Masaryk University
Faculty of Informatics

Security and cryptography in GO

Bachelor's Thesis

Lenka Svetlovská

Brno, Spring 2017

Masaryk University
Faculty of Informatics

Security and cryptography in GO

Bachelor's Thesis

Lenka Svetlovská

Brno, Spring 2017

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Lenka Svetlovská

Advisor: RNDr. Andriy Stetsko, Ph.D.

Acknowledgement

This is the acknowledgement for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can span multiple paragraphs.

Keywords

keyword1, keyword2, …

Contents

List of Tables

List of Figures

1 Introduction

2 Basic Terms

In this section, I will define the basic terms which are necessary to an understanding of the text presented in the following chapters. First I will describe the structure of certificates, standard X.509, certificate authority and signing process. I will explain self-signed certificate, PEM and DER formats, and the process of certificate exchange. Finally, I will describe protocols SSL/TLS and TLS_PSK used in the application.

2.1 Certificate

Certificate is digitally signed data structure whose foundation includes a public key of certificate owner. There are some standards of the certificate structure (X.509, EDI, WAP, etc.). The Internet is based on standard X.509 version 3, which was issued by the ITU. For the needs of the Internet, there is created an Internet profile of X.509 in the relevant RFC. Current Internet profile certificate is standard RFC-5280 [1].

2.1.1 Standard X.509

Standard X.509 was originally designed as an authentication framework for X.500 directories [2]. They have a hierarchical structure and the individual attributes can be assigned to individual computers of companies or individual printers. That is, each entity can be clearly identified, and each can have its private and public key. The proposal envisages the hierarchical structure of certificate authorities. Certificate X.509, see Figure 2.1.

The first version of the X.509 standard was already appeared in 1988 as the first proposal for PKI [2]. The first version was very simple and today is already inadequate. It contained only 7 fields:

- certificate version;
- certificate series number;
- algorithm which signed certificate;
- name of certification authority which certificate released;
- identity of certificate owner;

- public key of certificate owner;
- certificate validity.

The second version was introduced five years later in 1993 and brought only minimal changes. The second version did not solve the problems associated with the first version. Fields which was needed in this version, still missing [2]. It contained only two new fields:

- unique identifier of the certificate owner;
- unique identifier of the certification authority.

The third version was introduced in 1996. Its biggest and most important benefit is to support the expansion. It is defined a syntax that allows you to create custom certificate extension. It eliminates the major shortcomings of the two previous versions, but then appears some incompatibility.

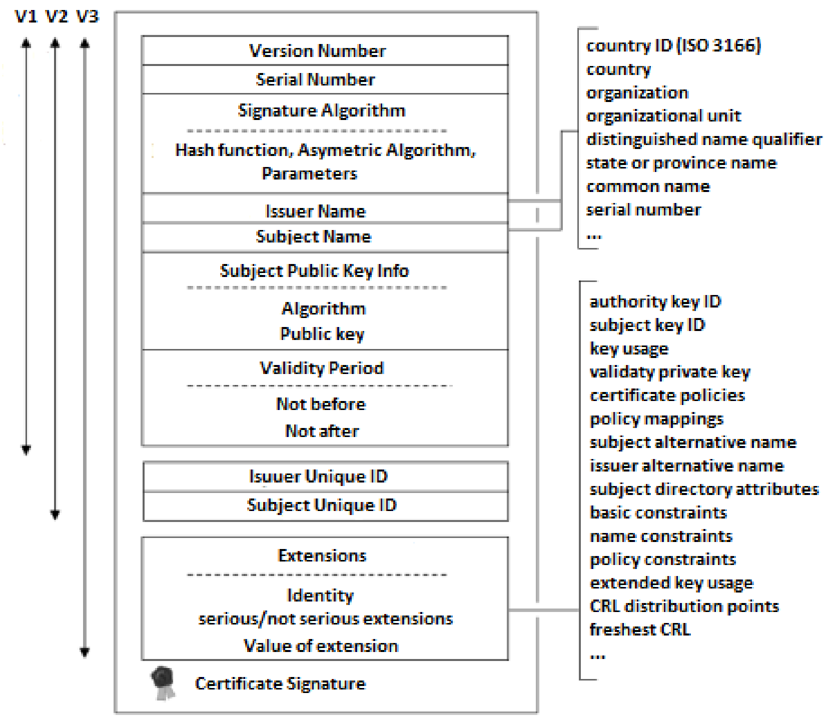


Figure 2.1: Structure X.509 Certificate

2.1.2 Certificate Authority

Certification Authority (CA) is an independent third party that issues certificates.

