

Протокол TLS

- Основные задачи протокола TLS
- Уровни протокола TLS
- Протокол Записи
- Протокол Рукопожатия
- Добавление дополнительных возможностей в протокол

Основные задачи протокола TLS

- Основная функция протокола TLS состоит в обеспечении конфиденциальности и целостности данных прикладного уровня, передаваемых между двумя взаимодействующими приложениями, одно из которых является клиентом, а другое - сервером.
- Протокол TLS (Transport Layer Security) разрабатывался на основе спецификации протокола SSL 3.0 (Secure Socket Layer), опубликованного корпорацией Netscape. Различия между данным протоколом и SSL 3.0 несущественны, но важно заметить, что TLS 1.0 и SSL 3.0 несовместимы, хотя в TLS 1.0 предусмотрен механизм, который позволяет TLS иметь обратную совместимость с SSL 3.0.

Основные задачи протокола TLS

■ Перечислим задачи протокола TLS в порядке их приоритета:

1. **Криптографическая безопасность:** TLS должен использоваться для установления криптографически безопасного соединения между двумя участниками.
2. **Интероперабельность:** независимые разработчики должны иметь возможность создавать приложения, которые будут взаимодействовать по протоколу TLS, что позволит устанавливать безопасные соединения.
3. **Расширяемость:** TLS определяет общий каркас, в который могут быть встроены новые алгоритмы открытого ключа и симметричного шифрования. Это избавляет от необходимости создавать новый протокол для использования новых алгоритмов, что сопряжено с опасностью появления новых слабых мест, и исключает необходимость полностью реализовывать новую библиотеку криптографических алгоритмов.
4. **Относительная эффективность:** криптографические операции интенсивно используют ЦП, особенно операции с открытым ключом. Для уменьшения вычислительной нагрузки вводится понятие сессии, в рамках которой может быть создано несколько TCP-соединений. TLS позволяет кэшировать параметры сессии для уменьшения количества выполняемых криптографических операций при установлении соединения. Это снижает нагрузку как на ЦП, так и на трафик.

Уровни протокола TLS

■ Протокол состоит из двух уровней. Нижним уровнем, расположенным выше некоторого надежного протокола (а именно, протокола TCP) является протокол Записи. Протокол Записи обеспечивает безопасность соединения, которая основана на следующих двух свойствах:

- **Конфиденциальность соединения.** Для защиты данных используется один из алгоритмов симметричного шифрования. Ключ для этого алгоритма создается для каждой сессии и основан на секрете, о котором договариваются в протоколе Рукопожатия. Протокол Записи также может использоваться без шифрования.
- **Целостность соединения.** Обеспечивается проверка целостности сообщения с помощью MAC с ключом. Для вычисления MAC используются хеш-функции SHA-1 и MD5. Протокол Записи может выполняться без вычисления MAC.

Уровни протокола TLS

- Протокол Записи используется для инкапсуляции различных протоколов более высокого уровня. Одним из протоколов более высокого уровня является протокол Рукопожатия, который использует протокол Записи в качестве транспорта для ведения переговоров о параметрах безопасности. Протокол Рукопожатия позволяет серверу и клиенту аутентифицировать друг друга и договориться об алгоритмах шифрования и криптографических ключах до того, как прикладной протокол, выполняющийся на том же уровне, начнет передавать или принимать первые байты данных.

Уровни протокола TLS

■ Протокол Рукопожатия обеспечивает безопасность соединения, которая основана на следующих свойствах:

1. Участники **аутентифицированы** с использованием криптографии с открытым ключом (т.е. с использованием алгоритмов RSA, DSS и т.д.). Эта аутентификация может быть необязательной, но обычно требуется по крайней мере для сервера.
2. Переговоры о разделяемом секрете **безопасны**, т.е. этот общий секрет невозможно подсмотреть.
3. Переговоры о разделяемом секрете **надежны**, если выполнена аутентификация хотя бы одной из сторон. В таком случае атакующий, расположенный в середине соединения, не может модифицировать передаваемый секрет незаметно для участников соединения.

Уровни протокола TLS

- Одно из преимуществ TLS состоит в том, что он **независим от прикладного протокола**.
- Определены следующие криптографические операции: цифровая подпись, блочное шифрование и шифрование с открытым ключом.
- При блочном шифровании каждый блок незашифрованного текста шифруется, в результате чего создается блок зашифрованного текста. Все алгоритмы блочного шифрования выполняются в режиме CBC, и длина всех шифруемых элементов должна быть кратна длине блока алгоритма шифрования.

