Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление подготовки: 09.03.01 “Информатика и вычислительная техника”
Профиль: “Вычислительные машины, комплексы, системы и сети”

Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

К защите допустить:
Заведующий кафедрой
д. т. н., профессор _____ М. С. Куприянов

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема: “БИБЛИОТЕКА ДЛЯ ОБЕСПЕЧЕНИЯ ШИФРОВАНИЯ КАНАЛА СВЯЗИ”

Студент _____ Н. В. Самоуков

Руководитель
_____ М. Н. Гречухин

Консультант
Доктор экон. наук, профессор _____ В. П. Семенов

Консультант от кафедры
к. т. н., доцент, с. н. с. _____ И. С. Зуев

Санкт-Петербург
2019 г.

Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление 09.03.01 “Информатика и вычислительная техника”
Профиль “Вычислительные машины, комплексы, системы и сети”
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 201__г.

ЗАДАНИЕ
на выпускную квалификационную работу

Студент Самоуков Н. В. Группа № **5307**

1. Тема **Библиотека для обеспечения шифрования канала связи**

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ»

2. Объект и предмет исследования: криптографические алгоритмы

3. Цель: создание системы криптографической защиты, передаваемой по отрытым каналам информации

4. Исходные данные: описания алгоритмов, документация по WinSock API

5. Содержание: анализ описания алгоритмов, изучение математических вопросов, непосредственно связанных с реализуемыми алгоритмами, реализация библиотеки «длинной алгебры» и генерации случайных чисел, реализация алгоритмов, тестирование программного продукта, оформление пояснительной записки

6. Технические требования создаваемая программа должна обеспечивать поддержку шифрования информации различными алгоритмами, отправку по сети с использованием штатных средств ОС. Криптографическая часть должна быть кроссплатформенная, сетевая часть реализуется при помощи

сетевых инструментов и API соответствующих систем. Программа должна работать в ОС Windows версии не ниже 7.

7. Дополнительные разделы: обеспечение качества разработки программного продукта

8. Результаты: Программный комплекс, пояснительная записка к ВКР, презентация.

Дата выдачи задания
«_____» _____ 201__ г.

Дата представления ВКР к защите
«_____» _____ 201__ г.

Руководитель _____ М. Н. Гречухин

Студент _____ Н. В. Самоуков

Санкт-Петербургский государственный электротехнический университет
“ЛЭТИ” им. В. И. Ульянова (Ленина)
(СПбГЭТУ “ЛЭТИ”)

Направление 09.03.01 “Информатика и вычислительная техника”
Профиль “Вычислительные машины, комплексы, системы и сети”
Факультет компьютерных технологий и информатики
Кафедра вычислительной техники

УТВЕРЖДАЮ
Заведующий кафедрой ВТ
д. т. н., профессор
(М. С. Куприянов)
“___” _____ 201__г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Тема **Библиотека для обеспечения шифрования канала связи**

Студент Самоуков Н. В. Группа № **5307**

№ этапа	Наименование работ	Срок выполнения
1	Подбор и изучение литературы по теме работы	
2	Обзор существующих решений	
3	Реализация библиотеки «длинной алгебры» и генерации случайных целых чисел	
4	Реализация алгоритмов шифрования и обмена ключами	
5	Реализация и тестирование сетевого обмена	
5	Тестирование производительности программы	
6	Оформление пояснительной записки	
7	Предварительное рассмотрение работы	
8	Представление работы к защите	

Руководитель _____ М. Н. Гречухин

Студент _____ Н. В. Самоуков

РЕФЕРАТ

Данная работа посвящена шифрованию канала связи. Целью работы является создание библиотеки шифрования. Эта библиотеки должна обеспечивать зашифрованный канал связи.

Для создания библиотеки изучаются криптографические алгоритмы и их реализация.

В результате создана библиотека шифрования реализующая гибридный метод, суть которого в комбинировании нескольких разных алгоритмов для закрытия их минусов. Написана она на C++. Скорость её работы на ПК с процессором 3.4 ГГц превышает скорость стандартного соединения интернет (100 Мбит/с) в 2 раза.

Данная библиотека может применятся на обычном ПК.

В результате получена библиотека шифрования, которая может иметь обширное применение. Например, обеспечить шифрование личной информации.

ABSTRACT

This work is devoted to encrypting the communication channel. The purpose of the work is to create an encryption library. This library should provide an encrypted communication channel.

To create a library, cryptographic algorithms and their implementation are studied.

As a result, an encryption library was created that implements a hybrid method, the essence of which is in combining several different algorithms to close their minuses. It is written in C ++. Its speed on a PC with a 3.4 GHz processor exceeds the speed of a standard Internet connection (100 Mbps) by 2 times.

This library can be used on a regular PC.

The result is an encryption library that can be widely used. For example, encrypt personal information.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В пояснительной записке присутствуют следующие термины с соответствующими определениями:

ПК – персональный компьютер

Простая задача – задача выполнимая за разумное время

Сложная задача – задача для решения которой требуется большие вычислительные мощности, например, 1000 лет работы суперкомпьютера

Односторонняя функция – функция вычисление которой является простой задачей, а её обращение, сложная задача

ОС – операционная система

ВВЕДЕНИЕ

Сейчас шифрование связи крайне важно, так как касается каждого. Сотовая связь, интернет сайты, сервисы почты везде может содержаться конфиденциальная и секретная информация.

На данный момент существует множество криптографических библиотек, в большинстве из них есть способы создания шифрованного канала связи. Однако, их исходный код имеет более 10000 строк, в которых могут содержаться закладки. Пример такой закладки – предварительно сгенерированное p для протокола Диффи-Хелмана [2]. Соответственно для того, чтобы исключить такую возможность планируется сделать библиотеку малого размера.

Цель данной работы обеспечить защищённое соединение, например, соединение клиент-сервер. Соединение типа клиент-сервер самое популярное сейчас, так как соединение большинства клиентов в интернете невозможно напрямую.

Объект исследования – криптографические алгоритмы.

Предмет исследования – шифрование канала связи.

Задачи: обеспечить надёжно зашифрованный канал связи, создать библиотеку шифрования, имеющую малый размер.

Для реализации канала связи предполагается выбрать несколько вариантов шифрования и реализовать их для сравнения. Предполагается, что при использовании канала будут передаваться сообщения разной длинны в обе стороны.

При реализации понадобятся криптографические алгоритмы, такие как хеш и цифровая подпись. Их использование и комбинирование и является сутью данной работы.

Первая глава посвящена проблемам безопасного сетевого обмена.

Вторая глава посвящена алгоритмам и протоколам шифрования.

Третья глава посвящена режимам шифрования канала связи.

Четвёртая глава о сравнении режимов шифрования и практических результатах.

Пятая глава об обеспечении качества разработки.

1 Проблемы безопасного сетевого обмена

На данный момент шифрование канала связи актуально. Существует множество готовых реализаций шифрования.

1.1 Актуальность

На данный момент шифрование сетевого обмена касается почти каждого Пользователя ПК. У большинства сайтов есть поддержка HTTPS обеспечивающего шифрование связи клиент-сервер. При общении через Интернет может передаваться конфиденциальная информация, которая интересует злоумышленников. При этом к сетевому трафику уже имеет доступ провайдер. Для того, чтобы никто не смог получить доступ к конфиденциальной информации необходимо использование шифрование.

При передачи важной и конфиденциальной информации обязательно сохранение секретности. Такая информация есть у коммерческих организаций, но и у обычных пользователей существует такая информация. Например, при использовании интернет банкинга, злоумышленники могут изменить номер счёта переводов и украсть средства пользователя. При передачи такой информации необходимо чтобы никто посторонний не мог прочитать или испортить данные.

Из этого следует, что шифрование связи важно, как для пользователей, так и для организаций.

1.2 Существующие решения

Сейчас существует множество решений проблем шифрования. В основном они различаются областью применения.

Самый часто используемый протокол шифрования HTTPS. HTTPS (аббревиатура от англ. HyperText Transfer Protocol Secure) — расширение протокола HTTP для поддержки шифрования в целях повышения безопасности. Данные в протоколе HTTPS передаются поверх криптографических

протоколов SSL или TLS. В отличие от HTTP с TCP-портом 80, для HTTPS по умолчанию используется TCP-порт 443 [3].

Этот протокол используется для того чтобы защитить соединение от просмотра. Также HTTPS позволяет быть уверенным, что соединение происходит именно с сервером. Для обеспечения гарантии соединения именно с сервером требуется чтобы подлинность сервера заверил центр сертификации. Центр сертификации считается абсолютно доверенным.

OpenSSL — полноценная криптографическая библиотека с открытым исходным кодом, широко известна из-за расширения SSL/TLS, используемого в веб-протоколе HTTPS [4]. Библиотека также имеет большое число криптографических функций и может быть использована через консоль. Использование через консоль даёт возможность пользователю проверять файлы и сертификаты на подлинность. Возможно подключение на на C.

Crypto++ - это свободная библиотека криптографических алгоритмов и схем класса C++ с открытым исходным кодом, написанная Wei Dai [5]. Ориентирована на удобное использование C++. Имеет возможности гибкого комбинирования алгоритмов шифрования.

Криптопровайдер (Cryptography Service Provider, CSP) — это независимый модуль, позволяющий осуществлять криптографические операции в операционных системах Microsoft, управление которым происходит с помощью функций CryptoAPI [6]. Ориентирован на Windows. Интерфейс и его использование схож с WinAPI.

OpenSSH (открытая безопасная оболочка) — набор программ, предоставляющих шифрование сеансов связи по компьютерным сетям с использованием протокола SSH [7]. Реализует протокол SSH.

Существует ещё множество различных библиотек и протоколов шифрования со схожими характеристиками.

1.3 Требования к создаваемой библиотеке шифрования

Для создания зашифрованного канала важно чтобы:

– присутствовал механизм аутентификации;

– шифрование было достаточно надёжным;

– невозможность искажения данных и подмены информации;

– достаточно быстрая скорость шифрования.

1.4 Вывод

Тема шифрования сейчас крайне актуальна. Существует множество готовых библиотек шифрования.

2 Алгоритмы и протоколы шифрования

Существует несколько алгоритмов шифровиния, каждый имеет свою область применения.

2.1 Генератор псевдослучайных чисел

В криптографии часто требуется получить случайное число при этом использование стандартных средств получения случайного числа недостаточно.

Предположим при шифровании используется стандартный генератор псевдослучайных чисел - rand. Его инициализация проводится 32 битным значением, поэтому существует только 2^32 вариантов псевдослучайных последовательностей. Их полный перебор вполне выполнимая задача.

Информационная энтропия — мера неопределённости некоторой системы, в частности непредсказуемость появления какого-либо символа первичного алфавита [8].

$$H_2(S) = -\sum p_i log_2 p_i \ (2.1)$$

Значении энтропии множества S вычисляется по формуле 2.1. Выражается в битах. Соответственно у аппаратной реализации энтропия каждого бита равна 1. К сожалению аппаратная реализация случайности обычно отсутствует на ПК.

