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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №4

по курсу «Защита информации»

на тему: «Электронная цифровая подпись»

Студент ИУ7-73Б _____ Лагутин Д. В.

(Группа) (Подпись, дата) (Фамилия И. О.)

Преподаватель _____ Чиж И. С.

(Подпись, дата) (Фамилия И. О.)

Москва, 2023 г.

Содержание

Введение

Целью работы является реализация программы создания и проверки электронной подписи для документа с использованием алгоритма RSA и алгоритмов хеширования MD5 (вариант 1). Предусмотреть работу программы с пустым, однобайтовым файлом. Программа также должна уметь обрабатывать файл архива (rar, zip или др.).

Задачи:

1) изучить алгоритм шифрования;
2) изучить алгоритм хеширования;
3) разработать алгоритм работы программы;
4) реализовать и протестировать разработанное программное обеспечение.

1 Аналитическая часть

1.1 Электронная цифровая подпись

Электронная цифровая подпись — набор криптографических методов, позволяющий подтвердить авторство электронного документа (будь то реальное лицо или, например, аккаунт в криптовалютной системе).

Как правило, подпись получается при помощи шифрования документа с использованием асимметричного шифрования закрытым ключом автора.

Для сокращения объема шифруемой и отправляемой информации подписываемая информация первоначально хешируется.

1.2 Алгоритм шифрования RSA

RSA — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших чисел.

Задача заключается в нахождении таких чисел e, d и n таких, что для любого $0 \leq m < n$

$$m^{ed} = m \mod n.$$

Открытым ключом в данном алгоритме является пара (e,n), закрытым — пара (d,n).

Таким образом шифр текст получается в результате следующей операции

$$c = m^e \mod n,$$

расшифрованный текст

$$r = c^d \mod n.$$

1.2.1 Алгоритм получения ключа

RSA-ключи генерируются следующим образом:

1) выбираются два различных случайных простых числа p и q заданного размера;
2) вычисляется их произведение $n = p \cdot q$, которое называется модулем;
3) вычисляется значение функции Эйлера от числа n: $\varphi(n) = (p-1) \cdot (q-1)$;
4) выбирается целое число $1 < e < \varphi(n)$, взаимно простое со значением функции $\varphi(n)$; обычно в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые из чисел Ферма: 17, 257 или 65537;
5) вычисляется число d, мультипликативно обратное к числу e по модулю $\varphi(n)$, то есть число, удовлетворяющее сравнению: $d \cdot e = 1 \mod \varphi(n)$ (обычно оно вычисляется при помощи расширенного алгоритма Евклида);

1.3 Алгоритм хеширования MD5

MD5 — 128-битный алгоритм хеширования. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности.

Последовательность действий для алгоритма:

— длина сообщения выравнивается до длины в 512 бит (даже если оно уже удовлетворяет этому условию) по следующим правилам
  - сообщение дополняется единичным битом;
  - сообщение дополняется нулевыми битами до длины, сравнимой 448 по модулю 512;
  - оставшиеся 64 бита заполняются соответствующим представлением длины сообщения в битах до расширения;

— инициализируются 4 32 битных буфера `A`, `B`, `C`, `D` следующими значениями (и 4 соответствующих раундовых `a`, `b`, `c`, `d`)

$$A = 67452301h$$
$$B = EFCDAB89h$$
$$C = 98BADCFEh$$
$$D = 10325476h$$

— основной этап хеширования 4 этапа по 16 раундов

  – текущие значения буферов заносятся в соответствующие раундовые;
  – значение буфера `a` вычисляется следующим образом

$$a = b + ((a + \mathrm{F_i}(b,c,d) + T[i] + X[i]) <<< s[i]),$$

    где $\mathrm{F_i}$ — раундовая функция, $T[i] = \lfloor 2^3 2 \cdot |\sin(i)| \rfloor$, $X[i]$ — текущее 32 битное слово сообщения, $s[i]$ — величина левого циклического сдвига;
  – производится циклический сдвиг вправо;
  – значение раундовых буферов складывается с результирующими;

— полученные буферы записываются в порядке `ABCD`, порядок байт — little-endian.

