МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Систем сбора и обработки данных

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

***Соснова Максима Евгеньевича***

(фамилия, имя, отчество автора)

***Разработка клиентского программного обеспечения для обмена данными на основе ассиметричного шифрования.***

|  |  |
| --- | --- |
| Направление подготовки | ***230200 Информационные системы*** |

|  |  |
| --- | --- |
| **Руководитель**  ***Воронов В.В.***  (фамилия, И.О.)  ***Старший преподаватель кафедры ССОД***  (уч. степень, уч. звание)  (подпись, дата) | **Автор**  ***Соснов М.Е.***  (фамилия, И.О.)  ***АВТ,*** ***АТ-03***  (факультет, группа)  (подпись, дата) |

Новосибирск, 2014 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Систем сбора и обработки данных

|  |  |
| --- | --- |
|  | Утверждаю  Зав. кафедрой  (подпись)  Белик Д.В.  (фамилия, инициалы)  « »   г. |

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

студенту ***Соснову Максиму Евгеньевичу***

(фамилия, имя, отчество студента)

|  |  |
| --- | --- |
| Направление подготовки | ***230200 Информационные системы*** |

***Факультет автоматики и вычислительной техники***

Тема  ***Разработка клиентского программного обеспечения для обмена данными на основе ассиметричного шифрования.***

Исходные данные (или цель работы):

*Разработка клиентского программного обеспечения, позволяющего производить обмен данными по защищенному протоколу через незащищенный канал.*

Структурные части работы:

Изучение предметной области. Выбор параметров будущей системы. Определение основных принципов. Создание протокола с учетом существующих методов шифрования и атак на них. Выбор инструментов для создания клиентского приложения. Создание клиентского приложения. Тестирование приложения и его компонентов. Описание полученной технологии. Создание ВКР.

**План-график выполнения работы**

|  |  |  |
| --- | --- | --- |
| № п/п | Наименование этапа | Планируемые сроки выполнения |
| 1 | Анализ существующих алгоритмов шифрования, их преимуществ/недостатков а также известных атак. Анализ способов защиты от известных атак. | 12.02.2014 – 01.04.2014 |
| 2 | Выбор параметров будущей системы. Определение основных принципов. | 05.04.2014 – 15.05.2014 |
| 3 | Создание протокола с учетом существующих методов шифрования и атак на них. | 20.05.2014 – 26.05.2014 |
| 4 | Выбор инструментов для создания клиентского приложения. | 26.05.2014 – 27.05.2014 |
| 5 | Создание клиентского приложения. Тестирование приложения и его компонентов. | 27.05.2014 – 09.06.2014 |
| 6 | Описание полученной технологии. Создание ВКР. | 10.06.2014 – 20.06.2014 |

Задание согласовано и принято к исполнению.

|  |  |
| --- | --- |
| **Руководитель**  ***Воронов В.В.***  (фамилия, И.О.)  ***Старший преподаватель***  (уч. степень, уч. звание)    ***(***подпись, дата) | **Автор**  ***Соснов М.Е.***  (фамилия, И.О.)  ***АВТ, АТ-03***  (факультет, группа)    ***(***подпись, дата) |

Тема утверждена приказом по НГТУ №***6548/2***  от « ***4*** »***декабря***  ***2013*** г.

(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

(фамилия, инициалы секретаря экзаменационной комиссии по защите ВКР)

**Аннотация**

Оглавление

[Введение 7](#_Toc390460671)

[Глава 1: Описание предметной области 9](#_Toc390460672)

[1.1 Исследование существующих методов шифрования, их достоинств и недостатков 9](#_Toc390460673)

[1.1.1 RSA 9](#_Toc390460674)

[1.1.1.1 Определение 9](#_Toc390460675)

[1.1.1.2 Описание алгоритма 9](#_Toc390460676)

[1.1.1.2.1 Создание ключей 9](#_Toc390460677)

[1.1.1.2.2 Шифрование и дешифрование 10](#_Toc390460678)

[1.1.2 Elliptic Curve 11](#_Toc390460679)

[1.1.2.1 Определение 11](#_Toc390460680)

[1.1.2.2 Описание алгоритма 11](#_Toc390460681)

[1.1.2.2.1 Параметры: 11](#_Toc390460682)

[1.1.2.2.2 Генерация ключей: 12](#_Toc390460683)

[1.1.2.2.3 Шифрование и дешифрование 12](#_Toc390460684)

[1.1.2.2.3.1 Шифрование 12](#_Toc390460685)

[1.1.2.2.3.2 Дешифрование 12](#_Toc390460686)

[1.1.3 Advanced Encryption Standrard 13](#_Toc390460687)

[1.1.3.1 Описание 13](#_Toc390460688)

[1.1.3.2 Алгоритм шифрования 13](#_Toc390460689)

[1.1.3.2.1 Определения 13](#_Toc390460690)

[1.1.3.2.2 Схема 13](#_Toc390460691)

[1.1.3.2.3 Подготовка данных 14](#_Toc390460692)

[1.1.3.2.4 SubButes() 15](#_Toc390460693)

[1.1.3.2.5 ShiftRows() 15](#_Toc390460694)

[1.1.3.2.6 MixColumns() 15](#_Toc390460695)

[1.1.3.2.7 AddRoundKey() 15](#_Toc390460696)

[1.1.3.2.8 KeyExpansion() 15](#_Toc390460697)

[1.1.3.2.8.1 Алгоритм дозаполнения KeySchedule: 16](#_Toc390460698)

[1.1.3.3 Алгоритм дешифрования 17](#_Toc390460699)

[1.1.3.3.1 Схема 17](#_Toc390460700)

[1.1.3.3.2 InvSubBytes() 17](#_Toc390460701)

[1.1.3.3.3 InvShiftRows() 18](#_Toc390460702)

[1.1.3.3.4 InvMixColumns() 18](#_Toc390460703)

[1.1.3.3.5 AddRoundKey() 18](#_Toc390460704)

[1.1.4 XOR 18](#_Toc390460705)

[1.1.5 Протокол Ди́ффи – Хе́ллмана 19](#_Toc390460706)

[1.1.5.1 описание 19](#_Toc390460707)

[1.1.5.2 Алгоритм 19](#_Toc390460708)

[1.1.5.3 Пример: 19](#_Toc390460709)

[1.1.6 SHA-2 20](#_Toc390460710)

[1.1.7 MD5 21](#_Toc390460711)

[1.1.8 HMAC 22](#_Toc390460712)

[1.2. Исследование существующих решений. 24](#_Toc390460713)

[1.2.1 Secure Sockets Layer 24](#_Toc390460714)

[1.2.1.1 Аутентификация и обмен ключами 25](#_Toc390460715)

[1.2.2 Transport Layer Security 25](#_Toc390460716)

[1.2.2.1 Описание 26](#_Toc390460717)

[1.3 Выводы по результатам проработки предметной области 27](#_Toc390460718)

[Глава 2: Разработка протокола 29](#_Toc390460719)

[2.1 Анализ известных атак на протоколы передачи данных 29](#_Toc390460720)

[2.2. Разработка протокола передачи данных с учетом проведенного исследования 29](#_Toc390460721)

[2.3 Анализ протокола на уязвимость к известным атакам 31](#_Toc390460722)

[Глава 3: Разработка клиентского приложения 33](#_Toc390460723)

[3.1. Описание дизайна приложения, выбор инструментов 33](#_Toc390460724)

[3.2. Разработка клиентского приложения 33](#_Toc390460725)

[3.3. Оценка получившегося решения 33](#_Toc390460726)

[Заключение 34](#_Toc390460727)

[1. Общая оценка работы 34](#_Toc390460728)

[2. Полнота решения поставленных задач 3. Экономическая и научная значимость работы 34](#_Toc390460729)

[Список использованных источников 35](#_Toc390460730)

[Приложения 36](#_Toc390460731)

# Введение

С появлением сети интернет открылась возможность передавать данные с одного конца мира в другой за короткое время. Однако сеть интернет не является защищенным каналом передачи данных исходя из его строения. При передаче данных пакет, следуя от отправителя к получателю, проходит множество узлов, каждый из которых при должном желании может посмотреть содержимое пакета.

