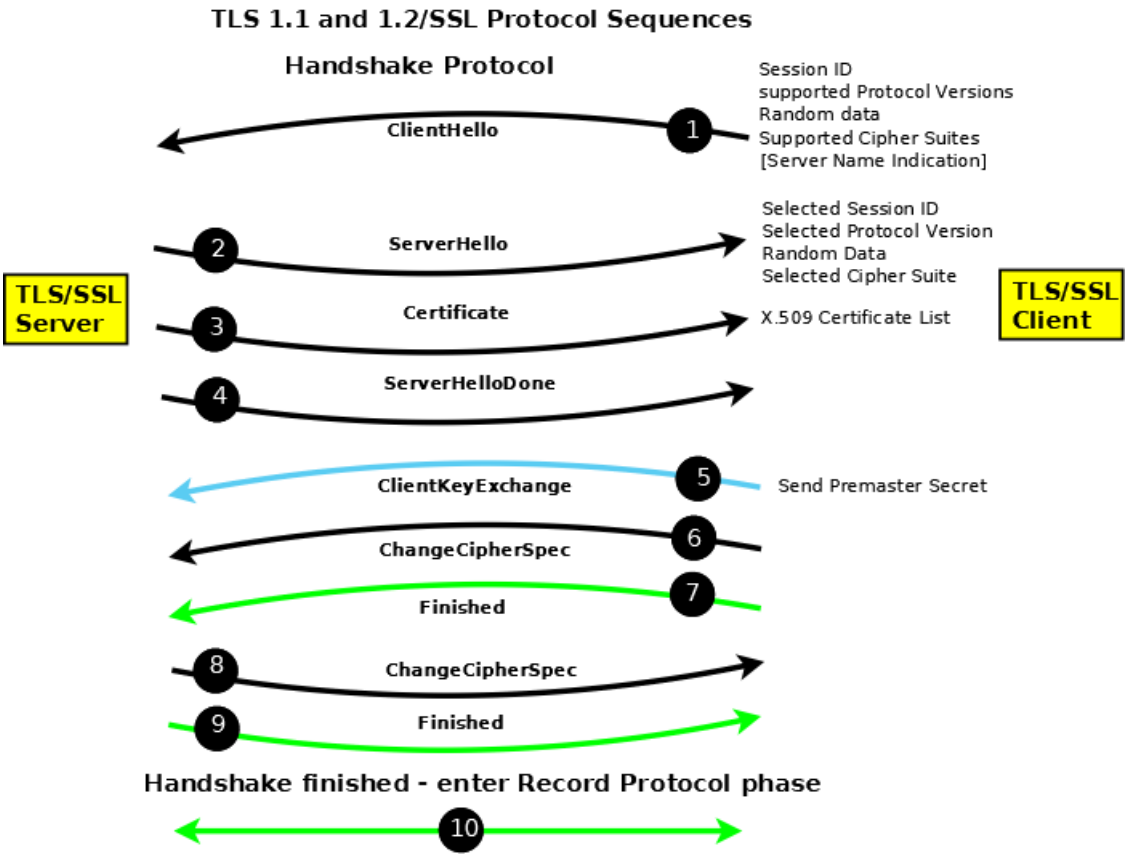
OpenSSL

Zanim zaczniemy rozważania nad OpenSSL należy zwrócić uwagę na specyfikę ruchu sieciowego. Wymiana informacji pomiędzy komputerami w czy to w sieci LAN, czy internecie odbywa się głównie za pomocą protokołów TCP oraz UDP. Dane, które chcemy wysłać dzielone są na małe części nazywane pakietami, a następnie wysyłane do odbiorcy. Po drodze przechodzą przez niezliczoną liczbę routerów, switchy i serwerów zanim dotrą do adresata. Na każdym z tych przystanków mogą zostać przeczytane (a nawet zmodyfikowane) na przykład przy pomocy sniffera. Każde logowanie na stronie internetowej przy pomocy protokołu HTTP oznacza wysłanie danych logowania, a najważniejsze hasła w postaci jawnej i zanim dotrą do celu mogą zostać podsłuchane, co jest ogromnym zagrożeniem dla bezpieczeństwa użytkownika. Problem ten nie dotyczy tylko protokołu HTTP, ale całego ruchu sieciowego. Jednym z rozwiązań tego problemu jest szyfrowanie danych.

