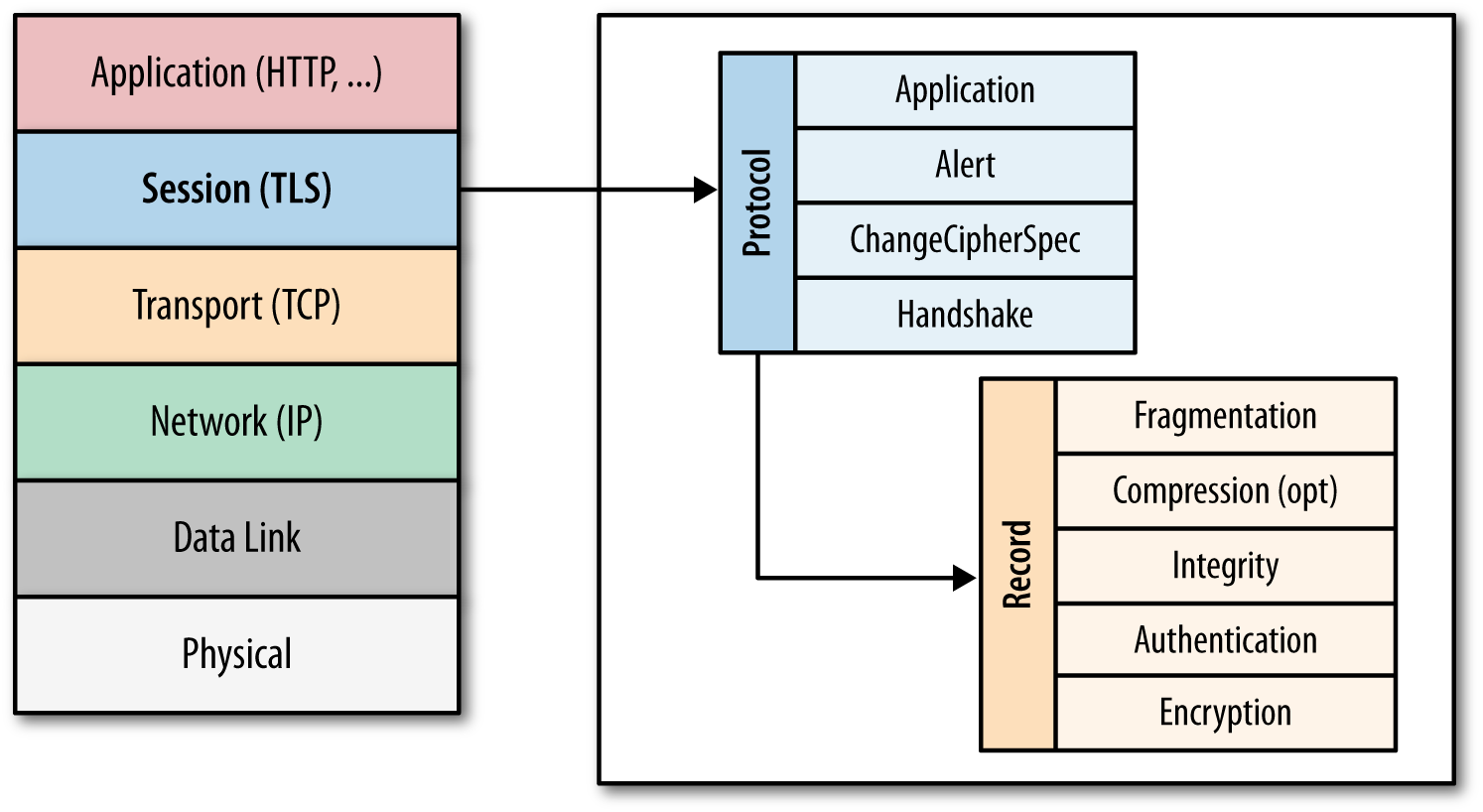
Протокол TLS (transport layer security) основан на протоколе SSL (Secure Sockets Layer), изначально разработанном в Netscape для повышения безопасности электронной коммерции в Интернете. Протокол SSL был реализован на application-уровне, непосредственно над TCP (Transmission Control Protocol), что позволяет 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.  
  
