Design and Implementation of a Configurable System for Managing X509 Certificates

Maja B. Vukasovic, Bogdana S. Veselinovic, and Zarko S. Stanisavljevic

Abstract — **In this paper a design and an implementation of a configurable system for managing the X509 certificates is described. The X509 certificates are one of the most important standards used today in a large number of different authentication mechanisms. As such, they represent one of the important topics at the Computer Security course at the School of Electrical Engineering, University of Belgrade. The system described in this paper has been used within the practical project at the Computer Security course. It provides the ability to generate, sign, and manipulate X509 certificates and it's implemented in such a way that the graphical user interface (GUI) is completely separated from the logic. This way of implementation relaxes the students from GUI programming and enables them to concentrate on programming the part related to the certificates – security part of the project. The system has been successfully used at the course in the previous school year and the new and improved version is going to be used in this school year.**

***Keywords* — Authentication, Computer Security, Information Security, E-learning Software Systems, X509 Certificates.**

I. Introduction

THE X509 certificates are an indispensable part of many widely used applications, such as TLS [1], which is a basis for HTTPS communication on the internet. They are also used in forming digital signatures in various applications. By validating a certificate, the one in its possession can use the public key stored within it to establish a secure connection with the certificate owner or to verify authenticity of a message sent by the certificate owner [2]. One way of performing digital signatures is by using asymmetric cryptographic algorithms that are based on using different keys for encryption and decryption. When a message is encrypted with the sender's private key, it can be revealed only by using sender's public key for decryption. That way whoever receives such a message can be assured of its integrity and sender authenticity, because only the owner of appropriate private key could prepare such a message. Certificate Authority (CA) represents one of the central entities in the public key infrastructure (PKI) [3]. CA is responsible for issuing certificates. The main purpose of a certificate is to bind information about the public key stored in the certificate with the owner of the certificate. They can be observed as a virtual personal ID documents. Like the physical ID documents the virtual ones also have information about the owner, date when they were issued, information about the issuer, issuer signature. However, unlike the physical ID document, owning a certificate that belongs to someone else is utterly useless for stealing that person's identity, since the most important element for proving identity is the private key, which is not the part of the certificate. Digital certificates can also be used for user authentication in network communication. In the age of mass use of social media, mobile devices and computers in everyday life, where privacy is a strong requirement, the significance of digital certificates becomes very high [4]. The sole process of creating, signing and validating digital certificates is very abstract and as such it is one of the topics at the Computer Security course, which is taught at the University of Belgrade, School of Electrical Engineering. In order to allow students to better understand the topic, one part of the course is the practical project that involves implementation of a Java GUI application for handling X509 certificates. The main goal of the project is to teach students how to program security part of such an application and that is why students are provided with a completed GUI part of the application. This paper describes how the template for project was designed and implemented. The GUI part is completely separated from the certificate handling part. As far as authors are aware no similar system is described in the open literature. The initial research was reported in [5].

The rest of the paper is organized as follows. In the second part of the paper X509 certificates are described. The third part of the paper explains the architecture and the functionalities of the developed system. Usage examples for different groups of users are shown in the fourth part of the paper. The fifth part of the paper gives conclusions.

II. X509 Certificates

X509 represents one of the most popular and mostly used standards that determine public key certificate format. Standard is created by ITU-T [6]. Digital certificates bind certificate owner information with specific owner's public key. In order to manage public key encryption and digital certificate creation and usage the standard defines a

Paper received May 9, 2018; accepted September 10, 2018. Date of publication December 25, 2018. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Miroslav Lutovac.

This paper is a revised and expanded version of the paper presented at the 25th Telecommunications Forum TELFOR 2017 [5].

Maja B. Vukasovic, University of Belgrade, School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia (phone: 381-11- 3218-392, e-mail: majav@etf.bg.ac.rs).

Bogdana S. Veselinovic, University of Belgrade, School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia.

Zarko S. Stanisavljevic, University of Belgrade, School of Electrical Engineering, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia (phone: 381-11-3218-392, e-mail: zarko.stanisavljevic@etf.bg.ac.rs).

formally specified set of rolls and procedures called PKI. PKI determines the way certificates are being created, managed, distributed, used, and withdrawn [3].

Up to now there have been three versions published incrementally – all versions contain fields from previous versions and add new fields introduced in the current version.

