

# Sprawozdanie końcowe

## Projekt: Mini TLS – wariant W1 (Encrypt-then-MAC)

Alesia Filinkova, Weronika Maślana, Diana Pelin

Dowód że program działa

Serwer

```
[+] Server listening on port 4444
server> [+] Client connected: ('172.18.0.3', 54554)
[+] Client ID=1 from ('172.18.0.3', 54554)
[1] secretMessage
EndSession 1
[+] Sent EndSession to client 1
server> [-] Client 1 disconnected
```

Klient

```
[+] Secure session established
>> secretMessage
>>
[!] Session ended by server
[+] Client terminated
```

Komunikacja między serwerem i klientem

Time	Source	Destination	Protocol	Length	Info
1 0.000000	172.18.0.3	172.18.0.2	TCP	80	54554 → 4444 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1556162827 TSecr=0 WS=128
2 0.000021	172.18.0.2	172.18.0.3	TCP	88	4444 → 54554 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1253500501 TSecr=1556162827 WS=128
3 0.000049	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1556162828 TSecr=1253500501
4 0.000408	172.18.0.3	172.18.0.2	TCP	86	54554 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=14 TSval=1556162828 TSecr=1253500501
5 0.000419	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [ACK] Seq=1 Ack=15 Win=65152 Len=0 TSval=1253500501 TSecr=1556162828
6 0.000956	172.18.0.2	172.18.0.3	TCP	86	4444 → 54554 [PSH, ACK] Seq=1 Ack=15 Win=65152 Len=14 TSval=1253500501 TSecr=1556162828
7 0.001011	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=15 Ack=15 Win=64256 Len=0 TSval=1556162828 TSecr=1253500501
8 22.165231	172.18.0.3	172.18.0.2	TCP	117	54554 → 4444 [PSH, ACK] Seq=15 Ack=15 Win=64256 Len=45 TSval=1556184993 TSecr=1253500501
9 22.206076	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [ACK] Seq=15 Ack=60 Win=65152 Len=0 TSval=1253522787 TSecr=1556184993
10 54.904099	172.18.0.2	172.18.0.3	TCP	114	4444 → 54554 [PSH, ACK] Seq=15 Ack=60 Win=65152 Len=42 TSval=1253555406 TSecr=1556184993
11 54.905051	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=60 Ack=57 Win=64256 Len=0 TSval=1556217733 TSecr=1253555406
12 57.844635	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [FIN, ACK] Seq=60 Ack=57 Win=64256 Len=0 TSval=1556220672 TSecr=1253555406
13 57.844872	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [FIN, ACK] Seq=57 Ack=61 Win=65152 Len=0 TSval=1253558346 TSecr=1556220672
14 57.844927	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=61 Ack=58 Win=64256 Len=0 TSval=1556220673 TSecr=1253558346

### 1. Nawiązanie połączenia

Time	Source	Destination	Protocol	Length	Info
1 0.000000	172.18.0.3	172.18.0.2	TCP	80	54554 → 4444 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1556162827 TSecr=0 WS=128
2 0.000021	172.18.0.2	172.18.0.3	TCP	88	4444 → 54554 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1253500501 TSecr=1556162827 WS=128
3 0.000049	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1556162828 TSecr=1253500501

  

Frame 1: Packet, 80 bytes on wire (640 bits), 80 bytes captured (640 bits)	0000 00 00 00 00 00 00 36 00 01 00 06 02 42 ac 12 .....6...B...
Linux cooked capture v2	0010 00 03 00 00 05 00 0c 92 61 00 00 00 00 50 31 .....E...@P1...
Internet Protocol Version 4, Src: 172.18.0.3, Dst: 172.18.0.2	0020 ac 12 00 03 ac 12 00 02 d5 1a 11 5c 58 36 9a 83 .....XX.....
Transmission Control Protocol, Src Port: 54554, Dst Port: 4444, Seq: 0, Len: 0	0030 00 00 00 0a 02 fa f0 58 58 00 00 02 04 05 b4 .....XX.....
	0040 02 08 0a 5c c1 29 0b 00 00 00 01 03 03 07 ..... ).....