Protokół TLS (ang. Transport Layer Security) jest obecnie standardem szyfrowania ruchu TCP. Powstał jako rozwinięcie protokołu SSL (ang. Secure Socket Layer) i umożliwia bezpieczną (szyfrowaną), integralną komunikację w sieci internet. Wykorzystywany jest głównie w zabezpieczaniu protokołów HTTP, czyli protokół HTTPS, POP3, IMAP, ale może być użyty do zabezpieczenia dowolnego protokołu opartego o TCP (protokoły oparte o UDP można zabezpieczyć za pomocą DTLS). Działa on w warstwie prezentacji modelu OSI. Dane szyfrowane przy użyciu tego protokołu są niemożliwe do odczytania przez urządzenia pośredniczące (dane są zaszyfrowane przy użyciu odpowiednio silnego algorytmu) i zapewniona jest ich integralność.

Teraz gdy już została omówiona idea i cel powstania TLS można przejść do zasady działania. Na poniższym obrazku zaprezentowany jest proces nawiązywania połączenia i wymiany klucza w protokole TLS w wersji 1.1 oraz 1.2 (w wersji 1.3 ten proces nieco się różni).



1. Klient → Serwer ClientHello
Klient wysyła wiadomość, która między innymi zawiera identyfikator sesji, obsługiwane wersje protokołu, losową liczę oraz obsługiwane sposoby szyfrowania.
2. Serwer → Klient ServerHello
Serwer zwraca podobną wiadomość, która zawiera wybraną wersję protokołu, sposób szyfrowania oraz inną losową liczbę.
3. Serwer → Klient Certificate
Serwer wysyła do klienta certyfikat w formacie X.509, klient weryfikuje go i może zakończyć połączenie jeśli okaże się nieprawidłowy.
4. Serwer → Klient ServerHelloDone
Serwer wysyła informację, że może przejść do następnej fazy połączenia.
5. Klient → Serwer ClientKeyExchange
Klient wysyła wstępny klucz sesji, jest on ustalany na podstawie liczb losowych wysłanych przez klienta i serwer (protokół diffiego-hellmana). Na podstawie tego klucza ustalany jest faktyczny klucz do komunikacji. Wiadomość jest szyfrowana za pomocą klucza publicznego, znajdującego się w certyfikacie serwera. Jest to bardzo ważny etap, ponieważ przesyłane dane będą szyfrowane za pomocą symetrycznego algorytmu (np. AES), a algorytm asymetryczny jest potrzebny tylko do bezpiecznej wymiany klucza. Dzięki temu zostaje zwiększona wydajność (szyfrowanie symetryczne jest szybsze od asymetrycznego) oraz możliwe jest utajnianie z wyprzedzeniem (ang. forward secrecy).
6. Klient → Serwer ChangeCipherSpec
Klient informuje serwer, że może przełączyć się na komunikację szyfrowaną.
7. Klient → Serwer Finished
Klient informuje serwer, że jest gotowy do odbierania zaszyfrowanych wiadomości.
8. Serwer → Klient ChangeCipherSpec
Serwer zawiadamia, że od tej pory będzie przesyłać zaszyfrowane wiadomości.
9. Serwer → Klient Finished
Serwer testuje połączenie wysyłając zaszyfrowaną widomość.
10. Nawiązywanie połączenia zostało zakończone, teraz wszystkie wiadomości będą szyfrowane za pomocą algorytmu symetrycznego i klucza ustalonego w kroku piątym.