В компьютере существует множество источников энтропии самый известный из них – время. Текущее время с точностью до миллисекунд, к сожалению, даёт мало энтропии и для получения достаточного количества придётся долго ждать.

Также есть внутренние регистры процессора, аналоговые входы, скорость чтения данных с диска и другие источники.

Для получения случайного числа в Windows можно воспользоваться криптопровайдером. При этом в реализации случайности используется множество источников энтропии:

– ID текущего процесса;

– ID текущей нити исполнения;

– Число тактов с момента последней загрузки;

– Текущее время;

– Различные высокоточные счётчики;

– Хеш-функции MD4 от персональных данных пользователя, таких как логин, имя компьютера, и другие;

– Высокоточные внутрипроцессорные счётчики, такие, как RDTSC, RDMSR, RDPMC.

И другие источники энтропии [9].

2.2 Криптографическая хеш-функция

Криптографические хеш-функции — это выделенный класс хеш-функций, который имеет определенные свойства, делающие его пригодным для использования в криптографии [10].

Требования к криптографической хеш-функции:

1) Отсутствие эффективного способа обратить функцию. Другими словами, зная H(m), невозможно восстановить m. То есть, хеш-функция односторонняя;

2) Невозможно найти коллизию 1 рода. Для некоторого сообщения m нет эффективного алгоритма поиска такого n, что H(m)=H(n);

3) Невозможно найти коллизию 2 рода. Нет эффективного алгоритма поиска таких n и m, что H(m)=H(n).

На 3 пункт существует известная атака «Дней рождения». Её суть заключается в том, что если длинна хеша n, то достаточно 2^(n/2) сообщений и их хеш, чтобы получить коллизию с высокой вероятностью.

Для практического применения придумано много хеш алгоритмов. (описание остальных алгоритмов, например, MD5, и пояснение, почему выдрана именно SHA-3) Для реализации выбран SHA-3.

SHA-3 (Keccak — произносится как «кечак») — алгоритм хеширования переменной разрядности, разработанный группой авторов во главе с Йоаном Дайменом, соавтором Rijndael, автором шифров MMB, SHARK, Noekeon, SQUARE и BaseKing. 2 октября 2012 года Keccak стал победителем конкурса криптографических алгоритмов, проводимым Национальным институтом стандартов и технологий США. 5 августа 2015 года алгоритм утверждён и опубликован в качестве стандарта FIPS 202. Алгоритм SHA-3 построен по принципу криптографической губки (данная структура криптографических алгоритмов была предложена авторами алгоритма Keccak ранее) [11].

Принцип криптографической губки изображён на рисунке 2.1.

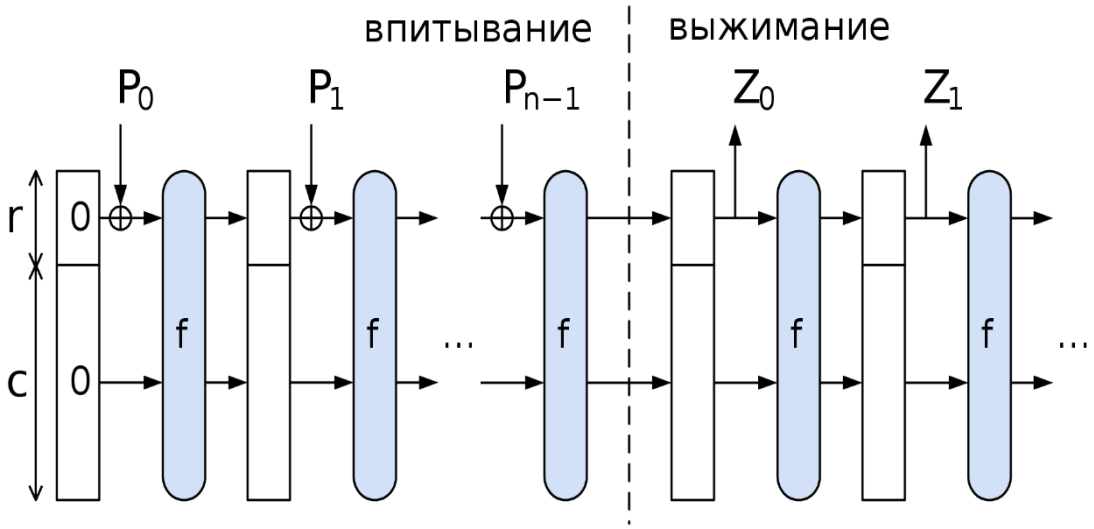


Рисунок 2.1 – Схема криптографической губки

Pn – часть исходного сообщения размером r бит.

Zn - часть результата размером r бит.

В случае SHA-3 512 бит f-это keccakf. И r=576, c=1024.

Keccakf – функция перетасовки битов состояния. Она состоит из 5 шагов, именуемых θ, ρ, π, χ, ι. Они повторяются 24 раза [11].

Обозначим состояние 1600 битов как массив A[5, 5] состоящий из 64 битных элементов. Также три вспомогательных массива B[5,5], C[5], D[5].

n-номер шага

R[5, 5] и RC[24] заранее рассчитанные массивы констант.

Шаг θ.

C[i] = A[i, 0] xor A[i, 1] xor A[i, 2] xor A[i, 3] xor A[i, 4], i=0..4;

D[i] = C[(i+4) mod 5] xor rotateleft(C[(i+1) mod 5],1), i=0..4;

A[i, r] = A[i, r] xor D[i], i=0..4, r=0..4;

Шаг ρ и π.

B[r, (2*i+3*r)] = rotateleft (A[i, r], R[i, r]), i=0..4, r=0..4;

Шаг χ.

A[i, r] = B[i, r] xor (~B[(i+1) mod 5, r] & B[(i+2) mod 5, r]), i=0..4, r=0..4;

Шаг ι.

A[0, 0] = A[0, 0] xor RC[n];

В программной реализации на C++ функция keccakf выглядит так:

```
const uint64_t RC[] = {
    0x0000000000000001L, 0x0000000000008082L, 0x800000000000808aL,
    0x8000000080008000L, 0x000000000000808bL, 0x0000000080000001L,
    0x8000000080008081L, 0x8000000000008009L, 0x000000000000008aL,
    0x0000000000000088L, 0x0000000080008009L, 0x000000008000000aL,
    0x000000008000808bL, 0x800000000000008bL, 0x8000000000008089L,
    0x8000000000008003L, 0x8000000000008002L, 0x8000000000000080L,
    0x000000000000800aL, 0x800000008000000aL, 0x8000000080008081L,
    0x8000000000008080L, 0x0000000080000001L, 0x8000000080008008L
};

const int R[] = {
    0, 1, 62, 28, 27, 36, 44, 6, 55, 20, 3, 10, 43,
    25, 39, 41, 45, 15, 21, 8, 18, 2, 61, 56, 14
};

void keccakf(FState &s) {

    uint64_t B[25];
    uint64_t C[5];
    uint64_t D[5];

    uint64_t *A = s.A;
    for (int n = 0; n < 24; n++) {

        //theta
        for (int i = 0; i < 5; i++)
            C[i] = A[i + 5 * 0] ^ A[i + 5 * 1] ^ A[i + 5 *
            2] ^ A[i + 5 * 3] ^ A[i + 5 * 4];
```

```
        for (int i = 0; i < 5; i++)
            D[i] = C[(i + 4) % 5] ^ rotl(C[(i + 1) % 5], 1);
        for (int i = 0; i < 5; i++)
            for (int r = 0; r < 5; r++)
                A[i + r * 5] ^= D[i];

        //rho pi
        for (int x = 0; x<5; x++)
            for (int y = 0; y<5; y++)
                B[y + ((2 * x + 3 * y) % 5) * 5] = rotl(A[x + y * 5],
                R[x + y * 5]);
        //chi
        for (int x = 0; x<5; x++)
            for (int y = 0; y<5; y++)
                A[x + y * 5] = B[x + y * 5] ^ ((~B[(x + 1) % 5 + y *
                5]) & B[(x + 2) % 5 + y * 5]);


        //ota
        A[0] ^= RC[n];

    }
}
```

Также необходимо учесть, что исходное сообщение дополняется до нужной длинны следующим образом: 0x06, 0x00, …. 0x00, 0x80 или 0x86 если надо дополнить 1 байт. Дополнение происходит в любом случае.

SHA-3 был выбран из-за его относительно простой реализации и при этом высокой стойкостью. Стойкость данного хеш алгоритма обуславливается сразу несколькими причинами:

– наличием запутанного алгоритма перестановки;

– большим размером хеша 512 бит;

– до сих пор не найдено ни 1 коллизии;

У уже взломанных алгоритмов хеширования в основном используются сдвиги и xor.

2.3 Симметричное шифрование

Симметричное шифрование означает, что для шифрования и дешифрования используется один и тот же ключ. Простейший пример такого шифрования – xor сообщения с ключом.

Симметричное шифрование можно разделить на 2 группы: потоковые и блочные шифры.

Блочные шифры работают по принципу шифрования исходного сообщения, разделённого на блоки. Для этого исходное сообщение m делится на блоки mi, размер блока в битах определяется алгоритмом шифрования. При этом последний блок может получится меньшего размера, и его требуется дополнить определённой последовательностью или случайными битами. При блочном шифровании если 2 блока исходного текста одинаковы, то и в зашифрованном виде они будут одинаковыми. Это недочёт системы шифрования который следует учитывать.

Потоковые шифры же зависят не только от ключа, но и от положения части сообщения mi в потоке. Одним из способов потокового шифрования является режим гаммирования [12]. В этом режиме некий генератор создаёт последовательность бит к которой применяется xor с исходным сообщением. Схема режима гаммирования представлена на рисунке 2.2.

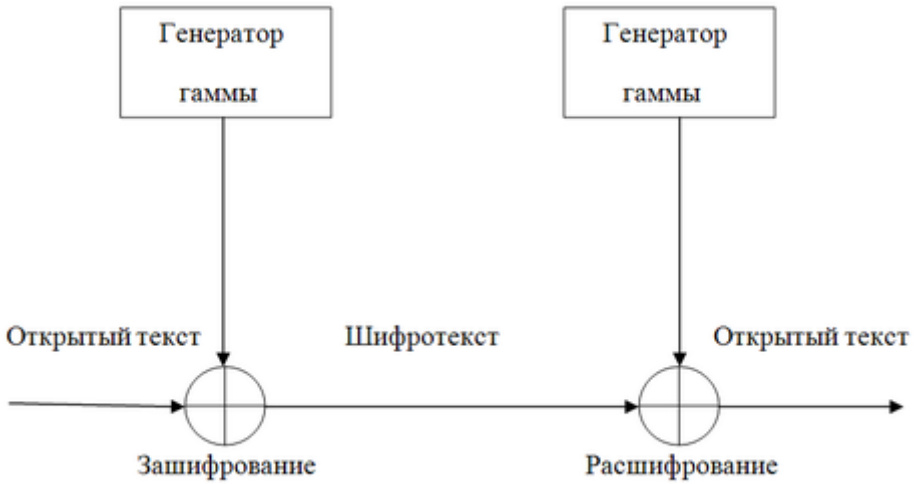


Рисунок 2.2 – Схема режима гаммирования

Для реализации выбран алгоритм IDEA, как довольно популярный.

