Симметричное и ассиметричное шифрование

Группа АС-24-05
Хациев Саламхан
Хлебников Матвей
Водопьянов Павел
Шипкова Виктория
Долгачева Александра
Сагайдак Сергей
Бурханов Тахир
Бардин Глеб

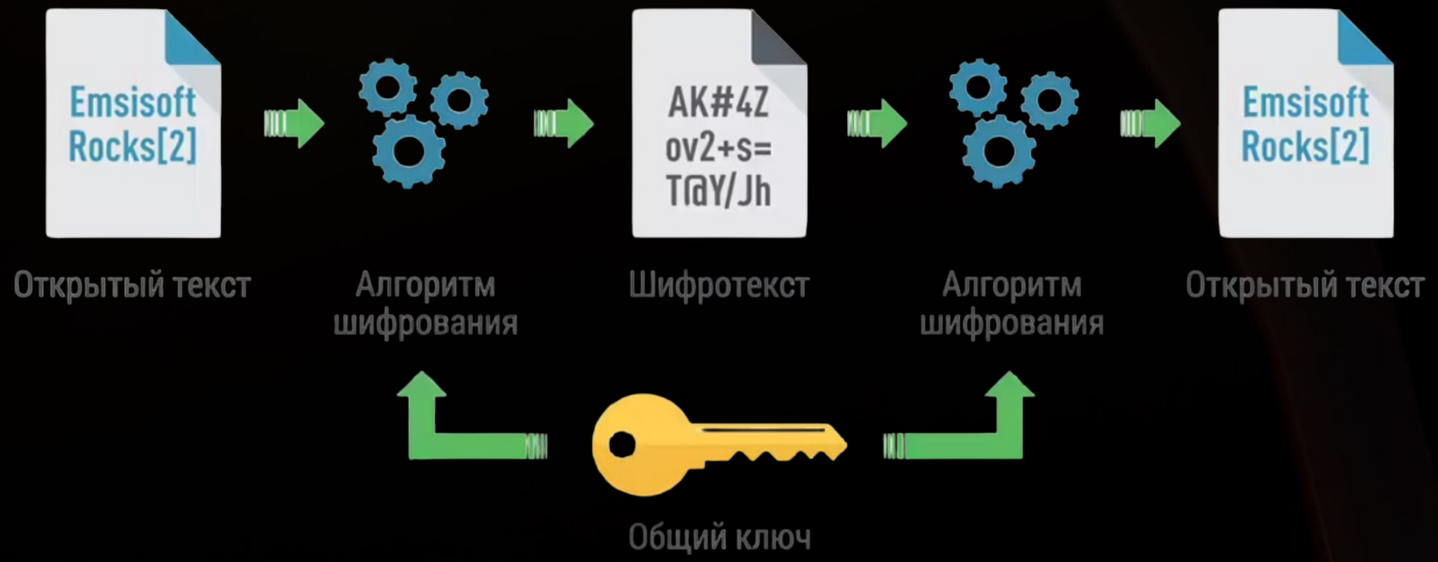
СИММЕТРИЧНОЕ ШИФРОВАНЕ

Для работы применяется всего один пароль.

Происходит всё следующим образом:

1. Существует алгоритм шифрования.
2. На его вход подаётся текст и ключ.
3. На выходе получаем зашифрованный текст.
4. Если хотим получить исходный текст,

применяется тот же самый ключ, но с алгоритмом дешифрования.



Алгоритм AES для симметричного шифрования

1. Ключ: Приводится к 16 байтам (дополняется нулями или обрезается)
2. Шифрование (3 раунда):
- Замена байт: byte = (byte + 123) & 0xFF
- Сдвиг данных: циклический сдвиг влево
- XOR с ключом
3. Дешифрование (3 раунда):
- XOR с ключом
- Обратный сдвиг: циклический сдвиг вправо
- Обратная замена: byte = (byte - 123) & 0xFF

Приведение ключа к 16 байтам

```
void SimpleAES::prepare_key() {
    if (key.size() < 16) {
        while (key.size() < 16) {
            key.push_back(0x00);
        }
    }
    else if (key.size() > 16) {
        key.resize(16);
    }
}
```

Сдвиг данных влево

```
void SimpleAES::shift_data(vector<unsigned char>& data) {
    if (data.empty()) return;

    unsigned char first = data[0];
    for (size_t i = 0; i < data.size() - 1; i++) {
        data[i] = data[i + 1];
    }
    data[data.size() - 1] = first;
}
```

Алгоритм AES для симметричного шифрования

Функция шифровки

```cpp
string aes_encrypt(const string& text, const string& key) {
    SimpleAES aes(key);

    vector<unsigned char> input_bytes;
    for (char c : text) {
        input_bytes.push_back(static_cast<unsigned char>(c));
    }

    vector<unsigned char> encrypted_bytes = aes.encrypt(input_bytes);

    string result;
    for (unsigned char byte : encrypted_bytes) {
        result += static_cast<char>(byte);
    }

    return result;
}
```

Функция дешифровки

```cpp
string aes_decrypt(const string& text, const string& key) {
    SimpleAES aes(key);

    vector<unsigned char> input_bytes;
    for (char c : text) {
        input_bytes.push_back(static_cast<unsigned char>(c));
    }

    vector<unsigned char> decrypted_bytes = aes.decrypt(input_bytes);

    // Удаляем нулевые байты в конце
    while (!decrypted_bytes.empty() && decrypted_bytes.back() == 0x00) {
        decrypted_bytes.pop_back();
    }

    string result;
    for (unsigned char byte : decrypted_bytes) {
        result += static_cast<char>(byte);
    }

    return result;
}
```

Тестирование алгоритма AES

```
string a(1000, '0');
string b(1000000, '0');
cout << Time(test_encryption, "Hello, World!", "mysecretkey").count() << " - время выполнения в мск";
cout << endl << endl;
cout << Time(test_encryption,a, "mysecretkey").count() << " - время выполнения в мск";
cout << endl << endl;
cout << Time(test_encryption,b, "mysecretkey").count() << " - время выполнения в мск";
```