Для конфиденциальности передачи данных используются различные алгоритмы, основанные на шифровании данных и электронных подписях. Например, существуют алгоритмы SSL и TLS, использующие асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Эти алгоритмы повсеместно используются в различных приложениях, использующих сеть Интернет (Веб-браузеры, приложения для работы с электронной почтой, мессенджеры, IP-телефония и тд). Существует множество бесплатных библиотек с открытым исходным кодом для установления безопасного соединения в открытом канале передачи данных по технологиям SSL/TLS, но, как показывают события последних месяцев библиотеки с открытым исходным кодом не всегда справляются со своей работой. Например, в апреле 2014го года была найдена уязвимость в библиотеке OpenSSL, названной “Heartbleed”, которая позволяла получить стороннему человеку расшифрованные переданные данные. Спустя 2 месяца была найдена еще одна уязвимость. Надо заметить, что эти уязвимости были внесены с функционалом в начале 2012-го года. Таким образом, начиная с 2012го года информация, передаваемая с помощью OpenSSL могла быть получена злоумышленниками, знающими об уязвимости. Например, Агентство Национальной Безопасности США использовало эту уязвимость в своих целях 2 года.

Крупным информационным системам, которым нужно обеспечивать конфиденциальность передачи данных, желательно использовать качественное программное обеспечение для этих целей. Например, крупный банк может позволить себе потратиться на разработку своего решения создания защищенного канала передачи данных в среде интернет.

Целями бакалаврской работы являются:

1. Определение протокола и криптографических систем, для обеспечения безопасного соединения.
2. Создание клиентского приложения, способного установить защищенное соединение с сервером, по созданному протоколу.

# Глава 1: Описание предметной области

## 1.1 Исследование существующих методов шифрования, их достоинств и недостатков

### 1.1.1 RSA

#### 1.1.1.1 Определение

**RSA** (аббревиатура от фамилий Rivest, Shamir и Adleman) — асимметричный криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптографические системы с открытым ключом используют так называемые односторонние функции, которые обладают следующим свойством:

Если известно x, то f(x) вычислить относительно просто

Если известно y=f(x), то для вычисления x нет простого (эффективного) пути.

Под односторонностью понимается не теоретическая однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования за разумное время (обратной операции) необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложения числа на простые множители.

#### 1.1.1.2 Описание алгоритма

##### 1.1.1.2.1 Создание ключей

RSA-ключи генерируются следующим образом:

* 1. Выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое).
  2. Вычисляется их произведение n=p\cdot q, которое называется *модулем*.
  3. Вычисляется значение функции Эйлера от числа n: \varphi(n) = (p-1)(q-1).
  4. Выбирается целое число e (1 < e < \varphi(n)),взаимно простое со значением функции \varphi(n). Обычно, в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые числа Ферма 17, 257 или 65537.
  5. Число e называется открытойэкспонентой (англ. *public exponent*)

Время, необходимое для шифрования с использованием быстрого возведения в степень, пропорционально числу единичных бит в e. Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.

* 1. Вычисляется число d, мультипликативно обратное к числу e по модулю \varphi(n), то есть число, удовлетворяющее условию: d\cdot e \equiv 1 \mod {\varphi(n)}.
  2. Число d называется *секретной экспонентой*. Обычно, оно вычисляется при помощи расширенного алгоритма Евклида.

Пара \left\{ e, n \right\} публикуется в качестве *открытого ключа RSA* (англ. *RSA public key*). Пара \left\{ d, n \right\} играет роль *закрытого ключа RSA* (англ. *RSA private key*) и держится в секрете.

##### 1.1.1.2.2 Шифрование и дешифрование

Шифрование сообщения m открытым ключом:  c \equiv m^e \pmod{n} , при этом *0 ≤ m < n*

Дешифрование сообщения c закрытым ключом:  m \equiv c^d \pmod{n} 

Пример алгоритма:

* 1. Выбрать простые числа p и q
  2. Вычислить n = p \* q
  3. Вычислить m = (p - 1) \* (q - 1)
  4. Выбрать число d взаимно простое с m
  5. Выбрать число e так, чтобы e \* d = 1 (mod m)
  6. Открытый ключ = (n,e). Закрытый ключ = (n,d)
  7. Шифрование: b = ae (mod n)
  8. Дешифровка: a = bd (mod n)

Практический пример:

* 1. Выбрали числа: p=61, q=53
  2. Вычисляем n=3233
  3. Вычисляем m`=3120
  4. Выбираем d=17
  5. Выбираем e=2753
  6. Выбираем сообщение m = 65
  7. Шифруем сообщение m c = 65^{17} \; \operatorname{mod}\; 3233 = 2790 
  8. Дешифруем сообщение m = 2790^{2753} \; \operatorname{mod}\; 3233 = 65 

### 1.1.2 Elliptic Curve

#### 1.1.2.1 Определение

Эллиптическая криптография — раздел [криптографии](http://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%8F), который изучает асимметричные криптосистемы, основанные на [эллиптических кривых](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%B2%D0%B0%D1%8F) над конечными полями. Основное преимущество эллиптической криптографии заключается в том, что на сегодняшний день неизвестно существование субэкспоненциальных [алгоритмов](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B) решения задачи [дискретного логарифмирования](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BB%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

При использовании алгоритмов на [эллиптических кривых](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%B2%D0%B0%D1%8F) полагается, что не существует субэкспоненциальных [алгоритмов](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B) для решения задачи [дискретного логарифмирования](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BB%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) в группах их точек. При этом [порядок группы](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D1%8B) точек эллиптической кривой определяет сложность задачи. Считается, что для достижения такого же уровня безопасности как и в RSA требуются группы меньших порядков, что уменьшает затраты на хранение и передачу информации.

#### 1.1.2.2 Описание алгоритма

##### 1.1.2.2.1 Параметры:

Пусть P - точка эллиптической кривой E над полем G(p), имеющая порядок n. Тогда циклическая подгруппа E порожденная точкой P будет состоять из точек {O, P,2P,3P, . . .,(n-1)P}. Характеристика поля p, уравнение эллиптической кривой E, точка P и ее порядок n являются параметрами кривой. Секретным ключом является число d, которое выбирается случайно из интервала [1, n-1], а открытым ключом является точка Q = dP. Задача вычисления d по известным параметрам кривой и точке Q называется проблемой дискретного логарифма в группе точек эллиптической кривой (ECDLP).

##### 1.1.2.2.2 Генерация ключей:

1. Вход: Параметры кривой (p, E, P, n).
2. Выход: Открытый ключ Q и секретный ключ d.
3. Алгоритм:
   1. Выбрать d из [1,n-1].
   2. Вычислить Q = dP.
   3. Вернуть: (Q,d)

##### 1.1.2.2.3 Шифрование и дешифрование

Открытый текст m представляется в виде точки *M*, а затем шифруется путем сложения с точкой *kQ*. Отправитель передает точку*C1 = kP* и *C2 = M + kQ* получателю, который использует свой секретный ключ *d* для вычисления *dC1 = d(kP) = kQ* и затем восстанавливает *M = C2 - k*q. Злоумышленник, желающий восстановить *M*, должен вычислить *kQ*. Проблема вычисления *kQ* при знании параметров кривой, *Q* и *C1 = kP*, является аналогом проблемы Диффи-Хеллмана для эллиптических кривых. Ниже предоставленалгоритм шифрования.

