Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени
Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»
КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине «Защита информации»

Студент:	Боженко Д.В.
Группа:	РК6-83Б
Тип задания:	Лабораторные работы №1-6
Название:	Работа с библиотекой OpenSSL и изучение алгоритма шифрования RSA
Вариант:	-

Студент _____ (подпись, дата) Боженко Д.В. (Фамилия, И.О.)

Преподаватель _____ (подпись, дата) Беломойцев Д. Е (Фамилия, И.О.)

Оценка: _____

Москва, 2023

Содержание

Работа с библиотекой OpenSSL и изучение алгоритма шифрования RSA

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы – ознакомиться с библиотекой OpenSSL и изучить алгоритм RSA

1 Задание

1. Сгенерировать пару ключей RSA, выполнить шифрование и расшифрование произвольного набора данных.

2. Осуществить генерацию запроса на сертификат и выпуск самоподписанного сертификата на данный запрос.

3. Рассчитать хеш по алгоритму SHA для произвольного набора данных.

4. Выпустить ЭЦП для произвольного набора данных на базе ключей и сертификатов из работы 2.

5. Необходимо составить программную реализацию алгоритма генерации общего секретного ключа (алгоритм Диффи-Хеллмана).

6. Необходимо составить программную реализацию алгоритма шифрования RSA.

2 Решение

Сгенерировать пару ключей RSA, выполнить шифрование и расшифрование произвольного набора данных

Для генерации пары ключей RSA, а также шифрования и расшифрования текстового файла выполним следующие действия:

1. Генерируем приватный ключ дешифрования (листинг 1).

Листинг 1. Генерация приватного ключа для дешифрования

```
genrsa -out lab1/privatekey.pem -des3 4096
```

2. На основе созданного приватного ключа создадим публичный ключ (листинг 2).

Листинг 2. Генерация публичного ключа для шифрования

```
rsa -in lab1/privatekey.pem -out lab1/pubkey.pem -pubout
```

3. Создадим файл doc.txt со строкой текста "Very very secret document".

4. Зашифруем файл doc.txt с помощью публичного ключа (листинг 3).

Листинг 3. Генерация зашифрованного файла

```
rsautl -in lab1/doc.txt -out lab1/secret_doc.cr -inkey lab1/pubkey.pem -pubin
    -encrypt
```

5. Дешифруем файл secret_doc.cr с помощью приватного ключа (листинг 4).

Листинг 4. Генерация дешифрованного файла

```
rsautl -in lab1/secret_doc.cr -out lab1/desh_doc.txt -inkey
    lab1/privatekey.pem -decrypt
```

Содержимое файлов doc.txt и desh_doc.txt представлено на рисунке 1.

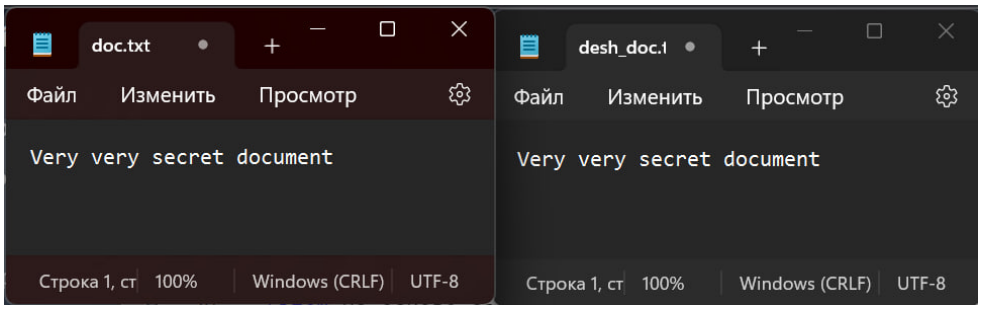

Рис. 1. Содержимое doc.txt и desh_doc.txt

Содержимое директории lab1 после выполнения вышеописанной последовательности действий представлено на рисунке 2.

