TECHNICAL UNIVERSITY OF LIBEREC

Faculty of Mechatronics, Informatics and Interdisciplinary Studies

Course of study: N2612 Electrical Engineering and Informatics

Field of study: 2612T071 Engineering of Interactive Systems

Storing and transferring data for mobile applications in online and offline mode

Ukládání a výměna dat u mobilní aplikace v datovém online a offline režimu

Diploma thesis

Author: **Martin Pelák**

Supervisor: Ing. Igor Kopetschke

Date of submission: 17.5.2013

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce.

Datum

Podpis

Poděkování

Chtěl bych poděkovat všem, kteří mi pomáhali s tvorbou diplomové práce. V první řadě bych chtěl poděkovat vedoucímu této diplomové práce Ing. Igor Kopetschkemu za jeho čas, poskytnuté rady a materiály.

Abstract

This diploma thesis deals with a problem of online and offline data transfer among mobile devices. The main aim is to find possible ways on how to transfer data securely. The theoretical part suggests possible solutions using public-key cryptography. It describes how to use certificates signed by a certificate authority or how to use self-signed certificates for the secure connection. Furthermore, it introduces a certificate-free solution based on manual data encryption.

Moreover, it describes and explains terms related to problems such as certification authority, digital signature and introduces tools for managing certificates.

The practical part of the diploma thesis is based on the theoretical part and it implements all suggested solutions into two demo applications. Each of them was created for different mobile operating system, one for Windows Phone 8 and the other one for Android.

Key words

Mobile application

Encrypted communication

Data storing

Android

Windows Phone 8

SSL

SMS

Table of Contents

List of figures

List of abbreviations

API – Application Programming Interface

CA – Certification Authority

CSR – Certificate Signing Request

GPS – Global Positioning System

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

NFC – Near Field Communication

SAOP – Simple Object Access Protocol

SHA1, SHA2 – Secure Hash Algorithm

SIM – Subscriber Identity Module

SMS – Short Message Service

SSL – Secure Sockets Layer

SQL – Structured Query Language

TSL – Transport Layer Security

USB – Universal Serial Bus

WP8 – Windows Phone 8

WSDL – Web Services Description Language

XML – Extensible Markup Language

Yii – Yes It Is!

1 Introduction

The mobile devices are under massive development. As the level of integration grows, the mobile devices are smaller, faster and offer new possibilities. These days, devices such as mobile phones or tablets integrate multiple devices into one. Apart from their primary purposes, which are text messaging, phone calls, internet access, Bluetooth and many others, they can be as well used as hand held consoles, GPS navigation devices or photo cameras. For these reasons, they are often referred to as smart devices.

Users may download free and paid applications and extend functionality of their devices. Nowadays, a smart mobile phone equals a smaller, but powerful computer.

The Internet is moving from ordinary computers into our hands. The data transfer becomes a crucial matter. The users login into various internet services, make online payments and transfer other sensitive personal data. Therefore, it is necessary to deals with a problem of security and to protect the data transfer against possible hacker attacks. Unfortunately, it is not possible to rely on functions provided by neither the internet provider nor the mobile operator.

Developers of mobile applications have to think of the secure data transfer and secure data storing. Different mobile devices offer different challenges. The diploma thesis investigates possible solutions on how to make safe transfers among mobile devices in online and offline mode. Mainly, it focuses on Android and Windows Phone 8 operating systems.

The diploma thesis describes a cryptographic method called public-key cryptography which is also referred to as asymmetric key cryptography. Furthermore, it deals with possible security risks such as the man-in-the-middle attack.

The next part of the diploma thesis introduces problems of authentication and digital certificates. As well as tools for creating and managing them.

The most of the described solutions were integrated into two demo applications. Each of them was developed for a different operating system.

2 Data transfer

A mobile device without any possibility of retrieving or providing data would make no use. All devices have some inputs and some outputs.

There are different producers of mobile devices. Their products (mobile phones, tablets) differ in many parameters.

The variety of I/O is quite wide. Each portable device is able to receive data via sensors such as buttons, microphone, touch screens, g-sensor, GPS, barometer, cameras, etc. Wireless technologies such as NFC, Bluetooth or Wi-Fi are not sensors in the traditional sense, but they are very often used as well. Another possibility is to use available ports (USB port) or memory cards.

In spite of the broad range of possible choices, only a limited number of them is suitable for a data exchange among mobile devices. Because not all of them are capable of a data sending.

A data transfer based on physical media exchange (memory cards, memory sticks) does not provide a needed operativeness and comfort. For example, it is not suitable for fast or long-distance data transfers.

It may be assumed that the vast majority of everyday users, who need to transmit data, do so via the Internet connection. Therefore, the further text will focus as well as on a data transfer among mobile devices on the Internet. However, it will take into account a communication among mobile devices and any device connected to the Internet.

Any communication based on the Internet will be considered as an online communication. Any other as an offline communication.

3 Data transfer security

Nearly any communication can be listened in on by anyone. Therefore, it should be made sure that intercepted messages are not understandable for anybody else but the client and server.

The protocol HTTP is basically used for transferring almost everything that ordinary internet users know as "WWW". Internet users interact with webservers via their web browsers. If a user enters a web address or fills in a login form with a username and password, their browser sends a request to the web server. The server's response is processed and displayed to the user.

The HTTP itself is not capable of data encoding and it transmits data as a plain text. For this reasons it should not be used for transferring sensitive data such as passwords, credit card numbers, etc. In order for HTTP to be secure, it is usually combined with the TLS/SSL protocol. TLS/SSL is placed between the transport (TCP/IP) and the application (HTTP) layer. This combination is known as HTTPS.

3.1 Hypertext Transfer Protocol Secure (HTTPS)

It allows a secure data exchange between a client and server. HTTPS is a designation for HTTP on the top of the TLS/SSL protocol. HTTPS enables to verify identities of both communicating parties and it is also capable of a data encryption.

The most common place where the HTTPS protocol can be found is an internet browser. The Figure 1 demonstrates an address bar in Google Chrome browser. The green lock indicates that the connection is secure and the server's identity was verified. In this particular case it is not necessary to verify client's identity. A search engine (Google) does not need to know who it communicates to.

Figure 1: HTTPS in Google Chrome

In comparison with HTTP, it is a little bit slower. For a big data transfer which does not require data encryption, it is better to use HTTP.

HTTPS URLs begin with "https://" whereas HTTP URLs begin with "http://". Both protocols are different in default port number. It is 443 for HTTPS and a number 80 for HTTP.

3.2 Asymmetric key cryptography

It is a cryptography method. For encryption and decryption are used different keys. That is the main difference against the symmetric cryptography where only one key is used for a data encryption and decryption.

Each member (client and server), participating in communication, has to have generated a pair of cryptographic keys – an encryption public and a decryption private key. See Figure 2. Before the client and the server move onto transmitting sensitive data, they exchange public keys. It is crucial to keep the private key in secret.

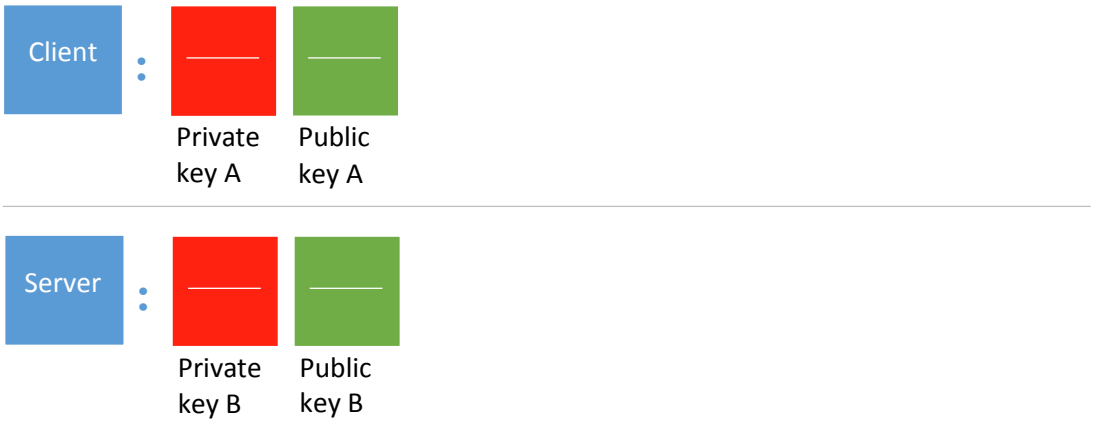

Figure 2: Each side has a pair of keys

In the client-server communication, the client uses server's public key to encrypt data. Therefore, the server is the only one capable of decrypting the data because nobody else knows the linked private key. The server encrypts data using the client's public key. It is only the client who knows the way on how to decrypt the data.

According to the description above it results that both keys are mathematically linked. Figure 3 illustrates a process of data exchange between the client and the server.