Уровни протокола TLS



Протокол Записи

- Протокол Записи состоит из нескольких уровней. Протокол Записи фрагментирует сообщение на блоки нужной длины, осуществляет сжатие данных, вычисляет НМАС и зашифровывает их. На другом конце соединения полученные данные расшифровываются, проверяется их целостность, далее они декомпрессируются, дефрагментируются и передаются протоколам более высокого уровня.
- Выше протокола Записи могут располагаться следующие протоколы: протокол Рукопожатия, Alert-протокол, протокол изменения шифрования и прикладной протокол, безопасность которого обеспечивается.

Протокол Записи

■ Состояния соединения

В протоколе вводится понятие состояния соединения, которое определяет **параметры выполнения протокола Записи**. Такими параметрами являются алгоритм сжатия, алгоритм шифрования и MAC-алгоритм, а также параметры этих алгоритмов, т.е. секреты MAC, ключи алгоритма шифрования и инициализационные вектора. Для каждого направления (соответственно **чтение** или **запись**) параметры соединения могут различаться. Существует **четыре состояния** соединения: **текущие состояния** чтения и записи и **ожидаемые состояния** чтения и записи. Параметры безопасности для ожидаемых состояний устанавливаются протоколом **Рукопожатия**, а протокол изменения шифрования делает ожидаемое состояние текущим, в результате чего соответствующие параметры текущего состояния сбрасываются и заменяются параметрами ожидаемого состояния. Параметры ожидаемого состояния инициализируются пустыми значениями. Вначале текущее состояние всегда определяется без использования шифрования, сжатия и MAC.

Протокол Записи

■ Определены следующие параметры состояния:

Конец соединения	Каждый участник является либо "клиентом", либо "сервером"
Алгоритм симметричного шифрования	Алгоритм, используемый для симметричного шифрования, и его параметры – длина ключа алгоритма, длина блока алгоритма, ключ шифрования, инициализационный вектор (IV) и др.
MAC алгоритм	Алгоритм, используемый для проверки целостности сообщения, секрет MAC.
Алгоритм сжатия	Алгоритм, используемый для сжатия данных.
Мастер-секрет	48-байтный секрет, разделяемый обоими участниками соединения.
Случайное число клиента SecurityParameters.client_random	32-байтное значение, создаваемое клиентом в протоколе Рукопожатия.
Случайное число сервера SecurityParameters.server_random	32-байтное значение, создаваемое сервером в протоколе Рукопожатия.
Последовательный номер	Каждое состояние соединения содержит последовательный номер, который вычисляется независимо для состояний чтения и записи. Последовательный номер должен устанавливаться в ноль при инициализации состояния. Последовательные номера не могут быть больше $2^{64} - 1$. Последовательный номер возрастает после создания очередной записи.

Протокол Записи

■ Из мастер-секрета создаются шесть ключей:

- `client write MAC secret`
- `server write MAC secret`
- `client write key`
- `server write key`
- `client write IV`
- `server write IV`

■ Ключи **client write** используются сервером, когда он получает сообщения и клиентом, когда тот посылает их. Ключи **server write** используются сервером, когда он посылает сообщения и клиентом, когда он получает их. После того как параметры безопасности установлены и ключи созданы, ожидаемые состояния соединения делаются текущими.

Протокол Записи

- Вычисление ключей
 - Протокол Записи использует следующий алгоритм для создания ключей, инициализационных векторов и секретов MAC из параметров безопасности, создаваемых протоколом Рукопожатия.
- **НМАС и псевдослучайная функция**
 - Для обеспечения целостности используется НМАС с хеш-функциями MD5 и SHA-1, обозначаемыми как **НМАС_MD5 (secret, data)** и **НМАС_SHA (secret, data)**.
 - В алгоритме определена псевдослучайная функция **PRF**, которая расширяет секрет до нужной длины для создания всех необходимых ключей. Эта функция получает в качестве входа секрет, "зерно" (seed - значение, которое с одной стороны является случайным, а с другой стороны не является секретным, т.е. может стать известно оппоненту) и стандартное значение, и создает выход требуемой длины.

