МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский технологический университет»

МИРЭА

Институт кибернетики

Кафедра информационной безопасности

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по дисциплине

«Криптографические протоколы»

На тему:

«Реализация криптографического протокола. Протокол защиты сетевого трафика TLS»

Подготовил

студент группы ККСО–01–14 А.С. Першин

Руководитель работы

А.П. Никитин

Москва, 2019

Оглавление

1. Описание

TLS и SSL упоминаются в последнее время все чаще и чаще, более актуальным становится использование цифровых сертификатов, и даже появились компании, готовые бесплатно предоставлять цифровые сертификаты всем желающим, чтобы гарантировать шифрование трафика между посещаемыми сайтами и браузером клиента. Нужно это, естественно, для безопасности, чтобы никто в сети не мог получить данные, которые передаются от клиента серверу и обратно.

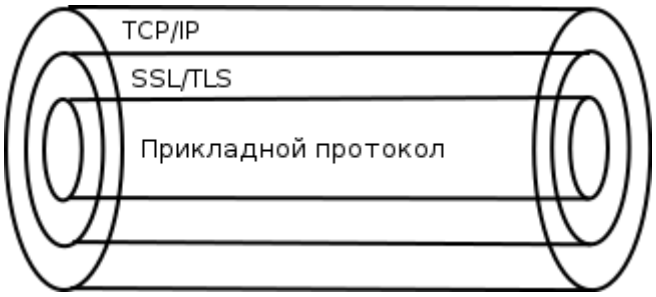
SSL — Secure Socket Layer, уровень защищенных сокетов. TLS — Transport Layer Security, безопасность транспортного уровня. SSL является более ранней системой, TLS появился позднее, он основан на спецификации SSL 3.0, разработанной компанией Netscape Communications. Тем не менее, задача у этих протоколов одна — обеспечение защищенной передачи данных между двумя компьютерами в сети Интернет.

Безопасная передача обеспечивается при помощи аутентификации и шифрования передаваемой информации. По сути эти протоколы, TLS и SSL, работают одинаково, принципиальных различий нет. TLS, можно сказать, является преемником SSL, хотя они и могут использоваться одновременно, причем даже на одном и том же сервере. Такая поддержка необходима для того, чтобы обеспечить работу как с новыми клиентами (устройствами и браузерами), так и с устаревшими, которые TLS не поддерживают. Последовательность возникновения этих протоколов выглядит вот так:

SSL 1.0 — никогда не публиковался
SSL 2.0 — февраль 1995 года
SSL 3.0 — 1996 год
TLS 1.0 — январь 1999 года
TLS 1.1 — апрель 2006 года
TLS 1.2 — август 2008 года

2. Принцип работы протокола SSL/TLS

Принцип работы SSL/TLS следующий. Поверх протокола TCP/IP устанавливается зашифрованный канал, внутри которого передаются данные по прикладному протоколу — HTTP, FTP, и так далее. Вот как это можно представить графически:



Прикладной протокол «заворачивается» в TLS/SSL, а тот в свою очередь в TCP/IP. По сути данные по прикладному протоколу передаются по TCP/IP, но они зашифрованы. И расшифровать передаваемые данные могут только те машины, которые установили соединение. Для всех остальных, кто получит передаваемые пакеты, эта информация будет бессмысленной, если они не смогут ее расшифровать.

Установка соединения обеспечивается в несколько этапов:

1) Клиент устанавливает соединение с сервером и запрашивает защищенное подключение. Это может обеспечиваться либо установлением соединения на порт, который изначально предназначен для работы с SSL/TLS, например, 443.

2) При установке соединения клиент предоставляет список алгоритмов шифрования, которые он «знает». Сервер сверяет полученный список со списком алгоритмов, которые «знает» сам сервер, и выбирает наиболее надежный алгоритм, после чего сообщает клиенту, какой алгоритм использовать

3) Сервер отправляет клиенту свой цифровой сертификат, подписанный удостоверяющим центром, и открытый ключ сервера.

