PSI - projekt

Adam Czupryński, Michał Sadlej, Szymon Makuch

Szyfrowany protokół oparty na protokole TCP, tzw. mini TLS. Zaimplementowaliśmy wariant W2 (MAC-then-Encrypt) - w języku Python.

Struktura

Struktury wiadomości składa się z następujących pól:

ClientHello

- typ wiadomości
- długość wiadomości
- `g` - generator do algorytmu Diffie-Hellman
- `p` - liczba pierwsza do algorytmu Diffie-Hellman
- `A` - klucz publiczny klienta

ServerHello

- typ wiadomości
- długość wiadomości
- `B` - klucz publiczny serwera

Szyfrowana wiadomość

- typ wiadomości
- długość wiadomości
- `iv` - wektor inicjalizacyjny
- zaszyfrowana wiadomość
- MAC

Szyfrowane EndSession

- typ wiadomości
- długość wiadomości
- powód zakończenia sesji

Wykorzystane algorytmy

Wymiana kluczy

Zastosowaliśmy algorytm wymiany kluczy Diffie-Hellman. Przebieg wymiany:

1. Klient wybiera losową liczbę pierwszą `g` (generator) i liczbę pierwszą `p`
2. Klient generuje prywatny klucz `a`
3. Klient oblicza wartość publiczną: `A = (g^a) mod p`
4. Serwer generuje prywatny klucz `b`

5. Serwer oblicza wartość publiczną: `B = (g^b) mod p`
6. Obie strony wspólnie obliczają klucz symetryczny: `K = (B^a) mod p = (A^b) mod p`

Szyfrowanie

Zastosowaliśmy prosty algorytm OTP (One-Time Pad):

- Każda wiadomość będzie szyfrowana przy użyciu jednorazowego klucza
- Klucz będzie generowany losowo dla każdej transmisji
- Szyfrowanie polega na operacji XOR między wiadomością a kluczem

Proces generowania klucza dla OTP:

- Initialization vector użyte jest jako pierszy blok
- W pętli generowane są kolejne bloki przy pomocy HMAC-SHA256 poprzez wyliczenie HMAC z poprzedniego bloku używając klucza sesji
- Po przekroczeniu zadanej długości pad jest przycinany do zadanej długości

Scenariusz przykładowy

1. Inicjacja połączenia i wymiana kluczy

1. Klient generuje liczbę pierwszą `p`, generator `g` i prywatny klucz `a` oraz oblicza `A = g^a mod p`
2. Klient wysyła ClientHello, w którym znajduje się `g`, `p` oraz `A`
3. Serwer generuje prywatny klucz `b` oraz oblicza `B = g^b mod p`
4. Serwer wysyła ServerHello, w którym znajduje się `B`
5. Obie strony obliczają wspólny klucz `K = A^b mod p = B^a mod p`

2. Wysyłanie zaszyfrowanej wiadomości

1. Klient przygotowuje wiadomość `M`
2. Oblicza `MAC = HMAC(M, K)`
3. Łączy wiadomość z MAC: `data = M || MAC`
4. Generuje losowy `IV`
5. Szyfruje `data` używając `K` i `IV`: `encrypted = encrypt(data, K, IV)`
6. Wysyła EncryptedMessage, zawierającą `IV` i `encrypted`

3. Odbieranie zaszyfrowanej wiadomości

1. Serwer odbiera EncryptedMessage
2. Deszyfruje dane używając `K` i `IV`: `decrypted = decrypt(encrypted_data, K, IV)`
3. Rozdziela odszyfrowane dane na wiadomość M i MAC
4. Oblicza własny `MAC' = HMAC(M, K)`
5. Porównuje `MAC'` z otrzymanym `MAC`
6. Jeśli `MAC' = MAC`, wiadomość jest poprawna

4. Zakończenie sesji

Dowolna ze stron może zakończyć sesję wysyłając EndSession

Realizacja mechanizmu integralności i autentyczności - MAC-then-Encrypt

Mechanizm MAC-then-encrypt ma prostszą implementację i mniejszą złożoność algorytmiczną niż Encrypt-then-MAC, jednocześnie zachowując mechanizmy integralności. Jednocześnie ma nieco niższy poziom bezpieczeństwa, a także wymaga pełnego odszyfrowania przed weryfikacją MAC.