Тест с hello world

```
Original text: Hello, World!
Encrypted text (hex): 9c 8d a6 e7 d8 85 95 65 d4 8c 8 71 71 b9 83 ef
Decrypted text: Hello, World!
Encryption/decryption successful!
66 - время выполнения в мск
```

Тесты с большими данными(a,b)

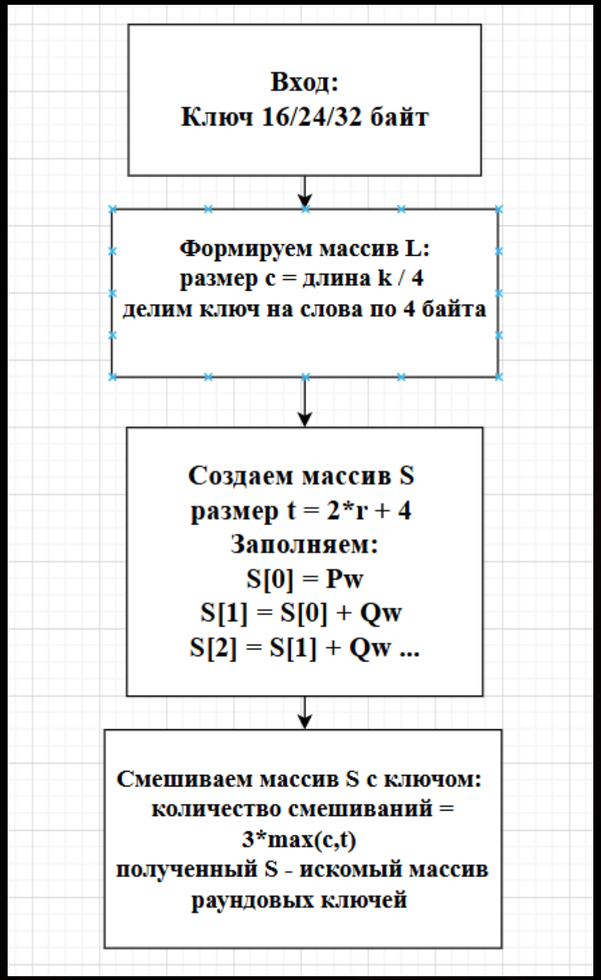
```
Original text: ...
Encrypted text (hex): ...
Decrypted text: ...
Encryption/decryption successful!
407 - время выполнения в мск
```

```
Original text: ...
Encrypted text (hex): ...
Decrypted text: ...
Encryption/decryption successful!
335833 - время выполнения в мск%
```

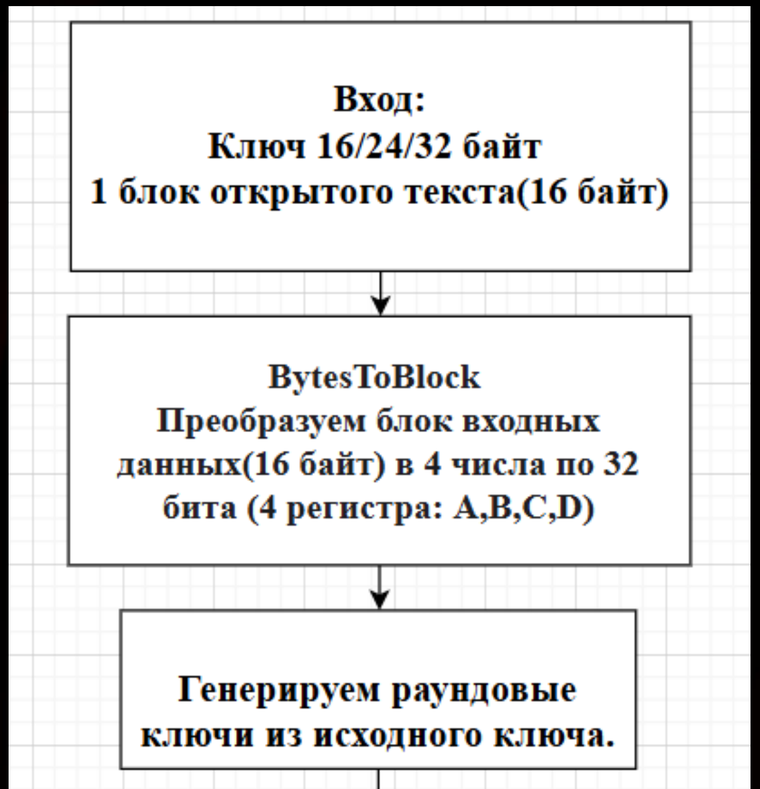
Алгоритм RC6 – симметричный блочный шифр

Входные данные разбиваются на блоки, которые обрабатываются последовательными раундами преобразований.

Рассмотрим процесс создания раундовых ключей:



Рассмотрим шифрование 1 блока:



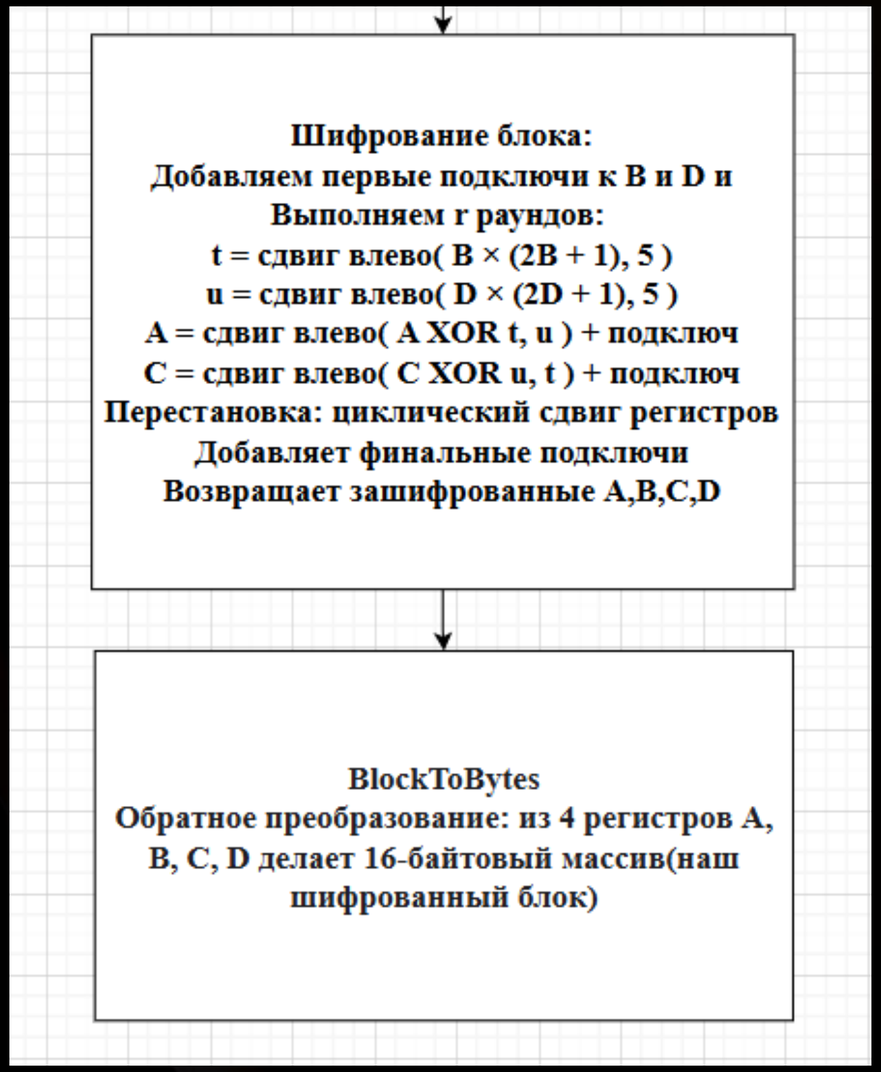
Циклический сдвиг влево и вправо:

```cpp
uint32_t RC6::ROTL(uint32_t x, uint32_t n) const {
    n %= w;
    return (x << n) | (x >> (w - n));
}

uint32_t RC6::ROTR(uint32_t x, uint32_t n) const {
    n %= w;
    return (x >> n) | (x << (w - n));
}
```

```cpp
for (size_t s = 0; s < v; ++s) {
    A = S[i] = ROTL(S[i] + A + B, 3);
    B = L[j] = ROTL(L[j] + A + B, (A + B) % w);
    i = (i + 1) % t;
    j = (j + 1) % c;
}
```

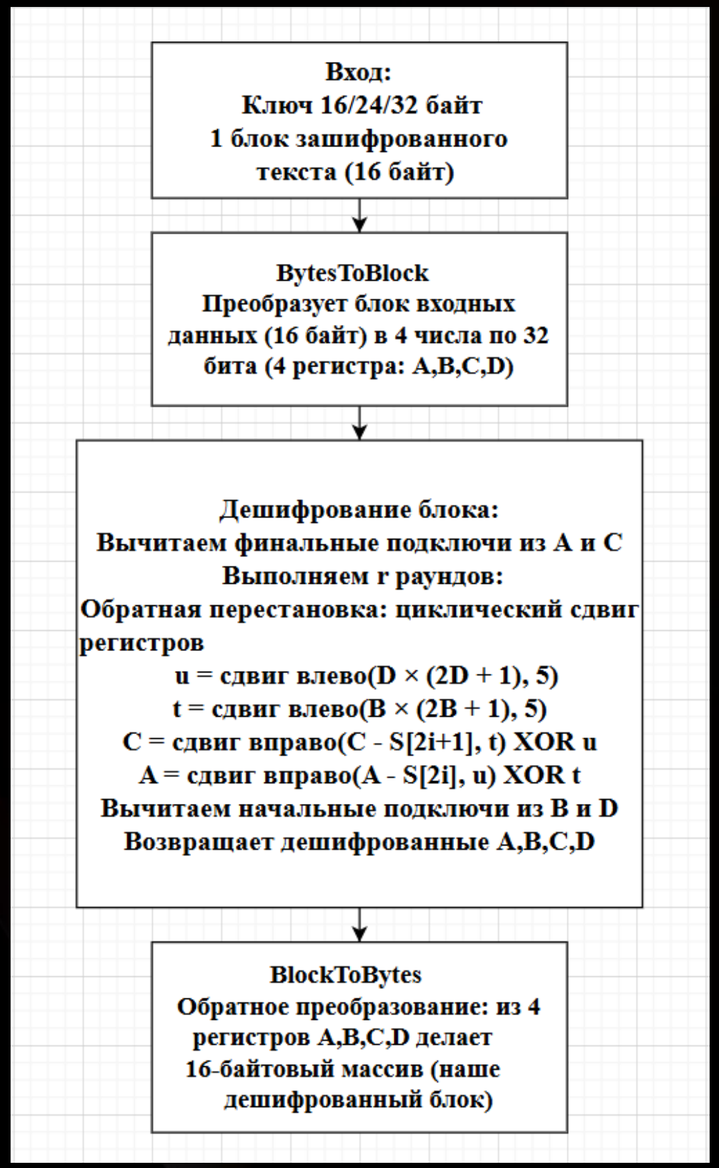
Алгоритм RC6 – симметричный блочный шифр

Шифрование блока




```cpp
vector<uint8_t> RC6::EncryptBlock(const vector<uint8_t>& plaintext) const {
    assert(plaintext.size() == 16);

    uint32_t A, B, C, D;
    uint32_t block[4];
    BytesToBlock(plaintext, block);

    A = block[0]; B = block[1]; C = block[2]; D = block[3];

    B += S[0];
    D += S[1];

    for (int i = 1; i <= r; ++i) {
        uint32_t t = ROTL(B * (2 * B + 1), lgw);
        uint32_t u = ROTL(D * (2 * D + 1), lgw);

        A = ROTL(A ^ t, u) + S[2 * i];
        C = ROTL(C ^ u, t) + S[2 * i + 1];

        uint32_t temp = A;
        A = B; B = C; C = D; D = temp;
    }

    A += S[2 * r + 2];
    C += S[2 * r + 3];

    block[0] = A; block[1] = B; block[2] = C; block[3] = D;

    vector<uint8_t> ciphertext;
    BlockToBytes(block, ciphertext);
    return ciphertext;
}
```

