Wydział Elektroniki,
Fotoniki i Mikrosystemów

Inżynieria Oprogramowania dla Elektromobilności

Ćwiczenie nr 7. Szyfrowanie komunikacji bezprzewodowej

Zagadnienia do opracowania:

- system kontroli wersji na przykładzie Git
- system budowania CMake
- paradygmat programowania obiektowego i jego cechy (hermetyzacja, abstrakcja, dziedziczenie, polimorfizm)
- kryptografia asymetryczna i jej zastosowanie w generowaniu podpisów cyfrowych (algorytm RSA)
- funkcje skrótu kryptograficznego (funkcje haszujące)
- standard certyfikatów klucza publicznego X.509
- protokoły SSL i TLS

Literatura

[1] *CMake. Documentation.* https://cmake.org/documentation/. [Online; dostęp 22.10.2022].

[2] W. Gajda. *Git. Rozproszony system kontroli wersji.* Gliwice, Polska: Helion, 2013.

[3] Kornelia Indykiewicz. *Wykład: Inżynieria Oprogramowania dla Elektromobilności.* 2022.

[4] Prata S. *Szkoła Programowania. Język C++.* Gliwice, Polska: Helion, 2006.

[5] S. Chacon; B. Straub. *Pro Git.* https://git-scm.com/book/pl/v2/. [Online; dostęp 22.10.2022]. 2014.

Spis treści

1 Cel ćwiczenia **2**

2 Wprowadzenie **2**

2.1 Pobieranie aplikacji serwera wiadomości 2
2.2 Synchronizacja repozytorium serwera wiadomości 3
2.3 Konfiguracja aplikacji serwera wiadomości 4
2.4 Pobieranie aplikacji graficznego interfejsu użytkownika 7
2.5 Synchronizacja repozytorium graficznego interfejsu użytkownika 7
2.6 Konfiguracja aplikacji graficznego interfejsu użytkownika . . . 9
2.7 Biblioteka OpenSSL 9
2.7.1 Generowanie certyfikatu klucza publicznego 9
2.7.2 Instalacja OpenSSL 20
2.8 Analiza komunikacji sieciowej 21
2.9 Zintegrowane środowisko programistyczne Qt Creator 22

3 Program ćwiczenia **23**

1. Cel ćwiczenia

Celem ćwiczenia jest zapoznanie z zagadnieniem bezpieczeństwa komunikacji bezprzewodowej. Tematyka zajęć obejmuje implementację szyfrowanej komunikacji bezprzewodowej w architekturze klient–serwer przy pomocy biblioteki OpenSSL.

2. Wprowadzenie

2.1. Pobieranie aplikacji serwera wiadomości

Jeżeli wcześniej pobrano aplikację serwera wiadomości, należy przejść od razu do punktu 2.2. Aby pobrać aplikację serwera wiadomości należy:

1. utworzyć swój katalog roboczy wewnątrz folderu **Documents** (jeżeli katalog jeszcze nie istnieje). Nazwa katalogu powinna być unikalna (np. stanowić numer indeksu);
2. z poziomu utworzonego katalogu uruchomić konsolę systemową (terminal) i pobrać repozytorium projektu wywołując polecenie:

   git clone *adres_repozytorium*

3. przenieść się do katalogu zawierającego pobrane repozytorium:

   cd messageserver

4. za pomocą konsoli systemowej wywołać polecenie:

   git submodule update --init --recursive

   w celu zaimportowania zależności projektowych (podmodułów);
5. jeżeli nie instalowano wcześniej biblioteki paho MQTT (patrz: Ćw. 3), to przenieść się do katalogu:

messageserver\third_parties\libipc\third_parties\paho.mqtt.c

i za pomocą konsoli systemowej wywołać polecenie:
Uwaga: To jest pojedyncze polecenie:

cmake -Bbuild -H. -DPAHO_ENABLE_TESTING=OFF -DPAHO_BUILD_STATIC=ON -DPAHO_WITH_SSL=ON -DPAHO_HIGH_PERFORMANCE=ON

a następnie polecenie:

sudo cmake --build build/ --target install

2.2. Synchronizacja repozytorium serwera wiadomości

W celu synchronizacji repozytorium serwera wiadomości należy kolejno:

1. przejść do głównego repozytorium projektu (**messageserver**)
2. ukryć lokalne zmiany w plikach repozytorium (jeśli wprowadzano zmiany w kodzie) wywołując za pomocą konsoli systemowej polecenie:

   git stash

3. zsynchronizować stan kopii lokalnej repozytorium ze stanem zdalnego repozytorium wywołując polecenie:

   git pull origin master

4. zsynchronizować stan zależności projektowych (podmodułów) wywołując polecenie:

   git submodule update --init --recursive

5. przywrócić lokalne zmiany w plikach repozytorium (jeśli zmiany zostały ukryte; patrz punkt 2.) wywołując polecenie:

   git stash pop

6. w głównym katalogu repozytorium (**messageserver**) utworzyć katalog roboczy (np. **build**):

   mkdir build

7. przenieść się do utworzonego katalogu roboczego i za pomocą konsoli systemowej wywołać polecenie:

   cmake ..

   w celu wykonania skryptu **CMakeLists.txt**;

8. za pomocą konsoli systemowej wywołać polecenie:

   make

   aby zbudować plik wykonywalny aplikacji *message_server*.

