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. Алгоритм RSA с хешированием SHA-1»

Вариант № 9

Студент ИУ7-74Б (Группа) _____ (Подпись, дата) Разин А. В. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Чиж И. С. (И. О. Фамилия)

*2024 г.*

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Целью данной лабораторной работы является реализация в виде программы, позволяющая с использованием алгоритмов MD5 и RSA создать и проверить электронную подпись для документа.

Для достижения поставленной цели необходимо решить следующие задачи:

1) описать алгоритмы RSA и SHA1;

2) определить средства реализации программы;

3) реализовать программу, использующую алгоритмы RSA и SHA1;

4) протестировать реализованную программу.

# 1 Аналитический раздел

## 1.1 Электронная цифровая подпись

Электронная (цифровая) подпись позволяет подвердить авторство электронного документа. Она связана не только с автором документа, но и с самим документов (при помощи криптографических методов) и не может быть подделана при поммощи обычного копирования.

Создание ЭП с использованием криптографического алгоритма RSA и алгоритма хеширования SH1/MD5 происходит следующим образом:

1) происходит хеширование сообщения при помощи SH1/MD5, сообщение — файл, который неообходимо подписать;

2) происходит шифрование с использованием закрытого ключа RSA последовательности 128/160 бит, полученных на предыдущем этапе;

3) значение подписи — результат шифрования.

Проверка ЭП с использованием криптографического алгоритма RSA и алгоритма хеширования SH1 происходит следующим образом:

1) происходит хеширование сообщения при помощи SH1/MD5, сообщение — файл, подпись которого необходимо проверить;

2) происходит расшифровка подписи с использованием открытого ключа RSA;

3) происходит побитовая сверка значений, полученных на предыдущих этапах, если они одинаковы, подпись считается подлинной.

## 1.2 Алгоритм RSA

RSA — криптографический алгоритм с открытым ключом, разработанный учеными Ривестом, Шамиром и Адлеманом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптографическая система с открытым ключом — система шифрования, при которой открытый ключ передаётся по незащищенному каналу

и используется для проверки электронных подписей и для шифрования сообщения. Для генерации электронной подписи и расшифровки сообщений используется закрытый ключ.

Алгоритм RSA состоит из следующих этапов [1]:

1) Создание открытого и закрытого ключа:

   — выбирается два простых числа $p$ и $q$;

   — вычисляется $n = p \cdot q$;

   — вычисляется функция Эйлера $\varphi(n) = (p-1) \cdot (q-1)$;

   — выбирается открытая экспонента $e \in (1; \varphi(n))$;

   — вычисляется закрытая экспонента $d$, где $(d \cdot e) \bmod \varphi(n) = 1$;

   — создается открытый ключ $(e, n)$ и закрытый ключ $(d, n)$;

2) Шифрование и дешифрование:

   — шифрование сообщения: $w = G(h) = h^e \bmod n$;

   — дешифрование сообщения: $h = Q(w) = w^d \bmod n$;

Надежность шифрования обеспечивается тем, что третьему лицу (старающемуся взломать шифр) очень трудно вычислить закрытый ключ по от крытому. Оба ключа вычисляются из одной пары простых чисел (v и u). Если число является произведением двух очень больших простых чисел, что его трудно разложить на множители [1].

## 1.3 Алгоритм SHA-1

SHA-1 (англ. Secure Hash Algorithm 1) — это алгоритм хеширования, предназначенный для получения последовательности длиной 160 бит, используемой для проверки целостности и подлинности сообщений произвольной длины.

На вход алгоритма поступает последовательность бит произвольной длины $L$, хеш которой нужно найти.

Алгоритм SHA-1 состоит из 4 следующих этапов:

1) выравнивание потока;

2) добавление длины сообщения;

3) инициализация буфера;

4) вычисления в цикле.

Выравнивание потока заключается в добавлении некоторого числа нулевых бит, чтобы новая длина последовательности $L'$ стала сравнима с 448 по модулю 512.

После этого добавляется 64-битное представление длины исходного сообщения (в битах) в конец выровненной последовательности.

Сначала записывают младшие 8 байтов, затем старшие.