Конкретное место TLS (SSL) в стеке протоколов Интернета показано на схеме:  
  
  
  
После того, как протокол SSL был стандартизирован IETF (Internet Engineering Task Force), он был переименован в TLS. Поэтому хотя имена SSL и TLS взаимозаменяемы, они всё-таки отличаются, так как каждое описывает другую версию протокола.  
  
Первая выпущенная версия протокола имела название SSL 2.0, но была довольно быстра заменена на SSL 3.0 из-за обнаруженных уязвимостей. Как уже упоминалось, SSL был разработан компанией Netscape, так что 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 известные уязвимости. Актуальная версия протокола на данный момент – TLS 1.2, выпущенная в августе 2008 года.  
  
Как уже говорилось, TLS был разработан для работы над TCP, однако для работы с протоколами дейтаграмм, такими как UDP (User Datagram Protocol), была разработана специальная версия TLS, получившая название DTLS (Datagram Transport Layer Security).  
 

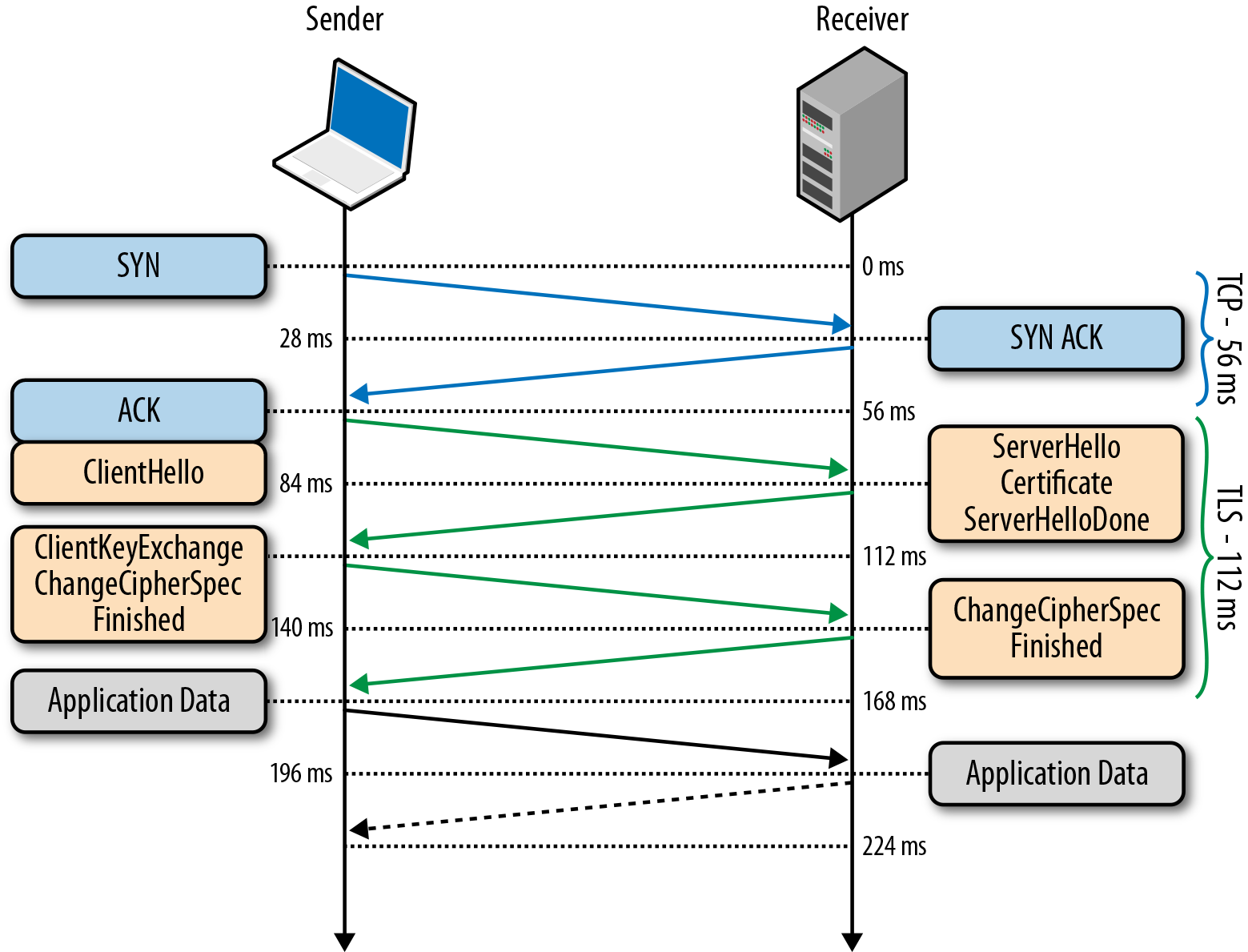
##### Шифрование, аутентификация и целостность