Также рассматривались алгоритмы: DES, 3DES, AES.

DES уже взломан на данный момент, и поэтому не может быть использован для надёжного шифрования.

3DES называют просто 3 раза шифрование с помошью DES. Такой подход в криптографии считается плохим, и 3DES не рекомендован к использованию.

AES имеет сложное описание, и обеспечивает такое же надёжное шифрование, что и IDEA. При этом имеет теоретическую уязвимость из-за простой математической основы.

IDEA (англ. International Data Encryption Algorithm, международный алгоритм шифрования данных) — симметричный блочный алгоритм шифрования данных, запатентованный швейцарской фирмой Ascom [13].

Размер шифруемого блока 64 бит. Размер ключа 128 бит.

Исходный ключ длинной 128 бит делится на 8 частей по 16 бит.

Далее ключ сдвигается на 25 бит влево. При этом необходимо учесть, что участки по 16 бит стыкуются в обратном порядке. То есть в исходном ключе надо поменять местами участки по 16 бит перед сдвигом. На C++ это выглядит так:

```
v[0] =
    ((v[0] & 0x000000000000ffff) << 48) |
    ((v[0] & 0x00000000ffff0000) << 16) |
    ((v[0] & 0x0000ffff00000000) >> 16) |
    ((v[0] & 0xffff000000000000) >> 48);
v[1] =
    ((v[1] & 0x000000000000ffff) << 48) |
    ((v[1] & 0x00000000ffff0000) << 16) |
    ((v[1] & 0x0000ffff00000000) >> 16) |
    ((v[1] & 0xffff000000000000) >> 48);
```

Сдвиг на 25 повторяется 6 раз, пока не будет сгенерировано 56 16 битных значений, называемых подключами. После этого последние 4 значения отбрасываются и формируются 8 групп по 6 значений и 1 группа из 4.

Далее с помощью операций:

– сложение по модулю 2^16;

– xor;

– умножение по модулю 2^16+1, при этом если исходный аргумент 0, то он заменяется на 2^16.

Исходное 64 битное сообщение делится на 4 части. Потом над ними 8 раз повторяется один и тот же блок операций. Общий вид Алгоритма представлен на рисунке 2.3.

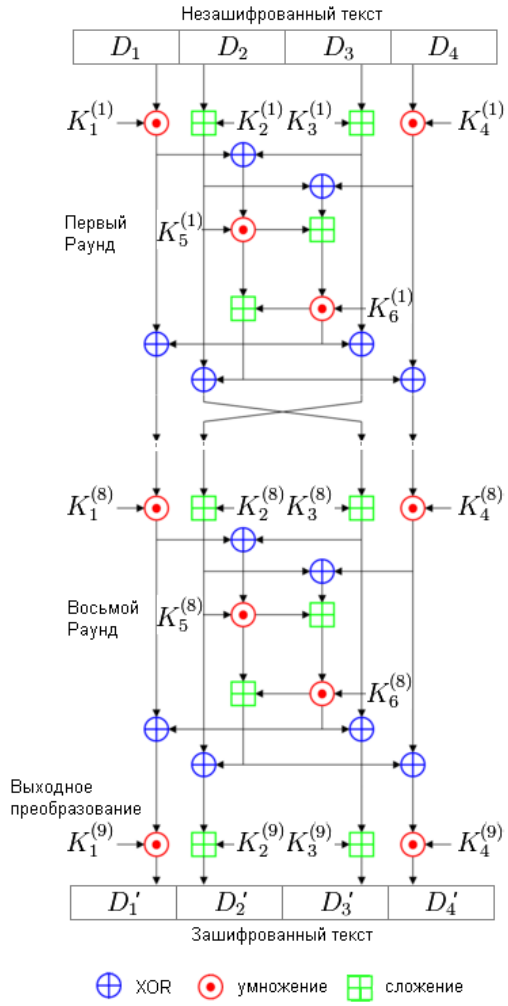


Рисунок 2.3 – Схема алгоритма IDEA

При расшифровке используется тот же алгоритм, но требуется изменить ключи. Изменить их следует как показано на рисунке 2.4.

Деление 1/K обозначает что: (1/k) * k = 1 (mod 2^16+1) при этом обратное к 0 это 0. Вычитание означает что K + (-K) =0 (mod 2^16)

На данный момент известно о наличии 2^53+2^55+2^64 слабых ключей, уязвимых к дифференциальному криптоанализу [13]. Но это не имеет особого значения так как общее число ключей 2^128. Это означает, что шанс выбрать такой ключ примерно 1/2^64.

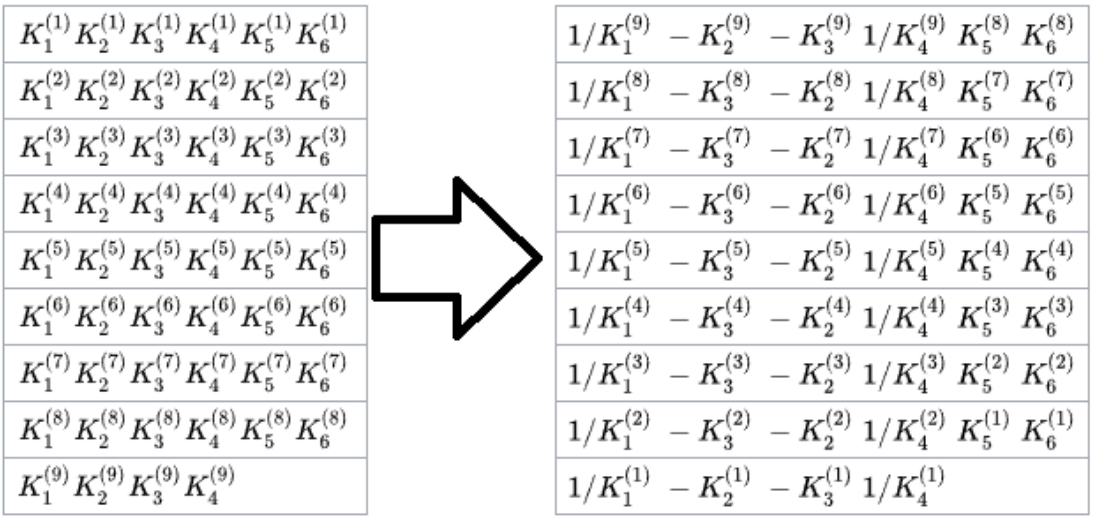


Рисунок 2.4 – Преобразование подключей

Стойкость данного алгоритма обуславливается наличием 3 различных операций с разными модулями.

2.4 Ассиметричное шифрование

Ассиметричное шифрование отличается от симметричного шифрования тем, что для шифрования и дешифровки используются разные ключи, при этом зная открытый ключ, используемый для шифрования, невозможно восстановить закрытый.

Такой вид шифрования может использоваться для цифровой подписи.

Самый известный алгоритм RSA. Его криптографическая стойкость основана на задаче факторизации.

Сначала требуется создать открытый и закрытый ключи. Для этого надо выполнить следующие шаги:

1) Сгенерировать 2 больших простых числа p и q;
2) Вычислить n=p*q;
3) Вычислить функцию Эйлера от n: f(n) = (p-1) * (q-1);
4) Выбрать 1<e<f(n). При этом следует учесть, что малые значения e ослабляют стойкость RSA [14];
5) Вычислить d: d * e = 1 (mod f(n)), это можно сделать с помощью расширенного алгоритма Евклида.

Пара значений (e, n) – открытый ключ.

Пара значений (d, n) – закрытый ключ.

Шифрование производится так: c=m^e mod n.

А дешифрование: m=c^d mod n.

Возведение в степень по модулю производится с помощью алгоритма ускоренного возведения в степень, его реализация на C++ выглядит так:

```
for (int i = b.firstbitpos(); i >= 0; i--) {
    res = (res * res);
    res = res % c;
    if (b.vdata[i / 64] & ((uint64_t)1 << (i % 64))) {
        res = (res * a) % c;
    }
}
```

На текущий момент размер n 1024 бит считается небезопасным, хотя факторизации чисел 1024 бит не опубликовано. Размер n 2048 бит считается условно безопасным. Взломать такой ключ теоретически можно, но это будет стоить около 10 000 000 руб.

2.5 Синхронизация общего ключа

Часто в криптографии возникает задача синхронизации ключа по открытому каналу. При этом важно, чтобы злоумышленник не смог восстановить ключ по перехваченным данным.

Самый популярный алгоритм: алгоритм Диффи — Хелмана.

Протокол Диффи — Хеллмана — криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования. [15]

Алгоритм состоит из 3 действий:

1) Клиент генерирует большое простое число p при этом q=(p-1)/2, тоже должно быть простым. Сгенерировать 1<g<p-2 при этом g^2 != 1 (mod p) и g^q != 1 (mod p). И выбрать случайное a. Далее вычисляется A=g^a mod p. После этого A, g, p посылаются серверу;

2) Сервер генерирует b. Вычисляет B=g^b mod p K=A^b mod p. После этого B посылается клиенту;
3) Клиент вычисляет K=B^a mod p.

2.6 Цифровая подпись

В криптографии бывает нужна так называемая электронная подпись для того, чтобы удостоверится, что сообщение отправлено именно от определённого сервера, а не злоумышленника. Для этого нужно подписать его.

Сейчас подпись происходит следующим образом [1]:

1) Подписывающий берёт сообщение m вычисляет его хеш, с помощью криптографической хеш функции и зашифровывает этот хеш своим закрытым ключом;
2) Проверяющий вычисляет хеш сообщения, и сверяет его с расшифрованным значением.

При этом:

– всегда точно известно точно, что подписывается. Это вытекает из определения хеш функции.

– нельзя использовать одну подпись для другого сообщения. Это вытекает из определения хеш функции.

– требуется чтобы изначально открытый ключ присутствовал у клиента.

2.7 Режим сцепления блоков

Изначально блоковый шифр можно преобразовать в потоковый. Это можно сделать многими способами. Существуют 2 часто используемых варианта: ECB и CBC [1]. Схемы алгоритмов представлены на рисунках 2.5 и 2.6.

ECB проще, но он оставляет информацию об исходном тексте в незашифрованном виде. CBC немного сложнее, и его нельзя распараллелить, но он не оставляет таких явных особенностей исходного текста. Пример шифрования представлен на рисунке 2.7.

