



uOttawa

FACULTY OF ENGINEERING – ELECTRICAL AND COMPUTER  
ENGINEERING

# Digital Signatures for Mobile Users

---

Master of Engineering Project Report in Cryptography

December 2014

*Submitted to Professor Carlisle Adams in partial fulfillment of the  
requirements for the course ELG 5900*

*Arulprakasam Ravinthiran(7612997)  
University of Ottawa*

# Contents

1. ABSTRACT .....	1
2. INTRODUCTION .....	1
3. BACKGROUND .....	2
3.1 KEY PAIR GENERATION .....	4
3.2 CERTIFICATE CREATION: ONE-TIME AND HISTORIC .....	5
3.3 SIGNATURE CREATION.....	5
3.4 SIGNATURE VERIFICATION .....	6
4. EXPERIMENTAL SET UP .....	7
5. WORKING OF THE PROJECT.....	9
5.1 WORKING OF THE SERVER.....	9
5.1.1 USER REGISTRATION.....	10
5.1.2 CERTIFICATE CREATION .....	10
5.1.3 SIGNATURE CREATION REQUEST.....	11
5.1.4 SIGNATURE VERIFICATION .....	12
5.2 WORKING OF THE CLIENT.....	13
5.2.1 HOME SCREEN .....	13
5.2.2 USER REGISTRATION SCREEN .....	14
5.2.3 CERTIFICATE CREATION SCREEN.....	15
5.2.4 CO-SIGNER SIGNATURE CREATION SCREEN .....	15
6. DISCUSSION ON PROBLEMS ENCOUNTERED DURING PROJECT EXECUTION.....	16
7. FUTURE WORK.....	17
8. CONCLUSION .....	18
9. ACKNOWLEDGEMENTS .....	18
10. REFERENCES .....	19

## LIST OF FIGURES

Figure 1 Working of the project: Overall skeleton .....	2
Figure 2 User registration flowchart of server .....	10
Figure 3 Certificate creation flow chart of server .....	10
Figure 4 Signature creation request flow chart of server .....	11
Figure 5 Signature verification flow chart of server .....	12
Figure 6 Home Screen .....	13
Figure 7 Signature status screen .....	14
Figure 8 User registration screen .....	14
Figure 9 Certificate creation screen .....	15
Figure 10 Co-signer signature creation screen.....	15
Figure 11 Progress window and the corresponding home screen.....	16

## LIST OF TABLES

Table 1 Project Specifications.....	7
Table 2 Server - Client functions .....	9

# 1. ABSTRACT

This report is about the proof-of-concept (POC) for the cryptographic project proposed in [1] by Professors Carlisle Adams and Guy-Vincent Jourdan. It includes a short theoretical background of the proposal, the experimental setup, the working of the desktop version of the project with a list of project specifications and screen shots, a discussion on problems encountered and future work.

# 2. INTRODUCTION

Digital signatures are the cryptographic techniques that enable signing of documents and verification of the same by others. The efficiency of the signature algorithms lie in the non-repudiation of the signers and the binding of the public and private keys to a user.

The proposal had given a few algorithms to create mobile digital signatures, where any user could generate digital signatures using his biometrics in any platform, say mobile or tablet or desktop with the same set of keys for all the platforms without storing the private keys anywhere. The main purpose of the proposal was to keep rid of the third party Certificate Authority (CA) by securely sending the keys to the users through independent channels and regenerate the keys to achieve true mobility of the user.

The POC that began in May 2014 continued till September for the desktop version and began its mobile version from then. This report focuses on the desktop version.

This POC has reached its goal by keeping the key pair generation in the client side without compromising the regeneration-at-all-needed-times feature which is the backbone of the proposal. The POC pretended texts in place of biometrics and integration with the finger print sensors will be fruitful had right techniques to generate unique keys as discussed in [2] been found. (Issues found on biometric sensor integration are discussed in the 6<sup>th</sup> section.)

In short, this desktop version of the POC does the following important functionalities among others:

- i. Certificate creation at the server end,
- ii. Key pair generation at the client end,
- iii. Signature creation at the client end,
- iv. Signature verification at the server end

Theoretical background of the proposal, the working of the POC and discussions on it are explained in the further sections.

### 3. BACKGROUND

Key pair generation, digital certificate creation, digital signature creation and digital signature verification algorithms used in the POC are explained in this section and the exact places where these will take place will be seen in section 5.

Overall process flow of the POC is shown below.

