

# Trabalho Prático: Implementação e Análise - TCP vs TLS

Dhener Rosemíro Silva / 121043412

10 de novembro de 2025

# Conteúdo

<b>1</b>	<b>Introdução e Objetivos</b>	<b>3</b>
<b>2</b>	<b>Funcionamento do TLS (Transport Layer Security)</b>	<b>3</b>
2.1	O Processo de Handshake . . . . .	3
2.2	Criptografia de Dados e Transferência Segura . . . . .	7
2.2.1	Ativação da Camada TLS no Código-Fonte . . . . .	7
2.2.2	Algoritmo e Evidência na Rede . . . . .	8
<b>3</b>	<b>Comparação de Implementações: Segurança e Evidências</b>	<b>9</b>
3.1	1. Confidencialidade (O Conteúdo dos Dados) . . . . .	9
3.1.1	Transmissão Sem TLS (TCP Puro) . . . . .	9
3.1.2	Transmissão Com TLS . . . . .	10
3.2	2. Integridade e Autenticidade . . . . .	11
<b>4</b>	<b>Análise e Discussão do Overhead (Custo do TLS)</b>	<b>11</b>
4.1	Definição das Métricas de Desempenho . . . . .	11
4.2	Dados Quantitativos do Overhead . . . . .	12
4.3	Análise Detalhada das Causas do Overhead . . . . .	12

# 1 Introdução e Objetivos

Esse trabalho tem como objetivo principal desenvolver e analisar a implementação de um sistema de transferência de arquivos utilizando dois protocolos distintos: o **TCP (Transmission Control Protocol)** e o **TLS (Transport Layer Security)** sobre o TCP.

Para isso, foi criado um programa cliente capaz de enviar um arquivo de texto para um servidor em ambas as formas de transmissão. Esse envio deve ser realizado:

- **Sem TLS:** Transmissão em texto plano (não criptografada).
- **Com TLS:** Transmissão utilizando criptografia para garantir a segurança.

Após isso, as diferenças de segurança e desempenho introduzidas pelo uso do TLS serão analisadas e comparadas, com base em evidências de captura de pacotes de rede e métricas de *overhead* temporal.

## 2 Funcionamento do TLS (Transport Layer Security)

O TLS é um protocolo criptográfico que opera na camada de transporte, oferecendo segurança, integridade e autenticidade sobre uma rede. É o sucessor do SSL e estabelece uma comunicação segura em duas fases: a negociação (*Handshake*) e a transferência de dados cifrados.

### 2.1 O Processo de Handshake

O **Handshake TLS** é a etapa inicial onde cliente e servidor negociam e estabelecem os parâmetros de segurança para a sessão. A análise da captura de pacotes confirma as principais etapas:

1. **Client Hello (Iniciação da Negociação):** O cliente inicia o *handshake* enviando esta mensagem para informar ao servidor suas capacidades de segurança e iniciar a negociação. O pacote contém:
  - **Versão e Cifras Suportadas:** O cliente lista as versões de TLS que suporta (ex: TLS 1.2, TLS 1.3) e envia a lista de algoritmos de cifras (*Cipher Suites*) preferidas. O servidor usará esta lista para escolher o conjunto criptográfico a ser usado na sessão.
  - **Client Random:** Um número aleatório de 32 *bytes*. Este dado é crítico pois, combinado com o número aleatório do servidor (*Server Random*), será usado no final do processo para derivar a **Chave Mestra** (e, consequentemente, a chave de sessão simétrica).
  - **ID de Sessão (Session ID):** Este campo é usado para o mecanismo de Retomada de Sessão (*Session Resumption*).
    - Um comprimento de \*\*zero\*\* (como visto no seu teste) indica que o cliente está iniciando um *Handshake Completo*, que exige mais processamento.
    - Um valor diferente de zero indicaria uma tentativa de retomar uma sessão anterior, o que otimiza o processo e reduz o *overhead*.

- **Extensões (Extensions):** São campos adicionais que expandem a funcionalidade do TLS. As mais relevantes são:

- **server\_name (SNI):** Permite ao cliente informar qual nome de domínio ele está tentando acessar, útil para servidores que hospedam múltiplos sites (virtual hosting).
- **supported\_groups:** O cliente lista as curvas elípticas (x25519) ou grupos Diffie-Hellman que ele aceita. O servidor escolherá um para realizar o cálculo de Troca de Chaves (ECDHE).