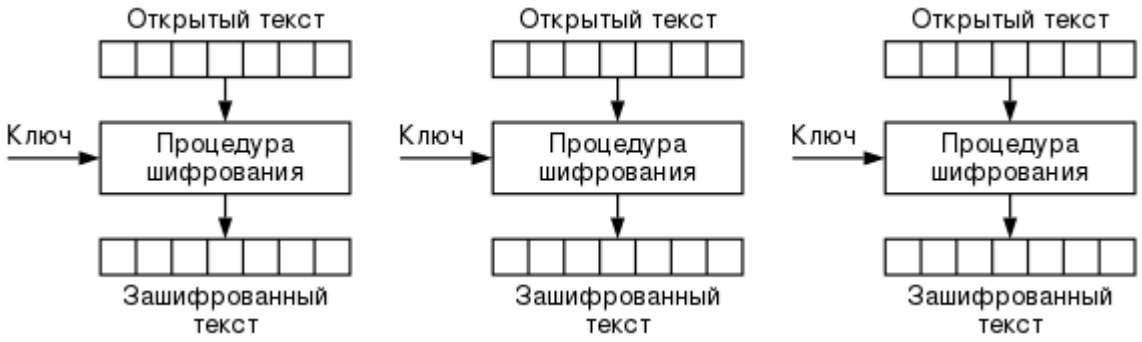


Рисунок 2.5 Схема ECB

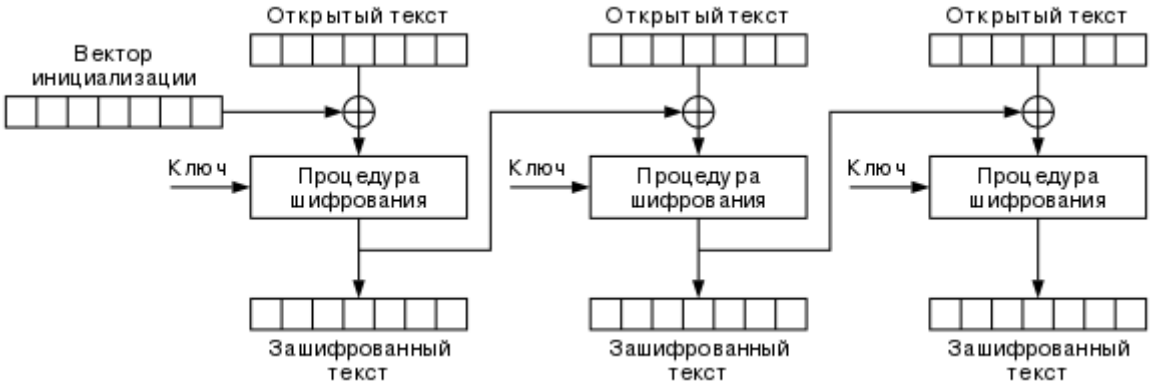
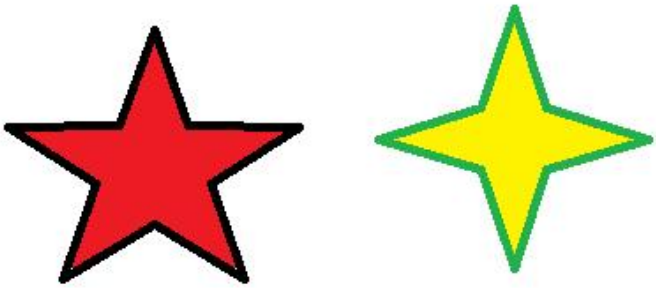
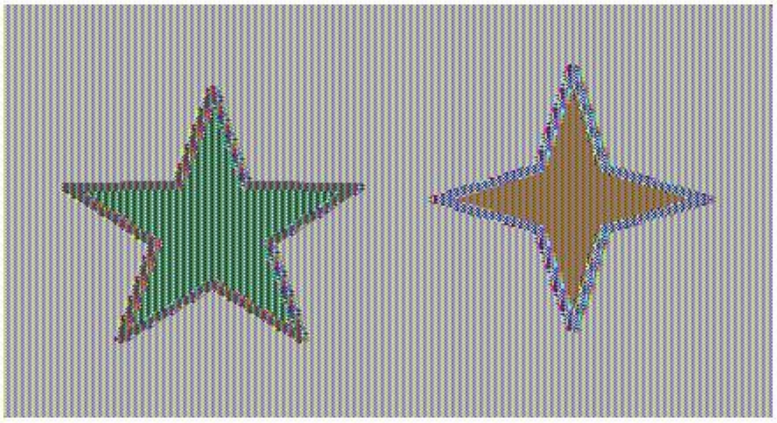


Рисунок 2.6 – Схема CBC

Исходное изображение



ECB



CBC

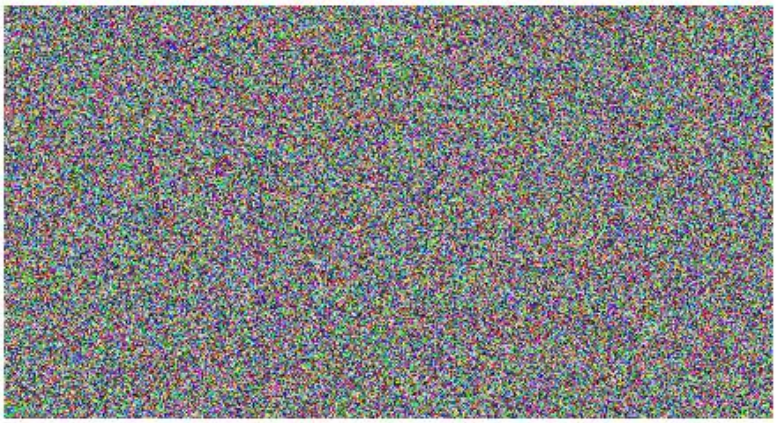


Рисунок 2.7 – Пример шифрования

2.8 Длинная алгебра

При шифровании часто возникает задача оперирования числами размером в сотни бит. Так как ПК имеет разрядность процессора 64, возникает задача оперирования числами большой длинны.

Так как процессор не поддерживает операции над такими числами, эти операции требуется реализовывать самому. В стандартных библиотеках C++ нет реализации длинной алгебры. Так как эта библиотека требует полного доверия, её лучше всего реализовать самому, а не брать готовую реализацию.

При реализации сначала требуется выбрать, как хранить числа в памяти. Для этого сначала надо выбрать систему счисления. Три основных варианта выбора:

– 10 самая понятная человеку система счисления;

– 10^19 также, как и 10 проста для понимания, но работает гораздо быстрее из-за большего основания;

– 2^64 сложна для понимания, и соответственно требует большего времени на разработку, однако при таком выборе не требуется делать остаток по модулю, его за нас сделает процессор автоматически. Поэтому такая система счисления гораздо быстрее других.

Был выбран 3 вариант, как самый быстрый.

Знак числа решено хранить отдельно. А для хранения самого числа использовать динамический массив. Реализация хранения на C++ выглядит так:

```
vector<uint64_t> vdata;
int sign;
```

Далее требуется реализовать операции: +, -, *, /, powmod.

Плюс и минус реализуются через цикл по алгоритму сложения в столбик. Реализация плюса на C++ выглядит так:

```
void bigint::opadd(const bigint &a)
{
```

```
    bool c = 0;
    for (int i = 0; i < a.vdata.size(); i++) {
        vdata[i] = addcarry(vdata[i], a.vdata[i], c);
    }
    size_t i = a.vdata.size();
    while (c) {
        vdata[i] = addcarry(vdata[i], 0, c);
        i++;
    }
}
```

Функция addcarry обозначает ассемблерную функцию процессора, именно из-за возможности использования таких функций напрямую и была выбрана такая система счисления.

Реализация умножения также идёт по алгоритму умножения в столбик, её реализация на C++ выглядит так:

```
bigint operator*(const bigint &a, const bigint &b)
{
    bigint res;
    auto N = a.vdata.size() + b.vdata.size();
    res.vdata.resize(N);
    res.sign = a.sign*b.sign;
    for (int i = 0; i < a.vdata.size(); i++) {
        bigint p;
        p.vdata.resize(N);

        for (int r = 0; r < b.vdata.size(); r += 2) {
            mulbig(a.vdata[i], b.vdata[r], p.vdata[i + r],
            p.vdata[i + r + 1]);
        }
        res.opadd(p);
        p.vdata.resize(0);
        p.vdata.resize(N);
        for (int r = 1; r < b.vdata.size(); r += 2) {
            mulbig(a.vdata[i], b.vdata[r], p.vdata[i + r],
            p.vdata[i + r + 1]);
        }
        res.opadd(p);
    }
    res.fit();
    return res;
}
```

При этом умножение на одну цифру идёт в 2 этапа: сначала нечётные цифры умножаются на неё, а потом чётные. Это позволяет не заниматься переводом разряда при каждом умножении на цифру.

Деление же происходит не в 2^64 системе счисления, а в двоичной. Алгоритм же используется деление треугольником. При использовании двоичной системы счисления не требуется подбирать цифру для умножения, как в десятичной, что ускоряет процесс деления. В результате получается частное и остаток. Реализация на C++ выглядит так:

```
bigint operator/(const bigint &a, const bigint &b)
{
    bigint resdiv;
    int numdiff = a.firstbitpos() - b.firstbitpos();
    //std::cout << "a:" << a << std::endl;
    //std::cout << "b:" << b << std::endl;
    //std::cout << "af:" << a.firstbitpos() << std::endl;
    //std::cout << "bf:" << b.firstbitpos() << std::endl;
    if (numdiff<0)
        return resdiv;
    bigint bmovv = b << numdiff;
    bigint resmod = a;
    //std::cout << "bmovv st:" << bmovv << std::endl;
    resdiv.vdata.resize(a.vdata.size(), 0);
    for (int i = numdiff; i >= 0; i--) {
        //std::cout << "resmod:" << resmod << std::endl;
        //std::cout << "bmovv :" << bmovv << std::endl;
        //std::cout << "resdiv:" << resdiv << std::endl;
        if (resmod.opbigger(bmovv) != -1) {
            resmod.opsub(bmovv);
            resdiv.vdata[i / 64] |= (uint64_t)1 << (i % 64);
        }
        bmovv.oprr();

    }

    resdiv.fit();
    resmod.fit();
    resdiv.sign = a.sign*b.sign;
    return resdiv;
}
```

Powmod(возведение в степень по модулю) требуется для криптографии, его следует реализовывать через аглоритм быстрого возведения в степень. Суть этого алгоритма в том, что берётся исходная степень и представляется в двоичной системе счисления. При этом возможно брать промежуточные результаты также по модулю. Пример:

7^11 mod 13 = 2

$11=1011_2$

1) 1*1 mod 13=1; 1*7 mod 13=7

2) 7*7 mod 13=10;

3) 10*10 mod 13=9; 9*7 mod 13=11

4) 11*11 mod 13=4; 11*7 mod 13=2

Реализация на C++ выглядит так:

```
bigint powmod(const bigint &a, const bigint &b, const bigint &c)
{

    bigint res(1);

    for (int i = b.firstbitpos(); i >= 0; i--) {
        res = (res * res);
        res = res % c;
        if (b.vdata[i / 64] & ((uint64_t)1 << (i % 64))) {
            res = (res * a) % c;
        }
    }

    return res;
}
```

Также для шифрования требуется генерировать большие простые числа. Для этого сначала берётся случайное число, затем к нему добавляется 1 пока не получится простое число, при этом требуется тест простоты, вероятностный алгоритм говорящий, что число не составное с некой вероятностью.

Для реализации взят тест Миллера-Рабина. Псевдокод этого алгоритма: Ввод: n > 2, нечётное натуральное число, которое необходимо проверить на простоту;

k — количество раундов.