Раундовые функции

$$\mathrm{F_0} = \mathrm{F} = (b \wedge c) \vee (\neg b \wedge d)$$
$$\mathrm{F_1} = \mathrm{G} = (b \wedge d) \vee (\neg d \wedge c)$$
$$\mathrm{F_2} = \mathrm{H} = b \oplus c \oplus d$$
$$\mathrm{F_3} = \mathrm{I} = c \oplus (\neg d \vee b).$$

2 Конструкторская часть

2.1 Схема алгоритма

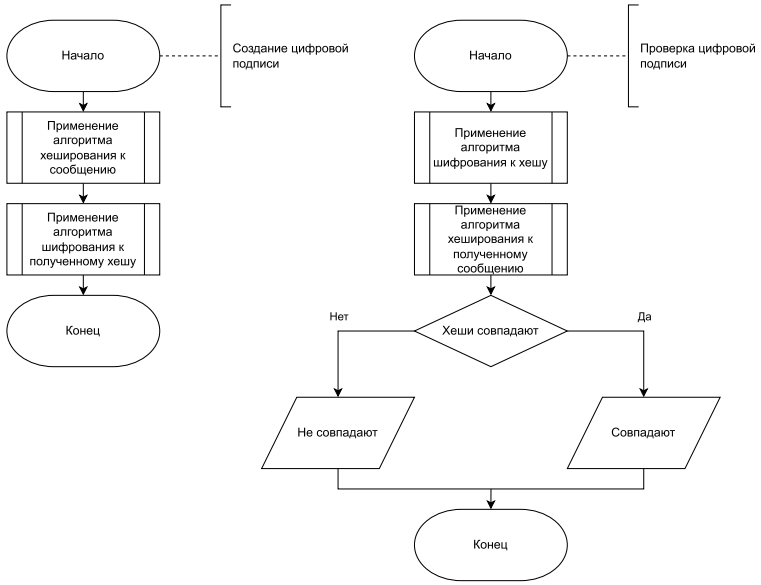

Рисунок 2.1 – Алгоритм получения и проверки цифровой подписи

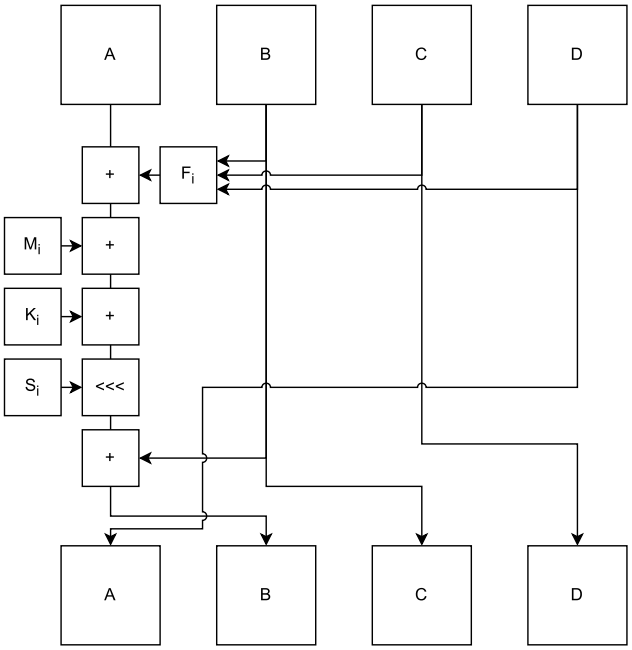

Рисунок 2.2 – Алгоритм работы MD5

3 Технологическая часть

3.1 Описание программного обеспечения

Для реализации машины использовался язык C++. Конфигурация осуществляется при помощи файлов private.json и public.json, расположенных в корневом каталоге, содержащих соответствующие значения закрытого и открытого ключей. Длина ключа выбрана равной 64 бита.

```
./public.json
{
    "e": 17,
    "n": 479653117661354309
}
./private.json
{
    "d": 282148891917378473,
    "n": 479653117661354309
}
```