Przebieg procesu

1. Dla oryginalnej wiadomości generowany jest MAC przez hashowanie przy użyciu wspólnego klucza
2. Oryginalna wiadomość wraz z wygenerowanym MAC jest szyfrowana
3. Weryfikacja i deszyfrowanie po stronie odbiorcy:
    - Odszyfrowanie całej wiadomości (wiadomość + MAC)
    - Ponowne wygenerowanie MAC z odebranej wiadomości
    - Porównanie wygenerowanego MAC z odebranym

Działanie programu

Przedstawione logi pokazują komunikację sieciową między serwerem a trzema klientami.

Konfiguracja systemu

- Serwer nasłuchuje na porcie 12345
- Trzej klienci (client1, client2, client3) próbują się połączyć
- Używane są adresy IP w sieci 172.23.0.x

```
z34_server   | [2025-01-17 21:19:06,711] INFO: Server started on
0.0.0.0:12345
z34_client3  | Client server:12345 started
z34_client1  | Client server:12345 started
z34_client2  | Client server:12345 started
z34_client1  | Client> help
z34_client1  | Available commands:
z34_client1  |   connect - Connect to server
z34_client1  |   disconnect - Disconnect from server
z34_client1  |   send <message> - Send encrypted message
z34_client1  |   help - Show this help
z34_client1  |   exit - Exit client
z34_client1  | Server> list
z34_client1  | No connected clients
```

Przebieg komunikacji

- Najpierw wszyscy klienci uruchamiają się
- Client1 łączy się jako pierwszy (z IP 172.23.0.5)
- Następnie łączy się client2 (IP 172.23.0.4)
- Na końcu łączy się client3 (IP 172.23.0.3)
- Każde połączenie inicjuje wymianę kluczy ("Key exchange completed")

```
z34_client1  | Client> help
z34_client1  | Available commands:
z34_client1  |   connect - Connect to server
z34_client1  |   disconnect - Disconnect from server
z34_client1  |   send <message> - Send encrypted message
z34_client1  |   help - Show this help
z34_client1  |   exit - Exit client
z34_client1  | Client> send test
z34_client1  | Not connected
z34_client1  | Client> connect
z34_client1  | [2025-01-17 21:21:08,806] INFO: Connected to server:12345
z34_server   | Server> [2025-01-17 21:21:08,806] INFO: New connection from
('172.21.0.5', 46904)
z34_server   | [2025-01-17 21:21:08,807] INFO: Key exchange completed with
Client(('172.21.0.5', 46904))
z34_client1  | Client> [2025-01-17 21:21:08,807] INFO: Key exchange
completed
z34_server   | list
z34_server   | 1. Client(('172.21.0.5', 46904))
z34_client2  | Client> connect
z34_server   | Server> [2025-01-17 21:21:17,485] INFO: New connection from
('172.21.0.4', 34398)
z34_server   | [2025-01-17 21:21:17,485] INFO: Key exchange completed with
Client(('172.21.0.4', 34398))
z34_client2  | [2025-01-17 21:21:17,485] INFO: Connected to server:12345
z34_client2  | Client> [2025-01-17 21:21:17,485] INFO: Key exchange
completed
z34_client3  | Client> connect
z34_server   | [2025-01-17 21:21:20,108] INFO: New connection from
('172.21.0.3', 49610)
z34_client3  | [2025-01-17 21:21:20,109] INFO: Connected to server:12345
z34_server   | [2025-01-17 21:21:20,109] INFO: Key exchange completed with
Client(('172.21.0.3', 49610))
z34_client3  | Client> [2025-01-17 21:21:20,109] INFO: Key exchange
completed
z34_server   | list
z34_server   | 1. Client(('172.21.0.5', 46904))
z34_server   | 2. Client(('172.21.0.4', 34398))
z34_server   | 3. Client(('172.21.0.3', 49610))
```

Interakcje

- Klienci wysyłają testowe wiadomości
- Client1 rozłącza się samodzielnie
- Server odpina client2 komendą "disconnect"
- Server kończy działanie, co powoduje rozłączenie client3