4) Клиент может связаться с сервером доверенного центра сертификации, который подписал сертификат сервера, и проверить, валиден ли сертификат сервера. Но может и не связываться. В браузерах обычно уже установлены корневые сертификаты центров сертификации, с которыми сверяют подписи серверных сертификатов.

5) Генерируется сеансовый ключ для защищенного соединения. Это делается следующим образом:

— Клиент генерирует случайную цифровую последовательность

— Клиент шифрует ее открытым ключом сервера и посылает результат на сервер
— Сервер расшифровывает полученную последовательность при помощи закрытого ключа

Учитывая, что алгоритм шифрования является асимметричным, расшифровать последовательность может только сервер. При использовании асимметричного шифрования используется два ключа — приватный и публичный. Публичным отправляемое сообщение шифруется, а приватным расшифровывается. Расшифровать сообщение, имея публичный, ключ нельзя.

6) В новой версии SSL/TLS протоколе TLS используются алгоритмы для выработки общего ключа для более быстрого симметричного шифрования.

7) Таким образом устанавливается зашифрованное соединение. Данные, передаваемые по нему, зашифровываются и расшифровываются до тех пор, пока соединение не будет разорвано.

3. Реализация протокола на основе принципов SSL/TLS

Пусть в соединении участвуют три стороны:

- CA – Certificate Authority Server
- Alice – User1
- Bob – User2

Ставится задача: необходимо установить защищенное соединение между пользователями Alice и Bob.

В свою очередь CA выполняет следующие функции:

- получение письменных заявок (с указанием идентификационных данных) от пользователей
- подпись публичных ключей асимметричного шифрования пользователей $sign_{hash_{PublicKey_{User}}}$
- хранение хеш-значения $hash_{PublicKey_{User}}$ публичного ключа пользователя и его заявления

Пользователи имеют в распоряжении следующие алгоритмы:

- RSA – для установления начального соединения, где шифрование описывается как $RSA_E(key, data)$, а расшифрование $RSA_D(key, data)$
- SHA512 – для вычисления хэш-значения параметров, где вычисление хеш-значения обозначается: $SHA512(data) = hash_{data}$

- ECDHE – протокол для выработки симметричного ключа шифрования (на основе Curve GOST256)
- ГПСЧ – для выработки случайных чисел (на основе AES256-OFB)
- ЦП (на основе Curve GOST256)

Иллюстрация работы протокола выглядит следующим образом:

Alice	CA	Bob
Имеет: 1. $PublicKey_{CA}$	Имеет: 1. $PrivateKey_{CA}$ 2. $PublicKey_{CA}$	Имеет: 1. $PublicKey_{CA}$
Используя RSA генерирует: 1. $PrivateKey_{Alice}$ 2. $PublicKey_{Alice}$		Используя RSA генерирует: 1. $PrivateKey_{Bob}$ 2. $PublicKey_{Bob}$

Шаг 1

Вычисляет:

$SHA512(PublicKey_{Alice}) =$
$= hash_{PublicKey_{Alice}}$

Вычисляет:

$SHA512(PublicKey_{Bob}) =$
$= hash_{PublicKey_{Bob}}$

$hash_{PublicKey_{Alice}}$ →

← $hash_{PublicKey_{Bob}}$

Вычисляет:

$RSA_E(PrivateKey_{CA}, hash_{PublicKey_{Alice}}) =$
$= sign_{hash_{PublicKey_{Alice}}}$

$RSA_E(PrivateKey_{CA}, hash_{PublicKey_{Bob}}) =$
$= sign_{hash_{PublicKey_{Bob}}}$

← $sign_{hash_{PublicKey_{Alice}}}$

$sign_{hash_{PublicKey_{Bob}}}$ →

Вычисляет:

$RSA_D(PublicKey_{CA}, sign_{hash_{PublicKey_{Alice}}}) \overset{?}{=}$
$\overset{?}{=} hash_{PublicKey_{Alice}}$