Протокол Записи

Сначала определяется функция расширения данных **P_hash (secret, data)**, которая использует хеш-функцию для расширения секрета до нужной длины следующим образом:

```
P_hash (secret, seed) = HMAC_hash (secret, A(1) || seed) ||  
    HMAC_hash (secret, A(2) || seed) ||  
    HMAC_hash (secret, A(3) || seed) ||
```

A(i) определяется следующим образом:

```
A(0) = seed
```

```
A(i) = HMAC_hash (secret, A (i - 1))
```

Протокол Записи

- **P_hash** может иметь столько итераций, сколько необходимо для создания данных требуемой длины. Например, если **P_SHA-1** используется для создания 64 байтов данных, то количество итераций должно быть равно 4, при этом будет создано 80 байтов данных; последние 16 байтов заключительной итерации будут отброшены, чтобы оставить только 64 байта выходных данных.
- Для получения ключевого материала нужной длины секрет, вычисленный в протоколе Рукопожатия, делится на две половины, одна половина используется для создания данных с помощью **P_MD5**, а другая - для создания данных с помощью **P_SHA-1**.
- **PRF** определяется как результат сложения по модулю 2 результатов выполнения **P_MD5** и **P_SHA-1**.

$$\text{PRF}(\text{secret}, \text{label}, \text{seed}) = \text{P_MD5}(\text{S1}, \text{label} + \text{seed}) \oplus \text{P_SHA-1}(\text{S2}, \text{label} + \text{seed})$$

- **Label** является фиксированной текстовой строкой.

Протокол Записи

■ Заметим, что поскольку MD5 создает 16-байтные значения, а SHA-1 создает 20-байтные значения, то количество итераций каждой из функций будет разным. Например, для создания 80-байтного значения необходимо выполнить 5 итераций **P_MD5** и 4 итерации **P_SHA-1**.

■ Для создания ключей вычисляется следующее значение:

```
key_block = PRF (SecurityParameters.master_secret,  
"key expansion",  
SecurityParameters.server_random + SecurityParameters.client_random)
```

■ Количество итерация в **PRF** определяется суммарной длиной ключей. Затем **key_block** разбивается на блоки для получения требуемых ключей.

Протокол Рукопожатия

- Протокол Рукопожатия состоит из трех подпротоколов, использование которых позволяет участникам согласовать криптографические алгоритмы, аутентифицировать друг друга, и сообщить друг другу о возникновении тех или иных ошибок.
- В результате выполнения протокола Рукопожатия будут созданы следующие элементы сессии:

Идентификатор сессии	Произвольная последовательность байтов, выбираемая сервером для идентификации активного или возобновляемого состояния сессии.
Сертификат участника	X.509 v3 сертификат участника. Этот элемент может быть нулевым.
Метод сжатия	Алгоритм, используемый для сжатия данных перед шифрованием.
Набор алгоритмов	Алгоритм симметричного шифрования данных (например, NULL, DES, AES и т.д.), MAC-алгоритм (такой как MD5 или SHA-1) и параметры этих алгоритмов.
Мастер-секрет	48-байтный секрет, разделяемый клиентом и сервером.
Возобновляемо 2025	Флаг, определяющий, может ли данная сессия использоваться для создания нового TCP-соединения.

Протокол Рукопожатия

Протокол изменения шифрования

- Протокол состоит из единственного сообщения, которое зашифровано и сжато, как определено в текущем состоянии соединения.
- Сообщение об изменении шифрования посылается как клиентом, так и сервером для уведомления противоположной стороны о том, что следующие записи будут защищены алгоритмами и ключами, о которых стороны только что договорились. При получении данного сообщения протокол Записи копирует ожидаемое состояние чтения в текущее состояние чтения. Сразу после отправки данного сообщения отправитель копирует ожидаемое состояние записи в текущее состояние записи. Сообщение об изменении шифрования посылается при Рукопожатии после того, как параметры безопасности согласованы, но перед тем, как посылается заключительное верифицирующее сообщение.

Протокол Рукопожатия

Alert-протокол

■ Одним из протоколов, выполняющихся выше протокола Записи, является протокол Alert. Содержимым протокола является либо фатальное, либо предупреждающее сообщение. Фатальное сообщение должно приводить к немедленному разрыву данного ТСП-соединения. В этом случае другие соединения, соответствующие данной сессии, могут быть продолжены, но идентификатор сессии должен быть помечен как недействительный для предотвращения использования данной сессии для установления новых соединений. Подобно другим сообщениям, сообщения Alert зашифрованы и сжаты, как определено в текущем состоянии соединения.

Протокол Рукопожатия

■ Криптографические параметры сессии создаются протоколом Рукопожатия, который выполняется выше протокола Записи. Когда клиент и сервер начинают взаимодействовать, они согласовывают версию протокола, выбирают криптографические алгоритмы, могут аутентифицировать друг друга, используя технологию с открытым ключом. Для создания разделяемого секрета также используется технология с открытым ключом.