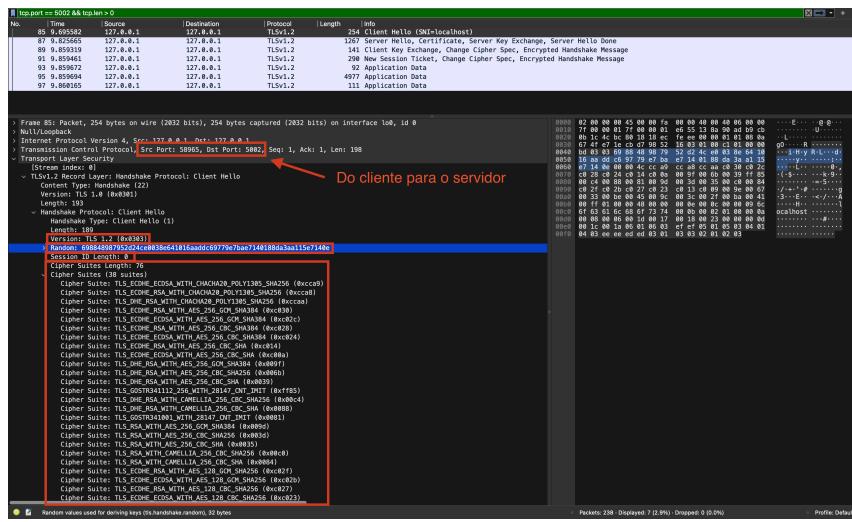


Figura 1: Captura do Client Hello no Wireshark na Transmissão TLS - Parte 1.

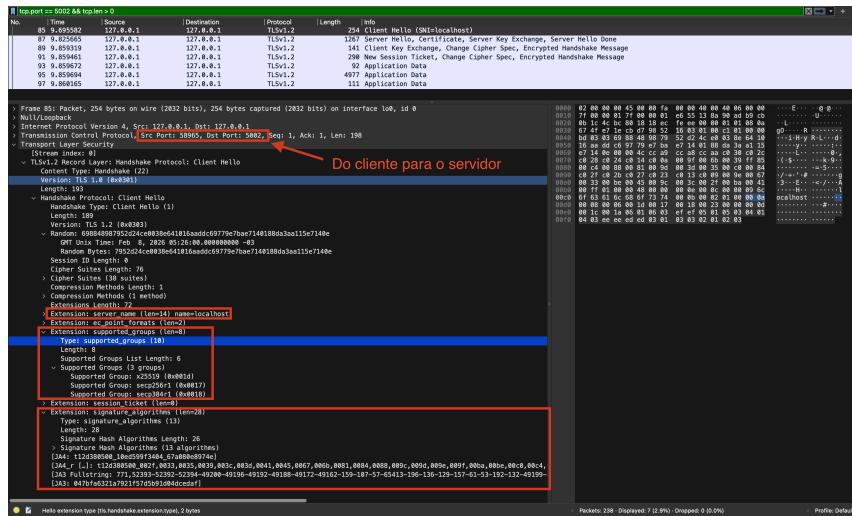


Figura 2: Captura do Client Hello no Wireshark na Transmissão - TLS Parte 2.

2. **Server Hello & Certificado:** O servidor escolhe a melhor cipher suite (campo **Cipher Suite**), envia o número aleatório (campo **Random**), que será combinado com o número aleatório do cliente para gerar a **Chave de Sessão Secreta**, confirma que o servidor aceitou a versão proposta (campo **Version**), que foi TLS 1.2

no caso, mostra o envio da Cadeia de Certificação para autenticação, no caso o certificado autoassinado server.crt que foi gerado, conforme o código server.py (campo **Handshake Protocol: Certificate**).