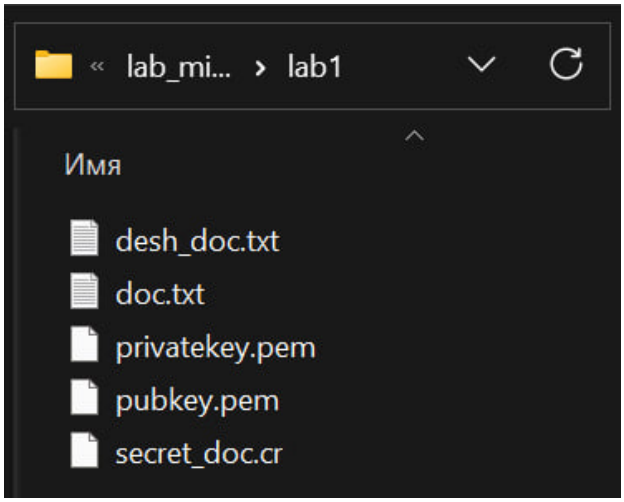

Рис. 2. Содержимое lab1

Осуществить генерацию запроса на сертификат и выпуск самоподписанного сертификата на данный запрос

Для генерации запроса на сертификат и выпуск самоподписанного сертификата на данный запрос выполним следующие действия:

1. Создаем конфиг файл config.txt, который заполняем необходимой нам информацией (произвольным образом).

2. Создаем запрос на сертификацию на основе создаваемого секретного ключа rsa (листинг 5).

Листинг 5. Создание запроса на сертификацию

```
req –new –newkey rsa:2048 –keyout lab2/rsa_key.pem –config lab2/config.txt
    –out lab2/certreq.pem
```

3. Выпускаем самоподписанный сертификат (-x509) (листинг 6).

Листинг 6. Генерация самоподписанного сертификата

```
req –x509 –new –key lab2/rsa_key.pem –config lab2/config.txt –out
    lab2/selfcert.pem –days 365
```

Содержимое директории lab2 после выполнения вышеописанной последовательности действий представлено на рисунке 3.

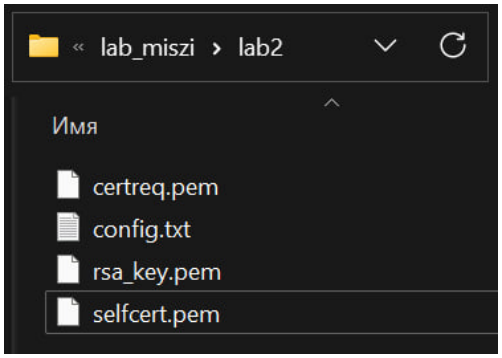

Рис. 3. Содержимое lab2

Рассчитать хеш по алгоритму SHA для произвольного набора данных

Для расчета хеша по алгоритму SHA для произвольного набора данных выполним следующие действия:

1. В рабочей директории создаем файл doc.txt.
2. Рассчитаем хеш по алгоритму SHA (листинг 7).

Листинг 7. Расчет хеша

```
sha1 lab3/doc.txt
```

Получаем:
SHA1(lab3/doc.txt)= 95c7405335c758b9f18df045566513611dacf8e0

Выпустить ЭЦП для произвольного набора данных на базе ключей и сертификатов из работы 2

Для выпуска ЭЦП для произвольного набора данных на базе ключей и сертификатов из работы 2 необходимо выполнить следующие действия:

1. Заносим в директорию lab4 файлы из директории lab2 (см. рис. 3) за исключением файла certreq.pem.

2. Создаем электронную цифровую подпись (ЭЦП) для текстового файла (листинг 8).

Листинг 8. Создание электронной цифровой подписи (ЭЦП)

```
dgst -sign lab4/rsa_key.pem -out lab4/signature -sha1 lab4/config.txt
```

3. Создаем публичный ключ на основе существующего приватного ключа rsa_-key.pem (листинг 9).

Листинг 9. Создание публичного ключа

```
rsa -in lab4/rsa_key.pem -out lab4/pubkey.pem -pubout
```

4. Осуществляем проверку ЭЦП (листинг 10).

