Sprawozdanie końcowe

Sofiya Yedzeika
Matvii Ivashchenko
Katsyaryna Anikevich

16 stycznia 2026

Spis treści

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 przesylanych 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, wysyłać 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.

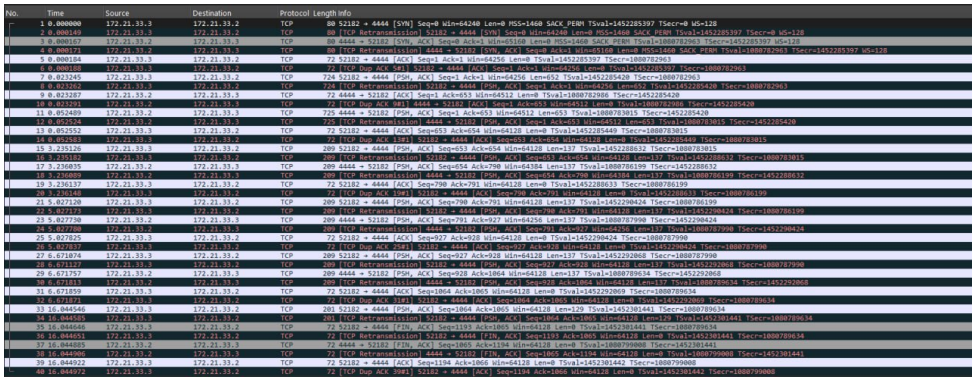
Przykłady poleceń:
./run.sh – wartości domyślne: tryb klienta, maksymalna liczba klientów – 5;
./run.sh client 1 – tryb klienta, maksymalna liczba klientów – 1;
./run.sh multi 7 – tryb testowy wieloklientowy, liczba klientów – 7.

3 Dowód poprawnego działania protokołu

3.1 Sekwencja komunikatów protokołu

Protokół realizuje następującą sekwencję komunikatów:

1. ClientHello – komunikat inicjujący połączenie, wysyłany przez klienta w postaci jawnej.
2. ServerHello – odpowiedź serwera zawierająca dane niezbędne do ustalenia wspólnego sekretu.
3. EncryptedData – szyfrowane komunikaty aplikacyjne przesyłane w trakcie trwania sesji.
4. EndSession – komunikat kończący sesje, po którym wymagane jest ponowne wykonanie procedury handshake.



3.2 Ręczne odszyfrowanie przechwyconych danych

W celu udowodnienia, że przesyłane komunikaty są rzeczywiście szyfrowane, a następnie poprawnie odszyfrowywane przez odbiorcę, klucz sesyjny został zapisany do pliku binarnego po zakończeniu procedury wymiany kluczy.

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

Rysunek 1: Danne wejściowe

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, wsród 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.