In other words, the CA is an institution which inspects the certificate request and issues certificates whose validity is verifiable using the responsive public key. The CA also takes care of archiving and distribution of certificates checks their validity and ensures their revocation. To the CA is truly credible, it must be impartial and, if it is possible, it should be subordinate any higher CA. Subordination means that it identifies using a certificate signed by the certification authority higher level [1].

Root CA, certification authorities at the highest level, uses the certificate signed by themselves (self-signed certificate).

2.1.3 Self-signed Certificate

The self-signed certificate is a certificate which the applicant gives by itself. The self-signed certificate has the same data structure as a certificate issues by CA. This certificate is recognized according to identical item Subject and item Issuer.

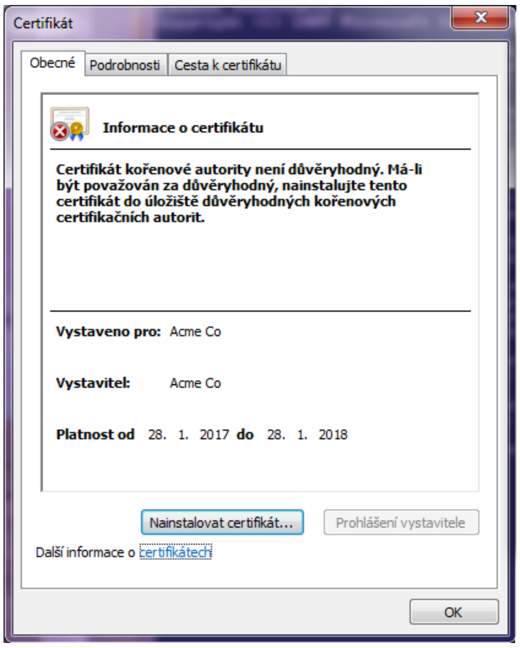
Creating self-signed certificate is not easy. To create certificate request we need to fill same items, which we do not know. For example, serial number, certificate validity, etc.

As proof of the private key possession by the user, the self-signed certificate uses a digital signature certificate which was made by the private key belonging to the public key in the certificate. Verification of the signature is carried out through a public key in the actual certificate [1].

Self-signed certificate is used by software internally. While the applicant generates CSR to be issued by the CA, the public key must be kept by the applicants. In this case, the public key is often maintained like a self-signed certificate. Subsequently it is rewritten by the certificate.

2.1.4 Formats PEM and DER

The PEM (Privacy Enhanced Mail) is a Base64 encoded DER certificate. It is designed to be safe for inclusion in ASCII or even rich-text documents. This means that we can simple copy and paste the content of a pem file to another document and back. Following is a sample PEM file containing a private key and a certificate 2.2.


Certifikát

Obecné | Podrobnosti | Cesta k certifikátu

Informace o certifikátu

Certifikát kořenové autority není důvěryhodný. Má-li být považován za důvěryhodný, nainstalujte tento certifikát do úložiště důvěryhodných kořenových certifikačních autorit.

Vystaveno pro: Acme Co

Vystavitel: Acme Co

Platnost od 28. 1. 2017 **do** 28. 1. 2018

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAz2DH3nJePLkL84mVCAgtO8I
MI5cI3e2JKRr/cDpYa/sm7I5WRxeOqtNIN+VU6m
A .... MANY LINES LIKE THAT ..........
sXeRvIEx3lQnC1HFKuJN/8dDcD/tQi2URl4lgpV
8PlZo5bit+nzU2fvJIKOoGLF1NyX18ImApWL8kj
kgL3iXR5t9lbHRsD85zzy3OYBKZBXE/T5cbW4uW
-----END RSA PRIVATE KEY-----

