



Міністерство освіти і науки України

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

**КОМП’ЮТЕРНИЙ ПРАКТИКУМ З ДИСЦИПЛІНИ  
«МЕТОДИ РЕАЛІЗАЦІЇ КРИПТОГРАФІЧНИХ  
МЕХАНІЗМІВ» №3**

Реалізація основних асиметричних криптосистем

**Виконала:**

студентка групи ФІ-12мн

Звичайна Анастасія Олександрівна

**Перевірила:**

Селюх Поліна Валентинівна

Київ 2021

**Мета роботи:** Дослідити можливість побудови загальних та спеціальних криптографічних протоколів за допомогою асиметричних криптосистем.






**Умова задачі:** Розробити реалізацію асиметричної криптосистеми Ель-Гамала, використовуючи бібліотеку OpenSSL (<https://www.openssl.org/docs/man1.1.1/>), навести приклади роботи.

### Виконання програми:

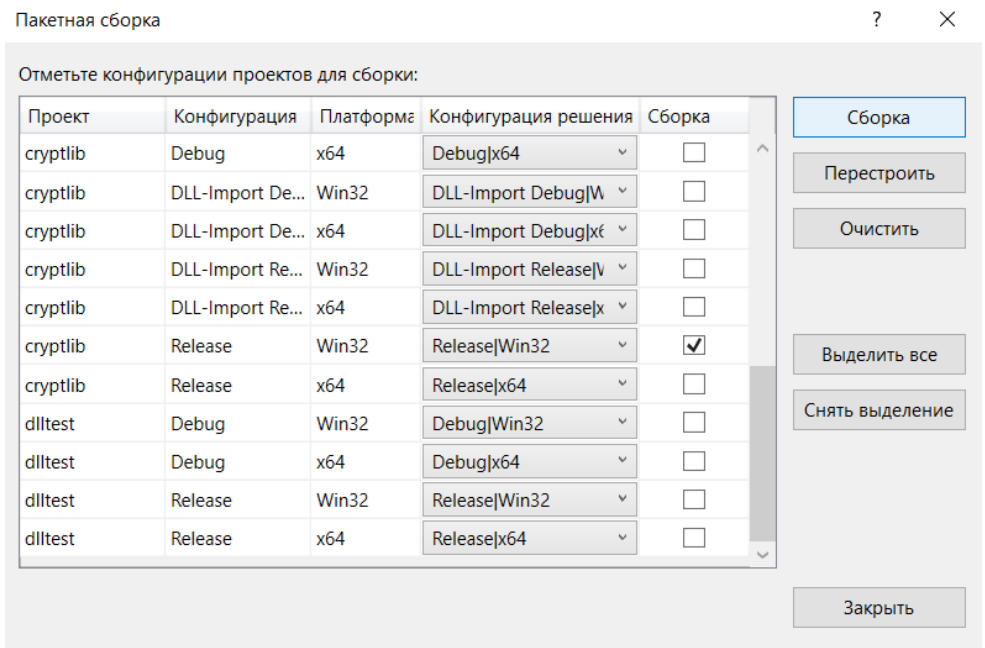
Вважаю за потрібним розписати покрокову схему підключення бібліотек crypto++ та OpenSSL у проект Microsoft Visual Studio 2017, адже це є дуже важливим кроком, без якого робота не може бути зробленою.

Спочатку з гітхабу офіційної розробки crypto++ качаємо дану бібліотеку у вигляді архіву (у моєму випадку cryptopp860.zip).

▼ Assets 4

 <a href="#">cryptopp860.zip</a> 	8.84 MB
 <a href="#">cryptopp860.zip.sig</a>	659 Bytes
 <a href="#">Source code (zip)</a>	
 <a href="#">Source code (tar.gz)</a>	