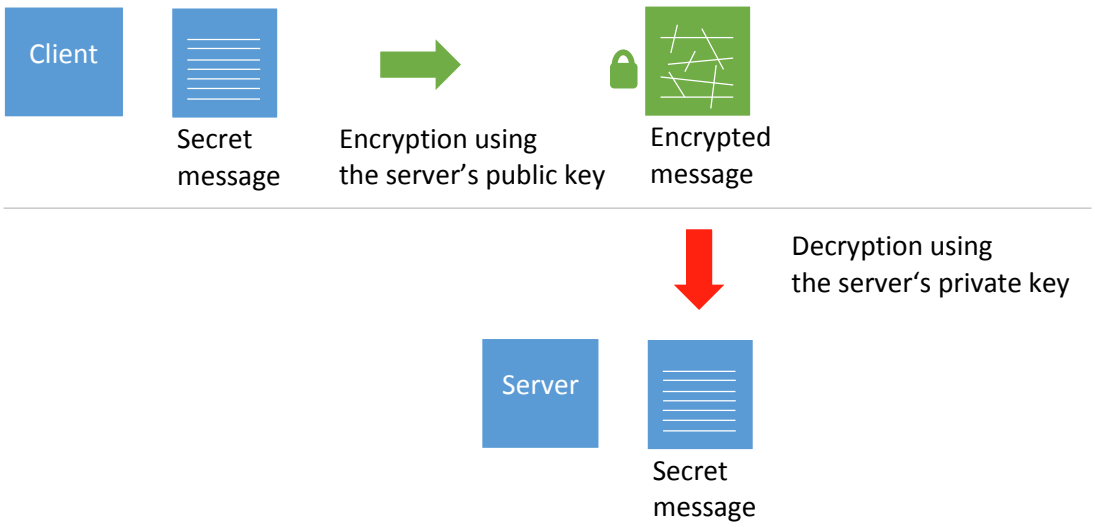

Figure 3: the process of message encryption and decryption

3.3 The man-in-the-middle

The communication based on a key exchange as it was described so far, may suffer from an attack called "the man-in-the-middle".

In such case, an attacker (man) intercepts messages between two parties. He may read, modify and retransmits them so that a client and a server do not know about the attacker's existence. The Figure 4 demonstrates the position of the attacker.

Figure 4: The man in the middle

The table below illustrates a possible message exchange between the client and the server with the man in the middle.

1. Client's key is intercepted by the Man. He replaces the client's key with his own.

Client	*->[Client's public key]*	**Man**	-> [Man's public key]	**Server**

2. The Man replaces the server's public key

Client	[Man's public key] <-	**Man**	[Server's public key] <-	**Server**

3. Because the client's secret message was encrypted by man's public key, the Man decrypts it and reads a secret content. Finally, he encrypts the message again and sends it to the server.

Client	-> "secret message"	**Man**	-> "secret message"	**Server**

4. This is the same scenario as number 3 in the opposite direction. The client and the server believe that their communication is secure.

Client	"secret message" <-	**Man**	"secret message" <-	**Server**

The attacker may succeed only if there is no user authentication. The earlier mentioned cryptography protocol SSL already contains authenticate method based on a mutually trusted certification authority and as such prevents from "the man-in-the-middle" attack.

4 Public key certificate

In the asymmetric cryptography, a public key certificate is a digitally signed document which binds a public key and information about an owner of the key. It also contains information about its issuer. The most commonly used type of certificates is X.509. The X.509 v3 certificate standard is specified in RFC 5280 [1].

Each certificate is issued by a certificate authority (CA). Before the certificate authority digitally signs the certificate, it has to verify all the information stated in the certificate. Credibility of certificate authorities differs and to a certain extent depends on the amount of information that CA requires from its applicants.

Certificates issued by commercial companies (e.g. VeriSign, Thawte, ipsCA, Comodo...) are paid. However, it is possible to obtain a certificate for free (e.g. CAcert, StartSSL).

On condition that the certificate authority is trusted, the owner of the certificate may be trusted as well. Because the certificate authority signs the certificate by its private key, a receiver of the certificate needs to know CA's public key. One possibility is to manually install a CA's root certificate, the other one is to make a use of trusted root certificates which are preinstalled in an operation system or come with web browsers.

It is very convenient for customers of CAs to have their certificates signed by authority having its root certificate included in as many browsers as possible. A certificate signed by such authority is automatically trusted and a web browser user is not bothered with a warning that the server's certificate is untrusted (Figure 5).

Figure 6 illustrates two dialog boxes provided by Firefox browsers after opening two websites. The first one uses a certificate signed by CA and the other one uses a self-signed certificate.

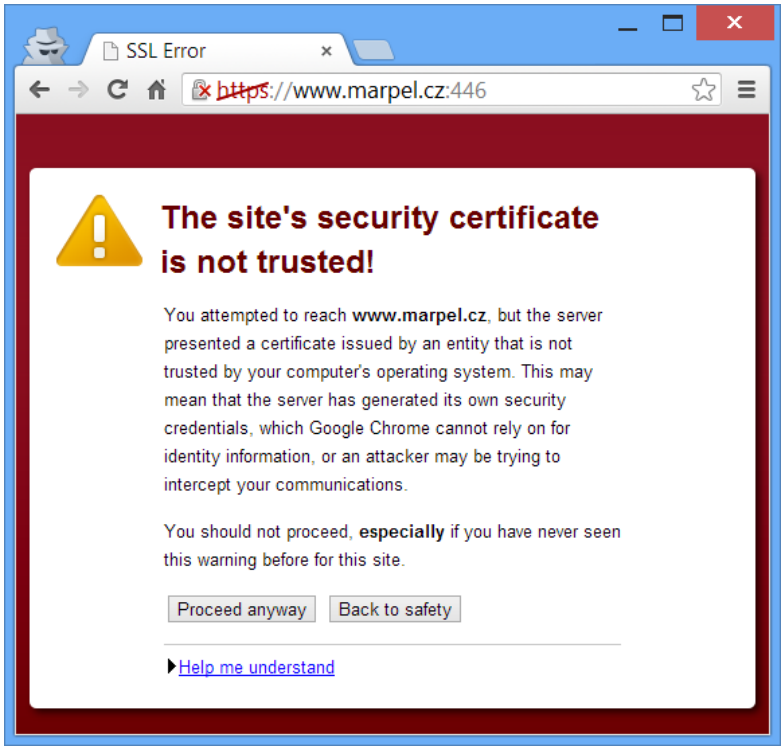

Figure 5: A warning saying that the certificate is not trusted

A certificate signed by an entity, which the certificate verifies, is called a self-signed certificate.

Commonly used extensions for storing X.509 certificates are as follows:

- `.pem` – usually Base64 .der certificate
- `.cer`, `.crt`, `.der` – usually in binary .der format
- `.p7b`, `.p7c` – PKCS#7 SignedData structure without data, just certificate(s) or CRL(s)
- `.p12` – PKCS#12, it may contain certificates and private keys
- `.pfx` – predecessor of PKCS#12 (.p12)

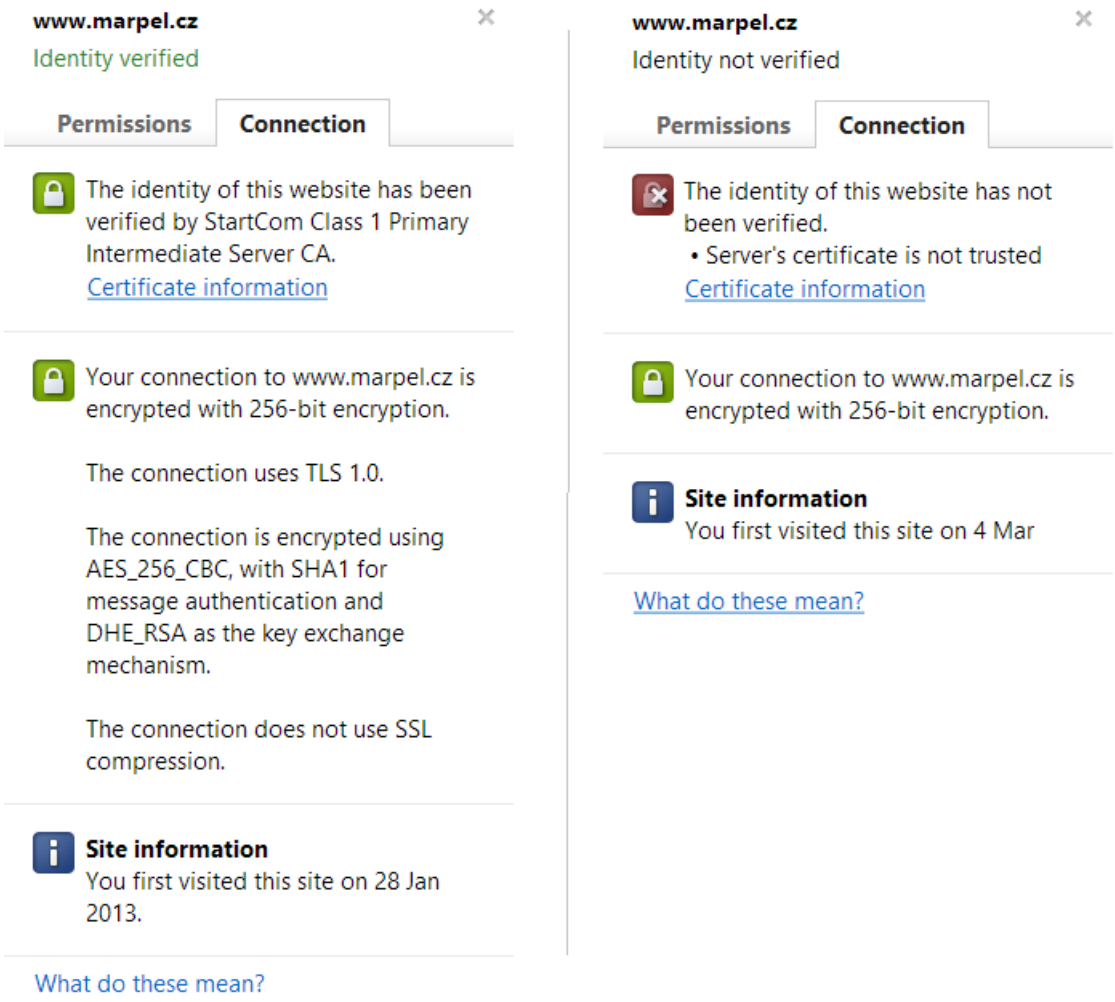

Figure 6: comparison of trusted and untrusted certificate

5 Digital signature

A digital signature, also known as a cryptographic signature, is a piece of data attached to a message. It is a digital equivalent of a real signature. It authenticates the identity of a signer of a document and ensures that the original content has not been changed.