###### 1.1.2.2.3.1 Шифрование

1. Вход: Параметры кривой (p, E, P,n), открытый ключ Q, текст m.
2. Вывод: Криптограмма (C1,C2).
3. Алгоритм:
   1. Представить сообщение m в виде точки M кривой E(Fp).
   2. Выбрать k из R [1,n - 1].
   3. . Вычислить C1 = kP.
   4. . Вычислить C2 = M +kQ.
   5. Вернуть: (C1,C2).

###### 1.1.2.2.3.2 Дешифрование

1. **Вход**: Параметры кривой *(p, E, P,n)*, секретный ключ *d*, криптограмма *(C1,C2)*.
2. **Выход**: Исходный текст *m*.
3. **Алгоритм**:
   1. Вычислить *M = C2 - dC1*,
   2. Вычислить *m* из *M*.
4. **Вернуть**: (*m*).

### 1.1.3 Advanced Encryption Standrard

#### 1.1.3.1 Описание

**Advanced Encryption Standard** (**AES**), также известный как Rijndael (произносится [rɛindaːl] (Рэндал) ) — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурс. Этот алгоритм хорошо проанализирован и сейчас широко используется. Национальный институт стандартов и технологий США (англ. National Institute of Standards and Technology, NIST) опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур. 26 мая 2002 года AES был объявлен стандартом шифрования. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования.

#### 1.1.3.2 Алгоритм шифрования

##### 1.1.3.2.1 Определения

1. State — промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов имеющий 4 строки и Nb колонок. Каждая ячейка State содержит значение размером в 1 байт
2. Nb — число столбцов (32-х битных слов), составляющих State. Для стандарта регламентировано Nb = 4
3. Nk — длина ключа в 32-х битных словах. Для AES, Nk = 4, 6, 8.
4. Nr — количество раундов шифрования. В зависимости от длины ключа, Nr = 10, 12 или 14

##### 1.1.3.2.2 Схема

Алгоритм имеет четыре трансформации, каждая из которых своим образом влияет на состояние State и в конечном итоге приводит к результату: *SubBytes(), ShiftRows(), MixColumns()* и *AddRoundKey()*. Общую схему шифрования можно представить как:



Рисунок 1.1Схема шифрования AES

##### 1.1.3.2.3 Подготовка данных

В начале заполняется массив State входными значениями по формуле State[r][c] = input[r + 4c], r = 0,1...4; c = 0,1..Nb. То есть по колонкам. За раз шифруется блок размером 16 байт.



Рис.2. Заполнение State.

Алгоритм оперирует байтами, считая их элементами конечного поля или поля Галуа GF(28). Элементами поля GF(28) являются многочлены степени не более 7, которые могут быть заданы строкой своих коэффициентов. Например, байту {1,1,1,0,0,0,1,1} соответствует элемент поля 1x7 + 1x6 + 1x5 + 0x4 + 0x3 + 0x2 + 1x1 + 1x0 = 1x7 + 1x6 + 1x5 + x +1. То, что мы работаем с элементами поля, очень важно потому, что это меняет правила операций сложения и умножения.

##### 1.1.3.2.4 SubButes()

Преобразование представляет собой замену каждого байта из State на соответствующий ему из константной таблицы Sbox.

##### 1.1.3.2.5 ShiftRows()

Простая трансформация. Она выполняет циклический сдвиг влево на 1 элемент для первой строки, на 2 для второй и на 3 для третьей. Нулевая строка не сдвигается.

##### 1.1.3.2.6 MixColumns()

В рамках этой трансформации каждая колонка в State представляется в виде многочлена и перемножается в поле GF(28) по модулю x4 + 1 с фиксированным многочленом 3x3 + x2 + x + 2.



Рис. 5. Mix Columns AES матричная запись предоставленная в официальном документе стандарта.

MixColumns() вместе с ShiftRows() добавляют диффузию в шифр.

##### 1.1.3.2.7 AddRoundKey()

Трансформация производит побитовый XOR каждого элемента из State с соответствующим элементом из RoundKey. RoundKey — массив такого же размера, как и State, который строится для каждого раунда на основе секретного ключа функцией KeyExpansion(), которую и рассмотрим далее.

##### 1.1.3.2.8 KeyExpansion()

Эта вспомогательная трансформация формирует набор раундовых ключей — KeySchedule. KeySchedule представляет собой длинную таблицу, состоящую из Nb\*(Nr + 1) столбцов или (Nr + 1) блоков, каждый из которых равен по размеру State. Первый раундовый ключ заполняется на основе секретного ключа, который вы придумаете, по формуле KeySchedule[r][c] = SecretKey[r + 4c], r = 0,1...4; c = 0,1..Nk.



Рис. 6. KeySchedule для AES 128.

На рисунке 6 изображен макет KeySchedule для AES-128: 11 блоков по 4 колонки. Для других вариаций алгоритма будет соответственно (Nr + 1) блоков по Nb колонок. Для преобразований определена константная таблица — Rcon

###### 1.1.3.2.8.1 Алгоритм дозаполнения KeySchedule:

На каждой итерации работаем с колонкой таблицы. Начинаем с колонки под номером Nk (в нашем случае с четвертой)

Если номер Wi колонки кратен Nk (в нашем случае каждая четвертая), то берем колонку Wi-1, выполняем над ней циклический сдвиг влево на один элемент, затем все байты колонки заменяем соответствующими из таблицы Sbox, как делали это в SubBytes(). Далее выполняем операцию XOR между колонкой Wi-Nk, измененной Wi-1 и колонкой Rconi/Nk-1. Результат записывается в колонку Wi. Чтобы было немного понагляднее, иллюстрация для i = 4.



Рис. 8. Алгоритм дозаполнения KeySchedule.

Для остальных колонок выполняем XOR между Wi-Nk и Wi-1. Результат записываем в Wi

Это, собственно, все, что касается процесса шифрования. Выходной массив зашифрованных байтов составляется из State по формуле **output[r + 4c] = State[r][c], r = 0,1...4; c = 0,1..Nb**.

#### 1.1.3.3 Алгоритм дешифрования

##### 1.1.3.3.1 Схема

Идея здесь проста: если с тем же ключевым словом выполнить последовательность трансформаций, инверсных трансформациям шифрования, то получится исходное сообщение. Такими инверсными трансформациями являются InvSubBytes(), InvShiftRows(), InvMixColumns() и AddRoundKey(). Общая схема алгоритма расшифровки:



Рис. 9. Схема дешифрования AES.

##### 1.1.3.3.2 InvSubBytes()

Работает точно так же, как и SubBytes(), за исключением того, что замены делаются из константной таблицы InvSbox.

Оставшиеся обратные трансформации тоже будут очень похожи на свои прямые аналоги, поэтому в коде не выделяем под них отдельных функций. Каждая функция, описывающая трансформацию, будет иметь входную переменную inv. Если она равна False, то функция будет работать в обычном или прямом режиме(шифрование), если True — в инверсном(дешифровка).

##### 1.1.3.3.3 InvShiftRows()

Трансформация производит циклический сдвиг вправо на 1 элемент для первой строки State, на 2 для второй и на 3 для третьей. Нулевая строка не поворачивается.

##### 1.1.3.3.4 InvMixColumns()

Операции те же что и в MixColumns(), но каждая колонка State перемножается с другим многочленом {0b}x3 + {0d}x2 + {09}x + {0e}.