2.3. Konfiguracja aplikacji serwera wiadomości

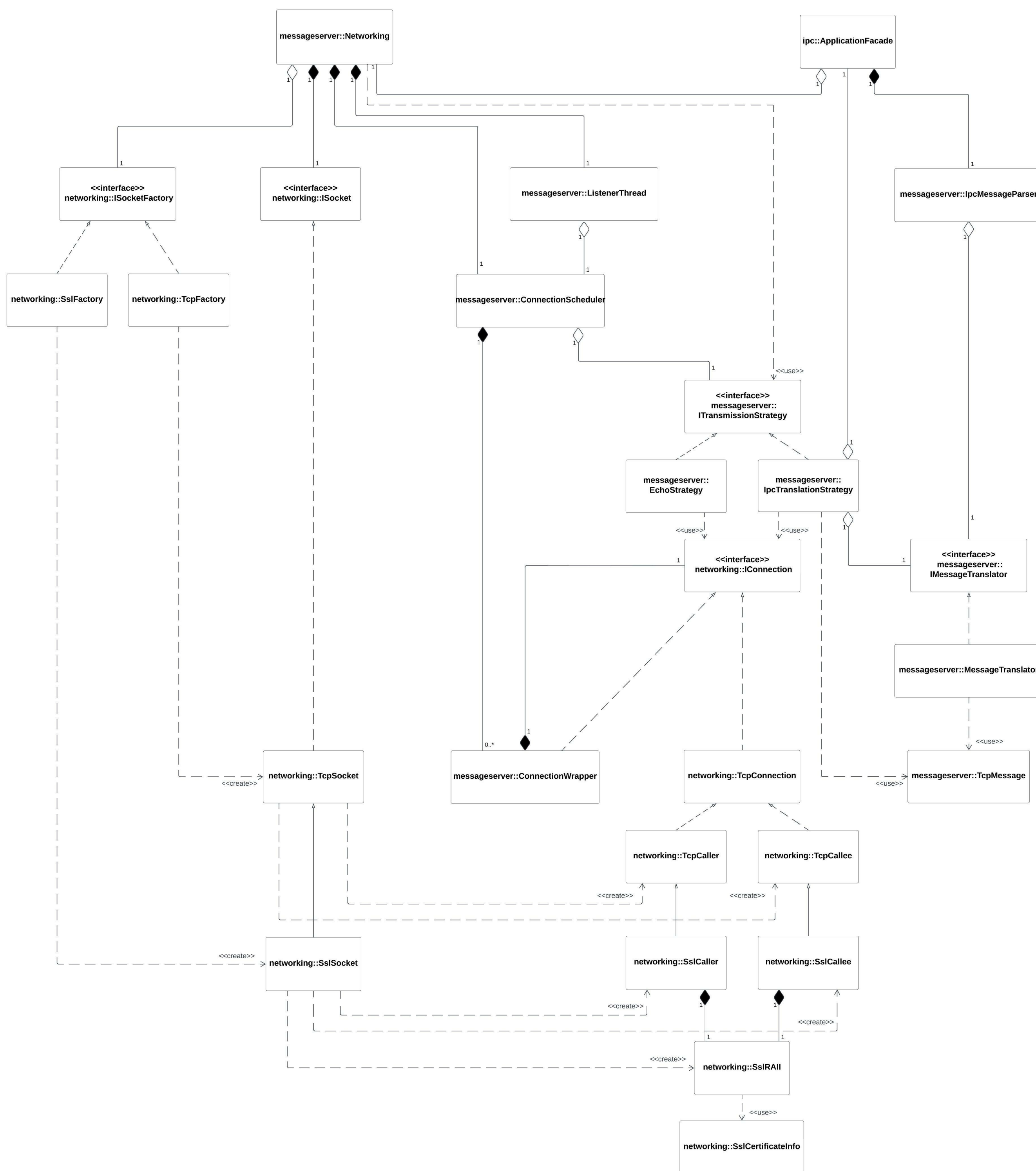
Katalog **data** znajdujący się w głównym katalogu repozytorium projektu (**messageserver**) zawiera plik konfiguracyjny w formacie JSON **app_config.json**, składający się z siedmiu atrybutów:

- **port_number** – numer portu protokołu TCP (domyślnie 8080);
- **awaiting_connections_queue_depth** – maksymalna głębokość kolejki oczekujących na nawiązanie połączenia klientów TCP;
- **timeout_in_seconds** – maksymalny czas (w sekundach) oczekiwania na wykonanie blokującego wywołania na gnieździe TCP;

- **enable_echo** – dopuszczalne wartości **true/false**; włącz albo wyłącz pracę w trybie echa (z pominięciem komunikacji z procesorem danych); **Uwaga: W celu realizacji Ćw. nr 7 należy ustawić wartość na true**;
- **enable_ssl** – dopuszczalne wartości **true/false**; włącz albo wyłącz komunikację z wykorzystaniem protokołu bezpieczeństwa SSL; **Uwaga: W celu realizacji Ćw. nr 7 należy ustawić wartość na true**;
- **path_to_ssl_certificate** – bezwzględna ścieżka do pliku zawierającego certyfikat klucza publicznego serwera (wykorzystywana przy komunikacji z użyciem protokołu SSL);
- **path_to_private_key_file** – bezwzględna ścieżka do pliku zawierającego prywatny klucz RSA serwera (wykorzystywana przy komunikacji z użyciem protokołu SSL).

Plik konfiguracyjny jest parsowany podczas uruchamiania aplikacji. W celu wczytania zmian w pliku należy ponownie uruchomić aplikację serwera wiadomości.

Strukturę aplikacji serwera wiadomości w postaci diagramu klas UML przedstawiono na rys. 2.1.