Листинг 10. Проверка ЭЦП

```
dgst -signature lab4/signature -verify lab4/pubkey.pem lab4/config.txt
```

Выходные данные:
Verified OK

Составить программную реализацию алгоритма генерации общего секретного ключа (алгоритм Диффи-Хеллмана)

Алгоритм Диффи-Хеллмана можно рассмотреть на примере. Предположим, что Алиса хочет установить общий секрет код с Бобом. Вот пример протокола с секретными значениями.

1. Алиса и Боб соглашаются использовать простое число $n = 23$ и основание деления по модулю $p = 5$. (Эти два значения выбраны таким образом, чтобы гарантировать, что результирующий общий секрет может принимать любое значение от 1 до $n - 1$).

2. Алиса выбирает секретное число $a = 6$, затем отправляет Бобу $A = p^a \mod n = 8$.

3. Боб выбирает секретное целое число $b = 15$, затем отправляет Алисе $B = p^b \mod n = 19$.

4. Алиса вычисляет $c = B^a \mod n = 2$.

5. Боб вычисляет $c = A^b \mod n = 2$.

Алиса и Боб теперь имеют общий секрет $c = 2$. Число, которое Алиса получила на шаге 4, совпадает с числом, полученным Бобом на шаге 5.

Алгоритм Диффи-Хеллмана в основном используется для обмена криптографическими ключами для использования в алгоритмах симметричного шифрования, таких как AES. Информация не передается во время обмена ключами. Здесь две стороны вместе создают ключ.

Программная реализация этого алгоритма представлена в листинге 11.

Листинг 11. Программная реализация алгоритма Диффи-Хеллмана

```
#include <stdio.h>

int compute(int p, int m, int n)
{
  int r;
  int y = 1;

  while (m > 0)
  {
    r = m % 2;

    if (r == 1)
      y = (y*p) % n;
    p = p*p % n;

    m = m / 2;
  }

  return y;
}

int main()
{
  int n = 23;
  int p = 5;
  srand(time(NULL));

  int a, b;
  int A, B;

  a = rand();
  A = compute(p, a, n);
  b = rand();
  B = compute(p, b, n);

  int keyA = compute(B, a, n);
  int keyB = compute(A, b, n);
```

```
38
39    printf("A: %d\nB: %d", keyA, keyB);
40
41    return 0;
42 }
```

Составить программную реализацию алгоритма шифрования RSA

Программная реализация алгоритма RSA представлена в листинге 12.