Nawiązanie połączenia pomiędzy klientem a serwerem odbywa się zgodnie z mechanizmem TCP three-way handshake. Jak pokazano na zrzucie z Wiresharka, klient inicjuje połączenie pakietem SYN, serwer odpowiada pakietem SYN-ACK, a następnie klient potwierdza połączenie pakietem ACK. Dopiero po poprawnym zestawieniu połączenia TCP rozpoczyna się właściwa komunikacja

## 2. Wysłanie wiadomości ClientHello

4	0.000408	172.18.0.3	172.18.0.2	TCP	86	54554 → 4444 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=14 TSval=1556162828 TSecr=1253500501
5	0.000419	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [ACK] Seq=1 Ack=15 Win=65152 Len=0 TSval=1253500501 TSecr=1556162828

  

> Frame 4: Packet, 86 bytes on wire (688 bits), 86 bytes captured (688 bits)	0000	08 00 00 00 00 00 36 00 01 04 06 02 42 ac 12	.....6....B....
> Linux cooked capture v2	0010	00 02 00 00 45 00 00 42 bf 49 40 00 40 06 23 43	....E..B...I@#@C
> Internet Protocol Version 4, Src: 172.18.0.3, Dst: 172.18.0.2	0020	ac 12 00 03 ac 12 00 02 d5 1a 11 5c 58 36 9a 84	....E..B...c@eP)
> Transmission Control Protocol, Src Port: 54554, Dst Port: 4444, Seq: 1, Ack: 1, Len: 14	0030	79 b5 26 58 00 18 01 f6 58 5e 00 00 01 01 08 0a	....y&X.....X^.....
> Data (14 bytes)	0040	5c c1 29 8c 4a b6 e6 55 43 6c 69 65 6e 74 48 65	..J.J..U ClientHe
Data: 436c69656e7448656c66f3a370a	0050	6c 6c 6f 3a 37 0a	llo:7)
[Length: 14]			

Po zestawieniu połączenia TCP klient wysła wiadomość ClientHello. Wiadomość tą przesyłana jest w postaci jawnej. Kolejny pakiet zawiera jedynie flagę ACK i stanowi potwierdzenie odebrania ClientHello przez serwer na poziomie protokołu TCP, bez przysyłania danych aplikacyjnych

## 3. Wysłanie wiadomości ServerHello

6	0.000956	172.18.0.2	172.18.0.3	TCP	86	4444 → 54554 [PSH, ACK] Seq=1 Ack=15 Win=65152 Len=14 TSval=1253500501 TSecr=1556162828
7	0.001011	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=15 Ack=15 Win=64256 Len=0 TSval=1556162828 TSecr=1253500501

  

> Frame 6: Packet, 86 bytes on wire (688 bits), 86 bytes captured (688 bits)	0000	08 00 00 00 00 00 36 00 01 04 06 02 42 ac 12	.....6....B....
> Linux cooked capture v2	0010	00 02 00 00 45 00 00 42 bf 49 40 00 40 06 23 43	....E..B...I@#@C
> Internet Protocol Version 4, Src: 172.18.0.2, Dst: 172.18.0.3	0020	ac 12 00 02 ac 12 00 03 11 5c d5 1a 79 b5 26 58	....E..B...I@#@C
> Transmission Control Protocol, Src Port: 4444, Dst Port: 54554, Seq: 1, Ack: 15, Len: 14	0030	58 36 9a 92 00 18 01 fd 58 5e 00 00 01 01 08 0a	....X6.....X^.....
> Data (14 bytes)	0040	4a b6 e6 55 5c c1 29 8c 53 65 72 76 65 72 48 65	..J..U..ServerHe
Data: 53657276657248656c66f3a360a	0050	6c 6c 6f 3a 36 0a	llo:6)
[Length: 14]			

W odpowiedzi na wiadomość ClientHello serwer przesyła wiadomość ServerHello. Wiadomość jest w postaci jawnej i zawiera klucz publiczny serwera. Następnie klient potwierdza odebranie wiadomości ServerHello pakietem ACK.