если да, имеет: $sign_{hash_{PublicKey_{Alice}}}$

если нет, то подделка подписи ключа, повторяем шаги начиная с генерации

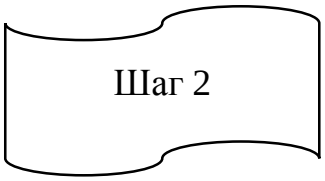
Вычисляет:

$RSA_D(PublicKey_{CA}, sign_{hash_{PublicKey_{Bob}}}) \overset{?}{=}$
$\overset{?}{=} hash_{PublicKey_{Bob}}$

если да, имеет: $sign_{hash_{PublicKey_{Bob}}}$

если нет, то подделка подписи ключа, повторяем шаги начиная с генерации

Примечание: на «Шаг 1» CA получает письменные заявки с указанием идентификационных данных заявителя. Получение подписи $sign_{hash_{PublicKey_{User}}}$ происходит в «письменном» порядке, $hash_{PublicKey_{User}}$ хранится в CA вместе с заявлением.



Имеет:

1. $PublicKey_{CA}$
2. $PrivateKey_{Alice}$
3. $PublicKey_{Alice}$
4. $sign_{hash_{PublicKey_{Alice}}}$

Имеет:

1. $PublicKey_{CA}$
2. $PrivateKey_{Bob}$
3. $PublicKey_{Bob}$
4. $sign_{hash_{PublicKey_{Bob}}}$

$sign_{hash_{PublicKey_{Alice}}}, PublicKey_{Alice}$ →

← $sign_{hash_{PublicKey_{Bob}}}, PublicKey_{Bob}$

Вычисляет:

$SHA512(PublicKey_{Bob}) = hash_{PublicKey_{Bob}}$

$RSA_D(PublicKey_{CA}, sign_{hash_{PublicKey_{Bob}}}) \overset{?}{=} \overset{?}{=} hash_{PublicKey_{Bob}}$

если да, имеет: $PublicKey_{Bob}$

если нет, то идет подмена (при поиске злоумышленника, он – в базе CA, т.к. $sign_{hash_{PublicKey_{User}}}$ производится на $PrivateKey_{CA}$, а все данные об обратившихся пользователях хранятся в CA)

Вычисляет:

$SHA512(PublicKey_{Alice}) = hash_{PublicKey_{Alice}}$

$RSA_D(PublicKey_{CA}, sign_{hash_{PublicKey_{Alice}}}) \overset{?}{=} \overset{?}{=} hash_{PublicKey_{Alice}}$

если да, имеет: $PublicKey_{Bob}$

если нет, то идет подмена (при поиске злоумышленника, он – в базе CA, т.к. $sign_{hash_{PublicKey_{User}}}$ производится на $PrivateKey_{CA}$, а все данные об обратившихся пользователях хранятся в CA)

Таким образом, после выполнения 2 шага пользователи обменялись публичными ключами. Теперь необходимо выработать общий сеансовый симметричный ключ для более быстрого обмена информацией. Для этого воспользуемся алгоритмом Диффи-Хеллмана на эллиптических кривых (ECDHE).

Шаг 3

Имеет:

1. $\mathrm{Pr}ivateKey_{Alice}$
2. $\mathrm{P}ublicKey_{Alice}$
3. $\mathrm{P}ublicKey_{Bob}$

Вычисляет:

вырабатывает случайное число

1. $ГПСЧ \rightarrow d_{Alice} (\mathrm{mod}\, n)$
2. $d_{Alice} \times G^{x,y} = Q^{x,y}{}_{Alice} = (Q^{x}{}_{Alice}, Q^{y}{}_{Alice})$
3. $RSA_E(PublicKey_{Bob}, Q^{x,y}{}_{Alice}) =$
$= Enc_{RSA}Q^{x,y}{}_{Alice} = (EncQ_{RSA}{}^{x}{}_{Alice}, Enc_{RSA}Q^{y}{}_{Alice})$

Имеет:

1. $\mathrm{Pr}ivateKey_{Bob}$
2. $\mathrm{P}ublicKey_{Bob}$
3. $\mathrm{P}ublicKey_{Alice}$