Алгоритм RC6 – симметричный блочный шифр



Дешифрование блока

```cpp
vector<uint8_t> RC6::DecryptBlock(const vector<uint8_t>& ciphertext) const {
    assert(ciphertext.size() == 16);

    uint32_t A, B, C, D;
    uint32_t block[4];
    BytesToBlock(ciphertext, block);

    A = block[0]; B = block[1]; C = block[2]; D = block[3];

    C -= S[2 * r + 3];
    A -= S[2 * r + 2];

    for (int i = r; i >= 1; --i) {

        uint32_t temp = D;
        D = C; C = B; B = A; A = temp;

        uint32_t u = ROTL(D * (2 * D + 1), lgw);
        uint32_t t = ROTL(B * (2 * B + 1), lgw);

        C = ROTR(C - S[2 * i + 1], t) ^ u;
        A = ROTR(A - S[2 * i], u) ^ t;
    }

    D -= S[1];
    B -= S[0];

    block[0] = A; block[1] = B; block[2] = C; block[3] = D;

    vector<uint8_t> plaintext;
    BlockToBytes(block, plaintext);
    return plaintext;
}
```

Алгоритм RC6 – симметричный блочный шифр

RC6 умеет шифровать только один блок — ровно 16 байт.
Если вы зашифруете один и тот же блок дважды одним и тем же ключом — получите один и тот же зашифрованный текст.

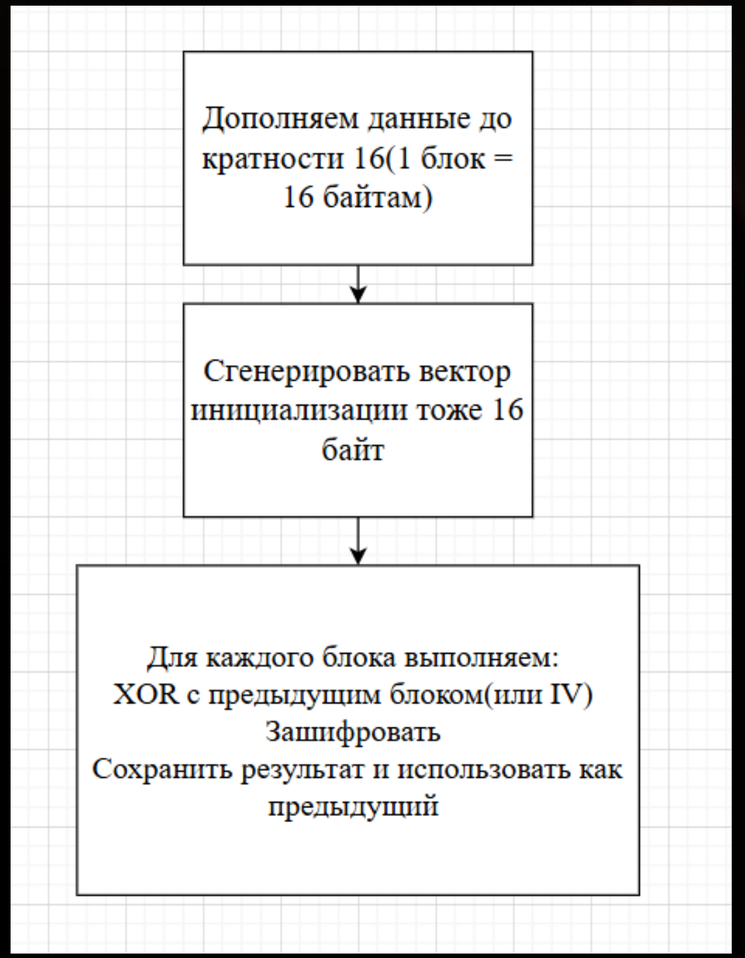
!!! Одинаковые блоки → одинаковый зашифрованный текст.
!!! Злоумышленник может подменить блоки.

Введение режима решает эти проблемы, он связывает блоки между собой, чтобы:

- Одинаковые блоки шифровались по-разному.
- Подмена блоков приводила к полной бессмыслице при расшифровке.

Режим CBC (Cipher Block Chaining) - Цепочка блоков шифрования.

Даже если два блока одинаковые → их вход в шифр разный → выход разный.



IV всегда отправляется вместе с зашифрованным сообщением (в начале).
Без него дешифрование будет невозможно.

Тестирование алгоритма RC6

```
cout << "\n=== ДЕМОНСТРАЦИЯ БЛОЧНОГО ШИФРОВАНИЯ RC6 ===" << endl;
std::string message1 = "Hello";
std::string message2 = "Hello, world! This is my block cipher implementation program! Let's check how it works,
string message3(1000000, '0');
string message4 = "";

cout << Time(block_cipher_RC6, message1).count() << " - время выполнения в мск \n" << endl;
cout << Time(block_cipher_RC6, message2).count() << " - время выполнения в мск \n"<< endl;
cout << Time(block_cipher_RC6, message3).count() << " - время выполнения в мск \n"<< endl;
cout << Time(block_cipher_RC6, message4).count() << " - время выполнения в мск \n"<< endl;
```

message1

```
Исходное сообщение: Hello
Сообщение зашифровано в режиме CBC
Расшифрованное сообщение: Hello
ШИФРОВАНИЕ RC6 РАБОТАЕТ КОРРЕКТНО
59 - время выполнения в мск
```

message3

```
Исходное сообщение: ...
Сообщение зашифровано в режиме CBC
Расшифрованное сообщение: ...
ШИФРОВАНИЕ RC6 РАБОТАЕТ КОРРЕКТНО
516853 - время выполнения в мск
```

message2

```
Исходное сообщение: ...
Сообщение зашифровано в режиме CBC
Расшифрованное сообщение: ...
ШИФРОВАНИЕ RC6 РАБОТАЕТ КОРРЕКТНО
110 - время выполнения в мск
```

message4

```
Исходное сообщение:
Сообщение зашифровано в режиме CBC
Расшифрованное сообщение:
ШИФРОВАНИЕ RC6 РАБОТАЕТ КОРРЕКТНО
44 - время выполнения в мск
```

Режим работы ECB с RC6

ECB (Electronic Codebook) — электронная кодовая книга — это самый простой и intuitive режим симметричного шифрования. Его название происходит от аналогии с кодовой книгой, где каждому исходному блоку соответствует определенный зашифрованный блок.