Рис. 12. InvMixColumns AES.

##### 1.1.3.3.5 AddRoundKey()

Эта трансформация обратна сама себе в силу свойства операции XOR: *(a XOR b) XOR b = a*

Набор раундовых ключей формируется таким же образом, как и для шифрования с помощью функции KeyExpansion(), но раундовые ключи необходимо подставлять в обратном порядке.

### 1.1.4 XOR

XOR шифрование основано на свойстве исключающего ИЛИ. Алгоритм:

1. a XOR 0 = a
2. a XOR a = 0
3. a XOR b = b XOR a
4. (a XOR b) XOR b = a

Таким образом, мы получаем самое быстрое симметричное шифрование. При этом в данной записи a – сообщение, b –ключ.

### 1.1.5 Протокол Ди́ффи – Хе́ллмана

#### 1.1.5.1 описание

**Протокол Ди́ффи — Хе́ллмана** (англ. *Diffie-Hellman*, *DH*) — 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.

#### 1.1.5.2 Алгоритм



Рис. 14. Схема обмена ключами по протоколу Диффи-Хеллмана.

#### 1.1.5.3 Пример:

Пусть Алена и Борис хотят обменяться ключами.

1. Алена и Борис соглашаются на том, чтобы использовать p = 23 и g = 5
2. Алена выбирает секретное число a=6 и посылает Борису *A* = *g****a*** mod *p*
   1. *A* = 5**6** mod 23 = **15,625** mod 23= 8
3. Борис выбирает секретное число b=15 и посылает Алене *B* = *g****b*** mod *p*
   1. *B* = 5**15** mod 23 = **30,517,578,125** mod 23 = 19
4. Алена вычисляет общий секретный ключ ***s*** = *B****a*** mod *p*
   1. ***s*** = 19**6** mod 23 = **47,045,881** mod 23= **2**
5. Борис вычисляет общий секретный ключ ***s*** = *A****b*** mod *p*
   1. ***s*** = 8**15** mod 23 = **35,184,372,088,832** mod 23 = **2**

В результате получаем общий секрет S. который неизвестен никому, кроме Алены и Бориса.

Примечание:

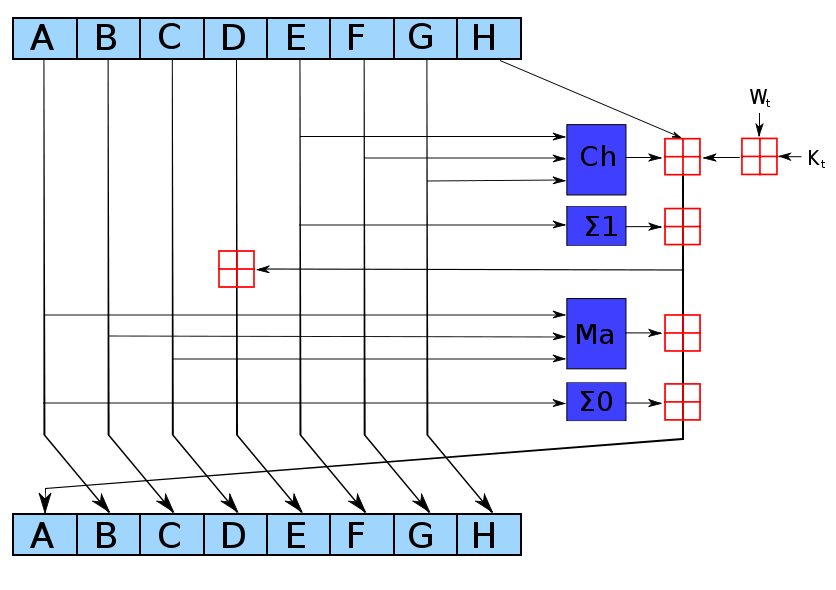
1. p - простое число
2. a,b - натуральные числа, такие, что a<p, b<p
3. g - первообразный корень по модулю p

### 1.1.6 SHA-2

SHA-2 ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Secure Hash Algorithm Version 2 — безопасный алгоритм хеширования, версия 2) — семейство [криптографических](http://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%8F) [алгоритмов](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) — однонаправленных [хеш-функций](http://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F), включающее в себя алгоритмы SHA-224, SHA-256, SHA-384 и SHA-512. Хеш-функции предназначены для создания «отпечатков» или «дайджестов» сообщений произвольной битовой длины. Применяются в различных приложениях или компонентах, связанных с [защитой информации](http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D1%89%D0%B8%D1%82%D0%B0_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8" \o "Защита информации).

Хеш-функции семейства SHA-2 построены на основе структуры Меркла — Дамгарда.

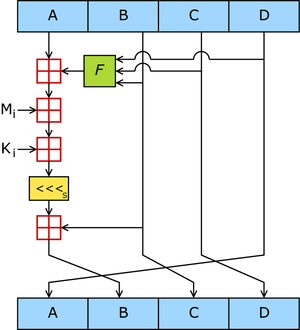
Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64-мя или 80-ю итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, с[умма](http://ru.wikipedia.org/wiki/%D0%A1%D1%83%D0%BC%D0%BC%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Сумма (математика)) является значением хеш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя.



### 1.1.7 MD5

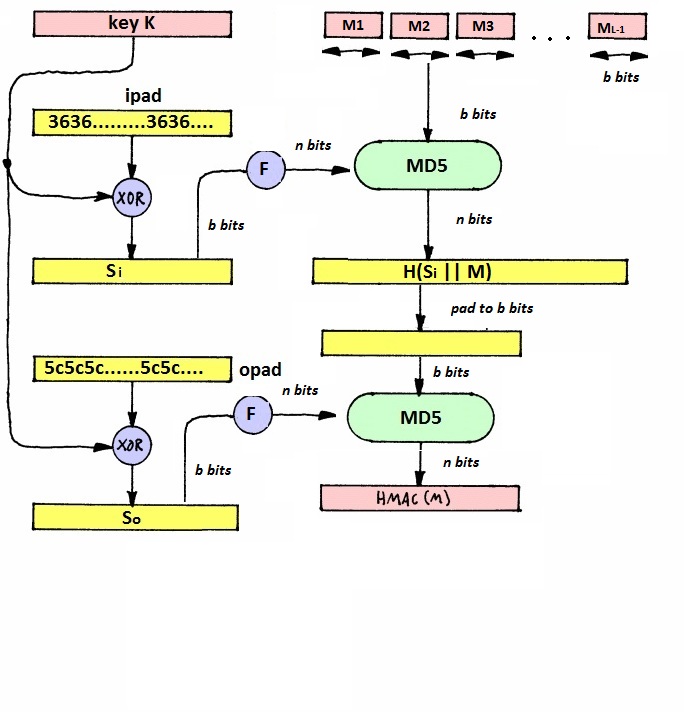
MD5 ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) Message Digest 5) — 128-битный алгоритм [хеширования](http://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5" \o "Хеширование), разработанный профессором [Рональдом Л. Ривестом](http://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%B2%D0%B5%D1%81%D1%82,_%D0%A0%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4" \o "Ривест, Рональд) из [Массачусетского технологического института](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82" \o "Массачусетский технологический институт) (Massachusetts Institute of Technology, MIT) в [1991 году](http://ru.wikipedia.org/wiki/1991_%D0%B3%D0%BE%D0%B4" \o "1991 год). Предназначен для создания «отпечатков» или [дайджестов сообщения](http://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%B9%D0%B4%D0%B6%D0%B5%D1%81%D1%82_%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F" \o "Дайджест сообщения) произвольной длины и последующей проверки их подлинности.

