Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра ПМиК

Расчетно-графическое задание

«Клиент-серверное приложение с использованием поточного шифрования»

Выполнил:

ст. гр. МГ-172

Тищенко И.В.

Проверил:

зав. каф. ПМиК

профессор

Фионов А.Н.

Новосибирск, 2018

# ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc515189853)

[1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 4](#_Toc515189854)

[1.1 Протокол MQV 4](#_Toc515189855)

[1.2 Потоковый шифр SALSA20 5](#_Toc515189856)

[2 СТРУКТУРНАЯ СХЕМА РАБОТЫ ПРОГРАММЫ 7](#_Toc515189857)

[3 ОПИСАНИЯ ФУНКЦИЙ ПРОГРАММЫ 8](#_Toc515189858)

[4 ЛИСТИНГ 9](#_Toc515189859)

[5 ПРИМЕР РАБОТЫ ПРОГРАММЫ 10](#_Toc515189860)

[ЗАКЛЮЧЕНИЕ 11](#_Toc515189861)

# ВВЕДЕНИЕ

Целью данной работы является разработка клиент серверного приложения с использованием поточного шифрования. Долгосрочные ключи и сеансовые вырабатываются с использованием протокола MQV, а в качестве протокола поточного шифрования выбран SALSA20.

# 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## Протокол MQV

Одним из наиболее обоснованных протоколов формирования общего ключа типа Диффи-Хеллмана является протокол MQV.

MQV – протокол распределения ключей, поддерживающий взаимную аутентификацию сторон и тем самым устраняющий уязвимость к атаке «человек посередине», присущей классическому Диффи-Хеллману. Помимо прочего, для аутентификации пользователей не используется никакая вспомогательная информация. Ниже будет приведен алгоритм выработки общего ключа.

Пусть имеются 2 пользователя: Алиса и Боб. Алиса и Боб имеют каждый свою ключевую пару, состоящие из открытых и закрытых ключей: image и image. Разумеется, Бобу известен открытый ключ Алисы, а той в свою очередь известен открытый ключ Боба.

Далее, Алиса и Боб генерируют сеансовую пару ключей: image и image.

Затем происходит обмен как в классическом протоколе Диффи-Хеллмана: Алиса к Бобу: image, Боб к Алисе: image.

Теперь Алиса знает: A, B, C, D, a, image. А Бобу известны: A, B, C, D, b, image.

Чтобы получить общий ключ K они должны проделать следующие операции:

Алиса:

Выбирает число l, равное размеру сообщения в битах деленному на 2. Так если используется EC-MQV и длина сообщения равна 160 бит, то l=80.

* Задает image
* Находит image
* Задает image
* Вычисляет image
* Находит image
* Вычисляет image

Боб проделывает те же действия, но со своими закрытыми ключами:

* Задает image
* Находит image
* Задает image
* Вычисляет image
* Находит image
* Вычисляет image

Получившиеся в результате вычислений числа image и есть общий секретный ключ.

## Потоковый шифр SALSA20

Блочный алгоритм предназначен для шифрования блоков определенной длины. Однако может возникнуть необходимость шифрования данных не блоками, а, например, по символам. Поточный шифр (stream cipher) выполняет преобразование входного сообщения по одному биту (или байту) за операцию. Поточный алгоритм шифрования устраняет необходимость разбивать сообщение на целое число блоков достаточно большой длины, следовательно, он может работать в реальном времени. Таким образом, если передается поток символов, каждый символ может шифроваться и передаваться сразу.

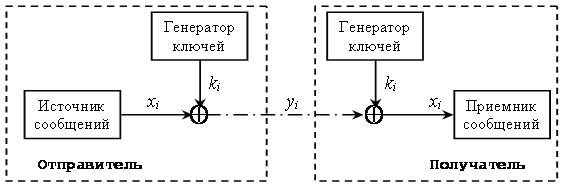


Рисунок 1 - Принцип работы поточного шифра

Генератор ключей выдает поток битов ki, которые будут использоваться в качестве гаммы. Источник сообщений генерирует биты открытого текста хi, которые складываются по модулю 2 с гаммой, в результате чего получаются биты зашифрованного сообщения уi:

