



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»  
Вариант № 1 (MD5)

Студент ИУ7-72Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Е. О. Карпова  
(И. О. Фамилия)

Преподаватель

\_\_\_\_\_  
(Подпись, дата)

И. С. Чиж  
(И. О. Фамилия)

2023 г.

# 1 Теоретический раздел

## 1.1 Алгоритм MD5

На рисунках 1.1–1.2 представлена общая схема реализации алгоритма хеширования MD5.

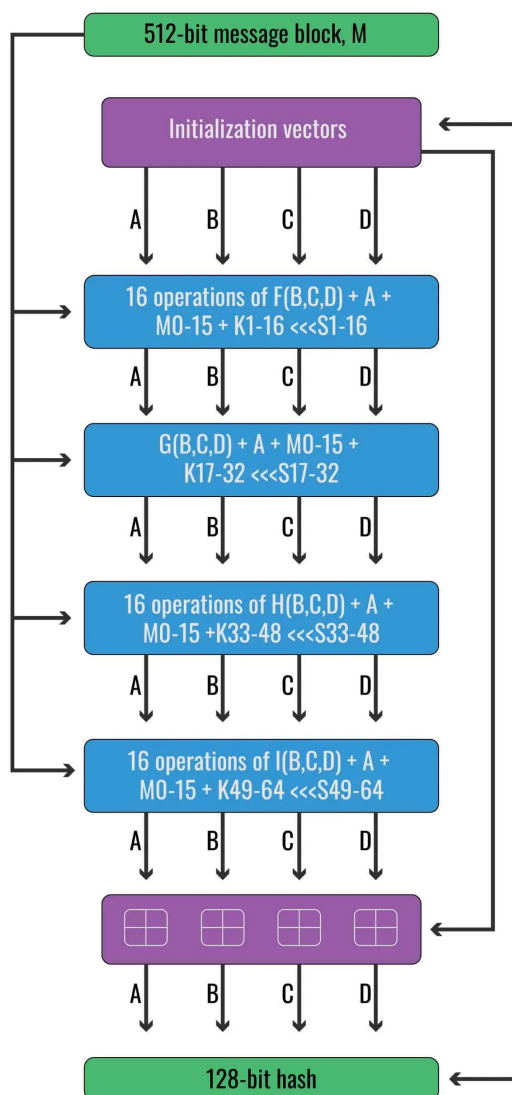


Рисунок 1.1 – Общая схема реализации алгоритма хеширования MD5

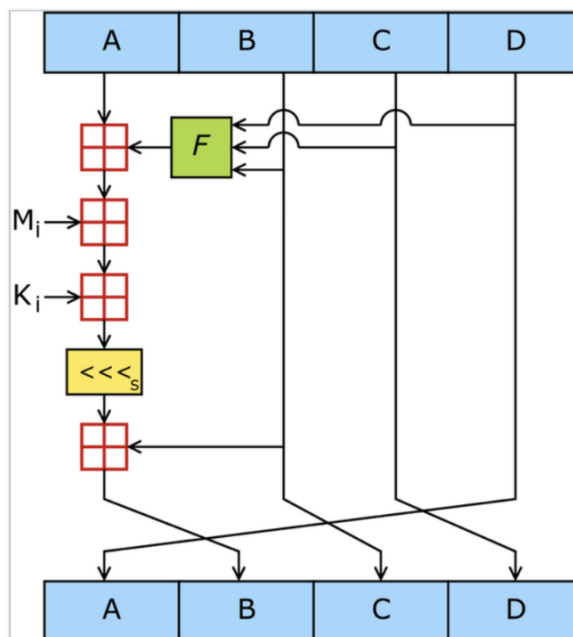


Схема работы алгоритма MD5.  $F$  — нелинейная функция.  $M_i$  обозначает 32-битный блок входного сообщения, а  $K_i$  — 32-битную константу.  $\lll_s$  обозначает **циклический сдвиг влево** на  $s$  бит.  $\boxplus$  обозначает сложение по модулю  $2^{32}$ .  $F$  зависит от раунда,  $K_i$  и  $s$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

Рисунок 1.2 – Схема реализации алгоритма хеширования MD5

## 1.2 Алгоритм RSA

На рисунках 1.3–1.4 представлена общая схема реализации алгоритма шифрования RSA.