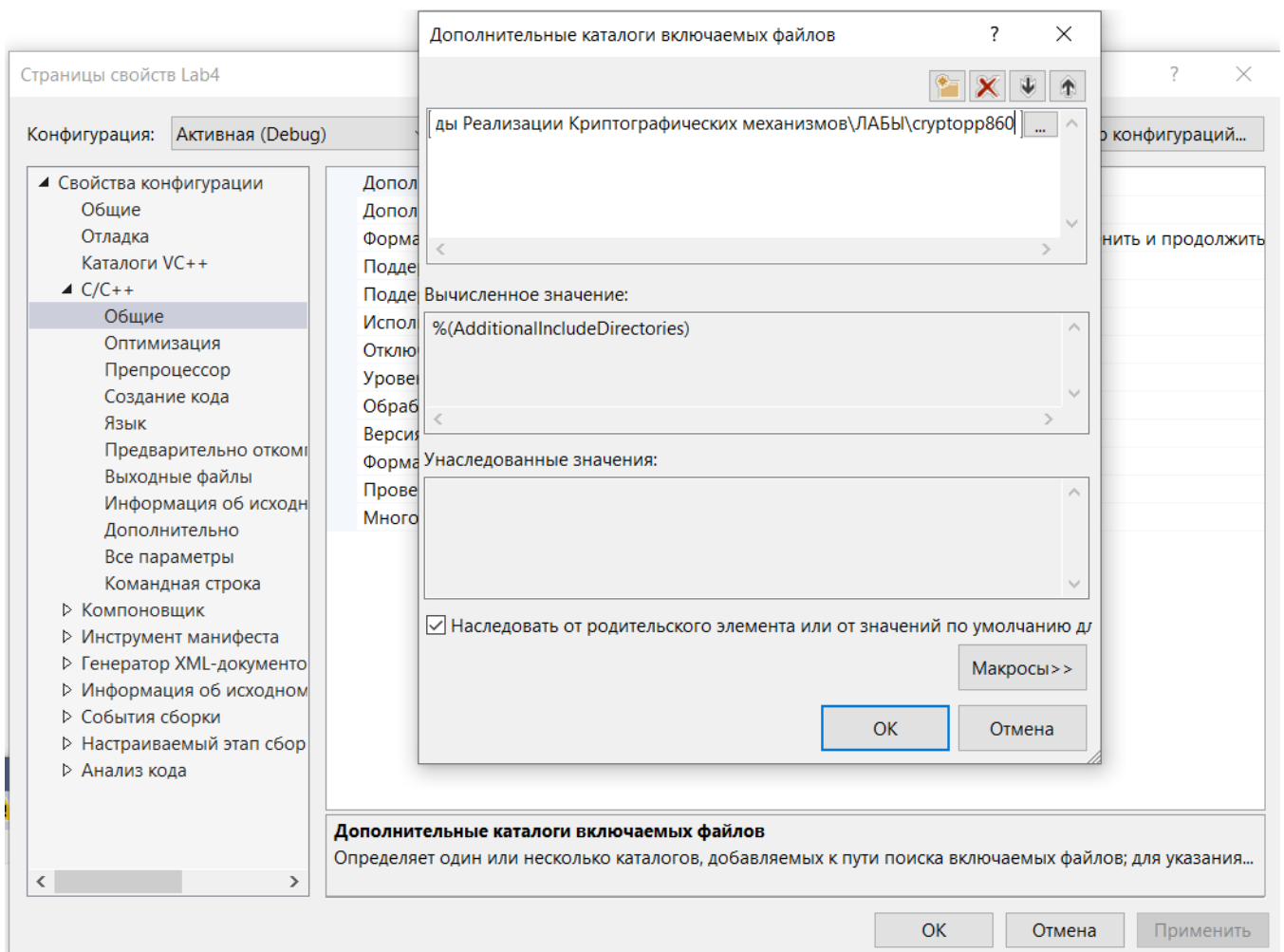
У розпакованому архіві знаходимо файл crypttest.sln, що відповідає за весь проект бібліотеки, та відкриваємо його у Microsoft Visual Studio 2017. Вибираємо пункт «Збірка» -> «Пакетна збірка». Я вибрала статичну