y_i = x_i \oplus k_i, i=1,2,â¦,n

Чтобы из шифротекста y1, y2, …, yn восстановить сообщение x1, x2, …, xn, необходимо сгенерировать точно такую же ключевую последовательность k1, k2, …, kn, что и при шифровании, и использовать для расшифрования формулу:

x_i = y_i \oplus k_i,\  i=1,2,â¦,n


Обычно исходное сообщение и ключевая последовательность представляют собой независимые потоки бит. Таким образом, так как шифрующее (и расшифрующее) преобразование для всех поточных шифров одно и то же, они должны различаться только способом построения генераторов ключей. Получается, что безопасность системы полностью зависит от свойств генератора потока ключей. Если генератор потока ключей выдает последовательность, состоящую только из одних нулей (или из одних единиц), то зашифрованное сообщение будет в точности таким же, как и исходный поток битов (в случае единичных ключей зашифрованное сообщение будет инверсией исходного). Если в качестве гаммы используется один символ, представленный, например, восемью битами, то хотя зашифрованное сообщение и будет внешне отличаться от исходного, безопасность системы будет очень низкой. В этом случае при многократном повторении кода ключа по всей длине текста существует опасность его раскрытия статистическим методом.

**Salsa20** — система поточного шифрования, разработанная Даниэлем Бернштейном. Алгоритм был представлен на конкурсе «eSTREAM», целью которого было создание европейских стандартов для шифрования данных, передаваемых почтовыми системами. Алгоритм стал победителем конкурса в первом профиле (поточные шифры для программного применения с большой пропускной способностью).

# СТРУКТУРНАЯ СХЕМА РАБОТЫ ПРОГРАММЫ

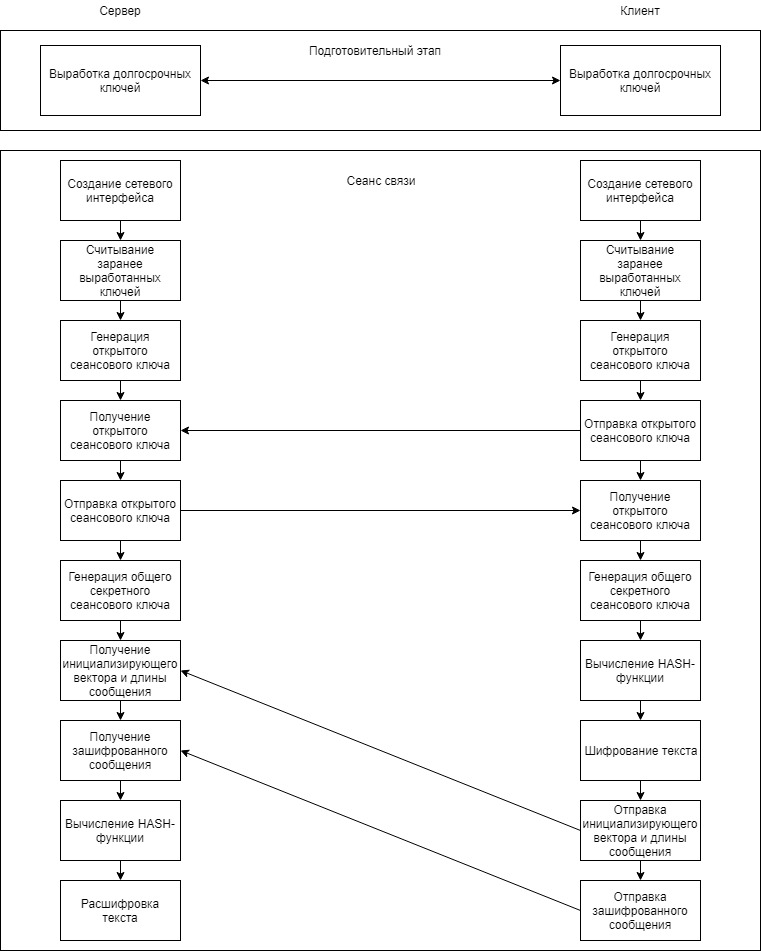


Рисунок 2 - Структурная схема

# ОПИСАНИЯ ФУНКЦИЙ ПРОГРАММЫ

Подготовительный этап

// Поиск генератора циклической подгруппы и выбор числа р для протокола MQV

gen\_p\_q\_g\_t(1024, 256, &p, &q, &g, &t);