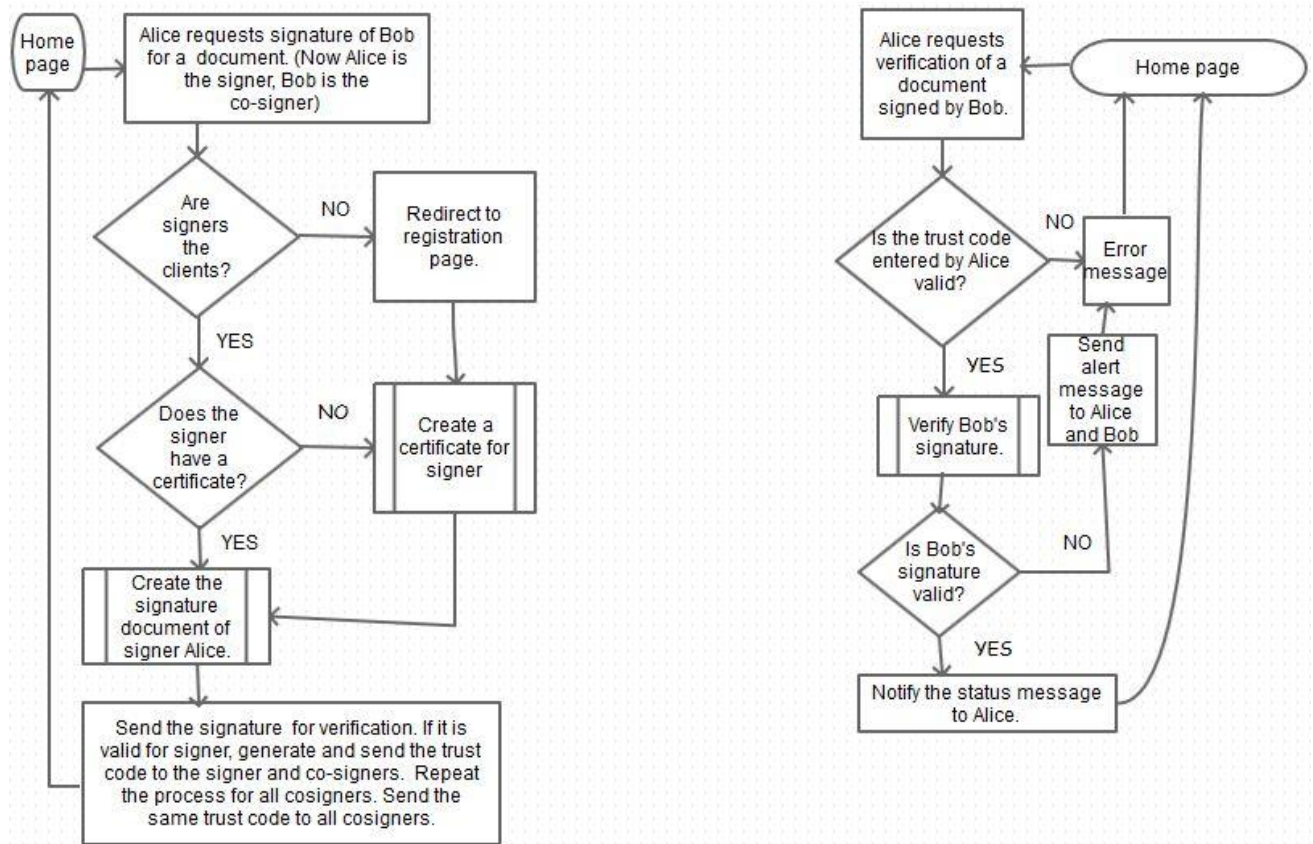


Figure 1 Working of the project: Overall skeleton

The flow charts (left for digital signature creation and right for digital signature verification) are self-explanatory given that the following business constraints hold true for the POC in consideration:

1. Digital signatures (**referred as signatures from this point**) can be created for a user provided that the user has a digital certificate (**referred as certificate from this point**) already.
2. The user requesting creation of a signature of a document, known as **signer**, can initiate signatures for the same document from others, known as **co-signers**, whose e-mail addresses are known to the user. Instead, the user can just sign a document without including co-signer for the same and just ask the company to validate the signature.
3. All the signers are registered members of the company.
4. When a signer initiates signatures for a document, first the signature is created for her. **Only if the signer's signature is valid, signature creation will be initiated for the co-signers.**
5. Whenever a signer initiates signatures, she must have a certificate for her signature creation whereas the co-signer may or may not possess a certificate but should be a registered member. When the signature of the signer is valid, certificate creation of the co-signer will be initiated if he has not one already.
6. Documents are mapped using a code called **trust code**. A document, for which signatures are requested for a number of users, has the same trust code. **No modifications** are allowed for the document when creating signatures for it by the signers.
7. Every user is identified by the e-mail address provided at the time of registration.
8. Users have mobiles to receive the key.
9. The channels chosen for the POC include the user's **e-mail address and mobile number**.
10. As soon as a signature is created, it is verified in the server. If it is invalid, **alerts are sent to the signature initiator viz. signer and the particular co-signer but not the remaining co-signers**. Say if Arul requests signature for a document from Shyam and Bernado and if Bernado's signature is invalid for some reasons, then alerts are sent only to Arul who is the signer and Bernado, the particular co-

signer whose signature is invalid and not Shyam.

11. Any user who has a trust code for a document can see the details of the signature for that document. But the trust codes are sent only to the persons involved in a signature.

