



Міністерство освіти і науки України

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

**КОМП'ЮТЕРНИЙ ПРАКТИКУМИ З ДИСЦИПЛІНИ
«МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ
МЕХАНІЗМІВ» №4**

Дослідження особливостей реалізації існуючих програмних систем, які використовують криптографічні механізми захисту інформації

Виконала:

студентка групи ФІ-12мн

Звичайна Анастасія Олександрівна

Перевірила:

Селюх Поліна Валентинівна

Київ 2021

Мета роботи: Отримати практичні навички побудови гібридних криптосистем.

Умова задачі: Розробити реалізацію асиметричної криптосистеми Ель-Гамалю, використовуючи бібліотеку OpenSSL (<https://www.openssl.org/docs/man1.1.1/>) та вимоги Crypto API або PKCS, навести приклади роботи.

Приклад виконання програми:

```
Консоль отладки Microsoft Visual Studio
Операции на сервере:
Генерация ключей Эль-Гамалю ...
Приватный ключ Эль-Гамалю: 258B2F2A46A0E1EB353265929641CB1F21B9AF5A94B834999CDA9430DB1F9A2B
Публичный ключ Эль-Гамалю: 3B06C2AB9527B33ECAAA086BD57D6B91E7D908104FA45C7043230AC9A6611388
Генерация сертификата ...
Отправка сертификата клиенту...

Операции на клиенте:
Генерация симметричного ключа ...
Симметричный ключ: EEA232AC66ADBF4413F8E00B44616503
Шифрование симметричного ключа ...
Зашифрованный симметричный ключ: 7BC2D45C743776FF59D76D5EB4BB5047011B5CE02B1BE231640F38ECFFACEBB76A0AC038C7D1DD193E7ABA8B1A9754C6637F685F1D53CD6A24CE1152A22B218A
Подписывание симметричного ключа ...
Подпись: 47D30E307E488C04496845D24DA78C80D9CE4C5FE33EC9975945D7689B9209262B4DC028FA18383DE472FD1550FF23A14ABFD242B5B89D77DD2531E9ED6CE64D
Отправка зашифрованного симметричного ключа и подписи на сервер ...

Операции на сервере:
Расшифровка симметричного ключа...
Симметричный ключ: EEA232AC66ADBF4413F8E00B44616503
Проверка подписи клиента...
Подпись верна.
Генерация случайного сообщения...
Сгенерированное сообщение: 02722E14CA04B346C99C589939AAED936CB86771BF8FDC9D0764A28042982AD767
Генерация вектора инициализации...
Вектор инициализации: A0516B02E097F3F23AD9E78D2068FB8A
Шифрование сообщения ...
Зашифрованное сообщение: F5B66BA588BBEF7632CD675342922E5F90ABBE1652C08336D1F28C12B40C8DC44E017CE01EFB5D4978E18DE7BE4D54
Отправка зашифрованного сообщения клиенту ...

Операции на клиенте:
Расшифровка сообщения...
Расшифрованное сообщение: 02722E14CA04B346C99C589939AAED936CB86771BF8FDC9D0764A28042982AD767
```

Операции на сервере:

Генерация ключей Эль-Гамалю ...

Приватный ключ Эль-Гамалю:

258B2F2A46A0E1EB353265929641CB1F21B9AF5A94B834999CDA9430DB1F9A2B

Публичный ключ Эль-Гамалю:

3B06C2AB9527B33ECAAA086BD57D6B91E7D908104FA45C7043230AC9A6611388

Генерация сертификата ...

Отправка сертификата клиенту...

Операции на клиенте:

Генерация симметричного ключа ...

Симметричный ключ: EEA232AC66ADBF4413F8E00B44616503

Шифрование симметричного ключа ...

Зашифрованный симметричный ключ:

7BC2D45C743776FF59D76D5EB4BB5047011B5CE02B1BE231640F38ECFFACEBB76A0AC038C7D1DD193E7ABA8

B1A9754C6637F685F1D53CD6A24CE1152A22B218A

Подписывание симметричного ключа ...

Подпись:

47D30E307E488C04496845D24DA78C80D9CE4C5FE33EC9975945D7689B9209262B4DC028FA18383DE472FD15
50FF23A14ABFD242B5BB9D77DD2531E9ED6CE64D

Отправка зашифрованного симметричного ключа и подписи на сервер ...

Операции на сервере:

Расшифровка симметричного ключа...

Симметричный ключ: EEA232AC66ADBF4413F8E00B44616503

Проверка подписи клиента...

