't exist.");

}

}

private void Button3\_Click(object sender, EventArgs e) // создание подписи

{

textBox3.Clear();

Dig\_key\_alg DKA = new Dig\_key\_alg();

alg\_flag = comboBox1.SelectedItem.ToString();

if (alg\_flag == "DSA")

{

sig = DKA.DSA\_Sig\_Create(b\_mess, alg\_flag + para\_file\_name);

for (int i = 0; i < sig.Length; i++)

textBox3.Text += sig[i].ToString("x") + " ";

textBox6.Text = (sig.Length \* 8).ToString();

}

if (alg\_flag == "ECDSA")

{

sig = DKA.ECDSA\_Sig\_Create(b\_mess, alg\_flag + para\_file\_name);

for (int i = 0; i < sig.Length; i++)

textBox3.Text += sig[i].ToString("x") + " ";

textBox6.Text = (sig.Length \* 8).ToString();

}

if (alg\_flag == "HMACSHA512")

{

sig = DKA.HMAC\_Sig\_Create(b\_mess, alg\_flag + para\_file\_name);

for (int i = 0; i < sig.Length; i++)

textBox3.Text += sig[i].ToString("x") + " ";

textBox6.Text = (sig.Length \* 8).ToString();

}

File.WriteAllBytes(alg\_flag + out\_file\_name, sig);

}

private void button4\_Click(object sender, EventArgs e) // модификация подписи

{

Dig\_key\_alg DKA = new Dig\_key\_alg();

string b\_num = textBox5.Text;

if (b\_num == "" || Convert.ToInt32(b\_num) > (sig.Length \* 8))

MessageBox.Show("Не корректный номер бита.");

else

sig = DKA.bit\_mod(b\_num, sig);

}

private void button2\_Click(object sender, EventArgs e) // проверка подписи

{

Dig\_key\_alg DKA = new Dig\_key\_alg();

alg\_flag = comboBox1.SelectedItem.ToString();

if (alg\_flag == "DSA")

{

if (DKA.DSA\_Sig\_Ver(b\_mess, sig, alg\_flag + para\_file\_name))

textBox4.Text = "Верный";

else

textBox4.Text = "Не верный";

}

if (alg\_flag == "ECDSA")

{

if (DKA.ECDSA\_Sig\_Ver(b\_mess, sig, alg\_flag + para\_file\_name))

textBox4.Text = "Верный";

else

textBox4.Text = "Не верный";

}

if (alg\_flag == "HMACSHA512")

{

if (DKA.HMAC\_Sig\_Ver(b\_mess, sig, alg\_flag + para\_file\_name))

textBox4.Text = "Верный";

else

textBox4.Text = "Не верный";

}

}

private void Button5\_Click(object sender, EventArgs e)

{

Dig\_key\_alg DKA = new Dig\_key\_alg();

alg\_flag = comboBox1.SelectedItem.ToString();

DKA.Key\_gen(alg\_flag, alg\_flag + para\_file\_name); // создаём ключ

}

}

Hash\_alg.cs

class Hash\_alg

{

public byte[] SHA\_512(byte[] data) // генератор хэша на основе SHA512

{

byte[] res = new byte[1];

SHA512 SHA = new SHA512Managed();

res = SHA.ComputeHash(data);

return res;

}

public byte[] RIPEMD\_160(byte[] data) // генератор хэша на основе RIPEMD160

{

byte[] res = new byte[1];

RIPEMD160 RIPEMD = new RIPEMD160Managed();

res = RIPEMD.ComputeHash(data);

return res;

}

}

Hash.cs

public partial class Hash : Form

{

public Hash()

{

InitializeComponent();

comboBox1.Items.Add("SHA512");

comboBox1.Items.Add("RIPEMD160");

comboBox1.SelectedIndex = 0;

}

byte[] b\_mess = new byte[1]; // сообщение в байтах

string[] mess = new string[1]; // сообщение в строках

string alg\_flag = ""; // название алгоритма

byte[] hash\_arr = new byte[1]; // хэш

private void Button3\_Click(object sender, EventArgs e)

{

File\_f F = new File\_f();

textBox2.Clear();

string file\_name = textBox1.Text;

if (file\_name == "") // проверка на ввод имени файла

MessageBox.Show("Enter file name.");

else

{

bool fl = File.Exists(file\_name); // проверка на существование файла

if (fl)

{

b\_mess = File.ReadAllBytes(file\_name);

mess = F.Read(file\_name);

string ms = "";

for (int i = 0; i < mess.Length; i++)

ms += mess[i];

textBox2.Text = ms;

}

else

MessageBox.Show("File don't exist.");

}

}

private void Button1\_Click(object sender, EventArgs e) // создаём хэш

{

textBox3.Clear();

string file\_sha = "SHA512.bin";

string file\_rip = "RIPEMD160.bin";

Hash\_alg hash = new Hash\_alg();

alg\_flag = comboBox1.SelectedItem.ToString();

if (alg\_flag == "SHA512")

{

hash\_arr = hash.SHA\_512(b\_mess);

File.WriteAllBytes(file\_sha, hash\_arr);

}

if (alg\_flag == "RIPEMD160")

{

hash\_arr = hash.RIPEMD\_160(b\_mess);

File.WriteAllBytes(file\_rip, hash\_arr);

}

for (int i = 0; i < hash\_arr.Length; i++)

textBox3.Text += hash\_arr[i].ToString("x") + " ";

}

}

Form1.cs

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

// вызов окон

private void button2\_Click(object sender, EventArgs e) // асимметричный алгоритм

{

Asym examp = new Asym();

examp.Show();

}

private void Button1\_Click(object sender, EventArgs e) // симметричные алгоритмы

{

Sym examp = new Sym();

examp.Show();

}

private void Button3\_Click(object sender, EventArgs e) // цифровые ключи

{

Dig\_key examp = new Dig\_key();

examp.Show();

}

private void Button4\_Click(object sender, EventArgs e) // хэш-функции

{

Hash examp = new Hash();

examp.Show();

}

}

1. **Выводы**

В ходе выполненной лабораторной было разработано программное средство (демонстрационный стенд), для демонстрации возможностей криптографической библиотеки System.Security.Cryptography.