Протокол TLS предназначен для предоставления трёх услуг всем приложениям, работающим над ним, а именно: шифрование, аутентификацию и целостность. Технически, не все три могут использоваться, однако на практике, для обеспечения безопасности, как правило используются все три:

* Шифрование – сокрытие информации, передаваемой от одного компьютера к другому;
* Аутентификация – проверка авторства передаваемой информации;
* Целостность – обнаружение подмены информации подделкой.

Для того чтобы установить криптографически безопасный канал данных, узлы соединения должны согласовать используемые методы шифрования и ключи. Протокол TLS однозначно определяет данную процедуру, подробнее это рассмотрено в пункте TLS Handshake. Следует отметить, что TLS использует криптографию с открытым ключом, которая позволяет узлам установить общий секретный ключ шифрования без каких-либо предварительных знаний друг о друге.  
  
Также в рамках процедуры TLS Handshake имеется возможность установить подлинность личности и клиента, и сервера. Например, клиент может быть уверен, что сервер, который предоставляет ему информацию о банковском счёте, действительно банковский сервер. И наоборот: сервер компании может быть уверен, что клиент, подключившийся к нему – именно сотрудник компании, а не стороннее лицо (данный механизм называется Chain of Trust и будет рассмотрен в соответствующем разделе).  
  
Наконец, TLS обеспечивает отправку каждого сообщения с кодом MAC (Message Authentication Code), алгоритм создания которого – односторонняя криптографическая функция хеширования (фактически – контрольная сумма), ключи которой известны обоим участникам связи. Всякий раз при отправке сообщения, генерируется его MAC-значение, которое может сгенерировать и приёмник, это обеспечивает целостность информации и защиту от её подмены.  
  
Таким образом, кратко рассмотрены все три механизма, лежащие в основе криптобезопасности протокола TLS.  
 

##### TLS Handshake

Перед тем, как начать обмен данными через TLS, клиент и сервер должны согласовать параметры соединения, а именно: версия используемого протокола, способ шифрования данных, а также проверить сертификаты, если это необходимо. Схема начала соединения называется TLS Handshake и показана на рисунке:  
  
  
Разберём подробнее каждый шаг данной процедуры:

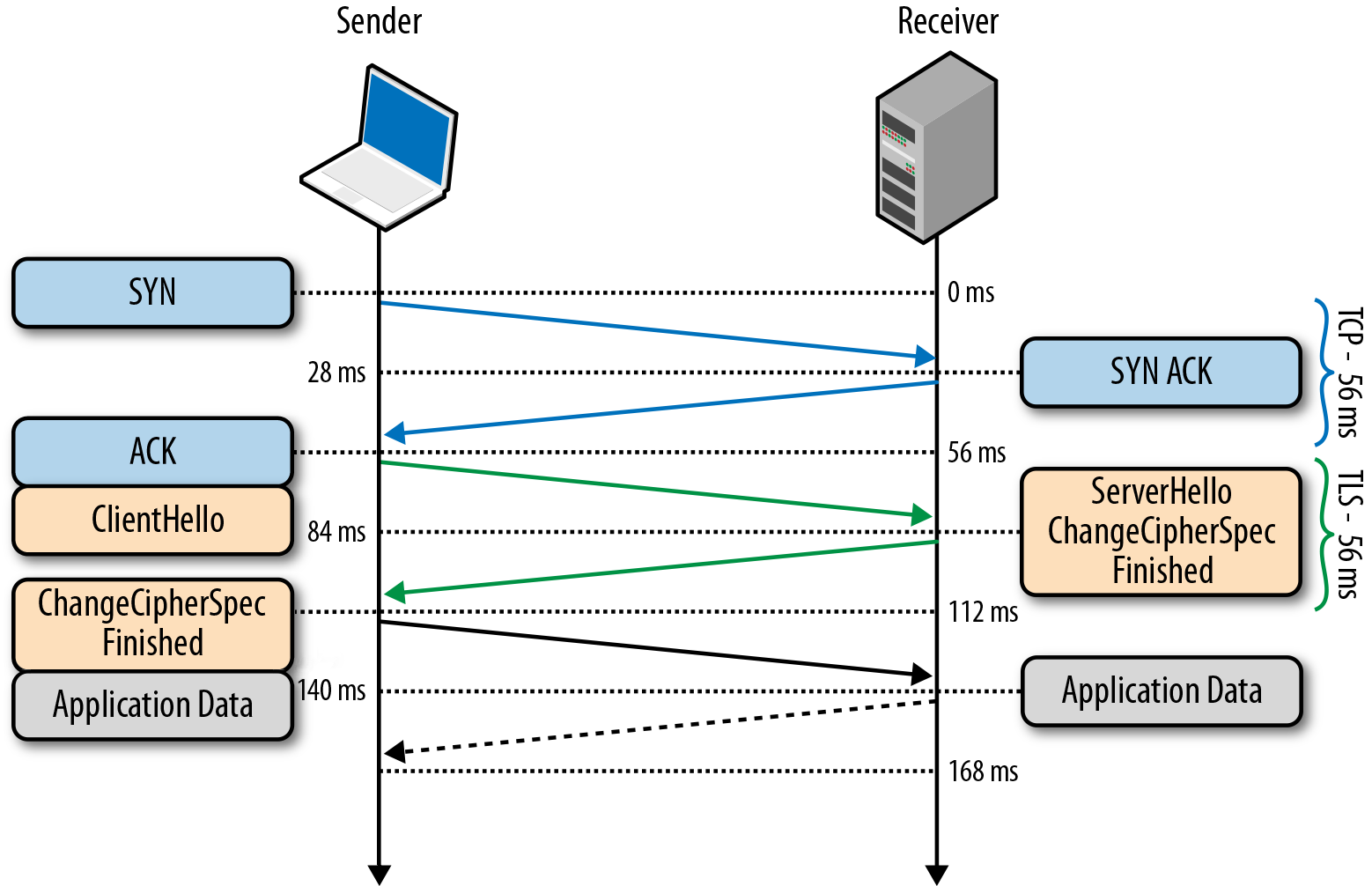
1. Так как TLS работает над TCP, для начала между клиентом и сервером устанавливается TCP-соединение.
2. После установки TCP, клиент посылает на сервер спецификацию в виде обычного текста (а именно версию протокола, которую он хочет использовать, поддерживаемые методы шифрования, etc).
3. Сервер утверждает версию используемого протокола, выбирает способ шифрования из предоставленного списка, прикрепляет свой сертификат и отправляет ответ клиенту (при желании сервер может так же запросить клиентский сертификат).
4. Версия протокола и способ шифрования на данном моменте считаются утверждёнными, клиент проверяет присланный сертификат и инициирует либо RSA, либо обмен ключами по Диффи-Хеллману, в зависимости от установленных параметров.
5. Сервер обрабатывает присланное клиентом сообщение, сверяет MAC, и отправляет клиенту заключительное (‘Finished’) сообщение в зашифрованном виде.
6. Клиент расшифровывает полученное сообщение, сверяет MAC, и если всё хорошо, то соединение считается установленным и начинается обмен данными приложений.