Подпись верна.

Генерация случайного сообщения...

Сгенерированное сообщение:

02722E14CA04B346C99C589939AAED936CB86771BF8FDC9D0764A28042982AD767

Генерация вектора инициализации...

Вектор инициализации: A0516B02E097F3F23AD9E78D2068FB8A

Шифрование сообщения ...

Зашифрованное сообщение:

F5BE66BA588BBEF7632CD675342922E5F90ABBE1652C0B8336D1F28C12B40C8DC44E017CE01EFB5D4978E18D
E7BE4D54

Отправка зашифрованного сообщения клиенту ...

Операции на клиенте:


Расшифровка сообщения...

Расшифрованное сообщение:

02722E14CA04B346C99C589939AAED936CB86771BF8FDC9D0764A28042982AD767

Далі отриманий сертифікат PKCS#12 було імпортовано, а через консоль керування Майкрософт (MMC) переглянутий формат x.509. Таким чином я стала центром сертифікації та дала собі сертифікат ☺

×


←  Мастер импорта сертификатов

Импортируемый файл
Укажите файл, который вы хотите импортировать.

Имя файла:

Замечание: следующие форматы файлов могут содержать более одного сертификата в одном файле:
Файл обмена личной информацией - PKCS #12 (.PFX,.P12)
Стандарт Cryptographic Message Syntax - сертификаты PKCS #7 (.p7b)
Хранилище сериализованных сертификатов (.SST)