Вычисляет:

вырабатывает случайное число

1. $ГПСЧ \rightarrow d_{Bob} (\mathrm{mod}\, n)$
2. $d_{Bob} \times G^{x,y} = Q^{x,y}{}_{Bob} = (Q^{x}{}_{Bob}, Q^{y}{}_{Bob})$
3. $RSA_E(PublicKey_{Alice}, Q^{x,y}{}_{Bob}) =$
$= Enc_{RSA}Q^{x,y}{}_{Bob} = (Enc_{RSA}Q^{x}{}_{Bob}, Enc_{RSA}Q^{y}{}_{Bob})$

$Enc_{RSA}Q^{x}{}_{Alice}, Enc_{RSA}Q^{y}{}_{Alice}$ →

← $Enc_{RSA}Q^{x}{}_{Bob}, Enc_{RSA}Q^{y}{}_{Bob}$

Вычисляет:

1. $RSA_D(\mathrm{Pr}ivateKey_{Alice}, Enc_{RSA}Q^{x,y}{}_{Bob}) =$
$= Q^{x,y}{}_{Bob} = (Q^{x}{}_{Bob}, Q^{y}{}_{Bob})$

Вычисляет:

1. $RSA_D(\mathrm{Pr}ivateKey_{Bob}, Enc_{RSA}Q^{x,y}{}_{Alice}) =$
$= Q^{x,y}{}_{Alice} = (Q^{x}{}_{Alice}, Q^{y}{}_{Alice})$

Имеет:

1. $\mathrm{Pr}ivateKey_{Alice}$
2. $\mathrm{P}ublicKey_{Alice}$
3. $\mathrm{P}ublicKey_{Bob}$
4. d_{Alice}
5. $Q^{x,y}{}_{Bob} = (Q^{x}{}_{Bob}, Q^{y}{}_{Bob})$

Имеет:

1. $\mathrm{Pr}ivateKey_{Bob}$
2. $\mathrm{P}ublicKey_{Bob}$
3. $\mathrm{P}ublicKey_{Alice}$
4. d_{Bob}
5. $Q^{x,y}{}_{Alice} = (Q^{x}{}_{Alice}, Q^{y}{}_{Alice})$

Вычисляет:

2. $d_{Alice} \times Q^{x,y}{}_{Bob} = \mathrm{Secret}^{x,y}{}_{Alice} = (\mathrm{Secret}^{x}{}_{Alice}, \mathrm{Secret}^{y}{}_{Alice})$
3. $SHA512(\mathrm{Secret}^{x}{}_{Alice}) = hash_{\mathrm{Secret}^{x}{}_{Alice}} = SessionKey$
4. $\mathrm{SHA512}(\mathrm{Secret}^{y}{}_{Alice}) = hash_{\mathrm{Secret}^{y}{}_{Alice}}$
5. $\mathrm{AES}_E(SessionKey, hash_{\mathrm{Secret}^{y}{}_{Alice}}) = Enc_{AES}hash_{\mathrm{Secret}^{y}{}_{Alice}}$

Вычисляет:

2. $d_{Bob} \times Q^{x,y}{}_{Alice} = \mathrm{Secret}^{x,y}{}_{Bob} = (\mathrm{Secret}^{x}{}_{Bob}, \mathrm{Secret}^{y}{}_{Bob})$
3. $SHA512(\mathrm{Secret}^{x}{}_{Bob}) = hash_{\mathrm{Secret}^{x}{}_{Bob}} = SessionKey$
4. $\mathrm{SHA512}(\mathrm{Secret}^{y}{}_{Bob}) = hash_{\mathrm{Secret}^{y}{}_{Bob}}$
5. $\mathrm{AES}_E(SessionKey, hash_{\mathrm{Secret}^{y}{}_{Bob}}) = Enc_{AES}hash_{\mathrm{Secret}^{y}{}_{Bob}}$