A digital signature is generated for every message that needs to be signed. This procedure consists of several steps

1. Create a hash of original data
2. Encrypt the hash using a private key
3. Attach a signer's certificate

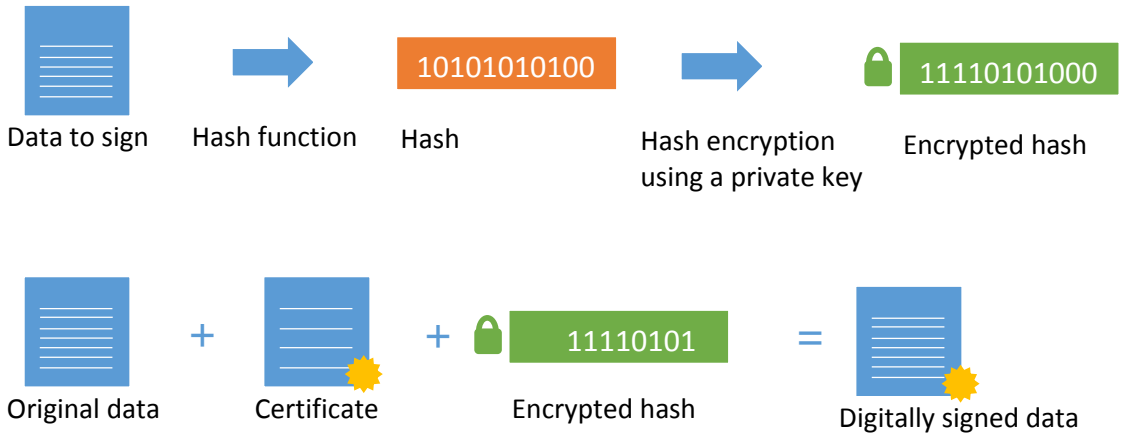

Figure 7: signing a document

In order to verify the signature, a receiver of the message has to generate a message hash. Then they decrypt the attached hash using a public key which is placed in the certificate. If the hashes match, the signature is valid.

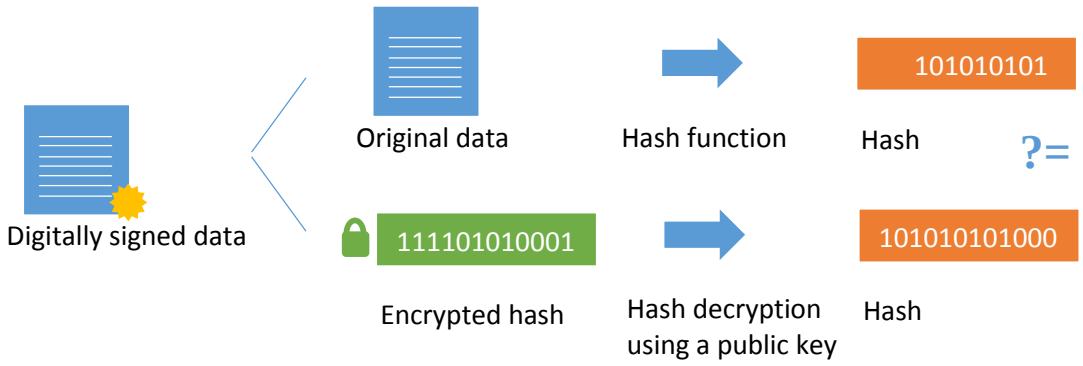

Figure 8: signature validation

Following algorithms are related to digital signatures

- RSA (Rivest-Shamir-Adleman)
- DSA (Digital Signature Algorithm)
- One-way cryptographic hash functions
    - MD5 (Message Digest 5)
    - SHA (Secure Hash Algorithm)

6 Transport Layer Security

Transport Layer Security (TLS) and its older version, Secure Socket Layer (SSL), are cryptographic protocols which secure communication between communicating applications. The TLS protocol consists of two layers: the TLS Record protocol and the TLS Handshake protocol. Their position according to the TCP/IP model illustrates Figure 9.

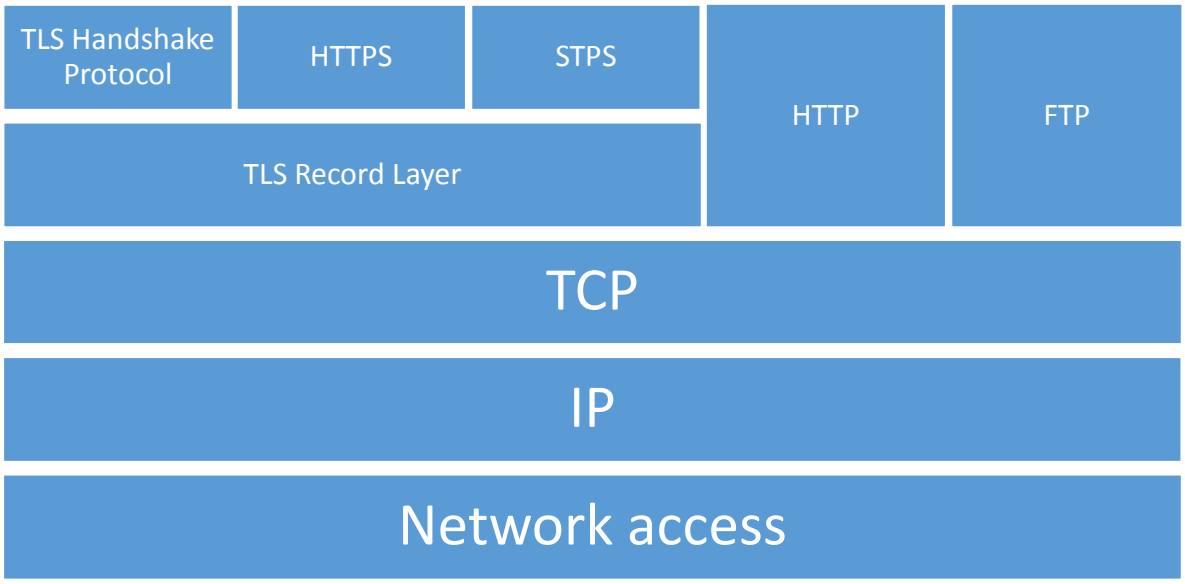

Figure 9: position of TLS protocols

The TLS Record protocol provides connection security and it is used for encapsulation of higher-level protocols.

"TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data." [2]

The advantage of TLS is that the application protocol is independent. Higher-level protocols may lie on the top of TLS (e.g. HTTPS, SFPT…).

TLS includes three main phases:

1. An agreement of participants on supported algorithms
2. A key exchange based on public key encrypting and authentication relying on certificates
3. A data encryption by a symmetric cipher

The client and server agree on cryptographic algorithms during the first phase. The present implementation allows using following options:

- For public key encryption: RSA, Diffie-Hellman, DSA
- For symmetric encryption: RC2, RC4, IDEA, DES, Triple DES, AES, Camellia
- For one-way hashes: Message-Digest algorithm (MD2, MD4, MD5), Secure Hash Algorithm (SHA-1, SHA-2)

7 Certificate management

The OpenSSL is an open-source security library. It is an implementation of the SSL and TLS cryptographic protocols. It offers a command line tool and it can be used for

- Creating, managing and converting private and public keys
- Creating X.509 certificates
- Using cryptography hash function
- Data encrypting and decrypting

The core of this library is written in C programming language. Different versions are available especially for Unix-based systems (Mac OS X, Linux…). However, it is possible to find versions for Microsoft Windows.

7.1 Practical OpenSSL Usage

A process of generation a self-signed certificate consists of several steps. Firstly, it is necessary to generate a private key. Then create a certificate signing request (CSR) and finally merge all into a desired self-signed certificate.

All commands are input into a command line.

Generating a Private Key

```
OpenSSL> genrsa -out private.key 4096
```

This command generates an RSA private key with 4096 bit long modulus. The output file is called a *private.key* and the key is stored in PEM format which is readable as ASCII text.

Generating a CSR (Certificate Signing Request)

```
OpenSSL> req -new -key private.key -out server.csr -config
./../openssl.cnf
```

CSR is a text containing information that will be included in the certificate such as distinguished name (DN), organization name and other important data. This text is usually sent along with a public key to certified authority which has to verify all information. Then it is allowed to generate an SSL certificate.