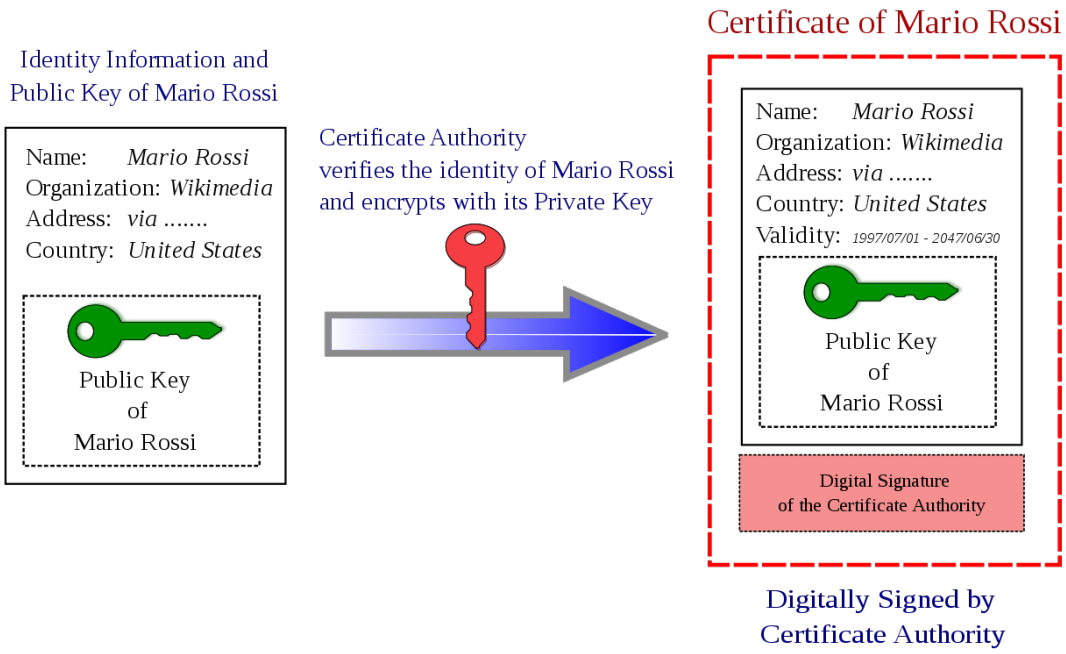
W punkcie trzecim przesyłany jest certyfikat, nasuwa się pytanie czym on dokładnie jest. Przedtem jednak musimy przedstawić ideę szyfrowania asymetrycznego. Cechą charakterystyczną jest obecność dwóch kluczy: publicznego i prywatnego. Klucz publiczny służy do szyfrowania danych, a klucz prywatny do deszyfrowania. Klucz publiczny może być przekazywany w dowolny sposób, ponieważ wydobycie z niego klucza prywatnego jest praktycznie niemożliwe, analogicznie zaszyfrownane dane są bezpieczne. Natomiast klucz prywatny musi być trzymany w bezpieczny sposób, ponieważ w razie jego kompromitacji osoby trzecie mogą odszyfrować wiadomości. W przypadku algorytmu RSA kluczami są pary liczb (n, e) w przypadku klucza publicznego oraz (n, d) w przypadku klucza prywatnego. Innym zasotsowaniem klucza prywatnego jest podpiswanie, polega ono na obliczeniu funkcji skrótu (hash) wiadomości, zaszyfrowaniu go kluczem prywatnym i dołączeniu go do wiadomości, taki podpis jest niemożliwy do podrobienia. Dzięki temu odbiorca posiadający klucz publiczny może odszyfrować wartość funkcji skrótu, a następnie porównać z wartością funkcji skrótu otrzymanej wiadomości. Pozwala to odbiorcy upewnić się, że wiadomość pochodzi od nadawcy oraz że nie została przez nikogo zmodyfikowana. Jedyny problem to czy klucz publiczny należy do osoby z którą chcemy się kontaktować. Do rozwiązania tego problemu powstał standard X.509 – podpisywania kluczy publicznych przez zaufaną stronę trzecią (urząd certyfikacji). Certyfikatem w standardzie X.509 nazywamy strukturę zbudowaną z:

- kontenera zawierającego właściwą zawartość certyfikatu:
  - numer seryjny certyfikatu
  - opis urzędu certyfikacji wystawiający certyfikat
  - czas początku i końca ważności certyfikatu

