МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Московский Государственный Университет им. М.В.Ломоносова

ДИСЦИПЛИНА

«Введение в проблемы информационной безопасности»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

по проведению лабораторной работы № 1

«Анализ защищенного протокола SSL\TLS»

Москва, 2016

1. Введение

Настоящее учебно-методическое пособие ориентировано на ознакомление студентов с основными принципами обеспечения информационной безопасности компьютерных сетей. Основным предметом рассмотрения является протокол защищенной передачи данных SSL\TLS, широко используемый различными приложениями для обеспечения безопасной передачи данных. Детальное изучение содержания информации имеет целью предотвращения атак злоумышленников на компьютерные сети при помощи выявления ошибок настройки протокола и обнаружения его слабостей.

В результате выполнения лабораторной работы студенты получают:

- в области знания и понимания - углубляют свои знания по основным подходам к защите данных от НСД в части безопасности компьютерных сетей;
- в области интеллектуальных навыков - укрепляют свои навыки по использованию механизмов решения задач анализа средств защиты информации на сетевом и транспортном уровне модели ISO\OSI;
- в области практических навыков - учатся анализировать сетевые протоколы передачи данных для решения профессиональных задач, а так же квалифицированно оценивать область применения конкретных механизмов защиты.

2. Цели и задачи лабораторной работы

Изучение особенностей реализации протокола TLS для установления HTTPS соединения с веб-серверами различных организаций: государственных, образовательных, торговых, корпоративных, СМИ и т.д. (в РФ и за Рубежом) для оценки и сравнительного различных подходов к обеспечению безопасности.

Учебные (исследовательские) задачи лабораторной работы

A. Изучение используемых на различных серверах версий протокола TLS
B. Изучение возможности принудительного понижения версий TLS соединений и стойкости используемых криптоалгоритмов.
C. Изучение используемы сертификатов для выявления предпочтительных удостоверяющих центров.
D. Сравнительный анализ используемых версий TLS, режим шифрования и удостоверяющих центров различными организациями в РФ и за рубежом.

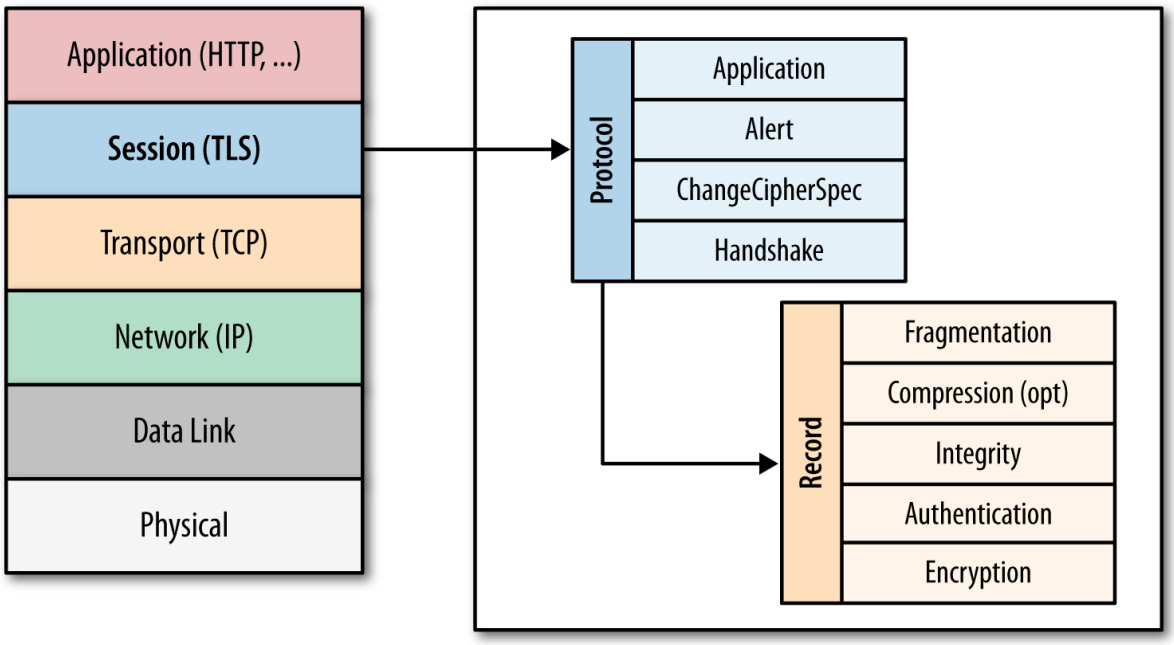
3. Общая характеристика TLS.

3.1. Стуктура протокола TLS.

Протокол TLS (transport layer security) основан на протоколе SSL (Secure Sockets Layer), изначально разработанном в Netscape для повышения безопасности электронной коммерции в Интернете. Протокол SSL был реализован на application-уровне, непосредственно над TCP (Transmission Control Protocol), что позволяет более высокоуровневым протоколам (таким как HTTP или протоколу электронной почты) работать без изменений. Если SSL сконфигурирован корректно, то сторонний наблюдатель может узнать лишь параметры соединения (например, тип используемого шифрования), а также частоту пересылки и примерное количество данных, но не может читать и изменять их. TLS используется для обеспечения безопасности целого ряда прикладных протоколов:

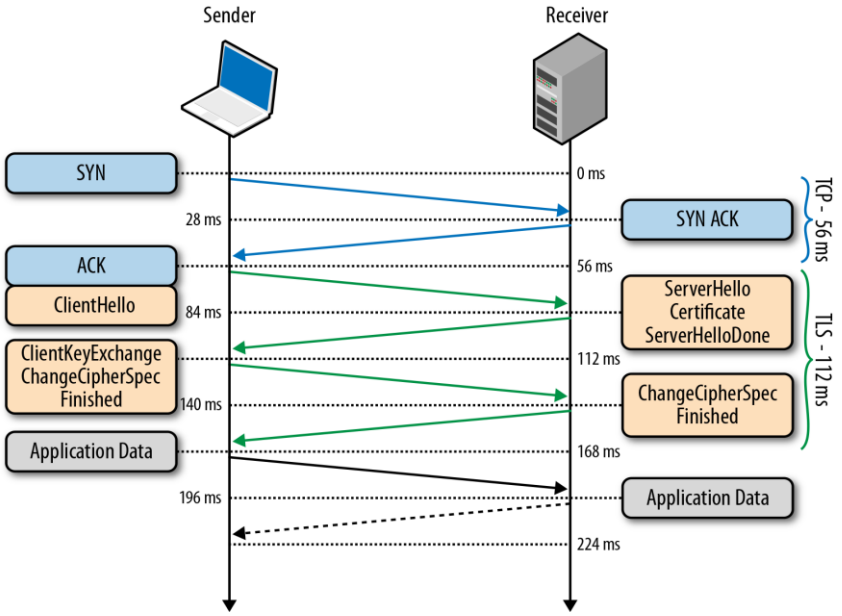
Протокол	Порт	Описание
HTTPS	443	HTTP по SSL/TLS
SMTPS	465	SMTP (электронная почта) по SSL/TLS
NNTPS	563	NNTP (новости) по SSL/TLS
LDAPS	636	LDAP (доступ к каталогам) по SSL/TLS
POP3S	995	POP (электронная почта) по SSLДLS
IRCS	994	IRC (чат) по SSL/TLS
IMAPS	993	ШАР (электронная почта) по SSL/TLS
FTPS	990	FTP (передача файлов) по SSL/TLS

После того, как протокол SSL был стандартизирован IETF (Internet Engineering Task Force), он был переименован в TLS. Первая выпущенная версия протокола имела название SSL 2.0, но была довольно быстра заменена на SSL 3.0 из-за обнаруженных уязвимостей. В январе 1999 года IETF открыто стандартизирует его под именем TLS 1.0. Затем в апреле 2006 года была опубликована версия TLS 1.1, которая расширяла первоначальные возможности протокола и закрывала новые известные уязвимости. Актуальная версия протокола на данный момент – TLS 1.2, выпущенная в августе 2008 года. Конкретное место TLS (SSL) в стеке протоколов Интернета показано на схеме[1]:



[1] схемы приведены по «High Performance Browser Networking» Chapter 4. «Transport Layer Security (TLS)» Ilya Grigorik O'Reilly Media September 2013.

Общая схема установления соединения с использование TLS имеет вид:



3.2. Временные издержки и ускорение реализации TLS.

Значения временных задержек приведенные зависят от типа оборудования, версии и реализации TLS. Эти временные затраты связаны с тем, что перед тем, как начать обмен данными через TLS, клиент и сервер должны согласовать параметры соединения: версия используемого протокола, способ шифрования данных, а также проверить сертификаты, если это необходимо. Отметим SSL/TLS в Windows 2000 работает немного быстрее, если используется 128-, а не 40- или 56-разрядный ключ, а при увеличении длины ключа с 512 до 1024 бит количество соединений в секунду резко падает и может уменьшиться в пять раз[2].

Более детальная схема начала установки соединения (TLS Handshake) показана на схеме[3]:

---

[2] см. http://www.cryptopro.ru/sites/default/files/docs/TLS_description.pdf

[3] http://pro-ldap.ru/tr/zytrax/tech/ssl.html

Когда версия протокола и способ шифрования считаются утвержденными, клиент проверяет присланный сертификат и инициирует обмен ключами либо на основе RSA, либо по Диффи-Хеллману, в зависимости от установленных параметров.

По различным историческим и коммерческим причинам чаще всего в TLS используется обмен ключами по алгоритму RSA: клиент генерирует симметричный ключ, подписывает его, шифрует с помощью открытого ключа сервера и отправляет его на сервер. Обмен ключами Диффи-Хеллмана представляется более защищенным, так как установленный симметричный ключ никогда не покидает клиента или сервера и, соответственно, не может быть перехвачен злоумышленником, даже если тот знает закрытый ключ сервера.

Производительность программных реализаций алгоритма RSA очень низка и очень быстро снижается при увеличении длины ключа.

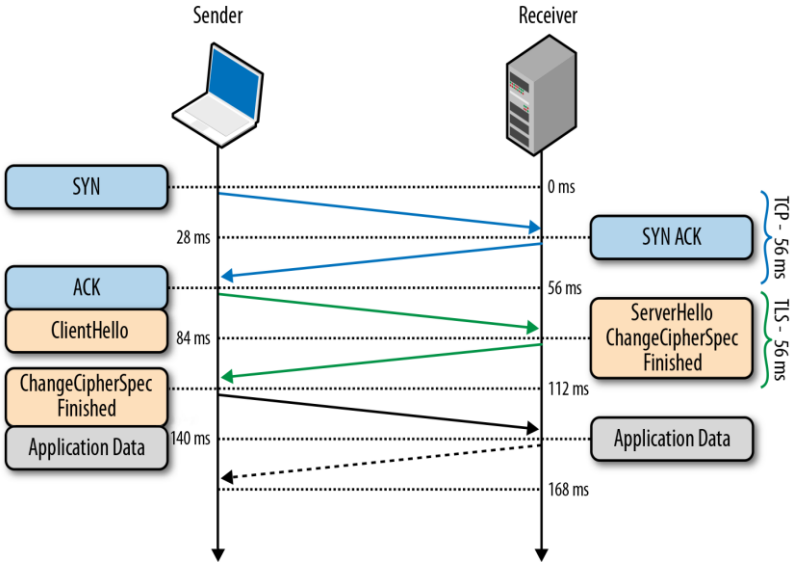
Начиная с первой публичной версии протокола (SSL 2.0) сервер в рамках TLS Handshake (а именно первоначального сообщения ServerHello) может сгенерировать и отправить 32-байтный идентификатор сессии (connection-id). Начальный обмен сообщениями:

Client-hello	C ® S:	challenge, cipher_specs
Server-hello	S ® C:	connection-id,server_certificate,cipher_specs
Client-master-key	C ® S:	{master_key}server_public_key
Client-finish	C ® S:	{connection-id}client_write_key
Server-verify	S ® C:	{challenge}server_write_key
Server-finish	S ® C:	{new_session_id}server_write_key

Если у сервера хранится кэш сгенерированных идентификаторов и параметров сеанса для каждого клиента, то в свою очередь клиент хранит у себя присланный идентификатор и включает его (конечно, если он есть) в первоначальное сообщение ClientHello. Если и клиент, и сервер имеют идентичные идентификаторы сессии, то установка общего соединения происходит по упрощенному алгоритму:

Client-hello	C ® S:	challenge, session_id, cipher_specs
Server-hello	S ® C:	connection-id, session_id_hit
Client-finish	C ® S:	{connection-id}client_write_key
Server-verify	S ® C:	{challenge}server_write_key
Server-finish	S ® C:	{session_id}server_write_key

Процедура возобновления сессии позволяет пропустить этап генерации симметричного ключа, что существенно повышает время установки соединения, но не влияет на его безопасность, так как используются ранее не скомпрометированные данные предыдущей сессии.