-----BEGIN CERTIFICATE-----
MIIC9jCCAd6gAwIBAgIRAOzV53Pr8ldSJfEtCYN
ODAwWjATMREwDwYDVQQKDAhMZW5rYSA8MzCCASI
A .... MANY LINES LIKE THAT ..........
vZk8K04FuGSxpY7mtPLfeHni9RpxQAoU187/DwU
vpJyk/+8o7e0U3Lil/O7iZjXr3jDSYENc/fj0Hx
r+uQEnxgWgeGyhaB6f15zo6mGY0r+Czkov2gdJH
-----END CERTIFICATE-----
```

Figure 2.2: A sample example of certificate

A certificate or key must start with header and the number of dashes is meaningful, and must be correct. A single PEM file can contain a number of certificates and a key, for example, a single file with public certificate, intermediate certificate, root certificate or private key [3].

The DER (Distinguished Encoding Rules) is a binary form of ASCII PEM format certificate. All types of certificates and private keys can be encoded in DER format, its information is stored in the binary DER for ASN.1 and applications providing RSA, SSL and TLS should handle DER encoding to read in the information [4].

2.1.5 Certificate Signing Request

The certificate authority must fill individual items of the certificate duly before it signs the result digitally. The certificate applicant can apply in two ways: submit a data structure called a certificate request or not so [1].

The certificate request should include:

- Applicant ID
- The public key
- Evidence of the possession of the private key
- Other information that the user wishes to insert certificate
- May contain evidence of the generation of paired data
- Data necessary for billing (in case issuance of certificates is paid)
- Passwords for communication with CA:
  - One-time password for issuing the certificate
  - One-time password for certificate revocation
  - Permanent password for personal (non-electronic) communication between user and CA
  - Phrase (in case losing all passwords)

2.1.6 Certificate exchange

2.2 Protocols SSL and TLS

Protocols SSL and TLS are used for secure communications between client and server. They create a framework for the use of encryption and hash functions.

SSL was developed by Netscape Communications which published three versions. The first version was only a test, the second has been used in practice. However, it still contained security vulnerabilities, the most important was susceptibility to attack *man in the middle*. The third version was introduced in 1996 and its specification could be found in the document *The SSL Protocol Version 3.0* [5]. Of this version was created TLS protocol which is currently the most widespread and supported [6]. There are three versions which have only minimal differences.

Protocol SSL/TLS provides authentication of the two communicating parties by using asymmetric encryption, message integrity by using MAC and confidentiality by encrypting all communications by selected symmetric cipher.

Protocol SSL/TLS is located between the application and the transport layer reference ISO/OSI model and consists of two main parts. They are *The Record Layer Protocol* (RLP) and *Handshake Protocol* (HP).

Record Layer Protocol processes application data, performs fragmentation, compression and data encryption. On the other hand, it decrypts the data again and verifies the checksums. RLP protocol does not care about the type of encryption algorithm or encryption key setting. This information is from HP.

Handshake Protocol is activated immediately after establishment the connection and provides identification of communicating parties, provision of cryptographic algorithms, compression algorithms and other attributes. Then it creates *a master secret* from which are derived encryption keys, initiation vectors and the MAC. The process of the protocol is:

1. The client wants to connect to the server and sends *ClientHello*, which contains the highest number of version supported by SSL/TLS, the number of session (it is empty if it is a new session), the list of supported ciphers and compression methods and a random number.

2. The client waits for a response in the form of a report *ServerHello*, which will contain the highest number of versions of SSL/TLS, which is supported by server and client. It will also contain encryption and compression method, which are selected from the list received in step one, a random number and its public key certificate (the server can also request authentication client).