- opis właściciela certyfikatu
- klucz publiczny właściciela
- oraz inne mniej ważne artybuty

- informacji o algorytmie użytym do podpisania
- podpisu cyfrowego (liczona jest funkcja skrótu kontenera, a następnie szyfrowana za pomocą klucza prywatnego urzędu certyfikacji)

System certyfikatów gwarantuje, że klucz publiczny faktycznie należy do osoby, z którą chcemy się kontaktować, gwarantuje to urząd certyfikacji weryfikując żądania podpisania certyfikatu (w przypadku certyfikatów SSL często trzeba ustawić odpowiednią wartość w polu TXT w DNS, aby certyfikat został wygenerowany). Urząd certyfikacji jest podmiotem, któremu całkowicie ufamy i wszystkie certyfikaty przez niego wydane uznajemy za bezpieczne. Można sprawdzić ważność takiego certyfikatu – czy jest przeterminowany lub znajduje się na liście unieważnionych certyfikatów. Właściciel certyfikatu w przypadku utracenia klucza prywatnego lub jego ujawnienia (lub innych rzadziej spotykanych powodów) może zgłosić prośbę o unieważnienie certyfikatu. Taki certyfikat dodawany jest do listy unieważnionych certyfikatów (CRL) i nie jest już uznawany za prawidłowy.



Powyższy obrazek pokazuje w jaki sposób generowane są certyfikaty przez urząd certyfikacji. Zgłaszający wysyła żądanie w odpowiednim formacie, następnie urząd weryfikuje tożsamość zgłaszającego i w przypadku pozytywnego wyniku generuje certyfikat na podstawie zgłoszenia i podpisuje go. Jednym z urzędów, które udostępniają certyfikaty SSL za darmo jest organizacja Let’s Encrypt.

Nadal jednak nie wyjaśniłem jednak tematu referatu. Czym w ogóle jest **OpenSSL**? OpenSSL – wieloplatformowa, otwarta implementacja protokołów SSL (wersji 2 i 3) i TLS (wersji 1) oraz algorytmów kryptograficznych ogólnego przeznaczenia. Udostępniana jest na licencji zbliżonej do licencji Apache(open source). Dostępna jest dla systemów uniksopodobnych (m.in. Linux, BSD, Solaris), OpenVMS i Microsoft Windows.

Trochę historii. Projekt OpenSSL powstał w 1998 roku w celu utworzenia darmowego zestawu narzędzi szyfrujących, przeznaczonych dla kodu używanego w sieci. Bazuje on na SSLeay, którego twórcami byli Eric Andrew Young i Tim Hudson. Nieoficjalne zakończenie jego rozwoju datowane jest na 17 grudnia 1998, kiedy to Young wraz z Hudsonen rozpoczęli pracę dla RSA Security. W skład zespołu, zajmującego się projektem OpenSSL wchodziło czterech Europejczyków. Cały zespół liczył 11 osób, z których 10 było wolontariuszami. Jedynym pracownikiem, pracującym na pełny etat, był szef projektu - Stephen Henson. Budżet projektu wynosił niecały million dolarów rocznie i polegał częściowo na dotacjach. Steve Marquess, były konsultant CIA w Maryland powołał fundację, zajmującą się darowiznami i umowami konsultingowymi. Pozyskał ponadto sponsoring od Departamentu Bezpieczeństwa Krajowego Stanów Zjednoczonych oraz Departamentu Obrony Stanów Zjednoczonych.
W roku 2013, WikiLeaks opublikowała dokumenty zdobyte przez Edwarda Snowdena, według których od 2010 roku, NSA pomyślnie złamała bądź obeszła zabezpieczenia SSL/TLS za pomocą luk, takich jak HeartBleed. W roku 2014 około dwie trzecie wszystkich serwerów korzystało z OpenSSL.