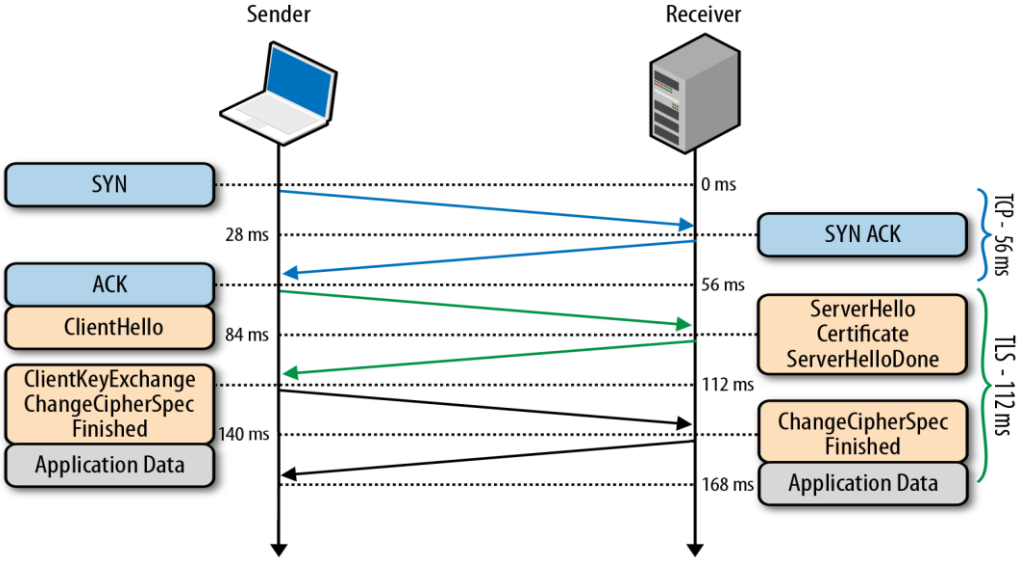


Считается что установка нового SSL/TLS-соединения занимает приблизительно впятеро больше времени, чем восстановление соединения, помещенного в кэш. Стандартный тайм — аут для такого соединения в Windows NT 4 увеличен с 2 до 5 минут. Время жизни соединения в кэше можно увеличить, но при слишком большом таймауте система будет тратить память на кэширование устаревших соединений.

Технология возобновления сессии бесспорно повышает производительность протокола и снижает вычислительные затраты, однако она не применима в первоначальном соединении с сервером, или в случае, когда предыдущая сессия уже истекла.

Для получения ещё большего быстродействия была разработана технология TLS False Start, являющаяся опциональным расширением протокола и позволяющая отправлять данные, когда TLS Handshake завершён лишь частично. Важно отметить, что TLS False Start никак не изменяет процедуру TLS Handshake. Он основан на предположении, что в тот момент, когда клиент и сервер уже знают о параметрах соединения и симметричных ключах, данные приложений уже могут быть отправлены, а все необходимые проверки можно провести параллельно. В результате соединение готово к использованию на одну итерацию обмена сообщениями раньше.

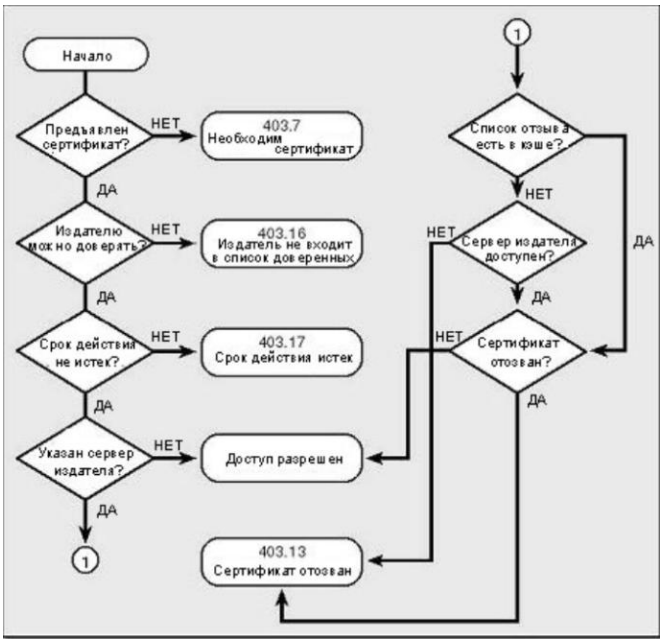
Процедура TLS False Start представлена на схеме:



3.3. Аутентификация в TLS

Содержание самой аутентификации предполагает выполнение ряда процедур сервером\клиентом:

1) выполняется криптографическая проверка наличия у сервера личного ключа, соответствующего открытому ключу, указанному в сертификате;
2) проверяется степень доверия издателю сертификата;
3) проверяется, не истек ли срок действия сертификата;
4) проверяется, не отозван ли сертификат.



В настоящее время в в качестве сертификата предлагается общий стандарт для Интернет с использованием формата X.509. Сертификат открытого ключа удостоверяет принадлежность открытого ключа некоторому субъекту, например, пользователю. Сертификат открытого ключа содержит имя субъекта, открытый ключ, имя удостоверяющего центра, политику использования соответствующего удостоверяемому открытому ключу закрытого ключа и другие параметры, заверенные подписью удостоверяющего центра.Формат сертификата открытого ключа X.509 v3 описан в RFC 5280.

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:

a) сообщение для проверки сертификата,
b) проверки ключей шифрования,
c) секрета MAC (message authentication code),
d) сообщения «finished».

Отсылая сообщение «finished», стороны указывают, что они знают верный секрет (pre_master_secret).

Отметим, что аутентификация **опционально** может быть **односторонней** (аутентификация сервера клиентом), **взаимной** (встречная аутентификация сервера и клиента) или **не использоваться** (в отличие от многих других реализаций TLS, например в HTTPS, большинство реализаций EAP-TLS требует предустановленного сертификата X.509 у клиента, не давая возможности отключить это требование, хотя стандарт RFC 5216 не требует этого в обязательном порядке — одна из причин высокой защищенности метода EAP-TLS).

3.4. Безопасность протокола TLS.

Вся история развития семейства протоколов SSL-TSL связана с обнаружением уязвимостей протокола и их устранением в следующей версии. В связи с этим активизирован переход с SSL на TSL[4] и на TLS последних версий, хотя по некоторым данным до 30% серверов и клиентов на старом ПО используют SSL, также отмечаются ошибки в настройках и даже “принудительное ослабление” TSL для уменьшения временных издержек.

Атаки можно условно разделить на несколько групп:

A. Атаки на протокол TLS, используемый для приложений сети интернет.
B. Атаки на ошибки в спецификациях протокола TLS.
C. Атаки на слабости используемых математических методов (генераторы случайных чисел, хэш-функции и криптографические алгоритмы)
D. Атаки на слабости реализации используемых математических методов.

Из приведенного ниже краткого обзора можно сделать вывод, что именно особенности установленного SSL\TLS соединения позволяют использовать тот или иной вид атаки и для подготовки атаки необходима тщательная предварительная разведка.

3.4.1. Атака Beast[5].

Атака проводится многоэтапно. Для успешного осуществления атаки требуется запуск JavaScript-кода в браузере клиента, который должен быть запущен после установки SSL-соединения браузера с сайтом, данные которого требуется перехватить (SSL-соединение остается открытым длительное время, поэтому у атакующего есть запас времени). JavaScript-код не требуется запускать в контексте атакуемого сайта, его достаточно открыть в новой вкладке, например, путем встраивания через iframe в какой-нибудь сторонний сайт, открытие которого можно стимулировать методами социальной инженерии. JavaScript-код используется для отправки на сайт, с которым работает жертва, фиктивных запросов с изначально известными контрольными метками, которые используются атакующим для воссоздания отдельных блоков для шифра TLS/SSL, работающего на базе алгоритма AES.

После того, как вспомогательный JavaScript-код запущен, на одном из промежуточных шлюзов осуществляется запуск сниффера, для выявления связанных с

[4] http://www.atraining.ru/beast-move-from-ssl-to-tls/

[5] theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl/

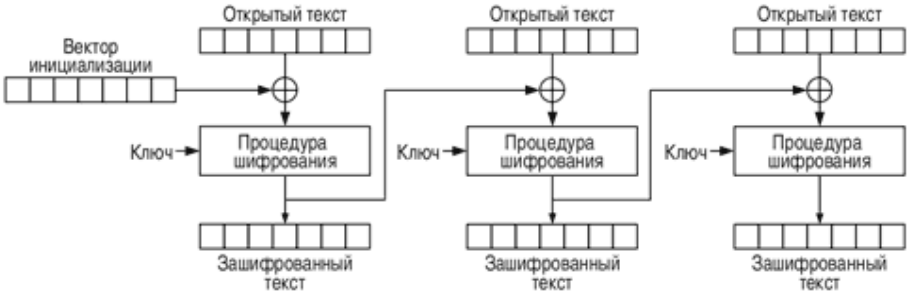
определенным сайтом TLS-соединений, их перехват и расшифровка начальной части HTTPS-запроса, в которой содержится HTTPS Cookie (запросы вспомогательного скрипта отправляются через ранее установленный SSL-канал, при этом браузер добавит к этим запросам все ранее установленные Cookie). После того, как удалось воссоздать идентифицирующую HTTPS Cookie, осуществляется классическая атака, позволяющая вклиниться в активную сессию жертвы. Расшифровка основана на методе угадывания содержимого отдельных блоков, часть которых содержит данные отправляемые подставным JavaScript-кодом.

3.4.2. Атака Crime[6]

(Compression Ratio Info-leak Made Easy). Атака эффективна для каналов связи на основе SSL/TLS и SPDY только при использовании сжатия передаваемых данных и требует выполнения JavaScript-кода злоумышленника в браузере клиента. По своей сути Crime является продолжением развития идей атаки Beast. Атака строится на возможности выделения в отслеживаемом зашифрованном трафике блоков данных с метками, отправляемыми подставным JavaScript-кодом на сайт, для которого требуется перехватить Cookie, в рамках общего шифрованного канала связи. Используя тот факт, что данные сжимаются на этапе до шифрования и не подвергаются дополнительной рандомизации, новая атака позволяет обойти средства защиты, добавленные производителями браузеров для блокирования атаки Beast.

3.4.3. Padding Oracle Attack[7].

Используется для атаки на использование симметричного блочного шифрования с режимом сцепления блоков шифротекста (Cipher Block Chaining) — один из режимов шифрования для симметричного блочного шифра с использованием механизма обратной связи. Каждый блок открытого текста (кроме первого) побитно складывается по модулю 2 (операция XOR) с предыдущим результатом шифрования.



Важно, что предпочтительным методом дополнения блоков шифротекста, является PKCS7 (есть другие варианты, например PKCS5, но для них так же возможно организация подобной атаки). В нем значение каждого дополняемого байта устанавливается равным количеству дополняемых байт. А знание факта, получается ли при расшифровке шифротекста открытый текст с корректным дополнением, достаточно атакующему для проведения успешной атаки на шифрование в режиме CBC. Если мы можем подавать какому-то сервису шифротексты, а он будет возвращать нам информацию, корректно ли дополнение (перехват ответа об ошибке) — мы сможем вскрыть любой шифротекст. Ruby OpenSSL подвержен данной проблеме.

3.4.4. Lucky 13[8].

Уязвимость связана именно с недоработкой в спецификациях TLS, а не с конкретными реализациями, поэтому баг присутствует во всех библиотеках. Уязвимость, которая потенциально позволяет извлечь конфиденциальную информацию из зашифрованного канала связи, в том числе пароли и аутентификационные cookies, использующего режим сцепления блоков шифротекста (CBC). Этот метод использует атаку типа padding oracle,

---

6 isecpartners.com/blog/2012/9/14/details-on-the-crime-attack.html

7 http://habrahabr.ru/post/247527/

8 http://www.isg.rhul.ac.uk/tls/

подменяя шифротекст и анализируя временную задержку при ответе для определения ошибки при дополнении блока шифротекста.

3.4.5. Атака Freak[9].

Суть атаки сводится к инициированию отката соединения на использование разрешенного для экспорта набора шифров, включающего недостаточно защищенные устаревшие алгоритмы шифрования. Проблема позволяет вклиниться в соединение и организовать анализ трафика в рамках защищенного канала связи, используя уязвимость[10], выявленную во многих SSL-клиентах и позволяющую сменить шифры RSA на RSA_EXPORT и выполнить дешифровку трафика, воспользовавшись слабым эфемерным ключом RSA (512 бит). Для подбора ключа RSA-512 исследователям потребовалось около семи с половиной часов[11]. На стороне клиента уязвимость затрагивает OpenSSL (исправлено в 0.9.8zd, 1.0.0p и 1.0.1k), браузер Safari и разнообразные встраиваемые и мобильные системы, включая Google Android и Apple iOS.

3.4.6. Атака Poodle[12].

Padding Oracle On Downgraded Legacy Encryption. Атака позволяет восстановить содержимое отдельных секретных идентификаторов, передаваемых внутри зашифрованного SSLv3-соединения, и по своей сути напоминает такие ранее известные виды атак на HTTPS, как BREACH, CRIME и BEAST, но значительно проще для эксплуатации и не требует выполнения каких-то особых условий. Проблеме подвержен любой сайт, допускающий установку защищенных соединений с использованием протокола SSLv3, даже если в качестве более приоритетного протокола указаны актуальные версии TLS. Для отката на SSLv3 атакующие могут воспользоваться особенностью современных браузеров переходить на более низкую версию протокола, в случае сбоя установки соединения. Атака строится на возможности выделения в отслеживаемом зашифрованном трафике блоков данных с метками, отправляемыми подставным JavaScript-кодом на сайт, для которого требуется перехватить идентификационные данные, в рамках общего шифрованного канала связи.

3.4.7. Атака Logjam.