After inserting the command above, a user is prompt to enter data about the applicant.

```
Loading 'screen' into random state - done
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
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]:CZ
State or Province Name (full name) [Some-State]:Czech
Republic
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty
Ltd]:www.marpel.cz
Organizational Unit Name (eg, section) []:www.marpel.cz
Common Name (eg, YOUR name) []:www.marpel.cz
Email Address []:info@marpel.cz

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:ychat
An optional company name []:www.marpel.cz
```

Generating a Self-Signed Certificate

```
OpenSSL> x509 -req -days 365 -in server.csr -signkey
private.key -out server.crt
```

This command generates a self-signed certificate. It will not be signed by any certification authority. The certificate is stored in a file called *server.crt*.

```
Loading 'screen' into random state - done
Signature ok
subject=/C=CZ/ST=Czech
Republic/O=www.marpel.cz/OU=www.marpel.cz/CN=www.marpel.cz/em
ailAddress=info@marpel.cz
Getting Private key
```

Getting information about certificate

```
OpenSSL> s_client -connect www.marpel.cz:444
OpenSSL> x509 -text -in server.crt
OpenSSL> pkcs12 -info -in openssl_key_crt.p12
```

The commands above print all available information about certificates stored in various locations.

Converting a certificate to .p12 format

```
OpenSSL> pkcs12 -export -inkey private.key -in
server.crt -out openssl_key_crt.p12 -name
www.marpel.cz
```

This command merges a private key and a certificate into one file.

```
Loading 'screen' into random state - done
Enter Export Password:
  Verifying - Enter Export Password:
```

Removing a pass from a private key

```
OpenSSL> rsa -in private.key -out
private_without_pass.key
```

This is another command which is very often used while handling certificate. It removes a passphrase and save the primary key to a new file.

8 Keystores

A keystore is a repository of security certificates. Different platforms store keys and certificates in different ways.

8.1 Managing keystores

Portecle is a GUI application for managing keys, keystores, certificates, certificate signing requests and other things related to certificates.

In particular, it is very useful for managing keystores.

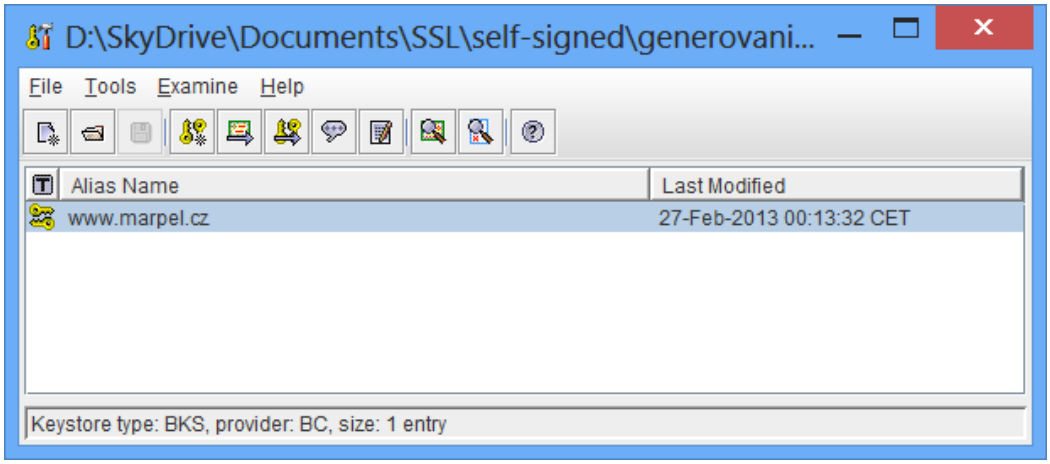

Figure 10: the main screen of Portecle program

According to the Portecle's documentation [3], it is able to manage following keystores:

- **JKS**: Java Keystore (Oracle's Keystore format)
- **PKCS #12**: Public-Key Cryptography Standards #12 Keystore (RSA's Personal Information Exchange Syntax Standard)
- **JCEKS**: Java Cryptography Extension Keystore (More secure version of JKS)
- **JKS (case sensitive)**: Case sensitive JKS
- **BKS**: Bouncy Castle Keystore (Bouncy Castle's version of JKS)
- **UBER**: Bouncy Castle UBER Keystore (More secure version of BKS)
- **GKR**: GNU Keyring keystore

9 Public-key cryptography

9.1 RSA algorithm

RSA is a public-key cryptography algorithm which takes advantage of presumed difficulty of factoring large integers known as the factoring problem.

9.1.1 Key generation

A pair of keys for RSA algorithm is generated in the following way:

1. Choose two pseudo random prime numbers *p* and *q*.
2. Compute their product *n=p*q*
3. Compute Euler's totient function $\phi(n) = (p - 1)*(q - 1)$.
4. Choose integer *e* smaller then $\phi(n)$, *e* and $\phi(n)$ are coprime.
5. Determine number *d* so that $de \equiv 1 \pmod{\phi(n)}$

There are two derived equations for data encryption and decryption

$c = m^e \bmod n$

$m = c^d \bmod n$.

9.1.2 Example

1. p=47, q=31
2. n=p*q=47*31= 1457
3. $\phi(1457) = (p - 1)*(q - 1) = 46*30 = 1380$
4. e=7, gcd(1380, 7) = 1
5. d=1183

Message to encrypt

m = 28

A process of encryption

$c = m^e \bmod n = 28^7 \bmod 1457 = 1192$

A process of decryption

$m = c^d \bmod n = 1192^{1183} \bmod 1457 = 28$

10 Network communication

There are several options which may be used. The following part of the diploma thesis introduces some possible solutions.

10.1 Socket communication

A socket is a software endpoint of bidirectional communication between two programs running on a network. Sockets, based on communication using the internet protocols, are called Internet sockets.

On start-up, a server creates sockets which are in a listening state and waiting for a connection request from a client. In order for the client to be able to contact the server, it needs to know the servers IP address and a port number that the server is listening on. The client also has to provide some information about itself. It binds a local port number and its IP address.

Each endpoint consists of a local IP address and a port number [4]. The combination of a server and a client endpoints is called socket pair. That means that every socket is made up of 4 parts; a local and remote IP addresses and port numbers.

A TCP server is usually able to server several clients simultaneously by creating child processes and establishing the connection between the client and the child process. It also applies for multiple connections on the same port.

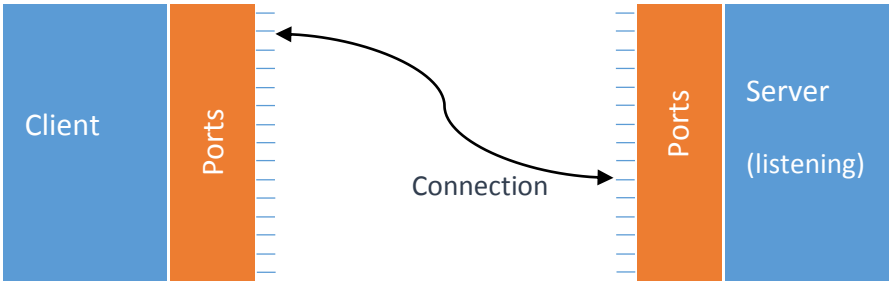

Figure 11: socket communication

Once the connection is established, both communicating parties may start a data exchange. A socket communication does not require a dedicate server and therefore it may be used even for peer-to-peer communication.

10.2 Remote procedure call

Remote procedure call (RPC) is technology allowing calling a procedure which may be stored on a completely different machine. It enables to run time-consuming operations on servers or retrieve data from remote server.

The main advantage is that the client does not need to know how the procedure is implemented and uses it as a black box. On the contrary, the client relies on the fact that the server is available and the internet is up and running.

XML-RPC is a protocol for calling remote procedures. Basically, it is a set of rules which tells you how to use already standardized technologies for RPC. Data is encapsulated using a XML (eXtensible Markup Language) and transferred over HTTP(S). It allows programmes written in different languages to use it.

The development of this protocol does not continue any longer. However, it was used as a base for SOAP.

10.3 Common Object Request Broker Architecture

(COBRA) Common Object Request Broker Architecture is a standard which allows distributed programs running on different computers to behave as one unit. COBRA applications operate with a term *object* which is a piece of running software. Typically, it is an instance of a singleton object.

Clients may invoke operations only on objects which are wrapped in OMG IDL (interface definition language). Such interface specifies the way objects are presented to the outside world. The client program may be an object itself.

The interface is written by a developer of the object. COBRA is then responsible for mapping (translating) from IDL to a specific language. It is able to map the interface into the most popular languages such as C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript. However, there is an unofficial support for other languages. Therefore, objects may be written in various programming languages on different operating systems.

Objects in the same network interact with each other through an object request broker (ORB). The ORB allows a client to request a service from any server without having to know where the server is situated or what programming language the server program is written in.

It is also possible to realize a request or to return a response between ORBs. Programs would use General Inter-ORB Protocol (GIOP) or Internet Inter-ORB Protocol (IIOP) for the Internet.