Rys. 2.1. Diagram klas UML aplikacji *message_server*

2.4. Pobieranie aplikacji graficznego interfejsu użytkownika

Jeżeli wcześniej pobrano aplikację graficznego interfejsu użytkownika, należy przejść od razu do punktu 2.5. Aby pobrać aplikację graficznego interfejsu użytkownika należy:

1. utworzyć swój katalog roboczy wewnątrz folderu **Documents** (jeżeli katalog jeszcze nie istnieje). Nazwa katalogu powinna być unikalna (np. stanowić numer indeksu);

2. z poziomu utworzonego katalogu uruchomić konsolę systemową (terminal) i pobrać repozytorium projektu wywołując polecenie:

   git clone *adres_repozytorium*

3. przenieść się do katalogu zawierającego pobrane repozytorium:

   cd messengergui

4. za pomocą konsoli systemowej wywołać polecenie:

   git submodule update --init --recursive

   w celu zaimportowania zależności projektowych (podmodułów).

2.5. Synchronizacja repozytorium graficznego interfejsu użytkownika

W celu synchronizacji repozytorium graficznego interfejsu użytkownika należy kolejno:

1. przejść do głównego repozytorium projektu (**messengergui**)

2. ukryć lokalne zmiany w plikach repozytorium (jeśli wprowadzano zmiany w kodzie) wywołując za pomocą konsoli systemowej polecenie:

git stash

3. zsynchronizować stan kopii lokalnej repozytorium ze stanem zdalnego repozytorium wywołując polecenie:

   git pull origin master

4. zsynchronizować stan zależności projektowych (podmodułów) wywołując polecenie:

   git submodule update --init --recursive

5. przywrócić lokalne zmiany w plikach repozytorium (jeśli zmiany zostały ukryte; patrz punkt 2.) wywołując polecenie:

   git stash pop

6. w głównym katalogu repozytorium (**messengergui**) utworzyć katalog roboczy (np. **build**):

   mkdir build

7. przenieść się do utworzonego katalogu roboczego i za pomocą konsoli systemowej wywołać polecenie:

   cmake ..

   w celu wykonania skryptu **CMakeLists.txt**;

8. za pomocą konsoli systemowej wywołać polecenie:

   make

   aby zbudować plik wykonywalny aplikacji *messenger_gui*.

2.6. Konfiguracja aplikacji graficznego interfejsu użytkownika

Aby skonfigurować aplikację graficznego interfejsu użytkownika do obsługi komunikacji szyfrowanej należy:

1. wpisać adres IPv4 serwera (w przypadku komunikacji w obrębie jednego hosta należy podać adres *localhost* – **127.0.0.1**)
2. wpisać numer portu nasłuchowego TCP serwera (domyślnie **8080**)
3. zaznaczyć opcję **Enable SSL**; zostanie otwarte okno, w którym należy wskazać plik w formacie PEM (*Privacy Enhanced Mail*) zawierający zaufany **certyfikat klucza publicznego** podmiotu certyfikującego serwer wiadomości
4. nacisnąć przycisk **Connect**

2.7. Biblioteka OpenSSL

OpenSSL jest wieloplatformowym projektem informatycznym wdrażającym rozwiązania z zakresu kryptografii na licencji **Open–Source**. Poza biblioteką programistyczną, **OpenSSL** dostarcza m.in. narzędzia do generowania **kluczy algorytmu RSA** czy **certyfikatów klucza publicznego**. W ramach laboratorium **OpenSSL** będzie wykorzystany do implementacji szyfrowanego połączenia bezprzewodowego między aplikacją graficznego interfejsu użytkownika (*messenger_gui*) a aplikacją serwera wiadomości (*message_server*) z wykorzystaniem protokołu **TLS**.

2.7.1. Generowanie certyfikatu klucza publicznego

Podczas nawiązywania komunikacji za pośrednictwem protokołu **TLS** (lub **SSL**), między klientem a serwerem przeprowadzany jest tzw. *handshake*. Aplikacje negocjują parametry połączenia (np. algorytm szyfrowania, używaną wersję protokołu TLS, itp.). Na tym etapie przeprowadzane jest również

uwierzytelnienie serwera[1]. Serwer potwierdza swoją tożsamość przesyłając do aplikacji klienta **certyfikat klucza publicznego**. Certyfikat zawiera:

- informacje o kluczu publicznym (najczęściej klucz algorytmu RSA);
- dane określające tożsamość podmiotu poddanego certyfikacji;
- podpis cyfrowy.

W przypadku standardu **certyfikatów X.509**, podpis cyfrowy składany jest przez odpowiedni urząd certyfikacji, zajmujący się wystawianiem certyfikatów cyfrowych. W ramach laboratorium wygenerowane certyfikaty zostaną podpisane samodzielnie, ponieważ będą wykorzystane wyłącznie na potrzeby testów komunikacji bezprzewodowej między aplikacjami.

W celu wygenerowania **certyfikatu klucza publicznego** można posłużyć się API biblioteki **OpenSSL**. Wywołując polecenie **openssl req** za pomocą konsoli systemowej, przesyła się żądanie podpisania **certyfikatu klucza publicznego**. Polecenie **openssl req** jest konfigurowane za pomocą dodatkowych parametrów wywołania, np.:

- **-in** *nazwa_pliku* – nazwa pliku, który ma zostać wczytany (jeśli nie zostały użyte opcje **-new** lub **-newkey**); jeśli parametr nie zostanie przekazany, to dane zostaną pobrane ze **standardowego wejścia**;
- **-out** *nazwa_pliku* – nazwa pliku, do którego ma zostać zapisany podpisany certyfikat; jeśli parametr nie zostanie przekazany, to dane zostaną przesłane na **standardowe wyjście**;
- **-keyout** *nazwa_pliku* – nazwa pliku, do którego ma zostać zapisany wygenerowany klucz prywatny;
- **-newkey** *argument* – wygeneruj nowy certyfikat i nowy klucz prywatny; przekazany *argument* definiuje m.in. długość i algorytm generacji klucza (np. **-newkey rsa:2048** w przypadku klucza algorytmu RSA o długości 2048 b);