Это атака на TLS, которой подвержено большое число клиентских и серверных систем, использующих HTTPS, SSH, IPsec, SMTPS и другие протоколы на базе TLS. По своей сути Logjam напоминает представленную в марте атаку FREAK и отличается тем, что вместо инициирования смены шифров RSA на RSA_EXPORT в Logjam производится откат протокола Диффи-Хеллмана (Diffie-Hellman), используемого для получения ключа для дальнейшего шифрования, до слабозащищённого уровня DHE_EXPORT, что в сочетании с использованием не уникальных начальных простых чисел позволяет применить методы подбора ключа.

3.4.8. Взлом TLS и SSL через уязвимость в шифре RC4[13].

Алгоритм RC4, как и любой потоковый шифр, строится на основе параметризованного ключом генератора псевдослучайных битов с равномерным распределением. Длина ключа может составлять от 40 до 256 бит. Основные преимущества шифра — высокая скорость работы и переменный размер ключа.

Уязвимость RC4 связана с недостаточной случайностью потока битов, которым скремблируется сообщение. Прогоняя через него одно сообщение много раз, удалось выявить достаточное количество повторяющихся паттернов для восстановления исходного сообщения. Однако для атаки нужно прогнать через шифр большой объем данных.

---

9 fhttp://www.reakattack.com

10 cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-0204

11 smacktls.com/#freak

12 openssl.org/~bodo/ssl-poodle.pdf

13 http://www.isg.rhul.ac.uk/tls/index.html

3.4.9. Новая MITM атака[14].

Для осуществления атаки злоумышленнику необходимо каким-либо образом убедить жертву установить специально изготовленный клиентский SSL-сертификат, ключ от которого известен атакующему. В дальнейшем это позволяет «убедить» клиентский софт в том, что он взаимодействует с доверенным сервером, в то время как на самом деле коммуникация осуществляется с атакующим. Реализация TLS в библиотеке BouncyCastle подвержена новой атаке. Кроме того, уязвима и TLS-библиотека операционной системы Mac OS X (Secure Transport).

3.4.10. Слабости ключей “малой” размерности[15].

Необходимым условием применения TLS в сетях с произвольными, меняющимися парами соединений является использование криптографии с открытым ключом. Однако в отличие от классической симметричной криптографии стойкость этих методов основывается на недоказанной пока сложности решения ряда задач (факторизация простых чисел и логарифмирование над конечным полем). Кроме того ввиду вычислительной сложности этих задач при реализации они ограничиваются практически соображениями по определению размеров ключевой информации. А вычислительные возможности атак на такие алгоритмы постоянно возрастают.

Еще в 1990-х годах был разработан алгоритм факторизации целых чисел под названием “метод решета числового поля”[16]. Для криптографии он был интересен тем, что позволил успешно получить доступ к данным, зашифрованным при помощи 768-битного ключа RSA[17].

В 2015 году опубликована статья[18], в которой NFS-метод, использующий результаты предвычислений, был успешно применен для решения задачи дискретного логарифмирования и атаки на процедуру Диффи-Хеллмана.

NFS-метод дает возможность для для любого простого числа p провести процедуру предвычислений, позволяющую в дальнейшем быстро получать любой конкретный дискретный логарифм (то есть решать относительно x уравнение gx =y mod p для любого элемента y фиксированной мультипликативной группы конечного поля вычетов). Задача упрощается тем, что для популярных интернет-протоколов (на текущий момент на них функционируют миллионы серверов) используются одни и те же группы (во многих реализациях простые числа либо зашиты в код, либо постоянно используются одни и те же наборы рекомендованных “безопасных” простых чисел).

3.4.11. Слабости генераторов случайных чисел.

Качественные случайные и псевдослучайные последовательности играют огромную роль при использовании криптографических средств, и зачастую от них полностью зависит стойкость системы. Можно привести следующие примеры случаев, в которых использование плохого источника случайности приводит к краху безопасности всей системы[19]:

- ✓ Использование одного и того же значения в качестве случайного параметра в алгоритмах подписи DSA, ECDSA, ГОСТ Р 34.10-94, ГОСТ Р 34.10-2001, ГОСТ Р 34.10-2012 при обработке двух разных сообщений на одном ключе позволяет восстановить секретные ключи подписи и подделывать подпись. Подобная ошибка в реализации ECDSA была успешно использована при обходе защиты в Sony PlayStation 3.
- ✓ Использование одного и того же случайного набора в качестве синхропосылки (IV) поточных шифров или блочных шифров, работающих в режимах CFB или гаммирования, приводит к перекрытию гаммы и к возможности восстановления частей открытого текста. Подобное уже случалось в реальных системах – неправильное использование

---

[14] http://habrahabr.ru/company/pt/blog/267069/
[15] http://www.cryptopro.ru/blog/2015/11/10/koe-chto-o-zakladkakh-ili-kak-anb-sledit-za-polzovatelyami
[16] https://ru.wikipedia.org/wiki/Общий_метод_решета_числового_поля
[17] https://eprint.iacr.org/2010/006.pdf
[18] https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf
[19] http://www.cryptopro.ru/blog/2015/11/10/koe-chto-o-zakladkakh-ili-kak-anb-sledit-za-polzovatelyami

шифра RC4 позволяло восстановить части документов MS Word и Excel по последовательности автосохраненных копий.

- ✓ Нарушение случайности генерации простых чисел для схемы RSA.

Кроме ошибок возможны и “закладки”-”ошибки” в алгоритмах ряда генераторов: Dual_EC_DRBG, Intel Secure Key.

Приведенный выше неполный список “свежих” уязвимостей показывает, что атаки направлены на вскрытие криптографических протоколов, но используют для этого «слабости» протокола TLS. В тоже время эти слабости связаны с использование TLS для HTTPS. Основной уязвимостью в этом случае является некорректное применение и реализация криптографических методов, что позволяет проводить на них атаки по известным фрагментам шифротекста, использовать криптозакладки[20] или «ослаблять» стойкость применяемых шрифтов.

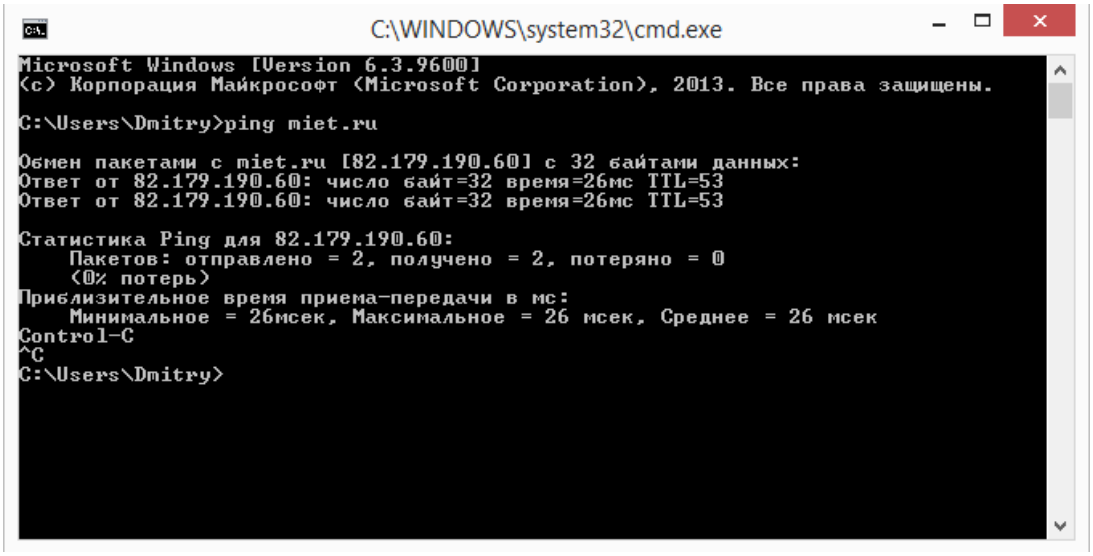
Кроме того практически все атаки носят многоэтапный характер, где, на основе атак по перехвату трафика и атаки MITM, возникает возможность использовать слабости протокола.

В качестве общих рекомендаций можно предложить использование TLS последних версий, тщательную настройку параметров протоколов, максимальное увеличение размера ключей в системах с открытым ключом. И как было указано выше из приведенного краткого обзора видно, что именно особенности установленного SSL\TLS соединения позволяют использовать тот или иной вид атаки, следовательно для подготовки атаки необходима тщательная предварительная разведка.

4. Пример анализа TLS соединения в Wireshark

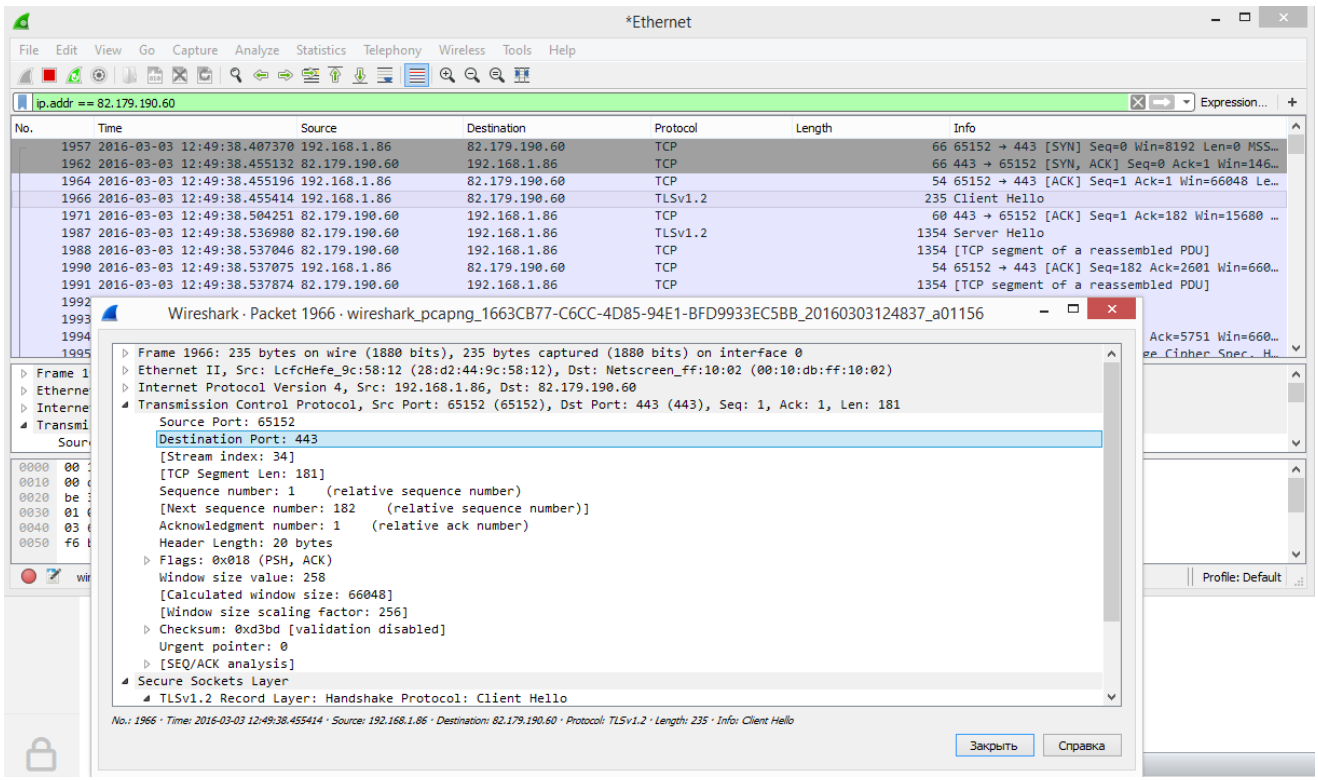
Ниже приведен пример анализа TLS соединения с сервером https://www.miet.ru по аналогии с анализом TLS соединения приведенным в https://habrahabr.ru/post/191954/, (которым также можно воспользоваться как примером анализа TLS соединения).

1) miet.ru ip-address 82.179.190.60

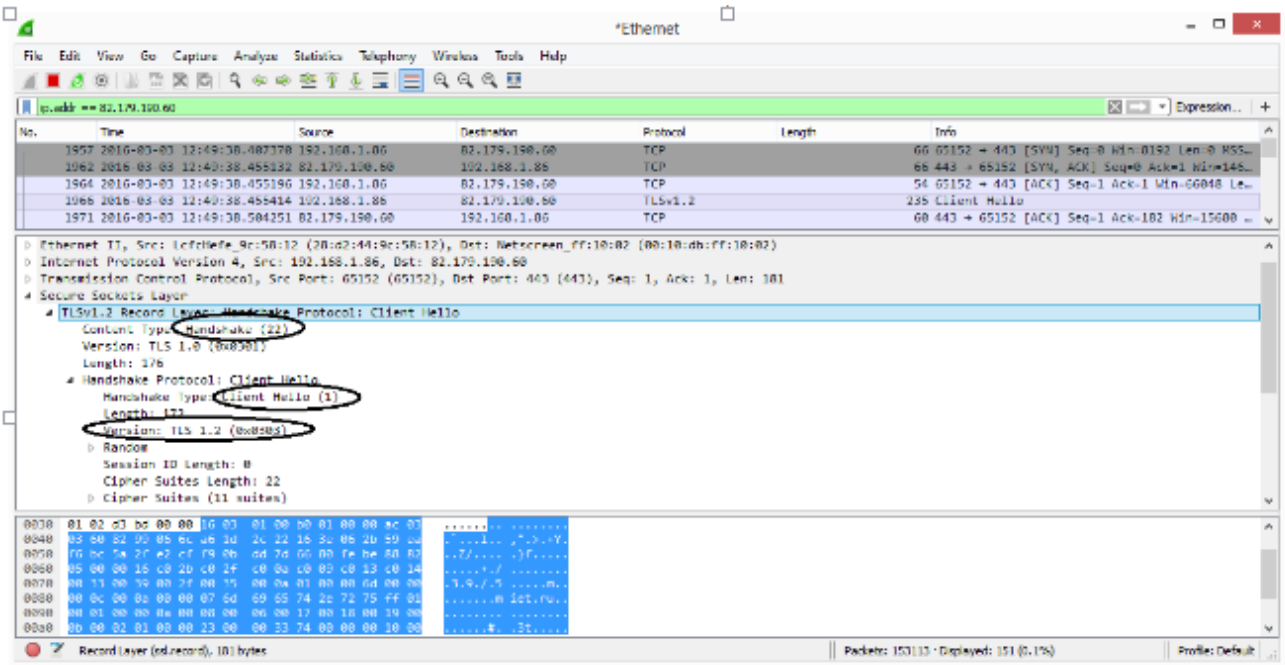


2) При соединении с miet.ru подключение происходит по 443 порту, что представлено на рисунке ниже.