$Enc_{AES}hash_{\mathrm{Secret}^{y}{}_{Alice}}$ →

← $Enc_{AES}hash_{\mathrm{Secret}^{y}{}_{Bob}}$

Имеет:

1. $PrivateKey_{Alice}$
2. $PublicKey_{Alice}$
3. $PublicKey_{Bob}$
4. d_{Alice}
5. $Q^{x,y}{}_{Bob} = (Q^{x}{}_{Bob}, Q^{y}{}_{Bob})$
6. *SessionKey*
7. $hash_{\mathrm{Secret}^{y}{}_{Alice}}$

Вычисляет:

$$AES_D(SessionKey, Enc_{AES} hash_{\mathrm{Secret}^{y}{}_{Bob}}) \overset{?}{=}$$
$$\overset{?}{=} hash_{\mathrm{Secret}^{y}{}_{Alice}}$$

если да, имеет: общий с Bob'ом симметричный *SessionKey*

если нет, то идет подмена *SessionKey*, повторить «Шаг 3»

Имеет:

1. $PrivateKey_{Alice}$
2. $PublicKey_{Alice}$
3. $PublicKey_{Bob}$
4. d_{Alice}
5. $Q^{x,y}{}_{Bob} = (Q^{x}{}_{Bob}, Q^{y}{}_{Bob})$
6. *SessionKey*
7. $hash_{\mathrm{Secret}^{y}{}_{Bob}}$

Вычисляет:

$$AES_D(SessionKey, Enc_{AES} hash_{\mathrm{Secret}^{y}{}_{Alice}}) \overset{?}{=}$$
$$\overset{?}{=} hash_{\mathrm{Secret}^{y}{}_{Bob}}$$

если да, имеет: общий с Alice симметричный *SessionKey*

если нет, то идет подмена *SessionKey*, повторить «Шаг 3»

Шаг 4

Имеет:

1. *SessionKey*
2. $Message_{Alice}$

Вычисляет:

1. $ГПСЧ \xrightarrow{\text{вырабатывает случайное число}} \mathrm{Secret}KeyDS \pmod{n}$
2. $KeyCheckDS_{Alice} = CreateKeyCheckDS(\mathrm{Secret}KeyDS)$
3. $DS_{Message_{Alice}} = CreateDS(\mathrm{Secret}KeyDS, Message_{Alice})$
4. $AES_E(SessionKey, Message_{Alice}) = Enc_{AES} Message_{Alice}$

Имеет:

1. *SessionKey*
2. $Message_{Bob}$

Вычисляет:

1. $ГПСЧ \xrightarrow{\text{вырабатывает случайное число}} \mathrm{Secret}KeyDS \pmod{n}$
2. $KeyCheckDS_{Bob} = CreateKeyCheckDS(\mathrm{Secret}KeyDS)$
3. $DS_{Message_{Bob}} = CreateDS(\mathrm{Secret}KeyDS, Message_{Bob})$
4. $AES_E(SessionKey, Message_{Bob}) = Enc_{AES} Message_{Bob}$

$Enc_{AES}Message_{Alice}, KeyCheckDS_{Alice}, DS_{Message_{Alice}}$ →

← $Enc_{AES}Message_{Bob}, KeyCheckDS_{Bob}, DS_{Message_{Bob}}$

Вычисляет:

1. $AES_D(SessionKey, Enc_{AES}Message_{Bob}) =$
$= Message_{Bob}$

2. $result =$
$= CheckDS(DS_{Message_{Bob}}, Message_{Bob}, KeyCheckDS_{Bob})$

если result = false, то:

$ГПСЧ \xrightarrow{80байт(IV+HASH)} Answer_{Alice}$

если result = true, то:

1. $SHA512(Message_{Bob}) =$
$= hash_{Message_{Bob}}$

2. $AES_E(SessionKey, hash_{Message_{Bob}}) =$
$= Answer_{Alice}$

Вычисляет:

1. $AES_D(SessionKey, Enc_{AES}Message_{Alice}) =$
$= Message_{Alice}$

2. $result =$
$= CheckDS(DS_{Message_{Alice}}, Message_{Alice}, KeyCheckDS_{Alice})$