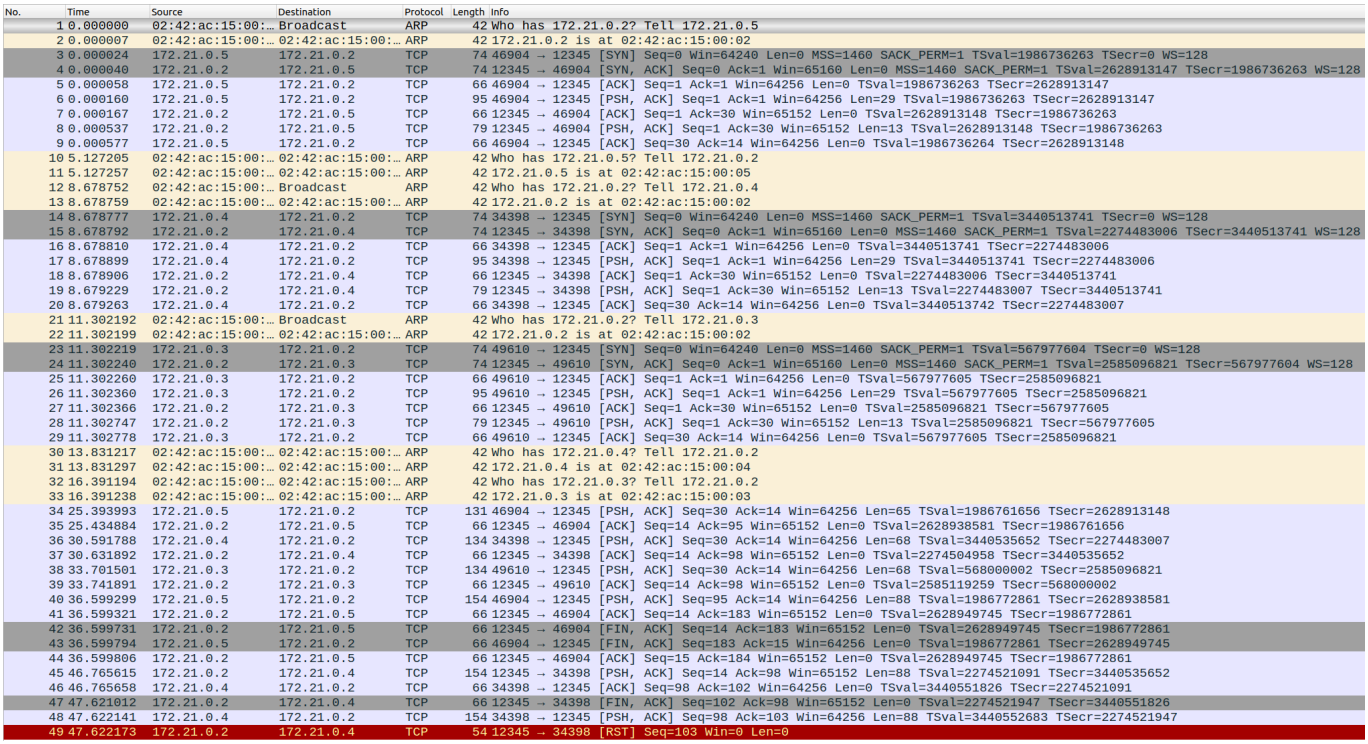
```
z34_client1  | send test
z34_server   | Server> [2025-01-17 21:21:34,200] INFO: Message from
Client(('172.21.0.5', 46904)): test
```

```
z34_client2  | send client2
z34_server   | [2025-01-17 21:21:39,398] INFO: Message from
Client(('172.21.0.4', 34398)): client2
z34_client3  | send client3
z34_server   | [2025-01-17 21:21:42,508] INFO: Message from
Client(('172.21.0.3', 49610)): client3
z34_client1  | Client> disconnect
z34_server   | [2025-01-17 21:21:45,405] INFO: Received EndSession from
Client(('172.21.0.5', 46904)): Client initiated disconnect
z34_server   | [2025-01-17 21:21:45,406] INFO: Disconnecting
Client(('172.21.0.5', 46904))
z34_client1  | [2025-01-17 21:21:45,406] ERROR: Connection closed by server
z34_client1  | [2025-01-17 21:21:45,406] WARNING: Not connected
z34_client1  | client receive loop ended
z34_client1  | [2025-01-17 21:21:45,406] INFO: Client disconnected
z34_server   | list
z34_server   | 1. Client(('172.21.0.4', 34398))
z34_server   | 2. Client(('172.21.0.3', 49610))
z34_server   | Server> disconnect 1
z34_server   | [2025-01-17 21:21:55,572] INFO: Disconnecting
Client(('172.21.0.4', 34398))
z34_client2  | Client> Received message: Server initiated disconnect
z34_client2  | [2025-01-17 21:21:56,427] ERROR: Connection closed by server
z34_client2  | [2025-01-17 21:21:56,428] INFO: Client disconnected
z34_client2  | client receive loop ended
z34_server   | Server> exit
z34_server   | [2025-01-17 21:22:28,495] INFO: Disconnecting
Client(('172.21.0.3', 49610))
z34_client3  | Client> [2025-01-17 21:22:28,495] ERROR: Connection closed
by server
z34_client3  | [2025-01-17 21:22:28,496] INFO: Client disconnected
z34_client3  | client receive loop ended
z34_server exited with code 0
```