Далее происходит инициализация буфера, состоящего из 5 переменных $A, B, C, D, E$ размерностью 32 бита, начальные значения которых задаются шестнадцатеричными числами (порядок от младших к старшим).

В этих переменных будут храниться результаты промежуточных вычислений.

Далее в цикле каждый блок длиной 512 бит проходит 80 раундов вычислений. Для этого блок представляется в виде массива $W$ из 80 слов по 32 бита. Первые 16 слов копируются из блока, а оставшиеся 64 слова формируются по специфическому правилу, основанному на предыдущих значениях.

Все раунды имеют однотипную структуру и могут быть описаны следующим образом:

$$A = B + (\text{LeftRotate}(A, 5) + \text{F}(B, C, D) + W[t] + K)$$

где $t$ — текущий номер раунда (от 0 до 79),
LeftRotate — операция циклического сдвига влево,
$F$ — нелинейная функция, определяющая раунд,
$W[t]$ — слово из массива,
$K$ — константа, специфичная для каждого раунда.
Результат вычислений хранится в переменных $A, B, C, D, E$.

# 2 Конструкторский раздел

## 2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

1) хеширование и генерация ЭП для произвольного файла;
2) возможность работы с пустым, однобайтовым файлом;
3) возможность обрабатывать файл архива.

## 2.2 Реализация алгоритмов

На рисунке 2.1 представлена схема алгоритма RSA.