3. The client verifies the server certificate, if all ciphers are satisfied. Next, he sends a request to server to exchange keys. At the end the server and client share a common 48 bits *premaster secret*. *The master secret* is derived from it. Then of the master secret and random numbers *ClientHello ServerHello* are derived two session keys to encrypt messages and two MAC initiation vectors for use symmetric cipher in CBC mode.

4. The client sends a confirmation selected ciphers. From this moment, the communication has been encrypted and the client sends a message that ends with this phase. In case the server

required the client authentication, it has been carried out in this step. Finally, the server sends a confirmation used ciphers and message about the completion of this phase, thereby HP ends.

The used algorithms:

- key exchange: RSA, Diffie-Hellman, ECDH;
- stream symmetric ciphers: RC4 with key length of 40-120 bits;
- block symmetric ciphers: DES, DES40, 3DES, IDEA;
- hashing algorithms: MD5, SHA.

TLS uses public key certificates for authentication. To establish a TLS connection are used symmetric keys (later called pre-shared keys or PSKs), shared in advance among the communicating parties [7].

2.2.1 PSK Key Exchange Algorithm

The cipher suites with PSK key exchange algorithm, use only symmetric key algorithms and are thus especially suitable for performance-constrained environments where both ends of the connection can be controlled. The cipher suites are intended for a rather limited set of applications, usually involving only a very small number of clients and servers.

The client indicates its willingness to use pre-shared key authentication by including one or more PSK ciphersuites in *the ClientHello message*. If the TLS server also wants to use pre-shared keys, it selects one of the PSK ciphersuites, places the selected ciphersuite in *the ServerHello message*, and can includes *ServerKeyExchange message*. If *ServerKeyExchange message* is included depends on the server provides to help the client in selecting which identity to use. The server can provide a "PSK identity hint" in *the ServerKeyExchange message*. The client indicates which key to use by including a "PSK identity" in *the ClientKeyExchange message*. If no hint is provided, *the ServerKeyExchange message* is omitted. Both clients and servers may

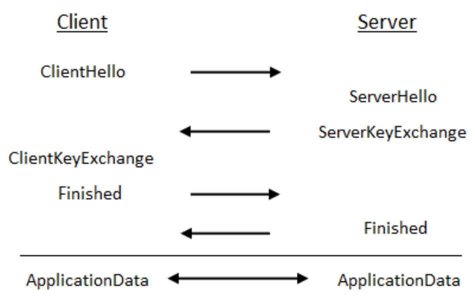


Figure 2.3: TLS PSK

have pre-shared keys with several different parties. The Certificate and CertificateRequest payloads are omitted from the response.

The TLS handshake is authenticated using *the Finished messages* as usual. If the server does not recognize the PSK identity, it may respond with an *unknown_psk_identity alert message*. Alternatively, if the server wishes to hide the fact that the PSK identity was not known, it may continue the protocol as if the PSK identity existed but the key was incorrect: that is, respond with *a decrypt_error alert* [7].

3 Language Go

In this chapter I will explain the main characters of language Go and show how to write the code. The next I will describe Go libraries called packages and the tool cgo.

3.1 Introduction

Go is a programming language developed at *Google* in year 2007 and announced in November 2009. Many companies have started using Go because of its performance, simplicity, ease of use and powerful tooling. Go programming language is a statically-typed language with advanced features and a clean syntax [8]. It combines the performance and security benefits associated with using a compiled language like *C++* with the speed of a dynamic language like *Python*. It provides:

- garbage collector - memory is cleaned up automatically when nothing refers to it anymore,
- fast compilation time - through effective work with addictions individual parts of the program and simple grammar,
- light-weight processes (via go-routines), channels,
- a rich standard library,
- easy testing - incorporated directly into the core language,
- one of the best documentation - a clear and full of examples.

Go excluded some features intentionally to keep language simple and concise. There is no support for type inheritance, method or operator overloading, circular dependencies among packages, pointer arithmetic, assertions nor for generic programming.

3.2 How to Start

3.2.1 Installation

Golang, the go official website, provides the installer of Go to download free for Windows, Linux, Mac OS X and FreeBSD tarballs. To confirm everything is working, open a terminal and type the following:

```
go version
```

Upgrading from an older version of Go must be precessed by removing the existing version.

3.2.2 Enviroment

The Go toolset uses an environment variable called GOPATH to find Go source code.

Set up your workplace, for example GOPATH=C:\Projects\Go\.

3.2.3 First Program

Traditionally, the first program in any programming language is called a hello program — a program that simply outputs "Hello World!" to terminal.

Open text editor, create file *hello.go*, and enter the following:

```
package main
import "fmt"
// this is a comment
func main() {
  fmt.Println("Hello World!")
}
```

Make you sure the file is identical to what is shown here and save it.

3.2.4 Compilation

Go is a compiled language, and like many languages, it makes heavy use of the command line. Open up a new terminal, set up directory where you saved the file *hello.go* and type in the following:

```
go run hello.go
```

You should see Hello World! displayed in your terminal.

3.2.5 Arguments

Command-line arguments are a common way to parameterize execution of programs. To build a binary with go build first.

```
go build cmd-arguments.go
./cmd-arguments a b c
```

In code, *os.Args* provides access to raw command-line arguments. The first valueis the path to the program, and os.Args[1:] holds the arguments to the program.

```
argsWithProg := os.Args
argsWithoutProg := os.Args[1:]
arg := os.Args[3]
```

Do not forget to import package *os*.

3.3 Packages

In Go, source files are organized into system directories called packages. To develop software applications, writing maintainable and reusable pieces of code is very important. Go provides the modularity and code reusability through it is package system. Go encourages programmers to write small pieces of software components through packages, and compose their applications with these small packages.

The packages from the standard library are available at the "pkg" subdirectory of the GOROOT directory. When we install Go, an environment variable GOROOT will be automatically added to our system for specifying the Go installer directory. The Go developer community is very enthusiastic for developing third-party Go packages. When you develop Go applications, you can leverage these third-party Go packages [9].

When programmers develop executable programs, they will use the package "main" for making the package as an executable program. The package "main" tells the Go compiler that the package should compile as an executable program instead of a shared library. When programmers build shared libraries, they will not have any main package and main function in the package.

To download third-party Go packages is used command:

```
go get example/exampleLib
```

After installing the exampleLib, put the import statement in programs for reusing the code, as shown below:

```
package db
import (
  "example/exampleLib"
  "example/exampleLib/sub"
)
func init {
  // initialization code here
}
```

To install third-party Go packages is used command:

```
go install
```

The go install command will build the package “sub” which will be available at the pkg subdirectory of GOPATH.

```
package main
import (
  "fmt"
  "example/exampleLib/sub"
)
func main() {
    sub.Add("dr","Dart")
    fmt.Println(sub.Get("dr"))
    // code here
}
```

3.4 C? Go? Cgo!

Cgo allows Go to interoperate with C code. Go source file imports "C" and it is immediately preceded by comment lines which may contain any C code, including function and variable declarations and definitions. It is necessary to say that there must be no blank lines in between the cgo comment and the import statement.

```
package main
/*
#include <stdio.h>

void hello() {
  printf("Hello World!\n");
}
*/
import "C"

func main() {
  C.hello()
}
```

Cgo allows including own C libraries. The header file *foo.h*, the library implementation *foo.c* and go wrapper *foo.go* are located in the same package $GOPATH\src\foo$.

```
#cgo CFLAGS: -I/{$GOPATH}/src/foo
#cgo LDFLAGS: -L/{$GOPATH}/src/foo -lfoo
```

Go wrapper contains the explicit paths in the cgo flags because of an environment issue where libfoo wasn't found by the Go linker.