Ясно, что установление соединения TLS является, вообще говоря, длительным и трудоёмким процессом, поэтому в стандарте TLS есть несколько оптимизаций. В частности, имеется процедура под названием “abbreviated handshake”, которая позволяет использовать ранее согласованные параметры для восстановления соединения (естественно, если клиент и сервер устанавливали TLS-соединение в прошлом). Данную процедура рассмотрена подробнее в пункте «Возобновление сессии».  
  
Также имеется дополнительное расширение процедуры Handshake, которое имеет название TLS False Start. Это расширение позволяет клиенту и серверу начать обмен зашифрованными данными сразу после установления метода шифрования, что сокращает установление соединения на одну итерацию сообщений. Об этом подробнее рассказано в пункте “TLS False Start”.  
 

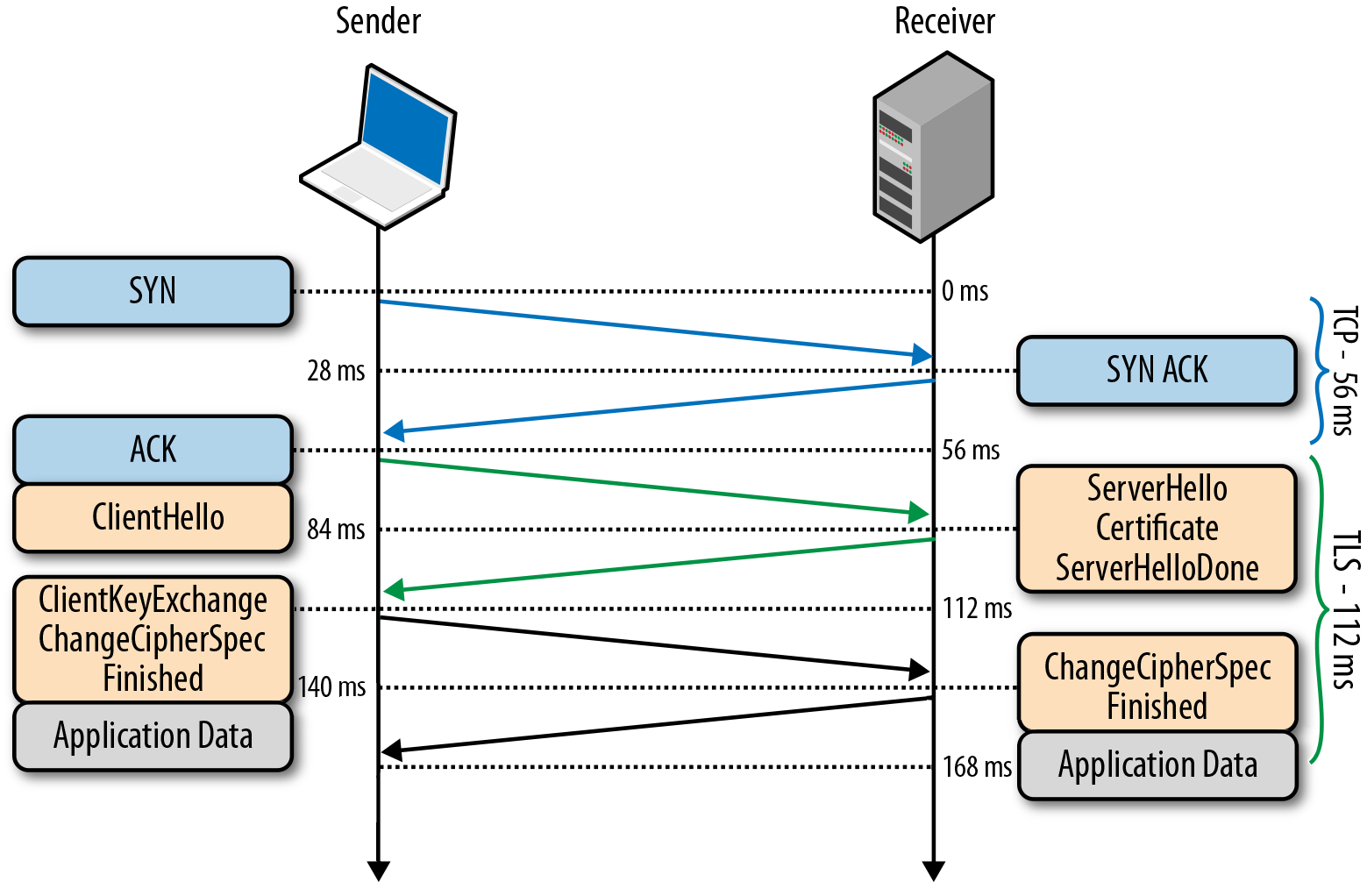
##### Обмен ключами в протоколе TLS

По различным историческим и коммерческим причинам чаще всего в TLS используется обмен ключами по алгоритму RSA: клиент генерирует симметричный ключ, подписывает его с помощью открытого ключа сервера и отправляет его на сервер. В свою очередь, на сервере ключ клиента расшифровывается с помощью закрытого ключа. После этого обмен ключами объявляется завершённым. Данный алгоритм имеет один недостаток: эта же пара отрытого и закрытого ключей используется и для аутентификации сервера. Соответственно, если злоумышленник получает доступ к закрытому ключу сервера, он может расшифровать весь сеанс связи. Более того, злоумышленник может попросту записать весь сеанс связи в зашифрованном варианте и занять расшифровкой потом, когда удастся получить закрытый ключ сервера. В то же время, обмен ключами Диффи-Хеллмана представляется более защищённым, 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. На этом основана служба снижения риска компрометации прошлых сеансов связи: для каждого нового сеанса связи создаётся новый, так называемый «временный» симметричный ключ. Соответственно, даже в худшем случае (если злоумышленнику известен закрытый ключ сервера), злоумышленник может лишь получить ключи от будущих сессий, но не расшифровать ранее записанные.  
  