The first two versions contain:

1. tbs (to be signed) certificate - holds public key, information related to certificate owner and issuer as well as expiration date, certificate version, etc.
2. signature algorithm - represents information about public key cryptography algorithm and hashing algorithm as well as their parameters used by certificate authority when signing certificate.
3. signature value - holds digital signature calculated with parameters from tbs certificate.

By generating the signature, as previously said, CA binds information about certificate owner with public key information and guarantees certificate validity. Additionally, version 3 introduces extensions as generic mechanism for binding information about owner, public key and additional attributes. Also it is a mechanism for specifying that certain certificate is CA and managing CAs hierarchy. All extensions include type identifier, criticality and value. An example of digital certificate usage scenario is establishing secure server-client HTTPS connection. It is necessary that client checks certificate validity by checking signature. If there is more than one certificate in the certificate chain every member of the chain must be checked. If the client can validate the signature and trusts authority who issued the certificate he can eventually establish secure server-client connection [7].

III. System implementation

The developed system is a Java [8] application for handling X509 digital certificates. Java programming language was chosen primarily due to portability. Used runtime environment is jre1.8.0_161. Application functionalities for cryptographic algorithms and digital certificates handling are mostly implemented using Bouncy Castle API [9] of version 1.59. The jdatepicker-1.3.4.jar library [10] was used in order to easily input and display data in the date format on the GUI. The application provides the following features:

- generating a new key pair using one of the supported cryptographic algorithms,
- importing a #PKCS12 formatted key pair from an existing file [11],
- exporting a key pair to a file according to #PKCS12 format additionally encrypted by AES algorithm [12],
- displaying information about an existing key pair,
- creating a certificate signing request for a selected key pair and exporting it to a #PKCS10 file [13],
- importing and displaying information about a certificate signing request from an existing #PKCS10 file,
- signing a certificate signing request and exporting the CA reply to a #PKCS7 file [14],
- exporting an existing X509 certificate or X509 certificate chain to a file according to PEM or DER format,
- importing an X509 certificate from an existing PEM or DER encoded file.

The graphical user interface (GUI) can be configured to enable handling versions 1 and/or 3 X509 certificates. GUI is also configurable in terms of algorithms for key pair generation and in case of version 3, certificate extensions and certificate extensions rules. GUI's appearance is adjusted using parameters that can be read either from a configuration file or initialized in the source code. Adjustable GUI leads to a large number of different algorithm combinations for key pair generation and extensions of version 3 certificates, and provides multiple levels of application complexity that students need to implement. Supported algorithms (Table 1) for generating key pairs are: DSA [15], RSA [1] and ECDSA [15]. The key lengths for the DSA algorithm range from 1024 to 2048 bits. The key sizes for the RSA algorithm range from 1024 to 4096 bits. Supported sets of elliptic curves for the ECDSA algorithm are: X9.62, SEC, and NIST.

Table 1 List of supported algorithms

Public key algorithm	Hash algorithm
DSA	SHA1
RSA	SHA1
	SHA224
	SHA256
	SHA384
	SHA512
ECDSA	SHA1
	SHA224
	SHA256
	SHA384
	SHA512

Supported X509 version 3 certificate extensions are authority key identifier, subject key identifier, key usage, certificate policies, subject alternative name, issuer alternative name, subject directory attributes, basic constraints, name constraints, extended key usage and inhibit any policy.

The system is designed in such a way that there is a clear boundary in the source code between the GUI implementation and the application logic (Fig. 1). This code organization emphasizes work with certificates and relieves students of the necessity to implement GUI.