OpenSSL zawiera biblioteki implementujące wspomniane standardy oraz mechanizmy kryptograficzne, a także zestaw narzędzi konsolowych (przede wszystkim do tworzenia kluczy oraz certyfikatów, zarządzania urzędem certyfikacji, szyfrowania, dekryptażu i obliczania podpisów cyfrowych). Za pomocą OpenSSL Crypto Library można m.in. obliczać funkcję skrótu wiadomości (m.in. MD5 i SHA-1) oraz szyfrować dane popularnymi algorytmami kryptograficznymi, m.in. Blowfish, AES, IDEA, 3DES.

Zasadniczo OpenSSL ma postac biblioteki ANSI C, która implementuje podstawowe operacje kryptograficzne. Poza funkcjami niezbędnymi do szyfrowania sieciowej warstwy transportu, zawiera również funkcje szyfrowania symetrycznego(dla plików), podpisy cyfrowe, kryptograficzne funkcje skrótu, generatory liczb losowych etc etc.

Program openssl umożliwia następujące operacje:

- Tworzenie i zarządzanie kluczami prywatnymi, publicznymi i ich parametrami,

- Operacje kryptograficzne z kluczem publicznym,

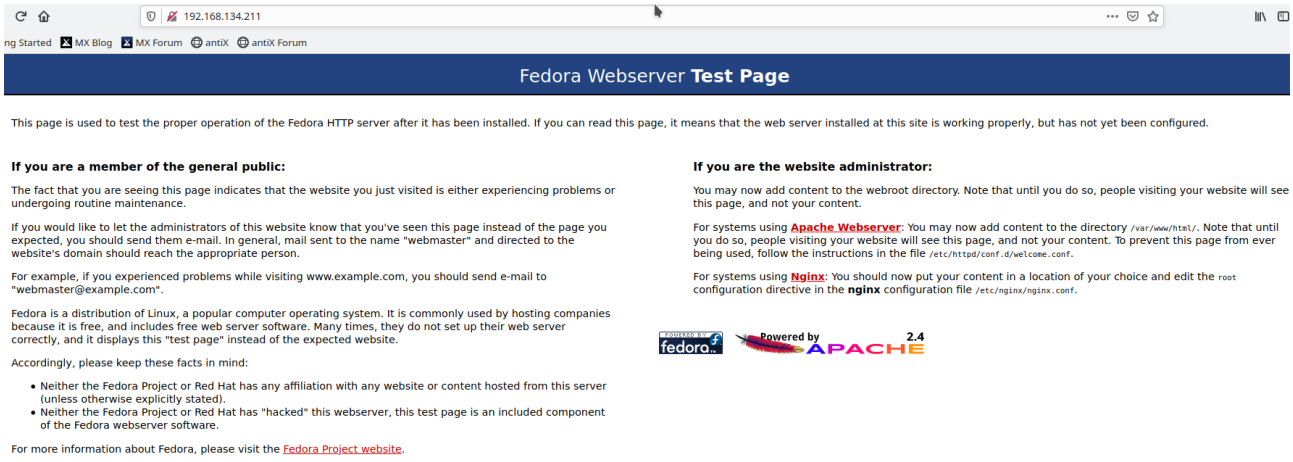
- Tworzenie certyfikatów X.5009, CSR oraz CLR,

- Obliczanie skrótów wiadomości.

OpenSSL posiada swoje alternatywy.

- Jednym z nich jest **Agglomerated SSL**. W roku 2009, Marco Peereboom (wówczas pracował jako programista OpenBSD), niezadowolony z oryginalnego interfejsu OpenSSL, stworzył fork oryginalnego interfejsu o nazwie Agglomerated SSL (assl). W dalszym ciągu wykorzystuje on „wewnętrzny” interfejs OpenSSL, lecz posiada znacznie prostszy interfejs zewnętrzny.
- Kolejny z nich o którym warto wspomnieć to **LibreSSL**. Po ujawnieniu błędu Heartbleed(krytyczny błąd wsytępujący w pewnej wersji OpenSSLa, pozwalający na niepożądany odczyt danych), członkowie projektu OpenBSD – wraz z wydaniem wersji 1.0.1g – stworzyli rozwidlenie OpenSSL'a o nazwie LibreSSL. W pierwszym tygodniu pracy nad porządkowaniem aplikacji usunięto 90 000 linii kodu źródłowego.
- Warto również wspomnieć o inicjatywie,znanej wszystkim, firmy Google. Mowa o **BoringSSL**. BoringSSL to fork OpenSSLa, który został zaprojektowane z myślą o potrzebach Google. Google przez lata użytkownia OpenSSL wprowadziło tam sporo własnych zmian i istotnych poprawek. Gdy portfolio produktów Google stało się bardziej złożone, pojawiło się więcej kopii OpenSSL, a wysiłek związany z utrzymaniem wszystkich tych poprawek w wielu miejscach stale wzrastał.
Z tych właśnie powodów powstał „guglowski” BoringSSL.