■ Протокол Рукопожатия состоит из следующих шагов:

1. Обмен сообщениями **Hello** для согласования алгоритмов, обмена случайными значениями и проверки возобновляемости сессии.
2. Обмен необходимыми **криптографическими параметрами**, которые позволяют клиенту и серверу **согласовать премастер-секрет**.
3. Обмен **сертификатами и криптографической информацией**, которая позволяет клиенту и серверу **аутентифицировать** друг друга.
4. Предоставление **параметров безопасности на уровень Записи**.
5. Возможность клиенту и серверу **проверить**, что они вычислили одни и те же параметры безопасности и что Рукопожатие произошло без вмешательства злоумышленника.

Протокол Рукопожатия

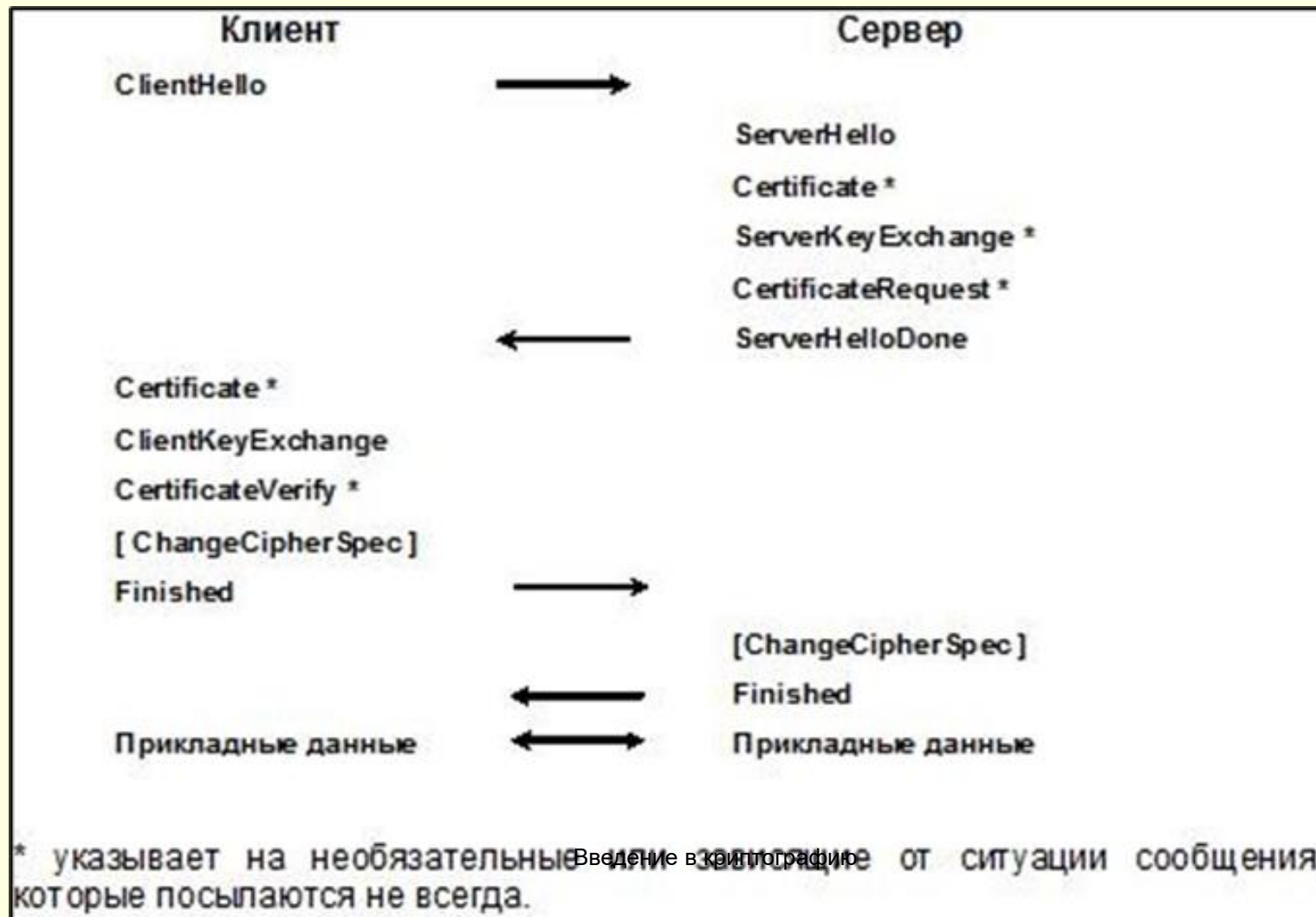
- Протокол разработан для минимизации риска атак «встреча посередине», но защита от атак, при которых злоумышленник может блокировать доступ к порту, не предусмотрена.
- Клиент посылает сообщение **ClientHello**, на которое сервер должен ответить сообщением **ServerHello** или фатальной ошибкой и разрывом соединения. **ClientHello** и **ServerHello** используются для определения максимального уровня безопасности между клиентом и сервером. **Client Hello** и **Server Hello** устанавливают следующие атрибуты: **Protocol Version**, **Session ID**, **Cipher Suite** и **Compression Method**. Дополнительно создаются и передаются два случайных значения: **ClientHello.random** и **ServerHello.random**.
- Аутентификация и обмен общим секретом осуществляются в четырех сообщениях: сертификат сервера, обмен ключа сервера, сертификат клиента и обмен ключа клиента. Общий секрет должен быть достаточно большим; текущие методы распределения ключа обмениваются секретами, длина которых находится в диапазоне от 48 до 126 байт.