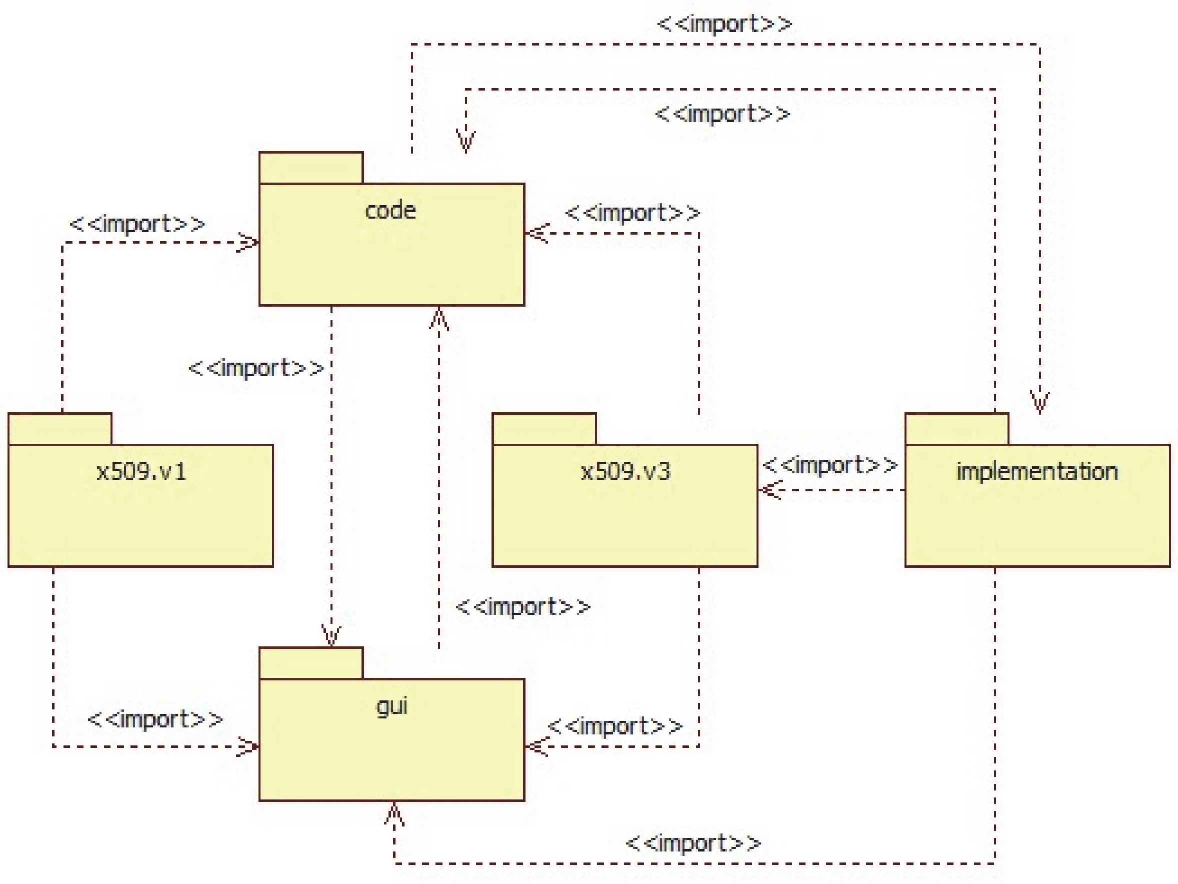


Fig. 1. UML chart of the system architecture.