Основная работа:

- Разделение данных: Сообщение разбивается на блоки одинакового размера (например, 128 бит для AES).
- Независимое шифрование: Каждый блок данных шифруется отдельно с помощью одного и того же ключа.

Адаптеры для интеграции RC6 в систему ECB режима.

```
void rc6_encrypt_adapter(const vector<uint8_t>& input, vector<uint8_t>& output, const vector<uint8_t>& key) {
    RC6 cipher(key, 20);
    output = cipher.EncryptBlock(input);
}

void rc6_decrypt_adapter(const vector<uint8_t>& input, vector<uint8_t>& output, const vector<uint8_t>& key) {
    RC6 cipher(key, 20);
    output = cipher.DecryptBlock(input);
}
```

Режим работы ECB с RC6

Функция шифровки

```
vector<uint8_t> ECB::encrypt(const vector<uint8_t>& data) {
    // Добавляем padding
    int padded_len = data.size() + (block_size - (data.size() % block_size));
    if (padded_len == data.size()) padded_len += block_size;

    vector<uint8_t> padded_data = data;
    uint8_t padding_value = padded_len - data.size();
    padded_data.resize(padded_len, padding_value);

    // Шифруем каждый блок в режиме ECB
    vector<uint8_t> ciphertext;
    for (size_t i = 0; i < padded_data.size(); i += block_size) {
        vector<uint8_t> block(padded_data.begin() + i, padded_data.begin() + i + block_size);
        vector<uint8_t> encrypted_block;
        encrypt_block(block, encrypted_block, key);
        ciphertext.insert(ciphertext.end(), encrypted_block.begin(), encrypted_block.end());
    }

    return ciphertext;
}
```

Функция дешифровки

```
vector<uint8_t> ECB::decrypt(const vector<uint8_t>& data) {
    if (data.size() % block_size != 0) {
        throw runtime_error("Invalid ciphertext length for ECB");
    }

    // Дешифруем каждый блок
    vector<uint8_t> decrypted_data;
    for (size_t i = 0; i < data.size(); i += block_size) {
        vector<uint8_t> block(data.begin() + i, data.begin() + i + block_size);
        vector<uint8_t> decrypted_block;
        decrypt_block(block, decrypted_block, key);
        decrypted_data.insert(decrypted_data.end(), decrypted_block.begin(), decrypted_block.end());
    }

    // Убираем padding
    if (decrypted_data.empty()) return decrypted_data;

    uint8_t padding_value = decrypted_data.back();
    if (padding_value > block_size || padding_value == 0) {
        throw runtime_error("Invalid padding in ECB decryption");
    }

    // Проверяем padding
    for (size_t i = decrypted_data.size() - padding_value; i < decrypted_data.size(); ++i) {
        if (decrypted_data[i] != padding_value) {
            throw runtime_error("Invalid padding bytes in ECB decryption");
        }
    }

    decrypted_data.resize(decrypted_data.size() - padding_value);
    return decrypted_data;
}
```

Тестирование режима работы ECB

```
string message_123(1000000, '0');

cout << Time(test_ecb_rc6,"Hello, World!", key16).count() << " – время выполнения в мкс \n"<< endl;
cout << Time(test_ecb_rc6, "Short", key32).count() << " – время выполнения в мкс \n" << endl;
cout << Time(test_ecb_rc6, message_123, key16).count() << " – время выполнения в мкс \n"<< endl;
```

```
Исходная строка: 'Hello, World!'
Зашифровано (16 байт): A4 15 95 AB FF 53 EC 02 FC 38 B8 D5 8F 36 44 26
Расшифровано: 'Hello, World!'
✅ Строки совпадают!

191 – время выполнения в мкс
```

```
Исходная строка: 'Short'
Зашифровано (16 байт): 30 D3 81 08 73 99 83 EE 1D A9 17 3F 24 E7 A5 3C
Расшифровано: 'Short'
✅ Строки совпадают!

160 – время выполнения в мкс
```

Тест на млн символов

```
Исходная строка: '0000000000000000000000000000000000000000...' (длина: 1000000)
Зашифровано (1000016 байт): 2A 89 D5 6F 11 BD 28 82 ... CC 81 92 79 26 07 F6 37 (первые и последние 8 байт)
Расшифровано: '0000000000000000000000000000000000000000...' (длина: 1000000)
✅ Строки совпадают!

1165247 – время выполнения в мкс
```

Алгоритм ARC4 - симметричный поточный шифр

В основе поточного шифрования лежит генератор псевдослучайных чисел, который создает длинную последовательность битов, называемую потоком ключей (keystream). Этот поток ключей генерируется на основе секретного ключа и, иногда, вектора инициализации.

Процесс шифрования выглядит следующим образом:

- Преобразование ключа в байты.
- Создание S-блока, перемешивание его ключом.
- Генерация потока ключей: ГПСЧ создает поток псевдослучайных байтов.
- Побитовое XOR: Каждый байт открытого текста объединяется с соответствующим байтом из потока ключей с помощью операции "исключающее ИЛИ" (XOR).
- Получение шифротекста: Результатом операции XOR является шифротекст.

