



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 работа №4

по курсу «Защита информации»

на тему: «Электронная цифровая подпись. Алгоритм RSA с хешированием  
SHA1 и MD5»

Вариант № 2

Студент ИУ7-73Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Лысцев Н. Д.  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

Чиж И. С.  
(И. О. Фамилия)

2024 г.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Электронная цифровая подпись . . . . .	4
1.2 Алгоритм RSA . . . . .	4
1.3 Алгоритм SHA1 . . . . .	6
<b>2 Конструкторский раздел</b>	<b>9</b>
<b>3 Технологический раздел</b>	<b>12</b>
3.1 Требования к программному обеспечению . . . . .	12
3.2 Средства реализации . . . . .	12
3.3 Сведения о модулях программы . . . . .	12
3.4 Тестирование . . . . .	12
3.5 Реализации алгоритмов . . . . .	12
<b>ЗАКЛЮЧЕНИЕ</b>	<b>17</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>18</b>

# ВВЕДЕНИЕ

Целью данной лабораторной работы является реализация программы создания и проверки электронной подписи для документа с использованием алгоритма RSA и алгоритма хеширования SHA1.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) описать алгоритмы RSA и SHA1;
- 2) выбрать средства программной реализации;
- 3) реализовать данные алгоритмы.

# 1 Аналитическая часть

В этом разделе будут рассмотрен криптографический алгоритм RSA, алгоритм хеширования SHA1 понятие электронной подписи и принципы её получения и проверки с использованием алгоритмов RSA, SHA1.

## 1.1 Электронная цифровая подпись

Электронная цифровая подпись (ЭЦП) позволяет подтвердить авторство электронного документа.

Создание ЭЦП с использованием криптографического алгоритма RSA и алгоритма хеширования SHA1 происходит следующим образом:

- 1) файл, который необходимо подписать, хешируется при помощи SHA1;
- 2) полученный на предыдущем этапе хеш шифруется с использованием закрытого ключа RSA;
- 3) значение подписи — результат шифрования.

Проверка подлинности ЭЦП с использованием криптографического алгоритма RSA и алгоритма хеширования SHA1 происходит следующим образом:

- 1) файл, который необходимо подписать, хешируется при помощи SHA1;
- 2) переданная подпись расшифровывается 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 (аббревиатура от фамилий *Rivest*, *Shamir* и *Adleman*) — ассиметричный алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел. В алгоритме RSA используется 2 ключа — открытый (публичный) и закрытый (приватный).

В ассиметричной криптографии и алгоритме RSA, в частности, открытый и закрытый ключи являются двумя частями одного целого и неразрывны

друг с другом. Для шифрования информации используется открытый ключ, а для её расшифровки закрытый.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи.

Открытый и закрытый ключи RSA генерируются следующим образом:

- 1) выбираются два разных случайных простых числа  $p$  и  $q$  заданного размера;
- 2) вычисляется их произведение  $n = p \cdot q$ , называемое модулем;
- 3) вычисляется значение функции Эйлера от числа  $n$  по следующей формуле:

$$\phi(n) = (p - 1) \cdot (q - 1)$$

- 4) выбирается целое число  $e$  ( $1 < e < \phi(n)$ ), взаимно простое со значением  $\phi(n)$ , называемое *открытой экспонентой*;
- 5) вычисляется число  $d$ , называемое *закрытой экспонентой* по следующей формуле:

$$d = e^{-1} \bmod(\phi(n))$$

- 6) пара  $(e, n)$  публикуются в качестве открытого ключа RSA;
- 7) пара  $(d, n)$  представляет собой закрытый ключ RSA.

Перевод исходного сообщения  $m$  в зашифрованное сообщение  $c$  производится по формуле

