Projekt PSI - sprawozdanie wstępne

Adam Czupryński, Michał Sadlej, Szymon Makuch 22.12.2024.

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

Struktura

Struktury wiadomości będą składać 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

EndSession

- typ wiadomości
- długość wiadomości
- powód zakończenia sesji

Wykorzystane algorytmy

Wymiana kluczy

Zastosujemy 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

Zastosujemy 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

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