The proposal had given both RSA and DSA algorithms for its implementations. The POC used **RSA** among the options as it was easier to test. The comparison of the performance of the proposal using RSA and DSA is out of scope of this report and could be left for future analysis.

### 3.1 KEY PAIR GENERATION

The key pair generation algorithm used in the POC accepts three input strings p1, p2 and S, where p1 and p2 are the keys sent to the user and **S is the secret sentence** entered by the user. The following steps summarize the algorithm:

1. Set the 512 bit prime and random number r.
2. Compute the public key e by concatenating p1 and p2.  
 **$e = p1 \parallel p2$ .**
3. Compute 512 bit  $p = (\text{first safe prime bigger than } S^{p1} \bmod r)$ .
4. Compute 512 bit  $q = (\text{first safe prime bigger than } S^{p2} \bmod r)$ .
5. Compute 1024 bit modulus  $n = p * q$ .
6. Compute 1024 bit  $\Phi(n) = (p-1) * (q-1)$ .
7. Compute the 1024 bit private key  **$d = e^{-1} \bmod \Phi(n)$ .**

Note that d and e are the private and the public keys respectively and  $\text{GCD}(e, \Phi(n))$  should be equal to 1 and so the last bit of p2 should not be even. In the POC, we had taken this bit one of 1, 3 and 7 values.

## 3.2 CERTIFICATE CREATION: ONE-TIME AND HISTORIC

Certificates are created by the company using the user's name,  $n$  and  $e$  parameters. They are used for signature creations later on.

But for the first time a user registers, the company should authenticate the user without knowing her private key.

This **one-time certificate creation** happens in the following ways:

1. Create an initial one time certificate  $C$  in the client with only the user's name and modulus  $n$ . Say  $C = (\text{"Arul"}, n)$ , where "Arul" is the user's name,  $n$  is his modulus value generated using the steps of section 3.1 above.
2. Compute file  $Y_0 = \text{Sig}(C)$  in the same client, where Sig is the signature of the user, Arul in this case, of the certificate  $C$  with the private key  $d$  generated in section 3.1. Note that signature creation happens by private key.
3. Send both  $C$  and  $Y_0$  to the company's server.
4. Compute  $H(C)$  and  $(Y_0)^e \bmod n$ , where  $H$  denotes a hashing function in the server.
5. If  $H(C)$  is equal to  $(Y_0)^e \bmod n$  then the user is authenticated for the set of key pairs  $d$  and  $e$ .

If a user is authenticated using this one-time certificate creation algorithm, then a certificate (I refer this as a user's HISTORIC certificate) can be created for the same user that has all the necessary details related to the user and the certificate, say user name, public key  $e$ , modulus  $n$ , key generation algorithm name, validity period of the certificate, date and time of creation of the certificate, certificate provider viz. the company's public key and so on.

## 3.3 SIGNATURE CREATION

Signatures are created with the help of private keys of the user in the client. This happens in the following way assuming that  $D$  is the document to be signed:



1. Compute  $\mathbf{h} = \mathbf{HASH}(\mathbf{D})$  using a suitable hashing algorithm.
2. Compute the 1024 bit signature file  $\mathbf{y} = \mathbf{h}^d \bmod \mathbf{n}$ , where  $d$  is signer's private key. Retrieve public keys  $n$  and  $e$  from the user's certificate. Compute  $p1$  and  $p2$  from  $e$ . Obtain  $s$  from the user. Thus calculate  $d$ .

This calculation is as follows:

$$e = p1 \parallel p2$$

$$p = \text{first (safe) prime number bigger than } s^{p1} \bmod r$$

$$q = \text{first (safe) prime number bigger than } s^{p2} \bmod r$$

$$n = p * q$$

$$\Phi(n) = (p - 1) * (q - 1)$$

$$d = e^{-1} \bmod \Phi(n)$$

3. Send the user's signature file  $y$ , document  $D$  to the server or verifier for verification.

### 3.4 SIGNATURE VERIFICATION

Whomsoever the verifier might be, the company's server initially verifies the signed file by these steps:

1. Get unsigned document  $D$ , signature file  $y$  and certificate of the signed user  $C$ .
2. Read the public keys of the signed user  $n$  and  $e$  from  $C$ .
3. Compute  $\mathbf{h} = \mathbf{HASH}(\mathbf{D})$  using the specified hash algorithm of signature creation.
4. Compute 1024 bit  $\mathbf{z} = \mathbf{y}^e \bmod \mathbf{n}$ .
5. Check if  $\mathbf{h} = \mathbf{z}$  condition is satisfied. If yes, the signature file is valid.

## 4. EXPERIMENTAL SET UP

The POC was implemented in an Intel Pentium CPU B950 @ 2.10GHz processor, 32 bit windows operating system. Both the server and the clients were in the same machine.