Листинг 3.1 – Класс, реализующий алгоритм RSA

```
1  class RSA : public Encoder
2  {
3      public:
4          RSA(unsigned long long n);
5          void setPublicKey(unsigned long long e);
6          void setPrivateKey(unsigned long long d);
7          virtual ~RSA(void) override = default;
8          virtual std::string encode(const std::string &origin) override;
9          virtual std::string decode(const std::string &origin) override;
10     private:
11         unsigned long long e;
12         unsigned long long d;
13         const unsigned long long n;
14 };
15
16 static unsigned long long modadd(unsigned long long a,
17                                  unsigned long long b,
```

```
18                                  unsigned long long mod);
19 static unsigned long long modmul(unsigned long long a,
20                                  unsigned long long b,
21                                  unsigned long long mod);
22 static unsigned long long modpow(unsigned long long a,
23                                  unsigned long long pow,
24                                  unsigned long long mod);
25
26
27 std::string RSA::encode(const std::string &origin)
28 {
29     if (0 == this->e)
30         throw;
31     size_t size = origin.length(), rest = size % 7;
32     size = size / 7 + (rest ? 1 : 0);
33     std::string out;
34     out.reserve(size * 8);
35     unsigned long long current, mask = 0x00ffffffffffffff;
36     for (size_t i = 0; size > i; i++)
37     {
38         if (size == i + 1 && rest)
39         {
40             mask = 0;
41             for (size_t j = 0; rest > j; j++)
42                 mask = (mask << 8) | 0xff;
43         }
44         current = *(unsigned long long *)(origin.data() + 7 * i)
45                   & mask;
46         current = modpow(current, this->e, this->n);
47         out += std::string((char *)&current, 8);
48     }
49     return out;
50 }
51
52 std::string RSA::decode(const std::string &origin)
53 {
54     if (0 == this->d)
55         throw;
56     size_t size = origin.length(), psize = size / 8;
57     if (0 != size % 8)
58         throw;
59     std::string out;
60     out.reserve(size - psize);
61     out.resize(size - psize);
62     const unsigned long long *inp =
63     (unsigned long long *)origin.data();
64     unsigned long long current;
65     char *outp = out.data(), *currentp = (char *)&current;
```

```
66      for (size_t i = 0; psize > i; i++)
67      {
68          current = inp[i];
69          current = modpow(current, this->d, this->n);
70          for (size_t j = 0; 7 > j; j++)
71              *(outp++) = currentp[j];
72      }
73      return out;
74  }
75
76  static unsigned long long modadd(unsigned long long a,
77                                   unsigned long long b,
78                                   unsigned long long mod)
79  {
80      a %= mod;
81      b %= mod;
82      unsigned long long diff = mod - b;
83      if (a >= diff)
84          return a - diff;
85      return a + b;
86  }
87
88  static unsigned long long modmul2(unsigned long long a,
89                                    unsigned long long mod)
90  {
91      unsigned long long shrinked = mod >> 1, rest = mod & 1;
92      if (a >= shrinked)
93          return ((a - shrinked) << 1) - rest;
94      return a << 1;
95  }
96
97  static unsigned long long modmul(unsigned long long a,
98                                   unsigned long long b,
99                                   unsigned long long mod)
100 {
101     a %= mod;
102     b %= mod;
103     unsigned long long result = 0;
104     for (; b; b >>= 1, a = modmul2(a, mod))
105         if (b & 1)
106             result = modadd(result, a, mod);
107     return result;
108 }
109
110 static unsigned long long modpow(unsigned long long a,
111                                  unsigned long long pow,
112                                  unsigned long long mod)
113 {
```

```
    a %= mod;
    unsigned long long result = 1;
    for (; pow; pow >>= 1)
    {
        if (pow & 1)
            result = modmul(result, a, mod);
        a = modmul(a, a, mod);
    }
    return result;
}
```

