# Sprawozdanie Końcowe

Projekt: Mini-TLS (Wariant W1)

**Zespół nr 39**
Skład zespołu:
Łukasz Szydlik (lider)
Jan Szwagierczak
Dominik Śledziewski

## 1 Wstęp

Celem projektu jest implementacja bezpiecznego protokołu komunikacyjnego (mini TLS) w architekturze klient-serwer. Protokół zapewnia poufność oraz integralność przesyłanych danych przy użyciu kryptografii symetrycznej i asymetrycznej. Projekt realizowany jest w języku Python, a komunikacja odbywa się w odizolowanym środowisku Docker.

Wybrany wariant realizacyjny to **W1**: Wykorzystanie mechanizmu *Encrypt-then-MAC* dla zapewnienia integralności i autentyczności wiadomości.

## 2 Opis użytych algorytmów

W projekcie wykorzystano następujące prymitywy kryptograficzne:

1. **Wymiana kluczy: Diffie-Hellman (DH)** - Parametry $P$ i $G$ są dynamicznie przesyłane przez klienta, co pozwala uniknąć ich hardkodowania po stronie serwera.
2. **Derywacja kluczy (KDF)** - Ze wspólnego sekretu wyprowadzane są dwa niezależne klucze przy użyciu SHA-256 z odpowiednimi sufiksami ("ENC"i "MAC").
3. **Szyfrowanie: XOR Stream Cipher** - Symulacja mechanizmu One-Time Pad (OTP). Klucz derywowany służy jako ziarno dla generatora PRNG, tworząc strumień klucza.
4. **Integralność: HMAC-SHA256** - Kod uwierzytelniający obliczany jest na szyfrogramie (*Encrypt-then-MAC*), co gwarantuje integralność danych i autentyczność pochodzenia.

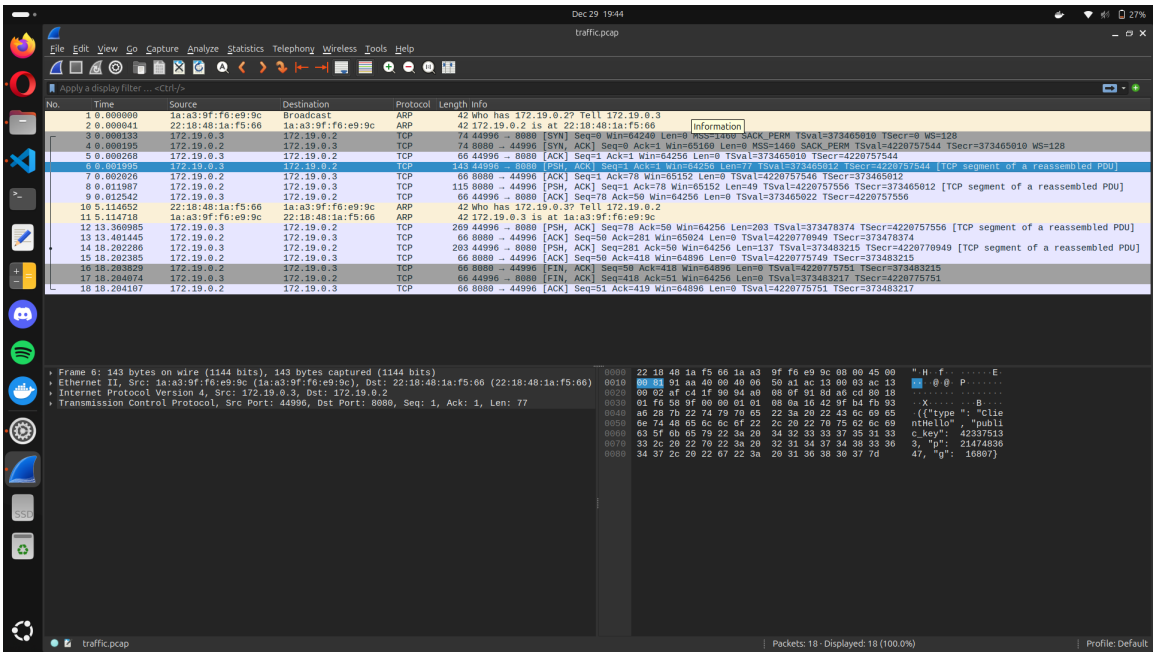
## 3 Dowód działania protokołu

Poniższa sekcja przedstawia poprawność przebiegu komunikacji przechwyconej za pomocą narzędzia Wireshark.