Wireshark

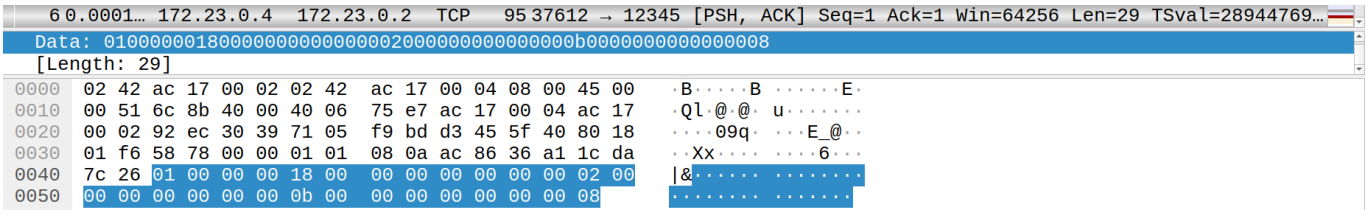
Poniższy zrzut ekranu pokazuje:

- Komunikację TCP między adresami 172.23.0.x
- Pakiety ARP służące do rozpoznawania adresów
- Wymianę pakietów SYN podczas nawiązywania połączeń
- Pakiety PSH+ACK przy przesyłaniu danych
- Różne długości pakietów wskazujące na szyfrowaną komunikację



