Dokumentacja Protokołu Ultra Simple Mail Transfer Protocol

Autorzy protokołu:

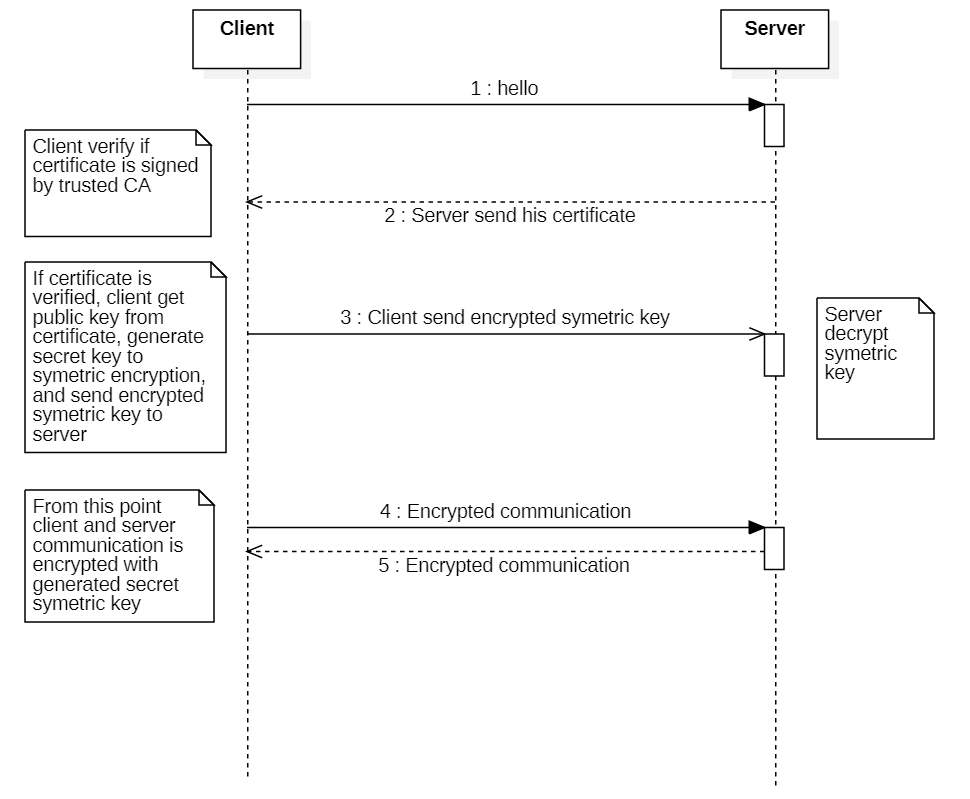
Michał Jakubowski

Mateusz Kopczan

W ramach dokumentacji chcielibyśmy przybliżyć schemat komunikacji według zaprojektowanego przez nas protokołu. Konkretne etapy komunikacji zostały poparte przykładami implementacji w języku Python. Protokół korzysta również z własnej wersji protokołu SSL.

**Protokół SSL:**

Protokół zapewnia poufność komunikacji w sieci poprzez szyfrowanie danych przesyłanych między podmiotami. Schemat komunikacji protokołu został zaprezentowany na rysunku nr 1.



**Rysunek 1** schemat komunikacji SSL

Klient wysyła wiadomość hello do serwera w celu zainicjowania komunikacji. Serwer w odpowiedzi odsyła swój certyfikat. Klient weryfikuje czy certyfikat został podpisany przez znany mu podmiot autoryzujący, jeśli tak to generuje klucz do szyfrowania symetrycznego. Następnie pobiera klucz publiczny serwera z certyfikatu i szyfruje nim swój klucz do komunikacji symetrycznej. Serwer odszyfrowuje z pomocą swojego klucza prywatnego klucz wysłany przez klienta. W ten sposób klient i serwer wymienili między sobą klucz do bezpiecznej komunikacji.

Do szyfrowania asymetrycznego jest wymagany klucz RSA w wersji 2048 bitowej, natomiast do szyfrowania symetrycznego jest wymagany algorytm AES 256 bitowy w trybie ECB.

Format przesyłanych wiadomości w ramach protokołu został zaprezentowany poniżej:

1.Inicjacja komunikacji(Klient):

Hello ssl1.0?**\r\n\r\n**

2.Odpowiedź na wersje(Serwer):

**201** ssl valid**\r\n\r\n**

3.Inicjacja szyfrowania(Klient):

support asymetric-rsa2048 symetric-aes-256-ecb?**\r\n\r\n**

4.Przesłanie certyfikatu(Serwer):

**202 algos valid** Treść\_certyfikatu**.\r\n\r\n**

5.Przesłanie zaszyfrowanego klucza do komunikacji(Klient):

Zaszyfrowana\_wiadomość( klucz )**\r\n\r\n**

Jak widać wszystkie komunikaty oprócz komunikatu z przesłanym certyfikatem kończą się podwójnym <crlfem>. Do komunikatu z wysyłanym certyfikatem przed <crlfem> zostaje dodana kropka, aby uniknąć problemów związanych z nowymi liniami w pliku. Powyżej został zaprezentowany schemat w wypadku poprawnej komunikacji, koniec komunikatów błędów wyznaczają również te same znaki, aby klient mógł w ten sam sposób odczytać błędne komunikaty jak i te prawidłowe.

**Kody odpowiedzi ssl:**

201 ssl valid - serwer odsyła informacje, że obsługuje tą wersję ssl

202 algos valid – serwer odsyła informacje, że obsługuje przesłane algorytmy szyfrowania asymetrycznego jak i symetrycznego

401 bad format – przesłano wiadomość w złym formacie

402 not supported asymetric algorithm – serwer nie obsługuje takiego asymetrycznego algorytmu

403 not supported symetric algorithm – serwer nie obsługuje takiego symetrycznego algorytmu

501 symetric key decription error – serwerowi nie udało się odszyfrować danych od klienta, może to być spowodowane nieprawidłowym szyfrowaniem

502 bad client key length – serwer dostał złą długość klucza dla algorytmu symetrycznego np. dla AES-256-ECB prawidłowe długości klucza to 16, 24, 32 bajtów

Powyżej przedstawiono kody odpowiedzi serwera na dane klienta