×

←  Мастер импорта сертификатов

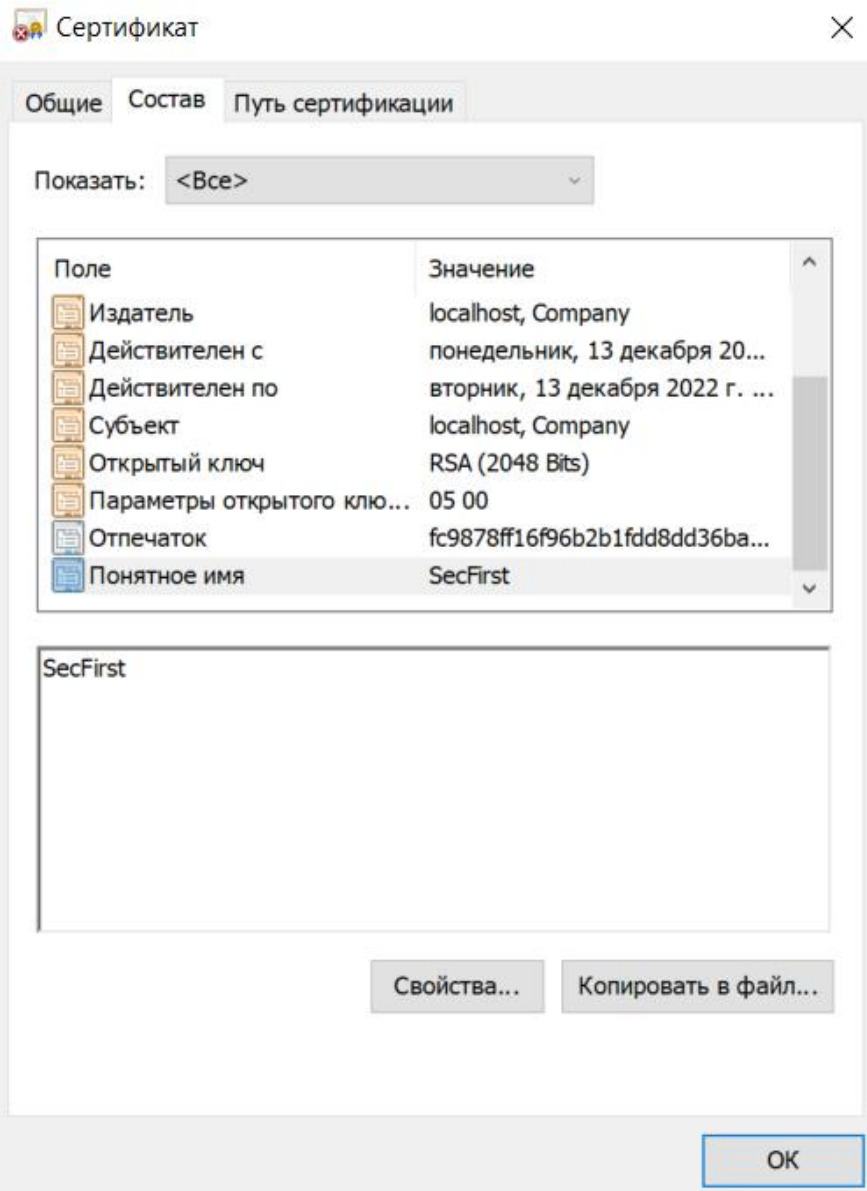
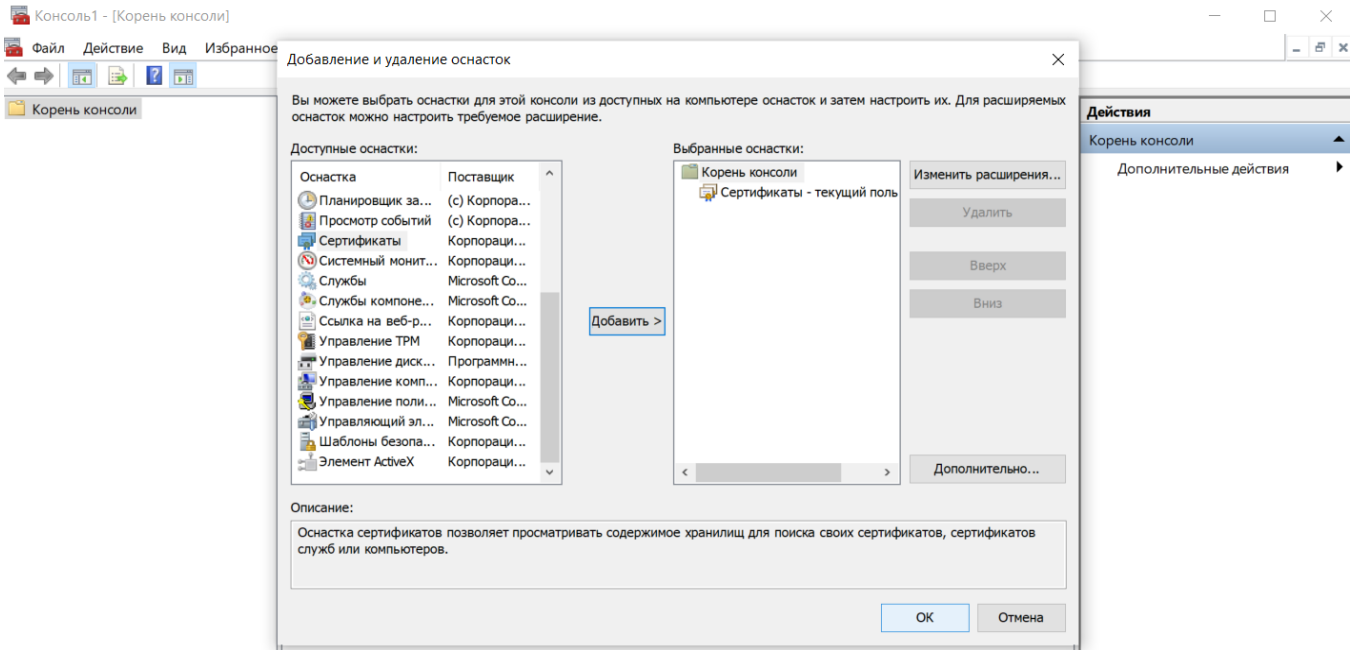
Защита с помощью закрытого ключа
Для обеспечения безопасности закрытый ключ защищен паролем.

Введите пароль для закрытого ключа.

Пароль:

☐ Показывать пароль

Параметры импорта:
☐ Включить усиленную защиту закрытого ключа. В этом случае при каждом использовании закрытого ключа приложением будет запрашиваться разрешение.
☐ Пометить этот ключ как экспортируемый, что позволит сохранять резервную копию ключа и перемещать его.
☐ Защита закрытого ключа с помощью безопасной виртуализации (неэкспортируемый)
☒ Включить все расширенные свойства.



Висновки:

У даній лабораторній роботі я дослідила стандарти Crypto API та PKCS. Зокрема створила самопідписаний сертифікат x.509 з терміном дії на один рік та PKCS#12, що описаний у RFC 7292 сандарті 2014 року, використовуючи бібліотеку OpenSSL.

