Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Криптографические методы защиты информации»

Тема: «Алгоритмы обмена ключами»

**Выполнил:**

Студент 3 курса

Группы ИИ-23

Романюк А. П.

**Проверил:**

Хацкевич А. С.

Брест 2024

**Цель:** изучить алгоритмы обмена ключами. Практически реализовать алгоритмы обмена ключами.

**Ход работы:**

**Вариант**:Усиленный EKE , RC-2

#include <openssl/evp.h>

#include <openssl/rand.h>

#include <openssl/provider.h>

#include <openssl/err.h>

#include <iostream>

#include <vector>

#include <cstring>

#include <stdexcept>

const int KEY\_SIZE = 32; // Размер ключа для AES-256

const int BLOCK\_SIZE = 16; // Размер блока для AES

std::pair<std::vector<unsigned char>, std::vector<unsigned char>> split\_vector(const std::vector<unsigned char>& original) {

size\_t mid = original.size() / 2;

std::vector<unsigned char> first\_half(original.begin(), original.begin() + mid);

std::vector<unsigned char> second\_half(original.begin() + mid, original.end());

return { first\_half, second\_half };

}

std::vector<unsigned char> getSubVector(const std::vector<unsigned char>& vec, size\_t start, size\_t end) {

// Проверяем корректность индексов

if (start > end || end > vec.size()) {

throw std::out\_of\_range("Некорректные индексы");

}

// Используем итераторы для создания подвектора

return std::vector<unsigned char>(vec.begin() + start, vec.begin() + end);

}

std::vector<unsigned char> xorVectors(const std::vector<unsigned char>& vec1, const std::vector<unsigned char>& vec2) {

// Проверяем, что размеры векторов одинаковы

if (vec1.size() != vec2.size()) {

throw std::invalid\_argument("Векторы должны быть одинакового размера");

}

// Результирующий вектор для хранения результата XOR

std::vector<unsigned char> result(vec1.size());

// Выполняем побитовый XOR для каждого элемента

for (size\_t i = 0; i < vec1.size(); ++i) {

result[i] = vec1[i] ^ vec2[i];

}

return result;

}

std::vector<unsigned char> generate\_random\_number(int size) {

std::vector<unsigned char> num(size);

if (!RAND\_bytes(num.data(), size)) {

throw std::runtime\_error("Не удалось сгенерировать случайное число");

}

return num;

}

std::vector<unsigned char> generate\_random\_key(size\_t key\_size) {

std::vector<unsigned char> key(key\_size);

if (RAND\_bytes(key.data(), key\_size) != 1) {

// Обработка ошибки

throw std::runtime\_error("Error generating random key");

}

return key;

}

std::vector<unsigned char> aes\_encrypt(const std::vector<unsigned char>& key, const std::vector<unsigned char>& plaintext) {

EVP\_CIPHER\_CTX\* ctx = EVP\_CIPHER\_CTX\_new();

std::vector<unsigned char> ciphertext(plaintext.size() + BLOCK\_SIZE);

std::vector<unsigned char> iv(BLOCK\_SIZE);

RAND\_bytes(iv.data(), BLOCK\_SIZE);

int len;

int ciphertext\_len;

if (!EVP\_EncryptInit\_ex(ctx, EVP\_aes\_256\_cbc(), NULL, key.data(), iv.data())) {

throw std::runtime\_error("Ошибка инициализации шифрования");

}

if (!EVP\_EncryptUpdate(ctx, ciphertext.data(), &len, plaintext.data(), plaintext.size())) {

throw std::runtime\_error("Ошибка шифрования");

}

ciphertext\_len = len;

if (!EVP\_EncryptFinal\_ex(ctx, ciphertext.data() + len, &len)) {

throw std::runtime\_error("Ошибка завершения шифрования");

}

ciphertext\_len += len;

EVP\_CIPHER\_CTX\_free(ctx);

ciphertext.resize(ciphertext\_len);

ciphertext.insert(ciphertext.end(), iv.begin(), iv.end());

return ciphertext;

}

std::vector<unsigned char> aes\_decrypt(const std::vector<unsigned char>& key, const std::vector<unsigned char>& ciphertext) {