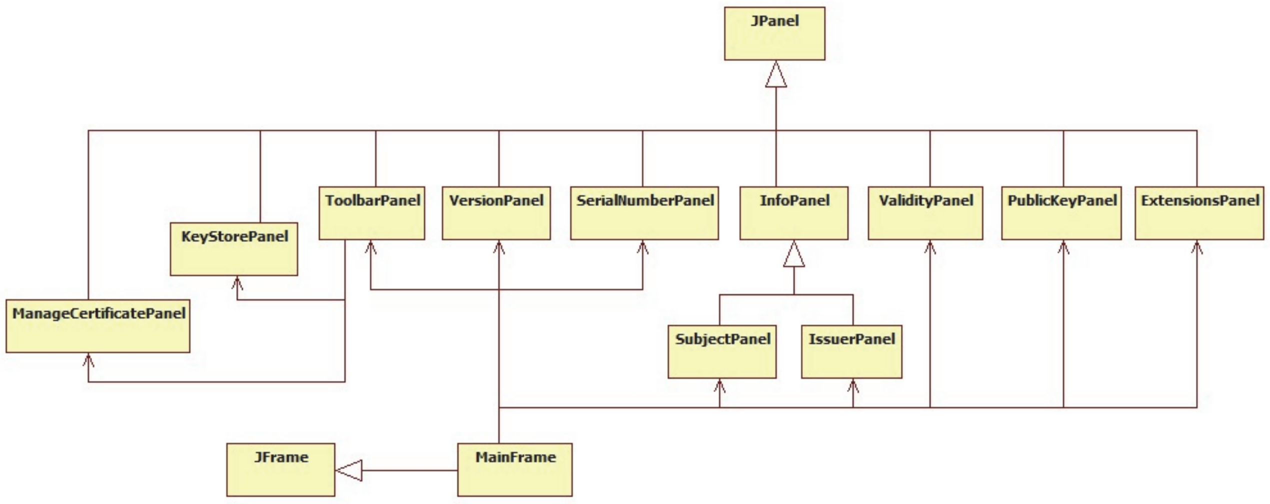


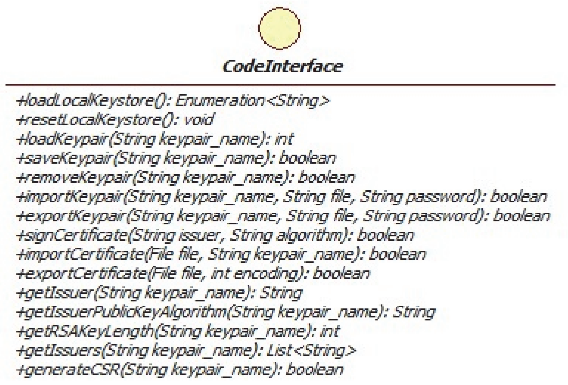
Fig. 2. UML chart of one GUI implementation part.

A. GUI Implementation

Gui package contains all the source code associated with GUI implementation (Fig. 2). The instance of MainFrame class represents the main window of the application. This is a user class extended from JFrame class from the javax.swing package. An instance of the MainFrame class contains references to all panels and dialogs along with buttons, input fields and components for displaying the data that lie on the MainFrame window. Class also contains methods with package access to retrieve and set the values of all input fields and to control the buttons. GUI configuration parameters and a reference to instance implementing CodeInterface, which methods are invoked either when users press buttons or select key pairs from a local key store on panel, are passed to the constructor of this class. GUI implementation provides appropriate built-in validation checks of the GUI's current state, enabling/disabling the buttons (depending on the state), data validation in the input fields and error reports, and invokes the corresponding methods of CodeInterface. The ToolbarListener class implements Java's ActionListener and ListSelectionListener interfaces. When a specific button is pressed or a list item is selected in the main window, it is triggered and performs all needed validation checks, and after that the defined CodeInterface method is called. The access to gui package is restricted for students. It is only permitted through publicly available constant values within the Constants class and public methods of abstract classes GuiInterfaceVX, where X represents the configured version of X509 certificates (1 or 3). This limits the set of operations students can perform within the GUI when they are implementing the application's logic.

B. *Application logic implementation*

The signatures of all methods that need to be implemented in order to provide a fully functional application are placed in the package *code* within the interface CodeInterface (Fig. 3).



```
+loadLocalKeystore(): Enumeration<String>
+resetLocalKeystore(): void
+loadKeypair(String keypair_name): int
+saveKeypair(String keypair_name): boolean
+removeKeypair(String keypair_name): boolean
+importKeypair(String keypair_name, String file, String password): boolean
+exportKeypair(String keypair_name, String file, String password): boolean
+signCertificate(String issuer, String algorithm): boolean
+importCertificate(File file, String keypair_name): boolean
+exportCertificate(File file, int encoding): boolean
+getIssuer(String keypair_name): String
+getIssuerPublicKeyAlgorithm(String keypair_name): String
+getRSAKeyLength(String keypair_name): int
+getIssuers(String keypair_name): List<String>
+generateCSR(String keypair_name): boolean
```

Fig. 3. CodeInterface specification.

CodeVX class from x509.vX package (X indicates the default version of the certificate - 1 or 3) encapsulates the interface to the application implementation and the GUI interface by inheriting CodeInterface methods, but not implementing them, and containing a protected GuiVX type field which represents an access point to the GUI and has public get/set methods for all input/output components on the graphical user interface. In order to get a functional application, it is mandatory that a source code contains a package named implementation and MyCode class in it that inherits the abstract CodeVX class and implements the inherited CodeInterface methods. MyCode class can access GUI using previously mentioned GuiVX type attribute. Complete implementation of this package can be left to students, with a constraint that the package must contain MyCode class. Regarding the specific system described in this paper, GUI is configured to work with certificates of version 1 and 3. Application logic implementation core is represented by MyCode class, as was previously mentioned. Besides the implementation of all CodeInterface methods, it also includes attributes that store the current state of the graphical user interface and helper functions. Data structure chosen for storing certificates is Java's KeyStore from java.security package. KeyStore class provides full support for storing private keys and digital certificates. Key pairs are stored according to the PKCS#12 format, and KeyStore content is stored in a .p12 file. The file is updated with each change, and is reloaded for every use. MyKeyStore utility class performs all operations that read from and write to the key store file. MyX509Cert class is responsible for generating key pairs, generating and signing certificates using specified algorithm, and extracting values of the certificate extensions. Methods of MyX509Cert class use data types from Bouncy Castle API and java.security package.
By choosing gui, code, and x509.v1 or x509.v3 package, it is easy to generate a .jar library which is delivered to students and used to develop a functional application that meets the given requirements. The application entry point is X509 class inside the code package. When user runs the application, GUI configuration parameters are initialized and MyCode class instance is created.