$$c = E(m, k_1) = E(m, n, e) = m^e \bmod(n)$$

Расшифровка выполняется по формуле

$$m = D(c, k_2) = D(c, n, d) = c^d \bmod(n)$$

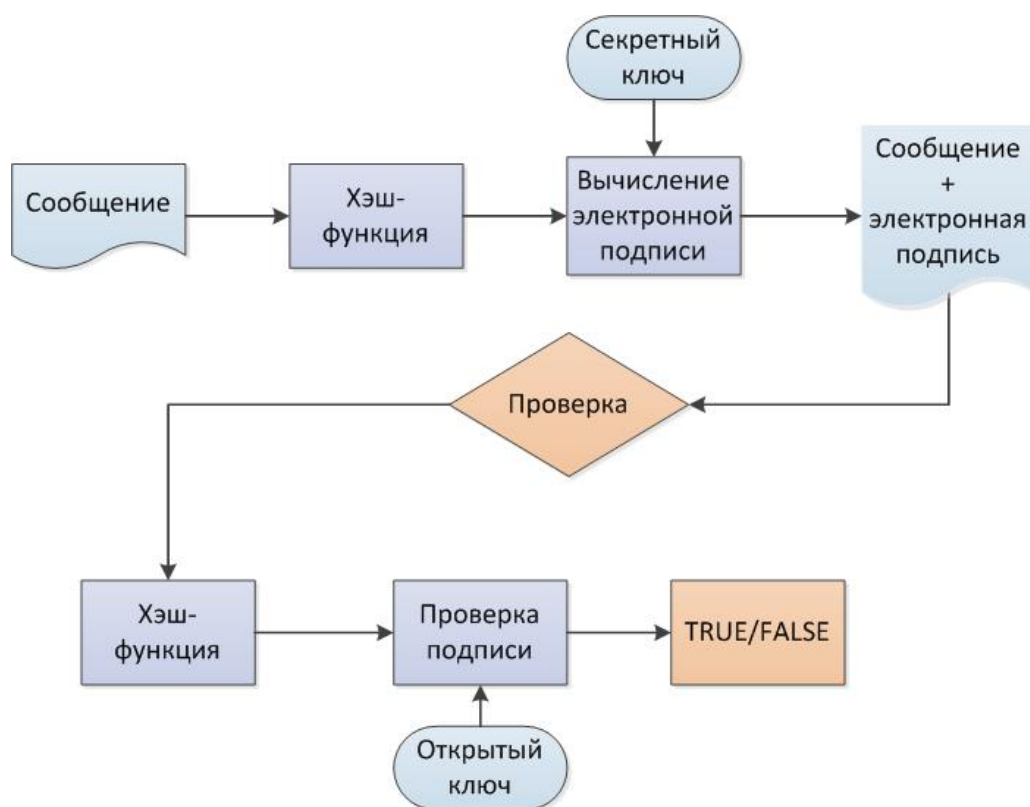


Рисунок 1.1 – RSA

### 1.3 Алгоритм SHA1

SHA1 (англ. *Secure Hash Algorithm 1*) — алгоритм криптографического хеширования. Для входного сообщения произвольной длины алгоритм генерирует 160-битное (20 байт) хеш-значение, называемое также дайджестом сообщения, которое обычно отображается как шестнадцатеричное число длиной в 40 цифр.

Алгоритм состоит из следующих шагов:

- 1) **добавление недостающих битов.** Сообщение расширяется таким образом, чтобы его длина была кратна 448 по модулю 512 (длина  $\equiv 448 \pmod{512}$ ). Добавление осуществляется всегда, даже если сообщение уже имеет нужную длину. Таким образом, число добавляемых битов находится в диапазоне от 1 до 512;
- 2) **расширение.** Результатом первых двух шагов является сообщение, длина которого кратна 512 битам. Расширенное сообщение может быть представлено как последовательность 512-битных блоков  $Y_0, Y_1, \dots, Y_{L-1}$ , так что общая длина расширенного сообщения есть  $L * 512$  бит. Таким

образом, результат кратен шестнадцати 32-битным словам.

- 3) **инициализация SHA-1 буфера.** Используется 160-битный буфер для хранения промежуточных и окончательных результатов хэш-функции. Буфер может быть представлен как пять 32-битных регистров A, B, C, D и E. Эти регистры инициализируются следующими шестнадцатеричными числами:  $A = 67452301$ ,  $B = EFCDAB89$ ,  $C = 98BADCFE$ ,  $D = 10325476$ ,  $E = C3D2E1F0$ .
- 4) **обработка сообщения в 512-битных (16-словных) блоках.** Основной алгоритма является модуль, состоящий из 80 циклических обработок, обозначенный как  $H_{SHA}$ . Все 80 циклических обработок имеют одинаковую структуру.
- 5) **выход.** После обработки всех 512-битных блоков выходом L-ой стадии является 160-битный дайджест сообщения.