Алгоритм MD5 уязвим к некоторым атакам, например возможно создание двух сообщений с одинаковой хеш-суммой, поэтому его использование не рекомендуется.



### 1.1.8 HMAC

HMAC (сокращение от [англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) hash-based message authentication code, [хеш](http://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%81%D1%83%D0%BC%D0%BC%D0%B0" \o "Хеш-сумма)-код аутентификации сообщений). Наличие способа проверить целостность информации, передаваемой или хранящийся в ненадежной среде является неотъемлемой и необходимой частью мира открытых вычислений и коммуникаций. Механизмы, которые предоставляют такие проверки целостности на основе секретного ключа, обычно называют [кодом аутентичности сообщения](http://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%B8%D1%82%D0%BE%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0" \o "Имитовставка) (MAC). Как правило, МАС используется между двумя сторонами, которые разделяют секретный ключ для проверки подлинности информации, передаваемой между этими сторонами. Этот стандарт определяет MAC. Механизм, который использует криптографические хеш-функции в сочетании с секретным ключом называется HMAC.



Полученный код аутентичности позволяет убедиться в том, что данные не изменялись каким бы то ни было способом с тех пор как они были созданы, переданы или сохранены доверенным источником. Для такого рода проверки необходимо, чтобы, например, две доверяющие друг другу стороны заранее договорились об использовании секретного ключа, который известен только им. Тем самым гарантируется аутентичность источника и сообщения. Недостаток такого подхода очевиден — необходимо наличие двух доверяющих друг другу сторон.

## 1.2. Исследование существующих решений.

### 1.2.1 Secure Sockets Layer

SSL (англ. Secure Sockets Layer — уровень защищённых сокетов) — криптографический протокол, который обеспечивает безопасность связи. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Протокол широко используется для обмена мгновенными сообщениями и передачи голоса через IP (англ. Voice over IP — VoIP), в таких приложениях, как электронная почта, Интернет-факс и др.

Протокол SSL позволяет общаться клиенту с сервером в сети, предотвращая перехват или фальсификацию. Так как протоколы могут работать либо без SSL либо поверх SSL, то для клиента необходимо указать серверу, хочет ли он установить соединение SSL или нет. После того как клиент и сервер решили использовать SSL, они ведут переговоры, отслеживая состояние соединения с помощью процедуры рукопожатия. Во время этого рукопожатия клиент и сервер соглашаются на различные параметры, используемые для установки безопасного соединения. После завершения процедуры рукопожатия начинается защищенное соединение. Клиент и сервер используют сеансовые ключи для шифрования и расшифрования данных, которые они посылают друг другу. Это нормальный алгоритм работы по защищенному каналу. В любое время, в связи с внутренним или внешним раздражителем (автоматическое вмешательство или вмешательство пользователя), любая из сторон может пересмотреть сеанс связи. В этом случае, весь процесс повторяется. SSL работает модульным способом.

В протоколе SSL все данные передаются в виде записей-объектов, состоящих из заголовка и передаваемых данных. Передача начинается с заголовка. Заголовок содержит либо два, либо три байта кода длины. Причём, если старший бит в первом байте кода равен единице, то полная длина заголовка равна двум байтам, иначе — длина заголовка равна трём байтам. Код длины записи не включает в себя число байт заголовка.

#### 1.2.1.1 Аутентификация и обмен ключами

SSL поддерживает 3 типа аутентификации:

1. аутентификация обеих сторон (клиент — сервер),
2. аутентификация сервера с неаутентифицированным клиентом,
3. полная анонимность.

Если сервер аутентифицирован, то его сообщение о сертификации должно обеспечить верную сертификационную цепочку, ведущую к приемлемому центру сертификации. Проще говоря, аутентифицированный клиент должен предоставить допустимый сертификат серверу. Каждая сторона отвечает за проверку того, что сертификат другой стороны ещё не истек и не был отменен. Всякий раз, когда сервер аутентифицируется, канал устойчив (безопасен) к попытке перехвата данных между веб-сервером и браузером, но полностью анонимная сессия по своей сути уязвима к такой атаке. Анонимный сервер не может аутентифицировать клиента. Главная цель процесса обмена ключами — это создание секрета клиента (pre\_master\_secret), известного только клиенту и серверу. Секрет (pre\_master\_secret) используется для создания общего секрета (master\_secret). Общий секрет необходим для того чтобы создать сообщение для проверки сертификата, ключей шифрования, секрета MAC (message authentication code) и сообщения «finished». Отсылая сообщение «finished», стороны указывают, что они знают верный секрет (pre\_master\_secret).

### 1.2.2 Transport Layer Security

TLS (англ. Transport Layer Security) — безопасность транспортного уровня, как и его предшественник SSL (англ. Secure Socket Layers — уровень защищённых сокетов) — криптографические протоколы, обеспечивающие защищённую передачу данных между узлами в сети Интернет. TLS и SSL используют асимметричную криптографию для аутентификации, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений.

#### 1.2.2.1 Описание

TLS даёт возможность клиент-серверным приложениям осуществлять связь в сети таким образом, чтобы предотвратить прослушивание и несанкционированный доступ.

Так как большинство протоколов связи могут быть использованы как с, так и без TLS (или SSL), при установке соединения необходимо явно указать серверу, хочет ли клиент устанавливать TLS. Как только клиент и сервер договорились об использовании TLS, им необходимо установить защищённое соединение. Это делается с помощью процедуры подтверждения связи. Во время этого процесса клиент и сервер принимают соглашение относительно различных параметров, необходимых для установки безопасного соединения.

Основные шаги процедуры создания защищённого сеанса связи:

* клиент подключается к серверу, поддерживающему TLS, и запрашивает защищённое соединение;
* клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;
* сервер выбирает из списка, предоставленного клиентом, наиболее надёжные алгоритмы среди тех, которые поддерживаются сервером, и сообщает о своём выборе клиенту;
* сервер отправляет клиенту цифровой сертификат для собственной аутентификации. Обычно цифровой сертификат содержит имя сервера, имя удостоверяющего центра сертификации и открытый ключ сервера;
* клиент может связаться с сервером доверенного центра сертификации и подтвердить аутентичность переданного сертификата до начала передачи данных;
* для генерации сеансового ключа для защищённого соединения, клиент шифрует случайно сгенерированную цифровую последовательность открытым ключом сервера и посылает результат на сервер. Учитывая специфику алгоритма асимметричного шифрования, используемого для установления соединения, только сервер может расшифровать полученную последовательность, используя свой закрытый ключ.

На этом заканчивается процедура подтверждения связи. Между клиентом и сервером установлено безопасное соединение, данные, передаваемые по нему, шифруются и расшифровываются с использованием ключа шифрования до тех пор, пока соединение не будет завершено.

При возникновении ошибки на любом из вышеуказанных шагов подтверждение связи завершится с ошибкой и соединение не будет установлено.

Согласно протоколу TLS приложения обмениваются записями, инкапсулирующими (хранящими внутри себя) информацию, которая должна быть передана. Каждая из записей может быть сжата, дополнена, зашифрована или идентифицирована MAC (код аутентификации сообщения) в зависимости от текущего состояния соединения (состояния протокола). Каждая запись в TLS содержит следующие поля: content type (определяет тип содержимого записи), поле, указывающее длину пакета, и поле, указывающее версию протокола TLS.