Lab4.cpp

```
#include "KeyGeneration.h"

#include "MillerRabin.h"

#include <stdio.h>

#include <openssl/x509.h>

#include <openssl/pem.h>

#include <openssl/pkcs12.h>

#include <openssl/err.h>

#include <openssl/applink.c>

#include <openssl/aes.h>

#include <cryptopp/elgamal.h>

#include <cryptopp/osrng.h>

#include <cryptopp/nr.h>

#include <iostream>

#include <iomanip>

#include <sstream>

using namespace std;

using namespace CryptoPP;

X509* generate_x509(EVP_PKEY* pkey) // Генерация сертификата x509

{

    X509* x509 = X509_new(); // Вывделение памяти

    if (!x509) // При ошибке

    {

        std::cerr << "Unable to create X509 structure." << std::endl; // Вывод ошибки
```

```

        return NULL;
    }

    ASN1_INTEGER_set(X509_get_serialNumber(x509), 1); // Формат x509 (номер сертификата)

    X509_gmtime_adj(X509_get_notBefore(x509), 0); // Формат x509

    X509_gmtime_adj(X509_get_notAfter(x509), 31536000L); // Формат x509

    X509_set_pubkey(x509, pkey); // Установка публичного ключа

    X509_NAME* name = X509_get_subject_name(x509); // Получение данных сертификата

    X509_NAME_add_entry_by_txt(name, "U", MBSTRING_ASC, (unsigned char*)"UK", -1, -1, 0); //
Установка данных сертификата

    X509_NAME_add_entry_by_txt(name, "O", MBSTRING_ASC, (unsigned char*)"Company", -1, -1, 0); //
Название компании

    X509_NAME_add_entry_by_txt(name, "CN", MBSTRING_ASC, (unsigned char*)"localhost", -1, -1, 0); //
Сервер

    X509_set_issuer_name(x509, name); // Установка данных сертификата

    if (!X509_sign(x509, pkey, EVP_sha1())) // Подпись сертификата
    {

        std::cerr << "Error signing certificate." << std::endl; // Ошибка

        X509_free(x509);

        return NULL;
    }

    return x509; // Вернуть сертификат x509
}

```

```

PKCS12* generate_pkcs12(EVP_PKEY* key, X509* main, STACK_OF(X509)* sacert) // Генерация сертификата
PKSC
{

    PKCS12* pkcs12;

    if ((pkcs12 = PKCS12_new()) == NULL) // Выделение памяти
    {

        std::cerr << "Error creating PKCS12 structure.\n"; // Ошибка

        return NULL;
    }
}

```

```

    }

    pkcs12 = PKCS12_create("4758vjhkbkerjhuyeduCUTYVB++jht%^&", "SecFirst", key, main, cacert, 0, 0,
0, 0, 0); // Создание сертификата

    if (pkcs12 == NULL) // Если ошибка

    {

        std::cerr << "Error generating a valid PKCS12 certificate.\n"; // Ошибка

        return NULL;

    }

    return pkcs12; // Вернуть сертификат
}

```

```

bool save_pkcs12(PKCS12* pkcs12, const char* path) // Сохранить сертификат PKSC
{

    FILE* pkcs12file; // Файл

    if (!(pkcs12file = fopen(path, "w"))) // Открытие для записи

    {

        std::cerr << "Error cant open pkcs12 certificate file for writing.\n"; // Вывод ошибки

        return false;

    }

    auto bytes = i2d_PKCS12_fp(pkcs12file, pkcs12); // Запись сертификата в файл

    if (bytes <= 0)

    {

        std::cerr << "Error writing PKCS12 certificate.\n"; // Ошибка

        fclose(pkcs12file); // Закрытие файла

        return false;

    }

    fclose(pkcs12file); // Закрытие файла

    return true;

}