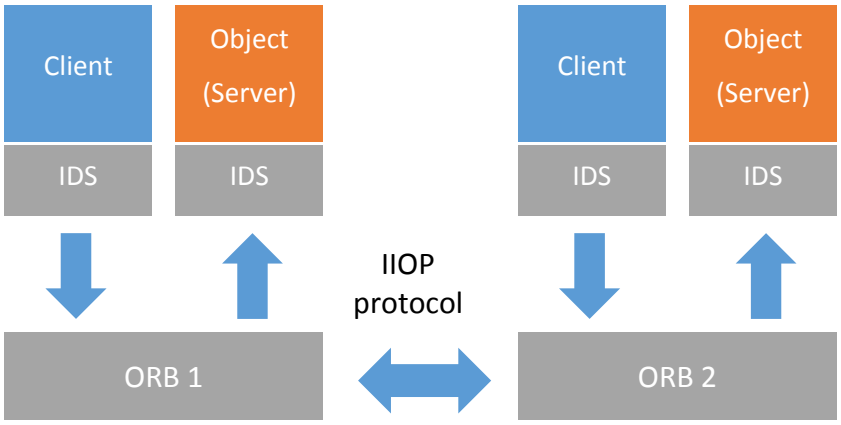

Figure 12: ORB-to-ORB communication [5]

The interoperability is achieved by very strict interface definition and using ORB.

10.4 SOAP Web service

A web service is a software supporting machine-to-machine interaction. It is a software function available on a network address. Its interface is described by web service description language (WSDL) which provides a description of available methods, their parameters and return types. The Web Service Architecture is defined by W3C at [6].

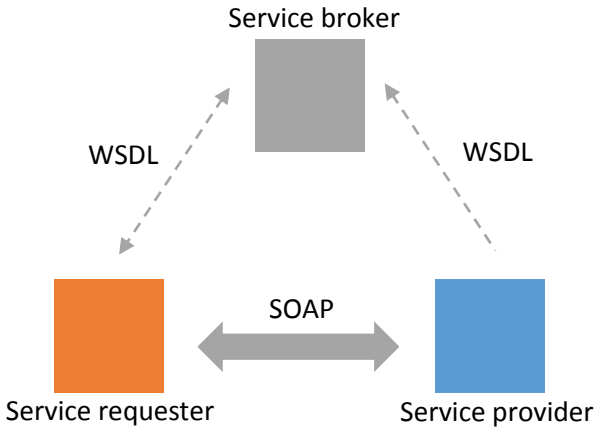

Figure 13: a general scheme of web service architecture

Other machines communicate with a web service using SOAP (Simple Object Access Protocol). SOAP is a protocol for exchanging XML messages over a network, mainly with help of HTTP.

10.5 Chosen solution

Having considered all mentioned options, I came to conclusion that a SOAP web service would be the most suitable. The service was built using a PHP framework Yii. It significantly simplifies the process of creating a web service. The framework allows a developer to imagine a web service as a PHP class [7], so it is necessary to define one class with functions. Then the framework translates it to WSDL file.

11 Generation of a certificate

Each application has different requirements on its certificate. In case of very limited budget, it is an acquisition price. Since the certificate is supposed to be used on mobile devices, the root certificate of the authority which signed the certificate should be preinstalled on as many devices as possible.

Fortunately, I found a CA meeting all my requirements. A company StartCom Ltd. provides certificates for non-commercial use, with time limited validity and it is free of charge.

Before applying for a certificate, it is necessary to sign up for a free account on https://www.startssl.com/. A web server SSL/TSL certificate can be easily generated via a validation wizard. During the generation process, the CA verifies the ownership of a particular domain name that a user wants to generate the certificate for.

That validation process supposes that it will not be a problem to receive an email with confirmation code. It is possible to choose an email address from three predefined options:

- postmaster@yourdomain.com
- hostmaster@yourdomain.com
- webmaster@yourdomain.com

After receiving the confirmation code, it shall be inserted into the validation wizard. If the code matches, the wizard is finished and a domain ownership is linked with the account.

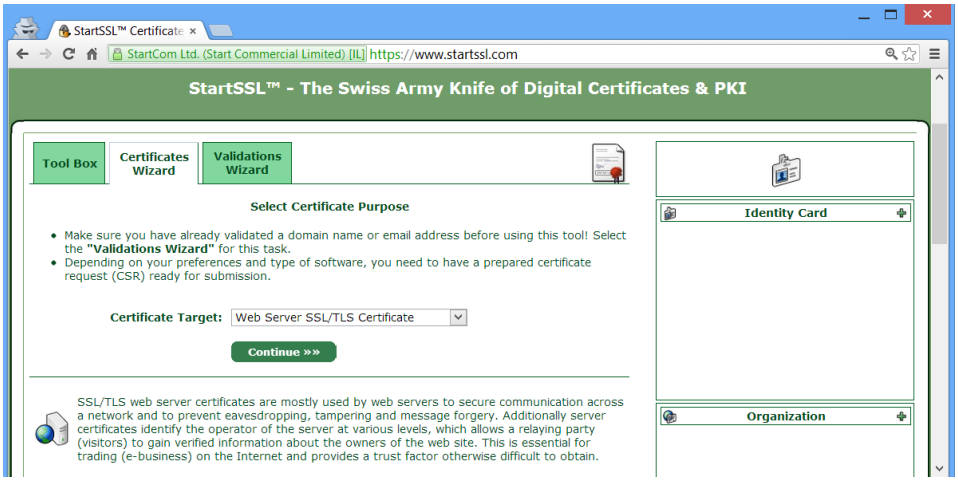

Figure 14: the certificates wizard available at startssl.com

Now, when the validation process is done, it may be proceed to generation of a web certificate. A signed certificate may be generated after submitting a private key and a certificate signing request (CSR). However, it is also possible to have them generated online by the certificate wizard.

The system needs some information to know, such as a password for the private key (10-32 characters), key size (2048, 4096), hash function (SHA1 or SHA2) and a domain name.

The final certificate is PEM encoded. OpenSSL tool is able to display all data hold by a certificate. It may convert the certificate to other formats suiting demanded needs.

12 Installing certificates on server

Following code installs two certificates on an Apache server. Each certificate is available on a different port number. It is number 446 for the self-signed certificate and 444 for the certificate signed by CA.

The stated code should be inserted into httpd.conf or ssl.conf file and it will not work if the mod_ssl module is not loaded.

Self-signed certificate

```
   Listen 446
<VirtualHost *:446>
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM
SSLCertificateFile /etc/httpd/conf/ssl-marpel.cz/ssl2.crt
SSLCertificateKeyFile /etc/httpd/conf/ssl-marpel.cz/ssl2.key
ServerName www.marpel.cz
ServerAlias marpel.cz.gargamel.multak.cz marpel.cz *.marpel.cz
*.marpel.cz.gargamel.multak.cz
DocumentRoot /home/users/MPeli/marpel.cz/web
php_admin_value open_basedir "/home/users/MPeli:/var/php/:/usr/share/pear/"
CBandUser MPeli
CBandScoreboard /etc/httpd/conf/cband-scoreboard-domains/marpel.cz
</VirtualHost>
```

A certificate signed by CA

```
Listen 444
<VirtualHost *:444>
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM
SSLCertificateFile /etc/httpd/conf/ssl-marpel.cz/ssl.crt
SSLCertificateKeyFile /etc/httpd/conf/ssl-marpel.cz/ssl.key
ServerName marpel.cz
ServerAlias *.marpel.cz *.marpel.cz.gargamel.multak.cz
DocumentRoot /home/users/MPeli/marpel.cz/web
php_admin_value open_basedir "/home/users/MPeli:/var/php/:/usr/share/pear/"
CBandUser MPeli
CBandScoreboard /etc/httpd/conf/cband-scoreboard-domains/marpel.cz
</VirtualHost>
```

13 Mobile applications

The main aim of this diploma thesis was to develop an application which would demonstrate an online and offline data transfer. I was lucky enough to have two mobile devices with different operating systems. I also had a web hosting support with possibility to use a PHP SAOP extension. Having consider all options, it was decided to create a two-platform chat.

The chat should be able to exchange data in the online and offline mode. The online mode requires the Internet connection and all data is exchanged via a web service. The Figure 15 illustrates the communication between two mobile devices. All messages are temporarily stored in a database.

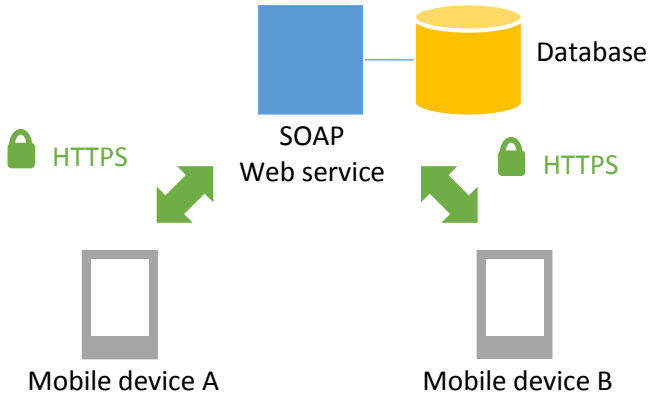

Figure 15: online data transfer

The offline mode processes messages via a short messages service (SMS). As the Figure 16 demonstrates, the offline communication does not use any remote database and all messages are transferred directly between both devices.