Преобразование ключа в байты

```
SimpleStreamCipher::SimpleStreamCipher(const string& key) : i(0), j(0) {
    vector<uint8_t> key_data(key.begin(), key.end());
    initialize(key_data);
}
```

Подготовка S-блока для генерации ключевого потока

```
void SimpleStreamCipher::initialize(const vector<uint8_t>& key) {
    S.resize(256);
    for (int k = 0; k < 256; k++) {
        S[k] = k;
    }

    int j = 0;
    for (int k = 0; k < 256; k++) {
        j = (j + S[k] + key[k % key.size()]) % 256;
        swap(S[k], S[j]);
    }
}
```

Алгоритм ARC4 - симметричный поточный шифр

Дешифрование:

- Восстановление потока ключей: При использовании того же секретного ключа и начального состояния генерируется идентичный ключевой поток.
- Обратное XOR: Шифротекст объединяется с ключевым потоком через ту же операцию XOR.
- Восстановление текста: Результат является исходным открытым текстом благодаря свойству двойного XOR: $(P \oplus K) \oplus K = P$

Генерация ключевого потока

```
uint8_t SimpleStreamCipher::generateKeyByte() {
    i = (i + 1) % 256;
    j = (j + S[i]) % 256;
    swap(S[i], S[j]);
    return S[(S[i] + S[j]) % 256];
}
```

Основная функция шифрования/дешифрования

```
vector<char> SimpleStreamCipher::process(const vector<char>& input) {
    i = 0;
    j = 0;
    vector<char> output;
    output.reserve(input.size());

    for (char input_byte : input) {
        uint8_t key_byte = generateKeyByte();
        output.push_back(static_cast<char>(input_byte ^ key_byte));
    }
    return output;
}
```

Тестирование алгоритма ARC4

```
string message9 = "Secret Message";
string message_1111 = "hello world";
string message_2222(1000000, '0');

cout << Time(demonstrateStreamCipher, message9).count() << " - время выполнения в мкс"<< endl;
cout << Time(demonstrateStreamCipher, message_1111).count() << " - время выполнения в мкс"<< endl;
cout << Time(demonstrateStreamCipher, message_2222).count() << " - время выполнения в мкс"<< endl;
```

```
Исходное сообщение: hello world
Секретный ключ: MySecretKey123

Зашифрованные байты (11 байт): 236 6 78 62 31 195 105 232 126 191 186
Расшифрованное сообщение: hello world
Расшифрованные байты как числа (11 байт): 104 101 108 108 111 32 119 111 114 108 100

Результат: Успех - сообщение восстановлено!

86 - время выполнения в мск
```

```
Исходное сообщение: Secret Message
Секретный ключ: MySecretKey123

Зашифрованные байты (14 байт): 215 6 65 32 21 151 62 202 105 160 173 139 150 154
Расшифрованное сообщение: Secret Message
Расшифрованные байты как числа (14 байт): 83 101 99 114 101 116 32 77 101 115 115 97 103 101

Результат: Успех - сообщение восстановлено!

110 - время выполнения в мск
```

Тестирование на млн элементов

```
Исходное сообщение: 00000000000000000000000000000000000000000... (длина: 1000000 символов)
Секретный ключ: MySecretKey123

Зашифрованные байты (1000000 байт): 180 83 18 98 64 211 46 183 ... 140 171 117 14 6 16 34 247 (первые и последние 8 байт)
Расшифрованное сообщение: 00000000000000000000000000000000000000000... (длина: 1000000 символов)
Расшифрованные байты как числа (1000000 байт): 48 48 48 48 48 48 48 48 ... 48 48 48 48 48 48 48 48 (первые и последние 8 байт)

Результат: Успех - сообщение восстановлено!

230359 - время выполнения в мск
```

АСИММЕТРИЧНОЕ ШИФРОВАНЕ

1. Два ключа
- Публичный (открытый) ключ
- Приватный (закрытый) ключ

2. Процесс:
- Шифрование: текст + публичный ключ → шифротекст
- Дешифрование: шифротекст + приватный ключ → текст

3. Особенности:
- Что зашифровано публичным ключом → расшифровывается приватным
- И наоборот (для подписей)
- Ключи математически связаны, но один нельзя получить из другого

Алгоритм RSA для асимметричного шифрования

Принцип работы RSA:

1. Генерируем ключи:

- p,q - большие простые числа
- n = p×q
- e - публичное число (обычно 65537)
- d = 1/e mod (p-1)(q-1)

2. Публичный ключ = (e,n) - даём всем

3. Приватный ключ = (d,n) - храним в секрете

4. Шифрование:

   Зашифровать = сообщение^e mod n

5. Дешифрование:

   Расшифровать = шифротекст^d mod n

6. Магия:

   (сообщение^e)^d mod n = сообщение

7. Безопасность:

   Нельзя найти d без p и q, а разложить n на множители - очень сложно.

Шифрока и дешифровка чисел

```cpp
long long RSA::encrypt(long long message, const pair<long long, long long>& publicKey) {
    long long e = publicKey.first;
    long long n = publicKey.second;
    return modPow(message, e, n);
}

long long RSA::decrypt(long long encrypted, const pair<long long, long long>& privateKey) {
    long long d = privateKey.first;
    long long n = privateKey.second;
    return modPow(encrypted, d, n);
}
```

Алгоритм RSA для асимметричного шифрования

Шифровка строки

```cpp
string RSA::encrypt(const string& message, const pair<long long, long long>& publicKey) {
    string encrypted;

    for (char c : message) {
        long long encryptedChar = encrypt(static_cast<long long>(c), publicKey);
        encrypted += to_string(encryptedChar) + " ";
    }

    return encrypted;
}
```

Дешифровка строки

```cpp
string RSA::decrypt(const string& encrypted, const pair<long long, long long>& privateKey, const string& original) {
    string decrypted;
    string encryptedCharStr;

    for (char c : encrypted) {
        if (c == ' ') {
            if (!encryptedCharStr.empty()) {
                long long encryptedChar = stoll(encryptedCharStr);
                long long decryptedChar = decrypt(encryptedChar, privateKey);
                decrypted += static_cast<char>(decryptedChar);
                encryptedCharStr.clear();
            }
        }
        else {
            encryptedCharStr += c;
        }
    }

    if (!original.empty()) {
        bool matches = (decrypted == original);
        cout << "Convergence check: " << (matches ? "SUCCESS" : "ERROR") << endl;
        if (!matches) {
            cout << "Expected: " << original << endl;
            cout << "Received: " << decrypted << endl;
        }
    }

    return decrypted;
}
```