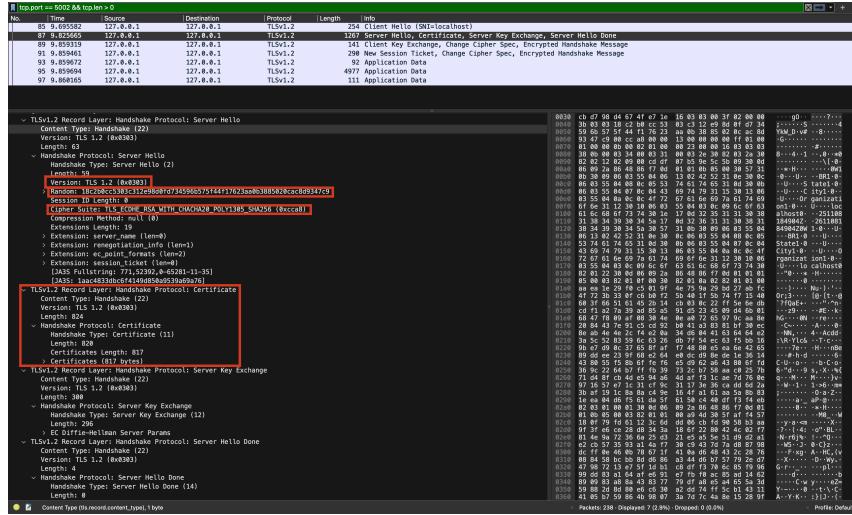


Figura 3: Captura do Server Hello e Certificate no Wireshark na Transmissão TLS.

**Troca de Chaves Efêmera (Server Key Exchange):** O pacote Server Key Exchange é a prova física de que a conexão utiliza o método avançado **ECDHE** (*Elliptic Curve Diffie-Hellman Ephemeral*), garantindo o **Perfect Forward Secrecy (PFS)**. Este processo de troca de chaves cumpre três funções cruciais:

- Derivação e Algoritmo:** O campo EC Diffie-Hellman Server Params confirma o uso do ECDHE, que permite que o cliente e o servidor **derivem** a *Chave Pré-Mestra* de forma independente, sem trocá-la de forma criptografada.
- Curva e Chave Pública Efêmera:** O servidor especifica a curva elíptica moderna x25519 (campo Named Curve) e envia sua **chave pública temporária** (Pubkey). Esta chave é usada apenas nesta sessão e será descartada, sendo a base do PFS.
- Autenticidade:** A chave pública efêmera e seus parâmetros são digitalmente **assinados** pelo servidor (campo **Signature**), utilizando a chave privada de longo prazo vinculada ao certificado. O cliente verifica essa assinatura, garantindo que a troca de chaves é autêntica e veio do servidor correto.

A *Chave Pré-Mestra* resultante, combinada com os *randoms* de ambos os lados, é usada para derivar a **Chave de Sessão Simétrica** final.

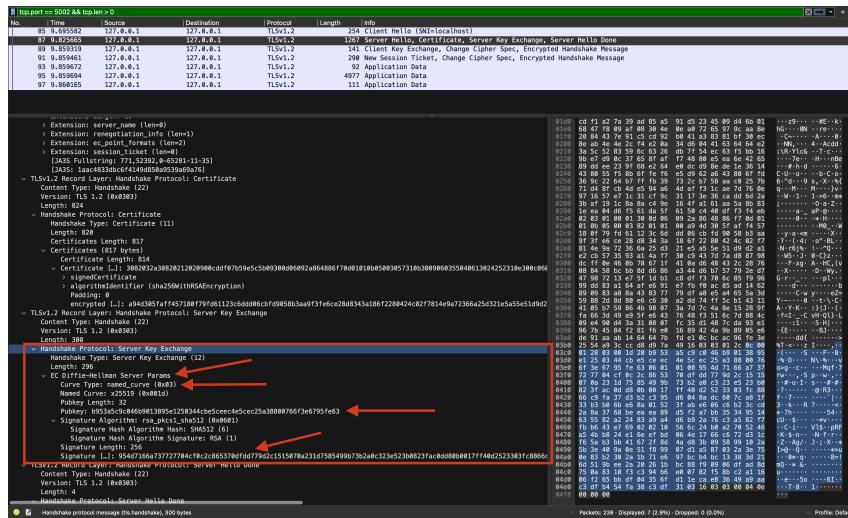


Figura 4: Captura do Server Hello para troca de chaves na Transmissão TLS.

**3. Change Cipher Spec:** Ambas as partes sinalizam que toda a comunicação subsequente será cifrada usando a chave de sessão recém-gerada. Essa mensagem, por si só, não contém dados de criptografia. Sua função é puramente sinalizadora. Ela é como um interruptor, dizendo à parte receptora: "A partir do próximo pacote que você receber, comece a usar as chaves secretas que acabamos de negociar."