```



```
PKCS12* load_pkcs12(const char* path) // Загрузка сертификата PKSC
```

```
{  
  
    FILE* fp; // Файл  
  
    PKCS12* p12; // Сертификат  
  
    if (!(fp = fopen(path, "rb"))) // Открытие файла для чтения  
    {  
        std::cerr << "Error opening file " << path << "\n"; // Ошибка  
        return NULL;  
    }  
  
    p12 = d2i_PKCS12_fp(fp, NULL); // Чтение сертификата  
  
    fclose(fp); // Закртие файла  
  
    return p12; // Вернуть сертификат  
}
```

```
SecByteBlock key_to_bytes(ElGamalKeys::PrivateKey& key) // Преобразование приватного ключа Эль-Гамала в байты
```

```
{  
  
    Integer exponent = key.GetPrivateExponent(); // Получение приватного элемента  
  
    unsigned byteCount = exponent.ByteCount(); // Получение количества байт  
  
    SecByteBlock buffer(byteCount); // Байты приватного ключа  
  
    for (size_t i = 0; i < byteCount; i++) // Проход по ключу  
        buffer[i] = exponent.GetByte(byteCount - i - 1); // Получение байта  
  
    return buffer; // Возврат байт  
}
```

```
EVP_PKEY* generate_key() // Генерация ключа для сертификата
```

```
{
```

```

EVP_PKEY* pkey = EVP_PKEY_new(); // Выделение памяти

if (!pkey) // При ошибке
{
    std::cerr << "Unable to create EVP_PKEY structure." << std::endl; // Вывод ошибки

    return NULL;
}

RSA* rsa = RSA_generate_key(2048, RSA_F4, NULL, NULL); // Генерация ключа 2048 бит
if (!EVP_PKEY_assign_RSA(pkey, rsa)) // Установка ключа в pkey
{
    std::cerr << "Unable to generate RSA key." << std::endl; // Ошибка

    EVP_PKEY_free(pkey);

    return NULL;
}

return pkey; // Возврат ключа
}

SecByteBlock key_to_bytes(ElGamalKeys::PublicKey& key) // Преобразование публичного ключа Эль-Гамала
в байты
{
    Integer exponent = key.GetPublicElement(); // Получение публичного элемента
    unsigned byteCount = exponent.ByteCount(); // Получение количества байт
    SecByteBlock buffer(byteCount); // Байты публичного ключа
    for (size_t i = 0; i < byteCount; i++) // Проход по ключу
        buffer[i] = exponent.GetByte(byteCount - i - 1); // Получение байта
    return buffer; // Возврат байт
}

string to_string(byte* key, size_t lenght) // Преобразование массива байт в hex строку
{

```

```

stringstream stream; // Поток

for (size_t i = 0; i < lenght; i++) // Проход по байтам

    stream << hex << setw(2) << setfill('0') << // Вывод байта в поток

        uppercase << (unsigned int)key[i];

return stream.str(); // Преобразование в строку
}

string to_string(SecByteBlock bytes) // Вывод массива байт в виде hex числа
{
    return to_string(bytes, bytes.size()); // Возврат hex числа из массива
}

typedef NR<CryptoPP::SHA256>::Signer ElGamalSigner; // Класс модифицированной подписи Эль-Гамала
typedef NR<CryptoPP::SHA256>::Verifier ElGamalVerifier; // Класс проверки подписи Эль-Гамала

constexpr int ELGAMAL_KEY_BIT_LENGTH = 256; // Длина ключей Эль-Гамала (в битах)
constexpr int AES_KEY_LENGTH = AES_BLOCK_SIZE; // Длина симметричного ключа (в байтах)
constexpr int ELGAMAL_KEY_LENGTH = ELGAMAL_KEY_BIT_LENGTH / 8; // Длина ключей Эль-Гамала (в байтах)
constexpr int AES_KEY_BIT_LENGTH = AES_KEY_LENGTH * 8; // Длина симметричного ключа (в битах)
constexpr int MESSAGE_BIT_LENGTH = 259; // Длина сообщения
constexpr int MESSAGE_LENGTH = MESSAGE_BIT_LENGTH / 8 + (MESSAGE_BIT_LENGTH % 8 ? 1 : 0); // Длина сообщения в битах
constexpr int ECRYPTED_MESSAGE_LENGTH = MESSAGE_LENGTH - MESSAGE_LENGTH % AES_BLOCK_SIZE + //
Длина зашифрованного сообщения

    (MESSAGE_LENGTH % AES_BLOCK_SIZE ? AES_BLOCK_SIZE : 0);

int main()
{
    setlocale(LC_ALL, "rus"); // Для корректного вывода кириллицы