## 1.3 Выводы по результатам проработки предметной области

Мы рассмотрели все основные успешные на данный момент алгоритмы шифрования, обмена ключами и цифровых подписей. Рассмотренные алгоритмы решают все основные проблемы реализации шифрования (передача ключа по незащищенному каналу, обеспечение аутентичности, повышение скорости после установления первоочередного “рукопожатия”). В выборе из двух алгоритмов обмена ключами (RSA против DH) более логичным выбором будет DH из-за более высокой скорости генерации ключей (при условии повторного использования некоторых параметров). Это помогает достичь *совершенной прямой секретности* (свойство алгоритма обмена ключами, которое гарантирует, что сессионные ключи, полученные при помощи набора ключей долговременного пользования, не будут скомпрометированы при компрометации одного из долговременных ключей) с использованием меньшей вычислительной мощности. Выбор AES для симметричного шифрования обусловлен достаточно высокой скоростью шифрования в сочетании с очень хорошей устойчивостью к взлому. Также, для алгоритма симметричной цифровой подписи был выбран SHA256, так как этот алгоритм хеширования является одобренным для применения в криптографических целях (для однонаправленной хеш-функции это значит, что не существует метода нахождения двух входных сообщений с одинаковой хеш-суммой более быстрого, чем перебор всех возможных входных сообщений).

Причины, описанные выше, являются лишь поверхностными описаниями. Более подробная информация доступна в следующей главе.

Т

# Глава 2: Разработка протокола

## 2.1 Анализ известных атак на протоколы передачи данных

Протоколы SSL и TLS на данный момент применяются очень широко. Разрабатываемый протокол будет основан на TLS и существующие атаки будут рассмотрены в контексте TLS. Первая реализация протокола будет создаваться с учетом контекста использования в банковских системах (все примеры будут приводиться с учетом этого). Описание создания протокола будет проходить “с нуля” (простое шифрование будет показано в начале и после указания на уязвимые места постепенно перейдет к финальной версии).

Для начала рассмотрим простейшую из схем:

1. Клиент соединяется с сервером.
2. Сервер передает клиенту общий ключ
3. Клиент и сервер переходят на схему, где каждый из них проводит побитовую операцию XOR чтобы зашифровать сообщение и отправляет его на другую сторону.
4. Получатель проводит ту же операцию еще раз, чтобы расшифровать сообщение.

Вышеописанная схема является одной из простейших и соответственно уязвимой ко всем основным видам даже простых атак, таких как:

* Network sniffing (прослушивание сети): атакующий, имеющий доступ к сети между клиентом и сервером может получить секретный ключ во время отправки его сервером. Атакующий сможет расшифровать любую информацию, переданную по этому соединению.
* Known-plaintext attack (атака “известного исходного текста”): в большинстве случаев атакующему уже известно (как минимум частично) какие именно сообщения отправляются по зашифрованному каналу (например сообщения “входа в систему” для протокола, расположенного поверх описываемого, могут всегда иметь слово ***login***, за которым следует пароль). В таком случае, атакующий может применить все ту же операцию XOR для извлечения ключа из отправленного любой из сторон сообщения. Любая известная информация об исходном тексте, таким образом, ускоряет расшифровку канала атакующим (а иногда и сводит время расшифровки к нулю).
* Replay (атака класса “повторной отправки”): атакующий может отправить любое сообщение одной из сторон повторно и получатель успешно расшифрует его (одинаковый исходный текст всегда шифруется в один и тот же зашифрованный текст при условии, что ключ не менялся) и примет за “чистую монету”. Для банковских транзакций, например, это означает повторное списание средств со счета (но не ограничивается этим).
* MITM (Man In The Middle, атака класса “человек посередине”): в дополнении к Network sniffing атакующий сможет передавать свои сообщения любой из сторон (путем подмены пакетов – этот тип атак является обыденным для беспроводных сетей по понятным причинам) выдавая их за подлинные, узнав ключ.

Также, атаки более высокого уровня применимы к описанному протоколу, но их использование не требуется (так как расшифровать все коммуникации возможно более простым способом).

В предыдущих главах описан алгоритм AES. AES является защищенным алгоритмом неуязвимым к атакам вида “известный исходный текст”. Заменив в нашем предыдущем примере XOR на AES мы избавимся от этого вида атаки. Также, используя AES в CFB (Cipher Feedback – каждый следующий зашифрованный блок данных зависит от предыдущего) режиме мы избавимся от атак “повторной отправки” (так как даже одинаковый исходный текст будет зашифрован по-разному при наличии различных векторов инициализации, применяемых перед началом шифрования первого блока).

Теперь протокол “менее уязвим” (в криптографии, на самом деле не существует такого понятия – протокол либо неуязвим либо полностью уязвим, так как одна уязвимость в большинстве случаев “тянет” за собой остальные в той или иной форме). Протокол до сих пор не обеспечивает аутентичности сообщений (получатель не знает кто именно отправил сообщение) – атакующий может перехватить первоочередные незашифрованные данные и установить два соединения (“человек по середине”). В таком случае, обе стороны будут думать, что общаются друг с другом, хотя в реальности будут общаться через атакующего (последний, естественно, получит возможность доступа и изменения любых данных).

Для решения проблемы аутентичности был изобретен MAC – Message Authentication Code (“код аутентификации сообщения”). Применение такой схемы добавляет к сообщениям “подпись” (аналогией будет печать из воска на бумажных письмах – письмо нельзя изменить не вскрыв печать, с той лишь разницей, что подписанное MAC сообщение может быть прочитано). Для реализации была выбрана схема основанная на хешировании – HMAC (Hash-based message authentication code). Описанный ранее алгоритм SHA-256 идеально подходит для этой цели. Ему соответствует алгоритм HMAC-SHA-256. Не вдаваясь в подробности, его отличие состоит в том, что он является функцией от двух параметров: **f(key, message)**, в отличие от хеш-функции, которая принимает один параметр. Это значит, что вывод HMAC является разным ***для каждой пары ключ+сообщение***. Хеш-функция же уникальна лишь для каждого сообщения. Атакующий не может вычислить правильное значение HMAC для сообщения не имея ключа. Добавляя значение HMAC в конце после каждого сообщения мы убеждаемся в аутентичности: “испорченное” сообщение при проверке выдаст другой HMAC с тем же ключом, испорченный HMAC не будет совпадать с тем, что получится при проверке (как уже было сказано, атакующий не может вычислить верный HMAC для своего измененного сообщения без ключа).

После использования AES в режиме CFB и HMAC-SHA-256 остается одна из самых важных и сложных проблем зашифрованной передачи данных: ***как передать первоначальный ключ*** (для нашего протокола нам также понадобится IV – вектор инициализации) ***по незашифрованному первоначальному каналу и не дать атакующему его узнать*** (ведь среда считается общедоступной по умолчанию).

Эта проблема оставалась нерешенной достаточно долгое время. В 70-х годах стали появляться первые схемы ***асимметричного шифрования***, которые решали эту проблему. Идея была в том, чтобы использовать два разных ключа – один для шифрования (“публичный ключ” - его можно было смело отправлять кому угодно по любому незащищенному каналу) и один для расшифровки (“приватный ключ” - получение этого ключа кем-либо кроме его владельца влекло возможность расшифровывать любые сообщения). Аналогией (хоть и не совсем правдоподобной) может служить пример, когда кто-либо публикует инструкции о том, как сделать замок (из которых невозможно определить то, как сделать ключ к этому замку. Ключ же этот человек хранит только у себя. Таким образом, любой может закрыть (зашифровать) что-либо таким замком, открыть же это сможет лишь владелец ключа (приватного ключа).

Самыми известными асимметричными криптосистемами являются Diffie-Hellman и RSA (обе описаны в предыдущих главах). RSA также включает в себя алгоритмы подписи (подобные HMAC, но в данном случае подпись создается приватным ключом, а проверяется с помощью публичного), тогда как DH является лишь схемой обмена ключами. Для протокола была выбрана схема подписей из RSA и обмен ключами DH. Причины описаны в следующей главе.