[1] W niektórych implementacjach stosowane jest również uwierzytelnienie klienta

- **-x509** – wygeneruj samodzielnie podpisany **certyfikat klucza publicznego** (podpisany własnym kluczem prywatnym);
- ***-funkcja_skrótu_kryptograficznego*** – algorytm funkcji skrótu kryptograficznego używanej do podpisania certyfikatu (np. **-sha224** w przypadku algorytmu SHA224);
- **-days** *liczba_dni* – okres ważności wygenerowanego certyfikatu (w dniach);
- **-nodes** – nie szyfruj pliku zawierającego wygenerowany klucz prywatny.

Przykład: Wywołanie polecenia:

openssl req -newkey rsa:3072 -x509 -sha1 -days 30 -nodes -keyout private_key.pem -out cert.pem

poskutkuje wygenerowaniem nowego prywatnego klucza RSA o długości 3072 b do pliku **private_key.pem**. Plik **private_key.pem** nie będzie zaszyfrowany. Klucz prywatny posłuży do samodzielnego podpisania nowego certyfikatu przy użyciu funkcji skrótu kryptograficznego SHA1. Wygenerowany certyfikat zostanie zapisany do pliku **cert.pem**. Ustalony okres ważności certyfikatu wynosi 30 dni.

Ponieważ nie został określony żaden plik zawierający dane wejściowe, to **OpenSSL** zapyta za pośrednictwem **standardowego wejścia** o dodatkowe informacje określające tożsamość certyfikowanego podmiotu:

```
1 You are about to be asked to enter information that
    will be incorporated into your certificate
    request.
2 What you are about to enter is what is called a
    Distinguished Name or a DN.
3 There are quite a few fields but you can leave some
    blank
```

```
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PL
State or Province Name (full name) [Some-State]:
    Dolnoslaskie
Locality Name (eg, city) []:Wroclaw
Organization Name (eg, company) [Internet Widgits
    Pty Ltd]:Politechnika Wroclawska
Organizational Unit Name (eg, section) []:W12N
Common Name (e.g. server FQDN or YOUR name) []:Jan
    Kowalski
Email Address []:jan.kowalski@pwr.edu.pl
```

Aby wyświetlić wygenerowany certyfikat w postaci odkodowanej można posłużyć się poleceniem:

openssl x509 -text -noout -in *nazwa_pliku_certyfikatu*

W rezultacie otrzymamy:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0b:46:12:28:ca:9d:e9:e3:8b:2a:65:df:aa:
    ea:e9:1b:94:06:45:a5
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C = PL, ST = Dolnoslaskie, L =
    Wroclaw, O = Politechnika Wroclawska, OU = W12N,
    CN = Jan Kowalski, emailAddress = jan.
    kowalski@pwr.edu.pl
```

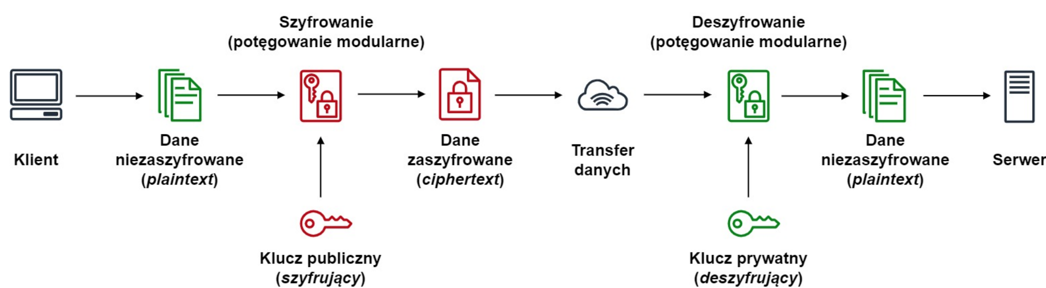
```
8           Validity
9               Not Before: Dec 18 15:34:18 2022 GMT
10              Not After : Jan 17 15:34:18 2023 GMT
11          Subject: C = PL, ST = Dolnoslaskie, L =
     Wroclaw, O = Politechnika Wroclawska, OU = W12N,
     CN = Jan Kowalski, emailAddress = jan.
     kowalski@pwr.edu.pl
12          Subject Public Key Info:
13              Public Key Algorithm: rsaEncryption
14                  Public-Key: (3072 bit)
15                  Modulus:
16                      00:ea:82:aa:e1:72:be:a8:47:1c:5e
     :3a:1f:4f:43:
17                      19:fd:f3:7d:47:41:8a:15:35:cb:d6
     :af:9b:15:5c:
18                      0e:4a:f4:26:98:91:28:35:6d:e5
     :98:e1:45:4d:c9:
19                      69:f7:ed:6f:ff:81:67:3a:05:5c:ae
     :e1:aa:d5:04:
20                      b7:1c:1f:d0:f4:7d:e5:0a:9d:ec:fa
     :b3:44:a2:58:
21                      d9:aa:3a:0d:12:57:5c:ad:a5:22:a8
     :be:99:c2:08:
22                      22:65:ab:52:c5:98:dc:4e:38:2e:3c
     :fb:45:b9:20:
23                      23:05:70:77:cf:75:ca:81:fb:97:ab
     :a4:fa:69:61:
24                      06:42:7b:8e:ab:f9:9c:e1:5d:c9:df
     :08:10:e6:ea:
25                      be:3b:db:bf:94:f4:fd:fd:af:4e:fa
     :7e:a7:dc:a9:
```