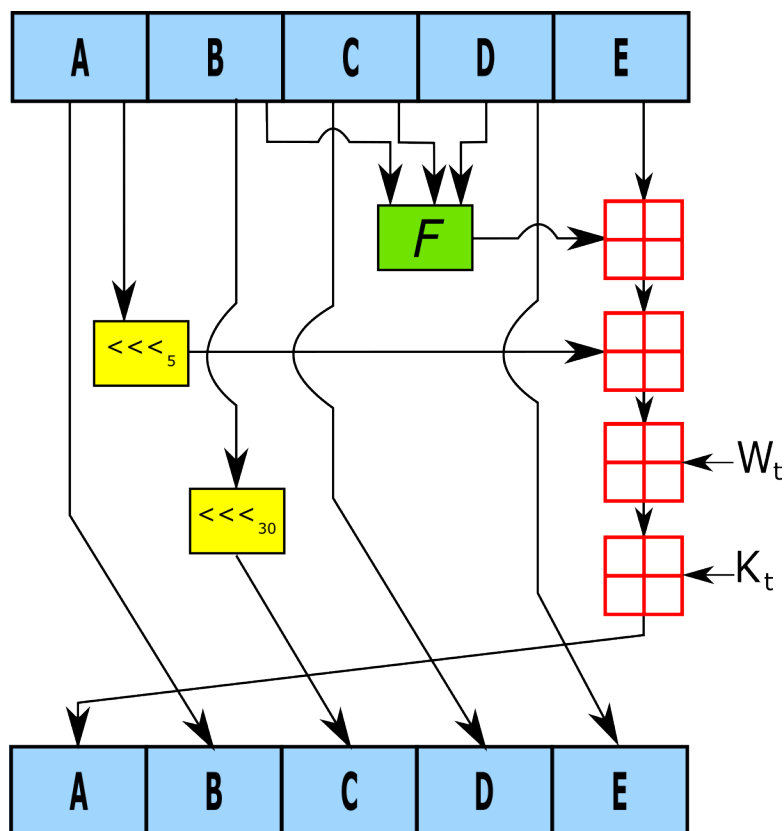


Рисунок 1.2 – Одна итерация алгоритма SHA-1

## Вывод

В этом разделе был рассмотрен криптографический алгоритм RSA, алгоритм хеширования SHA1 понятие электронной подписи и принципы её получения и проверки с использованием алгоритмов RSA, SHA1.



## 2 Конструкторский раздел

На рисунках 2.1-?? представлены схемы алгоритма RSA и SHA1.