На текущий момент, все браузеры при установке соединения TLS отдают предпочтение именно сочетанию алгоритма Диффи-Хеллмана и использованию временных ключей для повышения безопасности соединения.  
  
Следует ещё раз отметить, что шифрование с открытым ключом используется только в процедуре TLS Handshake во время первоначальной настройки соединения. После настройки туннеля в дело вступает симметричная криптография, и общение в пределах текущей сессии зашифровано именно установленными симметричными ключами. Это необходимо для увеличения быстродействия, так как криптография с открытым ключом требует значительно больше вычислительной мощности.  
 

##### Возобновление сессии TLS

Как уже отмечалось ранее, полная процедура TLS Handshake является довольно длительной и дорогой с точки зрения вычислительных затрат. Поэтому была разработана процедура, которая позволяет возобновить ранее прерванное соединение на основе уже сконфигурированных данных.  
  
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. Естественно, в таком случае у сервера хранится кэш сгенерированных идентификаторов и параметров сеанса для каждого клиента. 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 упрощённому алгоритму, показанному на рисунке. Если нет, то требуется полная версия TLS Handshake.  
  
  
  
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 нескомпрометированные данные предыдущей сессии.  
  
Однако здесь имеется практическое ограничение: так как сервер должен хранить данные обо всех открытых сессиях, это приводит к проблеме с популярными ресурсами, которые одновременно запрашиваются тысячами и миллионами клиентов.  
  
Для обхода данной проблемы был разработан механизм «Session Ticket», который устраняет необходимость сохранять данные каждого клиента на сервере. Если клиент при первоначальной установке соединения указал, что он поддерживает эту технологию, то в сервер в ходе TLS Handshake отправляет клиенту так называемый Session Ticket – параметры сессии, зашифрованные закрытым ключом сервера. При следующем возобновлении сессии, клиент вместе с ClientHello отправляет имеющийся у него Session Ticket. Таким образом, сервер избавлен от необходимости хранить данные о каждом соединении, но соединение по-прежнему безопасно, так как Session Ticket зашифрован ключом, известным только на сервере.  
 

