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: «Цифровая подпись (RSA)»

Вариант №1

Студент ИУ7-73Б _____ Авдейкина В. П.
(Группа) (Подпись, дата) (Фамилия И.О.)

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

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной лабораторной работы — реализовать программу для создания и проверки электронной подписи для документа с использованием алгоритма RSA и алгоритмов хеширования MD5/SHA1 (по варианту).

Для достижения поставленной цели требуется решить следующие задачи:

1) описать алгоритмы RSA и MD5/SHA1;
2) спроектировать описанные алгоритмы;
3) выбрать необходимые для разработки средства и разработать реализацию спроектированных алгоритмов.

Требования к выполнению лабораторной работы:

— обеспечить шифрование и расшифровку произвольного файла, а также текстового сообщения с использованием разработанной программы;
— необходимо предусмотреть работу программы с пустым, однобайтовым файлом;
— должна быть возможность обработки файла архива (rar, zip или др.).

1 Теоретическая часть

1.1 Алгоритм MD5

На рисунках 1–2 представлена общая схема реализации алгоритма хеширования MD5.

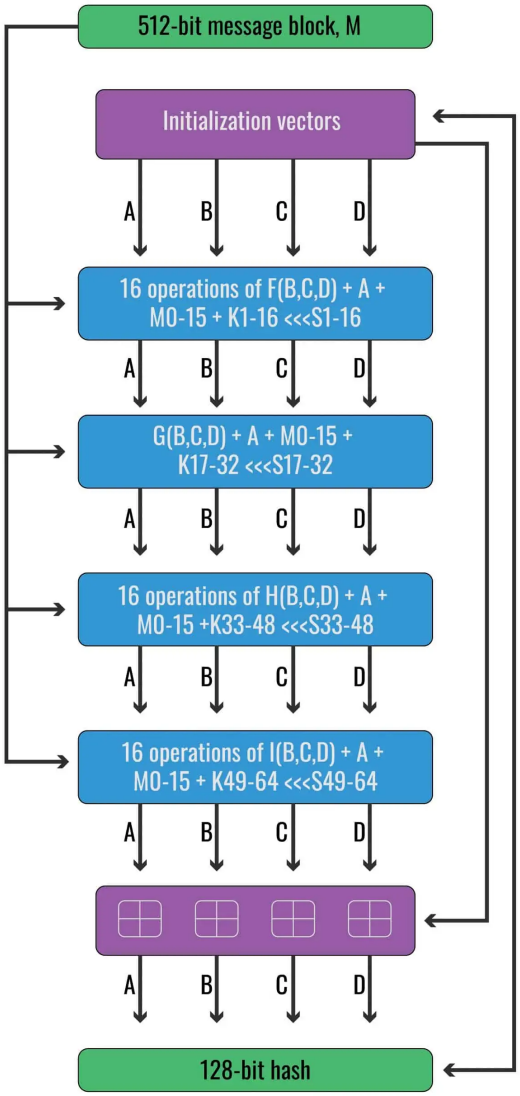

Рисунок 1 — Общая схема реализации алгоритма хеширования MD5

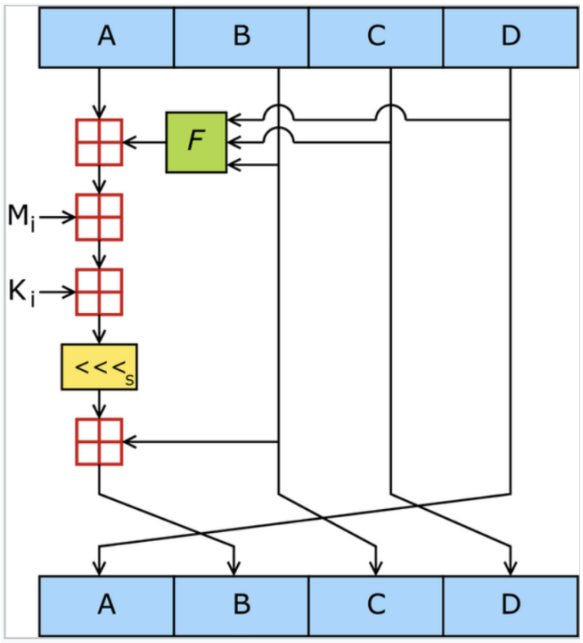

Схема работы алгоритма MD5. F — нелинейная функция. M_i обозначает 32-битный блок входного сообщения, а K_i — 32-битную константу. $<<<_s$ обозначает циклический сдвиг влево на s бит. ⊞ обозначает сложение по модулю 2^{32}. F зависит от раунда, K_i и s меняются каждую операцию.

Рисунок 2 — Схема реализации алгоритма хеширования MD5

1.2 Алгоритм RSA

На рисунках 3–4 представлена общая схема реализации алгоритма шифрования RSA.