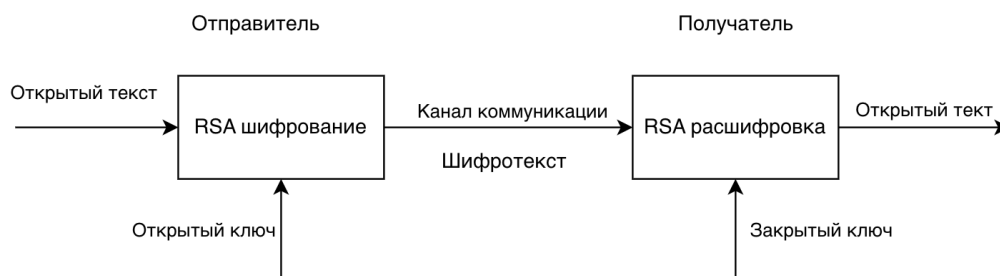


Рисунок 2.1 – Схема шифровального алгоритма RSA

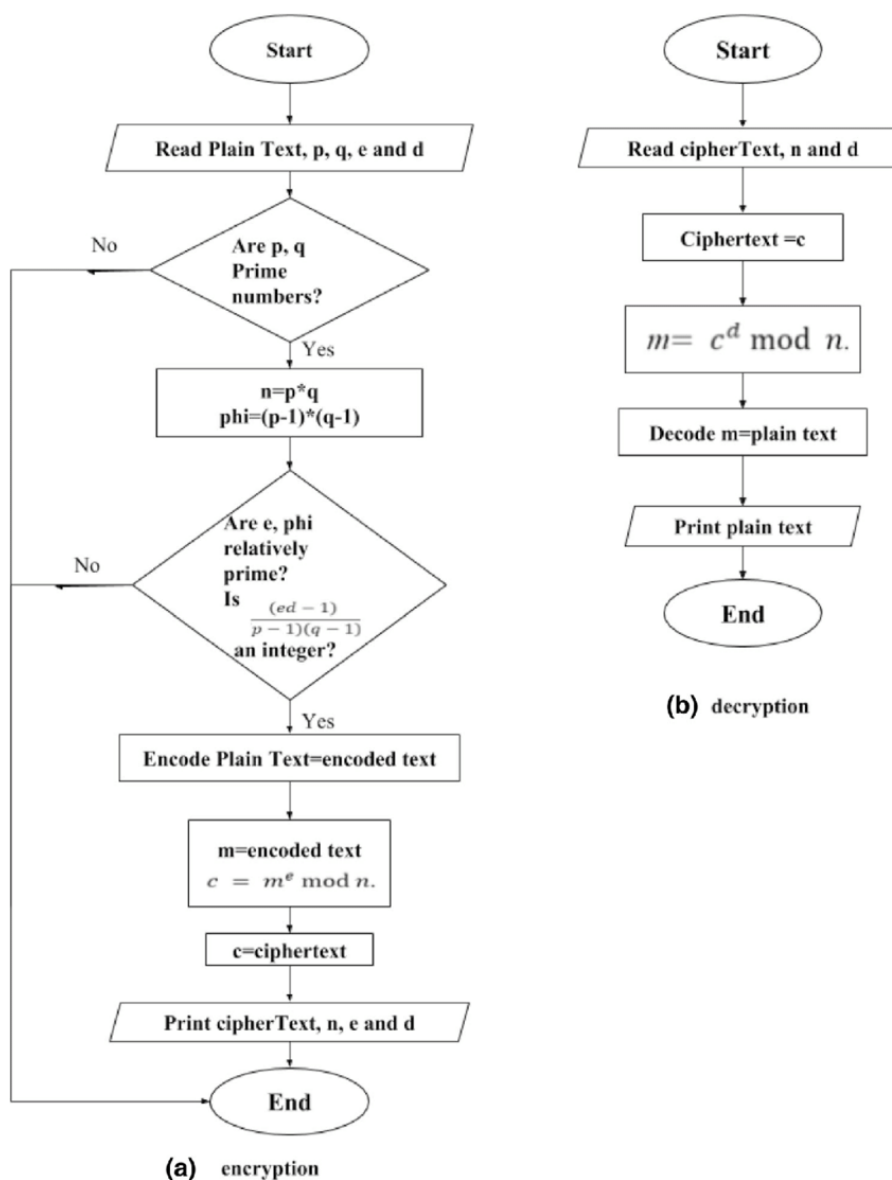


Рисунок 2.2 – Схема шифровального алгоритма работы RSA



Рисунок 2.3 – Схема алгоритма SHA1

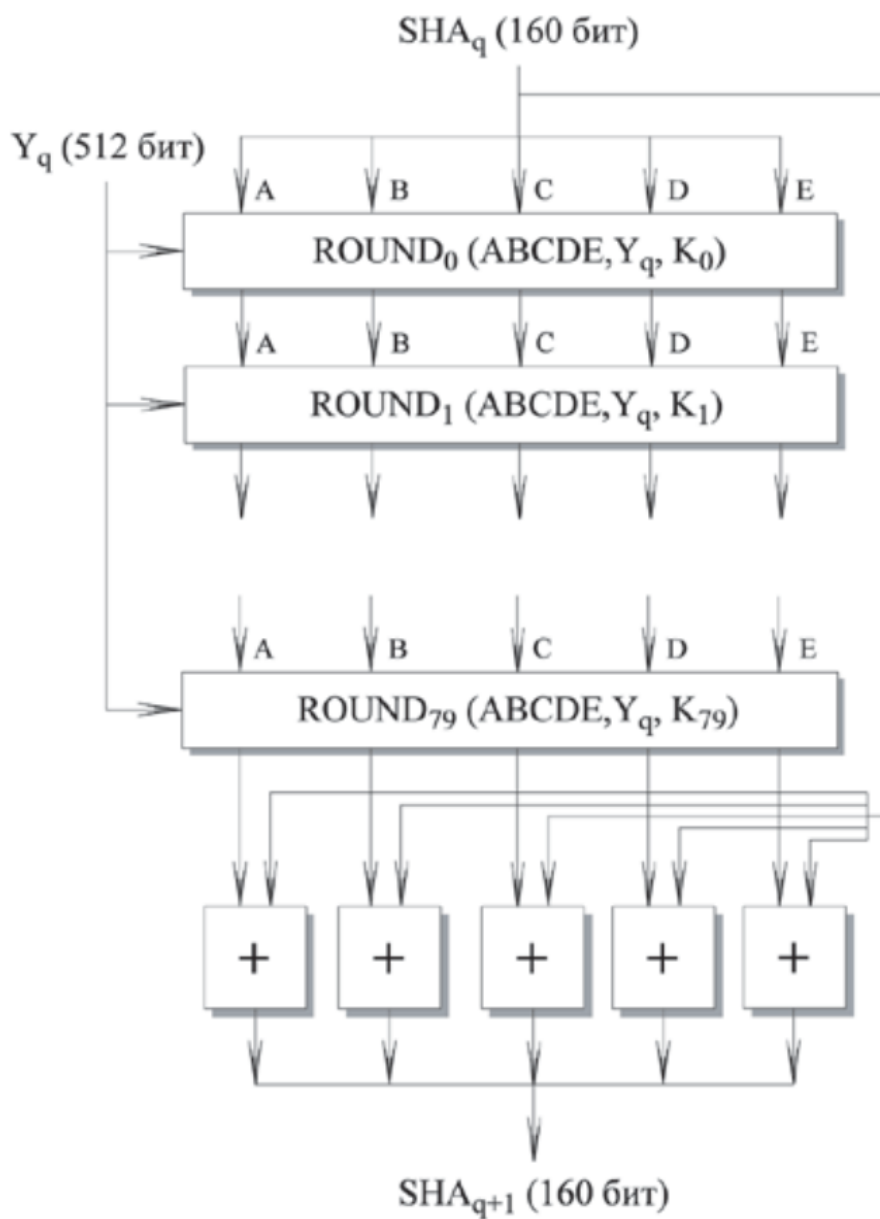


Рисунок 2.4 – Схема раунда алгоритма SHA1

## Вывод

В данном разделе были представлены схемы алгоритма RSA и SHA1.

## 3 Технологический раздел

В данном разделе будут перечислены требования к программному обеспечению, средства реализации и листинги кода.

### 3.1 Требования к программному обеспечению

К программе предъявляется ряд требований:

- программа должна предоставлять возможность шифрования и расшифровки произвольного файла;
- программа должна корректно работать с пустым однобайтовым файлом;
- программа должна уметь обрабатывать файл архива.

### 3.2 Средства реализации

В качестве языка программирования для этой лабораторной работы был выбран *C++* [1].

### 3.3 Сведения о модулях программы

Программа состоит из трех модулей:

- `main.cpp` — файл, содержащий точку входа в программу;
- `SHA1.cpp` — модуль, реализующий алгоритм SHA1;
- `RSA.cpp` — модуль, реализующий алгоритм RSA.