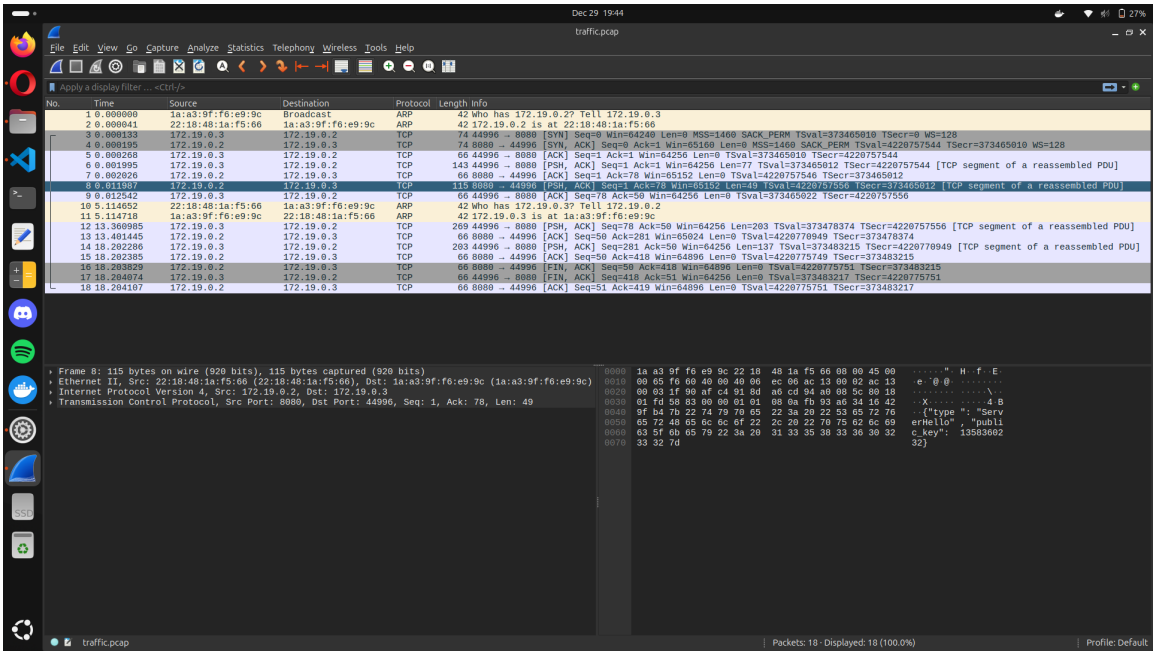
### 3.1 Faza Handshake (Nieszyfrowana)

Komunikacja rozpoczyna się od wymiany kluczy publicznych i parametrów DH jawnym tekstem w formacie JSON.

- **ClientHello**: Klient inicjuje połączenie, przesyłając swój klucz publiczny oraz proponowane parametry $P$ i $G$.
- **ServerHello**: Serwer akceptuje parametry i odpowiada własnym kluczem publicznym.




Rysunek 1: Przechwycony pakiet ClientHello z widocznymi parametrami P, G i kluczem publicznym.