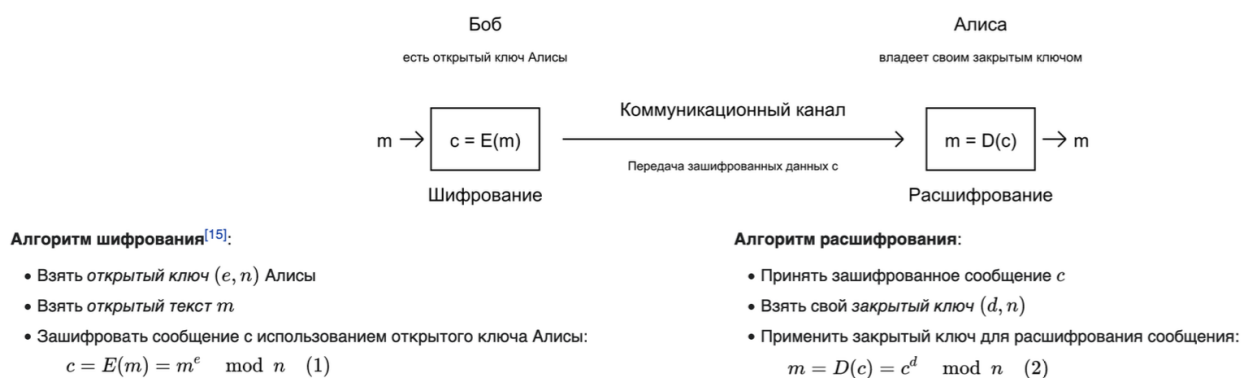


Рисунок 1.3 – Общая схема реализации алгоритма шифрования RSA



Рисунок 1.4 – Общая схема реализации алгоритма генерации ключей RSA

## 1.3 Цифровая подпись

На рисунке 1.5 представлена общая схема алгоритма электронной подписи и проверки электронной подписи.

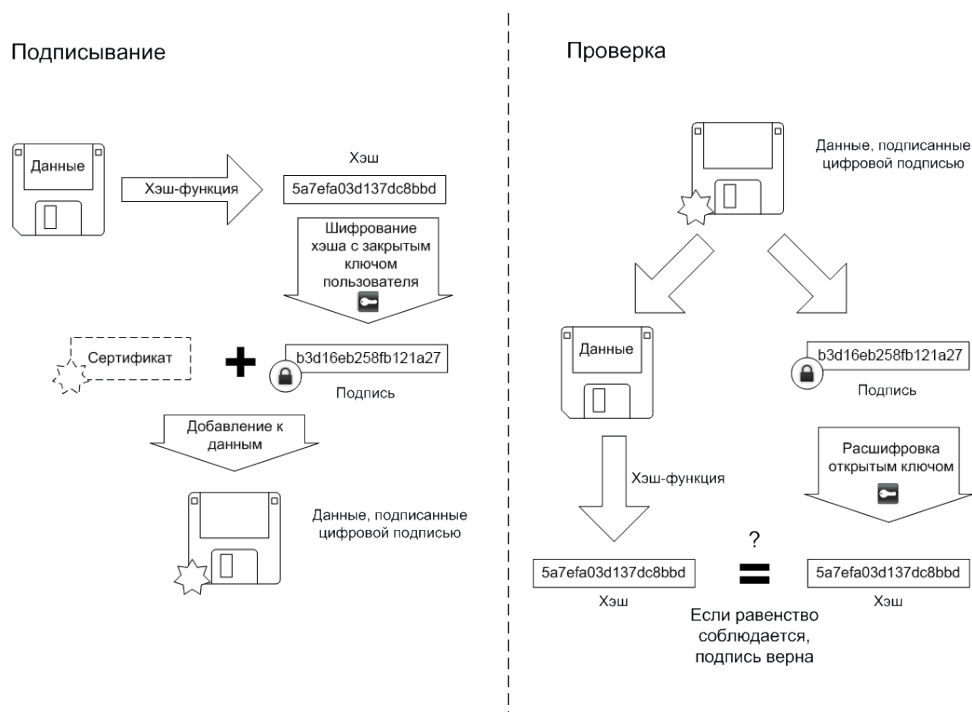


Рисунок 1.5 – Общая схема алгоритма электронной подписи и проверки электронной подписи