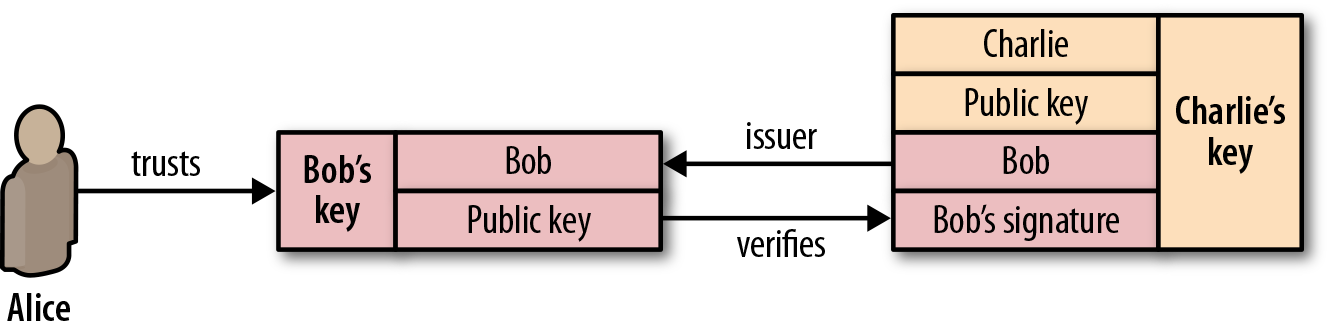
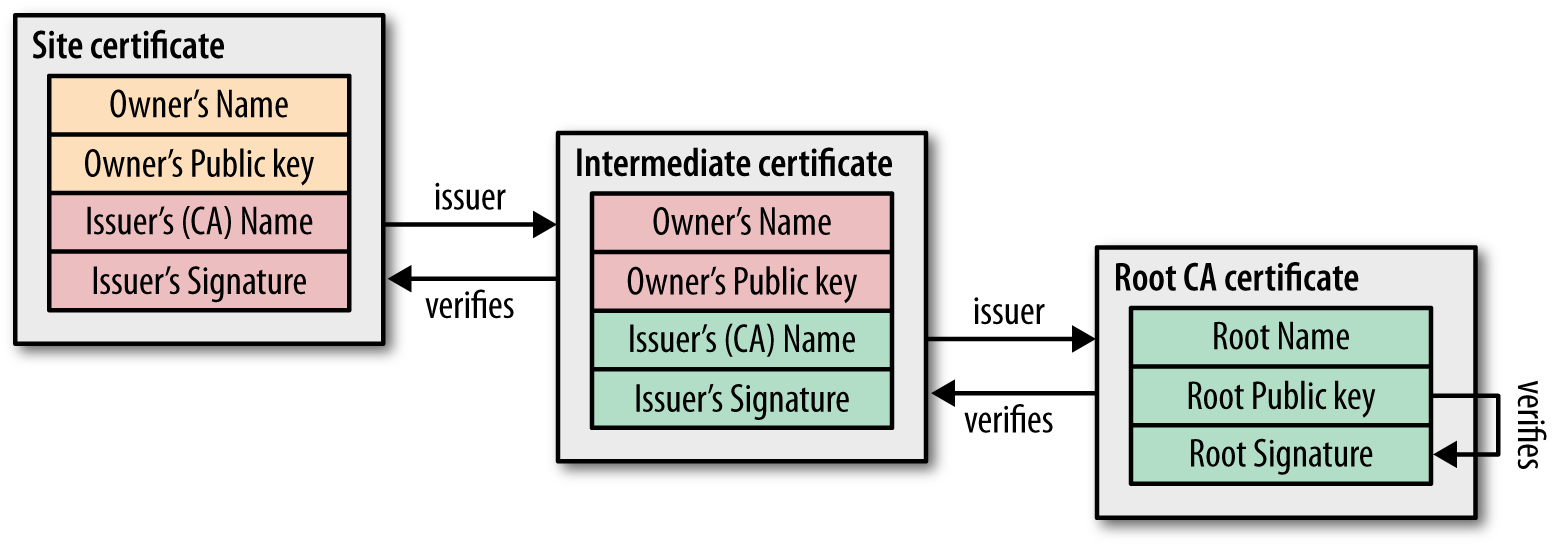
##### TLS False Start

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. Подробная схема TLS False Start представлена на рисунке:  
  
  
  
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.  
 