[20] https://dev.by/lenta/main/spetssluzhby-protiv-tls-ssl-perehvachennyy-trafik-zakrytye-klyuchi-kriptozakladki

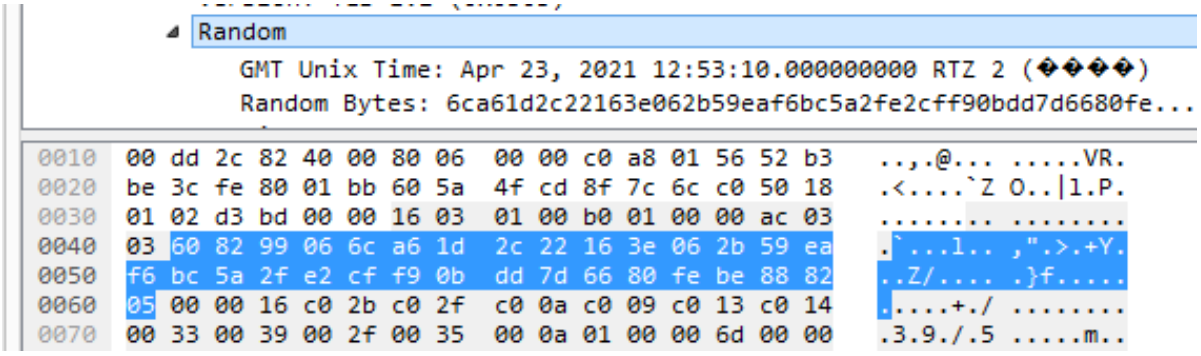


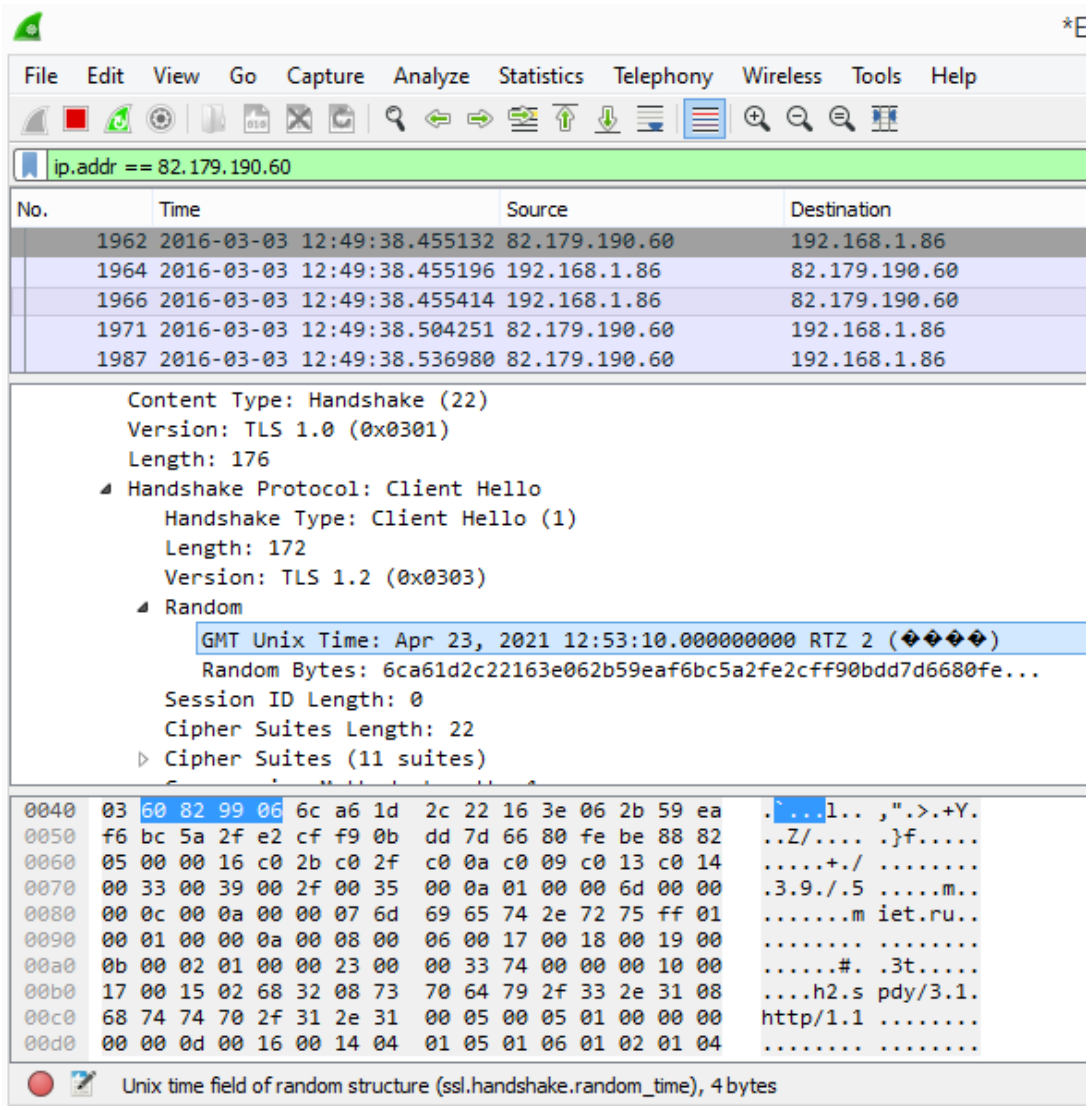
3) Приветствие клиента



первый байт пакета в HEX равен 0x16 = 22, что означает, что «запись» является «рукопожатием». Следующие два байта — 0x0303, означающие версию 3.3, что говорит о том, что TLS 1.2 на самом деле SSL 3.3.
Запись с рукопожатием разбита на несколько сообщений. Первый — «приветствие клиента» (0x01). Здесь есть несколько важных моментов:

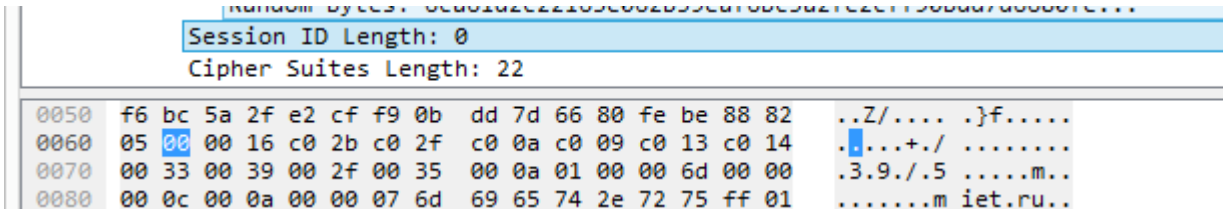
3.1) Случайность:





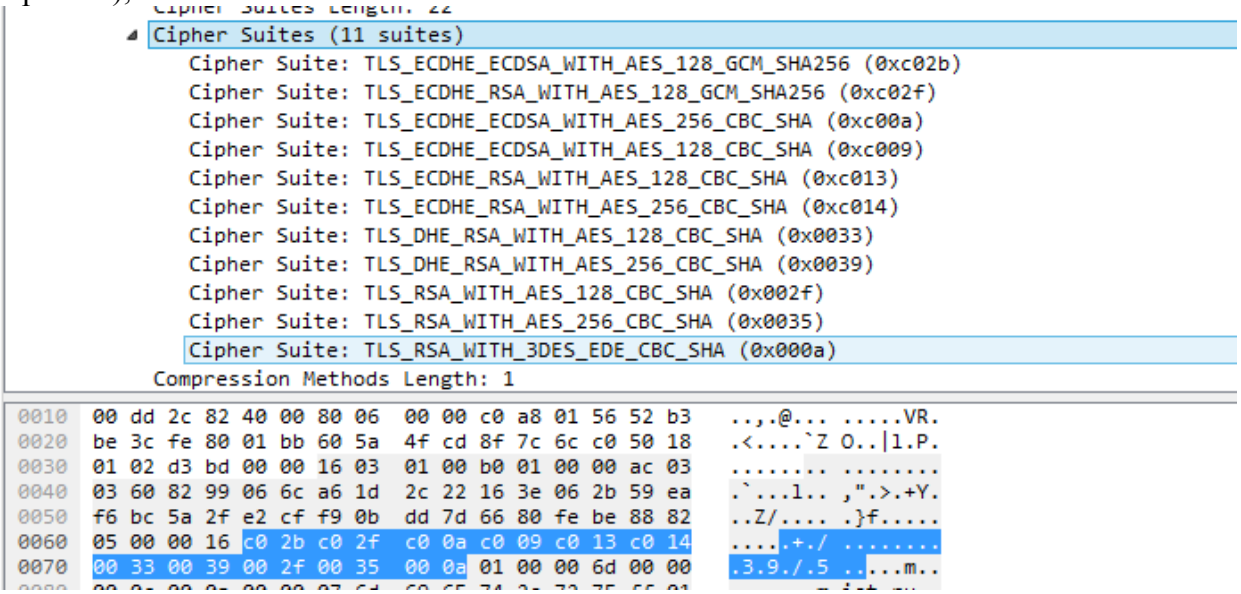
32 байта случайных значений - Client Random. В спецификации рекомендовалось использовать первые 4 байта для передачи UNIX-таймстемпа, а оставшиеся 28 - заполнять результатом работы криптографического генератора псевдослучайных чисел. Однако сейчас многие браузеры и веб-серверы генерируют все 32-байта случайным образом. Забегая чуть вперёд: изначально предполагалось, что наличие таймстемпа в handshake-сообщениях может помочь при обнаружении проблем с подменой времени на том или ином узле. Однако данный метод сейчас никак не используется, поэтому байты ClientRandom часто просто случайны, что и наблюдаем здесь.

3.2) ID сессии:

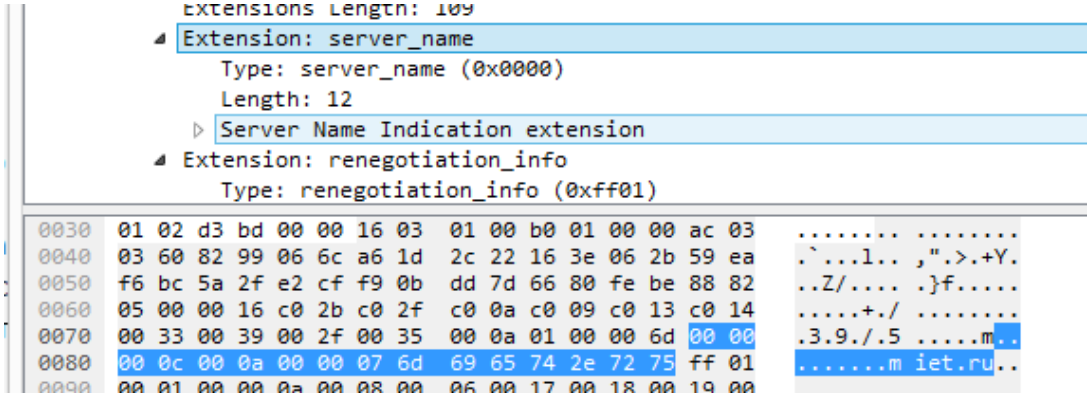


идентификатор TLS-сессии - SessionID: TLS позволяет возобновлять ранее установленные сессии, используя сокращённый вариант протокола установления соединения. Идентификатор сессии как раз содержит номер такой сессии, параметры которой (возможно) сохранены на сервере. В данном случае поле идентификатора сессии пусто.