Протокол Рукопожатия

■ После сообщений **Hello** сервер посылает сертификат, с помощью которого клиент выполняет аутентификацию сервера. Дополнительно может быть послано сообщение обмена ключа сервера, если сервер не имеет сертификата или его сертификат может использоваться только для проверки подписи. Если сервер аутентифицирован, он может запросить сертификат клиента, если того требует установленная политика безопасности на стороне сервера. После этого сервер посылает сообщение **Server Hello Done**, указывающее на то, что фаза **Hello**-сообщений рукопожатия завершена. Затем сервер ждет ответа клиента. Если сервер послал сообщение запроса сертификата, клиент должен послать сообщение **Certificate**. После этого посылается сообщение обмена ключа клиента. Содержимое этого сообщения зависит от выбранного алгоритма выработки общего секрета. Если клиент посылал свой сертификат, то он посылает сообщение, содержащее цифровую подпись для проверки всех сообщений Рукопожатия.

■ В данной точке клиентом посылается сообщение об изменении состояния, и клиент копирует ожидаемое состояние в текущее состояние. После этого клиент посылает заключительное сообщение с использованием новых алгоритмов, ключей и секретов. В ответ сервер посылает свое сообщение об изменении состояния, преобразует ожидаемое состояние в текущее состояние и посылает заключительное сообщение с использованием новых алгоритмов и ключей. После этого рукопожатие считается выполненным, и клиент и сервер могут начинать обмен данными прикладного уровня.

Протокол Рукопожатия



Протокол Рукопожатия

■ Когда клиент и сервер решают возобновить предыдущую сессию или дублировать существующую (вместо того чтобы вести новые переговоры о параметрах безопасности), выполняется так называемое сокращенное рукопожатие:

- Клиент посылает **Client Hello**, используя **Session ID** возобновляемой сессии.
- Сервер ищет соответствующий идентификатор сессии в своем кэше сессий. Если идентификатор существует, и сессия помечена как возобновляемая, сервер устанавливает соединение с параметрами указанной сессии, после чего посылает **Server Hello** с этим значением **Session ID**. Если соответствующий **Session ID** не найден, сервер создает новый ID сессии, и клиент и сервер выполняют полное рукопожатие.
- После этого и клиент, и сервер должны послать сообщения об изменении состояния, затем сразу послать завершающие сообщения.
- И клиент, и сервер начинают обмен данными прикладного уровня.

Протокол Рукопожатия

- Поток сообщений при сокращенном Рукопожатии



Протокол Рукопожатия

- Следует заметить, что, так как **Session ID** передается без шифрования и обеспечения целостности, он не содержит конфиденциальную информацию. Содержимое всего Рукопожатия, включая **Session ID**, защищено **Finished**-сообщениями, которыми участники обмениваются в конце рукопожатия.
- Список **Cipher Suite**, передаваемый от клиента серверу в сообщении **Client Hello**, содержит перечень криптографических алгоритмов, поддерживаемых клиентом, упорядоченный по предпочтениям клиента. Сервер выбирает по одному алгоритму из каждой категории, который он поддерживает. Если такого алгоритма не существует, сервер возвращает фатальный **Alert** и закрывает соединение.
- После отправки сообщения **Client Hello** клиент ждет сообщения **Server Hello**. Любое другое сообщение, возвращаемое сервером, за исключением **Hello Request**, трактуется как фатальная ошибка.
- Сервер посылает **Server Hello** в ответ на сообщение **Client Hello**, для того чтобы выбрать конкретный набор алгоритмов. Если для какого-то типа алгоритмов клиент и сервер не имеют одинакового алгоритма, TCP-соединение будет сброшено.