EVP\_CIPHER\_CTX\* ctx = EVP\_CIPHER\_CTX\_new();

std::vector<unsigned char> plaintext(ciphertext.size());

std::vector<unsigned char> iv(BLOCK\_SIZE);

std::memcpy(iv.data(), ciphertext.data() + ciphertext.size() - BLOCK\_SIZE, BLOCK\_SIZE);

int len;

int plaintext\_len;

if (!EVP\_DecryptInit\_ex(ctx, EVP\_aes\_256\_cbc(), NULL, key.data(), iv.data())) {

throw std::runtime\_error("Ошибка инициализации дешифрования");

}

if (!EVP\_DecryptUpdate(ctx, plaintext.data(), &len, ciphertext.data(), ciphertext.size() - BLOCK\_SIZE)) {

throw std::runtime\_error("Ошибка дешифрования");

}

plaintext\_len = len;

if (!EVP\_DecryptFinal\_ex(ctx, plaintext.data() + len, &len)) {

throw std::runtime\_error("Ошибка завершения дешифрования");

}

plaintext\_len += len;

EVP\_CIPHER\_CTX\_free(ctx);

plaintext.resize(plaintext\_len);

return plaintext;

}

std::vector<unsigned char> EKE\_demo(std::vector<unsigned char> password,std::vector<unsigned char> key\_) {

// Алиса шифрует ключ и отправляет

auto ekey = aes\_encrypt(password, key\_);

std::cout << "encrypted key\_:\n";

for (auto& byte : ekey) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Боб расшифровывает сообщение

std::vector<unsigned char> bobsKey\_ = aes\_decrypt(password, ekey);

std::cout << "decrypted key\_:\n";

for (auto& byte : bobsKey\_) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Боб генерирует сеансовый ключ

std::vector<unsigned char> bobsKey = generate\_random\_key(32);

std::cout << "key:\n";

for (auto& byte : bobsKey) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Боб шифрует открытым ключем сеансовый, а затем шифрует паролем и отправляет

std::vector<unsigned char> ebobsKey = aes\_encrypt(password, aes\_encrypt(bobsKey\_, bobsKey));

std::cout << "encrypted key:\n";

for (auto& byte : ebobsKey) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

/\*--------------------------------------------------------------------------\*/

// Алиса расшифровывает сообщение и получает сеансовый ключ

std::vector<unsigned char> alicesKey = aes\_decrypt(key\_, aes\_decrypt(password, ebobsKey));

std::cout << "decrypted key:\n";

for (auto& byte : alicesKey) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Алиса генерирует случайную строку и случайное чило

std::vector<unsigned char> Ra = generate\_random\_key(32);

std::vector<unsigned char> Sa = generate\_random\_number(32);

auto RaSa = Ra;

RaSa.insert(RaSa.end(), Sa.begin(), Sa.end());

std::cout << "Ra + Sa:\n";

for (auto& byte : RaSa) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Алиса шифрует строку и отправляет

std::vector<unsigned char> eRa = aes\_encrypt(alicesKey, RaSa);

std::cout << "encrypted Ra + Sa:\n";

for (auto& byte : eRa) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Боб расшифровывает сообщение и получает строку и число алисы

std::vector<unsigned char> BobsRa = split\_vector(aes\_decrypt(bobsKey, eRa)).first;

std::vector<unsigned char> BobsSa = split\_vector(aes\_decrypt(bobsKey, eRa)).second;

std::cout << "decrypted Ra + Sa:\n";

for (auto& byte : BobsRa) std::cout << std::hex << (int)byte;

std::cout << std::endl;

for (auto& byte : BobsSa) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Боб генерирует свою случайную строку и число, шифрует и отправляет

std::vector<unsigned char> Rb = generate\_random\_key(32);

std::vector<unsigned char> Sb = generate\_random\_key(32);

std::cout << "Rb:\n";

for (auto& byte : Rb) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

auto comb = BobsRa;

comb.insert(comb.end(), Rb.begin(), Rb.end());

comb.insert(comb.end(), Sb.begin(), Sb.end());

auto eBobsRaRb = aes\_encrypt(bobsKey, comb);

std::cout << "encrypted Ra + Rb:\n";

for (auto& byte : eBobsRaRb) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

// Алиса получает строки и расшифровывает их

auto aliceRaRb = aes\_decrypt(alicesKey, eBobsRaRb);

std::cout << "decrypted Ra + Rb:\n";

for (auto& byte : aliceRaRb) std::cout << std::hex << (int)byte;

std::cout << std::endl << std::endl;

std::vector<unsigned char> deRa = getSubVector(aliceRaRb,0,32);

std::vector<unsigned char> deRb = getSubVector(aliceRaRb, 32, 64);

std::vector<unsigned char> deSb = getSubVector(aliceRaRb, 64, 96);

if (!(deRa == Ra)) return{};

