|  |  |
| --- | --- |
| https://lh7-us.googleusercontent.com/QuBaagabzZYLr7U3ZbYOtMusd5FbWWxIvMUTN8jrLHzSLMg534z9gXRTIG1Us4i_lOwmWlaBxKedNt-SQ26dm4WmyqwjGDmEO6z8GE3QrZosqvHM88J2EFeVf1u0GzyCZQlhWmp1Zeo85tKo4LJVXQ | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» |

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Отчёт по лабораторной работе №2  
по учебной дисциплине «Web-технологии»

на тему «Модель OSI. Стек протоколов. Wireshark.»

Выполнил  
студент группы М3О-119БВ-24

Нарзиев А.Т.

Принял

Ассистент каф.304 Борисов А.И.

Москва  
2024

# Содержание

# Задание

1. Рассмотреть классическую многоуровневую модель OSI. Определить, какие протоколы возможны на каждом уровне (привести по 2 примера на каждом уровне).
2. Запустить программу Wireshark. Настроить программу на активное интернет-соединение. Рассмотреть различные TCP и UDP соединения.

* Привести в отчёте пример трехстороннего рукопожатия и трехстороннего завершения сеанса TCP (показать не только сам процесс, но и раскрыть флаги для каждого пакета).
* Привести пример UDP-пакета.

1. Рассмотреть процесс установления TLS соединения.

* В отчете привести заголовки стека протоколов основных сообщений при создании TSL соединения.
* Раскрыть все параметры TLS соединения, отобразить все Chiper Suites, Extension, Certificate,
* Показать данные протокола DH.

1. С помощью браузера отправить запрос для получения HTML-страницы по протоколу HTTP.

* Найти соответствующий запрос и ответ в программе Wireshark.
* В отчете полностью привести как HTTP запрос, так и HTTP ответ. Выделить содержимое тела запроса и тела ответа.

1. Проанализировать содержимое данных, отправленных в запросе и полученных в ответе. Расписать в отчете основные моменты соединения для каждого из протоколов.
2. Рассмотреть загрузку других ресурсов сайта (CSS/ PNG/ JPG и т.д.) как запрос на данный ресурс, так и ответ (привести только HTTP протокол).

# Решение

## Многоуровневая модель OSI

Классическая многоуровневая модель OSI состоит из 7 уровней. Рассмотрим каждый из них:

1. **Физический**

Отвечает за передачу необработанных битов по физическим каналам.

Примеры протколов:

* **Кабели HDMI:** Используются для передачи аудио и видео сигналов между устройствами.
* **USB (Universal Serial Bus):** Стандарт для подключения периферийных устройств к компьютерам.

1. **Канальный**

Отвечает за надежную передачу данных между узлами в одной сети

Примеры протоколов:

* **PPP (Point-to-Point Protocol):** Протокол для передачи данных по последовательным соединениям.
* **Ethernet:** Широко используемая технология для передачи данных в локальных сетях, использующая MAC-адреса для идентификации устройств.

1. **Сетевой**

Отвечает за маршрутизацию данных между различными сетями.

Примеры протоколов:

* **IP (Internet Protocol):** Основной протокол для передачи данных в интернете.
* **ICMP (Internet Control Message Protocol):** Протокол для обмена управляющими сообщениями и диагностикой сети (например, команды ping и traceroute используют ICMP для проверки доступности узлов).

1. **Транспортный**

Обеспечивает надежную передачу данных между конечными узлами.

Примеры протоколов:

* **TCP (Transmission Control Protocol)**: Протокол с управлением соединением, обеспечивающий надежную и упорядоченную передачу данных.
* **UDP (User Datagram Protocol):** Протокол без управления соединением, используемый для быстрой передачи данных, например, в потоковом видео.

1. **Сессионный**

Сессионный уровень управляет сессиями связи между приложениями, устанавливает, поддерживает и завершает соединения.

Примеры протоколов:

* **RPC (Remote Procedure Call):** Позволяет программам вызывать функции на удаленных компьютерах.
* **NetBIOS:** Протокол, который предоставляет интерфейс для сетевых приложений для управления сессиями.

1. **Уровень представления**

Уровень представления отвечает за преобразование данных, чтобы они могли быть восприняты приложениями.

Примеры протколов:

* **ASCII и EBCDIC:** Разные кодировки символов, которые могут быть преобразованы для совместимости.
* **SSL/TLS (Secure Sockets Layer/Transport Layer Security):** Протоколы для шифрования данных, передаваемых по сети.

1. **Прикладной**

Отвечает за взаимодействие пользователя с приложениями. Он обеспечивает интерфейс для программ и пользователей для доступа к сетевым службам.

Примеры протколов:

* **HTTP (Hypertext Transfer Protocol):** Протокол для передачи веб-страниц.
* **FTP (File Transfer Protocol):** Протокол для передачи файлов между компьютерами.

## WireShark

* 1. **Трёхстороннее рукопожатие**

**1. SYN (Synchronize)**

- Содержимое пакета:

**- Флаг SYN:** Указывает, что это запрос на установление соединения.

**- Номер последовательности (Sequence Number):** Это начальный номер, который клиент выбирает случайным образом. Он используется для отслеживания порядка отправленных байтов данных.

**Пример:** Клиент отправляет пакет с SYN и номером последовательности

client\_seq1 = 1000.

**2. SYN-ACK (Synchronize-Acknowledgment)**

- Содержимое пакета:

**- Флаги SYN и ACK:** Указывает, что сервер подтверждает получение SYN от клиента и сам хочет установить соединение.

**- Номер последовательности (Sequence Number):** Сервер выбирает свой собственный начальный номер последовательности, например, server\_seq = 2000.

**- Номер подтверждения (Acknowledgment Number):** Это номер последовательности клиента, увеличенный на 1 (в данном случае server\_ack = 1001), который указывает, что сервер получил первый пакет от клиента.

