Programowanie sieciowe

Miłosz Cieśla, Filip Ryniewicz, Aleksander Szymczyk

Sprawozdanie wstępne z projektu

22.12.2024

1 Treść zadania

Treść projektu

Celem projektu jest zaprojektowanie oraz implementacja szyfrowanego protokołu opartego na protokole TCP, tzw. mini TLS.

Założenia projektu

- Architektura klient-serwer.
- Serwer jest w stanie obsłużyć kilku klientów jednocześnie (liczba klientów powinna być podawana jako parametr uruchomienia, nie hardcodowana).
- Klient inicjuje połączenie z serwerem poprzez wysłanie wiadomości **ClientHello** (wiadomość nieszyfrowana), na którą serwer odpowiada wiadomością **ServerHello** (wiadomość nieszyfrowana).
- Sesja może zostać zakończona zarówno przez klienta, jak i przez serwer poprzez wysłanie wiadomości **EndSession**. Po odebraniu **EndSession** należy ponownie wysłać **ClientHello**, aby rozpocząć nową sesję.
- Wszystkie wiadomości poza **ClientHello** i **ServerHello** muszą być szyfrowane.
- Potencjalny napastnik po przechwyceniu wiadomości nie powinien być w stanie nic z niej odczytać.

Wymagania implementacyjne

- Wiadomości **ClientHello** i **ServerHello** służą do wymiany kluczy szyfrujących. Po ich wymianie klient oraz serwer muszą być w posiadaniu tego samego klucza szyfrującego, który będzie używany do szyfrowania kolejnych wiadomości.
- Użycie algorytmu wymiany kluczy, np. **Diffie-Hellman key exchange**. Samo szyfrowanie wiadomości może (ale nie musi) być zrealizowane prostym **OTP** (ang. One Time Pad).
- Komunikacja ma być sterowana z wiersza poleceń. Dostępne opcje:
  - Serwer:
    - Zakończ połączenie dla wybranego klienta.
  - Klient:
    - Zainicjuj połączenie z serwerem.
    - Wyślij wiadomość (treść wiadomości jest nieistotna).
    - Zakończ połączenie z serwerem.
- Serwer powinien wyświetlać listę wszystkich obecnie połączonych klientów oraz odbierane wiadomości.
- Komunikacja ma się odbywać w sieci **dockerowej**.
- Całość powinna być uruchamiana minimalną liczbą komend.

1.1 Wybrany wariant funkcjonalny

W1 – wykorzystanie mechanizmu **encrypt-then-mac** dla wysyłanych szyfrowanych wiadomości jako mechanizm integralności i autentyczności, implementacja w Pythonie.

2 Struktura wiadomości

- ClientHello:
  - niezaszyfrowany klucz publiczny klienta (domyślna długość 32 bajty)
  - zaszyfrowany przez klienta klucz wspólny
- ServerHello
  - niezaszyfrowany klucz publiczny serwera (domyślna długość 32 bajty)
  - zaszyfrowany przez serwer klucz wspólny
- Wiadomość
  - Bajty 0 - 31: MAC
  - Bajty 32+: szyfrogram
- EndSession - używa tej samej struktury co zwykła wiadomość, przesyłając na bajtach 32+ zaszyfrowaną wiadomość specjalną o treści "exit"

2.1 Przykładowy przebieg komunikacji

Krok	Klient	Serwer
1	• Ustalanie klucza sesyjnego.	• Utworzenie nowego wątku do obsługi połączenia. • Ustalanie klucza sesyjnego.
2	• Klient wczytuje wiadomość z terminala. • Wiadomość jest szyfrowana za pomocą klucza sesyjnego. • Generowany jest kod MAC na podstawie szyfrogramu i klucza sesyjnego. • Wiadomość jest łączona w podaną wyżej strukturę i wysyłana do serwera.	• Serwer przyjmuje wiadomość. • Dzieli ją na kod MAC oraz szyfrogram. • Oblicza kod MAC na podstawie klucza sesyjnego i szyfrogramu. Następnie sprawdza zgodność obliczonego kodu MAC z tym otrzymanym od klienta. • W zależności od zgodności kodów MAC: – Odsyła wiadomość o treści: "ERROR: MAC verification failed"jeśli kody nie były zgodne. – Odsyła wiadomość o treści: "ACK: received"jeśli autentyczność klienta była pozytywna.
3	• Klient przyjmuje odpowiedź serwera. • Wiadomość jest odszyfrowana i wyświetlana na terminalu.	• W przypadku pozytywnej weryfikacji autentyczności i integralności wiadomość jest odszyfrowana i wyświetlana na terminalu. • Oczekiwanie na kolejne wiadomości.

Tabela 1: Sekwencja wymiany kluczy pomiędzy Klientem a Serwerem (handshake).

3 Wykorzystane algorytmy

- Algorytm szyfrowania: Własna implementacja szyfru Vigenère'a, dla zachowania pierwotnej formy wiadomości używamy wszystkich znaków możliwych do wyświetlenia na konsoli jako słownika. Znaki wiadomości są przesuwane względem niego o wartość odpowiadającego im znaku klucza z uwzględnieniem cykliczności.
- Generowanie kodu MAC: wykorzystujemy gotową implementację HMAC opartą na funkcji skrótu SHA256 z biblioteki standardowej pythona.
- Generowanie klucza: Używając wszystkich znaków możliwych do wyświetlenia na konsoli, generujemy ich losowy ciąg o zadanej długości.