Przejdę teraz do części praktycznej i zadania. Potrzebne będzie połączenie z serwerem za pomocą SSH oraz już działający i skonfigurowany serwer www Apache (po wpisaniu adresu IP w przeglądarkę wyświetla się strona jak na poniższym obrazku). Wszystkie poniższe komendy powinny zostać wywołane przez użytkownika z uprawnieniami roota.

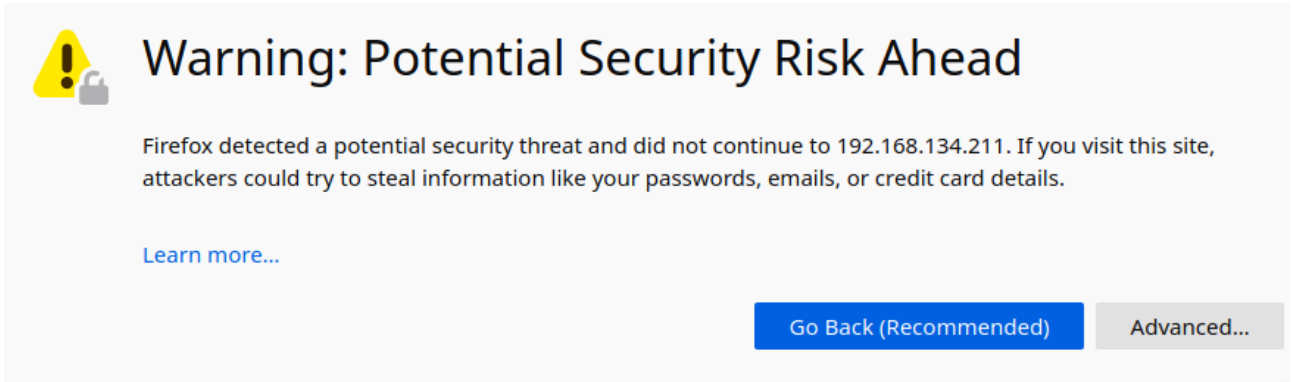


1. Pierwszym krokiem jest zainstalowanie modułu dodającego obsługę połączeń szyfrowanych*:*
`dnf install mod_ssl -y`

2. Po zakończonej instalacji należy skonfigurować firewall żeby nie blokował portu 443 (ten port jest używany przez protokół HTTPS)
`firewall-cmd --permanent --add-service=https`
`firewall-cmd --add-service=https`

3. Następnie należy zrestartować serwer www:
`systemctl restart httpd.service`

Podczas instalacji został wygenerowany domyślny certyfikat i można to sprawdzić wpisując w przeglądarkę adres https://192.168.134.211 (adres IP należy zmienić na adres swojego serwera).



Oczywiście w przeglądarce pojawia się ostrzeżenie, że certyfiakt jest niezaufany. Dzieje się tak dlatego, że nie jest on wystawiony przez zaufany urząd certyfikacji tylko samopodpisany (ang. self-signed). Po kliknięciu przycisku Advanced i zaakceptowaniu ryzyka pojawi się taka sama strona jak dwa obrazki wyżej.

Teraz można przejść do generowania samopodpisanego certyfikatu. Najpierw generujemy klucz prytwatny RSA o rozmiarze 4096:
`openssl genrsa -out localhost.key 4096`
Nazwę ‘localhost.key’ można zmienić na inną, zazwyczaj jest to nazwa domeny.