The specifications related to the project are given below:

*Table 1 Project Specifications*

Programming language (both server and client)	Java on platform JDK 7
Type of application	Desktop – based
Development Platform	Microsoft Windows 7
Target platforms	All available desktop platforms including all versions of Microsoft windows, Apple iMac, LINUX
Development tool	Eclipse Helios IDE (Integrated Development Environment)
Database	MySQL 5.6
Server	Three-tier Java application program
Client	Three-tier Java application with <b>SWING</b> framework that handles the GUI (Graphical user interface)
Server to Client communication	Sockets and thread (The server is multi-threaded so that it can handle many clients at a time)
SMS sender	“SMS Gateway” Android application installed in an android platform of a SAMSUNG mobile.
Key pair generation algorithm	Modified version of RSA (see section 3.1 above and the proposal in [1] )
Certificate creation, Certificate verification, Signature creation and Signature verification helper	Open source Bouncy Castle provider for Java
Hashing algorithm for signature creation and verifications	SHA256 provided by Bouncy Castle
Certificate specification	X.509 certificates created with the following parameters

	<p>for the root certificate C=CA, L=Ottawa, ST=Ontario, O=DigSigMobile, OU=DigSigMobile Development Unit,E=digsigmobile@gmail.com</p> <p>The root certificate is self-signed with one year warranty and is stored inside the folder called “Keys” in the server.</p>
Prime number generator	Prime number generation methods provided by Bouncy Castle provider
Types of prime numbers used for key pair generation	Safe primes of the form $2p+1$ where $p$ is also a prime (see in section 3.1 above and further details of the same in [1] )
Random prime number $r$ for key pair generation (512 bit number as seen in section 3.1 above. This number is chosen after various considerations by automated tests)	603706535958583574142044615180498973275792 29372171765262372195957732692125815878351 402731565366899041040973795656942771268234 57299655460482690745373593131
Public key modulus bit length ( $n$ as seen in section 3.1 above)	1024
Number of times the generated prime number passes the primality test	20
Finger print sensor specifications used	U.are.U 5160 Digital Persona finger print sensor for windows and android, U.are.U 4500 for windows [3]

## 5. WORKING OF THE PROJECT

The POC does the following in the server and the client ends:

*Table 2 Server - Client functions*

Server	Client
<ol style="list-style-type: none"><li>1. Database and all related activities</li><li>2. Certificate creation (except the one-time certificate as described in section 3.2)</li><li>3. Signature verification</li><li>4. Exception handling</li><li>5. SMS sending</li><li>6. E-mail sending</li><li>7. Sockets and thread handling for communication with the client</li></ol>	<ol style="list-style-type: none"><li>1. One-time certificate creation</li><li>2. Key pair generation</li><li>3. Signature creation</li><li>4. All user interface activities</li><li>5. Exception handling</li><li>6. Sockets and thread for communication with the server</li></ol>

**Note that certificate revocations are not handled in the POC** (Discussion related to the same are in section 7).

### 5.1 WORKING OF THE SERVER

Now the activities of the server at high level related to each functionality viz. user registration, certificate creation and signature verification are shown in this section by flow charts which are self-explanatory.

**Note that actual signatures are created in the client but before that ‘signature creation requests’ are sent by the client to the server** that will validate the signers and the documents if necessary.

Also **one-time certificates are created in the clients for the first time the users register**, upon the verification of which a historic certificate of one-year validity for that request is created and stored in the database of the server.

The business constraints detailed in the section 3 holds true for all the activities of the server.