3.3) Cipher Suites: список шифронаборов, которые поддерживает клиент - Cipher Suites. Порядок шифронаборов в списке отражает их степень предпочтения (предпочтительные идут первыми);



3.4) Расширение server_name:

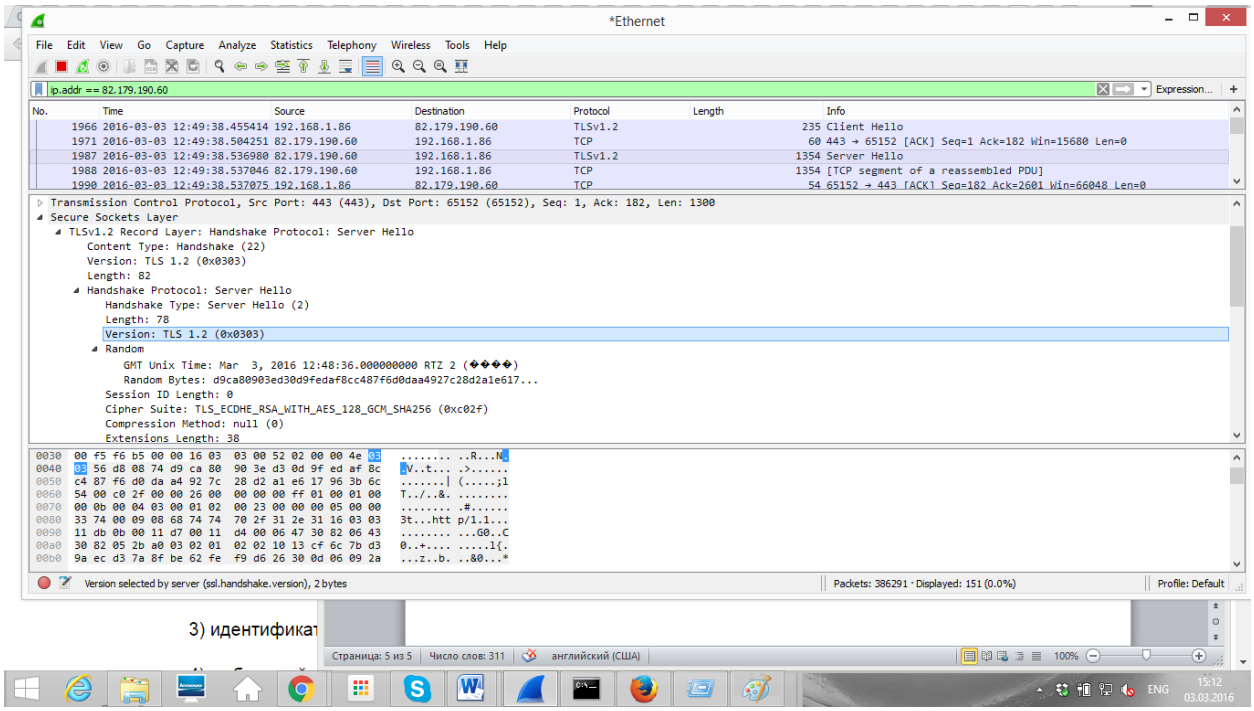


SNI (Server Name Indication), позволяющее браузеру указать имя веб-сервера, с которым он пытается установить соединение. SNI необходимо для того, чтобы на одном IP-адресе могли корректно откликаться по HTTPS веб-серверы, размещённые под разными доменами.

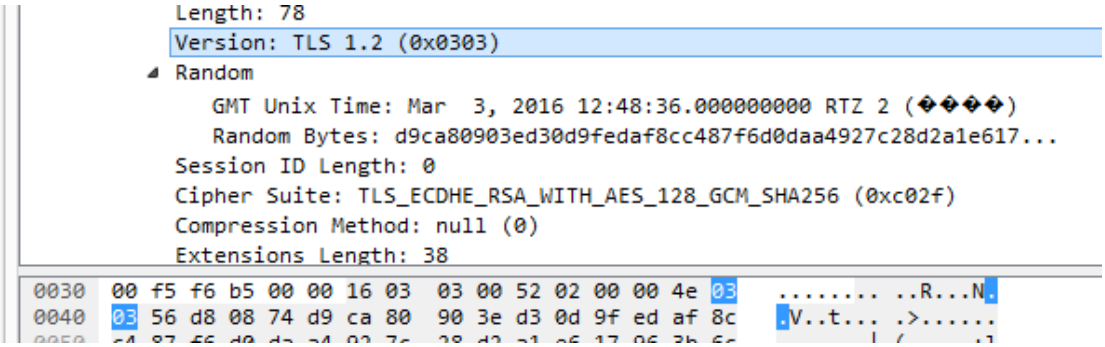
4) Согласно спецификации, после отправки ClientHello клиент ожидает ответа сервера. В ответ может прийти либо сообщение об ошибке, в виде Alert, либо сообщение ServerHello. Если сервер смог успешно обработать ClientHello, то он отвечает сообщением ServerHello. Это сообщение также имеет заголовок из четырёх байтов: тип сообщения (один байт) и

длина (три байта). ServerHello содержит следующие поля:

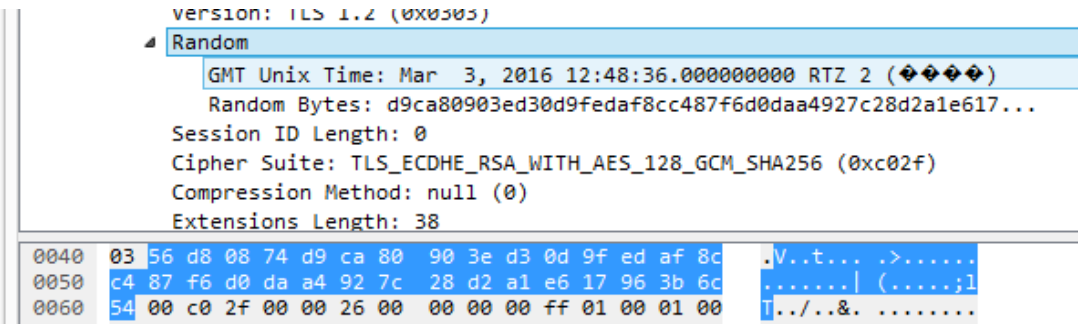
4.1) Сообщение ServerHello



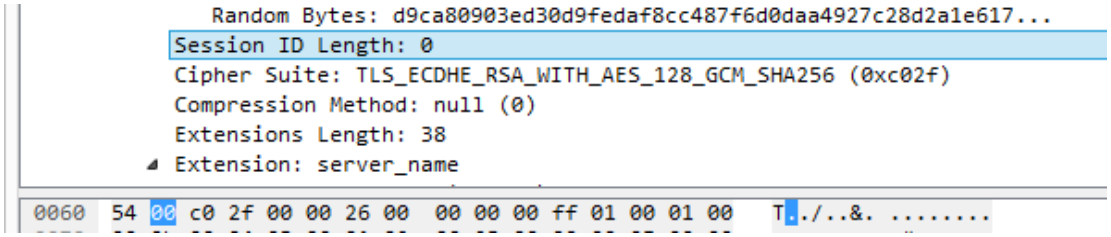
4.2) Версия протокола, которую будут использовать клиент и сервер;



4.3) 32 байта случайных значений - Server Random. С этой строкой ситуация такая же, как и с Client Random: первые четыре байта могут быть таймстемпом, а могут и не быть. В нашем случае это не таймстеп.

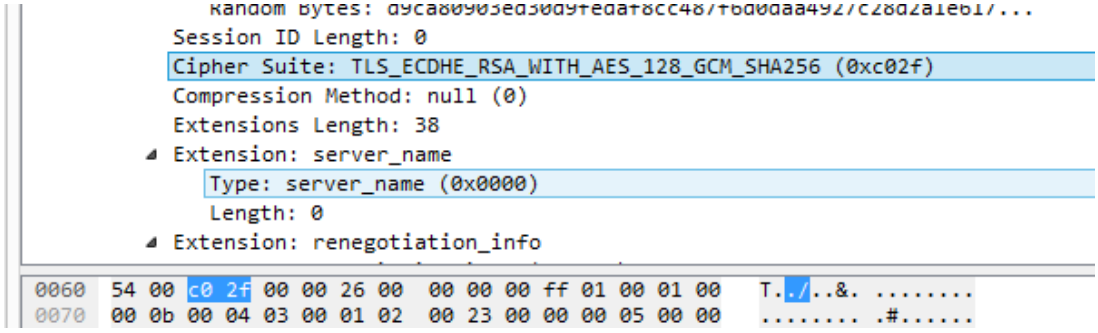


4.4) Идентификатор сессии - SessionID, присвоенный новой сессии сервером;

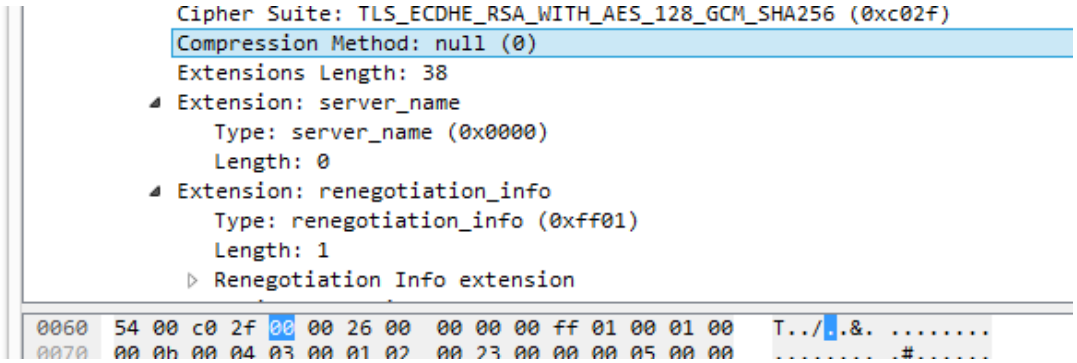


ID сессии назначен не был

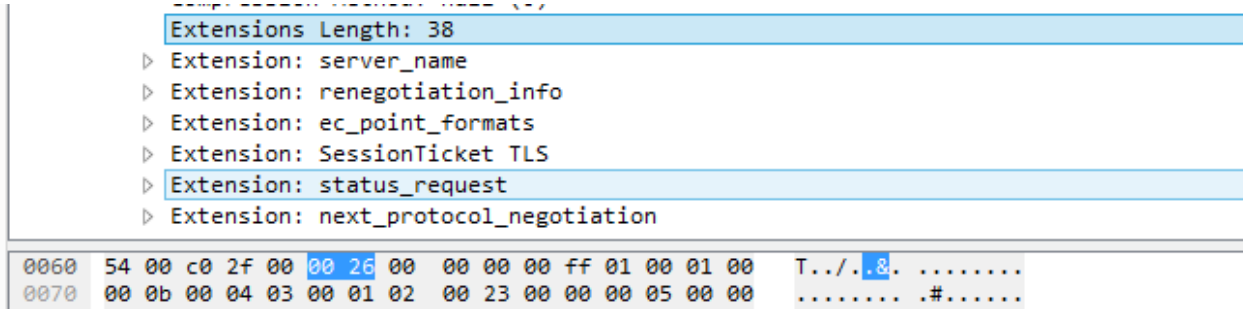
4.5) Выбранный сервером шифронабор - Cipher Suite, этот шифронабор будет использоваться в дальнейшем и клиентом, и сервером. Сервер выбирает шифронабор из предложенных клиентом в ClientHello;



4.6) Выбранный сервером метод сжатия - это null;



4.7) Некоторый набор расширений, который поддерживает сервер.



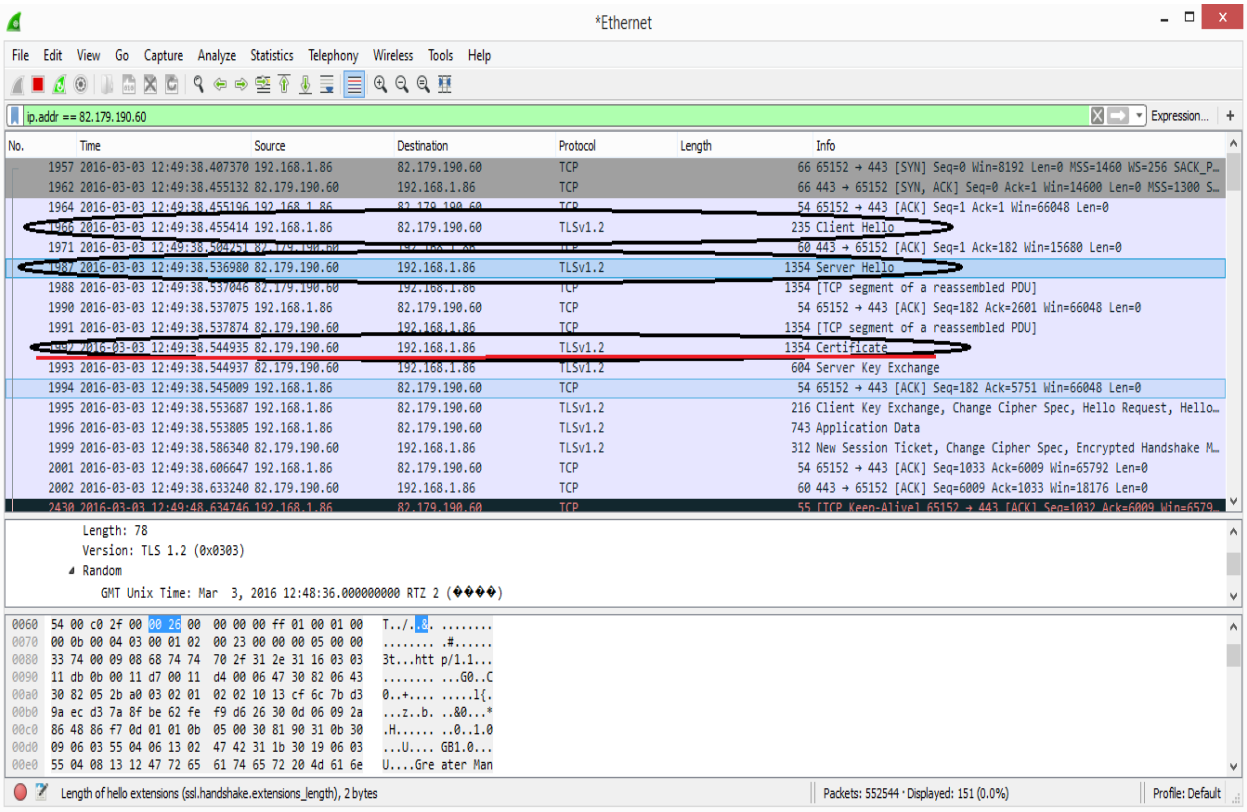
К примеру SessionTicket TLS , показывает что сервер поддерживает технологию выдачи тикетов сессии для клиента для последующего быстрого восстановления соединения TLS сессии.