## 4. Wysłanie dowolnej wiadomości

8	22.165231	172.18.0.3	172.18.0.2	TCP	117	54554 → 4444 [PSH, ACK] Seq=15 Ack=15 Win=64256 Len=45 TSval=1556184993 TSecr=1253500501
9	22.206076	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [ACK] Seq=15 Ack=60 Win=65152 Len=0 TSval=1253522707 TSecr=1556184993

  

> Frame 8: Packet, 117 bytes on wire (936 bits), 117 bytes captured (936 bits)	0000	08 00 00 00 00 00 36 00 01 04 06 02 42 ac 12	.....6....B....
> Linux cooked capture v2	0010	00 02 00 00 45 00 00 42 bf 49 40 00 40 06 23 43	....E..B...I@#@C
> Internet Protocol Version 4, Src: 172.18.0.3, Dst: 172.18.0.2	0020	ac 12 00 03 ac 12 00 02 d5 1a 11 5c 58 36 9a 92	....E..B...c@eP)
> Transmission Control Protocol, Src Port: 54554, Dst Port: 4444, Seq: 15, Ack: 15, Len: 45	0030	79 b5 26 66 00 18 01 f6 58 7d 00 00 01 01 08 0a	....y&f....X).....
> Data (45 bytes)	0040	5c c1 7f a1 4a b6 e6 55 b7 f6 95 83 23 1c 12 13	..J..U..#.....
Data: b7f69583231c1213c73c6d10af834df6b945379026c70d07e20f5fefe69a2dbd13643b39feaa2ec511839841e2	0050	c7 3c 6d 10 af 83 4d f6 b9 45 37 90 26 c7 0d 07	..W..E7.6....
[Length: 45]	0060	e2 0f 5f ef e6 9a 2d bd 13 64 3b 39 fe aa 2e c5	..-..W..E7.6....
	0070	11 83 98 41 e2	....A.....d9....

EncryptedMessage:

b7f69583231c1213c73c6d10af834df6b945379026c70d07e20f5fefe69a2dbd13643b39feaa2ec511839841e2

Kluczy użyte podczas szyfrowanie:

OTP\_KEY=c493f6f146685f76b44f0c77ca88120c

MAC\_KEY=b8bc89f534fe69b6828827b974e68849

Do odszyfrowania wiadomości użyto prostego kodu (innego niż używa serwer podczas odszyfrowania wiadomości)

```
3 msg_hex = "b7f69583231c1213c73c6d10af834df6b945379026c70d07e20f5fefe69a2dbd13643b39feaa2ec511839841e2"
4 otp_hex = "c493f6f146685f76b44f0c77ca88120c"
5 mac_key_hex = "b8bc89f534fe69b6828827b974e68849"
6
7 msg = unhexlify(msg_hex)
8 otp_key = unhexlify(otp_hex)
9 mac_key = unhexlify(mac_key_hex)
10
11 ciphertext = msg[:-32]
12 mac = msg[-32:]
13
14 plaintext = bytes(c ^ otp_key[i % len(otp_key)] for i, c in
15                  enumerate(ciphertext))
16 print("PLAINTEXT:", plaintext.decode())
17
```

Wynik: PLAINTEXT: **secretMessage**

Odszyfrowana wiadomość przez serwera

```
[+] Server listening on port 4444
server> [+] Client connected: ('172.18.0.3', 54554)
[+] Client ID=1 from ('172.18.0.3', 54554)
[1] secretMessage
EndSession 1
[+] Sent EndSession to client 1
server> [-] Client 1 disconnected
```