```
26                              63:1c:ba:92:20:74:0a:a9:e6:12:ab
    :35:ab:37:92:
27                              65:2b:02:1f:b7:be:e4:83:1e:47:fb
    :7f:53:16:fc:
28                              5f:6d:f0:37:a0:bb:d0:a9:6f
    :78:77:50:bf:9f:68:
29                              30:7e:d3:c9:fc:f8:4e:4c:96:5e:a4
    :b5:68:e6:79:
30                              09:0e:3a:ea:64:58:1d:8c:bb:3f
    :85:5c:5a:7d:9d:
31                              a7:5e:cd:b6:79:ea:5c:19:52:d2
    :08:62:0d:1c:81:
32                              6f:22:27:90:cb:94:af:91:77:8b:4a
    :08:4f:36:79:
33                              03:17:0f:ae:7d:8a:9e:59:5a:e0
    :50:e8:ab:67:6b:
34                              f6:04:d6:15:3b:c3:f4:09:01:73:3d
    :15:b7:f0:ec:
35                              76:7e:39:ba:72:65:3e:ab:c2:53:7c
    :46:8b:cb:cc:
36                              a8:a7:ce:a0:8b:76:11:0b
    :57:18:05:dd:fe:0c:86:
37                              bc:bf:3e:58:9f:7d:b6:13:82:cc:eb
    :ba:99:f8:53:
38                              7a:9e:bc:e2:43:43:c0:4b:4f:90:c8
    :68:0b:e7:9a:
39                              04:d6:87:95:34:9f:9c:0f:19:19:c6
    :5f:36:32:a2:
40                              db:c7:71:27:6a:84:5e:19:50:86:c5
    :e6:4d:ae:61:
41                              ea:f7:4f:9b:cf:b6:d2:74:77:ed
```

```
42                  Exponent: 65537 (0x10001)
43          X509v3 extensions:
44              X509v3 Subject Key Identifier:
45                  BC:D4:FF:FF:E6:E3:1A
    :65:09:40:03:16:28:CB:61:81:8C:7E:EF:15
46              X509v3 Authority Key Identifier:
47                  BC:D4:FF:FF:E6:E3:1A
    :65:09:40:03:16:28:CB:61:81:8C:7E:EF:15
48              X509v3 Basic Constraints: critical
49                  CA:TRUE
50      Signature Algorithm: sha1WithRSAEncryption
51      Signature Value:
52          e8:49:4c:29:0b:be:fa:e1:90:6d:b6:ef:87:a7:c9
    :35:53:52:
53          e0:76:06:cc:cc:52:19:e4:1f:75:fc:06:9d:0a
    :24:5b:e3:18:
54          ab:f7:31:5e:2f:63:d6:4d:bd:5e:5c:f2:af:7a:e6
    :60:7c:80:
55          6e:1a:a1:1f:40:8d:47:19:ba:e2:0a:c6:12:db
    :56:82:de:33:
56          55:61:ef:8a:74:a3:68:fc:9d:90:d1:35:5b
    :94:54:b8:66:e5:
57          7c:22:00:fc:bc:5b:2a:07:4f:f3:cb:3e:bd:e3:b9
    :80:40:2d:
58          ee:a8:8d:c3:a8:27:97:3d:90:9c:5f:28:db:a9:c5
    :aa:87:97:
59          1d:87:04:a8:44:05:bb:8b:3f:89:97:ea:31:6a:2e
    :23:e4:38:
60          6e:71:a9:9e:f6:17:05:56:49:1d:1f:76:eb:a5
    :66:68:c5:1f:
```

```
61          9d:21:93:b7:a5:19:97:28:94:5d:56:58:a9
    :09:83:57:a1:7d:
62          87:bb:e6:68:a9:c4:17:f7:a0:8f:9e:35:03:1d:6b
    :8c:84:ec:
63          73:f2:c5:16:0f:8e:03:c0:64:ec:fb:3c:5f:a7:ba
    :b4:ba:6e:
64          fe:74:10:86:47:e3:fe:a2:33:f4
    :98:93:43:29:56:f5:19:3d:
65          a0:a0:8f:a4:2b:84:d1:42:c8:b4:3c:2e:11:5d:c3
    :a1:d0:62:
66          f8:b2:05:09:dc:b5:6d:f0:7e:0f:8c:ba:96:17:5c
    :7e:b2:8d:
67          86:63:8e:d0:b7:39:c3:20:80:b2:b6:dc:55:74:f4
    :80:ee:8c:
68          f4:d9:1a:85:a4:c3:86:41:39:a4:9a:35:e0:7e:3d
    :cd:43:ff:
69          77:60:17:43:8c:5f:92:14:1f:d4:38:f2:d6:3a:1a
    :0b:47:23:
70          36:31:7a:c2:39:0f:6c:d0:7f:b4:df:d0:d4:99:7b
    :fe:5e:98:
71          1f:e8:2c:e0:de:29:e1:18:47:93:e2:06:51:f7
    :23:93:10:ef:
72          92:f5:30:15:40:c3:c6:29:79:da:33:1d:e5:dd
    :69:16:33:25:
73          68:c9:f1:2d:df:c1
```

Na rys. 2.2 pokazano ideę kryptografii asymetrycznej z wykorzystaniem kluczy algorytmu RSA. Klient wykorzystuje publiczny klucz serwera do zaszyfrowania przesyłanej wiadomości. Serwer odbiera dane w postaci zaszyfrowanej i wykorzystuje klucz prywatny (niejawny) do odszyfrowania wiadomości.



Rys. 2.2. Szyfrowanie danych z wykorzystaniem algorytmu RSA

Aby utworzyć żądanie podpisania certyfikatu (*ang. certificate signing request, CSR*) przez określony podmiot certyfikujący (*ang. certificate authority, CA*) należy pominąć parametr wywołania **-x509**.
Przykład: Wywołanie polecenia:

openssl req -newkey rsa:2048 -keyout my_key.pem -out signing_request.csr

poskutkuje wygenerowaniem nowego prywatnego klucza RSA o długości 2048 b do pliku **my_key.pem**. W trakcie generacji klucza użytkownik zostanie poproszony o hasło do zaszyfrowania pliku **my_key.pem**. Żądanie podpisania certyfikatu zostanie zapisane do pliku **signing_request.csr**. **OpenSSL** zapyta za pośrednictwem **standardowego wejścia** o dodatkowe informacje:

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
   Zachodni$Wiatr$Spienione$Goni$Fale666
An optional company name []:Samorzad studencki W12N
```