Вывод: составное, означает, что n является составным числом;

вероятно простое, означает, что n с высокой вероятностью является простым числом. Алгоритм [16]:

Представить n − 1 в виде 2^s·t, где t нечётно, можно сделать

последовательным делением n - 1 на 2.

цикл A: повторить k раз:

Выбрать случайное целое число a в отрезке [2, n − 2]

x ← a^t mod n, вычисляется с помощью алгоритма возведения в степень по модулю

если x = 1 или x = n − 1, то перейти на следующую итерацию цикла A

цикл B: повторить s − 1 раз

x ← x^2 mod n

если x = 1, то вернуть составное

если x = n − 1, то перейти на следующую итерацию цикла A

вернуть составное

вернуть вероятно простое

Реализация на C++ выглядит так:

```
bool testp(const bigint &n,int rounds = 100)
{

    int res = 1;
    bigint b = n - 1;
    int s=b.lastbitpos();
    bigint t = b >> s;
    for (int i = 0; i < rounds && res; i++) {
        bigint a,x;
        a.vdata.resize(n.vdata.size()+1);
        for (auto &e : a.vdata)
            e = rand() + ((uint64_t)rand() << 16) + ((uint64_t)rand()
            << 32) + ((uint64_t)rand() << 48);
        a = (a % (n - 3)) + 2;

        x = powmod(a, t, n);

        if (x != 1 && x != n - 1) {
            int f = 0;
            for (int r = 0; r < s - 1 && f==0; r++) {
                x = x * x % n;
                if (x == 1)
                    res = 0;
                if (x == n - 1)
                    f = 1;
            }
            if (f == 0)
                res = 0;

        }
```

```
    }

    return res;
}
```

Также для ускорения проверки сначала проверяется число на делении на первые 100 простых чисел. При такой проверке сначала рассчитываются остатки от деления с помощью длинной алгебры. Потом к ним прибавляется по 1 вместе с увеличением числа, это позволяет значительно ускорить поиск простого числа. Реализация ускоренного поиска на C++ выглядит так:

```
bigint nextPrime(bigint num) {

    bigint p1 = num + 1;

    const static int osn[100] = {
    2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,8
    9,97,101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,1
    79,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269
    ,271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,3
    73,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463
    ,467,479,487,491,499,503,509,521,523,541 };

    int val[100];
    int L = 100;
    for (int i = 0; i < L; i++) {
        val[i] = static_cast<int>((p1%osn[i]).getuint64_t());
        if (num < osn[i])
            return osn[i];
    }

    while (1) {
        int inc = 0;

        int t1 = 0;
        while (!t1) {
            t1 = 1;
            inc++;
            for (int i = 0; i < L; i++) {
                val[i] = (val[i] + 1) % osn[i];
                if (val[i] == 0)
                    t1 = 0;
            }
        }
        p1 = p1 + inc;
        if (testp(p1))
            return p1;
    }
```

```
}
```

2.9 Вывод

В криптографии существует множество инструментов, а именно: ассиметричный шифр, симметричный шифр, подпись, хеш, синхронизация ключа, и другие. Каждый занимает свою нишу и обеспечивает определённую функцию.

3 Режимы шифрования канала

Существует несколько вариантов реализации шифрованного канала, которые стоит рассмотреть.

3.1 По заранее согласованному ключу

Если ключ заранее согласован, то можно применить симметричный алгоритм, или же ассиметричный при использовании исходного ключа для генерации ключей ассиметричного шифрования.

Если есть возможность передать ключ заранее, то становится возможным использовать шифр Вернама. Суть этого шифра в том, что исходный случайный ключ имеет тот же размер, что и сообщение. Производится xor с сообщением и такое послание невозможно взломать никаким способом (атаки по сторонним каналам не рассматриваются) [1].

3.2 С помощью ассиметричного шифрования

Канал связи также можно зашифровать с помощью ассиметричного шифрования, для этого надо иметь открытые ключи друг друга. У такого шифрования теоретически большая надёжность, чем у симметричного.

3.3 Гибридный метод

В современном шифровании чаще всего используется гибридный метод шифрования. Например, в TLS. Суть гибридного метода состоит в том, чтобы сначала согласовать ключ для ассиметричного шифрования, а потом производить шифрование с помощью симметричного алгоритма.

Самый простой вариант это: согласовать ключ с помощью алгоритма Диффи — Хеллмана, а потом начать шифрование с помощью сгенерированного ключа. Но такой вариант имеет существенную уязвимость. Она заключается в том, что при связи A с B злоумышленник может представиться A как B, а B как A. Такая атака называется атакой посредника.

Есть 2 метода защиты от такой атаки:

1) Сделать невозможным посредничество, например, измерять скорость передачи по прямому соединению и при её понижении, при появлении посредника отключать соединение;
2) Добавить систему аутентификации, чтобы точно было известно, что соединение прошло именно с сервером, а не с посредником.

В протоколе Диффи-Хелмана достаточно, чтобы ответ от сервера имел подпись. Предположим, злоумышленник, понимая невозможность подделки подписи, решил использовать старое сообщение от сервера. Но при этом значения g и a различны и ответ сервера в виде B не совпадёт. Если же попытаться заставить сервер подписать нужное B то столкнёмся с тем, что сервер выбирает случайное b и таким образом можно заставить подписать только 1 и -1, что бесполезно для взлома.

Однако при этом возможно так, что после согласования получились разные ключи для симметричного шифрования. Требуется добавить алгоритм проверки шифрования перед началом связи

Также стоит учесть возможность искажения данных в симметричном шифровании. Так как в канале предполагается использование сообщений произвольной длинны, требуется добавить в поток размер и хеш сообщения, чтобы нельзя было исказить его.

В качестве алгоритма симметричного шифрования возьмём блочный шифр в режиме CBC.

Для этого решено добавить 3 64 битных значения: 2 с длинной сообщения и 1 с его хешом. 2 части с длинной призваны обеспечить защиту от полного изменения всего потока, в таком случае они не совпадут и не придётся дальше читать поток.

Основа реализации этого алгоритма: протокол Диффи — Хеллмана, SHA3, RSA, IDEA, криптопровайдер, WinSocket 2.

3.4 Вывод

Существует несколько алгоритмов шифрования каналов. Они отличаются сложностью.

4 Сравнение режимов

У 3 реализаций есть свои плюсы и минусы, которые и влияют на конечный выбор.

4.1 Практические результаты применения

В результате рассмотрения теоретических выкладок для рассмотрения выбраны и реализованы 3 варианта обеспечения канала связи:

1) Симметричное шифрование на основе XOR (алгоритм Вермана);
2) Потом шифрование на основе RSA;
3) Гибридный.

Эти реализации прошли тест на шифрование 167 МБ файла.

Время, которое потребовалось на шифрование:

1) 1197 мс (могло быть гораздо быстрее, но система рассчитана на то, что шифрование будет происходить значительное время);
2) 1400 мс на 8*31 байт при ключе 2048 бит на шифрование всего файла понадобилось бы 167*1024*1024/(8*31)*1400=988536567 мс;
3) 7407 мс + инициализация канала (~2000 мс без генерации P для Диффи-Хелмана).

При первом и третьем варианте скорость шифрования гораздо выше, чем скорость стандартной сетевой карты (100 Мбит/с). Значит при шифровании будет использоваться полная мощность соединения.

4.2 Плюсы и минусы

Первый вариант самый простой в программной реализации, но для него требуется генерация случайных бит на аппаратном уровне и передача ключа на другую сторону в виде жёсткого диска. При этом реализуется связь только между 2 абонентами. Если, например, требуется наладить связь между центральным банком и филиалами, то такой вариант вполне применим. Но для структуры клиент-сервер такой подход не применим. Главный плюс – это абсолютная криптостойкость. Взломать такую связь невозможно.

Второй вариант слишком медленный для практической реализации и может подойти только для обмена текстовыми сообщениями в чате. Не требует особых ухищрений при программировании. При этом не имеет обнаруженных дыр в криптостойкости.

Третий вариант самый сложный в реализации, и при этом самый оптимальный. Высокая скорость и надёжность. По сути это идеальный алгоритм шифрования на данный момент. В этом варианте комбинируются 4 алгоритма и если хоть один из них будет взломан, то соединение сразу станет уязвимым. Как и во 2 варианте требуется заранее узнать открытый ключ сервера, для этого обычно используют открытый канал и центр сертификации. Из-за комбинирования методов криптостойкость системы равняется криптостойкости самого слабого алгоритма.

4.3 Вывод

В результате к реализации выбрано 3 варианта реализации канала связи. Для практического применения лучшим вариантом признан гибридный вариант.

5 Обеспечение качества разработки

Библиотека шифрования, как и любой продукт требует оценки её качества.

5.1 Понятие качества

Качество, имеет много определений, отличающихся областью применения: качество продукции, качество услуг, качество шифрования. В общем виде качество – это совокупность свойств услуги/товара, отвечающая за его пригодность.

Качество является субъективным, так как оценку качества даёт потребитель товара, и для определения качества надо определить, что требует потребитель.

В стандарте ГОСТ Р ИСО 9000-2015 [17]: «Качество — степень соответствия совокупности присущих характеристик объекта требованиям». Характеристика – отличительное свойство. Требование - потребность или ожидание, которое установлено, обычно предполагается или является обязательным.

Операциональное определение или рабочее определение — описание явления (переменной, термина, объекта и т. д.) в терминах операций (процесса), которые необходимо произвести для подтверждения наличия явления, измерения его продолжительности и величины.

Такое определение требуется, для корректного понимание потребителя и производителя, ведь потребитель, говоря: “Быстро” может понимать минуту, час или день.

5.2 Расчёт критериев качества

Для разработанной библиотеки шифрования, потребителями являются программисты. При использовании библиотеки потребитель предъявляет следующие требования: надёжность, расширяемость.

Разработка этих 2 операциональных определений представлена в таблице 1.

Таблица 1 – Операциональные определения

Требование	Тест	Анализ
Аутентификация сервера. Надёжность шифрования данных. Невозможность изменения данных.	Расчёт вероятности подбора данных. Расчёт сложности взлома по самому лучшему алгоритму взлома.	Шанс случайного неправильного определения должен быть меньше 1/10^9. Взлом алгоритма аутентификации теоретически должен занимать годы работы суперкомпьютера. Шанс случайного дешифрования должен быть меньше 1/10^9. Взлом алгоритма шифрования теоретически должен занимать годы работы суперкомпьютера. Шанс принять изменённые данные за исходные должен быть меньше 1/10^9.
Возможность добавления новых алгоритмов.	Наличие абстрактных классов примитивов шифрования.	Схема реализации должна содержать абстрактные классы и использовать их при шифровании.

Алгоритмов поиска коллизии на хеш функцию SHA3 не найден, единственный известный алгоритм - перебор всех вариантов исходных сообщений. Размер генерируемого хеша 512 бит. Такой перебор имеет шанс 1/2^512 быть успешным. Даже при использовании суперкомпьютера потребуются десятилетия расчётов.

При использовании RSA 2048 бит, условно взлом требует нескольких лет. Шанс случайно угадать нужное значение 1/2^2048.