## 5. Wysłanie wiadomości EndSession

10	54.984899	172.18.0.2	172.18.0.3	TCP	114	4444 → 54554 [PSH, ACK] Seq=15 Ack=60 Win=65152 Len=42 TSval=1253555406 TSecr=1556184993
11	54.985051	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=60 Ack=57 Win=64256 Len=0 TSval=1556217733 TSecr=1253555406

  

> Frame 10: Packet, 114 bytes on wire (912 bits), 114 bytes captured (912 bits)	0000	00 00 00 00 00 00 36 00 01 04 06 02 42 ac 12	.....6....B.....
> Linux cooked capture v2	0010	00 02 00 00 45 00 00 5e bf 4b 40 00 40 06 23 25	....E..K@e#%.....
> Internet Protocol Version 4, Src: 172.18.0.2, Dst: 172.18.0.3	0020	ac 12 00 02 ac 12 00 03 11 5c d5 1a 79 b5 26 66	....\..y.f.....
> Transmission Control Protocol, Src Port: 4444, Dst Port: 54554, Seq: 15, Ack: 60, Len: 42	0030	58 36 9a bf 80 10 01 fd 58 7a 00 00 01 01 08 0a	X8.....X2.....
> Data (42 bytes)	0040	4a b7 bc ce 5c c1 7f a1 c8 7c 68 97 86 85 82 2f	J.....[h].....
Data: cd7c6897f685822f0731d996fcfff086482b327888ff010460550c6cfd5e979a495689f27c8785ee441c	0050	07 31 d9 96 fc ff f0 86 48 2b 32 78 88 ff 01 04	!.....H2x.....
[Length: 42]	0060	60 55 0c 6c fd 5e 97 9a 49 56 89 f2 7c 87 85 ee	U.L.....IV.....
	0070	44 1c	D.....

Zakończenie sesji realizowane jest poprzez wysłanie przez serwer zaszyfrowanej wiadomości EndSession. Treść wiadomości nie jest czytelna, ponieważ podlega szyfrowaniu z wykorzystaniem klucza sesyjnego. Następnie klient potwierdza odbiór wiadomości pakietem ACK, co kończy komunikację na poziomie protokołu aplikacyjnego.

## 6. Zakończenie połączenia

Wariant 1: serwer wywołuje endSession

12	57.844635	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [FIN, ACK] Seq=60 Ack=57 Win=64256 Len=0 TSval=1556220672 TSecr=1253555406
13	57.844872	172.18.0.2	172.18.0.3	TCP	72	4444 → 54554 [FIN, ACK] Seq=57 Ack=61 Win=65152 Len=0 TSval=1253558346 TSecr=1556220672
14	57.844927	172.18.0.3	172.18.0.2	TCP	72	54554 → 4444 [ACK] Seq=61 Ack=58 Win=64256 Len=0 TSval=1556220673 TSecr=1253558346

  

> Frame 12: Packet, 72 bytes on wire (576 bits), 72 bytes captured (576 bits)	0000	00 00 00 00 00 00 36 00 01 04 06 02 42 ac 12	.....6....B.....
> Linux cooked capture v2	0010	00 03 00 00 45 00 00 3a 92 67 40 00 40 06 50 33	....E..4.g0@P3.....
> Internet Protocol Version 4, Src: 172.18.0.3, Dst: 172.18.0.2	0020	ac 12 00 03 ac 12 00 02 d5 1a 11 5c 58 36 9a bf	....\..V6.....
> Transmission Control Protocol, Src Port: 54554, Dst Port: 4444, Seq: 60, Ack: 57, Len: 0	0030	79 b5 26 90 00 11 01 f6 58 50 00 00 01 01 08 0a	y.f.....XP.....
	0040	5c c2 0b 00 0a b7 bc ce	\..J.....