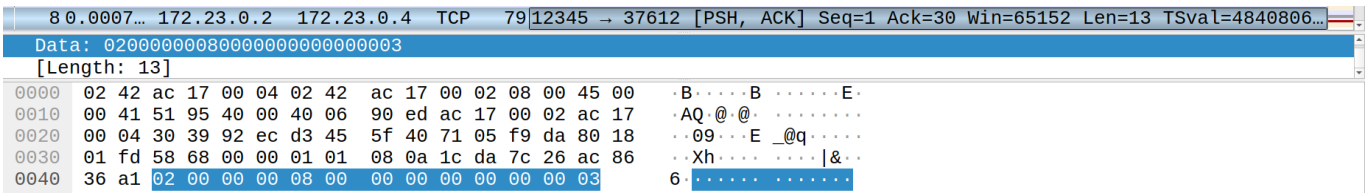
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	02:42:ac:15:00:…	Broadcast	ARP	42	Who has 172.21.0.2? Tell 172.21.0.5
2	0.000007	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	172.21.0.2 is at 02:42:ac:15:00:02
3	0.000024	172.21.0.5	172.21.0.2	TCP	74	46904 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=1986736263 TSecr=0 WS=128
4	0.000040	172.21.0.2	172.21.0.5	TCP	74	12345 → 46904 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2628913147 TSecr=1986736263 WS=128
5	0.000058	172.21.0.5	172.21.0.2	TCP	66	46904 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1986736263 TSecr=2628913147
6	0.000160	172.21.0.5	172.21.0.2	TCP	95	46904 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=29 TSval=1986736263 TSecr=2628913147
7	0.000167	172.21.0.2	172.21.0.5	TCP	66	12345 → 46904 [ACK] Seq=1 Ack=30 Win=65152 Len=0 TSval=2628913148 TSecr=1986736263
8	0.000537	172.21.0.2	172.21.0.5	TCP	79	12345 → 46904 [PSH, ACK] Seq=1 Ack=30 Win=65152 Len=13 TSval=2628913148 TSecr=1986736263
9	0.000577	172.21.0.5	172.21.0.2	TCP	66	46904 → 12345 [ACK] Seq=30 Ack=14 Win=64256 Len=0 TSval=1986736264 TSecr=2628913148
10	5.127205	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	Who has 172.21.0.5? Tell 172.21.0.2
11	5.127257	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	172.21.0.5 is at 02:42:ac:15:00:05
12	8.678752	02:42:ac:15:00:…	Broadcast	ARP	42	Who has 172.21.0.2? Tell 172.21.0.4
13	8.678759	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	172.21.0.2 is at 02:42:ac:15:00:02
14	8.678777	172.21.0.4	172.21.0.2	TCP	74	34398 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3440513741 TSecr=0 WS=128
15	8.678792	172.21.0.2	172.21.0.4	TCP	74	12345 → 34398 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2274483006 TSecr=3440513741 WS=128
16	8.678810	172.21.0.4	172.21.0.2	TCP	66	34398 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3440513741 TSecr=2274483006
17	8.678899	172.21.0.4	172.21.0.2	TCP	95	34398 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=29 TSval=3440513741 TSecr=2274483006
18	8.678906	172.21.0.2	172.21.0.4	TCP	66	12345 → 34398 [ACK] Seq=1 Ack=30 Win=65152 Len=0 TSval=2274483006 TSecr=3440513741
19	8.679229	172.21.0.2	172.21.0.4	TCP	79	12345 → 34398 [PSH, ACK] Seq=1 Ack=30 Win=65152 Len=13 TSval=2274483007 TSecr=3440513741
20	8.679263	172.21.0.4	172.21.0.2	TCP	66	34398 → 12345 [ACK] Seq=30 Ack=14 Win=64256 Len=0 TSval=3440513742 TSecr=2274483007
21	11.302192	02:42:ac:15:00:…	Broadcast	ARP	42	Who has 172.21.0.2? Tell 172.21.0.3
22	11.302199	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	172.21.0.2 is at 02:42:ac:15:00:02
23	11.302219	172.21.0.3	172.21.0.2	TCP	74	49610 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=567977604 TSecr=0 WS=128
24	11.302240	172.21.0.2	172.21.0.3	TCP	74	12345 → 49610 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=2585096821 TSecr=567977604 WS=128
25	11.302260	172.21.0.3	172.21.0.2	TCP	66	49610 → 12345 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=567977605 TSecr=2585096821