В современных реализациях TLS раздел "Расширения" (Extensions) имеет очень большое значение, так как в нём передаются параметры, определяющие схему работы и клиента, и сервера. Предназначение ServerHello - согласовать параметры соединения.

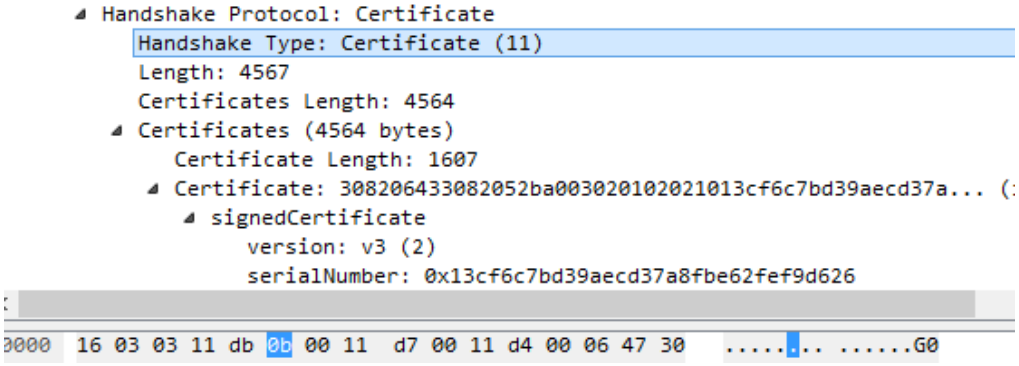
5) За отправкой ServerHello, со стороны сервера следует ряд других сообщений, состав и содержание этих сообщений зависят от выбранного сервером режима работы.

5.1) Certificate. Ключевым аспектом для современных реализаций TLS является использование SSL-сертификатов. Сертификаты передаются сервером в сообщении Certificate. Это сообщение присутствует практически всегда. Серверный сертификат содержит открытый ключ сервера. В теории, спецификация позволяет установить соединение без отправки серверных сертификатов. Это так называемый "анонимный" режим, однако он практически не используется, как и режимы TLS без шифрования. Так, примерно 100% веб-серверов, поддерживающих TLS, передают SSL-сертификаты. Для типичной конфигурации сообщение Certificate будет включать несколько SSL-сертификатов, среди которых один серверный сертификат и так называемые "промежуточные сертификаты", позволяющие клиенту выстроить цепочку валидации. Спецификация предписывает строгий порядок следования сертификатов в сообщении: первым должен идти серверный сертификат, а последующие - в порядке удостоверения предыдущего. Однако на практике серверы нередко возвращают сертификаты в произвольном порядке.

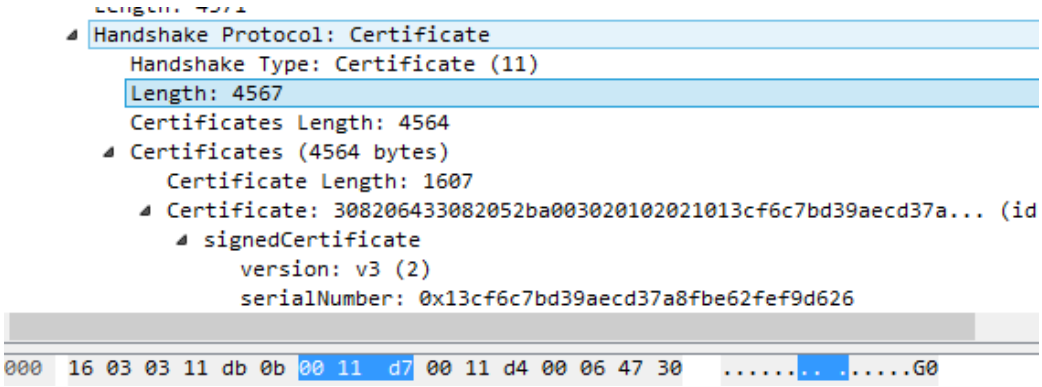
Действительно далее идет пакет с сертификатом



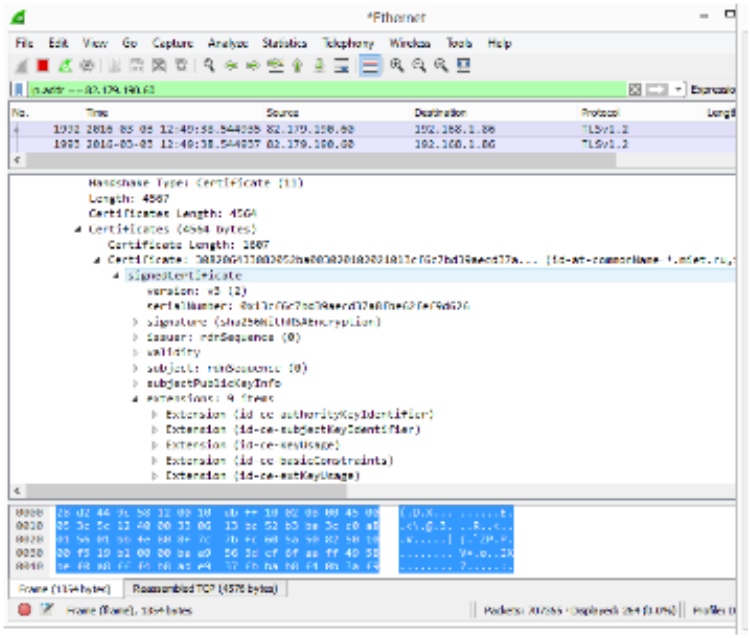
5.2) Сообщение Certificate (тип – 0x0b=11)

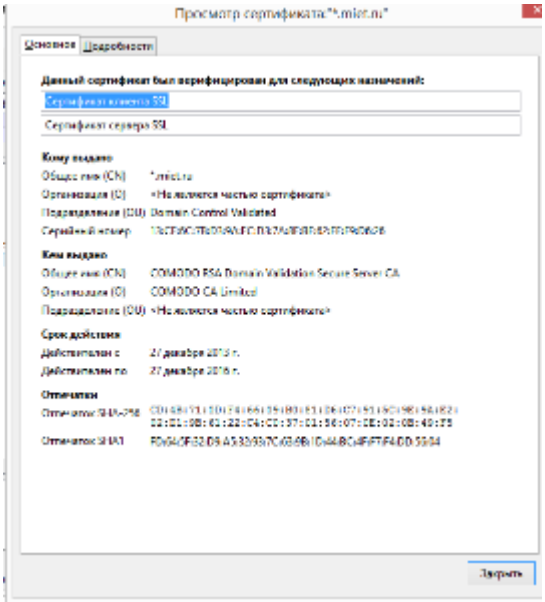


5.3) Длина сообщения (0x0011d7 = 4567)

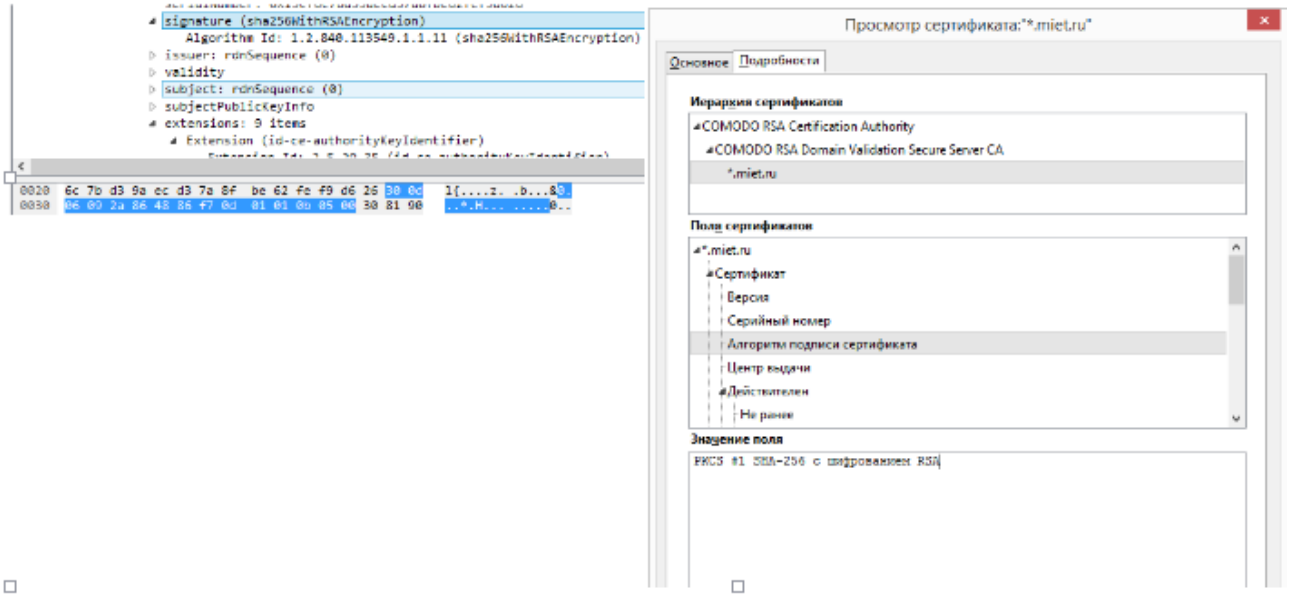


5.4) Данные отображаемые в Wireshark и информация о сертфикате из браузера. Как видим серийные номера совпадают.

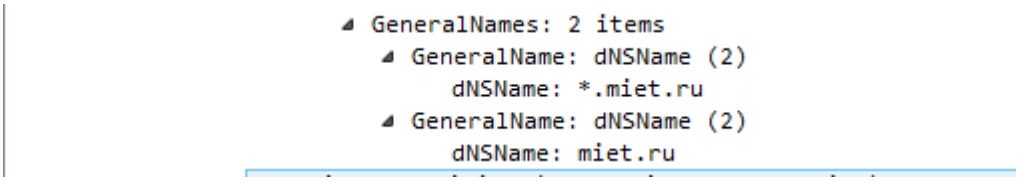




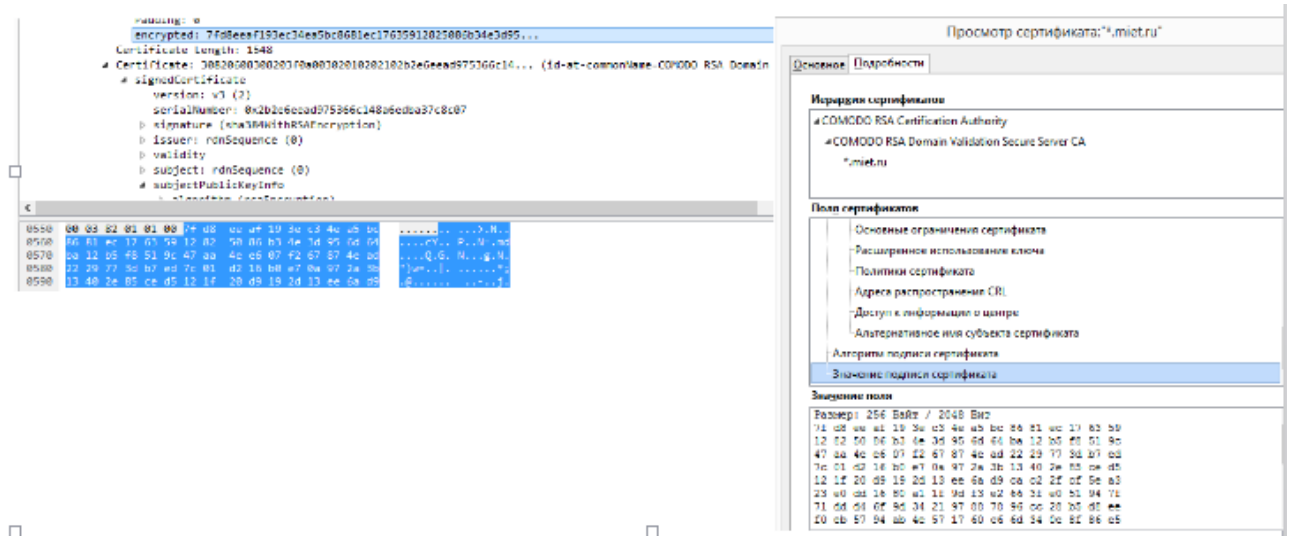
5.5) Алгоритм подписи сертификата.