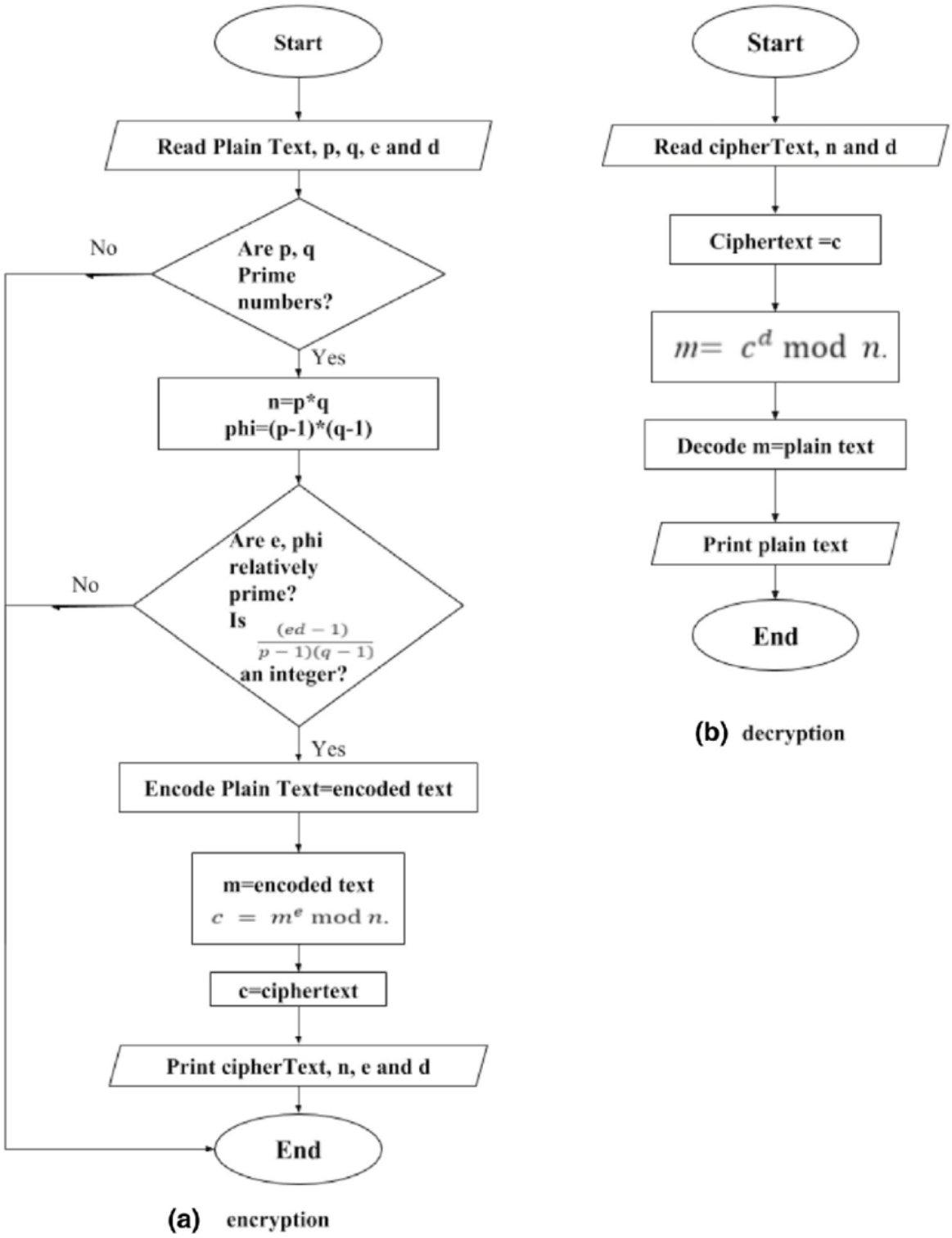


Рисунок 2.1 – Схема алгоритма RSA

На рисунке 2.2 представлена схема алгоритма SH-1.