### 5.1.1 USER REGISTRATION

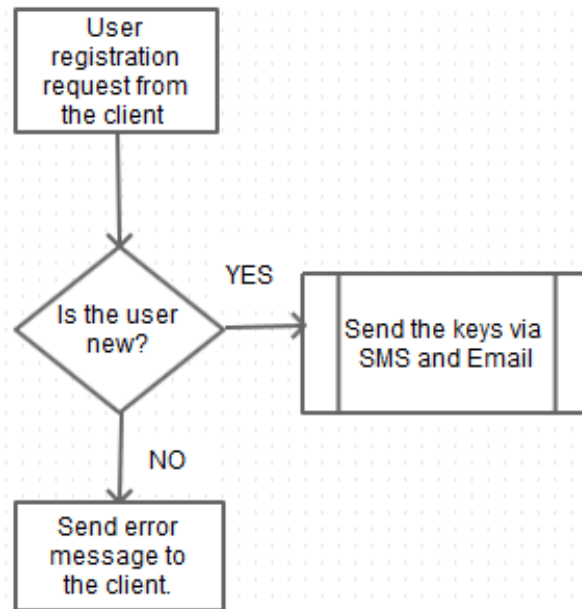


Figure 2 User registration flowchart of server

### 5.1.2 CERTIFICATE CREATION

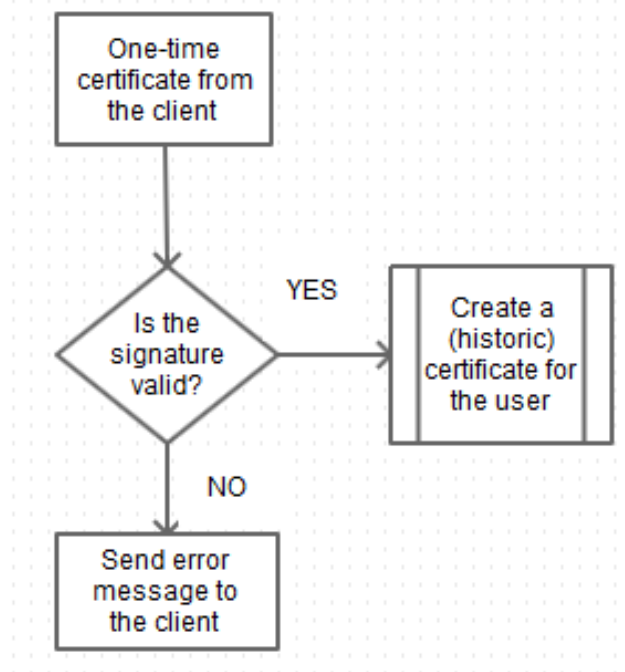


Figure 3 Certificate creation flow chart of server

### 5.1.3 SIGNATURE CREATION REQUEST

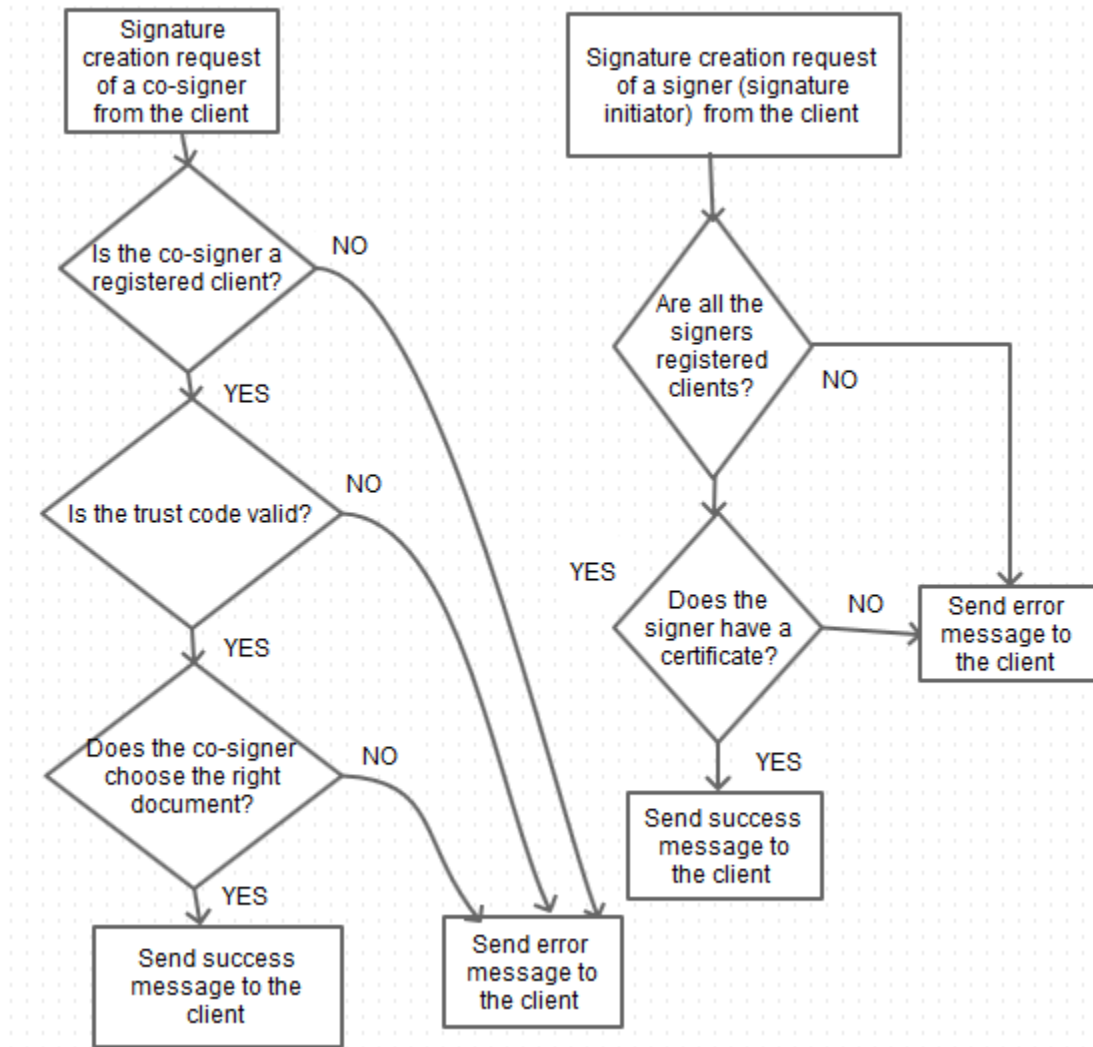


Figure 4 Signature creation request flow chart of server

The flow chart on left is the request from the client for a co-signer and the right is the request from a client for a signer or signature initiator. Note that the request for co-signers would have a trust code already.