Теперь мы можем обмениваться ключами (получить на стороне сервера и стороне клиента одинаковый “общий секрет”) с помощью DH и подписать наши первоначальные незашифрованные коммуникации с помощью RSA подписей. Энтропии полученного DH секрета достаточно, чтобы сгенерировать из него как ключ, так и вектор инициализации для AES.

Таким образом, мы можем начать с незащищенного соединения, постепенно переходя на полностью защищенное. В начале сервер отправит клиенту свой публичный ключ DH (подписанный по RSA схеме сервером для проверки на стороне клиента) и получит в ответ ключ клиента. После этого, можно получить из общего секрета DH наш ключ (для AES и HMAC) и вектор инициализации (для AES в CFB режиме). Такой набор “инструментов” позволяет защититься от всех базовых описанных атак. Тем не менее, для полной защиты требуется как теоретическая защищенность, так и правильная реализация на практике. Разработка протокола описана в следующей главе.

## 2.2. Разработка протокола передачи данных с учетом проведенного исследования

Протокол передачи данных разработан на основе SSL/TLS протоколов. Как и TLS, протокол будет являться надстройкой над TCP (TCP обеспечивает доставку сообщений, новый протокол – их защиту). Разработанный протокол более прост, нежели SSL и TLS (например, он не имеет возможности переключать алгоритм шифрования без переключения версии протокола). Для обеспечения конфиденциальности передаваемых данных было решено использовать RSA-2048 криптосистему для подписывания сообщений, DH-2048 для создания сеансового ключа, SHA-256 и HMAC-SHA256 для верификации и AES-256 для шифрования данных. Данный выбор был сделан исходя из достоинств этих алгоритмов, по сравнению с другими аналогами. Создание ключа для одного сеанса (и подписывание его с помощью постоянного приватного ключа RSA) обеспечивает “прямую секретность”: при потере приватного ключа RSA будут скомпрометированы лишь будущие соединения (если атакующему удалось захватить сессии, установленные до потери ключа, он не сможет их расшифровать). Будущие соединения в таком случае будут уязвимы лишь к атакам типа MITM.

RSA-2048 открытый и закрытый ключи имеет только сервер (т.к. RSA используется только для подписи). Сервер должен предоставлять открытый ключ RSA-2048 (и клиент должен уже иметь копию этого ключа перед началом соединения). Такая архитектура выбрана с целью простоты реализации, протокол обладает хорошей расширяемостью и может быть дополнен для аутентификации с обеих сторон.

Сообщения, передаваемые по протоколу имеют 3 основных поля:

1. Тип (Type) сообщения (1 байт). Числа описаны в десятичной системе.
2. Длина (Length) данных (2 байта в порядке от старшего байта к младшему) (необязательное поле).
3. Данные (Data) (блок байтов указанной в пункте 2 длины) (необязательное поле).

Для протокола версии 1 (0x01) доступны следующие типы сообщений:

1. Ack: Type=10 – сообщение подтверждения.
2. Close: Type=11 – сообщение о закрытии сессии.
3. Change cipher spec: 12 – сообщение о смене метода шифрования.
4. Client hello: Type=100, Length[2], Data(Список поддерживаемых протоколов, идентификаторы имеют длину в 1 байт) – сообщение отсылаемое клиентом в начале сессии.
5. Server hello: Type=101, Data(Протокол, выбранный из списка сообщения Client hello)[1].
6. Server DH begin: Type=110, Length[2], Data(открытый ключ DH-2048)[256], RSA signature Data [256] – сообщение, содержащее публичный ключ сервера для образования общего секрета по алгоритму DH-2048, подписанное личным ключом RSA сервера.
7. Client DH end: Type=111, Length[2], Data(открытый ключ DH-2048)[256] – сообщение, содержащее публичный ключ клиента для образования общего секрета по алгоритму DH-2048.
8. Data: Type=170, Length[2], Data[Length], HMAC(Data) – сообщение для передачи данных, подписывается HMAC-SHA256 для обеспечения защиты от подмены, данные в блоке Data шифруются по алгоритму AES-256 до расчета HMAC.

Создание безопасного соединения:

1. Клиент отправляет сообщение Client hello с поддерживаемыми протоколами.
2. Сервер отправляет Server hello с выбранным протоколом.
3. Клиент отправляет Ack – согласие на общение по этому протоколу.
4. Сервер отправляет Server DH begin со сгенерированным открытым ключом DH-2048 и RSA-2048 подписью.
5. Клиент проверяет подпись. Если подпись верна, Клиент генерирует свою пару ключей DH-2048 и может уже узнать общий секрет и создать ключ и вектор инициализации для AES-256.
6. Клиент отсылает Client DH end со сгенерированным открытым ключом DH-2048.
7. Сервер вычисляет общий секрет и создает ключ и вектор инициализации для AES-256.
8. Сервер отправляет Change cipher spec – указание перейти на шифрование по AES-256 с ключом, равным первым 32 байтам от общего секрета и вектором инициализации равным хешу SHA-256 от общего секрета.
9. Клиент отправляет сообщение Data, содержащие хеш SHA-256 от суммы всех предыдущих пересланных сообщений
10. Сервер проверяет HMAC-SHA256 и хеш SHA-256, присланный в сообщении. Если что-либо оказывается неверным сервер обрывает соединение.
11. Сервер отправляет сообщение Data, содержащие хеш SHA-256 от суммы всех предыдущих пересланных сообщений
12. Клиент проверяет HMAC-SHA256 и хеш SHA-256, присланный в сообщении. Если что-либо оказывается неверным сервер обрывает соединение.
13. Безопасное соединение установлено.

После того, как соединение установлено, все последующие сообщения должны быть типа Data или Close. Любое несовпадение длины сообщения или HMAC вызовет разрыв соединения.

Все сообщения описаны для версии протокола 1 (0x01 при передаче). При необходимости дополнения или изменения схемы коммуникации (изменение длин ключей или последовательности сообщений), следует изменить версию новой вариации. Это обеспечивает возможность любых изменений в протоколе без потери совместимости (если клиент выбирает протокол, не поддерживаемый сервером, соединение разрывается и эти программы не совместимы).

## 2.3 Анализ протокола на уязвимость к известным атакам

Рассмотрим несколько уже описанных атак с учетом разработанного протокола:

1. Network sniffing (прослушивание сети) – злоумышленник может прослушивать соединение до перехода на шифрование AES-256. Но использование протокола Диффи-Хелламана исключает возможность злоумышленника узнать симметричный ключ для AES. Вектор инициализации для AES получается из того же общего секрета DH детерминировано (на основе SHA-256).
2. Known-plaintext attack (атака “известного исходного текста”) – сервер всегда отправляет разный открытый ключ Диффи-Хеллмана (соответственно, ключ и вектор инициализации являются разными), тем самым заставляя шифрование по AES-256 в каждой сессии получать разный зашифрованный текст.
3. Replay (атака класса “повторной отправки”) – решается CFB режимом шифрования AES-256. В этом режиме два одинаковых сообщения, отправленные друг за другом, будут зашифрованы по-разному. Для каждого начального сообщения это обеспечивается разным вектором инициализации, а для всех последующих – вектор инициализации меняется после шифрования предидущих.
4. MITM (Man In The Middle, атака класса “человек посередине”) – решается подписью RSA, HMAC, хешем SHA-256 после Change Cipher Spec. Атакующий не может подписать неверный публичный ключ DH верной подписью, так как не обладает верным приватным ключом RSA.
5. Forward secrecy (прямая секретность) – если злоумышленник получил доступ к приватному RSA ключу и имеет записи соединений и переданных данных до этого момента, то он не сможет дешифровать эти данные т.к. все сессионные ключи лежат лишь в памяти программы. Когда соединение разрывается – ключ удаляется из памяти полностью. В случае утери приватного RSA ключа становится возможной атака типа MITM.
6. Downgrade cipher (атака “понижения версий”) – злоумышленник не может подменить содержимое пакета для понижения версии протокола (при наличии более ранних версий протоколов с подтвержденными уязвимостями это может значить взлом соединения еще до завершения переговоров о сессионном ключе) из-за использования хеша SHA-256 от всех переданных ранее сообщений после Change cipher spec.