## 2 Практический раздел

### 2.1 Листинг алгоритма MD5

Листинг 2.1 – Реализация алгоритма MD5

```
1 typedef struct {
2     uint64_t cur_len;
3     uint8_t cur_input[64];
4
5     uint32_t parts[4];
6
7     uint8_t digest[16];
8 } hasher_t;
9
10 void init(hasher_t *h) {
11     h->cur_len = (uint64_t) 0;
12
13     h->parts[0] = 0x67452301;
14     h->parts[1] = 0xefcdab89;
15     h->parts[2] = 0x98badcfe;
16     h->parts[3] = 0x10325476;
17 }
18
19 void step(uint32_t *buf, const uint32_t *input) {
20     uint32_t a = buf[0];
21     uint32_t b = buf[1];
22     uint32_t c = buf[2];
23     uint32_t d = buf[3];
24
25     for (unsigned int i = 0; i < 64; i++) {
26         uint32_t f, g;
27
28         if (i <= 15) {
29             f = (b & c) | (~b & d);
30             g = i % 16;
31         }
32         else if (i <= 31) {
33             f = (b & d) | (c & ~d);
34             g = ((i * 5) + 1) % 16;
35         }
36         else if (i <= 47) {
```

```

37         f = (b ^ c ^ d);
38         g = ((i * 3) + 5) % 16;
39     }
40     else {
41         f = (c ^ (b | ~d));
42         g = (i * 7) % 16;
43     }
44
45     f = f + a + k[i] + input[g];
46     a = d;
47     d = c;
48     c = b;
49     b = b + rotate_left(f, s[i]);
50 }
51
52 buf[0] += a;
53 buf[1] += b;
54 buf[2] += c;
55 buf[3] += d;
56 }
57
58 void update(hasher_t *h, const uint8_t *buf, size_t len) {
59     uint32_t input[16];
60     uint64_t offset = h->cur_len % 64;
61     h->cur_len += (uint64_t) len;
62
63     for (unsigned int i = 0; i < len; i++) {
64         h->cur_input[offset++] = buf[i];
65
66         if (offset % 64 == 0) {
67             for (unsigned int j = 0; j < 16; ++j) {
68                 input[j] = (uint32_t) (h->cur_input[(j * 4) + 3])
69                     << 24 |
70                     (uint32_t) (h->cur_input[(j * 4) + 2])
71                     << 16 |
72                     (uint32_t) (h->cur_input[(j * 4) + 1])
73                     << 8 |
74                     (uint32_t) (h->cur_input[(j * 4)]);
75             }
76             step(h->parts, input);
77             offset = 0;

```

```

75     }
76 }
77 }
78
79 void finalize(hasher_t *h) {
80     uint32_t input[16];
81     unsigned int offset = h->cur_len % 64;
82     unsigned int padding_length = offset < 56 ? 56 - offset : (56 +
83         64) - offset;
84
85     update(h, padding, padding_length);
86     h->cur_len -= (uint64_t) padding_length;
87
88     for (unsigned int j = 0; j < 14; ++j) {
89         input[j] = (uint32_t) (h->cur_input[(j * 4) + 3]) << 24 |
90             (uint32_t) (h->cur_input[(j * 4) + 2]) << 16 |
91             (uint32_t) (h->cur_input[(j * 4) + 1]) << 8 |
92             (uint32_t) (h->cur_input[(j * 4)]);
93     }
94     input[14] = (uint32_t) (h->cur_len * 8);
95     input[15] = (uint32_t) ((h->cur_len * 8) >> 32);
96
97     step(h->parts, input);
98
99     for (unsigned int i = 0; i < 4; ++i) {
100         h->digest[(i * 4) + 0] = (uint8_t) ((h->parts[i] & 0
101             x000000FF));
102         h->digest[(i * 4) + 1] = (uint8_t) ((h->parts[i] & 0
103             x0000FF00) >> 8);
104         h->digest[(i * 4) + 2] = (uint8_t) ((h->parts[i] & 0
105             x00FF0000) >> 16);
106         h->digest[(i * 4) + 3] = (uint8_t) ((h->parts[i] & 0
107             xFF000000) >> 24);
108     }
109 }

```

## 2.2 Листинг алгоритма RSA

Листинг 2.2 – Реализация алгоритма RSA

```

1 int rsa_with_key(const char *buf, int bytes, rsa_key_t *key, char *
    result) {

```



```

2    bignum *res , *plain;
3    int enc_bytes;
4
5    plain = from_bin(buf, bytes);
6    res = bignum_alloc();
7    bignum_pow_mod(res, plain, key->exponent, key->modulus);
8
9    enc_bytes = res->length * sizeof(uint32_t);
10
11    for (int i = 0; i < enc_bytes; i++)
12        result[i] = ((char *) res->data)[i];
13
14    bignum_free(res);
15    bignum_free(plain);
16
17    return enc_bytes;
18 }