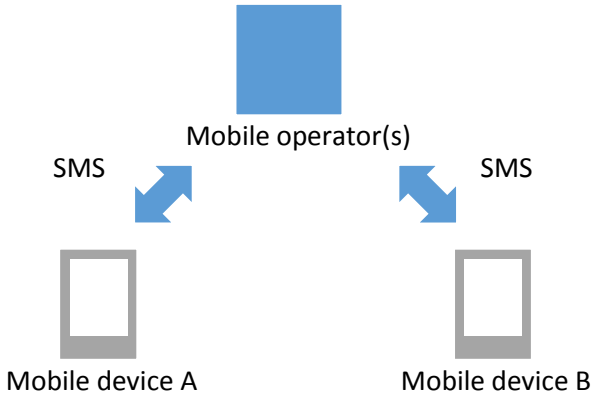

Figure 16: offline data transfer

Two demo applications will be created, one for Windows Phone 8 and the other one for Android 4. Both applications will use a secure connection over HTTPS with certificates signed by a certification authority. Furthermore, I will demonstrate a code for encryption with self-signed certificate and a solution for manual data encryption/decryption using available functions.

14 Binary to text encoding

Both applications make a use of a process called binary-to-text encoding for the offline communication. This kind of encoding is used when it is necessary to transmit binary data via a channel which does not allow that and transmits only text or if the binary data are supposed to be displayed on screen.

14.1 Base64

The Base64 is an encoding algorithm which splits binary data up into 6 bits chunks and turns them into characters using a conversion Table 2. This algorithm operates with 64 safe characters (6bit^2).

There are libraries available for languages such as C#, PHP, Java, Perl, Ruby and other popular languages which allow data encoding and decoding.

Assuming that a string “TUL” is supposed to be encoded. Firstly, it is necessary to look up all characters in the ASCII table. Each character matches a decimal number in the table. The obtained numbers are then converted into a bit form. The final bit stream contains 24-bits (3x8bit).

A next step is to divide the bit stream into 6 bit parts and covert them using the table stated in appendix A. The whole process of Base64 encoding is illustrated in table Table 1.

Table 1: Base64 encoding

String	T								U								L							
ACII	84								85								76							
ACII to bin	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0	1	1	0	0
Index	21						5						21				12							
Base64	V						F						V				M							

15 Offline data transfer

These days, the Internet connection is a regular thing which everybody has at home or at least at work. The mobile internet connection is not a matter-of-course and therefore there is a need to transmit data in different way.

In this part of this diploma thesis, GSM services such as phone calls and SMS as offline services will be considered.

The GSM was developed in late 80s and nobody assumed that it would be used in 21st century. The GSM uses several cryptographic algorithms. A cipher A5/1 is used in Europe. A little bit weaker cipher A5/2, used in other countries, is possible to break real time.

There is a group of devices called IMSI catchers. They basically create a false mobile tower (BTS) and intercept communication of all devices which are connected to it. This kind of attack is known as Man-in-the-middle which was already described earlier in this text.

The Figure 17 illustrates GSM networks. The red locks represent an unencrypted connection.

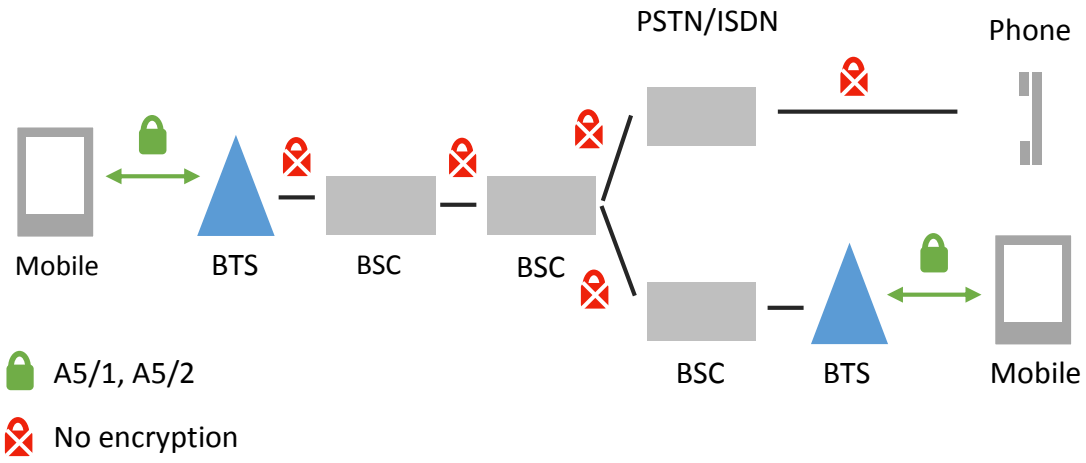

Figure 17: GSM network

- *BTS* Base Transceiver Station
- BSC Base Station *Controller*
- MSC Mobile Switching Centre
- PSTN Public Switched Telecommunication Networks

- ISDN Integrated Services Digital Network

The mobile applications will use text messages as an alternative. SMS is able to transfer certain number of characters per message. This number is derived from the number of bits allocated for message and from used encoding. It is possible to use 1120 bits. Messages sent via GSM may be encoded in 7bit, 8bit or 16bit data alphabet. The support of 7bit alphabet is mandatory.

It follows that the maximum number of characters in one message is 160 (1120/7). 7bit encoding creates 128 possible combinations. List of all 128 characters is stated in Table 1. Apart from alphanumerical and other characters, it contains some special ones such as LF, CR, ESC or SP. Different operating systems display these characters diversely. However, as long as they are processed programmatically, there are safe to use.

Nevertheless, messages longer the 1120 bits may be sent. A long message is split into several shorter messages (fragments). Due to the fact that the information about particular fragments has to be written into the message header (UDH - User Data Header), the maximum number of character is 153 (1071bits). In theory, a long message may consist of up to 255 fragments. However, mobile devices support only 6-8 fragments.

15.1 Message encryption

Before the message is encrypted, both parties have to agree on the encryption method and exchange key(s). There are basically the same options like for the online transfer – symmetric and asymmetric cryptography.

The asymmetric cryptography is slower and requires a stronger key then symmetric cryptography. Yet, both methods will be demonstrated.

The public-key cryptography was described earlier. A message will be encoded/decoded using public and private keys. Since an encrypted message has to fit into 160 characters (1120 bits), an appropriate length of keys must be chosen.

The length of encrypted string matches the length of used key. Since there are only 1120bit available to use, the maximum logic key size is 1024bit. The maximum input size may be computed according to this equation (applies for PKCS1 padding):

```
keySizeInBytes-11
```

So for 1024-bit long key, it is 936 bits (1024 - 11*8).

The AES encryption will be used for symmetric encryption. It is a block cypher and its input has to be the multiple blocks of 16 bytes. The size of encrypted message may be computed using following equation (applies for PKCS 5/7 padding):

```
cipherLenghtInBytes = [(numberOfInputBlock + 1) * 16]/2;
```

It implies that a secret message may consist of up to 138 bytes (69 blocks).

No matter which method is used, the encrypted string will be a byte array (Java, C#). Since the 7-bit GSM encoding will be used, the message may be composed only of characters stated in appendix B. That basically means that the array has to be converted into a sequence of bits and then split up into 7bit chunks. Each chunk is then replaced by particular character from the table.

It is an equivalent procedure to Base64. It operates with 6bits. A new class called Base128 was created for that purpose. Samples of codes using synchronous, anynchronous cryptography and the class Base128 are part of the appendix C.

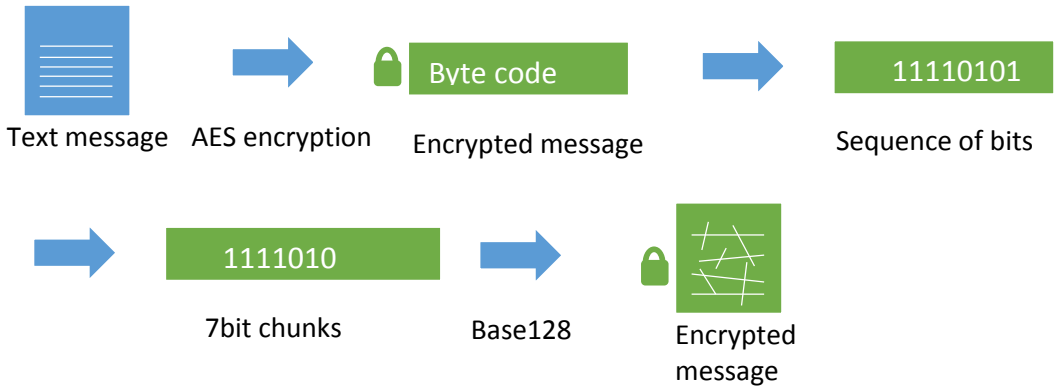

Figure 18: the process of SMS encryption

16 Application development

16.1 Windows Phone 8 development

The WP8 operating system is based on Windows NT kernel which also runs Windows 8. Both systems share partially the same API's and developments tools.

For an application development, the WP8 SDK is required. It contains all development tools including free Visual Studio 2012 Express.

The SDK is also able to extend a current installation of Visual Studio 2012.

A developer may use which programming language they want to use for WP8 development. Overall, there are three languages available .NET (C#, VB.NET), C++.

16.2 Android development

A developer may user chose from several Integrated Development Environments (IDE). There are available plugins for Eclipse, NetBeans, IntelliJ. However, the Eclipse is the only one IDE Google developed a plugin for.