IV. System usage

System is designed to be used by three different user groups – professors, students and application users (Fig. 4). The first role (professors) represents authorized users who have access to source code. Only necessary part of the code is available for the second role (students) and fully functional application is eventually delivered to application users (the third role). Each role's assignments will be explained more thoroughly.

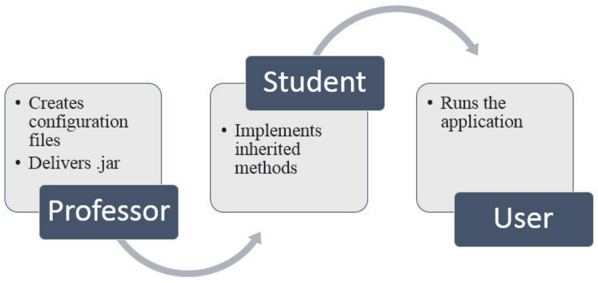


Fig. 4. User roles in the system.

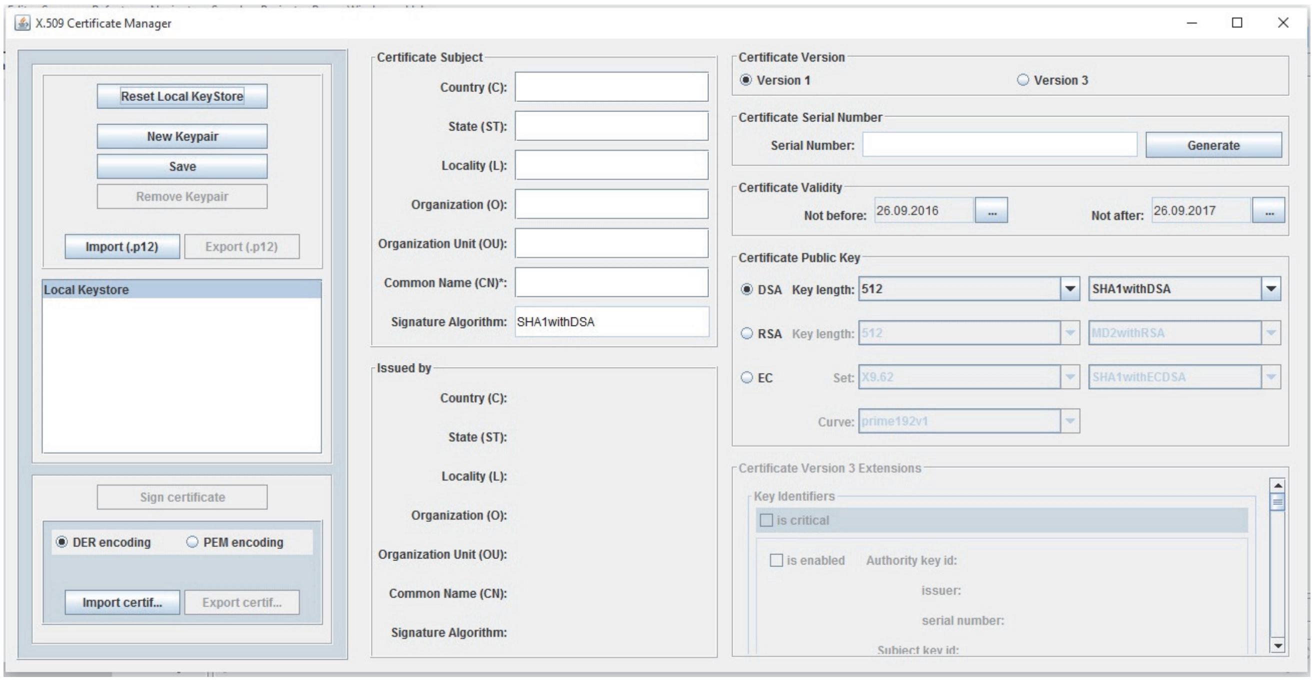
A. *Professor*

The first step is to create a configuration file with algorithm and extension parameters. The aim of this user is to export a .jar library and deliver it to students. Library contains implemented graphic user interface (GUI), methods for gathering and presenting data from and on GUI, as well as interface with methods that need to be implemented. These methods actually represent the application functionalities.