### 5.1.4 SIGNATURE VERIFICATION

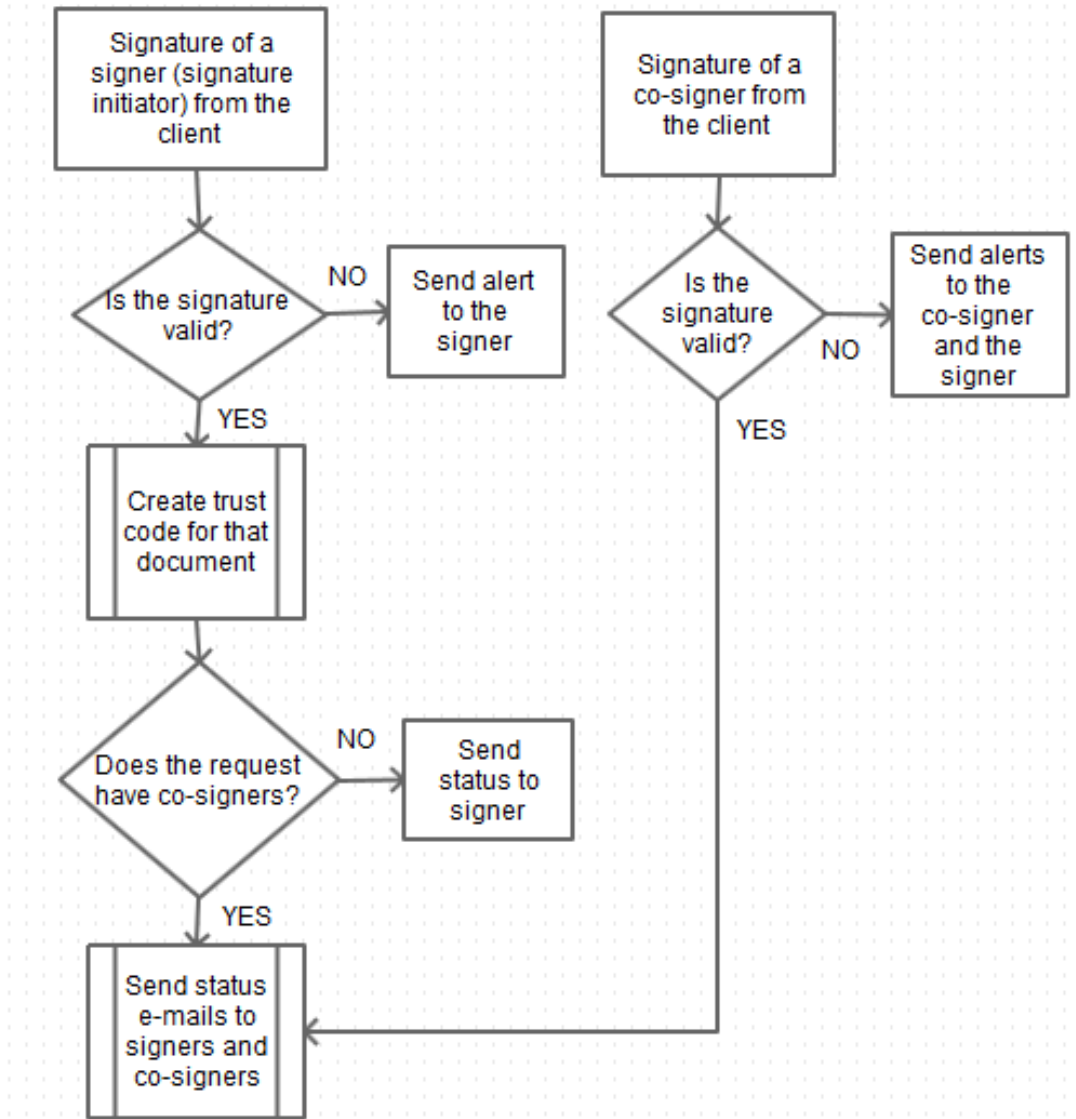


Figure 5 Signature verification flow chart of server

Note that a trust code should be created when there is a signature from the client that belongs to a signature initiator (See the business constraints in section 3).

## 5.2 WORKING OF THE CLIENT

The screen shots of the user interface (UI) of the SWING framework of the client and the functionalities corresponding to each UI are discussed in this section.