```

```

AutoSeededRandomPool generator; // Генератора случайных сивел для crypto++

ElGamalDecryptor decryptor; // Расшифровщик Эль-Гамаля

cout << "Операции на сервере:" << endl;

cout << "Генерация ключей Эль-Гамаля ..." << endl;

decryptor.AccessKey().GenerateRandomWithKeySize(generator, ELGAMAL_KEY_BIT_LENGTH); //
Генерация пары ключей Эль-Гамаля

ElGamalKeys::PrivateKey& private_key = decryptor.AccessKey(); // Получение приватного ключа Эль-
Гамаля

SecByteBlock private_key_bytes = key_to_bytes(private_key); // Преобразование приватного ключа
Эль-Гамаля в байты

cout << "Приватный ключ Эль-Гамаля: " << to_string(private_key_bytes) << endl; // Вывод
приватного ключа Эль-Гамаля

ElGamalEncryptor encryptor(decryptor); // Шифровщик Эль-Гамаля

ElGamalKeys::PublicKey& public_key = encryptor.AccessKey(); // Получение публичного ключа Эль-
Гамаля

SecByteBlock public_key_bytes = key_to_bytes(public_key); // Преобразование публичного ключа
Эль-Гамаля в байты

cout << "Публичный ключ Эль-Гамаля: " << to_string(public_key_bytes) << endl; // Вывод
публичного ключа Эль-Гамаля

cout << "Генерация сертификата ..." << endl;

EVP_PKEY* pkey = generate_key(); // Генерация ключа для сертификата

X509* x509 = generate_x509(pkey); // Генерация сертификата x509

STACK_OF(X509)* sacert = sk_X509_new_null(); // Генерация stack сертификата x509

PKCS12* pkcs12 = generate_pkcs12(pkey, x509, sacert); // Генерация сертификата pkcs12

cout << "Отправка сертификата клиенту..." << endl;

if (!save_pkcs12(pkcs12, "certificate.p12")) // Сохранение сертификата
{

    std::cout << "Ошибка отправки сертификата!\n"; // Ошибка

```

```

        return 1;
    }

    X509_free(x509); // Освобождение памяти

    PKCS12_free(pkcs12); // Освобождение памяти


    cout << "\nОперации на клиенте:" << endl;

    pkcs12 = load_pkcs12("certificate.p12"); // Загрузка сертификата
    if (!pkcs12)
    {
        std::cerr << "Ошибка получения файла сертификата!\n"; // Ошибка

        return 1;
    }

    pkey = NULL;

    if (!PKCS12_parse(pkcs12, "4758vjhkbkerjhuyeduCUTYVB++jht%^&", &pkey, &x509, &cert)) //
Парсинг сертификата
    {
        std::cerr << "Ошибка парсинга файла сертификата!\n";

        return 1;
    }

    PKCS12_free(pkcs12); // Освобождение памяти

    EVP_PKEY_free(pkey); // Освобождение памяти

    SecByteBlock client_symmetric_key(AES_KEY_LENGTH); // Симметричный ключ (для AES)

    auto encrypted_key_lenght = encryptor.CiphertextLength(client_symmetric_key.size()); // Длина
зашифрованного симметричного ключа

    SecByteBlock encrypted_symmetric_key(encrypted_key_lenght); // Зашифрованный симметричный
ключ

    cout << "Генерация симметричного ключа ..." << endl;

    generate_key(client_symmetric_key, AES_KEY_BIT_LENGTH); // Генерация симметричного ключа
(Генератором BBS и проверкой Миллера-Рабина)

    cout << "Симметричный ключ:          " << to_string(client_symmetric_key) << endl; // Вывод