Из этого следует, что аутентификация удовлетворяет критериям качества.

Шифрование связи производится с помощью IDEA. Единственный известный алгоритм взлома — перебор. Учитывая, что размер ключа 128 бит, а у CBC используется случайный вектор инициализации, требуется перебрать 2^(128+64) вариантов, что сделать даже за несколько лет невозможно.

Алгоритм Диффи — Хеллмана при использовании простого числа размером 2048 бит имеет туже сложность взлома, что и RSA [1].

Из этого следует, что шифрование удовлетворяет критериям качества.

Для сохранения целостности данных существует повторение размера пакета, и хеш сообщения. Оба они имеют размер 64 бит. Из этого следует, что шанс угадать нужные значения: 1/2^128. Учитывая то, что после неудачной попытки изменения данных соединение обрывается, практически исказить данные невозможно.

Из этого следует, что сохранение целостности удовлетворяет критериям качества.

Изначально вся библиотека основана на абстрактных классах примитивов шифрования. Сам класс соединения использует их.

Из этого следует, что аутентификация удовлетворяет критериям качества.

5.3 Вывод

Разработанная библиотека удовлетворяет поставленным критериям качества. Шансы случайного подбора правильных исходных данных далеки от пороговых.

ЗАКЛЮЧЕНИЕ

Реализована библиотека шифрования, с использованием современных алгоритмов шифрования, таких как RSA, SHA3 и другие. Главная функция данной библиотеки, обеспечение защищённого канала связи в сети интернет. Реализация соединения по сети использует TCP/IP.

Главное отличие от других библиотек – малое количество исходного кода. Это узкоспециализированная библиотека, направленная только на создание канала связи. Её вполне можно прочитать за несколько часов и убедиться, что закладок для взлома нет.

Созданная библиотека на C++ является рабочей и может использоваться для шифрования. В дальнейшем она может использоваться шифрования не только канала связи, но и файлов.

Одно из направлений развития библиотеки, создание защищённой системы обмена сообщениями, где пользователи передают друг-другу сообщения, не раскрывая их никому, даже серверам, обеспечивающим связь. Такая система не выдаст конфиденциальной информации, даже если сервер решит выдать персональные данные пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рябко, Б. Я. Криптография в информационном мире / Б. Я. Рябко, А. Н. Фионов. – Москва : Горячая линия – Телеком, 2018. – 300 с.
2. A kilobit hidden SNFS discrete logarithm computation. – https://eprint.iacr.org/2016/961.pdf (дата обращения 30.05.2019).
3. HTTPS – Википедия. — https://ru.wikipedia.org/wiki/HTTPS (дата обращения 23.05.2019).
4. OpenSSL – Википедия. — https://ru.wikipedia.org/wiki/OpenSSL (дата обращения 23.05.2019).
5. Crypto++ – Википедия. — https://en.wikipedia.org/wiki/Crypto%2B%2B (дата обращения 23.05.2019).
6. Криптопровайдер – Википедия. – https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%BF%D1%80%D0%BE%D0%B2%D0%B0%D0%B9%D0%B4%D0%B5%D1%80 (дата обращения 23.05.2019).
7. OpenSSH – Википедия. – https://ru.wikipedia.org/wiki/OpenSSH (дата обращения 24.05.2019).
8. Информационная энтропия. – https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%8D%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D1%8F (дата обращения 25.05.2019).
9. CryptGenRandom – Википедия. – https://ru.wikipedia.org/wiki/CryptGenRandom (дата обращения 25.05.2019).
10. Криптографическая хеш-функция – Википедия. – https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%85%D0%B5%D1%88-

%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F (дата обращения 25.05.2019).

11. SHA-3 – Википедия. – https://ru.wikipedia.org/wiki/SHA-3 (дата обращения 25.05.2019).
12. Потоковый шифр — Википедия. – https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%B2%D1%8B%D0%B9_%D1%88%D0%B8%D1%84%D1%80 (дата обращения 25.05.2019).
13. IDEA – Википедия. – https://ru.wikipedia.org/wiki/IDEA (дата обращения 25.05.2019).
14. RSA – Википедия. – https://ru.wikipedia.org/wiki/RSA (дата обращения 26.05.2019).
15. Протокол Диффи — Хеллмана – Википедия https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%94%D0%B8%D1%84%D1%84%D0%B8_%E2%80%94_%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0 (дата обращения 26.05.2019).
16. Тест Миллера-Рабина – Википедия https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82_%D0%9C%D0%B8%D0%BB%D0%BB%D0%B5%D1%80%D0%B0_%E2%80%94_%D0%A0%D0%B0%D0%B1%D0%B8%D0%BD%D0%B0 (дата обращения 28.05.2019).
17. ГОСТ_Р_ИСО_9000—2015 https://ru.wikisource.org/wiki/%D0%93%D0%9E%D0%A1%D0%A2_%D0%A0_%D0%98%D0%A1%D0%9E_9000%E2%80%942015 (дата обращения 27.05.2019).

ПРИЛОЖЕНИЕ А

Исходные коды программы

Файл NCryptBase.h

```
// Базовые абстрактные классы.

#pragma once

#include <cstdint>
#include <vector>
#include <memory>
#include "bigint.h"

// Абстрактный базовый класс генератора случайных чисел
class NRandomGen {
public:
     // Генерация случайного числа
     virtual bool GetRandom(void *, int64_t) = 0;
};

// Абстрактный класс симметричного шифрования
class NCryptSymmetric {
public:
     // Получить шаг данных в байтах
     virtual const uint64_t GetDataScale() = 0;

     // Получить размер ключа данных в байтах
     virtual const uint64_t GetKeySize() = 0;

     // Установить ключ
     virtual bool SetKey(std::vector<uint64_t>) = 0;

     // Зашифровать
     virtual bool Encode(void*, void*, uint64_t) = 0;

     // Расшифровать
     virtual bool Decode(void*, void*, uint64_t) = 0;

     virtual __declspec(dllexport)  ~NCryptSymmetric();
};

// Абстрактный класс ассимметричного шифрования
class NCryptAsymmetric {
public:
     // Получить шаг данных в байтах
     virtual const uint64_t GetDataScale() = 0;

     // Установить приватный ключ
     virtual bool SetPrivateKey(std::vector<uint64_t>) = 0;

     // Установить публичный ключ
     virtual bool SetPublicKey(std::vector<uint64_t>) = 0;

     // Зашифровать
     virtual bool Crypt(void*, uint64_t, void*, uint64_t) = 0;

     // Расшифровать
```

```
    virtual bool Encrypt(void*, uint64_t) = 0;

    virtual __declspec(dllexport)  ~NCryptAsymmetric();
};

// Абстрактный класс аутентификации(подписи)
class NCryptAutentification {
public:
    // Максимальный размер подписываемого сообщения
    virtual int MaxMSize() = 0;

    // Установить приватный ключ
    virtual bool SetPrivate(std::vector<uint64_t>) = 0;

    // Установить публичный ключ
    virtual bool SetToken(std::vector<uint64_t>) = 0;

    // Сравнение зашифрованного хеша с хешом сообщения
    virtual bool Test(std::vector<uint64_t> ,std::vector<uint64_t>) = 0;

    // Генерирование ключей
    virtual bool Generate(int64_t, std::vector<uint64_t>&, std::vector<uint64_t>&,
    std::shared_ptr<NRandomGen>) = 0;

    // Шифрование хеша
    virtual bool Crypt(std::vector<uint64_t>,std::vector<uint64_t> &) = 0;

    virtual __declspec(dllexport)  ~NCryptAutentification();
};

// Абстрактный класс хеш функции
class NCryptHash {
public:
    // Хеш от данных
    virtual bool Hash(void*, uint64_t, std::vector<uint64_t>&) = 0;

    // Получение размера хеша
    virtual int HashSize() = 0;

    virtual __declspec(dllexport) ~NCryptHash();
};

// Абстрактный класс синхронизации общего секрета
class NCryptSync {
public:
    // Выполнение шага для первого участника
    virtual int step1(const vector<uint64_t> &, vector<uint64_t> &, int,
    std::shared_ptr<NRandomGen>rg) = 0;

    // Выполнение шага для второго участника
    virtual int step2(const vector<uint64_t> &, vector<uint64_t> &, int,
    std::shared_ptr<NRandomGen>rg) = 0;

    // Размер входных данных для шага
    virtual int stepsize(int type, int step) = 0;

    // Получение общего секрета
    virtual vector<uint64_t> getResult() = 0;

    // Количество случайных бит в секрете
    virtual int randomBitsCount() = 0;
```

```
    virtual __declspec(dllexport) ~NCryptSync();
};

// Абстрактный класс незащищённого соединения
class NCryptConnection {
public:
    // Соединение с сервером
    virtual bool connectToServer(const char* ip, uint16_t port) = 0;

    // Ожидание клиентов
    virtual bool listenClients(uint16_t port) = 0;

    // Посылка данных
    virtual bool sendV(void*, uint64_t) = 0;

    // Приём данных
    virtual bool recvV(void*, uint64_t) = 0;

    // Обрыв соединения
    virtual void disconnectConn() = 0;

    // Генерация случайного числа
    virtual __declspec(dllexport) ~NCryptConnection();
};


#ifdef _WIN64
// Генератор случайных байтов
class NRandomGenWin :public NRandomGen {
    __declspec(dllexport) bool GetRandom(void *, int64_t);
};
#define NBASICRG std::shared_ptr<NRandomGen>(static_cast<NRandomGen*>(new NRan-
domGenWin()))
#endif


// Генератор случайного числа заданного размера
extern __declspec(dllexport) bigint randomNumberGen(std::shared_ptr<NRandomGen> rg,
int size, int addbit = 1);

// Генератор случайного простого числа заданного размера
extern __declspec(dllexport) bigint randomPrime(int size,
std::shared_ptr<NRandomGen>rg);

// Генератор числа p заданного размера такого, что p и (p-1)/2 простые
extern __declspec(dllexport) bigint randomExtraPrime(int size,
std::shared_ptr<NRandomGen>rg);
```