**Пример:** Сервер отправляет пакет с SYN-ACK, server\_seq = 2000 и server\_ack = 1001.

**3. ACK (Acknowledgment)**

- Содержимое пакета:

**- Флаг ACK:** Указывает, что это подтверждение.

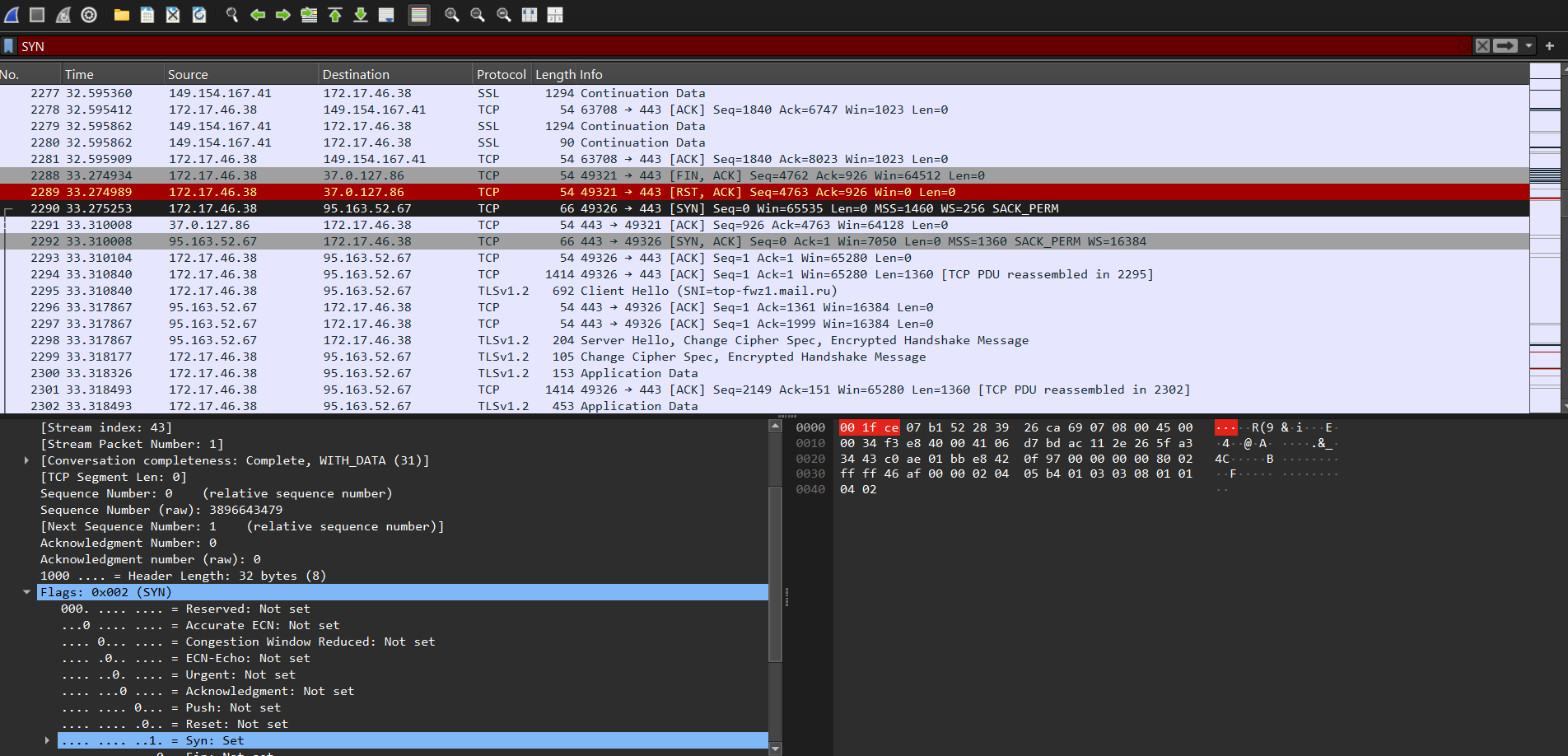
**- Номер последовательности (Sequence Number):** Клиент отправляет свой номер последовательности, увеличенный на 1 (в данном случае client\_seq2 = 1001).

**- Номер подтверждения (Acknowledgment Number):** Это номер последовательности сервера, увеличенный на 1 (в данном случае client\_ack = 2001), подтверждая, что клиент получил SYN-ACK от сервера.

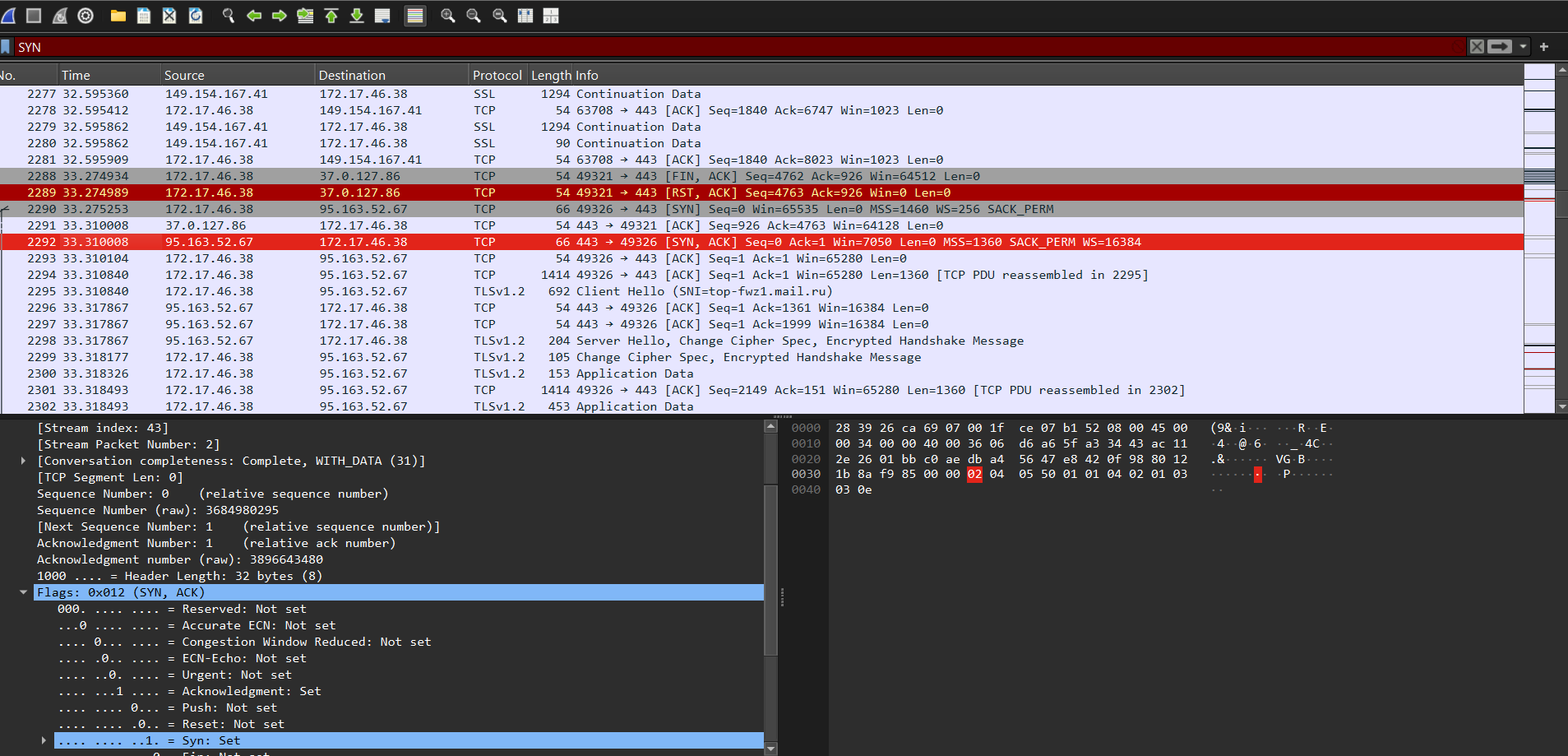
**Пример:** Клиент отправляет пакет ACK, client\_seq2 = 1001 и client\_ack = 2001.