```

симметричного ключа

```
cout << "Шифрование симметричного ключа ..." << endl;

encryptor.Encrypt(generator, client_symmetric_key, AES_KEY_LENGTH, encrypted_symmetric_key); //
Шифрование симметричного ключа

cout << "Зашифрованный симметричный ключ: " << to_string(encrypted_symmetric_key) << endl; //
Вывод зашифрованного симметричного ключа

cout << "Подписывание симметричного ключа ..." << endl;

ElGamalSigner signer(private_key); // Подписыватель Эль-Гамала

auto sign_length = signer.SignatureLength(); // Длина подписи Эль-Гамала

SecByteBlock sign(signer.SignatureLength()); // Подпись Эль-Гамала

signer.SignMessage(generator, client_symmetric_key, AES_KEY_LENGTH, sign); // Подписывание
симметричного ключа

cout << "Подпись:          " << to_string(sign) << endl; // Вывод подписи

cout << "Отправка зашифрованного симметричного ключа и подписи на сервер ..." << endl;

cout << "\nОперации на сервере:" << endl;

cout << "Расшифровка симметричного ключа..." << endl;

SecByteBlock server_symmetric_key(AES_KEY_LENGTH); // Расшифрованный симметричный ключ

decryptor.Decrypt(generator, encrypted_symmetric_key, encrypted_key_lenght,
server_symmetric_key); // Расшифрование симметричного ключа

cout << "Симметричный ключ:          " << to_string(server_symmetric_key) << endl; // Вывод
симметричного ключа

cout << "Проверка подписи клиента..." << endl;

ElGamalVerifier verifier(public_key); // Объект для проверка подписи Эль-Гамала

cout << (verifier.VerifyMessage(server_symmetric_key, AES_KEY_LENGTH, sign, sign_length) //
Проверка подписи Эль-Гамала

? "Подпись верна." : "Подпись не верна!") << endl;

cout << "Генерация случайного сообщения..." << endl;

SecByteBlock message(MESSAGE_LENGTH); // Сообщения
```

```

for (int i = MESSAGE_LENGTH - 1; i >= 0; i--) // Проход по сообщению
    message[i] = (i == 0 ? (rand_bbs() % (1 << (MESSAGE_BIT_LENGTH % 8))) : rand_bbs()); //
Случайный байт

cout << "Сгенерированное сообщение:   " << to_string(message) << endl; // Вывод сообщения

cout << "Генерация вектора инициализации..." << endl;

SecByteBlock server_init_vector(AES_BLOCK_SIZE); // Вектор инициализация сервера

SecByteBlock client_init_vector(AES_BLOCK_SIZE); // Вектор инициализация клиента

for (size_t i = 0; i < AES_BLOCK_SIZE; i++) // Проход по байтам ветора инициализации
{
    auto byte = rand_bbs(); // Случайный байт

    server_init_vector[i] = byte; // Установка байта в вектор инициализация сервера

    client_init_vector[i] = byte; // Установка байта в вектор инициализация клиента
}

cout << "Вектор инициализации:      " << to_string(server_init_vector) << endl; // Вывод вектора
инициализация

cout << "Шифрование сообщения ..." << endl;

AES_KEY aes_key; // Ключ для aes шифрования

SecByteBlock encrypted_message(ENCRYPTED_MESSAGE_LENGTH); // Зашифрованный байты
сообщения

AES_set_encrypt_key(server_symmetric_key, AES_KEY_BIT_LENGTH, &aes_key); // Установка ключа
шифрования

AES_cbc_encrypt(message, encrypted_message, MESSAGE_LENGTH, &aes_key, server_init_vector,
AES_ENCRYPT); // AES CBC шифрование

cout << "\rЗашифрованное сообщение:   " << to_string(encrypted_message) << endl; // Вывод
зашифрованного сообщения

cout << "Отправка зашифрованного сообщения клиенту ..." << endl;

cout << "\nОперации на клиенте:" << endl;

cout << "Расшифровка сообщения..." << endl;

SecByteBlock decrypted_message(MESSAGE_LENGTH); // Расшифрованное сообщение

```

```

        AES_set_decrypt_key(server_symmetric_key, AES_KEY_BIT_LENGTH, &aes_key); // Установка ключа
        дешифровки

        AES_cbc_encrypt(encrypted_message, decrypted_message, MESSAGE_LENGTH, &aes_key,
        client_init_vector, AES_DECRYPT); // AES CBC дешифровка

        cout << "Расшифрованное сообщение: " << to_string(decrypted_message) << endl; // Вывод
        расшифрованного сообщения

        return 0;

    }

```

KeyGeneration.cpp

```

#pragma once

#include "KeyGeneration.h"
#include "RandomBBS.h"
#include "MillerRabin.h"
#include <iostream>

void generate_key(unsigned char* key, size_t lenght) // Генерация симметричного ключа
{
    size_t byte_lenght = lenght / 8; // Количество байт

    Integer p, q; // Переменные

    auto bytes = new byte[byte_lenght]; // Байты ключа

    do
    {
        do
        {
            for (size_t i = 0; i < byte_lenght; i++) // Проход по массиву байт

                bytes[i] = rand_bbs(); // Случайный байт

            p = Integer(bytes, byte_lenght); // Создание большого числа из массива байт

        } while (!miller_rabin(p)); // Проверка p алгоритмом Миллера-Рабина

        q = 2 * p + 1; // Вычисление q = 2p + 1

    } while (!miller_rabin(q)); // Проверка q алгоритмом Миллера-Рабина

    for (int i = (int)byte_lenght - 1; i >= 0; i--) // Проход по байтам q

```



```

        key[i] = q.GetByte(byte_lenght - i - 1); // Копирование байта в ключ

        delete[] bytes; // Освобождение памяти
    }

KeyGeneration.h

#pragma once

void generate_key(unsigned char* key, size_t lenght); // Генерация симметричного ключа

RandomBBS.cpp

#pragma once

#include "RandomBBS.h"

Integer pow_mod(Integer base, Integer exp, Integer modulus) // Степень по модулю
{
    base %= modulus; // Остаток от деления

    Integer result = 1; // Единица по умолчанию

    while (exp > 0) // Пока степень больше нуля
    {
        if (exp.GetBit(0)) // Проверка на четность
            result = (result * base) % modulus; // Вычисление остатка от деления

        base = (base * base) % modulus; // Возведение в квадрат по модулю

        exp >>= 1; // Деление степени на 2
    }

    return result; // Возврат результата
}

Integer n = Integer("D5BBB96D30086EC484EBA3D7F9CAEB07") *
Integer("425D2B9BFDB25B9CF6C416CC6E37B59C1F"); // Начальное значение n для BBS

Integer r = Integer("675215CC3E227D321097E1DB049F1"); // Начальное значение r для BBS

unsigned char rand_bbs() // Случайный байт по алгоритму BBS
{