Wariant 2: klient wywołuje endSession

9	7.358962	172.18.0.2	172.18.0.3	TCP	72	4444 → 44248 [FIN, ACK] Seq=16 Ack=58 Win=65152 Len=0 TSval=2934202968 TSecr=4054867350
10	7.359662	172.18.0.3	172.18.0.2	TCP	72	44248 → 4444 [FIN, ACK] Seq=58 Ack=17 Win=64256 Len=0 TSval=4054867353 TSecr=2934202968
11	7.359681	172.18.0.2	172.18.0.3	TCP	72	4444 → 44248 [ACK] Seq=17 Ack=59 Win=65152 Len=0 TSval=2934202969 TSecr=4054867353

  

> Frame 9: Packet, 72 bytes on wire (576 bits), 72 bytes captured (576 bits)	0000	00 00 00 00 00 00 18 00 01 04 06 02 42 ac 12	.....B.....
> Linux cooked capture v2	0010	00 02 00 00 45 00 00 3a a0 de 40 00 40 06 41 bc	....E..4...@A.....
> Internet Protocol Version 4, Src: 172.18.0.2, Dst: 172.18.0.3	0020	ac 12 00 02 ac 12 00 03 11 5c ac d8 e2 66 ac 0d	....\..f.....
> Transmission Control Protocol, Src Port: 4444, Dst Port: 44248, Seq: 16, Ack: 58, Len: 0	0030	55 02 05 9b 80 11 01 fd 58 50 00 00 01 01 08 0a	U.f.....XP.....
	0040	ae e4 62 50 f1 b0 5d 9e	..dx].....

Po zakończeniu komunikacji na poziomie protokołu aplikacyjnego następuje poprawne zamknięcie połączenia TCP. Klient/serwer inicjuje zakończenie sesji pakietem FIN, ACK, na co serwer/klient odpowiada własnym pakietem FIN, ACK. Ostatecznie klient/serwer potwierdza zamknięcie połączenia pakietem ACK, co odpowiada klasycznemu mechanizmowi four-way handshake protokołu TCP

## Opis użytych algorytmów

### Algorytm wymiany kluczy(Diffie–Hellman key exchange)

```
9  def dh_generate_private():
10     return int.from_bytes(os.urandom(2), "big")
11
12
13  def dh_generate_public(priv):
14     return pow([G, priv, P])
```

Generowanie kluczy

Klient:

```

# --- ClientHello ---
priv = dh_generate_private()
pub = dh_generate_public(priv)
s.sendall(f"ClientHello:{pub}\n".encode())

# --- ServerHello ---
data = b""
while not data.endswith(b"\n"):
    chunk = s.recv(1)
    if not chunk:
        print("[!] Connection closed by server")
        return
    data += chunk

```

Server:

```

# --- ClientHello ---
data = recv_line(conn)
if not data or not data.startswith("ClientHello"):
    conn.close()
    return

client_pub = int(data.split(":")[1])

# --- ServerHello ---
server_priv = dh_generate_private()
server_pub = dh_generate_public(server_priv)
conn.sendall(f"ServerHello:{server_pub}\n".encode())

shared = dh_compute_shared(client_pub, server_priv)
otp_key, mac_key = derive_keys(shared)

```

W projekcie zastosowano algorytm wymiany kluczy Diffie–Hellman. Klient i serwer generują własne klucze prywatne oraz odpowiadające im klucze publiczne, które są wymieniane w wiadomościach ClientHello i ServerHello. Następnie obie strony obliczają wspólny/dzielony klucz przy użyciu otrzymanego klucza publicznego i własnego klucza prywatnego. Uzyskany sekret wykorzystywany jest jako podstawa do wyprowadzenia kluczy sesyjnych używanych do szyfrowania danych oraz generowania MAC.

### Algorytm wyprowadzania kluczy (Key derivation)

```

def derive_keys(shared_secret: int):
    digest = hashlib.sha256(shared_secret.to_bytes(2, "big")).digest()
    otp = digest[:16]
    mac = digest[16:]
    return otp, mac

```

W celu uzyskania kluczy sesyjnych zastosowano uproszczony mechanizm wyprowadzania kluczy (Key Derivation Function). Funkcja `derive_keys` przyjmuje jako wejście wspólny klucz uzyskany w wyniku algorytmu Diffie–Hellman i deterministycznie wyprowadza z niego dwa klucze: klucz szyfrujący (OTP\_KEY) oraz klucz do generowania kodu MAC (MAC\_KEY).