```

Листинг 2.3 – Реализация алгоритма RSA (генерация ключей)

```

1    int rsa_generate_keys(char *private_filename, char *public_filename
    , int bytes) {
2        bignum *p = bignum_alloc();
3        bignum *q = bignum_alloc();
4        bignum *n = bignum_alloc();
5        bignum *d = bignum_alloc();
6        bignum *e = bignum_alloc();
7        bignum *phi = bignum_alloc();
8
9        random_prime(bytes, p);
10       random_prime(bytes, q);
11
12       bignum_multiply(n, p, q);
13       bignum_isubtract(p, &NUMS[1]);
14       bignum_isubtract(q, &NUMS[1]);
15       bignum_multiply(phi, p, q);
16
17       find_e(phi, e);
18       find_d(e, phi, d);
19
20       rsa_key_t private = {
21           .exponent = e,
22           .modulus = n,

```

```

23     };
24
25     rsa_key_t public = {
26         .exponent = d,
27         .modulus = n,
28     };
29
30     if (rsa_write_key(private_filename, &private) != EXIT_SUCCESS)
31     {
32         return EXIT_FAILURE;
33     }
34
35     if (rsa_write_key(public_filename, &public) != EXIT_SUCCESS) {
36         return EXIT_FAILURE;
37     }
38
39     bignum_free(p);
40     bignum_free(q);
41     bignum_free(phi);
42
43     return EXIT_SUCCESS;
44 }

```

## 2.3 Листинг алгоритма цифровой подписи

Листинг 2.4 – Реализация алгоритма цифровой подписи на основе MD5 и RSA

```

1  int sign(char *filename, char *sign_filename, char *key_filename) {
2      uint8_t hash[1024] = { 0 };
3      if (md5(filename, hash) != EXIT_SUCCESS) {
4          printf("Cannot compute MD5.\n");
5          return EXIT_FAILURE;
6      }
7
8      printf("File checksum is ");
9      md5_print(hash);
10     printf("\n");
11
12     uint8_t sign_content[1024] = { 0 };
13     int size;
14     if ((size = rsa(hash, 16, "key", sign_content)) < 0) {
15         printf("Cannot compute RSA.\n");

```

```

16         return EXIT_FAILURE;
17     }
18
19     if (write_file(sign_filename, sign_content, size) !=
20         EXIT_SUCCESS) {
21         printf("Cannot write file.\n");
22         return EXIT_FAILURE;
23     }
24
25     printf("Sign file is %s.\n", sign_filename);
26
27     return EXIT_SUCCESS;
28 }
29
30 int check(char *filename, char *sign_filename, char *key_filename)
31 {
32     uint8_t hash[16] = { 0 };
33     if (md5(filename, hash) != EXIT_SUCCESS) {
34         printf("Cannot compute MD5.\n");
35         return EXIT_FAILURE;
36     }
37     printf("File checksum is ");
38     md5_print(hash);
39     printf(".\n");
40
41     uint8_t* sign_content;
42     int sign_size;
43     if (read_file(sign_filename, &sign_content, &sign_size) !=
44         EXIT_SUCCESS) {
45         printf("Read sign file.\n");
46         return EXIT_FAILURE;
47     }
48
49     uint8_t hash_from_sign[1024] = { 0 };
50     if (rsa(sign_content, sign_size, key_filename, hash_from_sign)
51         < 0) {
52         printf("Cannot compute RSA.\n");
53         return EXIT_FAILURE;
54     }
55
56     printf("Checksum from sign ");

```

```

53     md5_print(hash_from_sign);
54     printf(".\n");
55
56     for (int i = 0; i < 16; i++) {
57         if (hash_from_sign[i] != hash[i]) {
58             printf("File is corrupted.\n");
59             return EXIT_FAILURE;
60         }
61     }
62
63     printf("Check is successful.\n");
64
65     return EXIT_SUCCESS;
66 }

```



## 2.4 Тестирование

Корректность алгоритма проверялось путем применения дешифрации на зашифрованное 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 2.1 представлены тестовые данные.

Таблица 2.1 – Тестовые данные

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

## **Заключение**

В результате выполнения данной лабораторной работы был реализован в виде программы на языке Си алгоритм шифрования RSA и механизм цифровой подписи на основе алгоритма хеширования MD5.