**Пример**

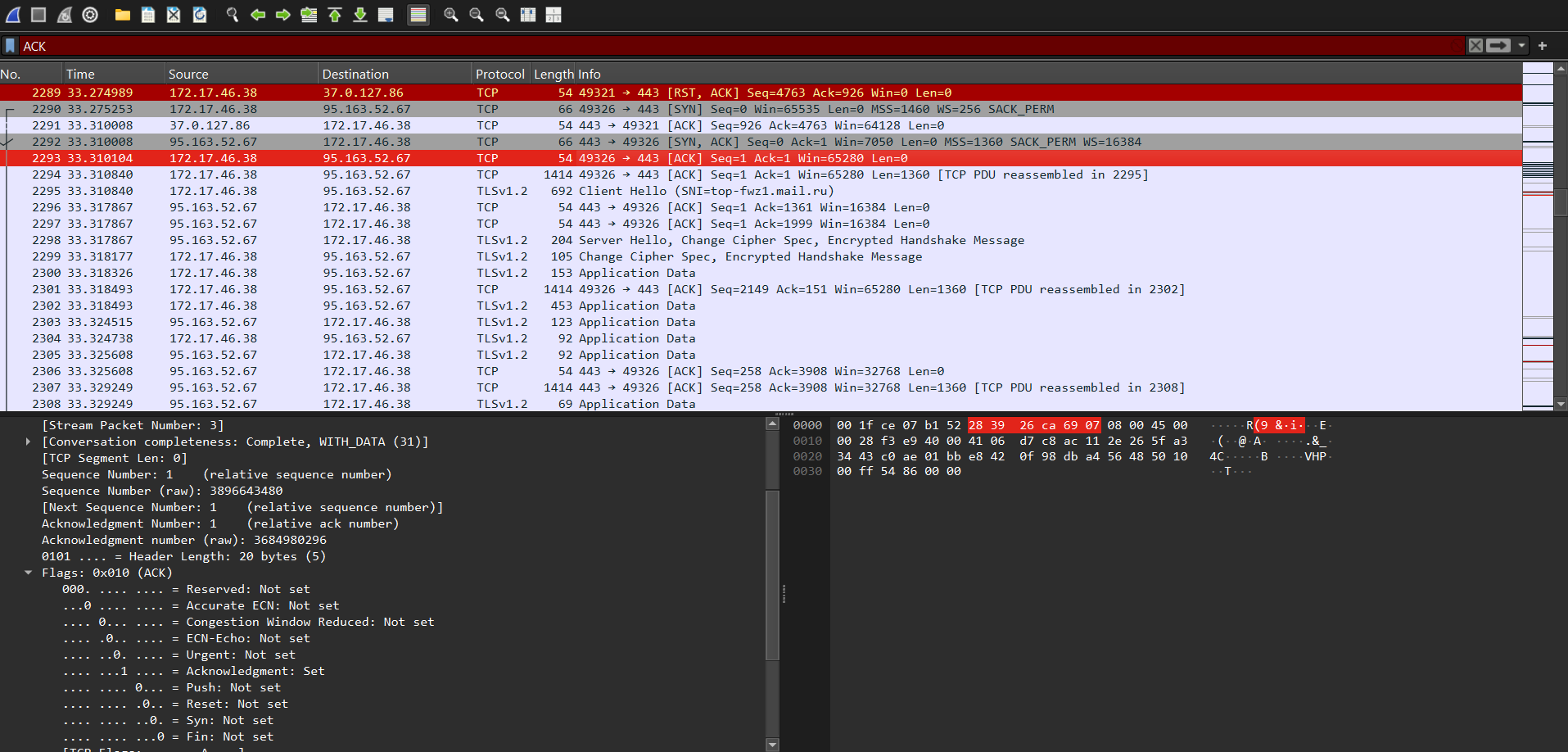
SYN:

****

SYN-ACK:



ACK:



* 1. **Трёхстороннее завершение**

**1. Клиент отправляет FIN-ACK:**

Когда клиент хочет завершить соединение, он отправляет пакет с флагом FIN-ACK. Это сигнализирует серверу о том, что клиент больше не будет отправлять данные.

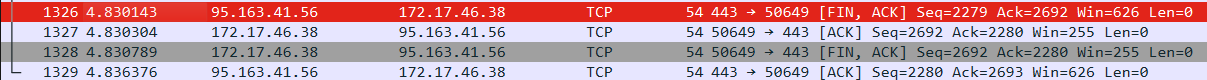
**2. Сервер отвечает FIN-ACK:**

Сервер получает пакет FIN и отвечает пакетом с флагами FIN и ACK и пакетом с флагом ACK, подтверждая готовность завершить соединение.