Atrybut ***challenge password*** stanowi sekret współdzielony między podmiotem wnioskującym o certyfikację, a urzędem certyfikacji, służącym do uwierzytelnienia.

W celu podpisania **certyfikatu klucza publicznego** przez swój własny podmiot certyfikujący można ponownie posłużyć się API biblioteki **OpenSSL**,

wywołując za pomocą konsoli systemowej polecenie **openssl x509**. Pierwszym krokiem jest utworzenie pliku zawierającego rozszerzenia, które mają zostać dodane do podpisywanego certyfikatu (np. pliku *cert.ext*) o zawartości:

```
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
subjectAltName = @alt_names
[alt_names]
DNS.1 = www.nowy_wspanialy_serwer.pl
```

Jako wartość *DNS.1* należy przypisać pełną nazwę domenową serwera (*ang. Fully Qualified Domain Name, FQDN*). Na potrzeby laboratorium należy podać *FQDN* hosta, które można pobrać wywołując z poziomu konsoli systemowej polecenie:

hostname –fqdn

Następnie, należy wywołać polecenie **openssl x509** skonfigurowane za pomocą dodatkowych parametrów wywołania:

- **-req** – przetwórz żądanie podpisania certyfikatu (*CSR*) jako dane wejściowe;
- **-CA** *nazwa_pliku* – nazwa pliku zawierającego certyfikat klucza publicznego podmiotu certyfikującego (*ang. CA certificate*);
- **-CAkey** *nazwa_pliku* – nazwa pliku zawierającego prywatny klucz podmiotu certyfikującego; ten klucz zostanie użyty do podpisania certyfikatu z przetwarzanego żądania;
- **-in** *nazwa_pliku* – nazwa pliku zawierającego żądanie podpisania certyfikatu (*CSR*);
- **-out** *nazwa_pliku* – nazwa pliku, do którego ma zostać zapisany podpisany certyfikat; jeśli parametr nie zostanie przekazany, to dane zostaną przesłane na **standardowe wyjście**;

- ***-funkcja_skrótu_kryptograficznego*** – algorytm funkcji skrótu kryptograficznego używanej do podpisania certyfikatu (np. **-sha224** w przypadku algorytmu SHA224);
- **-days** *liczba_dni* – okres ważności podpisanego certyfikatu (w dniach);
- **-CAcreateserial** – wygeneruj numer seryjny podmiotu certyfikującego (*CA serial number*);
- **-extfile** *nazwa_pliku* – nazwa pliku zawierającego rozszerzenia, które mają zostać dodane do podpisywanego certyfikatu.

Przykład: Wywołanie polecenia:

openssl x509 -req -CA ca_certificate.pem -CAkey ca_private_key.pem -in signing_request.csr -out server_certificate.pem -sha384 -days 90 -CAcreateserial -extfile cert.ext

poskutkuje przetworzeniem żądania podpisania certyfikatu klucza publicznego zawartego w pliku **signing_request.csr**. Podpis zostanie wykonany przy użyciu prywatnego klucza podmiotu certyfikującego zawartego w pliku **ca_private_key.pem**, przy użyciu funkcji skrótu kryptograficznego SHA384. Certyfikat klucza publicznego podmiotu certyfikującego zostanie pobrany z pliku **ca_certificate.pem**, natomiast ustalone rozszerzenia certyfikatu z pliku **cert.ext**. Podpisany certyfikat zostanie zapisany do pliku wynikowego **server_certificate.pem**. Ustalony okres ważności certyfikatu wynosi 90 dni.

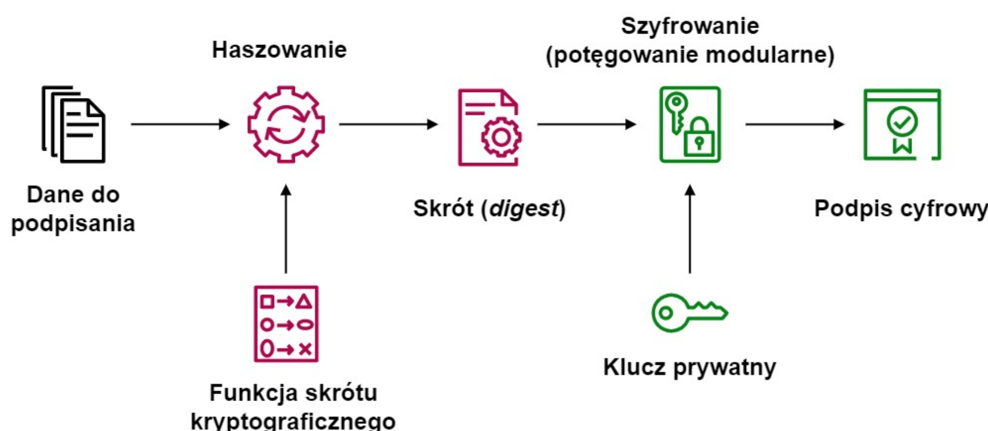
W celu weryfikacji podpisu cyfrowego można posłużyć się poleceniem **openssl verify**, np.:

openssl verify -CAfile ca_certificate.pem server_certificate.pem

Parametr **-CAfile** wymaga przekazania nazwy pliku zawierającego **certyfikat klucza publicznego** podmiotu certyfikującego. Drugi argument wywołania stanowi nazwa pliku zawierającego weryfikowany podpisany certyfikat. W przypadku poprawnej weryfikacji zostanie wyświetlony komunikat:

server_certificate.pem: OK.

Na rys. 2.3 przedstawiono mechanizm generowania podpisu cyfrowego z wykorzystaniem kluczy algorytmu RSA. Dane przeznaczone do podpisania (np. zawartość **certyfikatu klucza publicznego**) są wstępnie przetwarzane przez **funkcję skrótu kryptograficznego** (np. funkcje z rodziny SHA). Jest to jednokierunkowa operacja **haszowania**, w wyniku której powstaje nieodwracalny skrót danych wejściowych (*ang. digest*). **Podpis cyfrowy** powstaje w wyniku szyfrowania skrótu przy użyciu prywatnego klucza RSA.



Rys. 2.3. Generowanie podpisu cyfrowego z wykorzystaniem algorytmu RSA

2.7.2. Instalacja OpenSSL

Instalację biblioteki oraz narzędzi **OpenSSL** w systemie operacyjnym Linux można przeprowadzić za pomocą konsoli systemowej, wywołując polecenie:

sudo apt-get install openssl

Również wiele serwisów internetowych oferuje dostęp do API **OpenSSL** z poziomu przeglądarki internetowej, np. `https://www.cryptool.org/en/cto/openssl`, dzięki czemu możliwe jest korzystanie z narzędzi kryptograficznych bez konieczności ich instalacji w systemie operacyjnym.

2.8. Analiza komunikacji sieciowej

Pakiety wymieniane między aplikacjami w ramach komunikacji bezprzewodowej można obserwować korzystając z darmowego oprogramowania **Wireshark**. Jest to tzw. *sniffer*, czyli aplikacja przechwytująca i analizująca komunikację sieciową. Instalację programu można przeprowadzić z poziomu konsoli systemowej wywołując polecenie:

sudo apt install wireshark-qt

Uruchomienie aplikacji jest realizowane poleceniem:

sudo wireshark

Aby rozpocząć przechwytywanie komunikacji sieciowej należy nacisnąć przycisk **Start capturing packets**. W celu ograniczenia zakresu wyświetlania komunikacji sieciowej można posłużyć się filtrami (**Apply display filter**), np.:

- **tcp** – wyświetla tylko pakiety protokołu TCP;
- **ip.addr == 192.168.0.21** – wyświetla tylko pakiety wymienione z węzłem o adresie IP 192.168.0.21;
- **ip.addr == 192.168.0.193 && tls** – wyświetla tylko pakiety protokołu TLS wymienione z węzłem o adresie IP 192.168.0.193.

Aby obserwować wymianę pakietów protokołu TLS należy dodać obserwowany port TCP do portów SSL/TLS. W tym celu należy skonfigurować ustawienia programu **Wireshark**:

Edit → Preferences → Protocols → HTTP → SSL/TLS Ports

2.9. Zintegrowane środowisko programistyczne Qt Creator

Proponowanym zintegrowanym środowiskiem programistycznym (*IDE*) do pracy na zajęciach laboratoryjnych jest **Qt Creator**. Program można uruchomić za pomocą konsoli systemowej wywołując polecenie:

qtcreator

Otwarte zostanie okno główne programu. Aby zaimportować projekt *CMake* należy z paska menu wybrać **File** ⟶ **Open File or Project...** W oknie wyboru projektu należy przenieść się do katalogu repozytorium (**messageserver**) i zaznaczyć plik **CMakeLists.txt**, a następnie nacisnąć przycisk **Open** znajdujący się w prawym górnym rogu okna. W oknie konfiguracji projektu należy określić ustawienia narzędzi budowania aplikacji w środowisku **Qt Creator**. Zaznaczając opcję **Select all kits** wybieramy wszystkie dostępne konfiguracje. Kliknięcie przycisku **Configure Project** spowoduje wczytanie źródeł projektu i otwarcie okna edycji plików.

3. Program ćwiczenia

Uwaga: Warunkiem dopuszczenia do realizacji laboratorium jest przedstawienie rozwiązania Zadania 1. na początku zajęć.

Uwaga: Zadania należy realizować w kolejności numerycznej.

Zadanie 1. (Zadanie domowe) Korzystając z biblioteki **OpenSSL** wygeneruj nowy **prywatny klucz RSA** oraz samodzielnie podpisany **certyfikat klucza publicznego** (certyfikat podpisany własnym kluczem prywatnym). Wygenerowany certyfikat oraz klucz prywatny posłużą podpisaniu certyfikatu klucza publicznego aplikacji serwera wiadomości (**Zadanie 2.**). Dodatkowe warunki:

- długość klucza RSA wynosi 4096 b;
- podaj swoje dane jako dane określające tożsamość właściciela certyfikatu;
- certyfikat jest podpisany wygenerowanym kluczem prywatnym RSA z wykorzystaniem funkcji skrótu kryptograficznego SHA256;
- okres ważności wygenerowanego certyfikatu wynosi rok.