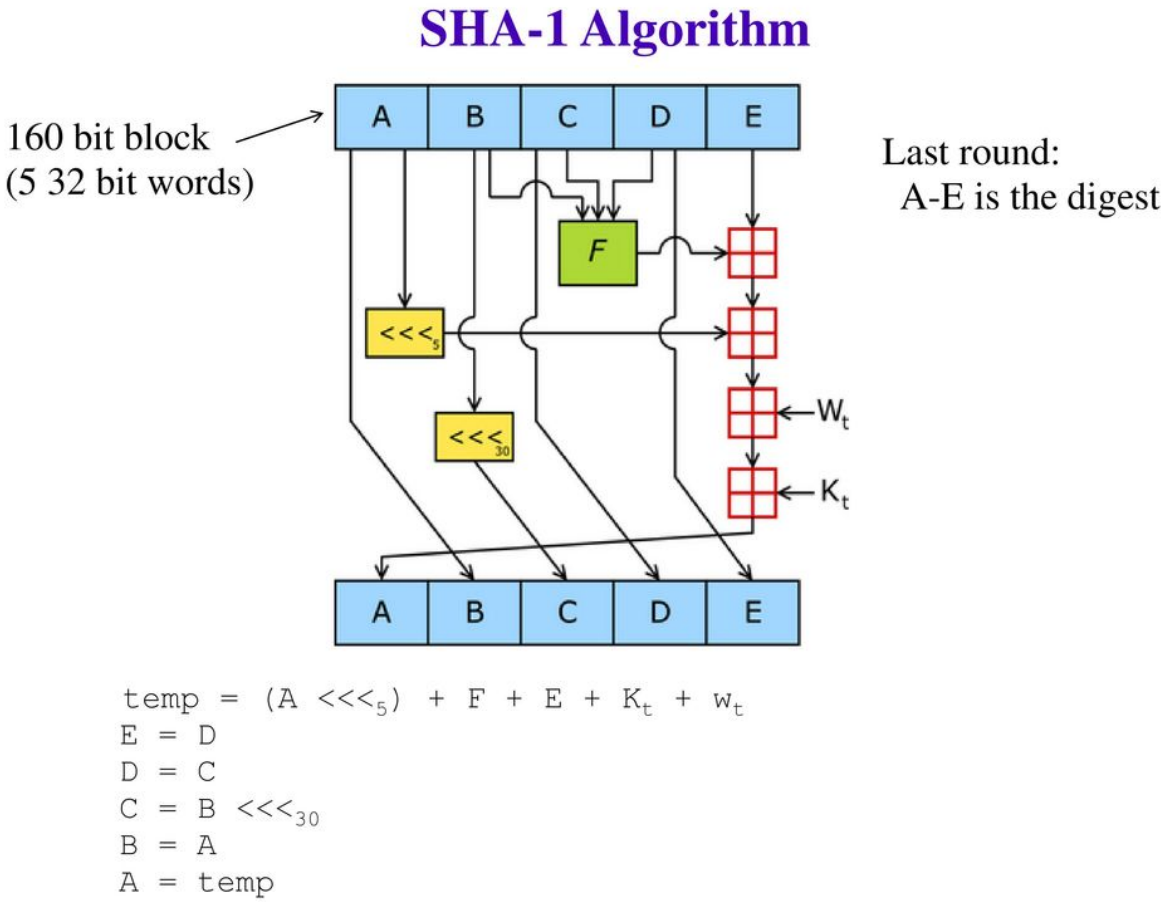


Рисунок 2.2 – Схема алгоритма SHA-1

# 3 Технологический раздел

## 3.1 Средства реализации

В качестве языка программирования, используемого при написании данной лабораторной работы, был выбран C++ [2].

## 3.2 Реализация алгоритмов

В листингах 3.1 – 3.3 представлены реализации разрабатываемых модулей.

Листинг 3.1 – Реализация класса алгоритма RSA (получение ключей)

```
#pragma once

#include <boost/multiprecision/cpp_int.hpp>

namespace lib
{

namespace bm = boost::multiprecision;

class RSA
{
public:
    RSA(bm::cpp_int p, bm::cpp_int q);

    bm::cpp_int Encrypt(bm::cpp_int message) const;
    bm::cpp_int Decrypt(bm::cpp_int ciphertext) const;

    std::pair<bm::cpp_int, bm::cpp_int> GetPublicKey() const {
       return { N, E }; }
    std::pair<bm::cpp_int, bm::cpp_int> GetPrivateKey() const {
       return { N, D }; }

private:
    bm::cpp_int N, E, D, Phi;
};

} // namespace lib
```

Листинг 3.2 – Реализация класса алгоритма RSA

```cpp
#include "RSA.h"

namespace lib
{

namespace
{

bm::cpp_int ModInverse(bm::cpp_int a, bm::cpp_int m)
{
    bm::cpp_int m0 = m, t, q;
    bm::cpp_int x0 = 0, x1 = 1;

    while (a > 1)
    {
        q = a / m;
        t = m;
        m = a % m, a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }

    if (x1 < 0)
    {
        x1 += m0;
    }

    return x1;
}

bm::cpp_int GCD(bm::cpp_int a, bm::cpp_int b)
{
    while (b != 0)
    {
        bm::cpp_int temp = a % b;
        a = b;
        b = temp;
    }
    return a;
}
```

```
42
43 } // namespace
44
45 RSA::RSA(bm::cpp_int p, bm::cpp_int q)
46 try : N{ p *q }, E{ 65537 }, Phi((p - 1) * (q - 1))
47 {
48     if (GCD(E, Phi) != 1)
49     {
50         throw std::runtime_error("e is not coprime with (n)");
51     }
52     D = ModInverse(E, Phi);
53 }
54 catch (const std::exception &e)
55 {
56     std::cerr << e.what() << std::endl;
57 }
58
59 bm::cpp_int RSA::Encrypt(bm::cpp_int plaintext) const { return
    bm::powm(plaintext, E, N); }
60
61 bm::cpp_int RSA::Decrypt(bm::cpp_int ciphertext) const { return
    bm::powm(ciphertext, D, N); }
62
63 } // namespace lib
```