Протокол Рукопожатия

Сообщение **Certificate** (сервера)

- Сервер должен посылать сертификат, если метод обмена ключей не является анонимным. Данное сообщение всегда следует сразу за сообщением **Server Hello**.
- Тип сертификата должен соответствовать выбранному алгоритму обмена ключа. Обычно это сертификат X.509v3. Он должен содержать ключ, который соответствует методу обмена ключа.
- Сообщение сервера обмена ключа посылается сервером только тогда, когда сообщение **Server Certificate** (если оно послано) не содержит достаточно данных для того, чтобы клиент мог осуществить обмен премастер-секретом.
- Данное сообщение передает криптографическую информацию, которая позволяет клиенту передавать премастер-секрет: премастер-секрет шифруется либо открытым ключом RSA, либо открытым ключом Диффи-Хеллмана.

Сообщение **Certificate Request**

- Неанонимный сервер может дополнительно запросить сертификат клиента, если это требуется политикой безопасности сервера.

Протокол Рукопожатия

Сообщение **Server Hello Done**

- Сообщение **Server Hello Done** посылается сервером как признак окончания фазы **Server Hello**.

Сообщение **Certificate** (клиента)

- Это первое сообщение, которое клиент посылает после получения сообщения **Server Hello Done**. Оно посылается только в том случае, если сервер запросил аутентификацию клиента.

Сообщение **Client Key Exchange**

- Данное сообщение посылается клиентом всегда. Оно следует сразу за сообщением **Client Certificate**, если оно посылалось. В противном случае это первое сообщение, посланное клиентом после получения сообщения **Server Hello Done**.
- После получения данного сообщения сервер может вычислить премастер-секрет, который передается либо с помощью RSA шифрования, либо вычисляется по алгоритму Диффи-Хеллмана. В любом случае каждая сторона вычисляет один и тот же премастер-секрет.

Протокол Рукопожатия

Проверка целостности с помощью сертификата клиента

■ Данное сообщение используется для выполнения проверки целостности переданных и полученных сообщений Рукопожатия и аутентификации клиента. Оно посылается только в том случае, если алгоритм открытого ключа, для которого создан сертификат клиента, имеет возможность подписывания. Это означает, что исключением являются сертификаты, созданные для открытого ключа алгоритма Диффи-Хеллмана.

Сообщение **Finished**

■ Сообщение **Finished** всегда посылается непосредственно после сообщения **Change Cipher Spec** для проверки успешного выполнения обмена ключа и процессов аутентификации. Сообщение **Change Cipher Spec** должно быть получено после остальных сообщений Рукопожатия и перед **Finished**-сообщением.

■ **Finished**-сообщение является первым сообщением, защищенным с помощью только что обговоренных алгоритмов и ключей. Получатели **Finished**-сообщения должны убедиться, что его содержимое корректно. После того как одна сторона послала свое **Finished**-сообщение, получила и проверила **Finished**-сообщение другой стороны, она может начинать посылать и получать

Протокол Рукопожатия

Вычисление мастер-секрета

- Независимо от методов обмена ключа используется следующий алгоритм для преобразования премастер-секрета в мастер-секрет. Премастер-секрет должен быть удален после того, как вычислен мастер-секрет.

```
master_secret = PRF(pre_master_secret, "master  
secret", ClientHello.random+ServerHello.random) [0..47]
```

- Длина мастер-секрета всегда равна 48 байтам. Длина премастер-секрета изменяется в зависимости от метода обмена ключа.

Добавление дополнительных возможностей в протокол

- Рассмотрим расширения, которые позволяют добавлять новые функциональности в TLS. Рассмотрим общие механизмы расширений для протокола Рукопожатия и конкретные расширения, используемые для добавления новых возможностей.
- Сейчас TLS используется в самых разных окружениях, возможности которых не учитывались при разработке протокола. Данные расширения разработаны для того, чтобы TLS мог максимально эффективно функционировать в новых окружениях, такие, например, как беспроводные сети.
- Беспроводные окружения часто имеют ряд ограничений, обычно отсутствующих в других окружениях. Эти окружения могут иметь ограничения на полосу пропускания, на вычислительные мощности клиента, на объем памяти и т.п.