```
go install foo
```

We install package and all is prepared for using in Go. Package is called in Go code through *import"foo"*.

In order to use cgo on Windows, we will also need to first install a gcc compiler and have *gcc.exe* in PATH environment variable before compiling with cgo will work.

4 Analysis

It was necessary to examine features of language GO related safety and cryptography, analyze libraries which GO contains and also search for available third-party libraries. This chapter contains the overview, evaluate and compare existing cryptographic liberates base on their measures and next base on expansion measures important for implementation.

Table 4.1 shows analyzed Go package Crypto created by Golang and other Go packages created by third-parties. Individually items mean:

- Name - name of founder and package.
- Cryptographic functions - support cryptographic functions, such as symmetric encryption, asymmetric encryption and hash functions (all/sym/asym/hash/unknown).
- Start of project - a year of creating project.
- Is still opened - contributors still work on the project (Yes/No).
- Issue tracker system - support of solving bugs, feature requests, tracking todos, and more (Yes/No).
- Documentation accessibility - assessed on a scale of 0-5 (5 is the highest).
- Downloads - number of downloads. In case the package is saved on *https://github.com/*, the information about downloads presents value of Fork. Downloads of package Crypto created by Golang is unknown, this package is automatically installed with installing program Go.
- License - availability of licenses for free (Yes/Payment). The payment indicates the amount of payment per some period. This case did not occur.

Table 4.1: Table of cryptographic libraries in Go

Name	Crypto. functions	Start of project	Is still opened	Issue tracker system	Doc. accessability	Downloads	License
golang/ crypto	all	2009	Y	Y	5	-	Y
ory-am/ fosite	asym, hash	2015	Y	Y	4	39	Y
square/ go-jose	asym, sym	2014	Y	Y	4	70	Y
Shopify/ ejson	hash	2014	Y	Y	3	22	Y
mitchellh/ go-mruby	asym	2014	Y	Y	4	19	Y
Sermo Digital/ jose	all	2015	Y	Y	4	28	Y
mozilla/ tls-observatory	hash	2014	Y	Y	4	31	Y
square/ certigo	hash	2016	Y	Y	3	12	Y
docker/ libtrust	all	2014	N	Y	4	26	Y
dedis/ crypto	sym, hash	2010	Y	Y	4	17	Y
dchest/ blake2b	hash	2012	N	Y	4	6	Y
enceve/ crypto	hash	2016	N	Y	4	1	Y
go-libp2p-crypto	all	2015	Y	Y	4	5	Y

Continued on next page

Table 4.1 – continued from previous page

Name	Crypto. functions	Start of project	Is still opened	Issue tracker system	Doc. accessability	Downloads	License
dchest/blake2s	hash	2012	N	N	3	3	Y
benburkert/openpgp	sym, hash	2012	N	N	3	0	Y
minio/sha256-simd	hash	2016	Y	Y	3	14	Y
avelino/awesome-go	unknown	2014	Y	Y	2	2287	Y
ethereum/go-ethereum	all	2013	Y	Y	4	1028	Y
chain/chain	hash	2015	Y	Y	4	113	Y
lightningnetwork/lnd	sym, hash	2015	Y	Y	4	50	Y
square/go-jose	asym, hash	2014	Y	Y	4	70	Y
dgrijalva/jwt-go	asym, hash	2012	Y	Y	4	226	Y
golang/oauth2	asym, hash	2014	Y	Y	4	279	Y

The table 4.1 was created on the base of information available on following websites [10] [11] [12] [13]. The table does not contain packages which do not support any cryptographic functions. Also, the table does not contain information from untrusted sources. During the analysis, I discovered sources which are not actual or supported [14].

The next step of analysis was select the libraries which fulfill the requirements. The important parameters were the support of all cryptographic functions, the topicality of project, the approach to solving problems and no payment license. In the table **??**, there are shown four libraries and compared to the base of the other measures:

- Crypto. key - possibilities for generation of cryptographic keys (Yes/No),
- X.509 Certificates - work with X.509 certificates (generation of self-signed certificates, support for certificate signing requests, support for different formats and extensions) (Yes/No),
- SSL/TLS - support of higher level protocols such as SSL/TLS (Yes/No),
- Use go lang crypto - dependence on Crypto package created by Golang (Yes/No/-).

	crypto	SermoDigital/ jose	go-libp2p-crypto	ethereum/go-ethereum
Crypto. functions	all	all	all	all
Start of project	2009	2015	2015	2013
Is still opened	Y	Y	Y	Y
Issue tracker system	Y	Y	Y	Y
Doc. accessability	5	4	4	4
Downloads	-	28	5	1028
License	Y	Y	Y	Y
Crypto. keys	Y	Y	Y	Y
X.509 Certificates	Y	Y	Y	Y
SSL/ TLS	Y	N	N	N
Use Golang Crypto	-	Y	Y	Y

Table 4.2: Filtered table of cryptographic libraries in Go with other measures

Bibliography

1. DOSTÁLEK, Libor; VOHNOUTOVÁ, Marta. *Velký průvodce infrastrukturou PKI*. Computer Press, Albatros Media as, 2016.
2. SCHMEH, Klaus. *Cryptography and public key infrastructure on the Internet*. John Wiley & Sons, 2006.
3. HOW TO SSL. *PEM Files*. 2017. Available also from: `http://how2ssl.com/articles/working_with_pem_files/`. [Online; 21-02-2017].
4. BAKKER, Paul. *ASN.1 key structures in DER and PEM*. 2014. Available also from: `https://tls.mbed.org/kb/cryptography/asn1-key-structures-in-der-and-pem`. [Online; 21-02-2017].
5. FREIER, Alan; KARLTON, Philip; KOCHER, Paul. The secure sockets layer (SSL) protocol version 3.0. 2011.
6. OPPLIGER, Rolf. *Security technologies for the world wide web*. Artech House, 2003.
7. ERONEN, Pasi; TSCHOFENIG, Hannes. *Pre-shared key ciphersuites for transport layer security (TLS)*. 2005. Technical report.
8. DOXSEY, Caleb. *Introducing Go*. O'Reilly Media, 2016.
9. VARGHESE, Shiju. *Understanding Golang Packages*. 2014. Available also from: `https://thenewstack.io/understanding-golang-packages/`. [Online; 27-02-2017].
10. GOLANGLIBS. *Security*. 2017. Available also from: `https://golanglibs.com/top?q=security`. [Online; 27-02-2017].
11. GOLANGLIBS. *Cryptography - Go libraries and apps*. 2017. Available also from: `https://golanglibs.com/category/cryptography?sort=top`. [Online; 27-02-2017].
12. GO DOC. *crypto - GoDoc*. 2017. Available also from: `https://godoc.org/golang.org/x/crypto`. [Online; 27-02-2017].
13. GOLANG. *Packages*. 2017. Available also from: `https://golang.org/pkg/`. [Online; 27-02-2017].
14. PURE-GO-LIBS. *Pure Go Libs*. 2017. Available also from: `http://go-lang.cat-v.org/pure-go-libs`. [Online; 27-02-2017].