17 Storing sensitive data

Android offers several options of storing data. It depends on how much space is actually needed and if the data is supposed to accessible to other applications, a user or kept private.

According to Android's documentation [10], it is possible to choose from following options:

- **Shared preferences** – storing primitive data in key-value pairs
- **Internal storage** – saving data into files, private by default
- **External storage** – SD cart or an internal (not removable) memory, open for everyone to read
- **SQLite database** – private relation SQL database
- **Network Connection** – storing data on remote server, requires the Internet connection

The Android is Unix based operating system and as such it servers many users (applications). Their possibilities are restricted by granted permissions. Other applications are prohibited from accessing a private storage of other applications because they do not have sufficient permissions.

Original Android devices are configured with limited user privileges. Users are not allowed to access the core of the operating system. Power users tend to remove this restriction by installing rooted custom ROMs. It allows them to gain root permissions and access all possible device functions.

The root user may access all files and do almost everything with the system. It may write on any storage, modify system settings, access private data, modify boot loader, pretend to be uninstalled, read any process's memory, etc.

A custom ROM may contain an application with root permissions and allow a user of the phone easily grant permissions to other applications. There is no sufficient way of protecting user's private data against an application with root access.

At least a security risk may be minimized by sensitive data encryption. The symmetric cryptography mentioned earlier above should be sufficient.

It is important to say that the reverse engineering can decode any application nearly to its original state and extract the source code. A possible hacker would be able to retrieve a key from the application.

It follows that it would be more secure to keep the key out of the application. The user would have to enter it from time to time. A user with root permissions would be still able to read the keyboard input and the entering of the key by the user would not be secure enough. However, this solution would be efficient in case of losing a phone.

It is only up to a publisher of the application what kind of level they require.

17.1 Windows Phone 8

By the time of writing this diploma thesis, there is no way of "rooting" a Windows Phone 8 device. However, it may be assumed that the situation will change soon and therefore consider the same security risks like with Android devices.

18 Demo applications

18.1 Windows Phone 8 demo application

The main screen of the application illustrates Figure 19. The sent messages are green and the received ones are red. There are also an input text field and a send button in the bottom. If a user hits the button, the message is automatically sent on the server. However, if there is no internet connection, a new screen appears and user may send the message via SMS. This process cannot be automatically done without user's interaction because of WP8 security policy. The SMS may be sent manually only. For the same security reasons, the application is not allowed to access received text messages and process them automatically.

The application periodically contacts the server and asks for new messages. This application was programmed in two versions. The client-server connection is always secure. Both application differ in used certificates. The first version uses a certificate signed by CA. The other one uses self-signed certificate.

In case of self-signed certificates, it is necessary to install them on user device. Otherwise the connection will not be established. WP8 does not allow automatic installation and it requires user's interaction. Following code offers a certificate to install (Figure 20):

```
StorageFolder packageLocation =
Windows.ApplicationModel.Package.Current.InstalledLocation;
StorageFolder certificateFolder = await
packageLocation.GetFolderAsync("Certificates");
StorageFile certificate = await certificateFolder.GetFileAsync("www_marpel_cz.cer");

await Launcher.LaunchFileAsync(certificate);
```

Figure 19: Windows Phone 8 demo app with loaded chat

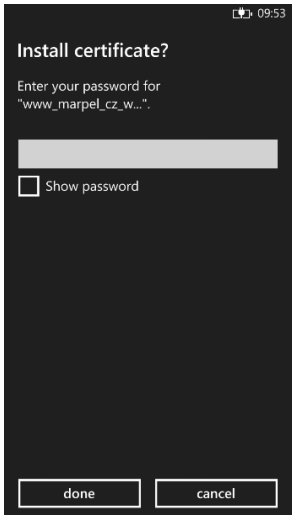

Figure 20: User has to confirm if they want to install a certificate

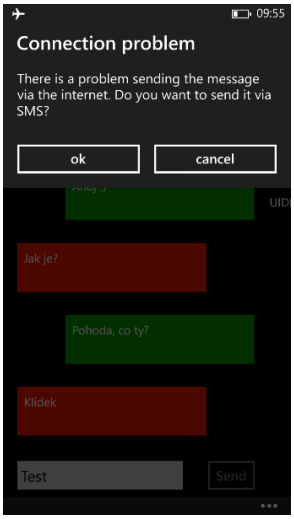

Figure 21: behavior of the system without internet connection

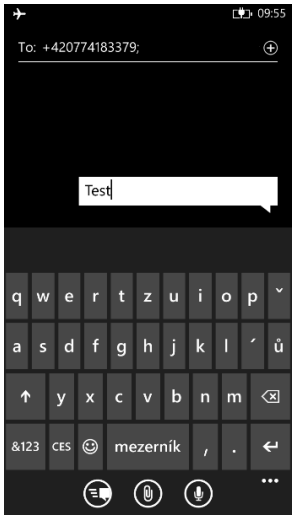

Figure 22: It is not possible to send an sms automatically

18.2 Android demo application

The Android application resembles the WP8 application in all aspects. Figure 23 and Figure 24 illustrate the design of application. However, the Android security policy is different and it allows automatic certificate installation and SMS manipulation. The user is not bothered with any unnecessary dialogs like in case of WP8.

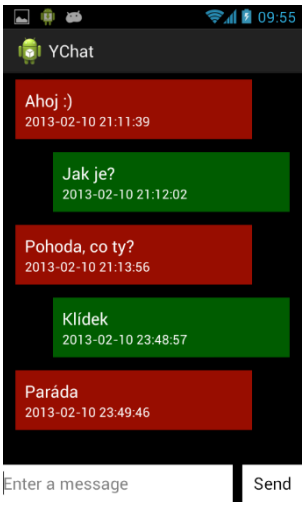

Figure 23: Android 4 demo app with loaded chat

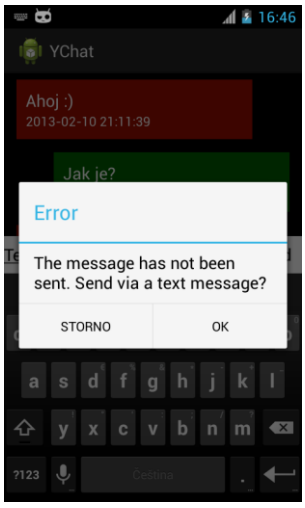

Figure 24: behavior of the system without an internet connection

19 Manual encryption

In case that SSL was not available on mobile devices, it would be necessary to find a way on how to encrypt and decrypt data manually. There are libraries which are able to extract keys from certificates and encrypt/decrypt data accordingly.

The manual encryption process was used for the SMS encryption and could be also applied on data stored in the database. The code examples are stated in the appendix D.

WP8 requires certificate in an XML form. Because WP8's API does not include advanced libraries for the certificate management, a new C# console desktop software was created. It converts keys from p12 to the XML format. Following piece of code converts a p12 key to XML:

```
X509Certificate2 cert = new X509Certificate2("./www_marpel_cz_with_private_key.p12",
"", X509KeyStorageFlags.Exportable | X509KeyStorageFlags.PersistKeySet);

RSACryptoServiceProvider cryptPrivate = (RSACryptoServiceProvider)cert.PrivateKey;
RSACryptoServiceProvider cryptPublic = (RSACryptoServiceProvider)cert.PublicKey.Key;

String privateXML = cryptPrivate.ToXmlString(true);
String publicXML = cryptPublic.ToXmlString(false);
…
```

The final XML files take the following form:

```
<RSAKeyValue>
   <Modulus>…</Modulus>
   <Exponent>…</Exponent>
   <P>…</P>
   <Q>…</Q>
   <DP>…</DP>
   <DQ>…</DQ>
   <InverseQ>…</InverseQ>
   <D>…</D>
</RSAKeyValue>
```

Conclusion

The diploma thesis focuses on an online and offline data transfer and a secure data storing.

The theoretical part describes the public cryptography in detail. Furthermore, it deals with possible threats such as the man-in-the-middle and suggests a solution using a mutual authentication based on certificates.

Several solutions of the online secure data transfer were suggested. They differ in a type of used certificates and in a method of authentication.

The first method uses certificates for a user authentication which are signed by a certification authority (CA). The second one uses so-called self-signed certificates.

For mobile devices, which are not able or do not want to make a use of SSL based protocols at all, a specific solution was designed. It demonstrates how to encrypt data manually using build-in libraries.

As long as the CA is trusted, the first suggested solution may be considered safe. Although, it is not a cost-free solution, it's a very secure and reliable way of securing data.

The last two solutions may be used for testing purposes, building proprietary software or local data encryption.

The diploma thesis introduces tools for generating, manipulating and managing keys and certificates. It describes how to obtain a certificate signed by a CA. It also describes how to generate a self-signed certificate. Moreover, it demonstrates the installation of both certificates on an Apache server.

Because whole online data transfer goes through a web server, it was necessary to create a SOAP web service and a MySQL database storage. The web service was built using a PHP framework.

Two demo applications were created. One for Android operating system. The other one for Windows Phone 8. Both applications are fully functional and are capable of the online secure data exchange.

The next part of the diploma thesis investigates how to use SMS for the offline data exchange. It notices security issues in the GSM network and suggests a possible solution.

It takes advantage of a symmetric cryptography. Particularly, it uses AES cypher. Finally, it provides an implementation for Android devices. It also comes up with an interesting finding that Windows Phone 8 applications are not allowed to control a message flow like the Android devices.