5.6) Разрешенные DNS имена на которые действует сертификат

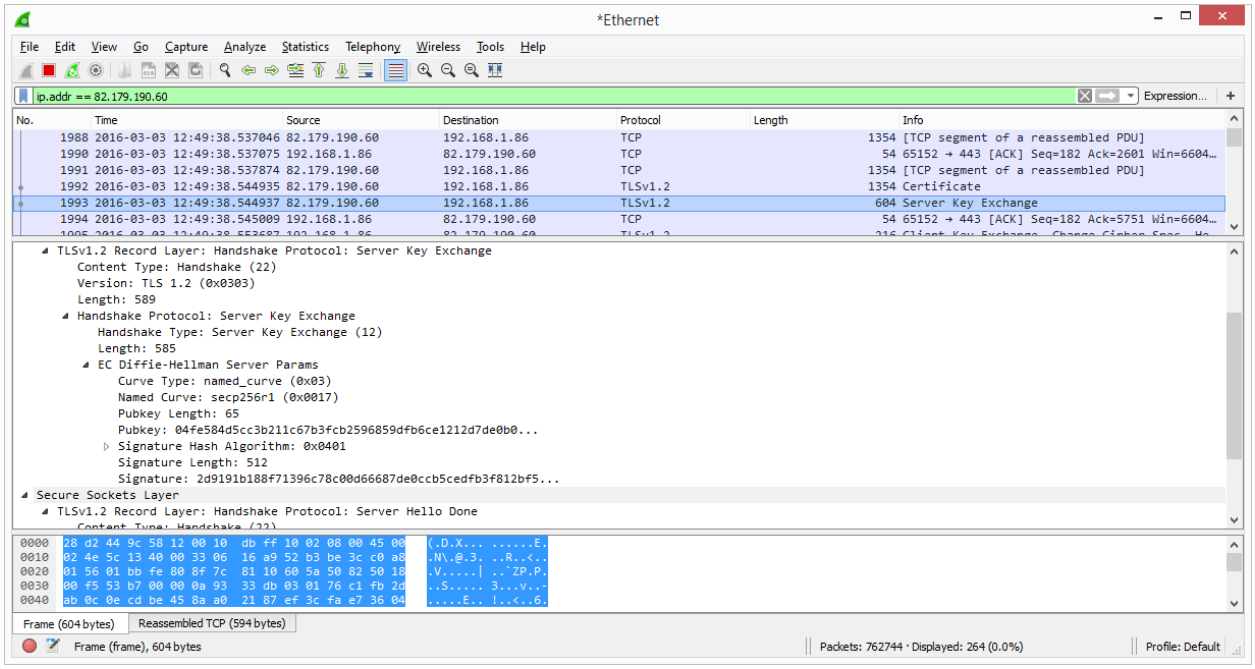


5.7) Значение подписи сертификата. Подпись вычисляется от значения хеш-функции, аргументом которой является весь сертификат (за исключением, соответственно, подписи). Генерируется подпись при помощи секретного ключа УЦ, соответствующего сертификату с именем, указанным в Issuer.

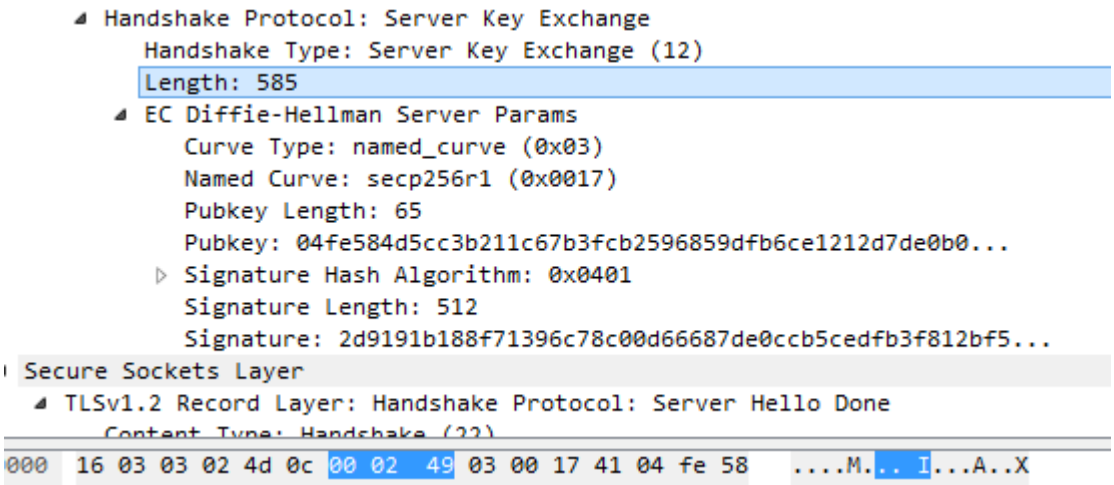


Следующий шаг при установке безопасного соединения это отправка параметров для генерации общего сеансового ключа.

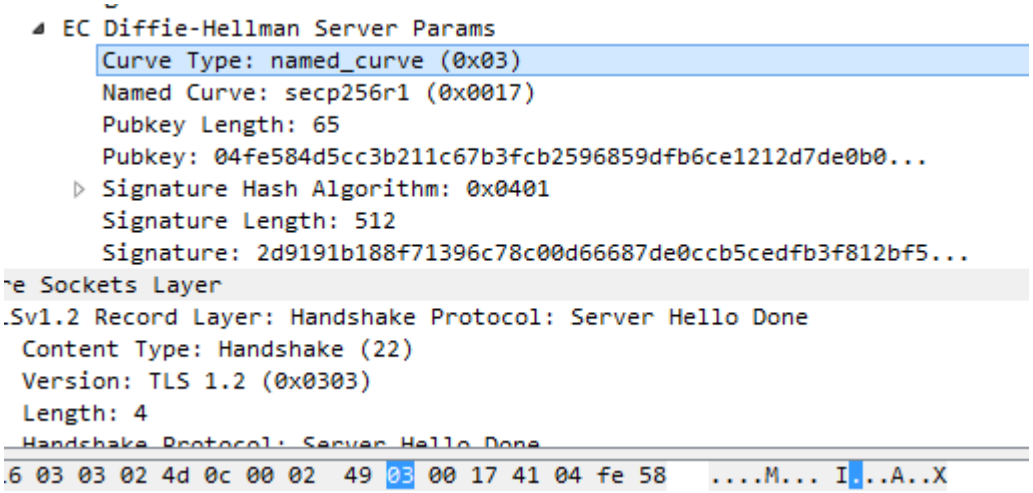
6) ServerKeyExchange. Это сообщение, содержащее серверную часть данных, необходимых для генерации общего сеансового ключа. Это сообщение может отсутствовать. В зависимости от выбранного шифронабора, данных, содержащихся в SSL-сертификате, может быть недостаточно для выработки общего ключа. Недостающие данные как раз и передаются в ServerKeyExchange. Обычно это параметры протокола Диффи-Хеллмана (DH). Если используется классический вариант, то в сообщении ServerKeyExchange передаётся значение модуля и вычисленный сервером открытый ключ DH. В варианте на эллиптических кривых (ECDH) - идентификатор самой кривой и, аналогично DH, открытый ключ. Параметры подписываются сервером, клиент может проверить подпись, используя открытый ключ сервера из SSL-сертификата. В зависимости от используемой криптосистемы подпись может быть DSA (сейчас практически не встречается), RSA или ECDSA.



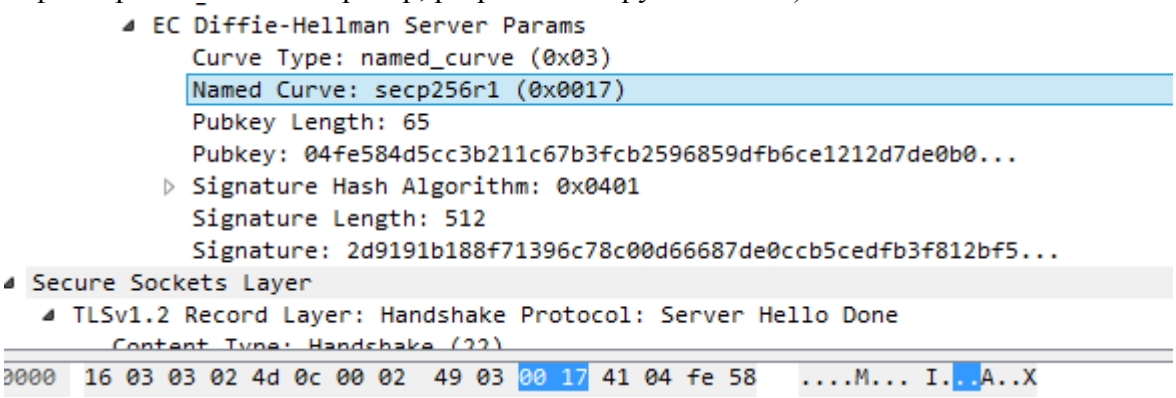
6.1) Сообщение ServerKeyExchange (тип - 0x0c = 12), длина данных (0x000249 = 585 байтов). Так как выбран шифронабор с ECDHE,то сообщение ServerKeyExchange содержит параметры именно для алгоритма ECDHE. Каких-то специальных флагов, обозначающих ServerKeyExchange как ECDHE, - не предусмотрено, всё определяется шифронабором.



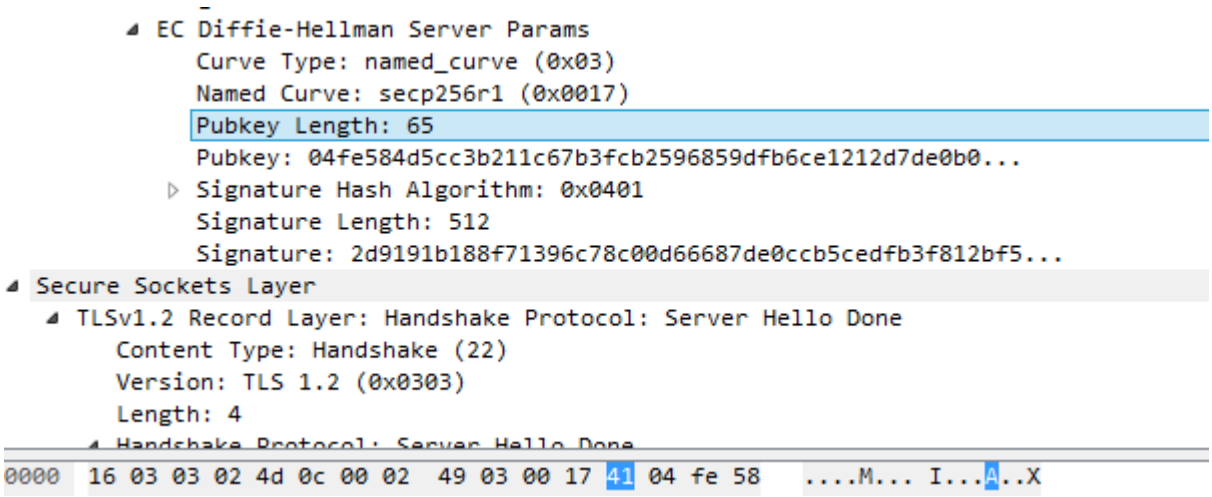
6.2) Тип кривой (0x03 - типовая именованная кривая).



6.3) Идентификатор выбранной сервером кривой (0x0017 - secp521r1, реестр кривых ведёт IANA, каждая из типовых кривых содержит реестр кривых ведёт IANA, каждая из типовых кривых содержит в своём определении необходимые для DH параметры, которые строго зафиксированы: это генератор, разрядность группы точек).



6.4) Длина поля публичного ключа, которое идёт следом (0x41 = 65 байта)



6.5) 65 байта, представляющие собой публичный ключ ECDHE сервера. Публичный ключ - это точка на кривой, которую сервер вычислил, используя параметры кривой. В соответствии с форматом записи точек, указанным клиентом в расширении ClientHello, передаются аффинные координаты точки (x,y) (запись предваряется заголовком). Клиенту параметры кривой известны, потому что используется типовая кривая.

```
Pubkey Length: 65
Pubkey: 04fe584d5cc3b211c67b3fcb2596859dfb6ce1212d7de0b0...
Signature Hash Algorithm: 0x0401
Signature Length: 512
Signature: 2d9191b188f71396c78c00d66687de0ccb5cedfb3f812bf5...
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 4
    Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

```
0000  16 03 03 02 4d 0c 00 02  49 03 00 17 41 04 fe 58   ....M... I...A..X
0010  4d 5c c3 b2 11 c6 7b 3f  cb 25 96 85 9d fb 6c e1   M\....{? .%....l.
0020  21 2d 7d e0 b0 28 33 96  af 00 c5 7e e0 99 0b ea   !-}..(3. ...~....
0030  1b 9a d4 60 f7 0a 93 33  db 03 01 76 c1 fb 2d ab   ...`...3 ...v..-.
0040  0c 0e cd be 45 8a a0 21  87 ef 3c fa e7 36 04 01   ....E..! ..<..6..
```

6.6) Длина подписи, соответствующей параметрам DH (0x0100 - 256 байтов). За значением открытого ключа DH следует серверная подпись, выполненная по алгоритму и с ключом, соответствующим указанным в серверном сертификате (в нашем случае это RSA).