**3. Клиент отвечает с ACK:**

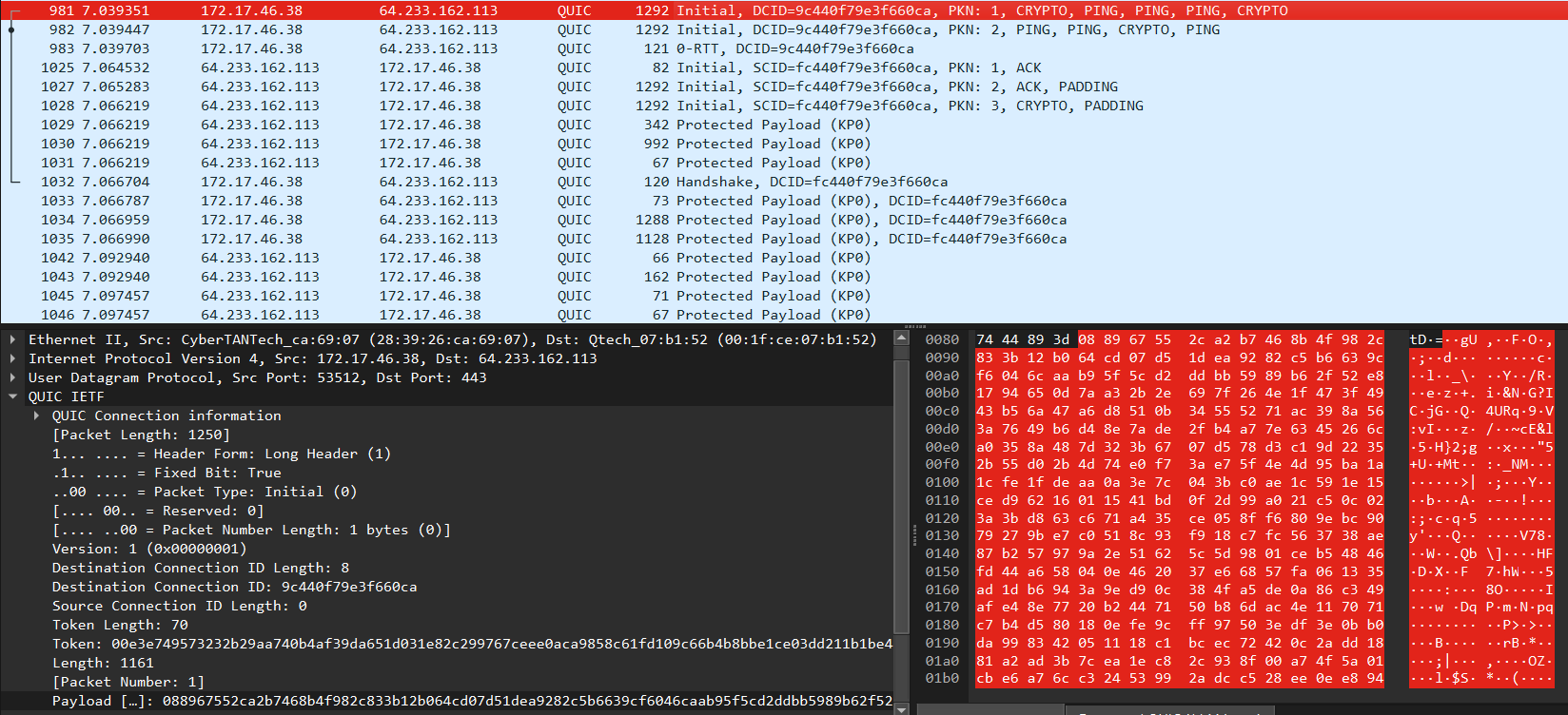
Клиент получает FIN от сервера и отвечает пакетом с флагом ACK, подтверждая завершение соединения.

**Пример:**

****

**Анализ этапов завершения TCP-соединения**

1. 54 50649 - [FIN, ACK] Seq=2279 Ack=2692 Win=626 Len=0
   * Описание: Клиент (порт 50649) отправляет серверу (порт 443) пакет с флагами FIN и ACK. Это означает, что клиент завершает отправку данных и подтверждает получение данных от сервера.
   * Seq=2279: Номер последовательности клиента.
   * Ack=2692: Номер подтверждения, указывающий, что клиент получил данные до номера 2691.
2. 443 [ACK] Seq=2692 Ack=2280 Win=255 Len=0
   * Описание: Сервер (порт 443) отвечает клиенту, отправляя пакет с флагом ACK. Это подтверждение получения FIN от клиента.
   * Seq=2692: Номер последовательности сервера.
   * Ack=2280: Номер подтверждения от сервера, указывающий, что он получил данные до номера 2279.
3. 54 50649 - 443 [FIN, ACK] Seq=2692 Ack=2280 Win=255 Len=0
   * Описание: Сервер отправляет клиенту пакет с флагами FIN и ACK, уведомляя клиента о том, что он тоже завершает отправку данных.
   * Seq=2692: Номер последовательности сервера.
   * Ack=2280: Номер подтверждения, указывающий, что сервер подтверждает получение данных до номера 2279 от клиента.
4. 54 443 - [ACK] Seq=2280 Ack=2693 Win=626 Len=0
   * Описание: Клиент отправляет последний пакет с флагом ACK, подтверждая получение FIN от сервера. Это завершает процесс закрытия соединения.
   * Seq=2280: Номер последовательности клиента.
   * Ack=2693: Номер подтверждения, указывающий, что клиент подтвердил получение данных до номера 2692 от сервера.
   1. **Пример UDP пакета**

****

QUIC (Quick UDP Internet Connections): Это современный транспортный протокол, который разработан Google для улучшения производительности передачи данных в интернете. QUIC работает поверх UDP и используется для повышения скорости загрузки веб-страниц и уменьшения задержек. Так как QUIC использует UDP, мы можем его тоже видеть в списке соединений при фильтре UDP

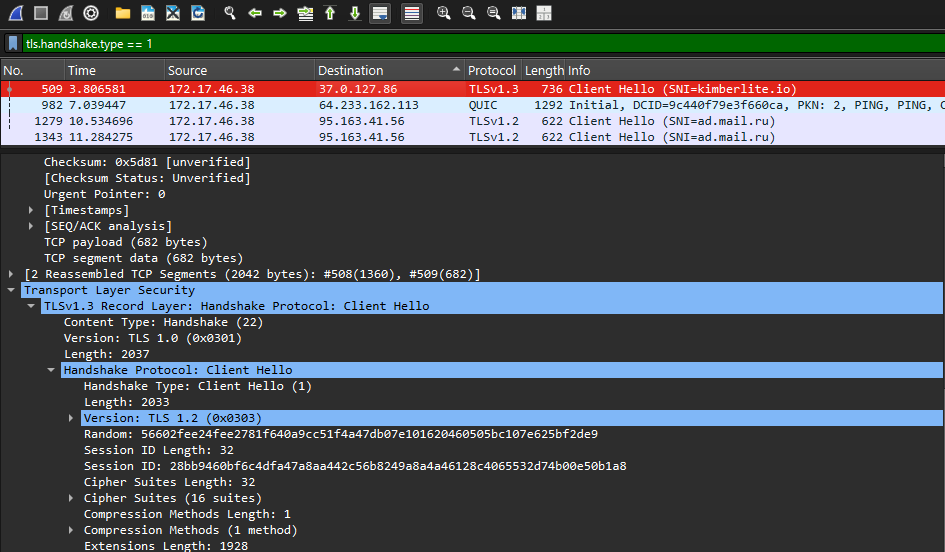
* **1292 Initial**: Указывает на размер пакета (в байтах) и что это начальный пакет.
* **DCID**: Destination Connection ID — это идентификатор соединения, который используется для идентификации конкретного соединения.
* **PKN**: Packet Number — номер пакета, который увеличивается при каждом отправленном пакете.
* **CRYPTO, PING**: Эти термины обозначают типы сообщений в пакете. Несколько запросов PING используются для проверки доступности соединения во время продолжительного взаимодействия, а CRYPTO — для передачи криптографических данных.