Листинг 3.2 – Класс, реализующий алгоритм MD5

```
class MD5 : public HashFunction
{
    public:
        virtual ~MD5(void) override = default;
        virtual std::string apply(const std::string &origin) override;
    private:
        using FFunc = unsigned int (*)(const unsigned int,
                                       const unsigned int,
                                       const unsigned int);
        using Iter = struct
        {
            FFunc func;
            size_t k;
        };
        static void getIteration(const size_t i, Iter &handler);
        static unsigned int F(const unsigned int b,
                              const unsigned int c,
                              const unsigned int d);
        static unsigned int G(const unsigned int b,
                              const unsigned int c,
                              const unsigned int d);
        static unsigned int H(const unsigned int b,
                              const unsigned int c,
                              const unsigned int d);
        static unsigned int I(const unsigned int b,
                              const unsigned int c,
                              const unsigned int d);
    private:
        static const unsigned char expansion[64];
        static const size_t S[64];
        static const unsigned int K[64];
};

const unsigned char MD5::expansion[64] = { 0x80 };

const size_t MD5::S[64] =
```

```
{
    ...
};

const unsigned int MD5::K[64] =
{
    ...
};

template <typename Type>
Type shift_left(const Type val, size_t size)
{
    static constexpr const size_t tsize = sizeof(Type) << 3;
    size %= tsize;
    return (val << size) | (val >> (tsize - size));
}

std::string MD5::apply(const std::string &origin)
{
    size_t size = origin.length(), esize;
    const size_t bits = size * 8;
    const size_t rest = size % 64;
    if (56 > rest)
        esize = 56 - rest;
    else
        esize = 120 - rest;
    size += esize + 8;
    std::string copy;
    copy.reserve(size);
    copy += origin + std::string((const char *)MD5::expansion, esize)
            + std::string((char *)&bits, 8);
    const unsigned char *data = (unsigned char *)copy.data();
    const unsigned int *current = nullptr;
    unsigned int A = 0x67452301, a,
                 B = 0xEFCDAB89, b,
                 C = 0x98BADCFE, c,
                 D = 0x10325476, d,
                 F = 0;
    Iter handler;
    for (size_t i = 0, blocks = size >> 6; blocks > i; i++)
    {
        a = A, b = B, c = C, d = D;
        current = (const unsigned int *)(data + i * 64);
        for (size_t j = 0; 64 > j; j++)
        {
            MD5::getIteration(j, handler);

            F = handler.func(b, c, d) + a + MD5::K[j]
```

```
85                  + current[handler.k];
86              a = d;
87              d = c;
88              c = b;
89              b += shift_left<unsigned int>(F, MD5::S[j]);
90          }
91          A += a, B += b, C += c, D += d;
92      }
93      return std::string((char *)&A, 4) + std::string((char *)&B, 4)
94             + std::string((char *)&C, 4) + std::string((char *)&D, 4);
95  }
96
97  void MD5::getIteration(const size_t i, Iter &handler)
98  {
99      if (16 > i)
100     {
101         handler.func = &MD5::F;
102         handler.k = i;
103     }
104     else if (32 > i)
105     {
106         handler.func = &MD5::G;
107         handler.k = (5 * i + 1) % 16;
108     }
109     else if (48 > i)
110     {
111         handler.func = &MD5::H;
112         handler.k = (3 * i + 5) % 16;
113     }
114     else
115     {
116         handler.func = &MD5::I;
117         handler.k = (7 * i) % 16;
118     }
119 }
120
121 unsigned int MD5::F(const unsigned int b, const unsigned int c,
122                     const unsigned int d)
123 {
124     return (b & c) | (~b & d);
125 }
126
127 unsigned int MD5::G(const unsigned int b, const unsigned int c,
128                     const unsigned int d)
129 {
130     return (b & d) | (~d & c);
131 }
132
```

```
133 unsigned int MD5::H(const unsigned int b, const unsigned int c,
134                     const unsigned int d)
135 {
136     return b ^ c ^ d;
137 }
138
139 unsigned int MD5::I(const unsigned int b, const unsigned int c,
140                     const unsigned int d)
141 {
142     return c ^ (~d | b);
143 }
```

3.2 Тестирование

Таблица 3.1 – Тестирование алгоритма MD5

№	Исходные данные	Ожидаемый результат	Фактический результат
1	<Пустая строка>	d41d8cd98f00b204e9800998ecf8427e	d41d8cd98f00b204e9800998ecf8427e
2	abcdefghijklm nopqstuvwxyz\0a	e302f9ecd2d189fa80aac1c3392e9b9c	e302f9ecd2d189fa80aac1c3392e9b9c
3	md5	1bc29b36f623ba82aaf6724fd3b16718	1bc29b36f623ba82aaf6724fd3b16718

Для тестирования алгоритма RSA использовались ключи, приведенные выше.

Таблица 3.2 – Тестирование алгоритма RSA (шифрование)

№	Исходные данные	Ожидаемый результат	Фактический результат
1	1	1	1
2	123	159162881495752659	159162881495752659
3	324440504654685957	64871042383186893	64871042383186893

Таблица 3.3 – Тестирование алгоритма RSA (расшифровка)

№	Исходные данные	Ожидаемый результат	Фактический результат
1	1	1	1
2	159162881495752659	123	123
3	64871042383186893	324440504654685957	324440504654685957

Заключение

Была разработана программа реализации электронной подписи с использованием алгоритма шифрования RSA и алгоритма хеширования MD5.

Были решены следующие задачи:

1) изучен алгоритм шифрования;
2) изучен алгоритм хеширования;
3) разработан алгоритм работы программы;
4) реализовано и протестировано программное обеспечение.