The thesis also discuss the safe data storing and makes a suggestion based on symmetric cryptography.

Overall, all task were completed and the diploma thesis meets all requirements.

No matter how sophisticated solution is used, the whole system remains only as strong as its weakest link. The security is relative and it is related to the amount of time and money that someone is willing to invest into a decryption. Considering the level of data sensitivity, the described solutions may be considered effective.

Literature

[1] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *IETF* [online]. 2008 [cit. 2013-04-26].
Available at: http://tools.ietf.org/html/rfc5280

[2] RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2. *IETF Tools* [online]. 2008 [cit. 2013-04-26].
Available at: http://tools.ietf.org/html/rfc5246#page-4

[3] How to create a new keystore. *Portecle* [online]. 2011 [cit. 2013-04-26].
Available at: http://portecle.sourceforge.net/create-keystore.html

[4] What Is a Socket?. *Oracle Documentation* [online]. [cit. 2013-04-26].
Available at: http://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

[5] CORBA® BASICS. *Object Management Group* [online]. 2013 [cit. 2013-04-26].
Available at: http://www.omg.org/gettingstarted/corbafaq.htm

[6] Web Services Architecture. World Wide Web Consortium (W3C) [online]. 2004 [cit. 2013-04-26].
Available at: http://www.w3.org/TR/ws-arch/

[7] Web Service. *Yii Framework* [online]. 2012 [cit. 2013-04-26].
Available at: http://www.yiiframework.com/doc/guide/1.1/en/topics.webservice

[8] The Base16, Base32, and Base64 Data Encodings. *IETF Tools* [online]. 2006 [cit. 2013-04-26].
Available at: http://tools.ietf.org/html/rfc4648

[9] 3GPP TS 23.038 V10.0.0: Alphabets and language-specific information [online]. zip .doc file. 2011 [cit. 2013-04-26].
Available at: http://www.3gpp.org/ftp/specs/archive/23_series/23.038/23038-a00.zip

[10] Storage Options. *Android Developers* [online]. 2012 [cit. 2013-04-26].
Available at: http://developer.android.com/guide/topics/data/data-storage.html

[11] Murphy, M.L., Android 2 Průvodce programováním mobilních aplikací, Computer Press a.s., 2011, ISBN 978-80-251-3194-7

[12] Petzold C., Programming Windows Phone 7, Microsoft Press, 2010, ISBN 978-0-7356-4335-2

[13] ALONSO, G., CASATI, F., KUNO, H., MACHIRAJU, V.: Web Services. Concepts, Architectures and Applications. Springer 2004. ISBN 3-540-44008-9

Appendix A – Base 64

Table 2: Base64 conversion table [8]

Value	Encoding		Value	Encoding		Value	Encoding		Value	Encoding
0	A		17	R		34	i		51	z
1	B		18	S		35	j		52	0
2	C		19	T		36	k		53	1
3	D		20	U		37	l		54	2
4	E		21	V		38	m		55	3
5	F		22	W		39	n		56	4
6	G		23	X		40	o		57	5
7	H		24	Y		41	p		58	6
8	I		25	Z		42	q		59	7
9	J		26	a		43	r		60	8
10	K		27	b		44	s		61	9
11	L		28	c		45	t		62	- (minus)
12	M		29	d		46	u		63	_ (underline)
13	N		30	e		47	v			(pad) =
14	O		31	f		48	w			
15	P		32	g		49	x			
16	Q		33	h		50	y			

Appendix B – basis character set

Table 3: Basic character set [9]

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x00	@	Δ	SP	0	¡	P	¿	p
0x01	£	_	!	1	A	Q	a	q
0x02	$	Φ	"	2	B	R	b	r
0x03	¥	Γ	#	3	C	S	c	s
0x04	è	Λ	¤	4	D	T	d	t
0x05	é	Ω	%	5	E	U	e	u
0x06	ù	Π	&	6	F	V	f	v
0x07	ì	Ψ	'	7	G	W	g	w
0x08	ò	Σ	(	8	H	X	h	x
0x09	Ç	Θ	)	9	I	Y	i	y
0x0A	LF	Ξ	*	:	J	Z	j	z
0x0B	Ø	ESC	+	;	K	Ä	k	ä
0x0C	ø	Æ	,	<	L	Ö	l	ö
0x0D	CR	æ	-	=	M	Ñ	m	ñ
0x0E	Å	ß	.	>	N	Ü	n	ü
0x0F	å	É	/	?	O	§	o	à

Table 4: Basic character set extension [9]

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x00					|			
0x01								
0x02								
0x03								
0x04		^						
0x05							€	
0x06								
0x07								
0x08			{					
0x09			}					
0x0A	FF							
0x0B		SS2						
0x0C				[				
0x0D	CR2			~				
0x0E				]				
0x0F			\					

Appendix C – SMS Encryption

Public key cryptography

```
// Load public key
cipherE = Cipher.getInstance("RSA/NONE/PKCS1Padding");
cipherE.init(Cipher.ENCRYPT_MODE,
keyStorePair.getCertificate("www.marpel.cz").getPublicKey());

// Encrypt with public key
byte[] encrypted = cipherE.doFinal("Secret string".getBytes("UTF-8"));

// Base128 encoding
String encodedSMS = Base128.encode(encrypted);

// Transfer SMS to client, following code decrypts the SMS

// Load private key
cipherD = Cipher.getInstance("RSA/NONE/PKCS1Padding");
cipherD.init(Cipher.DECRYPT_MODE, keyStorePair.getKey("www.marpel.cz",
"ychat".toCharArray()));

// Base128 decoding
byte[] decodedSMS = Base128.decode(encodedSMS);

// Decrypt with private key
byte[] decryptedSMS = cipherD.doFinal(decodedSMS);
String decryptedSMSString = new String(decryptedSMS, "UTF-8");
```

AES cryptography

```
/* AES */
byte[] keyStart = "my secret key".getBytes();
KeyGenerator kgen = KeyGenerator.getInstance("AES");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(keyStart);
kgen.init(256, sr);
SecretKey skey = kgen.generateKey();
byte[] key = skey.getEncoded();

// Encrypt
SecretKeySpec skeySpec = new SecretKeySpec(key, "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
byte[] encrypted = cipher.doFinal(new String("Secret
string").getBytes());

// Base128 encoding
String encodedSMS = Base128.encode(encrypted);

// Base128 decoding
byte[] decodedSMS = Base128.decode(encodedSMS);

// Decrypt
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
byte[] decryptedSMS = cipher.doFinal(decodedSMS);

String decryptedSMSString = new String(decryptedSMS, "UTF-8");
```

Appendix D – manual data encryption

Windows Phone 8

```
String privateKey = System.IO.File.ReadAllText(@"C:\keys\privateKey.xml");
String publicKey = System.IO.File.ReadAllText(@"C:\keys\publicKey.xml");

/* Encrypt with public key */
RSACryptoServiceProvider rsaPublic = new RSACryptoServiceProvider();
rsaPublic.FromXmlString(publicKey);
byte[] toEncrypt = System.Text.Encoding.UTF8.GetBytes("Secret string");
byte[] bytesPublic = rsaPublic.Encrypt(toEncrypt, false);
string base64Public = Convert.ToBase64String(bytesPublic);

/* Decrypt with private key */
RSACryptoServiceProvider rsaPrivate = new RSACryptoServiceProvider();
rsaPrivate.FromXmlString(privateKey);
byte[] toDecrypt = Convert.FromBase64String(base64Public);
byte[] bytesPrivate = rsaPrivate.Decrypt(toDecrypt, false);
string decryptedString = System.Text.Encoding.UTF8.GetString(bytesPrivate);
```

Android

```
/* Load public key */
cipherE = Cipher.getInstance("RSA/NONE/PKCS1Padding");
cipherE.init(Cipher.ENCRYPT_MODE,
keyStorePair.getCertificate("www.marpel.cz").getPublicKey());

/* Encrypt with public key */
byte[] encrypted = cipherE.doFinal("Secret string".getBytes("UTF-8"));
String encryptedBase64 = Base64.encodeToString(encrypted,
Base64.DEFAULT);

/* Load private key */
cipherD = Cipher.getInstance("RSA/NONE/PKCS1Padding");
cipherD.init(Cipher.DECRYPT_MODE, keyStorePair.getKey("www.marpel.cz",
"ychat".toCharArray()));

/* Decrypt with private key */
byte[] decrypted = cipherD.doFinal(Base64.decode(encryptedBase64,
Base64.DEFAULT));
String decryptedString = new String(decrypted, "UTF-8");
```

PHP

```
$publicKeyPath = 'file:////home/users…/self-signed/www_marpel_cz.cer';
$privateKeyPath = 'file:////home/users…/self-signed/ssl.key';
$passPhrase = 'ychat';

/* Load private key */
$this->privateKey = openssl_pkey_get_private($this->privateKeyPath,
$this->passPhrase);

/* Encrypt with public key */
openssl_public_encrypt("Sercrete string", $this->encrypted,
$this->publicKeyPath, OPENSSL_PKCS1_PADDING);

/* Decrypt with private key */
openssl_private_decrypt($this->encrypted, $this->decrypted,
$this->privateKey, OPENSSL_PKCS1_PADDING);
```