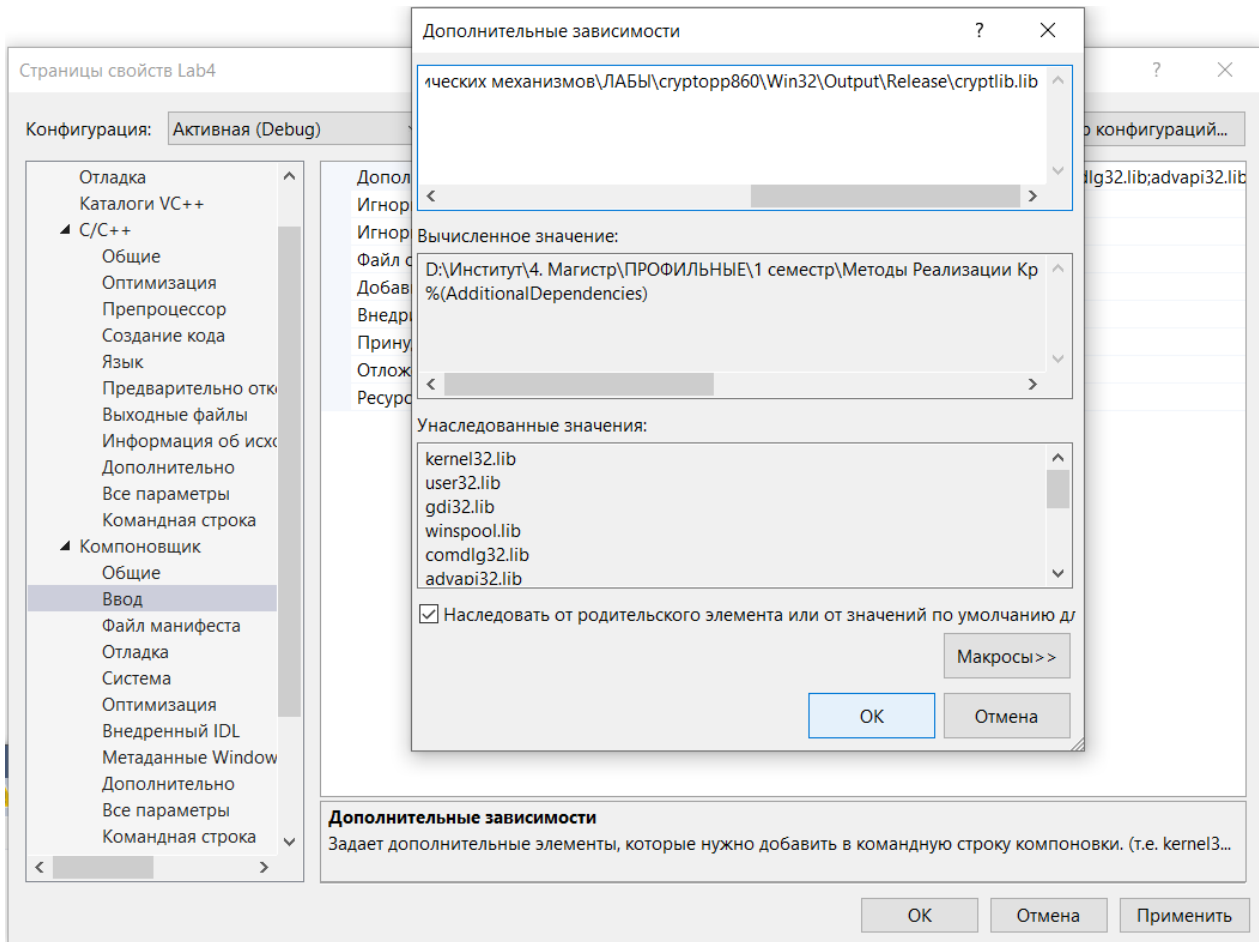
бібліотеку .lib, адже так менше проблем 😊. Відповідно ставимо галку біля cryptolib | Release | Win32 та тиснемо на «Збірка». У результаті отримуємо таке:

```
Вывод
Показать выходные данные из: Сборка
xtrcrypt.cpp
xts.cpp
zdeflate.cpp
zinflate.cpp
zlib.cpp
Создание кода...
chacha_avx.cpp
dll.cpp
iterhash.cpp
Создание кода...
cryptlib.vcxproj -> D:\Институт\4. Магистр\ПРОФИЛЬНЫЕ\1 семестр\Методы Реализации Криптографических механизмов\ЛАБЫ\cryptopp860\Win32\Output\Release\cryptlib.lib
===== Сборка: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 =====
|
```