Następnie generujemy żądanie podpisania certyfikatu:
```
openssl req -new -key localhost.key -out localhost.csr -sha512
```
Zostaniemy poproszeni o podanie min. ‘Common Name’ czyli domemy, z którą ma być skojarzony certyfikat, lub adresu IP.

Samopodpisujemy certyfikat poleceniem:
```
openssl x509 -req -days 365 -in localhost.csr -signkey localhost.key -out localhost.crt -sha512
```
Wygenerowany certyfikat będzie ważny 365 dni, zgodnie z podanym parametrem.

Usuwamy żądanie certyfikatu:
```
rm localhost.csr
```

Powyższe polecenia można zastąpić jednym, działającym dokładnie tak samo:
```
openssl req -x509 -newkey rsa:4096 -keyout localhost.key -out localhost.crt -days 365 -nodes -sha512
```

Kolejnym krokiem jest przeniesienie plików w odpowiednie miejsce i upewnienie się, że dostęp do plików jest odpowiednio ustawiony:
```
mv localhost.key /etc/pki/tls/private/
mv localhost.crt /etc/pki/tls/certs/
chown root.root /etc/pki/tls/private/localhost.key
chown root.root /etc/pki/tls/certs/localhost.crt
chmod 0600 /etc/pki/tls/private/localhost.key
chmod 0600 /etc/pki/tls/certs/localhost.crt
restorecon -RvF /etc/pki
```

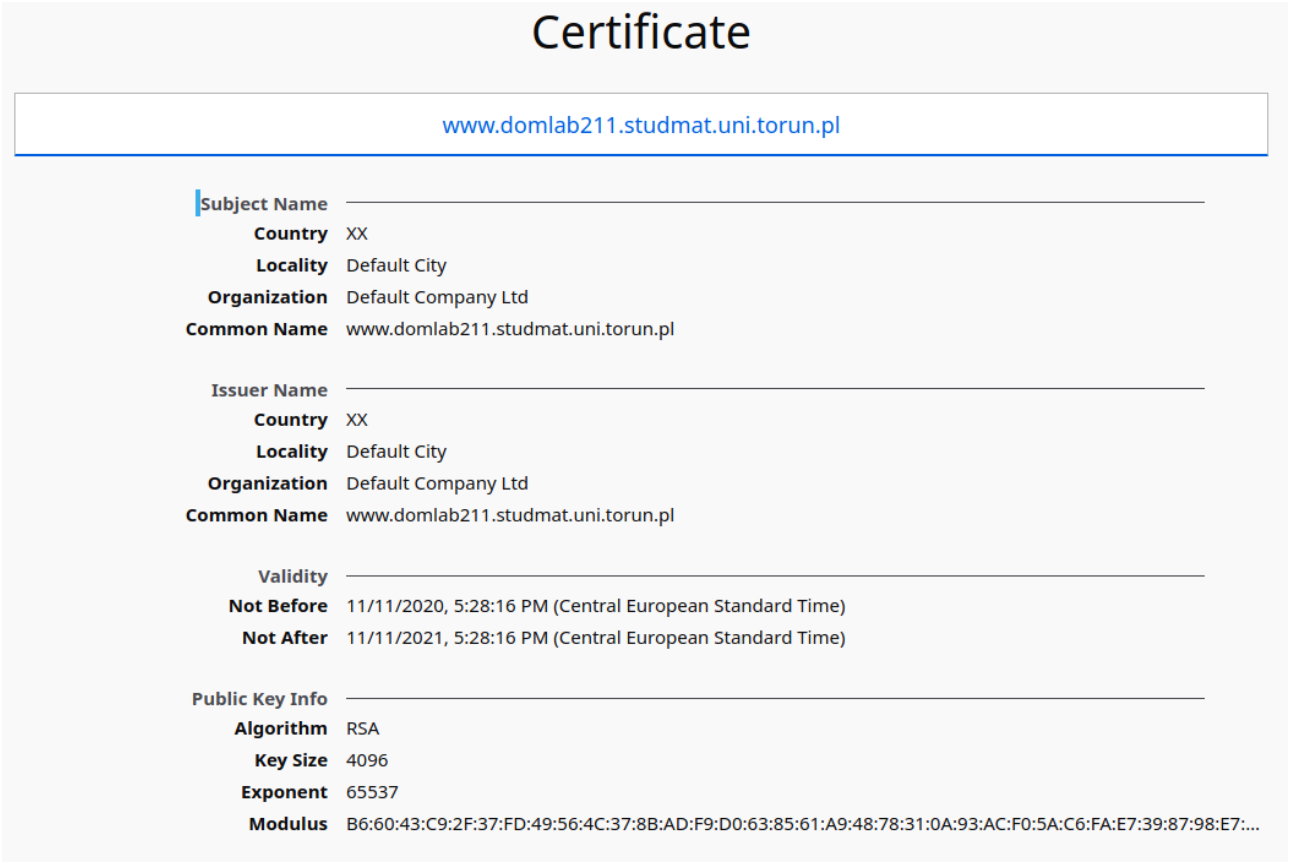
Przedostatnim etapem jest konfiguracja serwera Apache. W blocku VirtualHost należy zmienić port z 80 na 443 oraz dodać następujące linijki:
```
SSLCertificateFile /etc/pki/tls/certs/domena.crt
SSLCertificateKeyFile /etc/pki/tls/private/domena.key
```

