PSI dokumentacja wstępna

Paweł Spirydowicz, Hanna Zarzycka, Krzysztof Wojtaszko
29.12.2025 wersja 1

Treść zadania

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

Założenia

- Architektura klient serwer.
- Serwer jest w stanie obsłużyć kilku klientów jednocześnie (proszę nie hardcodować liczby klientów – oczekuję parametru uruchomienia).
- Klient inicjuje połączenie z serwerem poprzez wysłanie wiadomości ClientHello (nieszyfrowana),na którą serwer odpowiada wiadomością ServerHello (nieszyfrowana).
- Sesja może zostać zakończona zarówno przez klienta jak i przez serwer poprzez wysłanie wiadomości EndSession. Po odebraniu EndSession należy od nowa wysłać ClientHello.
- Wszystko poza ClientHello i ServerHello jest szyfrowane.
- Potencjalny napastnik po przechwyceniu wiadomości nie jest w stanie z nich nic odczytać.

Wariant

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

Struktura wiadomości

Typ wiadomości

Pierwsze dwa bity każdej wiadomości będzie zawierał informację o jej typie, według legendy:

- 0b00 - ClientHello
- 0b01 - ServerHello
- 0b10 - EncryptedMessage
- 0b11 - EndSession

ClientHello

Wiadomość ClientHello niesie informację o typie wiadomości, parametrach p i g algorytmu Diffiego-Hellmana oraz klucz publiczny klienta. Ma rozmiar 10 bajtów kolejno przydzielonych na dane informacje:

2 bity - typ
14 bitów - g
4 bajty - p
4 bajty - klucz publiczny

ServerHello

Wiadomość ServerHello niesie informację o typie wiadomości oraz kluczu publicznym serwera. Ma rozmiar 5 bajtów, kolejno przydzielnoych:

2 bity - typ (+**6** bitów dopełnienia)
4 bajty - klucz publiczny

EncryptedMessage

Zaszyfrowana wiadomość poprzedzona jest informacją o typie i długości wiadomości. Za treścią zaszyfrowanej wiadomości znajduje się MAC (Message Authentication Code). Cała struktura może mieć od 34 do 16417 bajtów:

2 bity - typ
14 bitów - długość wiadomości (n)
n bajtów - zaszyfrowana wiadomość
32 bajty - MAC

EndSession

Wiadomość EndSession zawiera 33 bajty:
2 bity - typ (+**6** bitów dopełnienia)
32 bajty - MAC

Wymiana kluczy

W celu wymiany kluczy korzystamy z algorytmu Diffiego-Hellmana.

Przykładowy scenariusz

Dla czytelności przykładu użyjemy znacznie niższych wartości niż będą stosowane w programie.

Klient wybiera liczbę pierwszą
p = 23,
podstawę g względnie pierwszą do p
g = 5,

oraz klucz prywatny

$a = 8$

i na ich podstawie oblicza swój klucz publiczny z wzoru:

$A = g^a = 5^8 \bmod 23 = 16.$

Klient wysyła wiadomość ClientHello:

0x00050000001700000010

Serwer odczytuje

typ - 0b00

g - 0b00000000000101 = 5

p - 0x00000017 = 23

A - 0x00000010 = 16

Serwer wybiera klucz prywatny

$b = 13$

i na jej podstawie oblicza klucz publiczny:

$B = g^b = 5^{13} \bmod 23 = 21$

Serwer wysyła wiadomość ServerHello:

0x4000000015

Klient odczytuje

Typ - 0b01

B - 0x00000015 = 21

Obie strony muszą obliczyć wspólny sekret:

Dla klienta

$s = B^a \bmod p = 21^8 \bmod 23 = 3$

Dla serwera

$s = A^b \bmod p = 16^{13} \bmod 23 = 3$

Uzyskany wspólny sekret można wykorzystać do wygenerowania klucza do szyfrowania wiadomości oraz do obliczania MAC.

Mechanizm integralności i autentyczności

Encrypt-then-MAC

Zdecydowaliśmy się na użycie mechanizmu integralności i autentyczności encrypt-then-MAC. Ta metoda polega na w pierwszej kolejności zaszyfrowaniu wiadomości kluczem szyfrującym, a następnie obliczeniu kodu uwierzytelniania wiadomości (MAC) na podstawie zaszyfrowanego tekstu. MAC stanowi rodzaj podpisuj pod wiadomością i jest w praktyce wyliczony przez funkcję haszującą lub szyfr blokowy z domieszanym kluczem tajnym. Zapewnia integralność, ponieważ każda próba zmiany treści zaszyfrowanej

wiadomości skutkuje zmianą wartości MAC przez co odbiorca odrzuca wiadomość. Zapewnia autentyczność, ponieważ wymaga znajomości tajnego współdzielonego klucza, aby uzyskać ten sam MAC, jak i odszyfrować wiadomość.

W implementacji tej metody planujemy zastosować HMAC, czyli typ kodu uwierzytelniającego, wykorzystujący funkcję haszującą. Jako funkcję haszującą użyjemy SHA256, która pozwoli nam otrzymać MAC o długości 32 bajtów. Funkcje potrzebne do realizacji tego zadania weźmiemy z modułów hmac i hashlib Pythona.