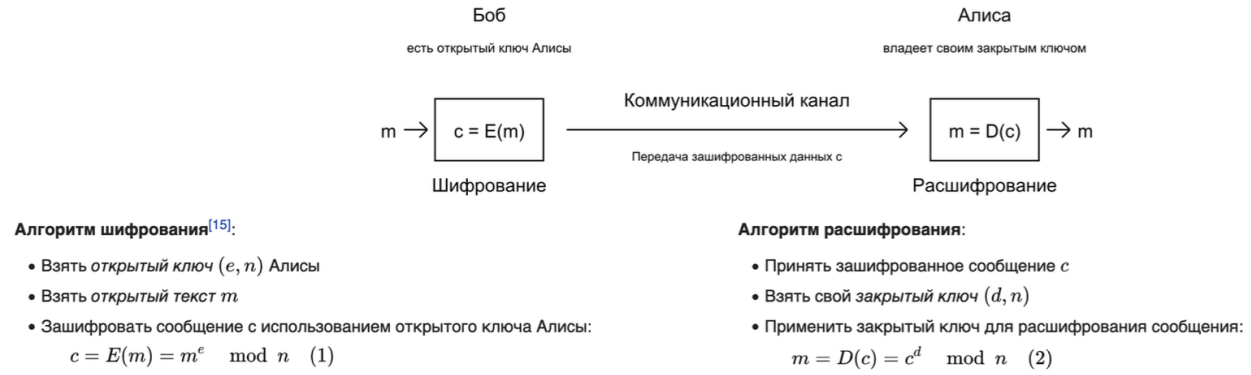

Рисунок 3 — Общая схема реализации алгоритма шифрования RSA

Рисунок 4 — Общая схема реализации алгоритма генерации ключей RSA

1.3 Цифровая подпись

На рисунке 5 представлена общая схема алгоритма электронной подписи и проверки электронной подписи.

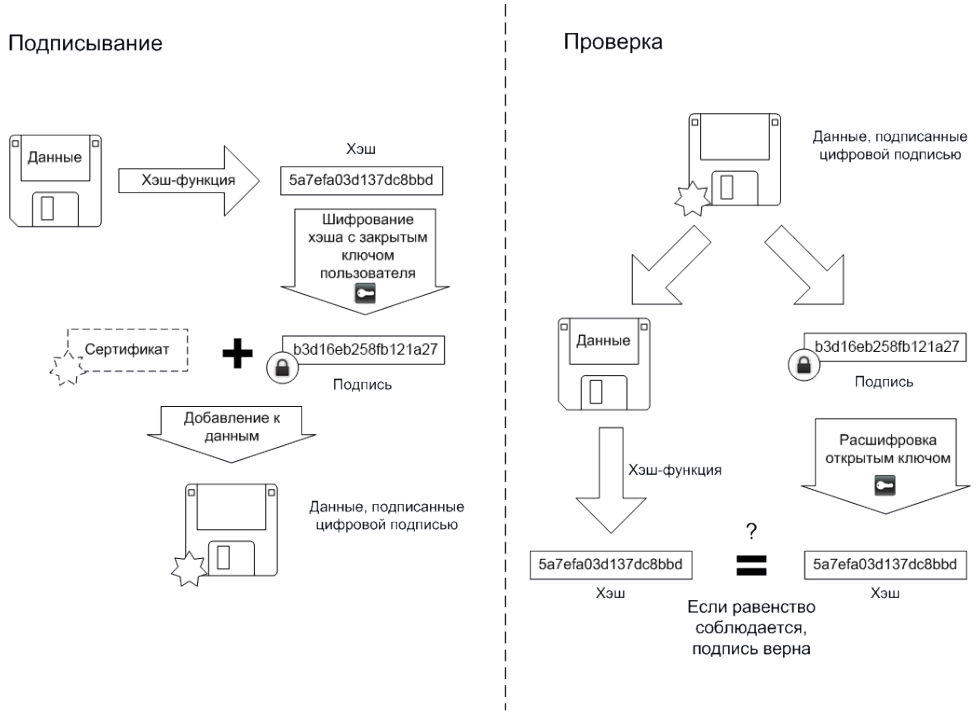

Рисунок 5 — Общая схема алгоритма электронной подписи и проверки электронной подписи

2 Практический раздел

2.1 Листинг алгоритма MD5

2.2 Листинг алгоритма RSA

2.3 Листинг алгоритма цифровой подписи

2.4 Тестирование

Корректность алгоритма проверялось путем применения дешифрации на шифрованное сообщение.

Тестирование было проведено на файлах с типами: текстовый (txt), графический (jpeg, png), архив (zip), несуществующий (ubc). Также, был проведен тест с повреждением зашифрованного файла.

В таблице 1 представлены тестовые данные.

