

Sprawozdanie końcowe

Sofiya Yedzeika

Matvii Ivashchenko

Katsyaryna Anikevich

16 stycznia 2026

Spis treści

1	Wstęp	3
2	Sposób uruchomienia projektu i parametry działania	3
3	Dowód poprawnego działania protokołu	4
3.1	Sekwencja komunikatów protokołu	4
3.2	Ręczne odszyfrowanie przechwyconych danych	4
4	Opis użytych algorytmów	5
4.1	Wymiana kluczy Diffie–Hellman	5
4.2	Funkcja wyprowadzania kluczy (KDF)	6
4.3	Szyfrowanie i integralność danych	6
5	Napotkane problemy	6
6	Wnioski	6

1 Wstęp

Celem niniejszego sprawozdania jest udokumentowanie poprawnego działania zaprojektowanego oraz zaimplementowanego protokołu komunikacyjnego, opartego na protokole TCP. Projekt został zrealizowany zgodnie z założeniami przedstawionymi w treści zadania i obejmuje architekturę klient-serwer, mechanizm bezpiecznej wymiany kluczy, szyfrowanie przesyłanych danych oraz obsługę wielu klientów jednocześnie.

W ramach pracy przeprowadzono testy funkcjonalne oraz analityczne, których celem było potwierdzenie spełnienia wszystkich wymagań projektowych.

2 Sposób uruchomienia projektu i parametry działania

Projekt został przygotowany w formie aplikacji klienckiej i serwerowej, uruchamianej w środowisku kontenerowym Docker. Całość została zaprojektowana w taki sposób, aby możliwe było szybkie uruchomienie systemu przy minimalnej liczbie poleceń.

Serwer uruchamiany jest w osobnym kontenerze Docker i nasłuchuje połączeń TCP na określonym porcie (domyślnie 4444). Podczas uruchamiania serwera możliwe jest przekazanie parametru określającego maksymalną liczbę jednocześnie obsługiwanych klientów. Parametr ten zapobiega przeciążeniu serwera oraz umożliwia zaprezentowanie obsługi wielu klientów jednocześnie.

Klient uruchamiany jest w osobnym kontenerze i może działać w dwóch trybach: interaktywnym oraz testowym (wieloklientowym). W trybie interaktywnym użytkownik może ręcznie zainicjować połączenie z serwerem, wysłać wiadomości tekstowe oraz zakończyć sesję przy użyciu odpowiedniej komendy. Każda sesja rozpoczyna się wysłaniem wiadomości `ClientHello`, a po jej zakończeniu możliwe jest ponowne nawiązanie połączenia poprzez ponowne wysłanie tej wiadomości.

Tryb testowy umożliwia automatyczne uruchomienie wielu klientów jednocześnie, co pozwala na demonstrację wielowątkowej obsługi połączeń po stronie serwera. Każdy klient w tym trybie wykonuje pełny proces nawiązania połączenia, wymiany kluczy oraz wysłania zaszyfrowanej wiadomości, po czym kończy sesję. Dzięki temu możliwe jest zaobserwowanie równoległego przetwarzania połączeń oraz poprawności działania protokołu w warunkach współbieżności. Zmiana wartości zmiennej `MINITLS_DEBUG` w skrypcie `run.sh` umożliwia kontrolę nad dodatkowymi wydrukami w konsoli.

Przykłady poleceń:

```
./run.sh – wartości domyślne: tryb klienta, maksymalna liczba klientów – 5;  
./run.sh client 1 – tryb klienta, maksymalna liczba klientów – 1;  
./run.sh multi 7 – tryb testowy wieloklientowy, liczba klientów – 7.
```