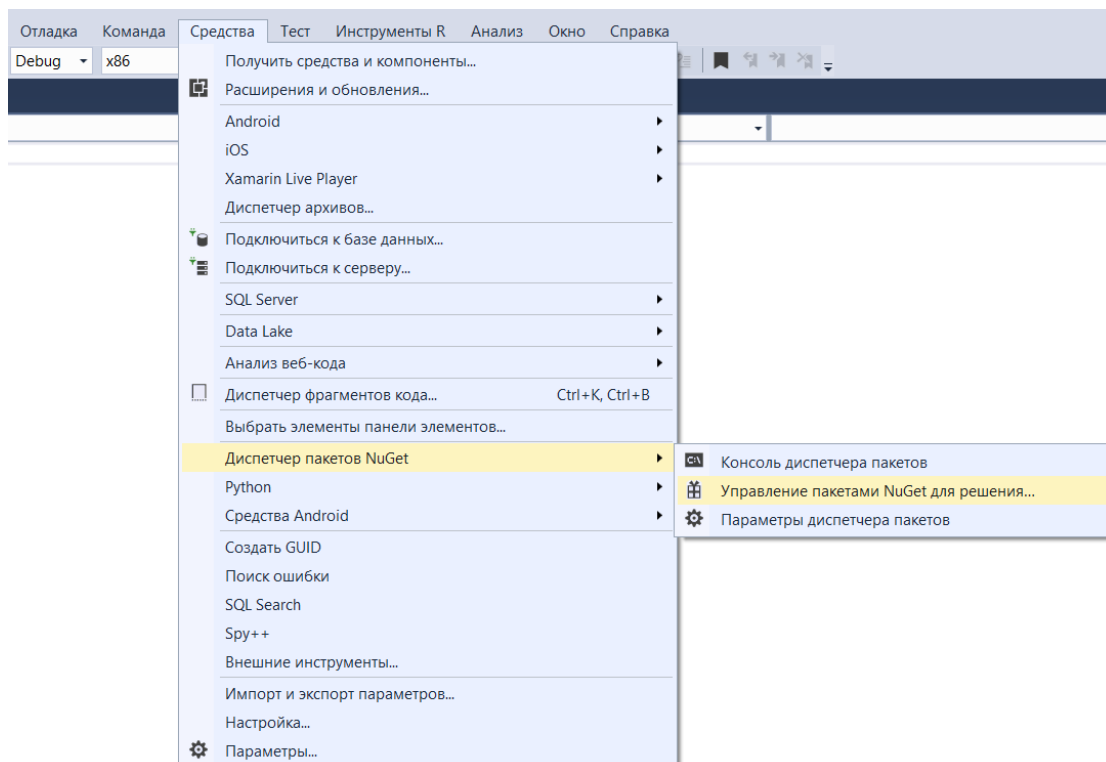
Тепер залишилось підключити дану бібліотеку до проекту Lab3. Для цього у властивостях прописуємо шлях до розпакованої бібліотеки (cryptopp860), а також до компілюемого файлу (cryptlib.lib) як на рисунках нижче. Окрім цього, змінюємо значення «Бібліотеки часу виконання» на «Багатопотокової/MT» (адже ми працюємо з .lib). Бібліотека готова до використання 😊







Тепер залишилось приєднати бібліотеку OpenSSL, що зробити легко, адже дана бібліотека є у NuGet, тому достатньо її просто приєднати у проект безпосередньо у Microsoft Visual Studio 2017 і все 😊



NuGet — решение Lab4.cpp

Обзор Установлено Обновления Консолидировать

Управление пакетами для решения

openssl-vc

Источник пакета: nuget.org

**openssl-vc141-static-x86\_64** автор: okanon, Скачиваний: **44,3K** v1.1.0

libssl library packaged for Visual Studio 2017 (msvc141). This library is Cryptography and SSL/TLS Toolkit.

**openssl-vc140-static-32\_64** автор: OpenSSL Software Foundation, Скачиваний: **28,8K** v1.1.1.1

OpenSSL is an open source project that provides a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library. For more informat...

**openssl-vc141** автор: The OpenSSL Project, Скачиваний: **7,89K** v1.1.0

OpenSSL library packaged for Visual Studio 2017 (msvc141). This package is experimental, so use at your own risk.