// Алиса шифрует Rb и отправляет

std::vector<unsigned char> eRb = aes\_encrypt(alicesKey, deRb);

// Боб расшифровывает и сравнивает значения и, если все ок то ок

if (aes\_decrypt(bobsKey, eRb) == Rb) {

std::cout << "You get private key!!!\n\n\n";

return xorVectors(Sa,Sb);

}

else return{};

}

void handleErrors() {

ERR\_print\_errors\_fp(stderr);

abort();

}

std::vector<unsigned char> encrypt(const std::vector<unsigned char>& plaintext, const std::vector<unsigned char>& key) {

EVP\_CIPHER\_CTX\* ctx = EVP\_CIPHER\_CTX\_new();

if (!ctx) handleErrors();

std::vector<unsigned char> ciphertext(plaintext.size() + EVP\_CIPHER\_block\_size(EVP\_rc2\_ecb()));

int len;

int ciphertext\_len;

if (1 != EVP\_EncryptInit\_ex(ctx, EVP\_rc2\_ecb(), nullptr, key.data(), nullptr)) {

handleErrors();

}

if (1 != EVP\_EncryptUpdate(ctx, ciphertext.data(), &len, plaintext.data(), plaintext.size())) {

handleErrors();

}

ciphertext\_len = len;

if (1 != EVP\_EncryptFinal\_ex(ctx, ciphertext.data() + len, &len)) {

handleErrors();

}

ciphertext\_len += len;

ciphertext.resize(ciphertext\_len);

EVP\_CIPHER\_CTX\_free(ctx);

return ciphertext;

}

std::vector<unsigned char> decrypt(const std::vector<unsigned char>& ciphertext, const std::vector<unsigned char>& key) {

EVP\_CIPHER\_CTX\* ctx = EVP\_CIPHER\_CTX\_new();

if (!ctx) handleErrors();

std::vector<unsigned char> plaintext(ciphertext.size());

int len;

int plaintext\_len;

if (1 != EVP\_DecryptInit\_ex(ctx, EVP\_rc2\_ecb(), nullptr, key.data(), nullptr)) {

handleErrors();

}

if (1 != EVP\_DecryptUpdate(ctx, plaintext.data(), &len, ciphertext.data(), ciphertext.size())) {

handleErrors();

}

plaintext\_len = len;

if (1 != EVP\_DecryptFinal\_ex(ctx, plaintext.data() + len, &len)) {

handleErrors();

}

plaintext\_len += len;

plaintext.resize(plaintext\_len);

EVP\_CIPHER\_CTX\_free(ctx);

return plaintext;

}

int main() {

OSSL\_PROVIDER\* default\_provider = OSSL\_PROVIDER\_load(NULL, "default");

OSSL\_PROVIDER\* legacy = OSSL\_PROVIDER\_load(NULL, "legacy");

if (!default\_provider || !legacy) {

std::cerr << "Не удалось загрузить провайдеры" << std::endl;

handleErrors();

}

std::vector<unsigned char> password = { 'p', 'a', 's', 's', 'w', 'o', 'r', 'd', '1', '2', '3', '!'};

bool work = 0;

std::vector<unsigned char> private\_key;

while (!work) {

try {

std::vector<unsigned char> open\_key = generate\_random\_key(32);

private\_key = EKE\_demo(password, open\_key); // Используйте глобальную переменную private\_key

work = true; // Если всё успешно, выходим из цикла

}

catch (const std::exception& e) {

std::cerr << "Произошла ошибка: " << e.what() << std::endl;

system("cls"); // Очистка консоли в Windows (для Linux используйте "clear")

}

}

for (auto& byte : private\_key) std::cout << std::hex << (int)byte;

std::cout << std::endl;

std::vector<unsigned char> text = { 'h','e','l','l','o',' ','w','o','r','l','d' };

// Шифрование

std::vector<unsigned char> ciphertext = encrypt(text , private\_key);

for (auto& byte : ciphertext) std::cout << std::hex << (int)byte;

std::cout << std::endl;

// Расшифрование

std::vector<unsigned char> decryptedtext = decrypt(ciphertext, private\_key);

for (auto& byte : decryptedtext) std::cout << byte;

}

**Вывод:** в ходе лабораторной работы я изучил алгоритмы обмена ключами, а также реализовал клиент-серверное приложение, позволяющее обмениваться зашифрованными сообщениями.