Протокол, который защищает данные во время передачи их по сети. Он работает на четвертом, транспортном уровне сетевой модели OSI. Отвечает за создание безопасных сессий обмена данными между браузером и сервером.

Как TLS шифрует данные

Чтобы защитить данные, TLS создает во время передачи специальный канал, где их нельзя прочитать или изменить без секретного ключа. Ключ - подсказка, как именно читать сообщение.
В зависимости от количества ключей в TLS используется один из двух классов шифрования:

- **Симметричное шифрование** - когда используется один и тот же ключ для шифрования и дешифрования данных. Оно работает эффективно и быстро, но требует предварительного обмена ключом между компьютером и сервером, в ходе которого ключ могут перехватить.
- **Ассиметричное шифрование** - использует два ключа: публичный для шифрования и приватный для дешифровки. Публичный ключ можно свободно распространить, а приватный должен быть хорошо защищён. Ассиметричное шифрование безопаснее, но требует больше вычислительных ресурсов и работает медленнее, чем симметричное.
  В протоколе TLS симметричное шифрование используют для шифрования непосредственно сообщений, а асимметричное шифрование - во время рукопожатия, то есть в начале сессии для обмена ключами и аутентификации.
  TLS обеспечивает проверку подлинности сервера при помощи специальных сертификатов, которые заверяет специальный центр сертификации.

Основные алгоритмы шифрования в TLS

- **AES (Advanced Encryption Standard)** - Симметричной алгоритм шифрования, использующий для защиты данных ключи длиной от 128 до 256 бит. Чем длиннее ключ, тем выше защита, но ниже производительность.
- **3DES (Triple Data Encryption Standard)** - Чуть менее безопасный алгоритм, в сравнении cAES. Он делает абсолютно тоже самое, но тремя разными ключами.

- **ChaCha20-Poly1305** - Симметричный алгоритм, который использует ChaCha20 для генерации ключей и Poly1305 для создания MAC - кода аутентификации сообщений. Сервер вычисляет MAC и сравнивает его с полученным значением от клиента. Если совпало - значит, данные никто не менял

Как работает TLS

- **Приветствие** - клиент и сервер обмениваются сообщениями, в которых они представляются друг другу и договариваются, какие алгоритмы шифрования будут использовать при передаче данных
- **Обмен ключами** - клиент и сервер убеждаются в подлинности друг друга и выполняют обмен ключами для дальнейшего использования при шифровании данных
- **Завершение рукопожатия** - клиент и сервер подтверждают, что установка соединения прошла успешно и что они готовы к защищенной передаче данных
- **Фрагментация** - данные разбиваются на фрагменты меньшего размера для последующего шифрования
- **Компрессия** - данные сжимаются, их объём уменьшается (если применяется сжатие)
- **Шифрование** - фрагменты данных шифруются по выбранному алгоритму симметричного шифрования, использующему общий секретный ключ, установленный на этапе рукопожатия
- **Аутентификация** - каждый фрагмент данных подписывается цифровой подписью или кодом аутентификации, чтобы соблюсти целостность и подлинность данных
- **Передача данных** - зашифрованные и аутентифицированные фрагменты данных передаются по сети между клиентом и сервером

Механизм возобновления сессий

Чтобы ускорить обмен сообщениями, разработчики добавили в TLS механизм возобновления сессий: **0-RTT (Zero Round Trip Time Resumption, нулевое время приема-передачи)**
Механизм работает следующим образом: во время обмена данными и клиент, и сервер кэшируют данные о сессии в специальные токены. Если эти токены совпадут, стороны могут пропустить этап рукопожатия и сразу перейти к обмену данными.

Недостатки механизма

- **Недостаточная безопасность** - клиент отправляет зашифрованные данные на сервер, используя предварительно согласованные симметричные ключи. Однако, если эти ключи перехватил злоумышленник, то он может вклиниться в сессию
- **"Перегрузка" токенами** - если сервер обслуживает сайт с миллионами пользователей, хранить в памяти токены каждой сессии может быть проблематично. К счастью, на этот случай придумали механизм `session ticket`, который позволяет серверу "не держать все в себе"
- Ограниченные динамические параметры - своего рода условные знаки, которые стороны используют во время сессии, чтобы подтвердить, что они - это все еще они. В механизме 0-RTT эти параметры используются реже, чем в обычном режиме.
- Задержки в ответах - в рамках сессии 0-RTT клиент отправляют порцию данных на сервер до того, как получит ответ о "доставке" предыдущей. Это может привести к рассинхрону между сервером и клиентом - клиент может отправить несколько запросов до того, как получит ответ на первый.