##### TLS Chain of trust

Аутентификация является неотъемлемой частью каждого TLS соединения. Рассмотрим простейший процесс аутентификации между Алисой и Бобом:

1. И Алиса, и Боб генерируют собственные открытые и закрытые ключи.
2. Алиса и Боб обмениваются открытыми ключами.
3. Алиса генерирует сообщение, шифрует его своим закрытым ключом и отправляет Бобу.
4. Боб использует полученный от Алисы ключ, чтобы расшифровать сообщение и таким образом проверяет подлинность полученного сообщения.

Очевидно, что данная схема построена на доверии между Алисой и Бобом. Предполагается, что обмен открытыми ключами произошёл, например, при личной встрече, и, таким образом, Алиса уверена, что получила ключ непосредственно от Боба, а Боб, в свою очередь, уверен, что получил открытый ключ Алисы.  
  
Пусть теперь Алиса получает сообщение от Чарли, с которым она не знакома, но который утверждает, что дружит с Бобом. Чтобы это доказать, Чарли заранее попросил подписать собственный открытый ключ закрытым ключом Боба, и прикрепляет эту подпись к сообщению Алисе. Алиса же сначала проверяет подпись Боба на ключе Чарли (это она в состоянии сделать, ведь открытый ключ Боба ей уже известен), убеждается, что Чарли действительно друг Боба, принимает его сообщение и выполняет уже известную проверку целостности, убеждаясь, что сообщение действительно от Чарли:  
  
  
  
Описанное в предыдущем абзаце и есть создание «цепочки доверия» (или «Chain of trust», если по-английски).  
В протоколе TLS данные цепи доверия основаны на сертификатах подлинности, предоставляемых специальными органами, называемыми центрами сертификации (CA – certificate authorities). Центры сертификации производят проверки и, если выданный сертификат скомпрометирован, то данный сертификат отзывается.  
  
Из выданных сертификатов складывается уже рассмотренная цепочка доверия. Корнем её является так называемый “Root CA certificate” – сертификат, подписанный крупным центром, доверие к которому неоспоримо. В общем виде цепочка доверия выглядит примерно таким образом:  
  
  
  
Естественно, возникают случаи, когда уже выданный сертификат необходимо отозвать или аннулировать (например, был скомпрометирован закрытый ключ сертификата, или была скомпрометирована вся процедура сертификации). Для этого сертификаты подлинности содержат специальные инструкции о проверке их актуальности. Следовательно, при построении цепочки доверия, необходимо проверять актуальность каждого доверительного узла.  
  
Механизм этой проверки прост и в его основе лежит т.н. «Список отозванных сертификатов» (CRL – «Certificate Revocation List»). У каждого из центров сертификации имеется данный список, представляющий простой перечень серийных номеров отозванных сертификатов. Соответственно любой, кто хочет проверить подлинность сертификата, попросту загружает данный список и ищет в нём номер проверяемого сертификата. Если номер обнаружится – это значит, что сертификат отозван.  
  
Здесь очевидно присутствует некоторая техническая нерациональность: для проверки лишь одного сертификата требуется запрашивать весь список отозванных сертификатов, что влечёт замедление работы. Для борьбы с этим был разработан механизм под названием «Протокол статусов сертификатов» (OCSP – Online Certificate Status Protocol). Он позволяет осуществлять проверку статуса сертификата динамически. Естественно, это снижает нагрузку на пропускную способность сети, но в то же время порождает несколько проблем:

* Центры сертификации должны справляться с нагрузкой в режиме реального времени;
* Центры сертификации должны гарантировать свою доступность в любое время;
* Из-за запросов реального времени центры сертификации получают информацию о том, какие сайты посещал каждый конкретный пользователь.

Собственно, во всех современных браузерах оба решения (OCSP и CRL) дополняют друг друга, более того, как правило имеется возможность настройки предпочитаемой политики проверки статуса сертификата.