4 Sposób realizacji mechanizmu integralności i autentyczności dla szyfrowanych wiadomości

Skorzystaliśmy z podejścia Encrypt-then-MAC. Polega ono na najpierw zaszyfrowaniu wiadomości, a następnie generowaniu z niej kodu MAC. Zdecydowaliśmy się na to rozwiązanie, gdyż jest ono najpopularniejsze oraz jest uważane za najbezpieczniejsze.

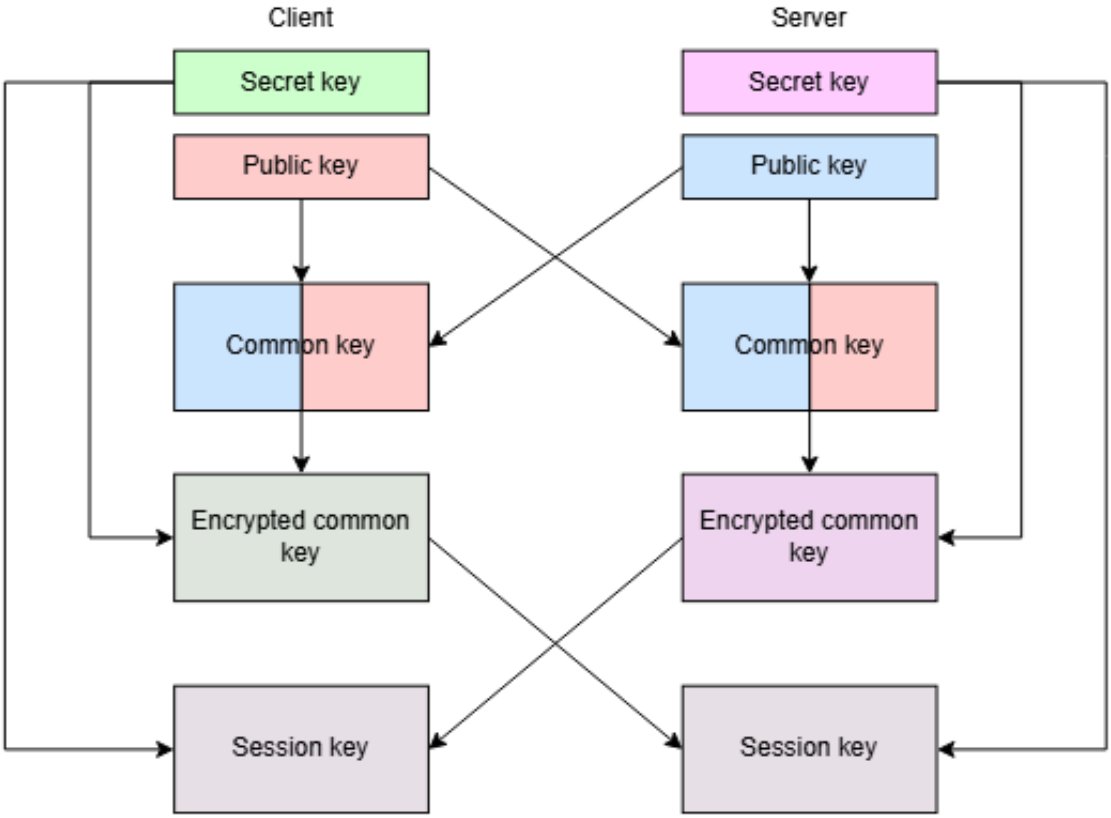
4.1 Autentyczność

Zapewnienie autentyczności opiera się na kodzie MAC. Jeśli odbiorca, korzystając z tajnego klucza sesyjnego, wygeneruje MAC zgodny z tym, który został dołączony do wiadomości, można założyć, że wiadomość pochodzi od nadawcy znającego klucz sesyjny. W przeciwnym razie, próba sfałszowania lub modyfikacji wiadomości przez atakującego, który nie zna klucza sesyjnego, spowoduje niezgodność kodów MAC i odrzucenie wiadomości.

4.2 Integralność

Integralność jest również zapewniona dzięki użyciu kodu MAC. Nadawca generuje kod MAC dla szyfrogramu, bazując na tajnym kluczu sesyjnym, i dołącza go do wiadomości. Odbiorca, po otrzymaniu wiadomości, samodzielnie oblicza MAC dla otrzymanego szyfrogramu i porównuje go z dołączonym kodem. Jeśli kody są zgodne, można mieć pewność, że treść wiadomości nie uległa modyfikacji w trakcie przesyłu.

5 Schemat ustalania klucza sesyjnego



Rysunek 1: Diagram ustalania klucza sesyjnego (handshake)

Krok	Klient	Serwer
1	• Generuje swój `secret_key` oraz `public_key`. • Wysyła `public_key` do serwera.	• Odbiera `client_public_key` (klucz publiczny Klienta). • Generuje swój `secret_key` oraz `public_key`. • Wysyła do Klienta swój klucz publiczny `server_key`.
2	• Odbiera `server_public_key` (klucz publiczny Serwera). • Tworzy `common_key` = (`server_public_key + client_public_key`).	• Tworzy `common_key` = (`server_public_key + client_public_key`).
3	• Szyfruje `common_key` swoim `secret_key` i wysyła wynik do Serwera.	• Szyfruje `common_key` swoim `secret_key` i wysyła wynik do Klienta.
4	• Odbiera od Serwera `common_encoded_server`.	• Odbiera od Klienta `common_encoded_client`.
5	• Szyfruje `common_encoded_server` swoim `secret_key`. • Otrzymany klucz jest wspólnym **kluczem sesji**.	• Szyfruje `common_encoded_client` swoim `secret_key`. • Otrzymany klucz jest wspólnym **kluczem sesji**.

Tabela 2: Sekwencja wymiany kluczy pomiędzy Klientem a Serwerem (handshake).