```
Signature Length: 512
Signature: 2d9191b188f71396c78c00d66687de0ccb5cedfb3f812bf5...
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 4
    Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

```
0050  02 00 2d 91 91 b1 88 f7  13 96 c7 8c 00 d6 66 87   ..-..... ......f.
```

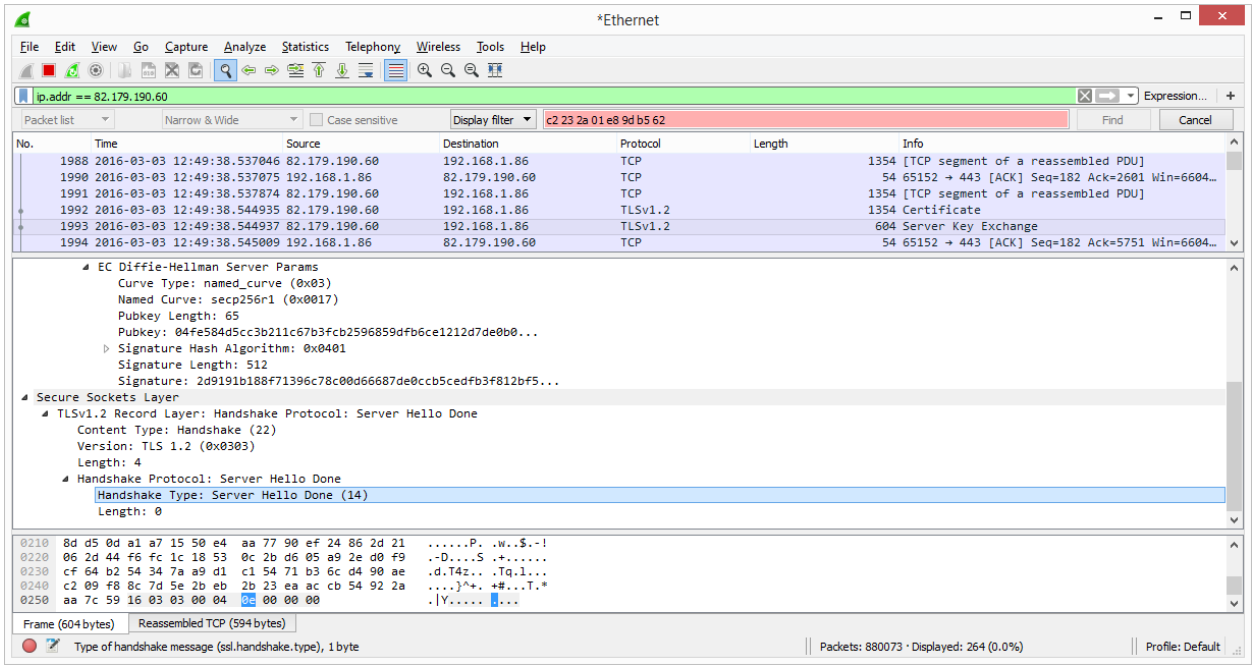
6.7) Значение подписи DH.

```
Signature: 2d9191b188f71396c78c00d66687de0ccb5cedfb3f812bf5...
Secure Sockets Layer
  TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 4
    Handshake Protocol: Server Hello Done
        Handshake Type: Server Hello Done (14)
        Length: 0
```

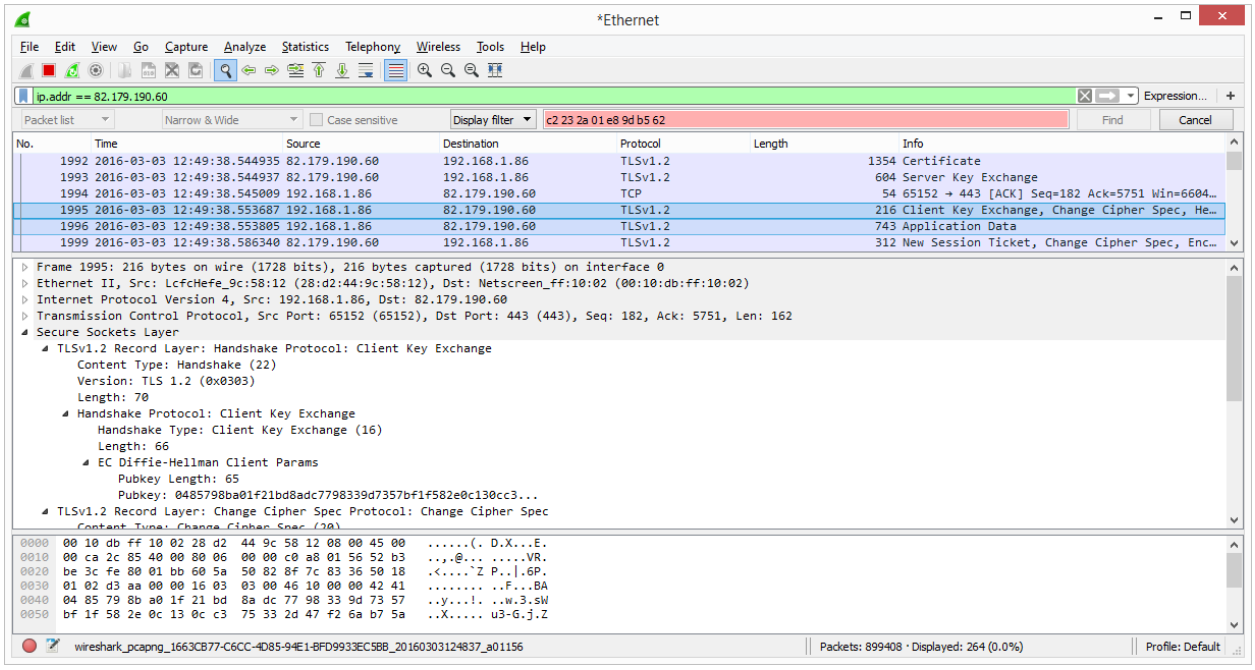
```
0050  02 00 2d 91 91 b1 88 f7  13 96 c7 8c 00 d6 66 87   ..-..... ......f.
0060  de 0c cb 5c ed fb 3f 81  2b f5 fe d1 1f a7 9b 18   ...\..?. +.......
0070  ba f7 43 80 f7 5f a2 10  a5 22 ed 74 44 0e b5 e9   ..C.._.. .".tD...
0080  93 46 56 e7 28 d8 0f 2c  1a ce ad 85 56 8e 7b c3   .FV.(.., ....V.{.
0090  72 02 2c 57 c4 9a 06 1f  4d 8b 25 5e 3d cc c9 ef   r.,W.... M.%^=...
```

7) ServerHelloDone - обозначает окончание пакета сообщений, возглавляемого ServerHello. Это сообщение имет нулевую длину (однако заголовок никто не отменял, поэтому в TLS-записи ServerHelloDone соответствует 4 байта) и служит простым флагом, обозначающим, что сервер передал свою часть начальных данных и теперь ожидает ответа от клиента.

Сообщение ServerHelloDone (тип - 0x0e = 14). Это сообщение имеет нулевую длину, поэтому следом за типом идут три байта с нулевыми значениями (это обозначающие длину байты).

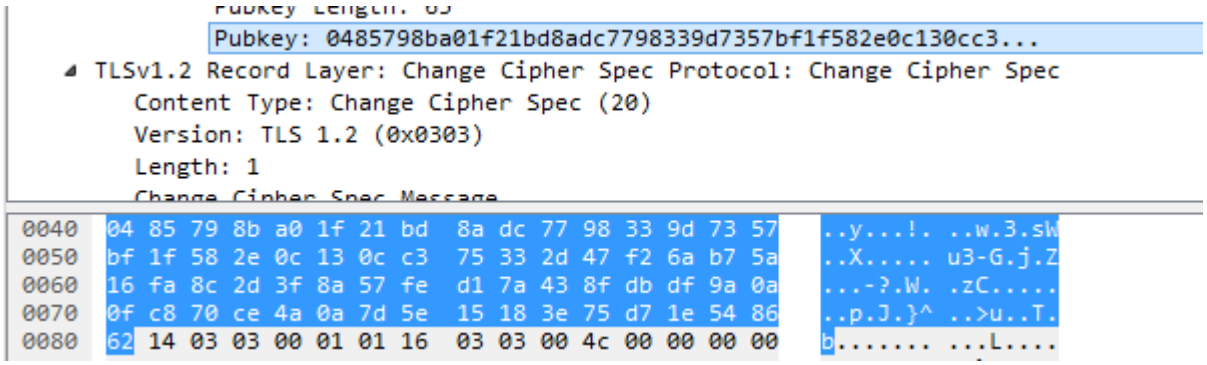


8) Следующий шаг установки TLS соединения ClientKeyExchange

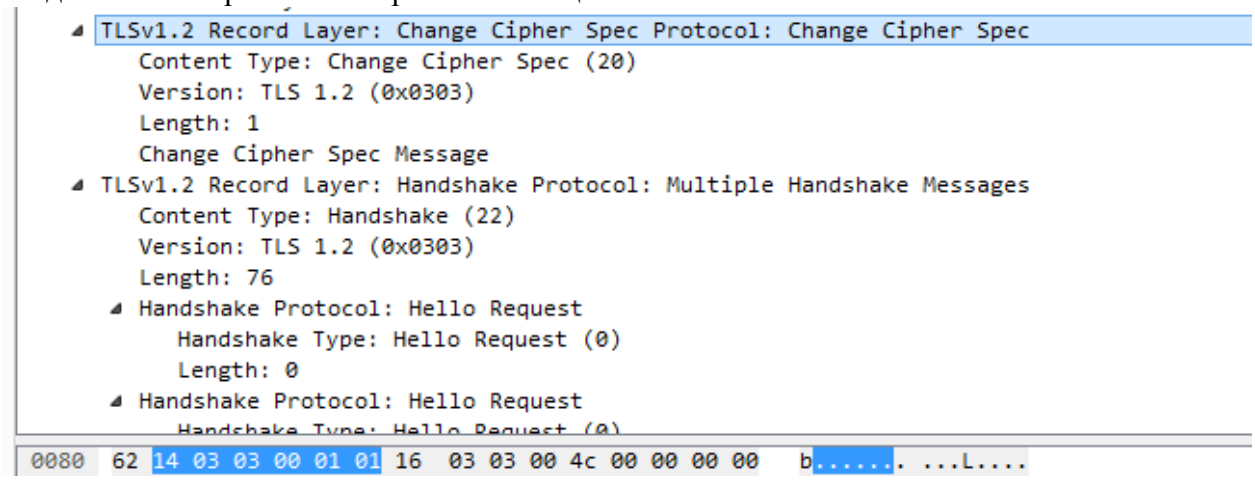


ClientKeyExchange - клиентская часть обмена данными, позволяющими узлам получить общий сеансовый. В этом случае, ClientKeyExchange содержит открытый ключ DH. Этот ключ генерируется клиентом либо в соответствии с параметрами, переданными сервером в ServerKeyExchange, либо в соответствии с параметрами, указанными в серверном SSL-сертификате, если последний поддерживает DH ключ.

Собственно сам ключ.



9) ChangeCipherSpec - это специальное сообщение-сигнал, обозначающее, что с данного момента клиент переходит на выбранный шифр, а следующие *TLS-записи* будут зашифрованы. ChangeCipherSpec (CCS) имеет собственный тип и, соответственно, передаётся в отдельной TLS-записи. Таким образом, CCS имеет весьма важное значение в TLS, отделяя открытую часть сеанса связи от закрытой. Со стороны клиента установление соединения завершается отправкой сообщения Finished.



10) Finished. Это сообщение передаётся в зашифрованной TLS-записи, так как следует за сигналом ChangeCipherSpec, который обозначает момент перехода к защищённому обмену информацией. Finished является первым защищённым сообщением, в рамках нового сеанса TLS. К моменту его отправки, если всё прошло успешно, сервер и клиент уже согласовали все необходимые параметры сессии: шифронабор, сеансовый ключ, алгоритм кода аутентификации сообщения. Предназначение сообщений Finished - получить криптографически стойкое подтверждение, что сервер согласовал эти параметры именно с тем узлом, с которым предполагалось, то же самое - и для клиента, который получит сообщение Finished от сервера. Клиентское Finished содержит отпечаток - значение хеш-функции, - от всех предыдущих сообщений Handshake, отправленных, на данный момент, и клиентом, и сервером. Получив это сообщение в защищённой TLS-записи, сервер может вычислить хеш-сумму от известных ему предшествовавших сообщений и сравнить результат. Таким образом подтверждается подлинность сообщений и, соответственно, оказываются криптографически защищены выбранный шифронабор и другие параметры соединения.

5. Расшифрование соединения TLS в Wireshark

Wireshark поддерживает в некоторых случаях возможность расшифрования соединения TLS при известном ключе. Примеры реализации подобных возможностей, которые требуют предварительной разведки и доступности ключа, описаны в различных источниках[21].

[21]https://habrahabr.ru/company/billing/blog/261301/ https://habrahabr.ru/post/253521/

.

6. Порядок выполнения лабораторной работы

Для выполнения лабораторной работы осуществить следующие действия:

1) Изучить информацию приведенную в разделе 3.
2) Проверить установку и при необходимости установить Wireshark и Firefox на компьютер.
3) Выбрать не менее 5-ти веб-серверов различной организационной и государственной принадлежности, как это указано в разделе 2.
4) Выбрать исследуемый веб-сервер
5) Запустить Wireshark и используя Firefox установить https соединение с выбранным сервером.
6) Остановить Wireshark и провести анализ соединения как это описано в разделе 4
7) Сохранить данные необходимы для последующего сравнительного анализа:
   a) Имя сервера, его характеристики
   b) Версия TLS
   c) Выбранные алгоритмы шифрования
   d) Полученный сертификат: версия. валидность сертификата, валидность ключа, удостоверяющий центр.
   e) Время установки соединения (от ClientHello до Finisfed)
8) Если список исследуемых серверов не исчерпан выбрать другой сервер и вернуться к пункту 3)
9) Если браузер поддерживал соединение TLS 1.2 принудительно изменить параметры TLS соединения в Firefox на TLS 1.0 (в браузере перейти по “about:config” и изменить раздел SSL\TLS) и перейти к пункту 4).
10) Провести сравнительный анализ полученной информации.
11) В качестве отчета представить результаты сравнительного анализа, выводы в отношении безопасности и корректности настройки веб-серверов с учетом их организационной и государственной принадлежности.