Листинг 12. Программная реализация алгоритма RSA

```
1  #include <iostream>
2  #include <cmath>
3  #include <cstring>
4  #include <ctime>
5  #include <cstdlib>
6  using namespace std;
7
8
9  int Plaintext [100]; // Открытый текст
10  long long Ciphertext [100]; // Зашифрованный текст
11 int n, e = 0, d;
12
13  // Двоичное преобразование
14 int BianaryTransform(int num, int bin_num[])
15 {
16
17    int i = 0, mod = 0;
18
19     // Преобразуется в двоичный, обратный временно сохраняется в массиве temp []
20    while(num != 0)
21    {
22      mod = num%2;
23      bin_num[i] = mod;
24      num = num/2;
25      i++;
26    }
27
28     // Возвращает количество цифр в двоичных числах
29    return i;
30 }
31
32  // Повторное возведение в квадрат в степень
33 long long Modular_Exonentiation(long long a, int b, int n)
34 {
```

```
35   int c = 0, bin_num[1000];
36   long long d = 1;
37   int k = BianaryTransform(b, bin_num)-1;
38
39   for(int i = k; i >= 0; i--)
40   {
41     c = 2*c;
42     d = (d*d)%n;
43     if(bin_num[i] == 1)
44     {
45       c = c + 1;
46       d = (d*a)%n;
47     }
48   }
49   return d;
50 }
51
52  // Генерация простых чисел в пределах 1000
53 int ProducePrimeNumber(int prime[])
54 {
55   int c = 0, vis[1001];
56   memset(vis, 0, sizeof(vis));
57   for(int i = 2; i <= 1000; i++)if(!vis[i])
58   {
59     prime[c++] = i;
60     for(int j = i*i; j <= 1000; j+=i)
61       vis[j] = 1;
62   }
63
64   return c;
65 }
66
67
68  // Расширенный алгоритм Евклида
69 int Exgcd(int m,int n,int &x)
70 {
71   int x1,y1,x0,y0, y;
72   x0=1; y0=0;
73   x1=0; y1=1;
74   x=0; y=1;
75   int r=m%n;
76   int q=(m-r)/n;
77   while(r)
78   {
79     x=x0-q*x1; y=y0-q*y1;
80     x0=x1; y0=y1;
```

```
81      x1=x; y1=y;
82      m=n; n=r; r=m%n;
83      q=(m-r)/n;
84    }
85    return n;
86  }
87
88   // Инициализация RSA
89  void RSA_Initialize()
90  {
91     // Вынимаем простые числа в пределах 1000 и сохраняем их в массиве prime []
92    int prime[5000];
93    int count_Prime = ProducePrimeNumber(prime);
94
95     // Случайно возьмем два простых числа p, q
96    srand((unsigned)time(NULL));
97    int ranNum1 = rand()%count_Prime;
98    int ranNum2 = rand()%count_Prime;
99    int p = prime[ranNum1], q = prime[ranNum2];
100
101   n = p*q;
102
103   int On = (p-1)*(q-1);
104
105
106    // Используем расширенный алгоритм Евклида, чтобы найти e, d
107   for(int j = 3; j < On; j+=1331)
108   {
109     int gcd = Exgcd(j, On, d);
110     if( gcd == 1 && d > 0)
111     {
112       e = j;
113       break;
114     }
115
116   }
117
118 }
119
120  // шифрование RSA
121 void RSA_Encrypt()
122 {
123   cout<<"Public Key (e, n) : e = "<<e<<" n = "<<n<<'\n';
124   cout<<"Private Key (d, n) : d = "<<d<<" n = "<<n<<'\n'<<'\n';
125
126   int i = 0;
```

```
127   for(i = 0; i < 100; i++)
128     Ciphertext[i] = Modular_Exonentiation(Plaintext[i], e, n);
129
130   cout<<"Use the public key (e, n) to encrypt:"<<'\n';
131   for(i = 0; i < 100; i++)
132     cout<<Ciphertext[i]<<" ";
133   cout<<'\n'<<'\n';
134 }
135
136  // Расшифровка RSA
137 void RSA_Decrypt()
138 {
139   int i = 0;
140   for(i = 0; i < 100; i++)
141     Ciphertext[i] = Modular_Exonentiation(Ciphertext[i], d, n);
142
143   cout<<"Use private key (d, n) to decrypt:"<<'\n';
144   for(i = 0; i < 100; i++)
145     cout<<Ciphertext[i]<<" ";
146   cout<<'\n'<<'\n';
147 }
148
149
150  // Инициализация алгоритма
151 void Initialize()
152 {
153   int i;
154   srand((unsigned)time(NULL));
155   for(i = 0; i < 100; i++)
156     Plaintext[i] = rand()%1000;
157
158   cout<<"Generate 100 random numbers:"<<'\n';
159   for(i = 0; i < 100; i++)
160     cout<<Plaintext[i]<<" ";
161   cout<<'\n'<<'\n';
162 }
163
164 int main()
165 {
166   Initialize();
167
168   while(!e)
169     RSA_Initialize();
170
171   RSA_Encrypt();
172
```

```
173   RSA_Decrypt();
174
175   return 0;
176 }
```

3 Вывод

В ходе выполнения лабораторной работы были получены навыки работы с библиотекой OpenSSL, а также реализован алгоритм RSA на языке C.

Постановка: © *кандитат технических наук, доцент, Беломойцев Д. Е.*
Решение и вёрстка: © *студент группы РК6-83Б, Боженко Д.В.*

2023, весенний семестр