Можно увидеть, что в данном протоколе существуют механизмы защиты от большинства известных атак. Естественно, нельзя гарантировать, что новые атаки не будут найдены в будущем.

# Глава 3: Разработка клиентского приложения

## 3.1. Описание дизайна системы, выбор инструментов

Для большей наглядности мы решили проектировать простейшую банковскую систему. Система пускает пользователя по логину и паролю и позволяет ему: смотреть состояние счета, снять или положить деньги на счет.

База данных банковской системы состоит всего из одной таблицы, хранящей логин, пароль и состояние счета пользователя.

Предполагаемый поток действий клиента:

1. Настройка соединения (IP адрес и порт сервера).
2. Установление безопасного соединения.
3. Логин в систему.
4. Операции со счетом.
5. Выход из системы.

Команды клиента для общения на уровне банковского приложения:

1. login <username> <password>; - сообщение от клиента для входа пользователя в систему.
2. balance alter [+-][number]; – сообщение от клиента для пополнения/снятия со счета. balance alter 0 – получение количества денег на счету.
3. disconnect; – сообщение завершение безопасного соединения.

Команды сервера для общения на уровне банковского приложения:

1. login: - предложение клиенту прислать логин и пароль
2. command: - предложение клиенту прислать команду
3. code [число] [data];– код ответа на запрос клиента. Код 0 – запрос прошел успешно. Код 1 – ошибка. data – причина в случае ошибки или состояние счета в случае просьбы на изменение/получение баланса.

Пример взаимодействия Клиента(C) и Сервера(S), комментарии обозначены двойными слешами(//):

1. C: login gear pwdpwd;
2. S: code 0;
3. S: command:
4. C: balance alter 0; // получение состояния счета.
5. S: code 0 101; // 101 денежная единица на счету.
6. C: balance alter -100; //снять 100 денежных единиц со счета.
7. S: code 0 1;// осталась одна денежная единица на счету.
8. C: balance alter -20;
9. S: code 1 недостаточно денег для операции на счету.
10. C: disconnect;

Инструменты для реализации:

1. Среда разработки на С++ QT Creator. QT Creator позволяет создавать кроссплатформенные приложения на С++ (Windows, UNIX, MacOS). Также QT обладает собственным фреймворком с различными полезными фичами. Позволяет разрабатывать юнит тесты.
2. Библиотека для шифрования Crypto++. Библиотека с открытым исходным кодом, позволяющая применять криптоалгоритмы на С++. Все алгоритмы протестированы, а на официальном сайте содержится документация и примеры.
3. Система контроля версий по технологий Git предоставляемой сервисом BitBucket. Используемый клиент для доступа к системе контроля версий – SourceTree.

## 3.2. Разработка клиентского приложения

Первой стадией разработки клиентского приложения является написание своей оболочки над библиотекой Crypto++ и ее тестирование. Это нужно для того, чтобы исключить прямое обращение клиента к библиотеке Crypto++ и локализовать будущие проблемы с Crypto++ в оболочке. В своей работе я создал оболочки на нужные нам для создания соединения криптоалгоритмы: SHA-256, HMAC-SHA256, RSA-2048, Diffie-Hellman-2048, AES-256. Разделил их на 4 класса, и каждый из них был протестирован с помощью юнит-тестов QT, т.к. нежелательно полностью доверять стороннему коду.

Краткий список тестов:

1. Шифрованное сообщение отличается от оригинального.
2. Восстановленное сообщение аналогично оригинальному.
3. Хеш, шифрование, общий секрет от разных входных данных – разный.
4. Две генерации ключей/хешей подряд генерируют разные объекты.

Следующая стадия – написание класса, который будет поднимать соединение по разработанному нами протоколу. Его задачи:

1. Поднять соединение по протоколу.
2. Отправлять шифрованные данные.
3. Получать и дешифровывать данные.
4. Валидация сообщений (проверка подписи, HMAC-SHA256, хеш SHA-256).
5. Генерация ключей.
6. Сообщение клиенту о статусе соединения.
7. Сообщение клиенту о получении нового сообщения
8. Логирование соединения.

Далее сам клиент. Клиент использует класс соединения в своих целях. Он не должен знать о шифровании, он просто должен уметь использовать класс соединения для подключения к серверу и отправки данных. Класс соединения отвечает за протокольный уровень обмена данными. Клиент отвечает за обмен данными на уровне приложений.

Задачи клиента:

1. Установка соединения с сервером через класс соединения.
2. Общение между сервером и клиентом по протоколу общения банковской системы.
3. Логирование.
4. Вывод данных для пользователя и прием команд от пользователя.

Класс соединения должен работать в отдельном потоке, чтобы клиент и соединение не блокировали работу друг друга. Для организации передачи данных между клиентом и соединением пользуемся библиотекой QT и ее концептом слотов и сигналов. Суть концепта: каждый объект QT может отправлять сигналы. Сигналы могут быть связаны с любым количеством слотов, при этом слот, по сути, представляет собой функцию для обработки сигнала, принимающий в свои аргументы аргументы сигнала. Обработка сигналов стоит в очереди событий EventLoop.

## 3.3. Оценка получившегося решения

Был разработан прототип клиента банковской системы, который устанавливает безопасное соединение с прототипов сервера по разработанному протоколу. Общее время становления безопасного соединения в пределах города составило около 10 миллисекунд, что не слишком существенно для клиента (в плане разницы между 10 миллисекундами и секундой), но весьма существенно для сервера, который должен открывать и держать открытыми множество соединений.

Анализ протокола показал, что протокол устойчив к известным атакам. Все конфиденциальные данные передаются в зашифрованном виде, сторонние пользователи общедоступного канала лишены возможности расшифровать пересылаемые сообщения за доступное время.

# Заключение

## 1. Общая оценка работы

Исходный код клиентского и серверного приложения, спецификация протокола и все дополнительные данные находятся в свободном доступе в Git репозитории проекта по адресу: https://bitbucket.org/gear54rus/bank-system.

## 2. Полнота решения поставленных задач 3. Экономическая и научная значимость работы

# Список использованных источников

1. <http://en.wikipedia.org/wiki/RSA_(cryptosystem)>
2. <http://ru.wikipedia.org/wiki/RSA>
3. <http://bibliofond.ru/view.aspx?id=550653>
4. <http://ru.wikipedia.org/wiki/Эллиптическая_криптография>
5. <http://habrahabr.ru/post/212235/>
6. <http://ru.wikipedia.org/wiki/AES>
7. <http://ru.wikipedia.org/wiki/SHA-2>
8. <http://ru.wikipedia.org/wiki/MD5>
9. <http://ru.wikipedia.org/wiki/HMAC>
10. <http://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана>
11. <http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange>
12. <http://ru.wikipedia.org/wiki/SSL>
13. <http://ru.wikipedia.org/wiki/TLS>

# Приложения