Przykładowo tak wygląda konfoguracja HTTP:
```
<VirtualHost *:80>
    ServerAdmin admin@example.com
    ServerName www.domlab211.studmat.uni.torun.pl
    DocumentRoot /var/www/domlab211
    ErrorLog /var/log/httpd/error-domlab211.log
    CustomLog /var/log/httpd/access-domlab211.log combined
</VirtualHost>
```

A tak konfiguracja HTTPS:
```
<VirtualHost *:443>
    ServerAdmin admin@example.com
    ServerName www.domlab211.studmat.uni.torun.pl
    DocumentRoot /var/www/domlab211
    ErrorLog /var/log/httpd/error-domlab211.log
    CustomLog /var/log/httpd/access-domlab211.log combined
    SSLCertificateFile /etc/pki/tls/certs/www.domlab211.studmat.uni.torun.pl.crt
    SSLCertificateKeyFile
/etc/pki/tls/private/www.domlab211.studmat.uni.torun.pl.key
</VirtualHost>
```

Na koniec należy zrestartować serwer:
```
systemctl restart httpd.service
```

Jeśli wszystko wykonaliśmy poprawnie teraz strona powinna być dostępna w wersji szyfrowanej oraz z certyfikatem, który właśnie wygenerowaliśmy. Można to sprawdzić wchodząc na stronę w przeglądarce i sprawdzić szczegóły certyfikatu.



Jak widać wszystko się zgadza, strona *https://www.domlab211.studmat.uni.torun.pl/* działa oraz certyfikat jest samopodpisany.

Autorzy:
Michał Guźlewski
Patryk Murawski

Bibliografia:

1. Transport Layer Security - Wikipedia polska
2. Transport Layer Security - Wikipedia angielska
3. Datagram Transport Layer Security - Wikipedia angielska
4. TLS/SSL and SSL (X.509) Certificates – zytrax
5. Utajnianie z wyprzedzeniem - Wikipedia polska
6. Forward secrecy - Wikipedia angielska
7. Protokół Diffiego-Hellmana - Wikipedia polska
8. Algorytm RSA - Wikipedia polska
9. Podpis cyfrowy - Wikipedia polska
10. X.509 - Wikipedia polska
11. Certyfikat X.509 - Wikipedia polska
12. Urząd certyfikacji - Wikipedia polska
13. Urząd certyfikacji - Wikipedia angielska
14. OpenSSL - Wikipedia polska
15. OpenSSL – Kcir
16. Alternatywy OpenSSLa - Wikipedia polska
17. BoringSSL - Google git
18. Konfigurowanie Apache - Fedora Docs
19. Generowanie certyfikatu - Fedora Wiki
20. Generowanie certyfikatu - Stackoverflow