Таблица 1 — Тестовые данные

Номер теста	Тип файла	Содержимое файла
1	txt	Тест проверка
2	ubc	∅
3	zip	Файлы с тестов 1, 2, 4
4	png	фото 1
5	jpeg	фото 2

Листинг 1 — Алгоритм MD5

```
typedef struct {
        uint64_t cur_len;
        uint8_t cur_input[64];

        uint32_t parts[4];

        uint8_t digest[16];
} hasher_t;

void init(hasher_t *h) {
        h->cur_len = (uint64_t) 0;

        h->parts[0] = 0x67452301;
        h->parts[1] = 0xefcdab89;
        h->parts[2] = 0x98badcfe;
        h->parts[3] = 0x10325476;
}

void step(uint32_t *buf, const uint32_t *input) {
        uint32_t a = buf[0];
        uint32_t b = buf[1];
        uint32_t c = buf[2];
        uint32_t d = buf[3];

        for (unsigned int i = 0; i < 64; i++) {
                uint32_t f, g;

                if (i <= 15) {
                        f = (b & c) | (~b & d);
                        g = i % 16;
                }
                else if (i <= 31) {
                        f = (b & d) | (c & ~d);
                        g = ((i * 5) + 1) % 16;
                }
                else if (i <= 47) {
                        f = (b ^ c ^ d);
                        g = ((i * 3) + 5) % 16;
                }
                else {
                        f = (c ^ (b | ~d));
                        g = (i * 7) % 16;
                }

                f = f + a + k[i] + input[g];
                a = d;
                d = c;
                c = b;
                b = b + rotate_left(f, s[i]);
        }

        buf[0] += a;
        buf[1] += b;
```

Листинг 2 — Алгоритм RSA (часть 1)

```
int rsa_with_key(const char *buf, int bytes, rsa_key_t *key, char *result) {
        bignum *res, *plain;
        int enc_bytes;

        plain = from_bin(buf, bytes);
        res = bignum_alloc();
        bignum_pow_mod(res, plain, key->exponent, key->modulus);

        enc_bytes = res->length * sizeof(uint32_t);

        for (int i = 0; i < enc_bytes; i++)
        result[i] = ((char *) res->data)[i];

        bignum_free(res);
        bignum_free(plain);

        return enc_bytes;
}
```

Листинг 3 — Алгоритм RSA (часть 2 — генерация ключей)

```
int rsa_generate_keys(char *private_filename, char *public_filename, int bytes)
        bignum *p = bignum_alloc();
        bignum *q = bignum_alloc();
        bignum *n = bignum_alloc();
        bignum *d = bignum_alloc();
        bignum *e = bignum_alloc();
        bignum *phi = bignum_alloc();

        random_prime(bytes, p);
        random_prime(bytes, q);

        bignum_multiply(n, p, q);
        bignum_isubtract(p, &NUMS[1]);
        bignum_isubtract(q, &NUMS[1]);
        bignum_multiply(phi, p, q);

        find_e(phi, e);
        find_d(e, phi, d);

        rsa_key_t private = {
                .exponent = e,
                .modulus = n,
        };

        rsa_key_t public = {
                .exponent = d,
                .modulus = n,
        };

        if (rsa_write_key(private_filename, &private) != EXIT_SUCCESS) {
                return EXIT_FAILURE;
        }

        if (rsa_write_key(public_filename, &public) != EXIT_SUCCESS) {
                return EXIT_FAILURE;
        }

        bignum_free(p);
        bignum_free(q);
        bignum_free(phi);

        return EXIT_SUCCESS;
}
```

Листинг 4 — Реализация алгоритма цифровой подписи на основе MD5 и RSA

```
int sign(char *filename, char *sign_filename, char *key_filename) {
uint8_t hash[1024] = { 0 };
if (md5(filename, hash) != EXIT_SUCCESS) {
        printf("Cannot compute MD5.\n");
        return EXIT_FAILURE;
}

printf("File checksum is ");
md5_print(hash);
printf(".\n");

uint8_t sign_content[1024] = { 0 };
int size;
if ((size = rsa(hash, 16, "key", sign_content)) < 0) {
        printf("Cannot compute RSA.\n");
        return EXIT_FAILURE;
}

if (write_file(sign_filename, sign_content, size) != EXIT_SUCCESS) {
        printf("Cannot write file.\n");
        return EXIT_FAILURE;
}

printf("Sign file is %s.\n", sign_filename);

return EXIT_SUCCESS;
}

int check(char *filename, char *sign_filename, char *key_filename) {
uint8_t hash[16] = { 0 };
if (md5(filename, hash) != EXIT_SUCCESS) {
        printf("Cannot compute MD5.\n");
        return EXIT_FAILURE;
}
printf("File checksum is ");
md5_print(hash);
printf(".\n");

uint8_t* sign_content;
int sign_size;
if (read_file(sign_filename, &sign_content, &sign_size) != EXIT_SUCCESS) {
        printf("Read sign file.\n");
        return EXIT_FAILURE;
}

uint8_t hash_from_sign[1024] = { 0 };
if (rsa(sign_content, sign_size, key_filename, hash_from_sign) < 0) {
        printf("Cannot compute RSA.\n");
        return EXIT_FAILURE;
}

printf("Checksum from sign ");
md5_print(hash_from_sign);
```

ЗАКЛЮЧЕНИЕ

Цель работы была достигнута — была реализована программа для создания и проверки электронной подписи для документа с использованием алгоритма RSA и алгоритмов хеширования MD5/SHA1 (по варианту).

В ходе работы были решены следующие задачи:

1) описаны алгоритмы RSA и MD5/SHA1;
2) спроектированы описанные алгоритмы;
3) выбраны необходимые для разработки средства и разработаны реализации спроектированных алгоритмов.