Листинг 3.3 – Реализация алгоритма MD5

```
1  #include <array>
2  #include <cstdint>
3  #include <cstring>
4  #include <fstream>
5  #include <iomanip>
6  #include <iostream>
7  #include <sstream>
8  #include <vector>
9
10 namespace lib
11 {
12
13 namespace
14 {
15
16 std::uint32_t Leftrotate(std::uint32_t x, std::uint32_t c) {
```

```
    return (x << c) | (x >> (32 - c)); }
17
18  std::vector<std::uint32_t> AddPadding(const std::uint32_t *data,
        std::size_t length)
19  {
20      std::size_t original_byte_len = length;
21      std::size_t bit_len = original_byte_len * 8;
22      std::size_t padding_len = 64 - ((original_byte_len + 8) %
            64);
23      std::size_t total_len = original_byte_len + padding_len + 8;
24
25      std::vector<std::uint8_t> padded_data(total_len, 0);
26      std::memcpy(padded_data.data(), data, original_byte_len);
27
28      padded_data[original_byte_len] = 0x80;
29      std::memcpy(padded_data.data() + total_len - 8, &bit_len, 8);
30
31      return std::vector<std::uint32_t>(
32          reinterpret_cast<std::uint32_t *>(padded_data.data()),
33          reinterpret_cast<std::uint32_t *>(padded_data.data() +
                total_len));
34  }
35
36  std::array<std::uint32_t, 4> ProcessChunk(std::uint32_t
        *current_hash, std::uint32_t *M)
37  {
38      const std::uint32_t s[] = { 7, 12, 17, 22, 7, 12, 17, 22, 7,
            12, 17, 22, 7, 12, 17, 22,
39                                  5, 9,  14, 20, 5, 9,  14, 20, 5, 9,
                                        14, 20, 5, 9,  14, 20,
40                                  4, 11, 16, 23, 4, 11, 16, 23, 4, 11,
                                        16, 23, 4, 11, 16, 23,
41                                  6, 10, 15, 21, 6, 10, 15, 21, 6, 10,
                                        15, 21, 6, 10, 15, 21 };
42
43      const std::uint32_t K[] = { 0xd76aa478, 0xe8c7b756,
            0x242070db, 0xc1bdceee, 0xf57c0faf, 0x4787c62a,
44                                  0xa8304613, 0xfd469501, 0x698098d8,
                                        0x8b44f7af, 0xffff5bb1, 0x895cd7be,
45                                  0x6b901122, 0xfd987193, 0xa679438e,
                                        0x49b40821, 0xf61e2562, 0xc040b340,
```

```
46                                  0x265e5a51, 0xe9b6c7aa, 0xd62f105d,
                                       0x02441453, 0xd8a1e681, 0xe7d3fbc8,
47                                  0x21e1cde6, 0xc33707d6, 0xf4d50d87,
                                       0x455a14ed, 0xa9e3e905, 0xfcefa3f8,
48                                  0x676f02d9, 0x8d2a4c8a, 0xfffa3942,
                                       0x8771f681, 0x6d9d6122, 0xfde5380c,
49                                  0xa4beea44, 0x4bdecfa9, 0xf6bb4b60,
                                       0xbebfbc70, 0x289b7ec6, 0xeaa127fa,
50                                  0xd4ef3085, 0x04881d05, 0xd9d4d039,
                                       0xe6db99e5, 0x1fa27cf8, 0xc4ac5665,
51                                  0xf4292244, 0x432aff97, 0xab9423a7,
                                       0xfc93a039, 0x655b59c3, 0x8f0ccc92,
52                                  0xffeff47d, 0x85845dd1, 0x6fa87e4f,
                                       0xfe2ce6e0, 0xa3014314, 0x4e0811a1,
53                                  0xf7537e82, 0xbd3af235, 0x2ad7d2bb,
                                       0xeb86d391 };
54
55      std::uint32_t A = current_hash[0];
56      std::uint32_t B = current_hash[1];
57      std::uint32_t C = current_hash[2];
58      std::uint32_t D = current_hash[3];
59
60      for (size_t j = 0; j < 64; ++j)
61      {
62          std::uint32_t F, g;
63
64          if (j < 16)
65          {
66              F = (B & C) | ((~B) & D);
67              g = j;
68          }
69          else if (j < 32)
70          {
71              F = (D & B) | ((~D) & C);
72              g = (5 * j + 1) % 16;
73          }
74          else if (j < 48)
75          {
76              F = B ^ C ^ D;
77              g = (3 * j + 5) % 16;
78          }
```

```
79          else
80          {
81              F = C ^ (B | (~D));
82              g = (7 * j) % 16;
83          }
84
85          std::uint32_t temp = D;
86          D = C;
87          C = B;
88          B = B + Leftrotate(A + F + K[j] + M[g], s[j]);
89          A = temp;
90      }
91
92      current_hash[0] += A;