ТЕСТИРОВАНИЕ АЛГОРИТМА RSA

```cpp
string text(1000000, '0');
pair<long long, long long> publicKey = make_pair(0, 0);
pair<long long, long long> privateKey = make_pair(0, 0);

cout << Time(processRSA, "HELLO WORLD", publicKey, privateKey).count() << " - время выполнения в мкс"<< endl;
cout << Time(processRSA,"SECRET MESSAGE32323232", publicKey, privateKey).count() << " - время выполнения в мкс"<< endl;
cout << Time(processRSA, text, publicKey, privateKey).count() << " - время выполнения в мкс"<< endl;
```

```
=== RSA PROCESSING ===
Input: "HELLO WORLD"
Public key: (65537, 10403)
Encrypted: 7347 3170 626 ... [11 numbers total]
Decrypted: "HELLO WORLD"
Verification: SUCCESS
=== COMPLETE ===

54 - время выполнения в мкс
```

```
=== RSA PROCESSING ===
Input: "SECRET MESSAGE32323232"
Public key: (65537, 11663)
Encrypted: 8814 5487 8869 ... [22 numbers total]
Decrypted: "SECRET MESSAGE323232..."
Decrypted length: 22 characters
Verification: SUCCESS
=== COMPLETE ===

60 - время выполнения в мкс
```

```
=== RSA PROCESSING ===
Input: "00000000000000000000000000000000..."
Input length: 1000000 characters
Public key: (65537, 14351)
Encrypted: 8975 8975 8975 ... [1000000 numbers total]
Decrypted: "00000000000000000000..."
Decrypted length: 1000000 characters
Verification: SUCCESS
=== COMPLETE ===

857410 - время выполнения в мкс
```

Схема Эль-Гамаля для асимметричного шифрования

Принцип работы:
1. Два ключа:
Открытый: (p, g, y) - можно всем
Закрытый: x - только получателю

2. Временный секрет:
Каждое сообщение шифруется со своим случайным k
k ≠ каждый раз, иначе нарушается безопасность

3. Общий секрет Диффи-Хеллмана:
Отправитель: y^k = (g^x)^k mod p
Получатель: a^x = (g^k)^x mod p
Результат одинаков: g^(xk) mod p

4. Маскирование сообщения:
Сообщение × на общий секрет = шифрование
Шифротекст ÷ на общий секрет = дешифрование

5. Математическая основа:
Сложность задачи дискретного логарифма
Нельзя вычислить x из y = g^x mod p
Нельзя вычислить k из a = g^k mod p

Генерация ключей

```
ElGamalUniversal::ElGamalUniversal(string size) : gen(rd()) {
    if (size == "small") {
        p = generate_prime(1000, 10000);
    }
    else if (size == "huge") {
        p = generate_prime(1000000000, 5000000000);
    }
    else {
        p = generate_prime(10000, 100000);
    }

    if (p == 0) {
        cout << "Ошибка: не удалось сгенерировать простое число" << endl;
        return;
    }

    g = find_primitive_root(p);
    uniform_int_distribution<long long> dis(2, p - 2);
    x = dis(gen);
    y = mod_pow(g, x, p);
}
```

Схема Эль-Гамаля для асимметричного шифрования

Функция шифрования

```
pair<long long, long long> ElGamalUniversal::encrypt_number(long long number) {
    if (number >= p) {
        cout << "Ошибка: число " << number << " слишком большое. Максимум: " << (p - 1) << endl;
        return make_pair(0, 0);
    }

    long long k;
    uniform_int_distribution<long long> dis(2, p - 2);
    do {
        k = dis(gen);
    } while (gcd(k, p - 1) != 1);

    long long a = mod_pow(g, k, p);
    long long b = (number * mod_pow(y, k, p)) % p;

    return make_pair(a, b);
}
```

Функция дешифрования

```
long long ElGamalUniversal::decrypt_number(pair<long long, long long> ciphertext) {
    long long a = ciphertext.first;
    long long b = ciphertext.second;

    if (a == 0 && b == 0) {
        cout << "Ошибка: некорректный шифротекст" << endl;
        return -1;
    }

    long long s = mod_pow(a, x, p);
    long long s_inv = mod_pow(s, p - 2, p);
    long long message = (b * s_inv) % p;

    return message;
}

void ElGamalUniversal::print_info() {
    cout << "Параметры системы ElGamal:" << endl;
    cout << "p: " << p << " (максимальное число: " << (p - 1) << ")" << endl;
    cout << "g: " << g << endl;
    cout << "x: " << x << " (закрытый ключ)" << endl;
    cout << "y: " << y << " (открытый ключ)" << endl;
    cout << "----------------------------" << endl;
}

bool ElGamalUniversal::is_valid() {
    return p != 0;
}
```