```

```

        r = pow_mod(r, 2, n); // Возведение в квадрат по модулю n

        return r.GetByte(0); // Получение младших 8 бит
    }

```

RandomBBS.h

```

#pragma once

#include <integer.h>

using namespace CryptoPP;

Integer pow_mod(Integer x, Integer y, Integer p); // Степень по модулю

unsigned char rand_bbs(); // Случайный байт по алгоритму BBS

```

MillerRabin.cpp

```

#pragma once

#include "MillerRabin.h"

#include <sstream>

#include <iomanip>

std::string to_string(byte* key, size_t lenght) // Преобразование массива байт в hex строку
{
    std::stringstream stream; // Поток

    for (size_t i = 0; i < lenght; i++) // Проход по байтам

        stream << std::hex << std::setw(2) << std::setfill('0') << std::uppercase << (unsigned int)key[i]; //
Вывод байта в поток

    return stream.str(); // Преобразование в строку
}

```

```

Integer big_rand(unsigned bit_number) // Генерация случайных <bit_number> битных чисел
{
    unsigned byte_number = bit_number / 8; // Количество байт

    if (bit_number > 0 && byte_number == 8) // Если меньше 1 байта но больше 0 бит

        byte_number = 1; // 1 Байт
}

```

```

if (byte_number) // Если больше нуля байт
{
    SecByteBlock bytes(byte_number); // Байты числа

    for (size_t i = 0; i < byte_number; i++) // Проход по байтам
        bytes[i] = rand_bbs(); // Случайный байт по алгоритму BBS

    Integer number(bytes, byte_number); // Создание большого числа

    if (number < 0) // Если отрицательное
        number *= -1; // Преобразование в положительное

    return number; // Возврата числа
}

else

    return Integer(0); // Вернуть ноль
}

Integer big_rand(Integer min, Integer max, unsigned bit_nuber) // Генерация случайных <bit_number> битных
чисел от min до max
{
    return min + big_rand(bit_nuber) % (max - min); // Возврат числа min до max
}

bool miller_rabin(const Integer p, int k) // Алгоритм Миллера-Рабина
{
    if (p < 2 || (p != 2 && p % 2 == 0) || (p != 3 && p % 3 == 0) ||
        (p != 5 && p % 5 == 0)) // Проверка на то что p < 2, или p четное, или p кратно 3, 5
        return false;

    Integer pm1 = p - 1; // p - 1 (Что бы каждый раз не считать)

    // Разложение на d * 2 ^ s
    Integer d = pm1; // P - 1

    while (d % 2 == 0) // Пока d четное

```

```

d = d / 2; // Делим на 2

for (int i = 0; i < k; i++) // k итераций
{
    Integer x = big_rand(1, pm1), t = d; // Случайное число от 1 до p - 1;

    Integer xr = pow_mod(x, t, p); // x ^ d

    while (t != pm1 && xr != 1 && xr != pm1) // Пока xr не равно 1 или -1 по модулю p и t != p -
1 (от 1 до s)
    {
        xr = pow_mod(xr, 2, p); // xr = x ^ (d * 2 ^ r)

        t = t * 2; // Домножает d на 2 (r принадлежит [1, s])

    }

    if (xr != pm1 && t % 2 == 0) // Проверяем условия псевдопростоты

        return false; // Возврат (составное)

    }

    return true; // Возврат (простое)
}

```

MillerRabin.h

```

#pragma once

#include "RandomBBS.h"

#include <string>

std::string to_string(byte* key, size_t lenght); // Преобразование массива байт в hex строку

Integer big_rand(unsigned bitNuber); // Генерация случайных <bit_number> битных чисел

Integer big_rand(Integer min, Integer max, unsigned bitNuber = 256); // Генерация случайных <bit_number>
битных чисел от min до max

bool miller_rabin(const Integer p, int k = 15); // Алгоритм Миллера-Рабина

```