Uwaga: Wygenerowany klucz prywatny oraz certyfikat są niezbędne do realizacji Zadania 2.

Zadanie 2. (na ocenę 3.0) Skonfiguruj i przetestuj szyfrowaną komunikację między aplikacjami graficznego interfejsu użytkownika i serwera wiadomości z wykorzystaniem protokołu TLS. W tym celu:

1. W głównym katalogu repozytorium projektu serwera wiadomości (**messageserver**) utwórz nowy katalog (np. **auth**)

2. Skopiuj pliki zawierające prywatny klucz RSA oraz certyfikat klucza publicznego, wygenerowane w ramach **Zadania 1.**, do nowo utworzonego katalogu. Skopiowany klucz i certyfikat definiują podmiot certyfikujący (CA)

3. Wygeneruj nowy prywatny klucz RSA o długości 2048 b oraz żądanie podpisania certyfikatu klucza publicznego dla aplikacji serwera wiadomości

4. Podpisz certyfikat klucza publicznego serwera wiadomości wykorzystując prywatny klucz RSA podmiotu certyfikującego (CA), wygenerowany w ramach **Zadania 1.** Jako funkcję skrótu kryptograficznego wykorzystaj algorytm SHA256. Ustal ważność wygenerowanego certyfikatu na 31 dni

5. Zweryfikuj podpis cyfrowy na certyfikacie serwera wiadomości korzystając z polecenia **openssl verify**

6. Skonfiguruj aplikację serwera wiadomości do obsługi komunikacji szyfrowanej (SSL) w trybie **ECHO**

7. W głównym katalogu repozytorium projektu graficznego interfejsu użytkownika (**messengergui**) utwórz nowy katalog (np. **auth**)

8. Skopiuj plik zawierający certyfikat klucza publicznego, wygenerowany w ramach **Zadania 1.**, do nowo utworzonego katalogu. Skopiowany certyfikat stanowi zaufany certyfikat klucza publicznego podmiotu certyfikującego serwer wiadomości (CA)

9. Skonfiguruj aplikację graficznego interfejsu użytkownika do nawiązywania komunikacji szyfrowanej

10. Przetestuj nawiązywanie połączenia między aplikacjami. Rozpatrz następujące przypadki:

Przypadek testowy 1.

- aplikacja graficznego interfejsu klienta skonfigurowana do nawiązywania komunikacji szyfrowanej;
- aplikacja serwera wiadomości skonfigurowana do nawiązywania komunikacji szyfrowanej;
- certyfikat klucza publicznego serwera wiadomości posiada poprawny podpis cyfrowy złożony z wykorzystaniem prywatnego klucza RSA podmiotu certyfikującego

Przypadek testowy 2.

- aplikacja graficznego interfejsu klienta skonfigurowana do nawiązywania komunikacji szyfrowanej;
- aplikacja serwera wiadomości skonfigurowana do nawiązywania komunikacji szyfrowanej;
- **podpis cyfrowy złożony na certyfikacie klucza publicznego serwera wiadomości jest błędny (zmodyfikuj wartość pola *Signature Value* na certyfikacie klucza publicznego serwera wiadomości)**

W protokole z przebiegu ćwiczenia zapisz wnioski z przeprowadzonych testów.

Zadanie 3. (na ocenę 4.0) Wykorzystując załączoną formatkę, napisz dokumentację klasy **networking::SslSocket** biblioteki **networking**. W pliku **TcpSocket_Documentation.pdf** zamieszczono przykład dokumentacji klasy **networking::TcpSocket** biblioteki **networking**. W protokole z przebiegu ćwiczenia zapisz najważniejsze informacje potrzebne do wypełnienia formatki dokumentacji.

Zadanie 4. (na ocenę 5.0) Uruchom aplikację serwera wiadomości, a następnie wykorzystując program **Wireshark** zaobserwuj pakiety protokołu TLS wymieniane między serwerem a aplikacją graficznego interfejsu użytkownika. Na ich podstawie:

- wykorzystując diagram sekwencji UML zaprezentuj jak przebiega nawiązywanie połączenia szyfrowanego (*handshake*) między aplikacjami
- odczytaj wynegocjowaną wersję protokołu TLS
- odczytaj wynegocjowany algorytm szyfrowania komunikacji

W protokole z przebiegu ćwiczenia zapisz zaobserwowane pakiety TLS, ich nagłówki oraz informację o kierunku komunikacji (klient → serwer / serwer → klient). **Uwaga: program wireshark nie umożliwia obserwacji pakietów wymienianych w ramach hosta lokalnego (adres IP 127.0.0.1).**

W sprawozdaniu zawrzeć:

- protokół z przebiegu ćwiczenia;
- najważniejsze cechy protokołów SSL i TLS;
- wszystkie polecenia API OpenSSL użyte do wygenerowania, podpisania i weryfikacji certyfikatu klucza publicznego serwera wiadomości;
- wnioski z realizacji **Zadania 2.** (przeprowadzone testy, analiza i uzasadnienie otrzymanych wyników);
- dokumentację klasy **networking::SslSocket** biblioteki **networking** [*jeśli zostało zrealizowane **Zadanie 3.***];
- diagram sekwencji UML prezentujący *handshake* między aplikacją graficznego interfejsu użytkownika a aplikacją serwera wiadomości [*jeśli zostało zrealizowane **Zadanie 4.***];
- wynegocjowaną w ramach nawiązywania połączenia szyfrowanego wersję protokołu TLS oraz wynegocjowany algorytm szyfrowania komunikacji [*jeśli zostało zrealizowane **Zadanie 4.***].