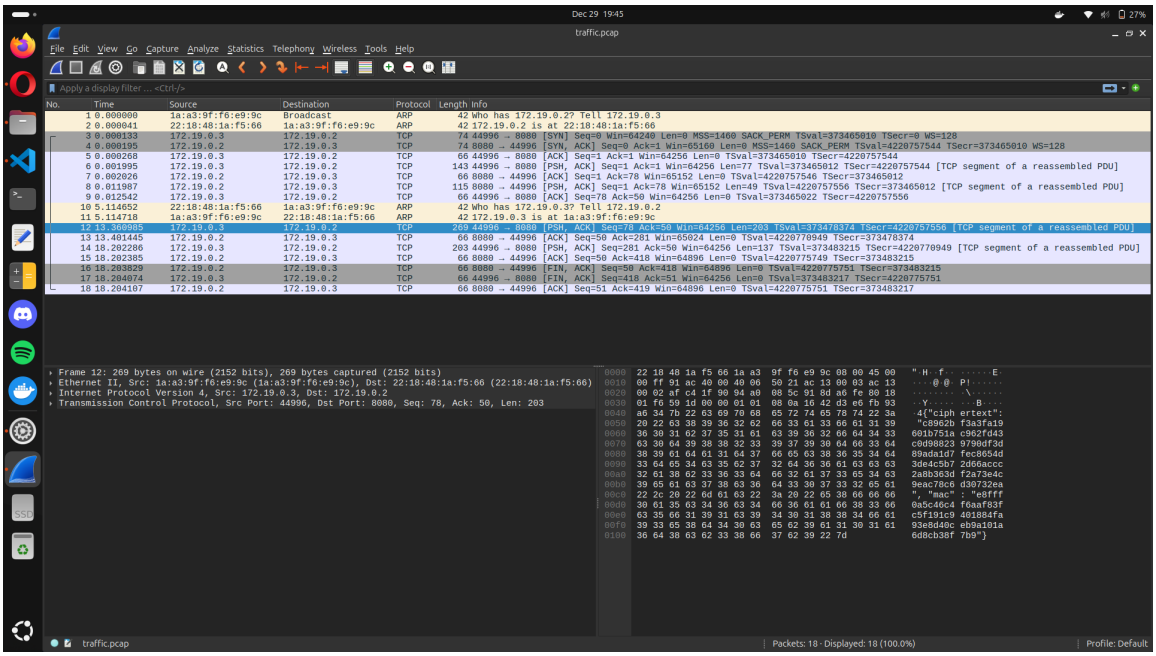



Rysunek 2: Przechwycony pakiet ServerHello (odpowiedź serwera).

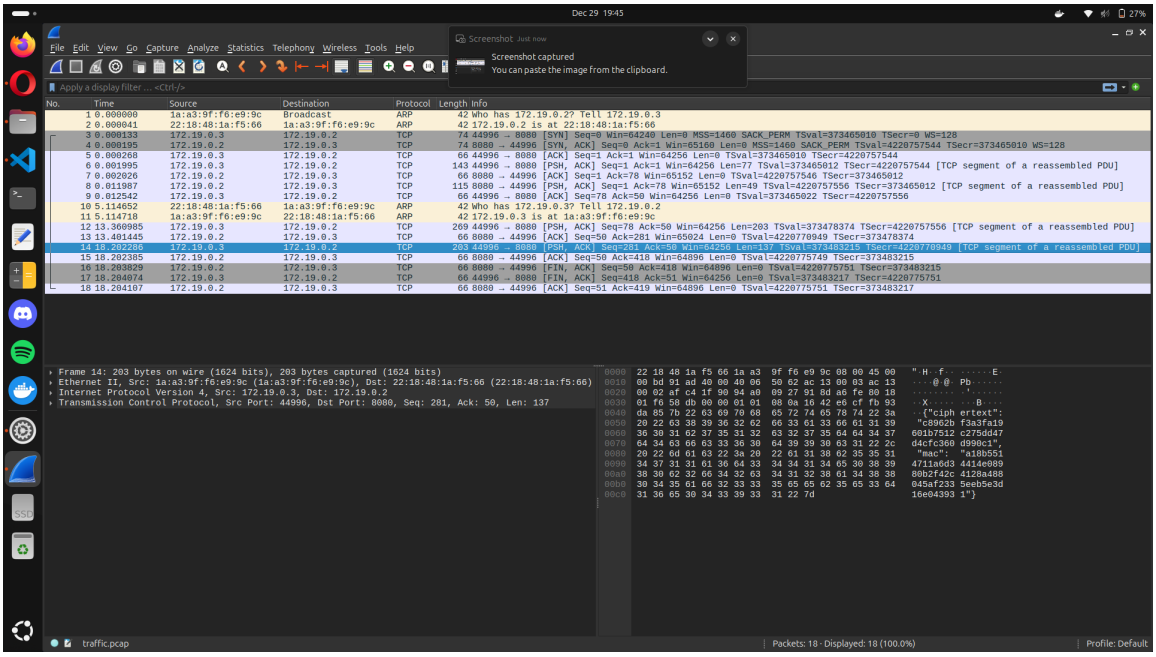
## 3.2 Faza Transferu Danych (Szyfrowana)

Po ustaleniu wspólnego sekretu, wszystkie kolejne wiadomości są szyfrowane. Pakiety przesyłane są w formacie JSON zawierającym wyłącznie pola `ciphertext` (dane zaszyfrowane w hex) oraz `mac` (tag uwierzytelniający).