### 5.2.1 HOME SCREEN

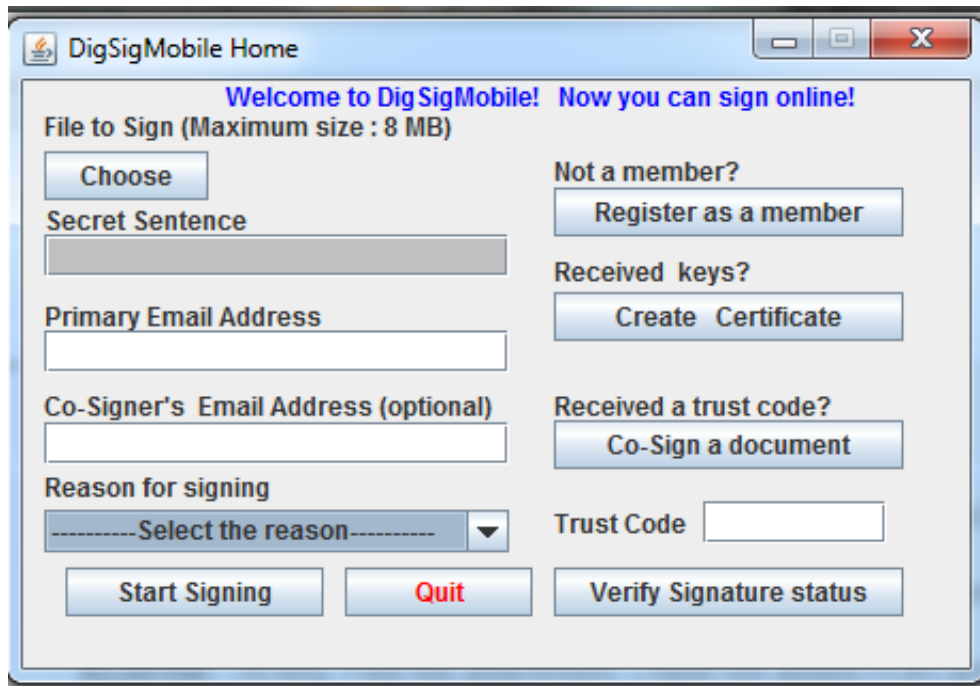


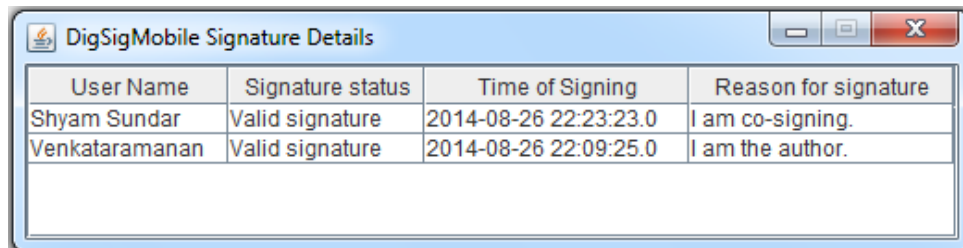
Figure 6 Home Screen

“DigSigMobile” is the imaginary company for which the POC had been developed. The home screen allows any user to sign a document or navigate to registration or certificate creation or co-signing a document or check the status of a signature if trust code for that document is known.

If the user clicks “Start Signing” button, there will be validation of the entered e-mail addresses as shown in section 5.1.3 (figure 4 flow chart on the right) and signature will be created in the client upon getting a success message from the server.



When the user enters a trust code and clicks “Verify signature status”, the client displays the status of the signatures corresponding to the trust code as shown below:



The image shows a window titled "DigSigMobile Signature Details". It contains a table with four columns: "User Name", "Signature status", "Time of Signing", and "Reason for signature". There are two rows of data. The first row shows "Shyam Sundar" with a "Valid signature" at "2014-08-26 22:23:23.0", with the reason "I am co-signing.". The second row shows "Venkataramanan" with a "Valid signature" at "2014-08-26 22:09:25.0", with the reason "I am the author.". Below the table is an empty space.

User Name	Signature status	Time of Signing	Reason for signature
Shyam Sundar	Valid signature	2014-08-26 22:23:23.0	I am co-signing.
Venkataramanan	Valid signature	2014-08-26 22:09:25.0	I am the author.

Figure 7 Signature status screen

## 5.2.2 USER REGISTRATION SCREEN



The image shows a window titled "DigSigMobile Registration". It contains a form with various fields for user registration. The fields are arranged in two columns. The left column includes "Name \*", "Primary Mobile Number \*", "Primary Email Address \*", "Residential Address \*", and "City \*". The right column includes "Family Name", "Secondary Mobile Number", "Secondary Email Address", "Province \*", "Country \*", and "Postal Code \*". There are also "Close" and "Register" buttons. A legend indicates that fields marked with an asterisk are required.