26	11.302360	172.21.0.3	172.21.0.2	TCP	95	49610 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=29 TSval=567977605 TSecr=2585096821
27	11.302366	172.21.0.2	172.21.0.3	TCP	66	12345 → 49610 [ACK] Seq=1 Ack=30 Win=65152 Len=0 TSval=2585096821 TSecr=567977605
28	11.302747	172.21.0.2	172.21.0.3	TCP	79	12345 → 49610 [PSH, ACK] Seq=1 Ack=30 Win=65152 Len=13 TSval=2585096821 TSecr=567977605
29	11.302778	172.21.0.3	172.21.0.2	TCP	66	49610 → 12345 [ACK] Seq=30 Ack=14 Win=64256 Len=0 TSval=567977605 TSecr=2585096821
30	13.831217	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	Who has 172.21.0.4? Tell 172.21.0.2
31	13.831297	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	172.21.0.4 is at 02:42:ac:15:00:04
32	16.391194	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	Who has 172.21.0.3? Tell 172.21.0.2
33	16.391238	02:42:ac:15:00:…	02:42:ac:15:00:…	ARP	42	172.21.0.3 is at 02:42:ac:15:00:03
34	25.393993	172.21.0.5	172.21.0.2	TCP	131	46904 → 12345 [PSH, ACK] Seq=30 Ack=14 Win=64256 Len=65 TSval=1986761656 TSecr=2628913148
35	25.434884	172.21.0.2	172.21.0.5	TCP	66	12345 → 46904 [ACK] Seq=14 Ack=95 Win=65152 Len=0 TSval=2628938581 TSecr=1986761656
36	30.591788	172.21.0.4	172.21.0.2	TCP	134	34398 → 12345 [PSH, ACK] Seq=30 Ack=14 Win=64256 Len=68 TSval=3440535652 TSecr=2274483007
37	30.631892	172.21.0.2	172.21.0.4	TCP	66	12345 → 34398 [ACK] Seq=14 Ack=98 Win=65152 Len=0 TSval=2274504958 TSecr=3440535652
38	33.701501	172.21.0.3	172.21.0.2	TCP	134	49610 → 12345 [PSH, ACK] Seq=30 Ack=14 Win=64256 Len=68 TSval=568000002 TSecr=2585096821
39	33.741891	172.21.0.2	172.21.0.3	TCP	66	12345 → 49610 [ACK] Seq=14 Ack=98 Win=65152 Len=0 TSval=2585119259 TSecr=568000002
40	36.599299	172.21.0.5	172.21.0.2	TCP	154	46904 → 12345 [PSH, ACK] Seq=95 Ack=14 Win=64256 Len=88 TSval=1986772861 TSecr=2628938581
41	36.599321	172.21.0.2	172.21.0.5	TCP	66	12345 → 46904 [ACK] Seq=14 Ack=183 Win=65152 Len=0 TSval=2628949745 TSecr=1986772861
42	36.599731	172.21.0.2	172.21.0.5	TCP	66	12345 → 46904 [FIN, ACK] Seq=14 Ack=183 Win=65152 Len=0 TSval=2628949745 TSecr=1986772861
43	36.599794	172.21.0.5	172.21.0.2	TCP	66	46904 → 12345 [FIN, ACK] Seq=183 Ack=15 Win=64256 Len=0 TSval=1986772861 TSecr=2628949745
44	36.599806	172.21.0.2	172.21.0.5	TCP	66	12345 → 46904 [ACK] Seq=15 Ack=184 Win=65152 Len=0 TSval=2628949745 TSecr=1986772861
45	46.765615	172.21.0.2	172.21.0.4	TCP	154	12345 → 34398 [PSH, ACK] Seq=14 Ack=98 Win=65152 Len=88 TSval=2274521091 TSecr=3440535652
46	46.765658	172.21.0.4	172.21.0.2	TCP	66	34398 → 12345 [ACK] Seq=98 Ack=102 Win=64256 Len=0 TSval=3440551826 TSecr=2274521091
47	47.621012	172.21.0.2	172.21.0.4	TCP	66	12345 → 34398 [FIN, ACK] Seq=102 Ack=98 Win=65152 Len=0 TSval=2274521947 TSecr=3440551826
48	47.622141	172.21.0.4	172.21.0.2	TCP	154	34398 → 12345 [PSH, ACK] Seq=98 Ack=103 Win=64256 Len=88 TSval=3440552683 TSecr=2274521947
49	47.622173	172.21.0.2	172.21.0.4	TCP	54	12345 → 34398 [RST] Seq=103 Win=0 Len=0