Następnie, na podstawie przechwyconych danych z Wireshark oraz zapisanego klucza, przeprowadzono ręczne odszyfrowanie wybranego komunikatu. Uzyskany tekst jawny był zgodny z oryginalną treścią wysłaną przez klienta, co potwierdza poprawność implementacji algorytmów kryptograficznych.

```
[i] Starting interactive client
Connected to z33_server:4444
>> {"dh_pub": 66857915230131239036477633331409766135810672056538512411343722650114148199330924016174186295041228244787943052071025997795719168222728931141
0027180740483737506298869402389443333735972205369983212948808476338004122784694528333817292048875019820519885318835189786564230692231238789811338637677480
62795627757689922648582924095530146402237165272258856373208574834505599297322374478845387168203618586853617419130246527364517039328747054116907031199658400
62464190365185650330229145962123914763092409044224037612788781084038210522362731557679924133168772752454022716167866571913649509578420129399673753194878495
7021615843, "type": "ClientHello"}
<< {"dh_pub": 248666056092414655992117396809534250954659638047162684844176615050686574123408851028050240991335342741658499951175601245468054201720424287590
11630844756235893421196567310617889502997028255364051320234752162503932229077541621480483625007871112782063402492655300870685986917670818186256025495463827
54690073648268190444953973754460027957074217550749483191253925928458617439302953633282299600121666416118655599542403214957524263357321610666249465535508464
65252310254403494615036539080453178316650599084051988243327707054906678036950203708294995651595592070475321336449439399880367102454244243997900435785536846
76888683717, "type": "ServerHello"}
Session keys derived
Type messages. Use /quit to end.

> psi
<< {"ct": "f4Ea2igvlvqgy0ZP06QuUAPls92Tj1Kd1XNAPQ==", "mac": "JGB7UmiNAFI7GRRJX7taQRX20s/frQcLZgJzPo4h6pM=", "seq": 0, "type": "EncryptedData"}
<< {"ct": "3GmQ+XDKGgBIZJaYS7yGjRstprjOKV26abmBw==", "mac": "NXlItIdIs+60VBH+FsJPC4PcC84GlcqC9VrzeIAdnM=", "seq": 0, "type": "EncryptedData"}
Server: psi

> psi
<< {"ct": "RYipFiildwvL+H56r0wJmRjt+Afn1/BaoFc0A==", "mac": "yETJFCJkjuOUq3/J3NZehjeSdbtkQ5BDCNVkDqmI9E=", "seq": 1, "type": "EncryptedData"}
<< {"ct": "DFTSnbuDD7MUXb1HX9JdwyIdPzW0BK702Eehg==", "mac": "eQBwNfskF7ML2EZJdLBpfjgrQ9PB/h5tsw6wahjdFM=", "seq": 1, "type": "EncryptedData"}
Server: psi

> psi
<< {"ct": "wSunPeI3BQmKdnaywJ0tw3tvhiQ8mryJigfuw==", "mac": "VFG2zqM3LlHFRk84HbBX5AZ+Pc8+DmSqWsgs7Nk10=", "seq": 2, "type": "EncryptedData"}
<< {"ct": "OkT8Im7ilnu1w1cd4AK/SNBVcfvi4gyvDQ9L6A==", "mac": "2Fvff+1bLS2SEtuo6j9CPTAUAnZPULFuVnvffevvm0=", "seq": 2, "type": "EncryptedData"}
Server: psi
```

Rysunek 1: Danne wejściowe

```
5
6 ct_b64 = "f4Ea2igvlvqgy0ZP06QuUAPls92Tj1Kd1XNAPQ=="
7 seq = 0
8
9 with open("PSI_Z33/project/session_keys_client_1.bin", "rb") as f:
10     for line in f:
11         if line.startswith(b"c2s_enc="): # or c2s_enc
12             key = line[len(b"c2s_enc="):].strip()
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
mivashch@fedora:~/PW/PSI$ /bin/python /home/mivashch/PW/PSI/PSI_Z33/project/decrypt.py
PT HEX : 7b2274797065223a2244415441222c2274657874223a22707369227d
PT RAW : b'{"type":"DATA","text":"psi"}'
JSON: {'type': 'DATA', 'text': 'psi'}
```

Rysunek 2: Ręczne odszyfrowanie komunikatu przechwyconego w Wireshark

4 Opis użytych algorytmów

4.1 Wymiana kluczy Diffie–Hellman

Do bezpiecznej wymiany kluczy zastosowano algorytm Diffie–Hellman oparty na arytmetyce modularnej. Zarówno klient, jak i serwer generują parę kluczy (prywatny i publiczny), a następnie wymieniają klucze publiczne w ramach komunikatów ClientHello oraz ServerHello.

Na podstawie otrzymanego klucza publicznego drugiej strony oraz własnego klucza prywatnego obliczany jest wspólny sekret, który nie jest nigdy przesyłany przez sieć.

4.2 Funkcja wyprowadzania kluczy (KDF)

Ze wspólnego sekretu generowane są klucze sesyjne przy użyciu funkcji skrótu SHA-256. Zastosowano separację kluczy na potrzeby komunikacji w obu kierunkach (klient–serwer oraz serwer–klient), co zwiększa bezpieczeństwo protokołu.

4.3 Szyfrowanie i integralność danych

Do szyfrowania danych wykorzystano prosty szyfr strumieniowy oparty na operacji XOR z generowanym strumieniem klucza. Integralność i autentyczność komunikatów zapewnia mechanizm *encrypt-then-MAC*, w którym po zaszyfrowaniu danych obliczany jest znacznik HMAC.

Każdy komunikat zawiera numer sekwencyjny, co dla każdej wiadomości zmienia MAC i dodatkowo chroni protokół przed atakami typu replay.

5 Napotkane problemy

Podczas realizacji projektu napotkano szereg problemów technicznych, wśród których można wyróżnić:

- synchronizację numerów sekwencyjnych pomiędzy klientem a serwerem,
- poprawną obsługę wielu klientów jednocześnie przy użyciu wątków,
- problemy z przechwytywaniem ruchu sieciowego w środowisku Docker na systemie Linux,

Każdy z powyższych problemów został rozwiązany poprzez analizę logów, modyfikację kodu źródłowego oraz testy praktyczne.

6 Wnioski

Zrealizowany projekt spełnia wszystkie założenia określone w treści zadania. Opracowany protokół umożliwia bezpieczną komunikację klient–serwer, zapewnia poufność, integralność oraz podstawowy poziom autentyczności przesyłanych danych.