### 3.4 Тестирование

Все тесты с файлами были успешно пройдены.

### 3.5 Реализации алгоритмов

На листинге 3.1 представлена функция, реализующая генерацию публичного и приватного RSA-ключа.

На листинге 3.2 представлены функции создания подписи, создания прототипа сообщения и проверки подписи.

На листингах 3.3-3.4 представлена функция хеширования алгоритмом SHA1.

Листинг 3.1 – Функция, реализующая генерацию публичного и приватного RSA-ключа

```
pair<PrivateKey, PublicKey> RSA::genPublicAndSecretKeys() {
    mpz_class minVal(1), maxVal(string(40, '9'));

    auto [p, q] = RSA::_getRandomPrimePQ(minVal, maxVal);

    mpz_class n = p * q;
    mpz_class euler = (p - 1) * (q - 1);

    mpz_class e;

    while (true) {
        e = RSA::_genRandomNumber(1, euler);

        mpz_class resultGcd;
        mpz_gcd(resultGcd.get_mpz_t(), e.get_mpz_t(),
            euler.get_mpz_t());

        if (resultGcd == 1) {
            break;
        }
    }

    mpz_class g, s, t;

    mpz_gcdext(g.get_mpz_t(), s.get_mpz_t(), t.get_mpz_t(),
        e.get_mpz_t(), euler.get_mpz_t());

    mpz_class d(s);

    PublicKey publicKey{.e = e, .n = n};
    PrivateKey privateKey{.d = d, .n = n};

    return {privateKey, publicKey};
}
```