- O cliente envia primeiro, sinalizando que ele está pronto para usar as chaves de sessão recém-derivadas para cifrar a próxima mensagem. Rótulo do Pacote (Client → Server): TLSv1.2 Record Layer: Change Cipher Spec

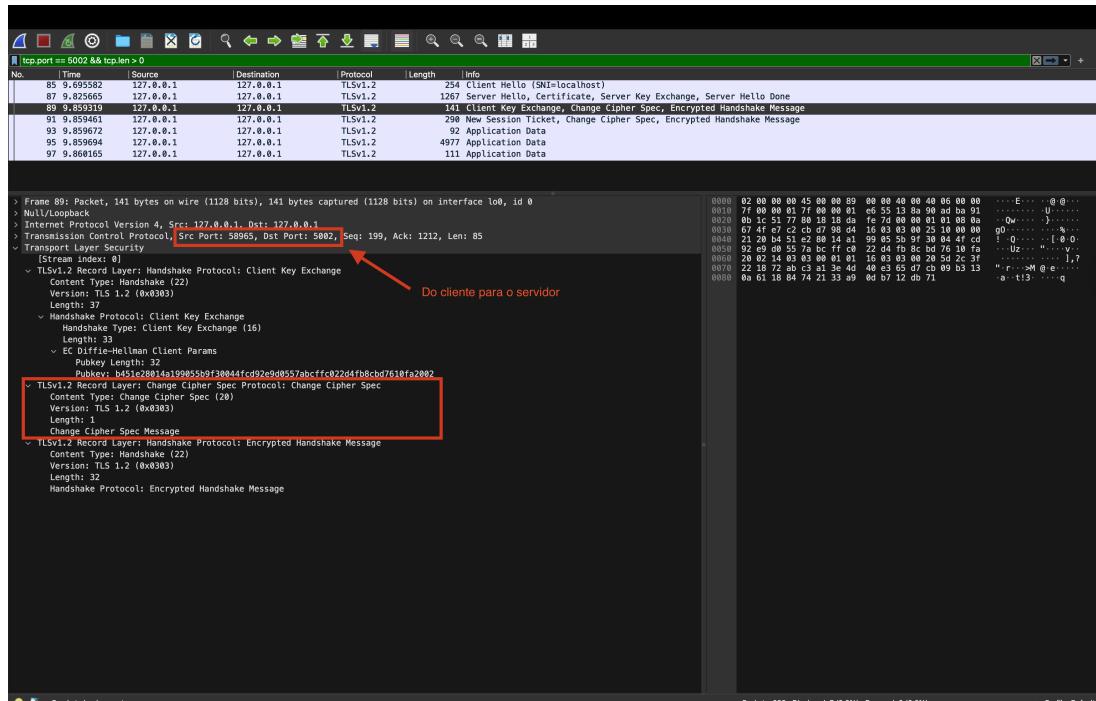


Figura 5: Captura do Change Cipher Spec do cliente para o servidor na Transmissão TLS.

- O servidor envia sua própria mensagem Change Cipher Spec em resposta, confirmando que ele também está ativando as chaves de sessão simétricas. Rótulo do Pacote (Server → Client): TLSv1.2 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec

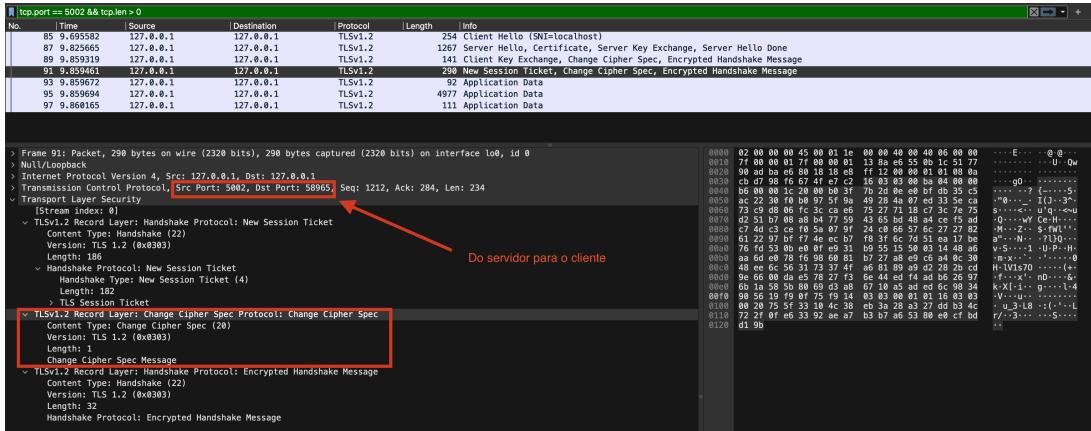


Figura 6: Captura do Change Cipher Spec do servidor para o cliente na Transmissão TLS.

Este processo complexo garante a autenticidade do servidor e o sigilo na negociação da chave.

## 2.2 Criptografia de Dados e Transferência Segura

Uma vez que o *handshake* é concluído e a **Chave de Sessão Simétrica** é estabelecida, a comunicação de dados prossegue no protocolo de registro TLS (TLS Record Protocol). Esta fase é crucial para garantir a **Confidencialidade** e a **Integridade** dos dados transferidos.

### 2.2.1 Ativação da Camada TLS no Código-Fonte

No lado do cliente, a criptografia é ativada envolvendo um *socket* TCP base com uma camada SSL/TLS, utilizando o módulo padrão `ssl` do Python.

O trecho de código no `client.py` demonstra a criação do contexto e o envolvimento do *socket*:

```

def connect_tls(self, certfile: Optional[str] = None) -> bool:
    """Estabelece conexão TCP com TLS"""
    try:
        # Cria socket base
        base_socket = self._create_socket()

        # Configura contexto SSL/TLS
        context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)

        # Para ambiente de desenvolvimento (aceita certificados auto-assinados)
        context.check_hostname = False
        context.verify_mode = ssl.CERT_NONE

        # Envolve o socket com TLS
        self.socket = context.wrap_socket(
            base_socket,
            server_hostname=self.host
        )

        print(f"[TLS] Conectando a {self.host}:{self.port}...")
        self.socket.connect((self.host, self.port))

        # Exibe informações da conexão TLS
        cipher = self.socket.cipher()
        version = self.socket.version()
        print(f"[TLS] Conexão estabelecida com sucesso!")
        print(f"[TLS] Protocolo: {version}")
        print(f"[TLS] Cipher: {cipher[0]}")

        return True
    except Exception as e:
        print(f"[ERRO] Falha na conexão TLS: {e}")
        return False

```

Figura 7: Captura de parte do client.py do método connect\_tls.

Após o sucesso do `context.wrap_socket`, qualquer chamada subsequente ao método `send()` neste `socket` automaticamente cifra os dados antes da transmissão.

### 2.2.2 Algoritmo e Evidência na Rede

A segurança na transferência ocorre pelos seguintes mecanismos, que podem ser verificados nas capturas de pacotes:

- **Criptografia Simétrica (Confidencialidade):** O algoritmo de cifragem simétrica é o responsável por codificar os dados.
  - **Algoritmo Negociado:** O *Server Hello* confirmou o uso do conjunto de cifras `TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256`. Isso estabelece que a criptografia de dados é realizada pelo algoritmo **ChaCha20** (cifra de fluxo moderna).
  - **Evidência:** Os pacotes de dados são identificados como `Application Data` e o seu *payload* contém *bytes* cifrados, confirmando que o ChaCha20 está ativo.

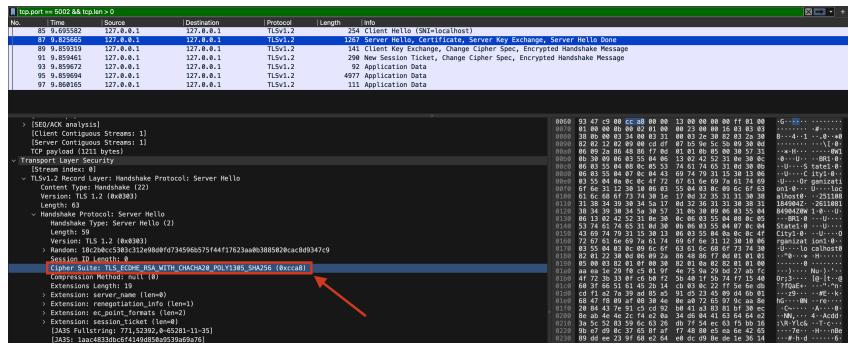


Figura 8: Captura do Server Hello do campo `cipher_suite`.