**openssl-vc140-vc141-x86\_64** автор: OpenSSL, Скачиваний: **9,14K** v1.1.5

OpenSSL prebuild for VC140 and VC141

**openssl-vc141-static-x86\_64**

Версии — 1

Проект	Версия
Lab4	1.1.0

## Приклад виконання програми:

```
Консоль отладки Microsoft Visual Studio

Операции на сервере:
Генерация ключей Эль-Гамала ...
Приватный ключ Эль-Гамала: 22C4CEC6CB114F675A259993FCA71A08239AC402E2AFE21E7866079894681FD5
Публичный ключ Эль-Гамала: 1CDA7EEFDAEC1A4FCD0C492CB01AE6F985D9AF9E77B5F73C466A3494DDB414DF
Отправка публичного ключа клиенту ...

Операции на клиенте:
Генерация симметричного ключа ...
Симметричный ключ: EEA232AC66ADB4413F8E00B44616503
Шифрование симметричного ключа ...
Зашифрованный симметричный ключ: 0AD0A57EF57ED41C7822010D7D762EF36378FEAD6135753017E34295FC584CD48B90B81A8D32466199A8B83BE6F7F28500BA27B125669D77B31FC3CD503A0C
Подписывание симметричного ключа ...
Подпись: 0C1698DE9C784C26735E9AC370E5F0FF476BA718C8598E1787FD0FA4294A46211A7731CF0799CA33B5AB40EE27AEFBEADEE5ECE6EAE00FC133469E858568D63C
Отправка зашифрованного симметричного ключа и подписи на сервер ...

Операции на сервере:
Расшифровка симметричного ключа...
Симметричный ключ: EEA232AC66ADB4413F8E00B44616503
Проверка подписи клиента...
Подпись верна.

Генерация случайного 128-битного сообщения...
Сгенерированное сообщение: 67D72A984280A264079DDC8F8BF7167B8
Шифрование сообщения ...
Зашифрованное сообщение: B228F50827D50D36034A3CC9EB7AC7E2
Отправка зашифрованного сообщения клиенту ...

Операции на клиенте:
Расшифровка сообщения...
Расшифрованное сообщение: 67D72A984280A264079DDC8F8BF7167B8

Генерация случайного 259-битного сообщения...
Сгенерированное сообщение: 02FB68208DE7D93AF2F397E0026B51A092722E14CA04B346C99C589939AAED936C
Генерация вектора инициализации...
Вектор инициализации: 6F76BECC533990330E648FEB24B3C10
Шифрование сообщения ...
Зашифрованное сообщение: 525C7E445D095A501EF89995D032FF2CE2F570CBED99771B2144258F0496745725721399C4F44D2795A6418AD0CB20C
Отправка зашифрованного сообщения клиенту ...

Операции на клиенте:
Расшифровка сообщения...
Расшифрованное сообщение: 02FB68208DE7D93AF2F397E0026B51A092722E14CA04B346C99C589939AAED936C
```

Операции на сервере:

Генерация ключей Эль-Гамала ...

Приватный ключ Эль-Гамала:

22C4CEC6CB114F675A259993FCA71A08239AC402E2AFE21E7866079894681FD5

Публичный ключ Эль-Гамала:

1CDA7EEFDAEC1A4FCD0C492CB01AE6F985D9AF9E77B5F73C466A3494DDB414DF

Отправка публичного ключа клиенту ...

Операции на клиенте:

Генерация симметричного ключа ...

Симметричный ключ: EEA232AC66ADBF4413F8E00B44616503

Шифрование симметричного ключа ...

Зашифрованный симметричный ключ:

0AD0A57EF57ED41C7822010D7D762EF36378EFEAD6135753017E34295FC584CD4BEB90B81A8D32466199A8B8  
3BE6F7F2B500BA27B125669D77B31FC3CD503A0C

Подписывание симметричного ключа ...

Подпись:

0C1698DE9C784C26735E9AC370E5F0FF476BA718C8598E1787FD0FA4294A46211A7731CF0799CA33B5AB40EE2  
7AEFBEADEE5ECE6EAE00FC133469E85856BD63C

Отправка зашифрованного симметричного ключа и подписи на сервер ...

Операции на сервере:

Расшифровка симметричного ключа...

Симметричный ключ: EEA232AC66ADBF4413F8E00B44616503

Проверка подписи клиента...

Подпись верна.

Генерация случайного 128-битного сообщения...

Сгенерированное сообщение: 67D72A984280A264079DDC8FBF7167B8

Шифрование сообщения ...