Листинг 3.2 – Функции создания подписи, создания прототипа сообщения и проверки подписи

```
mpz_class RSA::calcSignature(const mpz_class &d, const mpz_class
    &n, const mpz_class &hash) {
    mpz_class result;

    mpz_powm(result.get_mpz_t(), hash.get_mpz_t(),
        d.get_mpz_t(), n.get_mpz_t());

    return result;
}

mpz_class RSA::calcPrototypeSignature(const mpz_class &e, const
    mpz_class &n, const mpz_class &signature) {
    mpz_class result;

    mpz_powm(result.get_mpz_t(), signature.get_mpz_t(),
        e.get_mpz_t(), n.get_mpz_t());

    return result;
}

bool RSA::isCorrectSignature(const mpz_class &originalHash,
    const mpz_class &newHash) {
    if (originalHash != newHash) {
        return false;
    }

    return true;
}
```

### Листинг 3.3 – Функции хеширования алгоритмом SHA1 (начало)

```
mpz_class SHA1::_hash(vector<uint8_t> &data) {
    uint32_t h0 = 0x67452301;
    uint32_t h1 = 0xEFCDAB89;
    uint32_t h2 = 0x98BADCFE;
    uint32_t h3 = 0x10325476;
    uint32_t h4 = 0xC3D2E1F0;

    SHA1::_preprocessing(data);

    array<uint32_t, 80> words = {0};

    for (int i = 0; i < data.size(); i += 64) {
        vector<uint8_t> chunk = SHA1::_getSlice(data, i, i + 64);

        for (uint8_t wid = 0; wid < 16; ++wid) {
            words[wid] = 0;

            for (uint8_t cid = 0; cid < 4; cid++) {
                words[wid] = (words[wid] << 8) + chunk[wid * 4 +
                    cid];
            }
        }

        for (uint8_t j = 16; j < 80; ++j) {
            words[j] = SHA1::_cyclicShiftLeft(words[j - 3] ^
                words[j - 8] ^ words[j - 14] ^ words[j - 16], 1);
        }

        uint32_t a = h0, b = h1, c = h2, d = h3, e = h4;

        for (uint8_t ind = 0; ind < 80; ++ind) {
            uint32_t f = 0, k = 0;

            if (ind < 20) {
                f = (b & c) | ((~b) & d);
                k = 0x5A827999;
            } else if (ind < 40) {
                f = b ^ c ^ d;
                k = 0x6ED9EBA1;
            } else if (ind < 60) {
```

### Листинг 3.4 – Функции хеширования алгоритмом SHA1 (конец)

```
        f = (b & c) | (b & d) | (c & d);
        k = 0x8F1BBCDC;
    } else {
        f = b ^ c ^ d;
        k = 0xCA62C1D6;
    }

    uint32_t temp = SHA1::_cyclicShiftLeft(a, 5) + f + e
        + k + words[i];
    e = d;
    d = c;
    c = SHA1::_cyclicShiftLeft(b, 30);
    b = a;
    a = temp;
}

h0 = h0 + a;
h1 = h1 + b;
h2 = h2 + c;
h3 = h3 + d;
h4 = h4 + e;
}

string resultStr = mpz_class(h0).get_str(16) +
    mpz_class(h1).get_str(16) +
    mpz_class(h2).get_str(16) +
    mpz_class(h3).get_str(16) +
    mpz_class(h4).get_str(16);

return mpz_class(resultStr, 16);
}
```

## Вывод

В данном разделе были перечислены требования к программному обеспечению, средства реализации и листинги кода.



## ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были решены следующие задачи:

- 1) описаны алгоритмы RSA и SHA1;
- 2) выбраны средства программной реализации;
- 3) реализованы данные алгоритмы.

Цель работы, а именно реализация программы создания и проверки электронной подписи для документа с использованием алгоритма RSA и алгоритма хеширования SHA1, также была достигнута.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник по языку C++ [Электронный ресурс]. — Режим доступа: <https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170> (дата обращения: 28.09.2022).