- **Integridade (MAC/Tag de Autenticação):** Cada registro TLS é protegido contra adulteração.
  - A cifra ChaCha20 é emparelhada com o \*\*Poly1305\*\* (do conjunto de cifras negociado), formando um algoritmo de *cifra autenticada* (AEAD).
  - O Poly1305 gera um *Authentication Tag* para cada bloco de dados. O receptor verifica essa *tag*, garantindo a **Integridade** e a autenticidade dos dados do arquivo.

### 3 Comparação de Implementações: Segurança e Evidências

A comparação entre as implementações sem e com TLS é fundamental para demonstrar o valor da criptografia em redes. Esta seção contrasta os protocolos com base nas garantias de segurança e utiliza as capturas de pacotes de rede (Wireshark) como evidência empírica.

#### 3.1 1. Confidencialidade (O Conteúdo dos Dados)

##### 3.1.1 Transmissão Sem TLS (TCP Puro)

Na implementação sem TLS (porta 5001), o protocolo TCP puro não adiciona nenhuma camada de segurança, e a transmissão ocorre em **texto plano**.

- **Análise:** A captura de pacotes (Figura 9) mostra o conteúdo literal do arquivo de teste ("Este é um arquivo de teste...") visível no *payload* do segmento TCP.
- **Conclusão:** A confidencialidade é **nula**. Qualquer *sniffing* na rede expõe os dados.

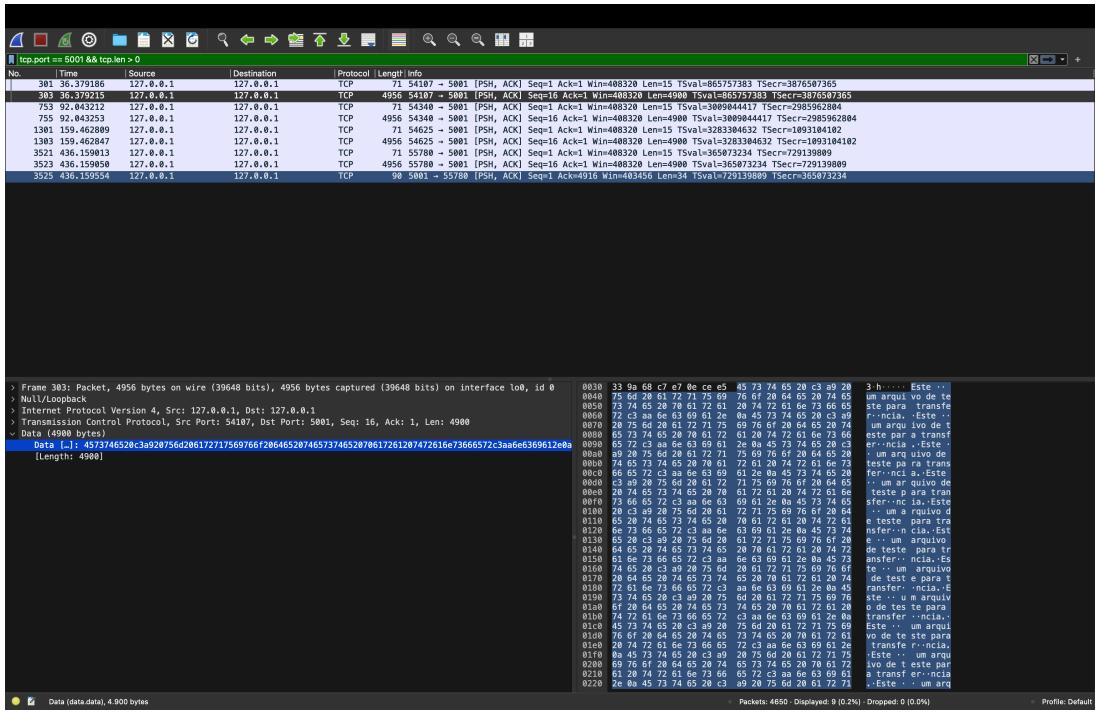


Figura 9: Evidência de Texto Plano (TCP Puro). Conteúdo visível no payload.