Добавление дополнительных возможностей в протокол

■ Данные расширения предназначены для обеспечения следующих возможностей:

- Позволить клиентам предоставлять серверу имя сервера, с которым они устанавливают соединение. Данная функциональность обеспечивает возможность установления безопасных соединений с хостами, имеющими несколько виртуальных серверов на одном сетевом адресе.
- Вести переговоры о максимальной длине передаваемых фрагментов. Данная функциональность необходима из-за ограничений памяти у некоторых клиентов и ограничений полосы пропускания в некоторых типах сетей.
- Вести переговоры об использовании URL для указания сертификатов клиента. Данная функциональность нужна для экономии памяти клиента.
- Позволить клиентам указать серверам сертификаты каких корневых СА они имеют. Данная функциональность необходима, чтобы клиенты с ограниченной памятью могли хранить только небольшое число сертификатов корневых СА.
- Позволить клиентам и серверам вести переговоры об использовании урезанных MAC. Данная функциональность дает возможность уменьшить трафик, что может быть необходимо в определенных типах сетей.

Добавление дополнительных возможностей в протокол

- Позволить клиентам и серверам вести переговоры о том, чтобы сервер при Рукопожатии посылал клиенту информацию о статусе сертификата (например, ответ OCSP). Данная функциональность позволяет избежать посылки CRL и тем самым сократить трафик и вычислительную нагрузку на клиента.
 - Для того чтобы поддерживать перечисленные выше расширения, вводятся дополнительные механизмы для сообщений **Hello** клиента и сервера.
 - Описываемые расширения могут использоваться клиентами и серверами, поддерживающими версию TLS 1.0. Расширения поддерживают обратную совместимость – это означает, что клиенты версии TLS 1.0, которые поддерживают расширения, могут общаться с серверами TLS 1.0, не поддерживающими расширения, и наоборот.
 - Обратная совместимость достигается следующим образом.
- Клиент запрашивает использование расширений с помощью расширенного сообщения **Client Hello**, описанного ниже. TLS 1.0 требует, чтобы серверы принимали расширенные сообщения **Client Hello**, даже если они не понимают расширения.

Добавление дополнительных возможностей в протокол

- Сервер может не посылать никакого ответа на расширения, которые он не поддерживает.
 - Однако заметим, что хотя обратная совместимость требуется, некоторые клиенты из могут не устанавливать соединения с серверами, которые не поддерживают требуемые клиентам расширения.
 - В качестве имени сервера как правило, поддерживается только DNS-имя.
 - Сервер, который получил сообщение **Client Hello**, содержащее расширение **Server Name**, может использовать данную информацию для выбора сертификата, возвращаемого клиенту, либо для каких-то других аспектов безопасности. В данном случае сервер должен включить расширение типа **Server Name** в расширенное **Server Hello**.

Добавление дополнительных возможностей в протокол

Переговоры о максимальной длине фрагмента

- Препятствие версии TLS указывает, что максимальная длина незашифрованного фрагмента равна 2^{14} байт. Для некоторых клиентов может потребоваться использовать меньшую длину фрагмента из-за ограничений памяти или ограничений полосы пропускания.
- Сервер, получивший расширенный **Client Hello** с расширением **Max Fragment Length**, может принять эту длину и включить расширение **Max Fragment Length** в **Server Hello**.
- После того, как стороны успешно договорились о максимальной длине фрагмента, отличной от 2^{14} , клиент и сервер должны немедленно начать фрагментировать сообщения (включая сообщения Рукопожатия), чтобы гарантировать, что не посылаются сегменты большей длины, чем та, о которой договорились.
- Новая длина применяется в течение всей сессии, включая возобновляемые сессии.

Добавление дополнительных возможностей в протокол

URL сертификат клиента

- TLS требует, чтобы при выполнении аутентификации клиента сертификаты клиента посылались серверу в протоколе Рукопожатия. Для клиентов, имеющих ограничения на память, существует возможность посылать URL сертификата вместо самих сертификатов, чтобы они могли не хранить свои сертификаты и тем самым не занимать память.
- Для ведения переговоров о посылке серверу URL сертификата клиенты могут включать расширение типа **Client Certificate Url** в **Client Hello**.
- Сервер, получивший расширенный **Client Hello**, содержащий расширение **Client Certificate Url**, может указать, что имеет возможность принимать URL сертификата, указывая расширение типа **Client Certificate Url** в **Server Hello**.