Znając przyjętą przez nas strukturę wiadomości możemy zobaczyć co znajduje się w pakietach. Struktura każdej wiadomości to: typ, długość, ładunek.



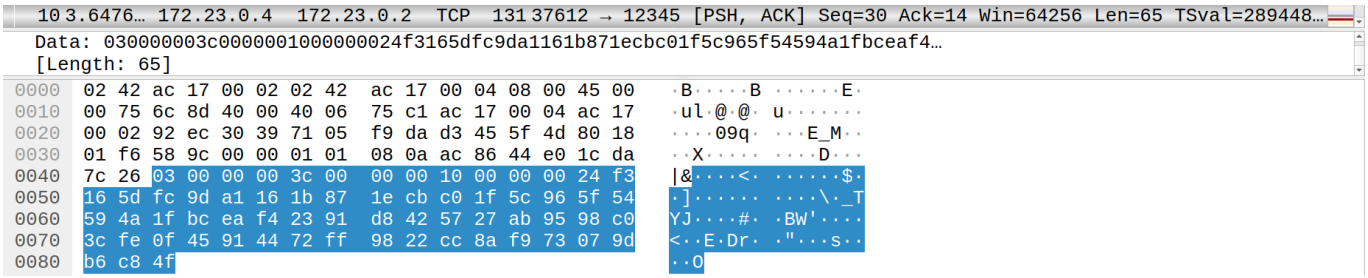
```
  6 0.0001… 172.23.0.4  172.23.0.2  TCP   95 37612 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=29 TSval=28944769…
Data: 0100000018000000000000000200000000000000b0000000000000008
[Length: 29]
0000  02 42 ac 17 00 02 02 42  ac 17 00 04 08 00 45 00   ·B·····B ······E·
0010  00 51 6c 8b 40 00 40 06  75 e7 ac 17 00 04 ac 17   ·Ql·@·@· u·······
0020  00 02 92 ec 30 39 71 05  f9 bd d3 45 5f 40 80 18   ····09q· ···E_@··
0030  01 f6 58 78 00 00 01 01  08 0a ac 86 36 a1 1c da   ··Xx···· ····6···
0040  7c 26 01 00 00 00 18 00  00 00 00 00 00 00 02 00   |&····· ········
0050  00 00 00 00 00 00 0b 00  00 00 00 00 00 00 08      ········ ·······
```

dane: 0100000018000000000000000200000000000000b0000000000000008
01000000 (1): typ wiadomości - ClientHello
1800000000000000 (24): długość
0200000000000000 (2): `g` - generator
0b00000000000000 (11): `p` - liczba pierwsza
08 (8): `A` - klucz publiczny klienta

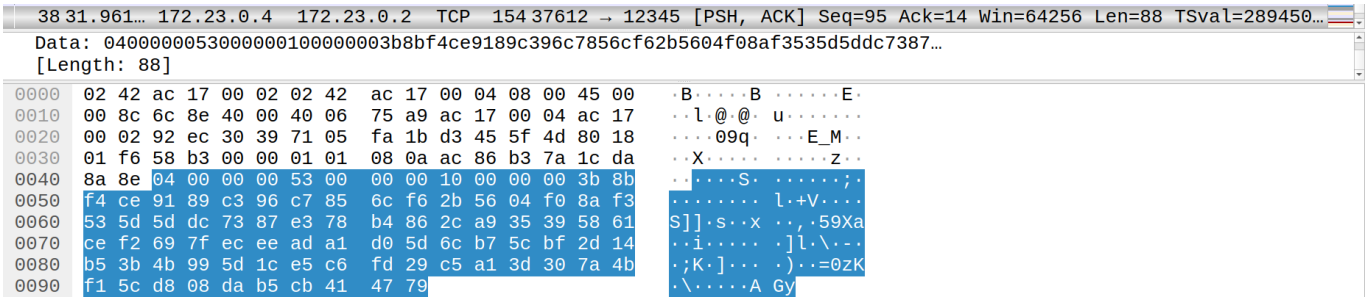


```
  8 0.0007… 172.23.0.2  172.23.0.4  TCP   79 12345 → 37612 [PSH, ACK] Seq=1 Ack=30 Win=65152 Len=13 TSval=4840806…
Data: 02000000080000000000000003
[Length: 13]
0000  02 42 ac 17 00 04 02 42  ac 17 00 02 08 00 45 00   ·B·····B ······E·
0010  00 41 51 95 40 00 40 06  90 ed ac 17 00 02 ac 17   ·AQ·@·@· ········
0020  00 04 30 39 92 ec d3 45  5f 40 71 05 f9 da 80 18   ··09···E _@q·····
0030  01 fd 58 68 00 00 01 01  08 0a 1c da 7c 26 ac 86   ··Xh···· ····|&··
0040  36 a1 02 00 00 00 08 00  00 00 00 00 00 00 03      6····· ·······
```

dane: 02000000080000000000000003
02000000 (2): typ wiadomości - ServerHello
0800000000000000 (8): długość
03 (3): `B` - klucz publiczny serwera