- Poniższy zrzut ekranu przedstawia pierwszy zaszyfrowany pakiet, zawierający treść właściwej wiadomości tekstowej wysłanej przez klienta.
- Kolejny przechwycony pakiet zawiera zaszyfrowany komunikat zakończenia sesji (`EndSession`).



Rysunek 3: Pierwsza zaszyfrowana wiadomość w sesji (widoczne nieczytelne pola ciphertext i mac).



Rysunek 4: Druga zaszyfrowana wiadomość (EndSession).

# 4 Weryfikacja szyfrowania i integralności

Zgodnie z wymogami, przeprowadzono ręczną weryfikację poprawności deszyfrowania oraz odporności na manipulację przy użyciu zewnętrznego skryptu weryfikującego.

```
{
=== Mini-TLS Message Verification ===

[INPUT] Message file: tests/ciphered_massege1.json
[INPUT] Secrets file: captures/server_secrets.log
```

```

[DATA] Ciphertext: c8962bf3a3fa19601b751ac962fd43c0d988239790df3d89...
[DATA] MAC: e8fff0a5c46c4f6aaf83fc5f191c9401884fa93e8d40ceb9a101a6d8cb38
   f7b9
[DATA] ENC_KEY: 02f64d7068e17b5c04ed147051ddbafd...
[DATA] MAC_KEY: 0d295c9e6d4a1e5f67919fb24912a491...

[1/3] Verifying MAC integrity...
      PASS - MAC valid, message authentic

[2/3] Tampering simulation...
      PASS - Modified message rejected

[3/3] Decrypting message...
      PASS - Decryption successful
      Plaintext: {"type": "Message", "content": "TajnaWiadomoscTestowa"}
=== Verification Complete ===
}
```

```
{
=== Mini-TLS Message Verification ===

[INPUT] Message file: tests/ciphered_massege2.json
[INPUT] Secrets file: captures/server_secrets.log

[DATA] Ciphertext: c8962bf3a3fa19601b7512c275dd47d4cfc360d990c1...
[DATA] MAC: a18b5514711a6d34414e08980b2f42c4128a488045af2335eeb5e3d16e04
   3931
[DATA] ENC_KEY: 02f64d7068e17b5c04ed147051ddbafd...
[DATA] MAC_KEY: 0d295c9e6d4a1e5f67919fb24912a491...

[1/3] Verifying MAC integrity...
      PASS - MAC valid, message authentic

[2/3] Tampering simulation...
      PASS - Modified message rejected

[3/3] Decrypting message...
      PASS - Decryption successful
      Plaintext: {"type": "EndSession"}
=== Verification Complete ===

}
```

### 4.1 Proces deszyfrowania

Do weryfikacji użyto kluczy sesji zapisanych w pliku `server_secrets.log` oraz szyfrogramów z Wiresharka. Wyniki testów potwierdziły poprawność implementacji:

- **Wiadomość 1**: Szyfrogram `c896...` został odszyfrowany do postaci: `{"type": "Message", "content": "TajnaWiadomoscTestowa"}`.
- **Wiadomość 2**: Szyfrogram `c896...` (krótszy) został odszyfrowany do postaci: `{"type": "EndSession"}`.

### 4.2 Test integralności (Tampering)

W ramach weryfikacji symulowano próbę ataku poprzez modyfikację ostatniego bajtu szyfrogramu. Skrypt weryfikujący poprawnie wykrył niezgodność tagu MAC i odrzucił zmodyfikowany pakiet, co dowodzi skuteczności mechanizmu *Encrypt-then-MAC*.

## 5 Napotkane problemy i wnioski

### 5.1 Napotkane problemy

- **Dynamiczne parametry DH**: Początkowo parametry były zahardkodowane. Zostało to poprawione poprzez dodanie pól $P$ i $G$ do wiadomości `ClientHello`.

### 5.2 Wnioski

Zaimplementowany protokół skutecznie chroni przesyłane dane przed podsłuchem i modyfikacją. Wybór wariantu *Encrypt-then-MAC* okazał się optymalny, gdyż pozwala na weryfikację integralności bez konieczności kosztownego (i potencjalnie podatnego na ataki) deszyfrowania nieautentycznych danych. Użycie standardu JSON do strukturyzacji pakietów zapewniło czytelność i łatwość rozbudowy protokołu.