Добавление дополнительных возможностей в протокол

Указание доверенного СА

- Клиенты, которые имеют ограниченную память, могут хранить только небольшое количество сертификатов корневых СА. В этом случае они могут указать серверу, какими корневыми сертификатами они владеют.
- Для этого клиенты могут включить расширение типа **Trusted Ca Keys** в **Client Hello**. Поле **Extension Data** данного расширения должно содержать **Trusted Authorities**.

Добавление дополнительных возможностей в протокол

Урезанный HMAC

- В настоящий момент набор шифрования использует в качестве MAC HMAC либо с MD5, либо с SHA-1 для аутентификации соединений уровня записи. Результат вычисления хеш-функции используется в качестве значения MAC. Однако в некоторых ограниченных окружениях может оказаться желательным использовать 80-битные значения MAC.
- Для того чтобы вести переговоры об использовании 80-битного урезанного MAC, клиенты могут включить расширение **Truncated Hmac** в расширенный **Client Hello**.
- При получении расширенного **Hello**, содержащего расширение **Truncated Hmac**, сервер может согласиться использовать урезанный MAC, включая расширения **Truncated Hmac** с пустым **Extension Data** в расширенный **Server Hello**.

Добавление дополнительных возможностей в протокол

Запрос статуса сертификата

- Клиенты, имеющие ограничения, могут захотеть использовать протокол статуса сертификата, такой как возвращаемый протоколом OCSP, для проверки действительности сертификатов сервера, чтобы избежать получения и проверки CRL и тем самым сохранить ширину полосы пропускания в ограниченных сетях.

Сравнение версий TLS

- **TLS 1.0** - реализован в 1999 и опубликован в [RFC 2246](#). данная версия TLS аналогична SSL 3.0
- **TLS 1.1** - реализован в 2006 и опубликован в [RFC 4346](#).
- **TLS 1.2** - реализован в 2008 и опубликован в [RFC 5246](#).
- **TLS 1.3** - реализован в августе 2018 и опубликован в [RFC 8446](#).

TLS 1.1

- Неявный инициализационный вектор (IV) заменен **явным инициализационным вектором** для защиты от атак на Cipher Block Chaining.
- Padding error handling is **modified to use bad_record_mac alert** rather than decryption_failed alert. Again, to protect against CBC attacks.
- IANA registries are **defined for protocol parameters**.
- A premature close **no longer causes a session to be non-resumable**.
- Additional notes were added regarding new attacks and a number of clarifications and editorial improvements were made.

TLS 1.2

- Комбинация MD5/SHA-1 в псевдослучайной функции (**PRF**) **заменена на SHA-256** с возможностью использования наборов алгоритмов.
- Комбинация MD5/SHA-1 **при подписывании** элемента заменена на **единственный хеш**, о котором ведутся переговоры при Рукопожатии.
- Добавлена поддержка **режимом аутентифицированного шифрования**.
- Добавлены **расширения TLS extensions** и **AES**
- Добавлена поддержка **криптографии на эллиптических кривых (ECC)**
- Добавлено **расширение о ведении повторных переговоров о криптографических параметрах** для существующего соединения для предотвращения найденной уязвимости.
- Добавление (Padding) используется для заполнения открытого текста сообщения так, чтобы его размер был того же размера, что и размер блока шифра, используемого для шифрования сообщения. TLS 1.1 позволяет использовать небезопасные схемы заполнения, такие как схема добавления SSL 3.0/TLS 1.0, которая уязвима для таких атак, как атака BEAST. В TLS 1.2 представлены **новые схемы добавления**, более безопасные и устойчивые к атакам.
- TLS 1.2 **не имеет обратной совместимости** с TLS 1.1 или SSL 3.0.
- TLS 1.2 быстрее, чем TLS 1.1.

TLS 1.3

- **Добавлена** поддержка алгоритмов **AEAD**
- Удалена поддержка алгоритмов RSA и статического Диффи-Хеллмана
- Кроме того, при доступе к сайту, который уже посещался ранее, клиент может отправить данные в первом сообщении на сервер, используя предварительно общие ключи (**PSK**) из предыдущего сеанса — так называемое свойство **0- RTT** - «нулевое время прохождения туда и обратно»
- Все сообщение Рукопожатия **после ServerHello зашифрованы**.
- Введено сообщение **EncryptedExtensions**, которое позволяет различным расширениям, ранее отправленным в открытом виде в ServerHello, посылаться **в зашифрованном виде**.
- Новая функция **вычисления ключевого материала**.
- Алгоритмы на **эллиптических кривых** стали основными для цифровых подписей.