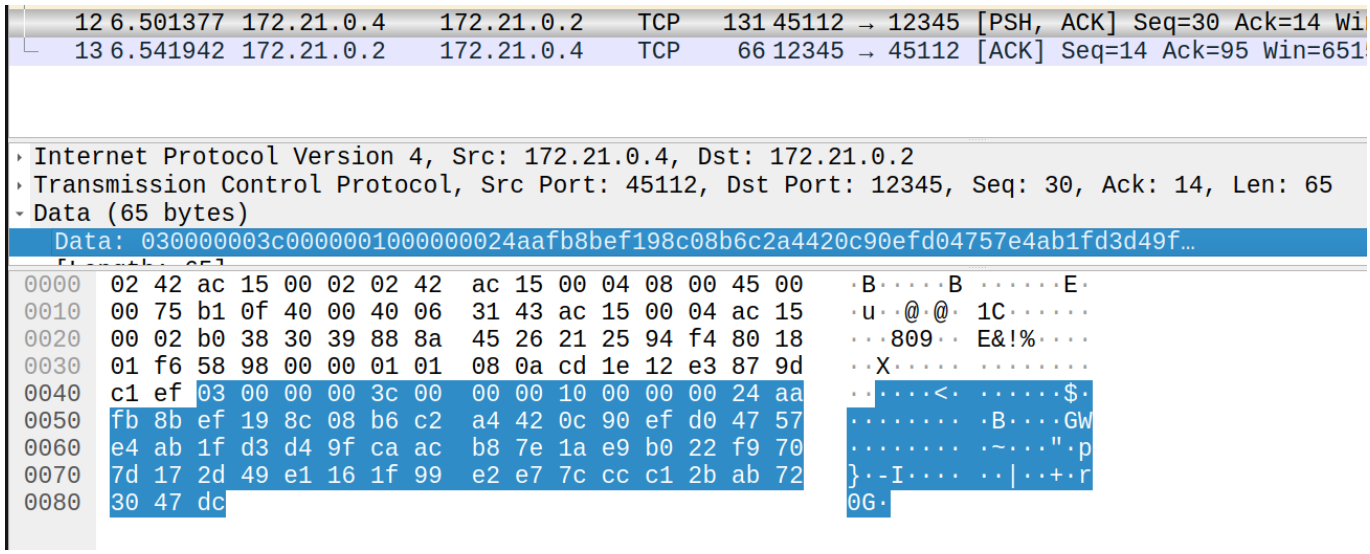

```
dane:
030000003c0000001000000024f3165dfc9da1161b871ecbc01f5c965f54594a1fbceaf4239
1d8425727ab9598c03cfe0f45914472ff9822cc8af973079db6c84f
03000000 (3): typ wiadomości - EncryptedMessage
3c00000010000000 (60): długość
24f3165dfc9da1161b871ecbc01f5c965f54594a1fbceaf42391d8425727ab9598c03cfe0f4
5914472ff9822cc8af973079db6c84f - zaszyfrowana wiadomość
```




```
dane:
040000005300000010000000 3b8bf4ce9189c396c7856cf62b5604f08af3535d5ddc7387e37
8b4862ca935395861cef2697feceeada1d05d6cb75cbf2d14b53b4b995d1ce5c6fd29c5a13d
307a4bf15cd808dab5cb414779
04000000 (4): typ wiadomości - EndSession
5300000010000000 (83): długość
3b8bf4ce9189c396c7856cf62b5604f08af3535d5ddc7387e378b4862ca935395861cef2697
feceeada1d05d6cb75cbf2d14b53b4b995d1ce5c6fd29c5a13d307a4bf15cd808dab5cb4147
79 - zaszyfrowany powód zakończenia sesji
```

Odszyfrowywanie wiadomości

Skrypt `encryption_test.py` służy odszyfrowywaniu wiadomości w postaci ciągu szesnatkowego, znając klucz szyfrujący - zapisany podczas tworzenia klienta



Skrypt poprawnie odszyfrował ciąg `aafb8bef198c08b6c2a4420c90efd04757e4ab1fd3d49fcaacb87e1ae9b022f9707d172d49e1161f99e2e77cccc12bab723047dc` jako napis "test".