если result = false, то:

$ГПСЧ \xrightarrow{80байт(IV+HASH)} Answer_{Bob}$

если result = true, то:

1. $SHA512(Message_{Alice}) =$
$= hash_{Message_{Alice}}$

2. $AES_E(SessionKey, hash_{Message_{Alice}}) =$
$= Answer_{Bob}$

$Answer_{Alice}$ →

← $Answer_{Bob}$

Вычисляет:

$AES_D(SessionKey, Answer_{Bob}) \stackrel{?}{=}$
$\stackrel{?}{=} hash_{Message_{Alice}}$

если да, то сообщение передалось успешно

если нет, то сообщение не передалось, повторить «Шаг 4»

Вычисляет:

$AES_D(SessionKey, Answer_{Alice}) \stackrel{?}{=}$
$\stackrel{?}{=} hash_{Message_{Bob}}$

если да, то сообщение передалось успешно

если нет, то сообщение не передалось, повторить «Шаг 4»

4. Свойства, характеризующие безопасность протокола

Свойства, характеризующие безопасность протоколов:

1) Аутентификация (не широковещательная)
   - G1 (аутентификация субъекта);
   - G2 (аутентификация сообщения);
   - G3 (защита от повтора)
2) Аутентификация при рассылке по многим адреса
   - G4 (неявная скрытая аутентификация получателя);
   - G5 (аутентификация источника)
3) Авторизация 3-ей доверенной стороной
   - G6 (авторизация 3-ей доверенной стороной);
4) Свойства совместной генерации ключа
   - G7 (аутентификация ключа);
   - G8 (подтверждение правильности ключа);
   - G9 (защита от чтения назад);
   - G10 (формирование новых ключей);
   - G11 (защита от возможности договориться о параметрах безопасности)
5) Конфиденциальность
   - G12 (конфиденциальность)
6) Анонимность
   - G13 (защита идентификатора от прослушивания);
   - G14 (защита идентификатора от других участников)
7) Защита от отказа в обслуживании
   - G15 (защита то DDoS);
8) Инвариантность
   - G16 (инвариантность отправителя)
9) Невозможность отказа от ранее совершенных действий
   - G17 (подотчетность);
   - G18 (доказательство источника);
   - G19 (доказательство получателя)
10) Временное свойство
   - G20 (безопасное временное свойство)

Данному протоколу присущи следующие свойства: G1, G2, G3, G6, G7, G8, G9, G10, G11, G12, G13, G14, G16, G17, G18, G19, G20.

5. Результаты реализации протокола

Разработка производилась в IDE Microsoft Visual Studio 15 Pro. Для реализации задания лабораторной работы было создано общее решение с именем CryptoProtocols. Реализация протокола входит в главный проект CryptoProtocols решения CryptoProtocols. В главный проект были подключены проекты AES_BlocksCipher, CSPRNG, ECDSA, SHA512_Hash, которые собираются в подключаемые статические библиотеки и реализуют: блочный шифр, ГПСЧ, ЭЦП, Хеш-функцию. Реализация ассиметричного шифрования была взята из библиотеки OpenSSL.

Результат выполнения тест кейсов для проверки корректности работы протокола и фиксации времени выполнения для подсчета производительности работы приведены на Рис. 1.