### Algorytm szyfrowania danych (One-Time Pad (OTP) / XOR stream)

```
def otp_encrypt(key: bytes, plaintext: bytes, counter: int = 0) -> bytes:
    ciphertext = bytearray()
    key_len = len(key)
    for i, b in enumerate(plaintext):
        k = key[(i + counter) % key_len]
        ciphertext.append(b ^ k)
    return bytes(ciphertext)

def otp_decrypt(key: bytes, ciphertext: bytes, counter: int = 0) -> bytes:
    return otp_encrypt(key, ciphertext, counter)
```

Do szyfrowania danych zastosowano algorytm One-Time Pad w postaci strumienia XOR. Szyfrowanie polega na wykonaniu operacji XOR pomiędzy kolejnymi bajtami wiadomości a bajtami klucza sesyjnego (OTP\_KEY). W przypadku gdy długość wiadomości przekracza długość klucza, klucz jest powtarzany cyklicznie do długości danych.

Ten sam mechanizm wykorzystywany jest zarówno do szyfrowania, jak i do odszyfrowywania wiadomości, ponieważ operacja XOR jest odwracalna. Klucz szyfrujący wykorzystywany w algorytmie OTP pochodzi z procesu wymiany kluczy Diffie–Hellman i jest wspólny dla obu stron sesji.

#### Schemat zabezpieczenia wiadomości (Encrypt-then-MAC)

```
41 def encrypt_then_mac(
42     otp_key: bytes, mac_key: bytes, plaintext: bytes, counter: int = 0
43 ) -> bytes:
44     ciphertext = otp_encrypt(otp_key, plaintext, counter)
45     mac = hmac.new(mac_key, ciphertext, hashlib.sha256).digest()
46     return ciphertext + mac
```

W projekcie zastosowano schemat zabezpieczenia wiadomości typu Encrypt-then-MAC. Oznacza to, że treść wiadomości jest w pierwszej kolejności szyfrowana przy użyciu algorytmu OTP, a następnie dla powstałego ciphertextu obliczany jest kod MAC z wykorzystaniem algorytmu HMAC-SHA256.

Po stronie odbiorcy najpierw weryfikowana jest poprawność kodu MAC, a dopiero po jej pozytywnym zakończeniu następuje odszyfrowanie wiadomości.

#### Napotkane problemy

Podczas realizacji projektu pojawiły się trudności związane z poprawnym zrozumieniem kolejności komunikatów protokołu oraz rozróżnieniem wiadomości aplikacyjnych od pakietów sterujących TCP, takich jak ACK. Wyzwaniem była również analiza ruchu sieciowego w Wiresharku, szczególnie w środowisku Docker, gdzie pakiety pochodzą z wielu interfejsów. Dodatkowo problematyczne okazało się ręczne odszyfrowywanie przechwyconych danych i prawidłowe wydzielenie ciphertextu oraz MAC. Wszystkie te trudności zostały rozwiązane poprzez testy oraz stopniową analizę przechwyconych pakietów.

## Wnioski

Zrealizowany projekt potwierdza możliwość implementacji uproszczonego, szyfrowanego protokołu komunikacyjnego opartego na TCP, zgodnie z założeniami zadania. Zastosowanie algorytmu Diffie–Hellman umożliwiło bezpieczne uzgodnienie klucza sesyjnego, a schemat encrypt-then-MAC zapewnił integralność i autentyczność przesyłanych danych. Analiza ruchu sieciowego wykazała, że treść zaszyfrowanych wiadomości nie jest możliwa do odczytania bez znajomości klucza sesyjnego. Projekt pozwolił na praktyczne zrozumienie mechanizmów stosowanych w protokołach bezpieczeństwa, takich jak TLS