93      current_hash[1] += B;
94      current_hash[2] += C;
95      current_hash[3] += D;
96
97      return { current_hash[0], current_hash[1], current_hash[2],
            current_hash[3] };
98  }
99
100 } // namespace
101
102 namespace md5
103 {
104
105 std::string ToString(const std::array<std::uint32_t, 4>
        &hash_parts)
106 {
107     std::ostringstream result;
108     for (std::uint32_t part : hash_parts)
109     {
110         result << std::hex << std::setfill('0') << std::setw(8)
                << part;
111     }
112     return result.str();
113 }
114
115 std::string HashFile(const std::string &filename)
116 {
```

```
117     std::ifstream file(filename, std::ios::binary);
118     if (!file.is_open())
119     {
120         throw std::runtime_error("Couldn't open file.");
121     }
122     std::uint32_t current_hash[4] = { 0x67452301, 0xefcdab89,
          0x98badcfe, 0x10325476 };
123     std::streamsize buffer_size = 512;
124     std::vector<std::uint8_t> buffer(buffer_size);
125     std::uint64_t total_bits = 0;
126
127     while (file)
128     {
129         file.read(reinterpret_cast<char *>(buffer.data()),
              buffer_size);
130         std::streamsize bytes_read = file.gcount();
131         total_bits += bytes_read * 8;
132
133         std::size_t chunk_size = bytes_read / 64;
134
135         for (std::size_t i = 0; i < chunk_size; ++i)
136         {
137             ProcessChunk(current_hash, reinterpret_cast<uint32_t
                  *>(buffer.data() + i * 64));
138         }
139
140         if (bytes_read % 64 != 0)
141         {
142             std::size_t remain = bytes_read % 64;
143             buffer[remain] = 0x80;
144             if (remain < 56)
145             {
146                 std::memset(buffer.data() + remain + 1, 0, 56 -
                      remain - 1);
147                 buffer[56] =
                      static_cast<std::uint8_t>(total_bits);
148                 buffer[57] =
                      static_cast<std::uint8_t>(total_bits >> 8);
149                 buffer[58] =
                      static_cast<std::uint8_t>(total_bits >> 16);
150                 buffer[59] =
```

```
                    static_cast<std::uint8_t>(total_bits >> 24);
151                 ProcessChunk(current_hash,
                        reinterpret_cast<uint32_t *>(buffer.data()));
152             }
153             else
154             {
155                 std::memset(buffer.data() + remain + 1, 0, 64 -
                        remain - 1);
156                 ProcessChunk(current_hash,
                        reinterpret_cast<uint32_t *>(buffer.data()));
157                 std::memset(buffer.data(), 0, 56);
158                 buffer[56] =
                        static_cast<std::uint8_t>(total_bits);
159                 buffer[57] =
                        static_cast<std::uint8_t>(total_bits >> 8);
160                 buffer[58] =
                        static_cast<std::uint8_t>(total_bits >> 16);
161                 buffer[59] =
                        static_cast<std::uint8_t>(total_bits >> 24);
162                 ProcessChunk(current_hash,
                        reinterpret_cast<uint32_t *>(buffer.data()));
163             }
164         }
165     }
166
167     return ToString({ current_hash[0], current_hash[1],
            current_hash[2], current_hash[3] });
168 }
169
170 } // namespace md5
171
172 } // namespace lib
```

## 3.3 Тестирование

В таблице 3.1 представлены функциональные тесты.

Таблица 3.1 – Функциональные тесты

| № | Входные данные | Выходные данные |
|---|---|---|
| 1 | 2 пустых файла | Signature is valid |
| 2 | 2 одинаковых текстовых файла | Signature is valid |
| 3 | 2 разных текстовых файла | Signature is invalid |
| 4 | 2 одинаковых архива | Signature is valid |
| 5 | 2 разных архива | Signature is invalid |

# ЗАКЛЮЧЕНИЕ

Цель, поставленная в начале работы, была достигнута, выполнены все поставленные задачи:

1) описаны алгоритмы RSA и SHA1;

2) определены средства реализации программы;

3) реализована программа, использующая алгоритмы RSA и MD5;

4) реализованная программа протестирована.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Алексеевна К. Д.* СХЕМА ШИФРОВАНИЯ RSA И ЕЁ ПРОГРАММНАЯ РЕАЛИЗАЦИЯ // Евразийский научный журнал. — 2017. — № 1.
2. C++ language. — [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/language` (дата обращения: 22.11.2024).