### 3.1.2 Transmissão Com TLS

Na implementação com TLS (porta 5002), a chave de sessão secreta (negociada no *handshake*) é usada para criptografar o arquivo antes do envio.

- Análise:** A captura (Figura 10) mostra que o tráfego de dados é categorizado como **TLSv1.2 Record Layer: Application Data**. O *payload* contém apenas sequências ilegíveis de bytes cifrados.
- Conclusão:** A confidencialidade é garantida, pois os dados estão protegidos contra interceptação.

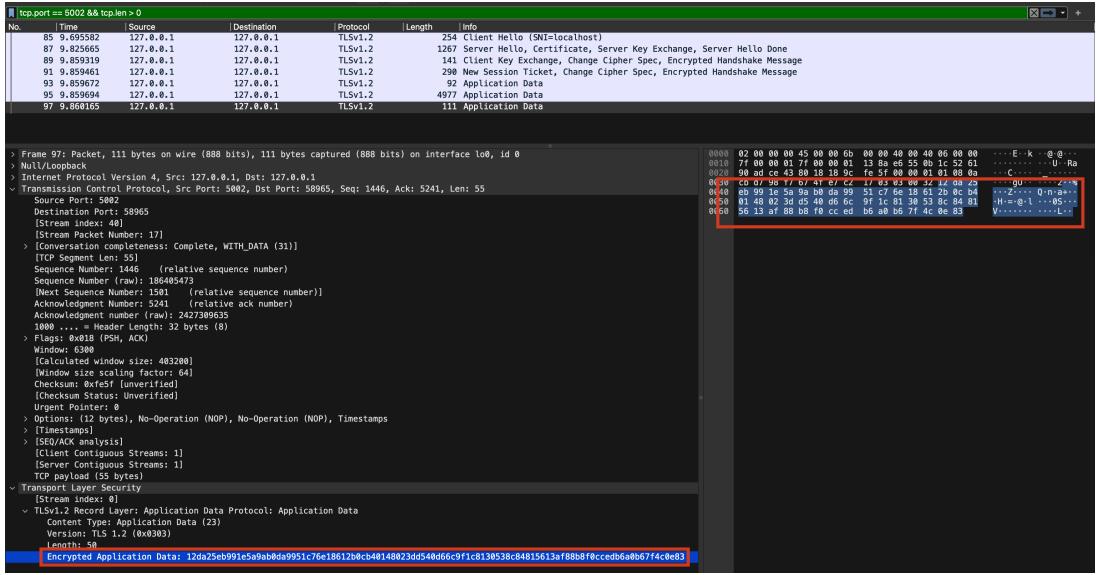


Figura 10: Evidência de Envio Cifrado (TLS). Payload ilegível (Encrypted Application Data).

### 3.2 2. Integridade e Autenticidade

Garantia de Segurança	TCP Puro (Sem TLS)	TLS
<b>Autenticidade (Identidade)</b>	Nenhuma	<b>Garantida</b> (via Certificado Digital)
<b>Integridade (Anti-adulteração)</b>	Nenhuma	<b>Garantida</b> (via MAC/Hash)

Tabela 1: Comparaçāo de Mecanismos de Segurança

- Autenticidade:** Na implementação TCP, não há verificação de identidade; o cliente aceita a conexão de qualquer servidor na porta 5001. No TLS, o servidor envia seu certificado digital para provar sua identidade durante o *handshake*, impedindo ataques *Man-in-the-Middle*.
- Integridade:** O TCP usa apenas um *checksum* básico. O TLS, por outro lado, anexa um **Message Authentication Code (MAC)** criptográfico a cada registro de dados, garantindo que o pacote não foi alterado intencionalmente no caminho.

## 4 Análise e Discussão do Overhead (Custo do TLS)

O uso do TLS, embora indispensável para garantir confidencialidade e integridade, introduz um custo de desempenho conhecido como *overhead*. Este custo é a consequência direta do trabalho criptográfico exigido pelo protocolo. Esta seção analisa o impacto quantitativo do TLS comparado ao TCP puro, utilizando os dados coletados através do `performance_analyzer.py`.