Зашифрованное сообщение: B228F50827D50D36034A3CC9EB7AC7E2

Отправка зашифрованного сообщения клиенту ...

Операции на клиенте:

Расшифровка сообщения...

Расшифрованное сообщение: 67D72A984280A264079DDC8FBF7167B8

Генерация случайного 259-битного сообщения...

Сгенерированное сообщение:

02FB68208DE7D93AF2F397E0026B51A092722E14CA04B346C99C589939AAED936C

Генерація вектора ініціалізації...

Вектор ініціалізації: 6F76BECC5339903330E64BFEB24B3C10

Шифрування повідомлення ...

Зашифроване повідомлення:

525C7E445D095A501EF89995D032FFF2CE2F570CBED99771B2144258F0496745725721399C4F44D2795A6418A  
D0CB20C

Отправка зашифрованного сообщения клиенту ...

Операции на клиенте:

Расшифровка сообщения...

Расшифрованное сообщение:

02FB68208DE7D93AF2F397E0026B51A092722E14CA04B346C99C589939AAED936C

## Висновки:

У даній лабораторній роботі я реалізувала криптосистему Ель-Гамалю на мові C++, використовуючи бібліотеку `crypto++`. Окрім цього, використовуючи `OpenSSL`, я реалізувала симетричну схему шифрування AES з режимом роботи CBC. Для генерації ключів Ель-Гамалю я використовувала вбудований генератор `crypto++` [AutoSeededRandomPool](#), а для генерації симетричного ключа використовувала власний алгоритм побудови ключа за допомогою генератора BBS та власну реалізацію тесту Міллера-Рабіна. Для генерації вектора ініціалізації використовувала BBS. Таким чином я практично ознайомила з гібридними криптосистемами.

## Lab3.cpp

```
#include "KeyGeneration.h"
```

```
#include "MillerRabin.h"
```

```
#include <iostream>
```

```
#include <sstream>
```

```
#include <iomanip>
```

```
#include <elgamal.h>
```

```
#include <osrng.h>
```

```
#include <nr.h>
```

```
#include <openssl/aes.h>
```



```

using namespace std;

using namespace CryptoPP;

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
}

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
}

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') << // Вывод байта в поток
            std::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<SHA256>::Signer ElGamalSigner; // Класс модифицированной подписи Эль-Гамала
typedef NR<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 ENCRYPTED_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-Гамала

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

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

```

```

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

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

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

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

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

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

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

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

    cout << "Отправка публичного ключа клиенту ..." << endl;

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

    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_length, 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 проверки 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 << endl << "Генерация случайного 128-битного сообщения..." << endl;

    unsigned message_bytes_length = 128 / 8; // Длина сообщения (128 бит)
    SecByteBlock message2(message_bytes_length); // Сообщение
    for (size_t i = 0; i < message_bytes_length; i++) // Проход по сообщению
        message2[i] = rand_bbs(); // Случайный байт

    cout << "Сгенерированное сообщение: " << to_string(message2) << endl; // Вывод сообщения

```

```

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

AES_KEY aes_key2; // Ключ для aes шифрования

AES_set_encrypt_key(server_symmetric_key, AES_KEY_BIT_LENGTH, &aes_key2); // Установка
симметричного ключа

SecByteBlock encrypted_message2(message_bytes_lenght); // Зашифрованные байты сообщения

AES_encrypt(message2, encrypted_message2, &aes_key2); // AES шифрование

cout << "\rЗашифрованное сообщение:      " << to_string(encrypted_message2) << endl; // Вывод
зашифрованного сообщения

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

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

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

SecByteBlock decrypted_message2(message_bytes_lenght); // Расшифрованное сообщение

AES_set_decrypt_key(client_symmetric_key, AES_KEY_BIT_LENGTH, &aes_key2); // Установка
симметричного ключа

AES_decrypt(encrypted_message2, decrypted_message2, &aes_key2); // Расшифровка сообщения

cout << "Расшифрованное сообщение:      " << to_string(decrypted_message2) << endl; // Вывод
расшифрованного сообщения


cout << endl << "Генерация случайного 259-битного 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(0l); // Вернуть ноль
    }

Integer big_rand(Integer min, Integer max, unsigned bit_nuber) // Генерация случайных <bit_nuber> 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

```