Configuration file content determines GUI appearance and defines algorithms, parameters and extensions that different students need to support within the solution. Various configuration vectors that are either loaded from file or initialized in the source code lead to different task combinations and provide diverse application complexity.

B. *Student*

Student creates new Java project and imports libraries delivered from professor. Afterwards he creates MyCode class extended from a specific base class placed in .jar file. In order to get a fully functional application, student needs to implement inherited methods according to project specification.


X.509 Certificate Manager

Reset Local KeyStore
New Keypair
Save
Remove Keypair
Import (.p12) Export (.p12)
Local Keystore

Sign certificate
(●) DER encoding ( ) PEM encoding
Import certif... Export certif...

Certificate Subject
Country (C):
State (ST):
Locality (L):
Organization (O):
Organization Unit (OU):
Common Name (CN)*:
Signature Algorithm: SHA1withDSA

Issued by
Country (C):
State (ST):
Locality (L):
Organization (O):
Organization Unit (OU):
Common Name (CN):
Signature Algorithm:

Certificate Version
(●) Version 1 ( ) Version 3

Certificate Serial Number
Serial Number: Generate

Certificate Validity
Not before: 26.09.2016 Not after: 26.09.2017

Certificate Public Key
(●) DSA Key length: 512 SHA1withDSA
( ) RSA Key length: 512 MD2withRSA
( ) EC Set: X9.62 SHA1withECDSA
Curve: prime192v1

Certificate Version 3 Extensions
Key Identifiers
☐ is critical
☐ is enabled Authority key id:
issuer:
serial number:
Subject key id:

Fig. 5. Initial application screen.

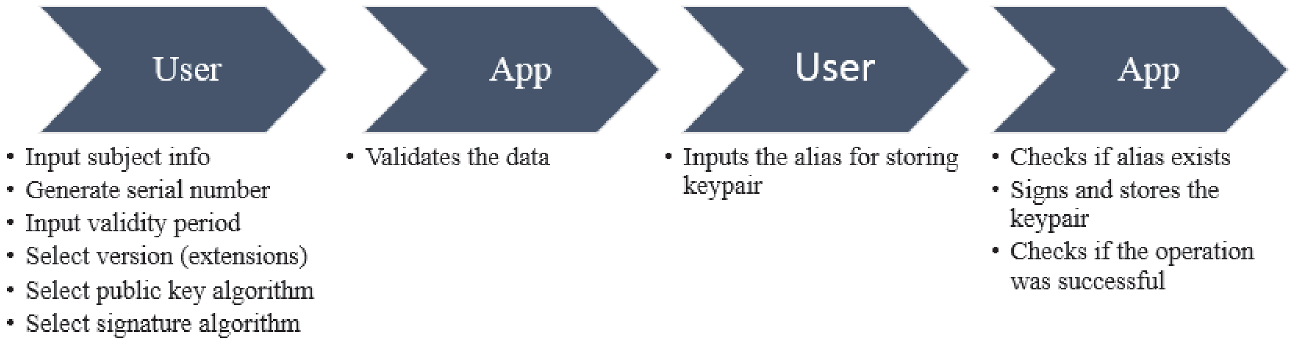

User
• Input subject info
• Generate serial number
• Input validity period
• Select version (extensions)
• Select public key algorithm
• Select signature algorithm
App
• Validates the data
User
• Inputs the alias for storing keypair
App
• Checks if alias exists
• Signs and stores the keypair
• Checks if the operation was successful

Fig. 6. Use case scenario – Generating new key pair.

C. Application user

Once students implement the certificate part of the application it can then be used for its defined purpose, i.e. handling X509 certificates. Fig. 5 shows how an initial application screen looks like and once the functionalities are implemented it can be used by anyone.