### 4.1 Definição das Métricas de Desempenho

Para quantificar o impacto, foram coletadas as seguintes métricas, que contrastam o desempenho da transferência em texto plano (TCP) e cifrada (TLS):

- **Tempo Médio:** O tempo total gasto, em segundos, desde o estabelecimento da conexão até a conclusão da transferência do arquivo. No TLS, esta métrica inclui o tempo do *handshake* e o tempo de criptografia/descriptografia dos dados.
- **Throughput Médio (Vazão):** A taxa de transferência efetiva de dados, medida em megabytes por segundo (MB/s). É calculada dividindo o tamanho do arquivo pelo tempo de transferência. O *throughput* é inversamente proporcional ao *overhead* de tempo.
- **Fator de Desaceleração:** Um multiplicador que indica quantas vezes o processo com TLS é mais lento que o processo sem criptografia (TCP Puro). Um fator de 2.22x, por exemplo, significa que o TLS leva 2.22 vezes mais tempo.
- **Impacto / Overhead:** A variação percentual que o TLS introduziu na métrica. Um *overhead* de tempo de **+121.95%** significa que o tempo de transferência aumentou 121.95%. Uma redução de *throughput* de **-69.04%** indica que a vazão caiu quase 70%.

## 4.2 Dados Quantitativos do Overhead

As métricas foram coletadas através de 10 testes para cada protocolo, utilizando o script `performance_analyzer.py`. Os resultados médios (Tabela 2) demonstram a diferença de eficiência.

Tabela 2: Comparação Quantitativa de Desempenho (Média de 10 Testes)

Métrica	TCP Puro	TLS	Impacto / Overhead
Tempo Médio	0.000129s	0.000286s	<b>+121.95%</b>
Throughput Médio	520.84 MB/s	161.28 MB/s	<b>-69.04%</b>
Fator de Desaceleração	1.00x	2.22x	TLS é 2.22x mais lento

## 4.3 Análise Detalhada das Causas do Overhead

O *overhead* de tempo de **121.95%** e a consequente **redução de 69.04% no throughput** são atribuídos a fatores que consomem ciclos da CPU e introduzem latência de rede.

1. **Latência do Handshake Inicial:** O processo de negociação de cifras, troca de chaves (ECDHE) e autenticação de certificado (RSA) é executado antes que o primeiro *byte* de dado do arquivo seja transmitido. Este *handshake* exige múltiplas trocas de mensagens (\*Round-Trip Times\*), consumindo o tempo inicial e impactando diretamente o **Tempo Médio** total, especialmente em transferências de arquivos pequenos.
2. **Criptografia e Descriptografia Contínua (Overhead de CPU):** O custo mais significativo no *throughput* é introduzido pela execução dos algoritmos criptográficos (ChaCha20-Poly1305 no seu caso). Cada bloco de dados enviado exige que o cliente o cifre e que o servidor o decifre, consumindo ciclos da CPU em tempo real durante toda a transferência. Esse gasto computacional prolonga o **Tempo Médio** de forma linear ao tamanho do arquivo.

**3. Processamento de Integridade e Overhead de Tamanho:** O TLS não apenas cifra, mas também garante a integridade e a autenticidade. O cálculo e a inclusão do *Authentication Tag* (Poly1305), juntamente com o cabeçalho do protocolo de registro TLS, adicionam um *overhead* de tamanho a cada pacote. Os dados do teste indicam um **overhead médio de 28 bytes** por pacote. Embora o impacto do tamanho seja menor que o impacto temporal, ele contribui para o *throughput* reduzido.

Em conclusão, a análise demonstra que o \*\*TLS é 2.22x mais lento\*\* (Fator de Desaceleração) que o TCP Puro. No entanto, este custo é amplamente compensado pela garantia de **confidencialidade, integridade e autenticidade** que o TLS oferece, sendo um requisito indispensável para a transmissão segura de informações em qualquer ambiente de rede profissional.

## Conclusão

Este trabalho demonstrou a diferença crítica entre o transporte de dados em texto plano (TCP) e o transporte seguro (TLS). Apesar do *overhead* de desempenho, o custo é justificado pela garantia de confidencialidade, integridade e autenticidade, essenciais para a segurança moderna em redes de computadores.