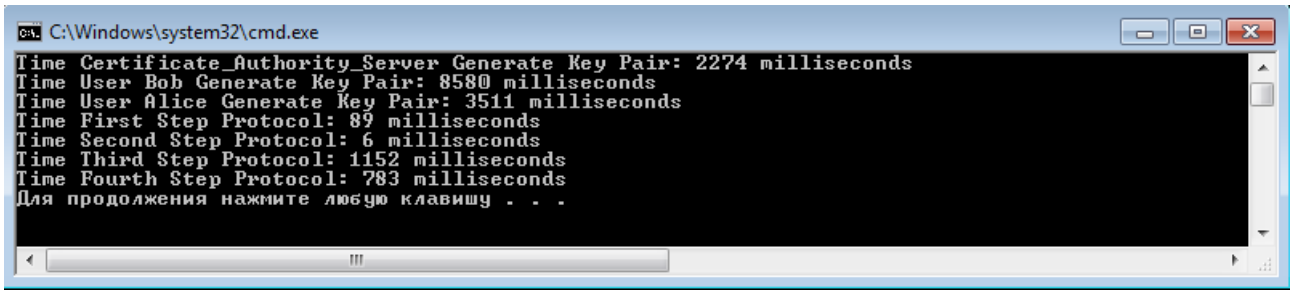


Рис. 1. Результат тестирования реализованного протокола

Запускался тест на ЦП AMD A6-3410MX (4 ядра, 4 потока) на Рис.2.

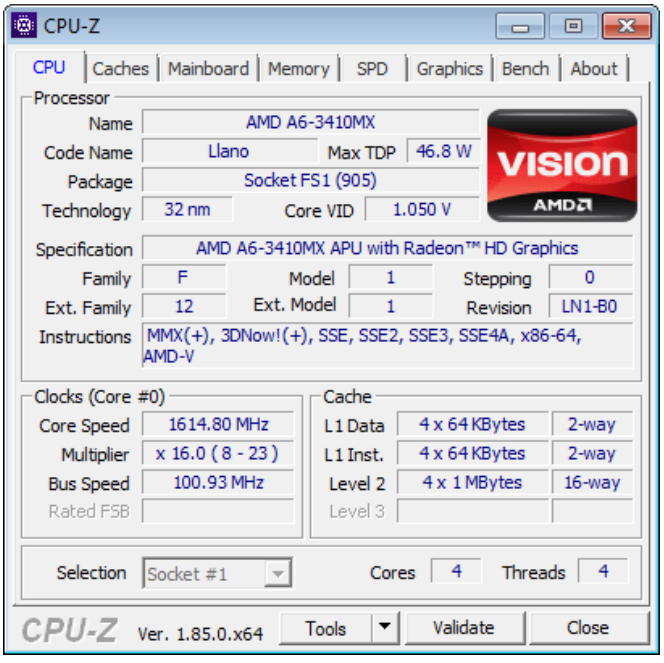


Рис. 2. ЦП AMD A6-3410MX (4 ядра, 4 потока)

Таким образом, общее время установки соединения (без времени выработки ключей ассиметричного шифрования) составляет ~1,3 секунды, а обмена

сообщениями ~ 0,4 секунды для каждой из сторон, что подтверждает быструю работы данного протокола.

Табл. 1. Скорость выполнения этапов протокола

Пользователь	Операция	Время выполнения [секунд]
Генерация ассиметричной ключевой пары (выполняется до начала протокола)		
CA	Выработка ключевой пары	2,274
Alice	Выработка ключевой пары	3,511
Bob	Выработка ключевой пары	8,580
Шаг 1 «Получение подписанных CA ключей пользователей»		
Alice/Bob	Получение подписанных CA ключей пользователей	0,089
Шаг 2 «Обмен публичными ключами между собеседниками»		
Alice/Bob	Обмен публичными ключами между собеседниками	0,006
Шаг 3 «Генерация сеансового ключа собеседников»		
Alice/Bob	Генерация сеансового ключа собеседников	1,152
Шаг 4 «Обмен сообщениями между собеседниками»		
Alice/Bob	Обмен сообщениями между собеседниками	0,783

Литература

1. «Протокол Диффи — Хеллмана на эллиптических кривых» [Интернет ресурс], ссылка: https://ru.wikipedia.org/wiki/Протокол_Диффи_—_Хеллмана_на_эллиптических_кривых
2. «TLS и SSL: Необходимый минимум знаний» [Интернет ресурс], ссылка: https://mnorin.com/tls-ssl-neobhodimy-j-minimum-znanij.html
3. Никитин А.П., курс лекций «Криптографические протоколы», РТУ(МИРЭА), 2019г.