// Генерация долговременного секретного ключа для пользователей А и В

gen\_a\_b(q, &a, &b);

// Генерация долговременного открытого ключа для пользователей А и В

gen\_A\_B(p, g, a, b, &A, &B);

Сеансовый этап (клиент)

// Считывание данных для пользователя B, полученные на первом шаге

get\_data(&p, &q, &g, &b, &A, &B);

// Генерация открыто сеансового ключа

gen\_Y(p, g, q, &y, &Y);

// Вычисление числа секретного сеансового ключа

gen\_sb(p, X, Y, A, b, y, q, &sb);

// Инициализация структуры состояния генератора

ECRYPT\_init();

// Установка секретного ключа в структуру

ECRYPT\_keysetup(&ctx, key, 128, 128);

// Установка инициализирующего вектора

ECRYPT\_ivsetup(&ctx, iv);

// Шифрование

ECRYPT\_encrypt\_bytes(&ctx, plaintext, ciphertext, (num\_symbols + 1));

Сеансовый этап (сервер)

// Считывание данных для пользователя A, полученные на первом шаге

get\_data(&p, &g, &q, &a, &A, &B);

// Генерация открыто сеансового ключа

gen\_X(p, g, q, &x, &X);

// Вычисление числа секретного сеансового ключа

gen\_sa(p, X, Y, B, a, x, q, &sa);

// Инициализация структуры состояния генератора

ECRYPT\_init();

// Установка секретного ключа в структуру

ECRYPT\_keysetup(&ctx, key, 128, 128);

// Установка инициализирующего вектора

ECRYPT\_ivsetup(&ctx, iv);

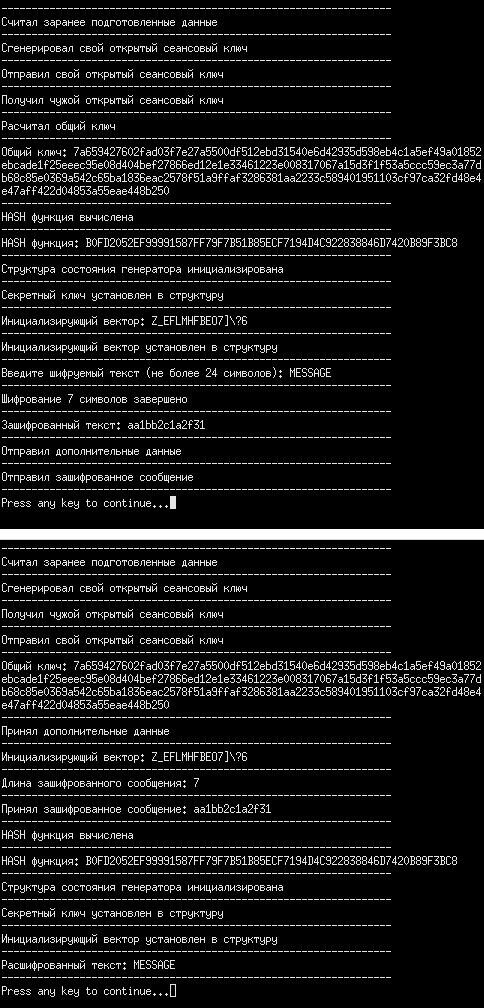
// Расшифровка

ECRYPT\_encrypt\_bytes(&ctx, ciphertext, plaintext, num\_symbols + 1);

# ЛИСТИНГ

С полным листингом программы можно ознакомиться на сайте по адресу https://github.com/VANTARR/2\_semester/tree/master/ПТЗИ.

# ПРИМЕР РАБОТЫ ПРОГРАММЫ



# ЗАКЛЮЧЕНИЕ

В процессе выполнения данной работы я познакомился с основами криптографии. При написании программы мною были получены навыки работы с такими протоколами как: протокол формирования общего ключа MQV и протокол потокового шифрования SALSA20.