As described previously, one of the functionalities is to implement new key pair generation. This use case scenario is represented in Fig. 6. First, the user inputs subject info, generates serial number, inputs validity period, selects version and public key algorithm, as well as signature algorithm. If the user chose version 3, different set of extensions can be chosen also. Afterwards user saves the key pair and the application checks if the input was regular. If data wasn't valid, the error is reported and new input is enabled. When the input is correct, key pair is signed (self-signed certificate) and stored into local key store. If operation has executed successfully, a message is shown, otherwise an error is reported.

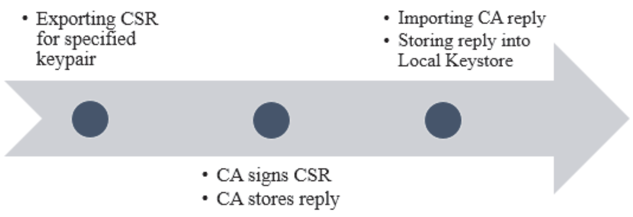

• Exporting CSR for specified keypair
• Importing CA reply
• Storing reply into Local Keystore
• CA signs CSR
• CA stores reply

Fig. 7. Use case scenario – Signing certificate.

Fig. 7 shows a key pair signing process in details. After choosing a specific key pair from local key store, user creates Certificate signing request (CSR) and exports it to a chosen file. After that user needs to choose a key pair that can act as a CA and select option to sign a CSR and then choose the previously saved file. CSR data can be displayed on GUI and edited by CA. After the CSR is signed, CA stores the reply into a file which afterwards can be imported and stored into local key store, thus adding a signature to the selected key pair.

V. CONCLUSION

The main goal of the software system for which the development and implementation were described in this paper was to enable students to make a functional application for handling X509 certificates, but without the overhead of developing a GUI. In that way students are allowed to get a much better insight into one of the important and current topics in the area of information security. In the previous school year students had to program GUI part as well, but they were paired up in teams. However, that approach led to a common situation where one student programmed GUI part and the other programmed security part, which was not the intended outcome. Usage of the system described in this paper was meant to eliminate the previously described problem and allow all students who decide to work on the project to focus on the security part of the application. Since the project component of the course is not mandatory about 45 percent of students (144 out of 314) completed the project successfully in the first year of the newly developed system usage. An additional analysis will be done in order to determine what needs to be changed to make this percentage higher in the following years.

REFERENCES

[1] T. Dierks, E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", 2008, available at: https://tools.ietf.org/html/rfc5246, accessed April 2018.

[2] W. Stallings, "Cryptography and Network Security Principles and Practices," 7th ed., Pearson Education, 2016.

[3] Public key infrastructure, aailable at: https://en.wikipedia.org/wiki/Public_key_infrastructure, acessed April 2018.

[4] J. V. Boettcher, A. Powell, "Digital Certificates What Are They, and What Are They Doing in My Browser?", available at: http://www.cren.net/crenca/docs/syllabus.pdf, accessed: April 2018.

[5] M. Vukasović, B. Veselinović, and Ž. Stanisavljević, "A development of a configurable system for handling X509 certificates", TELFOR, 2017. (in Serbian)

[6] ITU-T X.509 Recommendation, available at: http://www.itu.int/rec/T-REC-X.509-201610-I, accessed: April 2018.

[7] D. Cooper et al, „Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL)“, available at: https://tools.ietf.org/html/rfc5280, accessed: April 2018.

[8] Java programming language, available at: https://java.com/en/, accessed: April 2018.

[9] Bouncy Castle API, available at: https://www.bouncycastle.org/, accessed: April 2018.

[10] JDatePicker library, available at: https://jdatepicker.org/, accessed: April 2018.

[11] K. Moriarty et al, „PKCS #12: Personal Information Exchange Syntax v1.1“, available at: https://tools.ietf.org/html/rfc7292, accessed: April 2018.

[12] NIST FIPS 197, available at: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, accessed: April 2018.

[13] M. Nystrom et al, „PKCS #10: Certification Request Syntax Specification“, available at: https://tools.ietf.org/html/rfc2986, accessed: April 2018.

[14] B. Kaliski, „PKCS #7: Cryptographic Message Syntax“, available at: https://tools.ietf.org/html/rfc2315, accessed: April 2018.

[15] NIST FIPS 186-4, available at: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, accessed: April 2018.