Файл NCryptBase.cpp

```
#include "NCryptBase.h"

NCryptSymmetric::~NCryptSymmetric() {

}

NCryptAsymmetric::~NCryptAsymmetric() {
```

```
}

NCryptAutentification::~NCryptAutentification() {

}

NCryptHash::~NCryptHash(){

}

#ifdef _WIN64
#include <winsock2.h>
#include <windows.h>
#undef max
#undef min
#include <bcrypt.h>
#pragma comment( lib, "Bcrypt.lib" )

#include <winternl.h>

bool NRandomGenWin::GetRandom(void *data, int64_t size)
{
     NTSTATUS    Status;

     Status = BCryptGenRandom(
          NULL,
          (PUCHAR)data,
          static_cast<ULONG>(size),
          BCRYPT_USE_SYSTEM_PREFERRED_RNG);

     if (!NT_SUCCESS(Status))
     {
          return false;
     }
     return true;
}
#endif

bigint randomNumberGen(std::shared_ptr<NRandomGen> rg,int size, int addbit) {
     if (size < 1)
          throw;
     if (size == 1) {
          if (addbit)
               return 1;
          char ch;
          if (!rg->GetRandom(&ch, 1))
               throw;
          return ch % 2;
     }

     vector<uint64_t> arr;
     arr.resize((size - 1) / 64 + 1, 0);

     if (!rg->GetRandom(arr.data(), arr.size() * sizeof(uint64_t)))
          throw;

     if (addbit)
          arr[(size - 1) / 64] |= (uint64_t)1 << ((size - 1) % 64);

     if (size % 64 != 0)
```

```
          arr[(size - 1) / 64] &= (((uint64_t)1 << (size - (size - 1) / 64 * 64)) -
          1);

     return bigint(arr,1);
}

bigint randomPrime(int size, std::shared_ptr<NRandomGen> rg) {

     bigint res = nextPrime(randomNumberGen(rg,size));

     if (
          res.data().size() > (size - 1) / 64 + 1 ||
          ((size % 64 != 0) && res.data()[(size - 1) / 64]>((uint64_t)1 << ((size) %
          64)))
          ) {
          res = nextPrime(res >> 1);
     }
     return res;
}

bigint randomExtraPrime(int size, std::shared_ptr<NRandomGen> rg)
{
     bigint res = nextPrimeEX(randomNumberGen(rg, size));

     if (
          res.data().size() > (size - 1) / 64 + 1 ||
          ((size % 64 != 0) && res.data()[(size - 1) / 64] > ((uint64_t)1 << ((size)
          % 64)))
          ) {
          res = nextPrimeEX(res >> 1);
     }


     return res;
}

NCryptConnection::~NCryptConnection()
{
}

NCryptSync::~NCryptSync()
{
}
```

Файл NCryptChannel.h

```
// Шифрованный канал связи

#pragma once
#include "NCryptBase.h"

// Структура с ссылками на классы реализации алгоритмов
class NCryptSTR {
public:
     std::shared_ptr<NCryptAutentification> auth;
     std::shared_ptr<NCryptHash> hash;
     std::shared_ptr<NCryptSync> sync;
     std::shared_ptr<NCryptSymmetric> symm;
     std::shared_ptr<NCryptConnection> connect;
     std::shared_ptr<NRandomGen> rg;
```

```
};

// Инверсия порядка битов в байте
uint8_t reverseBits(uint8_t a) {
     // 76543210
     // 6 4 2 0
     //  7 5 3 1
     // 67452301
     //   67  23
     // 45  01
     // 45670123
     // 0123
     //     4561
     // 01234567
     a = (a & 0x55) << 1 | (a & 0xAA) >> 1;
     a = (a & 0x33) << 2 | (a & 0xCC) >> 2;
     a = (a & 0x0F) << 4 | (a & 0xF0) >> 4;
     return a;
}

// Класс шифрованного канала связи
class NCryptChannel {
private:
     NCryptSTR crypt;
     vector<uint8_t> symmChainData;
     vector<uint8_t> symmChainDataDecode;
     int myType = 0;

     // Шифрование
     void SCrypt(void *data, int size);

     // Дешифровка
     void SEncrypt(void *data, int size);

public:

     // Конструктор(инициализация канала)
     __declspec(dllexport) NCryptChannel(NCryptSTR &s);

     // Соединение с сервером
     __declspec(dllexport) bool Connect(std::string ip, int port);

     // Ожидание клиентов
     __declspec(dllexport) bool Listen(int port);

     // Синхронизации общего секрета
     __declspec(dllexport) bool synckey();

     // Посылка пакета данных
     __declspec(dllexport) bool senddata(const void *data, int size);

     // Приём пакета данных
     __declspec(dllexport) bool recvdata(vector<uint8_t> &vec);
};
```

Файл NCryptChannel.cpp

```
#include "NCryptChannel.h"
```

```
inline NCryptChannel::NCryptChannel(NCryptSTR & s) {
     crypt = s;
}

inline bool NCryptChannel::Connect(std::string ip, int port) {
     bool b;
     b = crypt.connect->connectToServer(ip.c_str(), port);
     std::cout << "CH conn:" << b << std::endl;
     if (!b)
          return 0;
     myType = 1;
     return 1;
}

inline bool NCryptChannel::Listen(int port) {
     bool b;
     b = crypt.connect->listenClients(port);
     std::cout << "CH listen:" << b << std::endl;
     if (!b)
          return 0;
     myType = 2;
     return 1;
}

inline bool NCryptChannel::synckey() {
     int res;

     vector<uint64_t> keyres;
     int randBits = crypt.sync->randomBitsCount();
     if (randBits == 0)
          return 0;
     int currKeyBits = 0;
     int keyReady = 0;
     int keyNeedSize = crypt.symm->GetKeySize() * 8;
     keyres.resize((crypt.symm->GetKeySize() + 7) / 8);

     do {

          vector<uint64_t> vec, vec1;

          int st = 0;

          if (myType == 1) {
               std::cout << "CH synckey cl" << std::endl;
               do {
                    vec.resize(crypt.sync->stepsize(1, st));

                    if (vec.size() > 0) {
                         std::cout << "CH synckey cl recv st:" << vec.size() <<
                         std::endl;

                         vector<uint64_t> arrh, arrauth;
                         arrh.resize(crypt.hash->HashSize());
                         arrauth.resize(crypt.auth->MaxMSize(), 0);
                         crypt.connect->recvV(vec.data(), vec.size() * 8);
                         crypt.connect->recvV(arrauth.data(), arrauth.size() * 8);
                         crypt.hash->Hash(vec.data(), vec.size(), arrh);
                         int resauth = crypt.auth->Test(arrauth, arrh);

                         if (resauth == 0)
```

```
                    return 0;

                std::cout << "CH synckey cl recv end:" << bigint(vec, 1) <<
                std::endl;
            }

            res = crypt.sync->step1(vec, vec1, st, crypt.rg);
            std::cout << "CH synckey cl step:" << res << std::endl;
            if (vec1.size() > 0 && res != -1) {
                crypt.connect->sendV(vec1.data(), vec1.size() * 8);

            }
            st++;
        } while (res == 0);
    }
    if (myType == 2) {
        std::cout << "CH synckey sv" << std::endl;
        do {
            vec.resize(crypt.sync->stepsize(2, st));

            if (vec.size() > 0) {
                std::cout << "CH synckey sv recv st:" << vec.size() <<
                std::endl;
                crypt.connect->recvV(vec.data(), vec.size() * 8);
                std::cout << "CH synckey sv recv end:" << bigint(vec, 1) <<
                std::endl;
            }

            res = crypt.sync->step2(vec, vec1, st, crypt.rg);
            std::cout << "CH synckey sv step:" << res << std::endl;
            if (vec1.size() > 0 && res != -1) {
                vector<uint64_t> arrh, arrauth;
                arrh.resize(crypt.hash->HashSize());
                crypt.hash->Hash(vec1.data(), vec1.size(), arrh);
                crypt.auth->Crypt(arrh, arrauth);
                std::cout << "CH synckey sv send H:" << bigint(arrh, 1) <<
                std::endl;
                std::cout << "CH synckey sv send A:" << bigint(arrauth, 1)
                << std::endl;
                arrauth.resize(crypt.auth->MaxMSize(), 0);
                crypt.connect->sendV(vec1.data(), vec1.size() * 8);
                crypt.connect->sendV(arrauth.data(), arrauth.size() * 8);
            }
            st++;
        } while (res == 0);
    }

    if (res != 1)
        return 0;

    vector<uint64_t> arr2 = crypt.sync->getResult();
    arr2.resize((randBits + 63) / 64, 0);

    int addd = currKeyBits / 64;
    int addm = currKeyBits % 64;

    if (randBits % 64 != 0)
        arr2[(randBits - 1) / 64] &= ((uint64_t(1) << (randBits % 64)) - 1);

    std::cout << bigint(arr2, 1) << std::endl;
```

```
        for (int i = 0; i < arr2.size() && (i + addd)<keyres.size(); i++)
            keyres[i + addd] |= arr2[i] << addm;
        if (addm != 0)
            for (int i = 0; i < arr2.size() && (i + addd + 1)<keyres.size(); i++)
                keyres[i + addd + 1] |= arr2[i] >> (64 - addm);

        currKeyBits += randBits;
        if (currKeyBits >= keyNeedSize)
            keyReady = 1;
        std::cout << "---------------" << std::endl;
    } while (keyReady == 0);

    std::cout << bigint(keyres, 1) << std::endl;

    int testres = 1;

    if (myType == 1) {
        vector<uint8_t> testrandval, testrandvalret, mess;
        mess.resize(crypt.symm->GetDataScale(), 0);
        testrandval.resize(crypt.symm->GetDataScale(), 0);
        testrandvalret.resize(crypt.symm->GetDataScale(), 0);
        crypt.rg->GetRandom(testrandval.data(), testrandval.size());

        crypt.symm->SetKey(keyres);

        crypt.symm->Encode(testrandval.data(), mess.data(), mess.size());

        crypt.connect->sendV(mess.data(), mess.size());
        crypt.connect->recvV(mess.data(), mess.size());

        crypt.symm->Decode(mess.data(), testrandvalret.data(), mess.size());

        for (int i = 0; i < testrandval.size(); i++) {
            if ((testrandval[i] ^ testrandvalret[i]) != 0xFF)
                testres = 0;
        }

        for (int i = 0; i < testrandval.size() / 2; i++) {
            std::swap(testrandvalret[i], testrandvalret[testrandval.size() - 1 -
            i]);
        }
        for (int i = 0; i < testrandval.size(); i++)
            testrandvalret[i] = reverseBits(testrandvalret[i]);

        crypt.symm->Encode(testrandvalret.data(), mess.data(), mess.size());

        crypt.connect->sendV(mess.data(), mess.size());

        symmChainData = testrandvalret;
        symmChainDataDecode = testrandvalret;
    }
    if (myType == 2) {
        vector<uint8_t> testrandval, testrandvalret, mess;
        mess.resize(crypt.symm->GetDataScale(), 0);
        testrandval.resize(crypt.symm->GetDataScale(), 0);
        testrandvalret.resize(crypt.symm->GetDataScale(), 0);

        crypt.connect->recvV(mess.data(), mess.size());

        crypt.symm->SetKey(keyres);
```

```
            crypt.symm->Decode(mess.data(), testrandval.data(), mess.size());

            for (int i = 0; i < testrandval.size(); i++)
                testrandval[i] = ~testrandval[i];

            crypt.symm->Encode(testrandval.data(), mess.data(), mess.size());

            crypt.connect->sendV(mess.data(), mess.size());

            for (int i = 0; i < testrandval.size() / 2; i++) {
                std::swap(testrandval[i], testrandval[testrandval.size() - 1 - i]);
            }
            for (int i = 0; i < testrandval.size(); i++)
                testrandval[i] = reverseBits(testrandval[i]);

            crypt.connect->recvV(mess.data(), mess.size());

            crypt.symm->Decode(mess.data(), testrandvalret.data(), mess.size());

            for (int i = 0; i < testrandval.size(); i++) {
                if (testrandval[i] != testrandvalret[i])
                    testres = 0;
            }
            symmChainData = testrandvalret;
            symmChainDataDecode = testrandvalret;

        }

        std::cout << "Test:" << testres << std::endl;


        return testres;

    }

    inline void NCryptChannel::SCrypt(void * data, int size) {
        int scale = crypt.symm->GetDataScale();
        if (size%scale != 0)
            throw;
        uint8_t *lbytedata = reinterpret_cast<uint8_t*>(data);

        for (int i = 0; i < size / scale; i++) {
            for (int r = 0; r < symmChainData.size(); r++)
                lbytedata[i * scale + r] ^= symmChainData[r];
            crypt.symm->Encode(lbytedata + i * scale, lbytedata + i * scale, scale);
            for (int r = 0; r < symmChainData.size(); r++)
                symmChainData[r] = lbytedata[i * scale + r];
        }

    }

    inline void NCryptChannel::SEncrypt(void * data, int size) {
        int scale = crypt.symm->GetDataScale();
        if (size%scale != 0)
            throw;
        uint8_t *lbytedata = reinterpret_cast<uint8_t*>(data);
        vector<uint8_t> temp(symmChainDataDecode.size());
        for (int i = 0; i < size / scale; i++) {
            for (int r = 0; r < symmChainDataDecode.size(); r++)
                temp[r] = lbytedata[i * scale + r];
```

```
            crypt.symm->Decode(lbytedata + i * scale, lbytedata + i * scale, scale);

            for (int r = 0; r < symmChainDataDecode.size(); r++)
                lbytedata[i * scale + r] ^= symmChainDataDecode[r];

            symmChainDataDecode = temp;
        }
    }

    inline bool NCryptChannel::senddata(const void * data, int size) {
        if (crypt.symm->GetDataScale() % 8 != 0)
            throw;
        if (size < 0)
            return 0;

        //hhhhhhhhssssssssSSSSSSSSddddddddddddddddddddRRRRRR

        int sizepadd = 3 * 8 + size;
        if (sizepadd % crypt.symm->GetDataScale() != 0)
            sizepadd += crypt.symm->GetDataScale() - (sizepadd % crypt.symm-
            >GetDataScale());
        vector<uint64_t> vecsend(sizepadd / 8, 0);

        vecsend[1] = size;
        vecsend[2] = size;

        uint8_t *lbytevec = reinterpret_cast<uint8_t*>(&(vecsend[3]));
        const uint8_t *lbytedata = reinterpret_cast<const uint8_t*>(data);
        for (int i = 0; i < size; i++)
            lbytevec[i] = lbytedata[i];

        if (((3 * 8 + size) % crypt.symm->GetDataScale())  > 0)
            crypt.rg->GetRandom(&lbytevec[size], crypt.symm->GetDataScale() - ((3 * 8
            + size) % crypt.symm->GetDataScale()));

        uint64_t xorhash = 0;
        for (int i = 2; i < vecsend.size(); i++)
            xorhash ^= vecsend[i];

        vecsend[0] = xorhash;

        SCrypt(vecsend.data(), vecsend.size() * 8);

        return crypt.connect->sendV(vecsend.data(), vecsend.size() * 8);

    }

    inline bool NCryptChannel::recvdata(vector<uint8_t>& vec) {
        if (crypt.symm->GetDataScale() % 8 != 0)
            throw;
        int sizehpadd = 3 * 8;
        if (sizehpadd % (crypt.symm->GetDataScale()) != 0)
            sizehpadd += crypt.symm->GetDataScale() - (sizehpadd % crypt.symm-
            >GetDataScale());

        vector<uint64_t> vechead(sizehpadd / 8, 0);

        int resrecv = crypt.connect->recvV(vechead.data(), vechead.size() * 8);
        if (!resrecv)
            return 0;
```

```
     SEncrypt(vechead.data(), vechead.size() * 8);

     if (vechead[1] != vechead[2])
          return 0;
     int size = vechead[1];

     int sizepadd = 3 * 8 + size;
     if (sizepadd % crypt.symm->GetDataScale() != 0)
          sizepadd += crypt.symm->GetDataScale() - (sizepadd % crypt.symm->
          GetDataScale());

     if (sizepadd / 8 - vechead.size() > 0) {
          vechead.resize(sizepadd / 8, 0);

          int resrecv = crypt.connect->recvV(vechead.data() + sizehpadd / 8,
          vechead.size() * 8 - sizehpadd);
          if (!resrecv)
               return 0;

          SEncrypt(vechead.data() + sizehpadd / 8, vechead.size() * 8 - sizehpadd);
     }

     uint64_t xorhash = 0;
     for (int i = 2; i < vechead.size(); i++)
          xorhash ^= vechead[i];

     if (xorhash != vechead[0])
          return 0;

     uint8_t *lbytevec = reinterpret_cast<uint8_t*>(&(vechead[3]));
     vec.resize(size);
     for (int i = 0; i < size; i++) {
          vec[i] = lbytevec[i];
     }

     return 1;

}
```