## TLS соединения

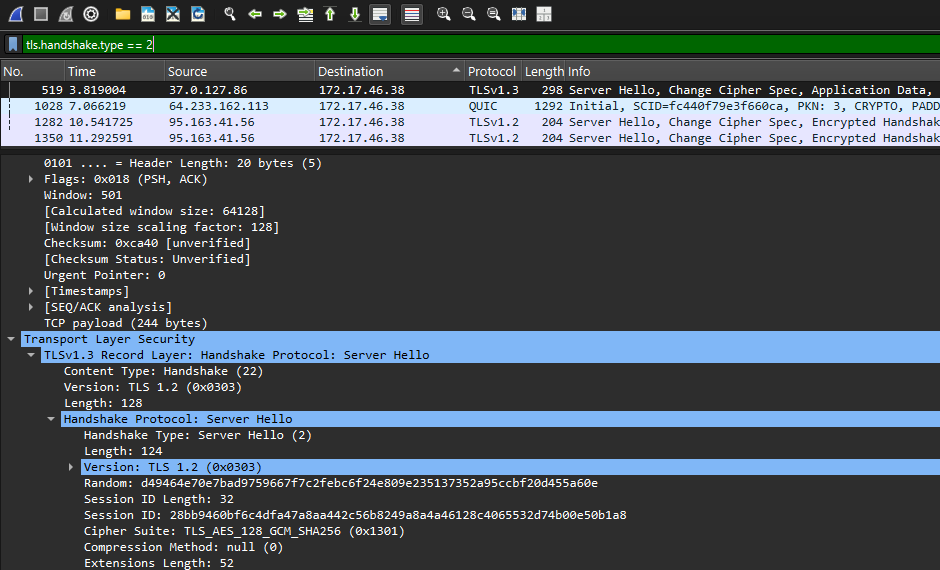
* 1. Стек протоколов

Основные шаги:

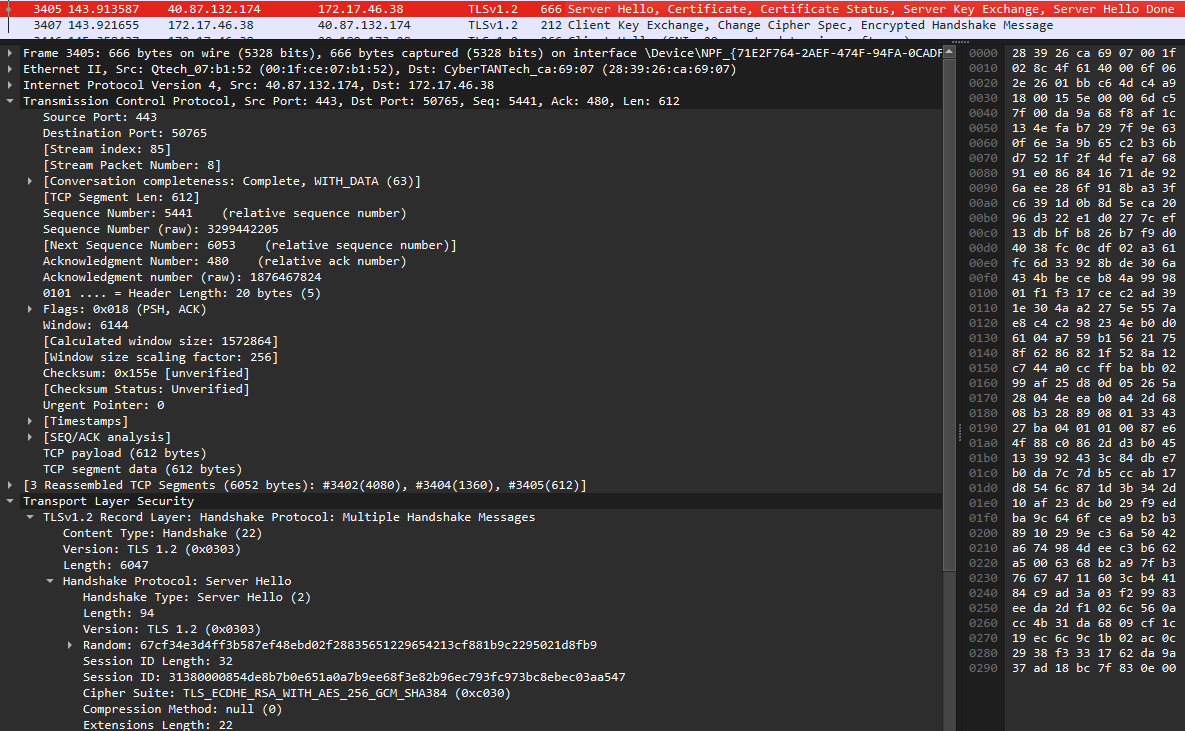
**1. ClientHello:** Клиент отправляет сообщение ClientHello, в котором указывает поддерживаемые версии протокола TLS, шифровые наборы, расширения и случайное число (random).



**2. ServerHello:** Сервер отвечает сообщением ServerHello, выбирая версию протокола, шифровый набор и отправляя свое случайное число.



**3. Server Certificate:** Сервер отправляет свой сертификат, который подтверждает его личность.



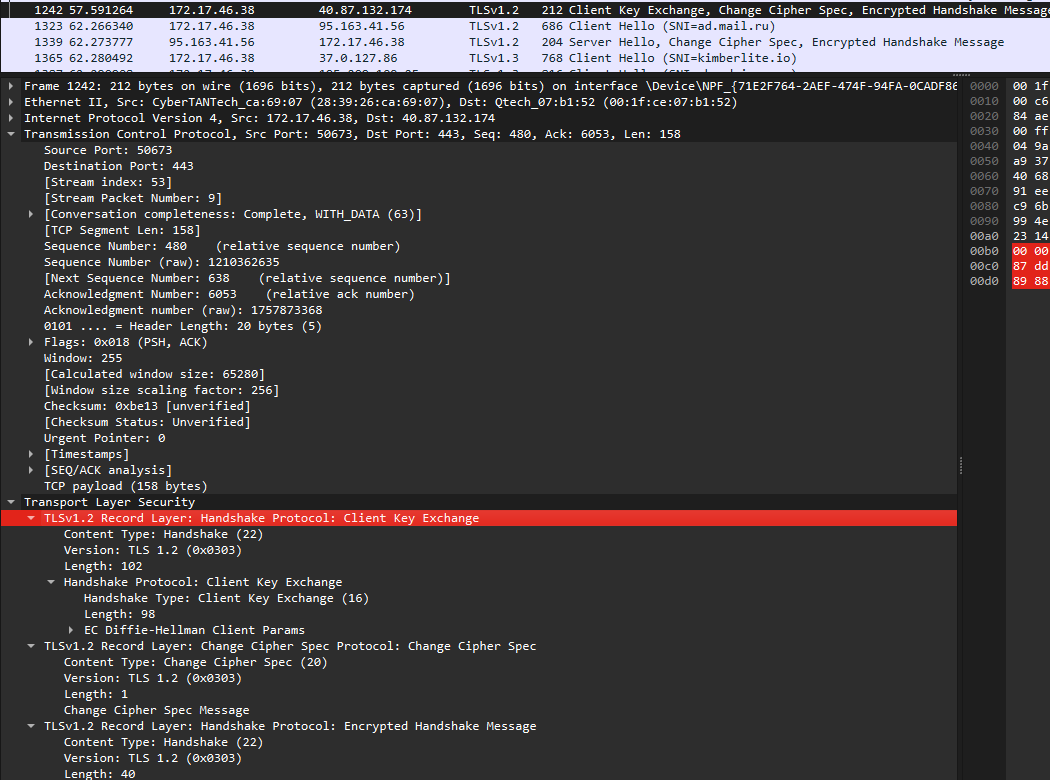
**4. Server Key Exchange:** Если выбранный шифровый набор требует дополнительной информации для обмена ключами, сервер отправляет это сообщение.