Тестирование Схемы Эль-Гамаля

```
vector<int> small_numbers = { 1, 42, 100, 255, 512, 999, 1024, 2048 };
vector<long long> huge_numbers = { 1000000000, 1500000000 };

cout << Time(test_small_numbers, small_numbers).count() << " - время выполнения в мск" << endl;
cout << Time(test_huge_numbers, huge_numbers).count() << " - время выполнения в мск" << endl;
```

```
=== ТЕСТ МАЛЕНЬКИХ ЧИСЕЛ ===
Параметры системы ElGamal:
p: 6421 (максимальное число: 6420)
g: 6
x: 4059 (закрытый ключ)
y: 3176 (открытый ключ)
--------------------------
1(Исходное) = 1(расшифрованное)
1 → (922, 2434)

42(Исходное) = 42(расшифрованное)
42 → (1779, 2197)

100(Исходное) = 100(расшифрованное)
100 → (4155, 6350)

255(Исходное) = 255(расшифрованное)
255 → (4342, 1324)

512(Исходное) = 512(расшифрованное)
512 → (1275, 1117)

999(Исходное) = 999(расшифрованное)
999 → (545, 236)

1024(Исходное) = 1024(расшифрованное)
1024 → (4206, 1439)

2048(Исходное) = 2048(расшифрованное)
2048 → (3424, 244)


173 - время выполнения в мск
```

```
173 - время выполнения в мск
=== ТЕСТ БОЛЬШИХ ЧИСЕЛ ===
Параметры системы ElGamal:
p: 2171481131 (максимальное число: 2171481130)
g: 2
x: 369290623 (закрытый ключ)
y: 546490307 (открытый ключ)
----------------------------
1000000000(Исходное) = 1000000000(расшифрованное)
1000000000 → (1574097399, 335499922)

1500000000(Исходное) = 1500000000(расшифрованное)
1500000000 → (1135092323, 1832364176)

569 - время выполнения в мск
```

Алгоритм ECC для асимметричного шифрования

1. Создание ключей
Каждый пользователь выбирает секретное число (приватный ключ)
Вычисляет публичный ключ = секрет × базовая_точка

2. Обмен публичными ключами
Пользователи обмениваются публичными ключами
Сохраняют свои приватные ключи в секрете

3. Вычисление общего секрета
Пользователь 1: его_секрет × публичный_ключ_пользователя2
Пользователь 2: его_секрет × публичный_ключ_пользователя1
Получается одинаковое число (общий секрет)

4. Использование секрета
Общий секрет используется как ключ для XOR шифрования
XOR шифрует сообщение: каждый_символ ^ секрет

Создание ключевой пары

```
SimpleECC::SimpleECC(int secret) {
    private_key = secret;
    // Вычисляем открытый ключ: Public = Private * BasePoint
    public_x = (private_key * base_x) % prime;
    public_y = (private_key * base_y) % prime;
}
```

Шифрование и дешифрование простым XOR

```
std::string xorEncrypt(std::string text, int key) {
    std::string result = "";
    for (char c : text) {
        // XOR каждого символа с ключом
        result += c ^ (key % 128);  // %128 чтобы остаться в пределах ASCII
    }
    return result;
}
```

Алгоритм ECC для асимметричного шифрования

Основная работа алгоритма

```
void demonstrateEncryption(const std::string& message) {
    cout << "=== Демонстрация шифрования ECC ===" << endl << endl;

    // Создаём двух пользователей
    SimpleECC user1(7);
    SimpleECC user2(13);

    // Выводим ключи
    cout << "Пользователь 1:" << endl;
    cout << "Приватный: " << user1.getPrivateKey() << endl;
    cout << "Публичный: (" << user1.getPublicX() << ", " << user1.getPublicY() << ")" << endl << endl;

    cout << "Пользователь 2:" << endl;
    cout << "Приватный: " << user2.getPrivateKey() << endl;
    cout << "Публичный: (" << user2.getPublicX() << ", " << user2.getPublicY() << ")" << endl << endl;

    // Вычисляем общий секрет
    int secret1 = user1.getSharedSecret(user2.getPublicX(), user2.getPublicY());
    int secret2 = user2.getSharedSecret(user1.getPublicX(), user1.getPublicY());

    cout << "Общий секрет пользователя 1: " << secret1 << endl;
    cout << "Общий секрет пользователя 2: " << secret2 << endl;
    cout << "Секреты совпадают: " << (secret1 == secret2 ? "да" : "нет") << endl << endl;

    // Шифрование и дешифрование
    cout << "Исходное сообщение: " << message << endl;

    string encrypted = xorEncrypt(message, secret1);
    cout << "Зашифрованное сообщение: ";
    for (char c : encrypted) cout << (int)c << " ";
    cout << endl;

    string decrypted = xorEncrypt(encrypted, secret2);
    cout << "Расшифрованное сообщение: " << decrypted << endl;

    // Проверка корректности
    cout << endl << "Результат: " << (message == decrypted ? "Успех - сообщение восстановлено!" : "Ошибка - сообщение повреждено!") << endl;
}
```

ТЕСТИРОВАНИЕ АЛГОРИТМА ECC

```
string message_0 = "Secret Message";
string message_1(1000, '0');
string message_2(1000000, '0');

cout << Time(demonstrateEncryption, message_0).count() << " – время выполнения в мкс"<< endl;
cout << Time(demonstrateEncryption, message_1).count() << " – время выполнения в мкс"<< endl;
cout << Time(demonstrateEncryption, message_2).count() << " – время выполнения в мкс"<< endl;
```

```
Исходное сообщение: Secret Message
Зашифрованное сообщение: 65 119 113 96 119 102 50 95 119 97 ...
Расшифрованное сообщение: Secret Message

Сравнение: УСПЕХ – сообщения идентичны!
39 – время выполнения в мкс
```

```
Исходное сообщение: 00000000000000000000000000000...
Зашифрованное сообщение: 34 34 34 34 34 34 34 34 34 34 ...
Расшифрованное сообщение: 00000000000000000000000000000...

Сравнение: УСПЕХ – сообщения идентичны!
200 – время выполнения в мкс
```

Тест на млн символов

```
Исходное сообщение: 00000000000000000000000000000...
Зашифрованное сообщение: 34 34 34 34 34 34 34 34 34 34 ...
Расшифрованное сообщение: 00000000000000000000000000000...

Сравнение: УСПЕХ – сообщения идентичны!
48937 – время выполнения в мкс
```

Сравнительная таблица времени выполнения алгоритмов

Алгоритм	Мал. (13–14)	Ср. (1000)	Бол. (1M)
AES	112	510	347237
RC6 (CBC)	85	564	512026
ECB (RC6)	111	36	1148407
ARC4	111	281	219254
RSA	204	1102	860423
ElGamal	100	–	924
ECC	21	67	47245

СПАСИБО ЗА ВНИМАНИЕ

QR код репозитория