Файл NCryptAuthRSA.h

```
#pragma once
#include "NCryptBase.h"
#include <numeric>

bigint gcdex(bigint a, bigint b, bigint & x, bigint & y) {
     if (a == 0) {
          x = 0; y = 1;
          return b;
     }
     bigint x1, y1;
     bigint d = gcdex(b%a, a, x1, y1);
     x = y1 - (b / a) * x1;
     y = x1;
     return d;
}

class NCryptAuthRSA :public NCryptAutentification {
private:
     bigint key_n, key_e, key_d;
public:
```

```
    //Максимальный размер подписываемого сообщения
    __declspec(dllexport) int MaxMSize();

    //Установить приватный ключ
    __declspec(dllexport) bool SetPrivate(std::vector<uint64_t> PrivateKey);

    //Установить публичный ключ
    __declspec(dllexport) bool SetToken(std::vector<uint64_t> PublicKey);

    //Сравнение зашифрованного хеша с хешом сообщения
    __declspec(dllexport) bool Test(std::vector<uint64_t> arrm,
    std::vector<uint64_t> arrh);

    //Генерирование ключей
    __declspec(dllexport) bool Generate(int64_t size, std::vector<uint64_t>&
    PrivateKey, std::vector<uint64_t>& PublicKey, std::shared_ptr<NRandomGen> rg);

    //Шифрование хеша
    __declspec(dllexport) bool Crypt(std::vector<uint64_t> harr,
    std::vector<uint64_t> &res);

    __declspec(dllexport) ~NCryptAuthRSA();
};
```

Файл NCryptAuthRSA.cpp

```
#include "NCryptAuthRSA.h"

int NCryptAuthRSA::MaxMSize() {
    return key_n.data().size();
}

bool NCryptAuthRSA::SetPrivate(std::vector<uint64_t> PrivateKey) {
    if (PrivateKey.size() < 4)
        return false;

    uint64_t n1 = PrivateKey[0];

    if (PrivateKey.size() < n1 + 1)
        return false;
    vector<uint64_t> arr1(n1);
    for (int i = 0; i < n1; i++) {
        arr1[i] = PrivateKey[i + 1];
    }


    if (PrivateKey.size() < n1 + 2)
        return false;
    uint64_t n2 = PrivateKey[n1 + 1];

    if (PrivateKey.size() < n1 + 1 + n2 + 1)
        return false;
    vector<uint64_t> arr2(n2);
    for (int i = 0; i < n2; i++) {
        arr2[i] = PrivateKey[i + n1 + 2];
    }

    key_d = bigint(arr1, 1);
    key_n = bigint(arr2, 1);
```

```
     return true;
}

bool NCryptAuthRSA::SetToken(std::vector<uint64_t> PublicKey) {
     if (PublicKey.size() < 4)
          return false;

     uint64_t n1 = PublicKey[0];

     if (PublicKey.size() < n1 + 1)
          return false;
     vector<uint64_t> arr1(n1);
     for (int i = 0; i < n1; i++) {
          arr1[i] = PublicKey[i + 1];
     }


     if (PublicKey.size() < n1 + 2)
          return false;
     uint64_t n2 = PublicKey[n1 + 1];

     if (PublicKey.size() < n1 + 1 + n2 + 1)
          return false;
     vector<uint64_t> arr2(n2);
     for (int i = 0; i < n2; i++) {
          arr2[i] = PublicKey[i + n1 + 2];
     }

     key_e = bigint(arr1, 1);
     key_n = bigint(arr2, 1);

     return true;
}

bool NCryptAuthRSA::Test(std::vector<uint64_t> arrm, std::vector<uint64_t> arrh) {
     bigint h(arrh, 1);
     h = h % key_n;
     bigint m(arrm, 1);
     if (key_n == 0)
          return false;
     m = powmod(m, key_e, key_n);
     return m == h;
}

bool NCryptAuthRSA::Generate(int64_t size, std::vector<uint64_t>& PrivateKey,
std::vector<uint64_t>& PublicKey, std::shared_ptr<NRandomGen> rg) {
     bigint p1, p2, n, eiler_n, e, d;
     p1 = randomPrime(static_cast<int>(size), rg);
     p2 = randomPrime(static_cast<int>(size), rg);
     n = p1 * p2;
     eiler_n = (p1 - 1) * (p2 - 1);

     vector<uint64_t> arr;
     arr.resize(eiler_n.data().size());
     bigint gx, gy;
     bigint gres = 0;
     while (gres != 1) {

          rg->GetRandom(arr.data(), arr.size() * sizeof(uint64_t));
          e = bigint(arr, 1);
          e = e % eiler_n;
```

```
            if (e < (bigint(1) << static_cast<int>(size / 2))) {
                e = e + (bigint(1) << static_cast<int>(size / 2));
            }

            gres = gcdex(e, eiler_n, gx, gy);
        }
        d = (gx + eiler_n) % eiler_n;
        auto &narr = n.data();
        auto &earr = e.data();
        auto &darr = d.data();


        bigint m = 2;
        m = powmod(m, e, n);
        m = powmod(m, d, n);
        if (m != 2)
            throw;


        PrivateKey.resize(darr.size() + narr.size() + 2);

        PrivateKey[0] = darr.size();
        for (int i = 0; i<darr.size(); i++)
            PrivateKey[i + 1] = darr[i];

        PrivateKey[darr.size() + 1] = narr.size();
        for (int i = 0; i<narr.size(); i++)
            PrivateKey[i + darr.size() + 2] = narr[i];



        PublicKey.resize(earr.size() + narr.size() + 2);

        PublicKey[0] = earr.size();
        for (int i = 0; i<earr.size(); i++)
            PublicKey[i + 1] = earr[i];

        PublicKey[earr.size() + 1] = narr.size();
        for (int i = 0; i<narr.size(); i++)
            PublicKey[i + earr.size() + 2] = narr[i];

        return true;
    }

    bool NCryptAuthRSA::Crypt(std::vector<uint64_t> harr, std::vector<uint64_t>& res) {

        bigint h = bigint(harr, 1);

        h = h % key_n;
        h = powmod(h, key_d, key_n);

        res = h.data();

        return true;
    }

    NCryptAuthRSA::~NCryptAuthRSA() {

    }
```