**5. Certificate Request:** Сервер может запросить сертификат клиента для аутентификации.

**6. ServerHelloDone:** Сервер отправляет сообщение ServerHelloDone, указывая на завершение своей части обмена.

**7. Client Certificate:** Клиент отправляет свой сертификат, если сервер запросил его.

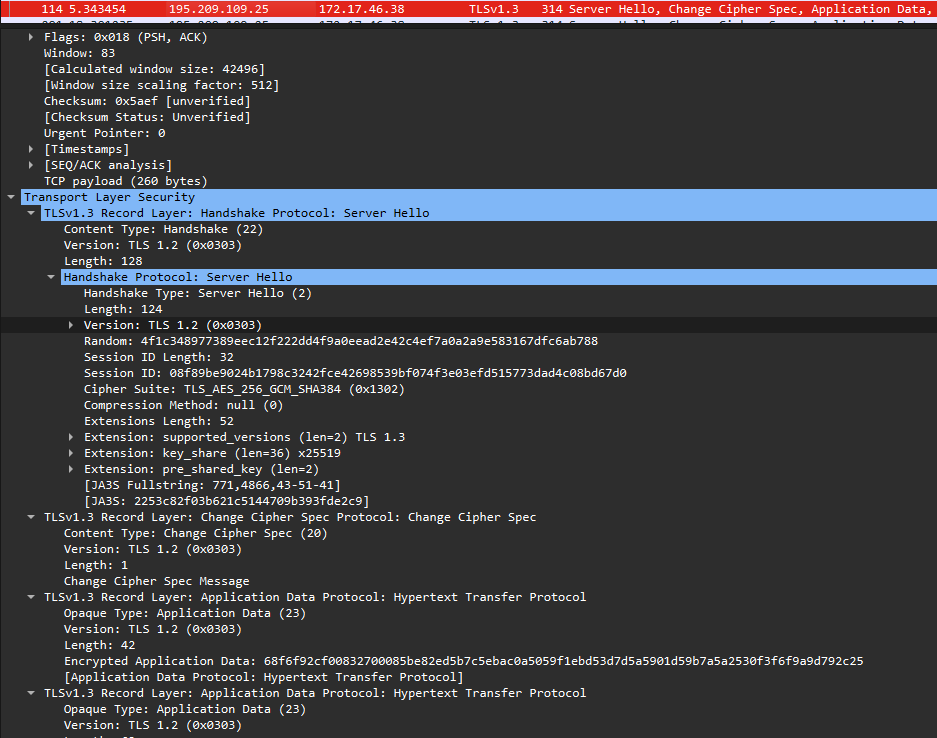
**8. Client Key Exchange:** Клиент отправляет сообщение Client Key Exchange, содержащее информацию для генерации общего секрета (например, параметр DH).



**9. ChangeCipherSpec (клиент):** Клиент отправляет сообщение ChangeCipherSpec, указывая, что последующие сообщения будут зашифрованы с использованием выбранного шифрового набора.

**10. Finished (клиент):** Клиент отправляет сообщение Finished, которое подтверждает, что клиент завершил свою часть установки соединения.

**11. ChangeCipherSpec (сервер**): Сервер отправляет свое сообщение ChangeCipherSpec.

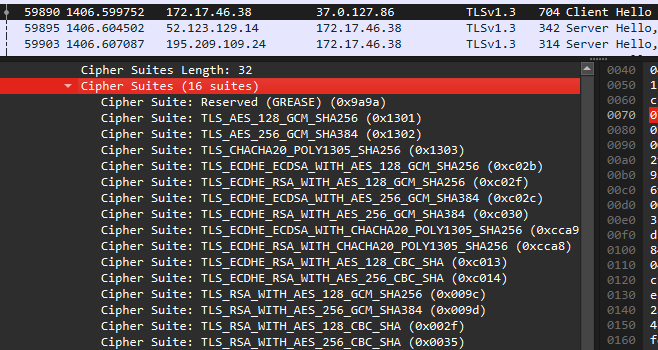


**12. Finished (сервер):** Сервер отправляет сообщение Finished, подтверждая, что он завершил установку соединения.

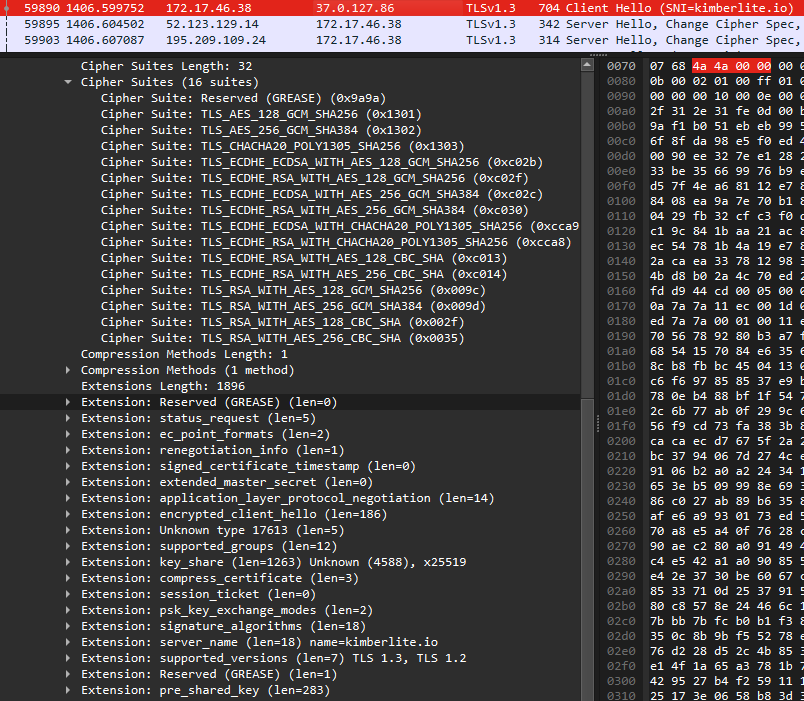
**13. (иногда)** **HelloRetryRequest** — это сообщение, которое используется в процессе установки TLS 1.3 для обработки ситуаций, когда сервер не может завершить процесс **ClientHello** из-за недостатка информации или необходимости повторного запроса от клиента. Вот в каких случаях оно отправляется:

**Когда отправляется HelloRetryRequest:**

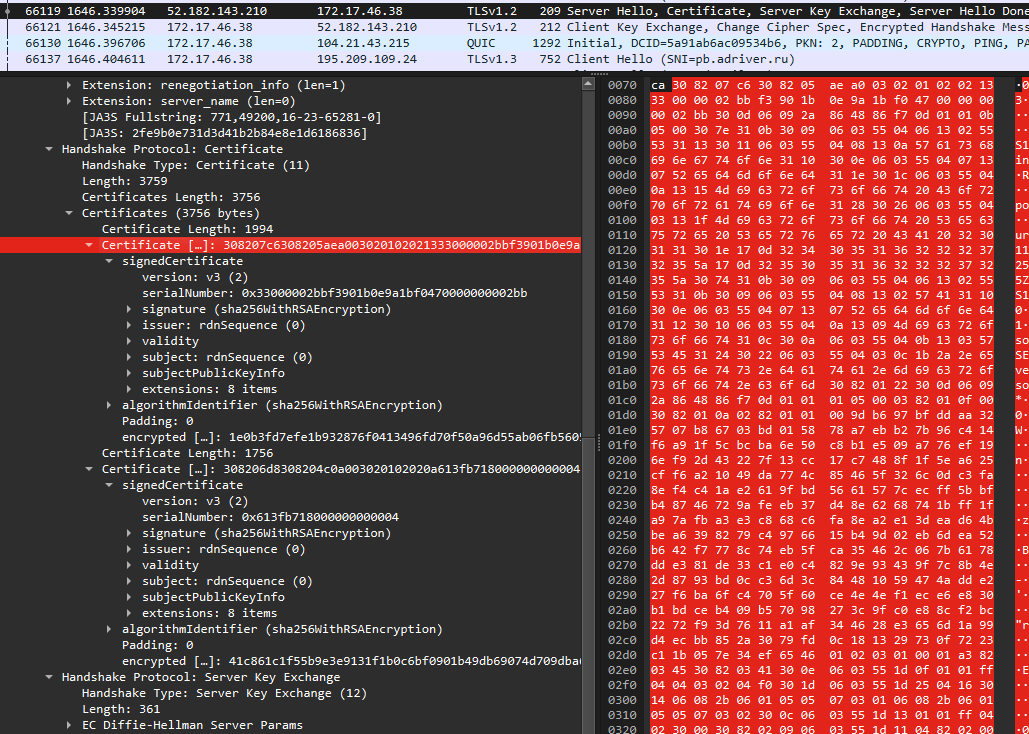
* 1. **Необходимость дополнительной информации**: Если сервер получает **ClientHello** и обнаруживает, что для успешного завершения рукопожатия ему нужна еще информация (например, поддержка определённого шифрового набора или параметров), он может отправить **HelloRetryRequest**.
  2. **Проблемы с параметрами**: Если клиент отправляет параметры, которые сервер не поддерживает или которые не соответствуют его ожиданиям, сервер может запросить повторный **ClientHello** с изменёнными параметрами.
  3. **Изменение параметров шифрования**: В некоторых случаях сервер может захотеть, чтобы клиент использовал другой шифровый набор или параметры, которые были не указаны в первоначальном **ClientHello**.
  4. Параметры TLS-соединений
* Cipher Suites



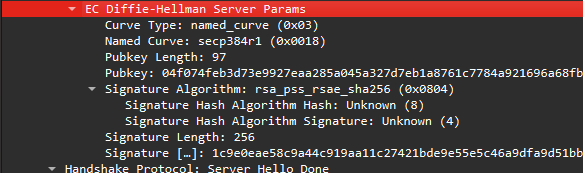
* Extensions



* Certificates



* 1. Протокол Диффи-Хеллмана

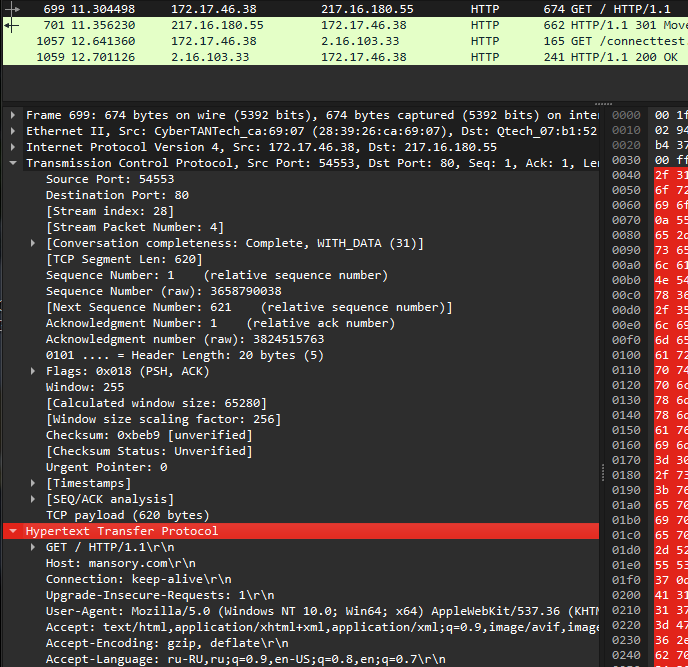


**Параметры EC Diffie-Hellman:**

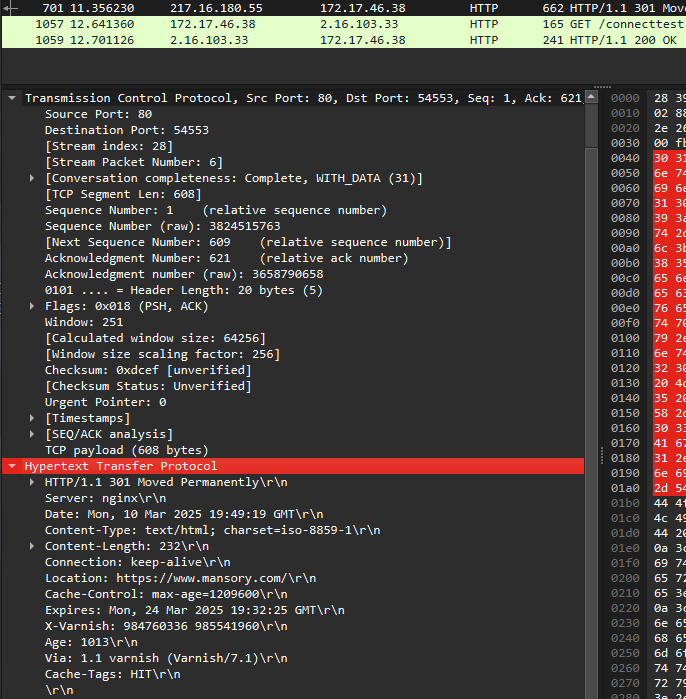
1. **Curve Type: named\_curve (0x03)**:
   * Указывает, что используется именованная кривая для обмена ключами.
2. **Named Curve: secp384r1 (0x0018)**:
   * Это конкретная кривая, используемая для EC Diffie-Hellman. secp384r1 — это кривая с 384-битным ключом, которая обеспечивает высокий уровень безопасности.
3. **Pubkey Length: 97**:
   * Длина публичного ключа в байтах. В данном случае 97 байт — это стандартная длина для публичного ключа на кривой secp384r1.
4. **Pubkey**:
   * Это сам публичный ключ, представленный в шестнадцатеричном формате. Он начинается с 04, что указывает на то, что это не сжатый ключ.
5. **Signature Algorithm: rsa\_pss\_rsae\_sha256 (0x0804)**:
   * Указывает алгоритм подписи, используемый для аутентификации. В данном случае используется RSA-PSS (Probabilistic Signature Scheme) с SHA-256 в качестве хэш-функции.
6. **Signature Hash Algorithm Hash: Unknown (8)** и **Signature Hash Algorithm Signature: Unknown (4)**:
   * Эти поля обозначают алгоритмы хэширования для подписи, но в данном случае они не определены.
7. **Signature Length: 256**:
   * Длина подписи в байтах. Здесь она составляет 256 байт, что соответствует размеру подписи для RSA с 2048-битным ключом.
8. **Handshake Protocol: Server Hello Done**:
   * Указывает, что сервер завершил свою часть рукопожатия и отправил все необходимые параметры клиенту.
9. **Signature**:
   * Это сама подпись, которая подтверждает подлинность сообщения. Она используется для проверки, что сообщение действительно было отправлено сервером и не было изменено.

## Протокол HTTP

* 1. HTTP-запрос



1. **Host: mansory.com/pln**:
   * **Host**: Заголовок, указывающий на адрес сервера, к которому отправляется запрос. Здесь запрашивается ресурс на mansory.com по пути /pln. Этот заголовок необходим для виртуального хостинга.
2. **Connection: keep-alive**:
   * **Connection**: Заголовок, указывающий, что клиент хочет поддерживать соединение открытым после завершения текущего запроса. Это позволяет отправлять несколько запросов по одному и тому же соединению.
3. **Upgrade-Insecure-Requests:**:
   * **Upgrade-Insecure-Requests**: Заголовок, который сообщает серверу, что клиент предпочел бы получить защищенную версию ресурса (HTTPS) вместо незащищенной (HTTP). Если он пустой, это может означать, что клиент не делает конкретных запросов на обновление.
4. **User -Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)**:
   * **User -Agent**: Заголовок, который предоставляет информацию о клиентском программном обеспечении, отправляющем запрос. В данном случае это браузер Chrome на операционной системе Windows 10. Сервер может использовать эту информацию для адаптации ответа в зависимости от типа устройства или браузера.
5. **Accept: text/html, application/xhtml+xml, application/xml;q=0.9,image/avif,image/\*;q=0.8**:
   * **Accept**: Заголовок, который указывает, какие типы контента клиент может обрабатывать. Здесь клиент предпочитает HTML и XHTML, но также может принимать XML и другие форматы изображений. Значения q указывают на приоритет (1.0 — максимальный приоритет, а 0.0 — минимальный).
6. **Accept-Encoding: gzip, deflate**:
   * **Accept-Encoding**: Заголовок, который указывает, какие методы сжатия клиент поддерживает. В этом случае клиент может принимать сжатые данные с использованием алгоритмов gzip или deflate, что помогает уменьшить объем передаваемых данных.
   1. HTTP-ответ



1. **HTTP/1.1 301 Moved Permanently\r\n**

* HTTP/1.1: Версия протокола HTTP, используемая для ответа.
* 301: Код состояния, указывающий, что запрашиваемый ресурс был перемещен навсегда.
* Moved Permanently: Описание кода состояния, объясняющее, что ресурс теперь доступен по другому URL.

1. **Server: nginx\r\n**

* Server: Заголовок, указывающий на серверное программное обеспечение, которое обрабатывает запрос. В данном случае это Nginx.

1. **Date: Mon, 10 Mar 2025 19:49:19 GMT\r\n**

* Date: Заголовок, указывающий дату и время, когда был сгенерирован ответ. Время указано в формате GMT.

1. **Content-Type: text/html; charset=iso-8859-1\r\n**

* Content-Type: Заголовок, указывающий тип содержимого ответа. Здесь это HTML с кодировкой iso-8859-1.

1. **Content-Length: 232\r\n**

* Content-Length: Заголовок, указывающий размер тела ответа в байтах. В данном случае тело ответа содержит 232 байта.

1. **Connection: keep-alive\r\n**

* Connection: Заголовок, указывающий, что сервер хочет поддерживать соединение открытым после завершения текущего запроса, что позволяет отправлять несколько запросов по одному и тому же соединению.

1. **Location: https://www.mansory.com/\r\n**
   * Location: Заголовок, который указывает новый URL, на который клиент должен перенаправиться. В данном случае это https://www.mansory.com/.
2. **Cache-Control: max-age=1209600\r\n**

* Cache-Control: Заголовок, который указывает, как долго (в секундах) ответ может кэшироваться. Здесь указано 1209600 секунд.

1. **Expires: Mon, 24 Mar 2025 19:32:25 GMT\r\n**

* Expires: Заголовок, указывающий дату и время, после которых ответ считается устаревшим. Здесь указана дата 24 марта 2025 года.

1. **X-Varnish: 984760336 985541960\r\n**

* X-Varnish: Заголовок, который используется для отслеживания запросов в системе кэширования Varnish. Указаны идентификаторы запросов.

1. **Age: 1013\r\n**

* Age: Заголовок, указывающий, сколько времени (в секундах) ответ находился в кэше до того, как был отправлен клиенту. Здесь указано 1013 секунд.

1. **Via: 1.1 varnish (Varnish/7.1)\r\n**

* Via: Заголовок, указывающий на промежуточные прокси-серверы, через которые прошел запрос. Здесь указано, что запрос прошел через Varnish версии 7.1.

1. **Cache-Tags: HIT\r\n**

* Cache-Tags: Заголовок, который может использоваться для управления кэшированием. В данном случае указано, что ответ был "HIT", то есть он был найден в кэше.
* \r\n Это разделитель, указывающий на конец заголовков и начало тела ответа.

## Пример с .json