**Name \***

**Primary Mobile Number \***

**Primary Email Address \***

**Residential Address \***

**City \***

**Family Name**

**Secondary Mobile Number**

**Secondary Email Address**

**Province \***

**Country \***

**Postal Code \***

**\* Required Field**

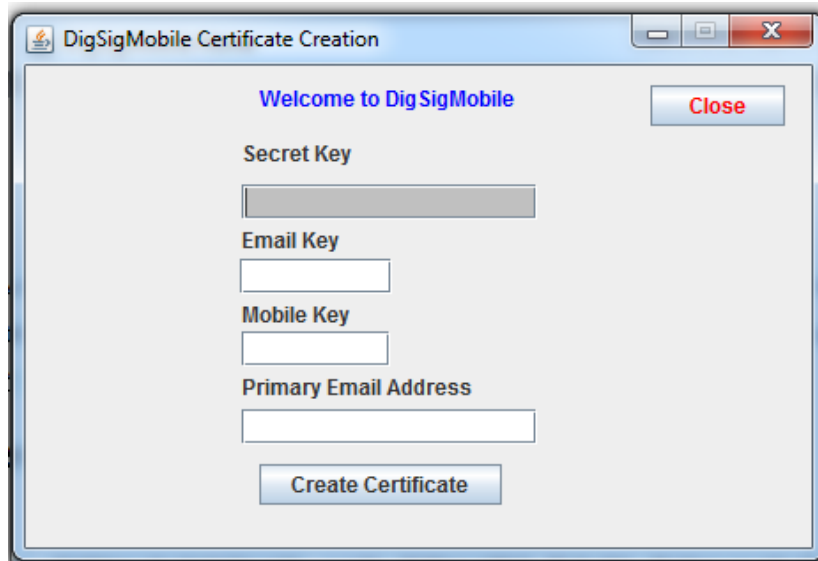
**Register**

**Close**

Figure 8 User registration screen

As soon as the user clicks “Register” button, after validating for all the required fields, the client sends the request to the server, which sends back keys as explained in section 5.1.1 if the user is new.

### 5.2.3 CERTIFICATE CREATION SCREEN

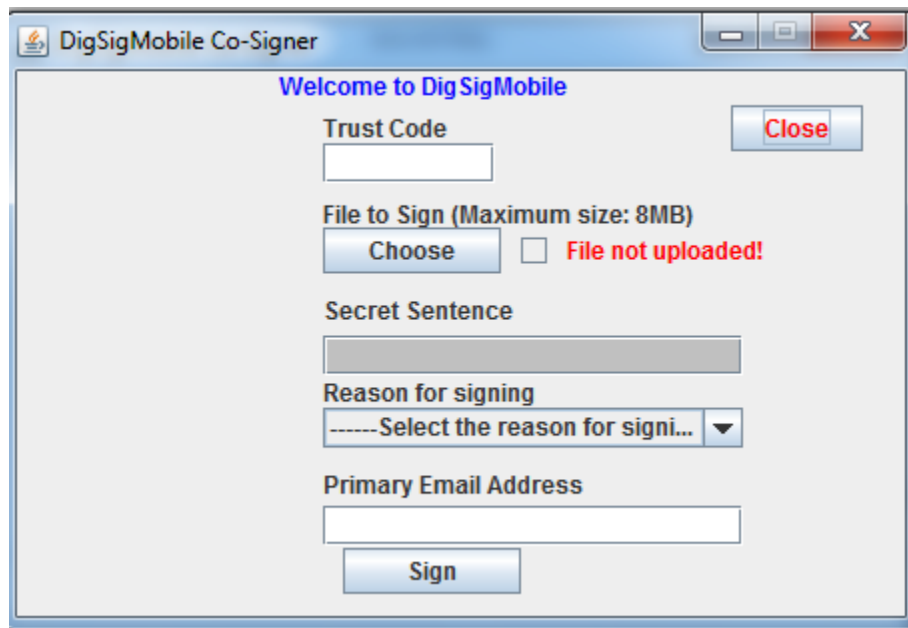


The screenshot shows a window titled "DigSigMobile Certificate Creation". It has a "Welcome to Dig SigMobile" header with a "Close" button. Below the header are four input fields: "Secret Key", "Email Key", "Mobile Key", and "Primary Email Address". At the bottom is a "Create Certificate" button.

Figure 9 Certificate creation screen

As soon as the user receives her keys as SMS and e-mail, she could create certificate using this screen. First the entered keys are validated by sending a request to the server. If the server validated the keys, then one-time certificate is created in the client (see section 3.2) and sent to the server which would create a historic certificate (as seen in section 5.1.2).

### 5.2.4 CO-SIGNER SIGNATURE CREATION SCREEN



The screenshot shows a window titled "DigSigMobile Co-Signer". It has a "Welcome to Dig SigMobile" header with a "Close" button. Below the header are several fields and controls: "Trust Code" with an input field; "File to Sign (Maximum size: 8MB)" with a "Choose" button and a checkbox labeled "File not uploaded!"; "Secret Sentence" with a text area; "Reason for signing" with a dropdown menu showing "-----Select the reason for signi..."; and "Primary Email Address" with an input field. At the bottom is a "Sign" button.

Figure 10 Co-signer signature creation screen

Co-signers utilize this screen to create their signature. First the entered trust code, e-mail address and the chosen document are validated as explained in section 5.1.3 and then the signature is created upon getting success message from the server.

## 6. DISCUSSION ON PROBLEMS ENCOUNTERED DURING PROJECT EXECUTION

Some of the problems encountered during the execution of the POC are discussed in this section:

1. Signature creation and certificate creation takes a considerable amount of time **according to the processor**. It ranges from a **minimum of 1 minute to a maximum of 4 minutes**. Also time varies for the same machine when its battery is in full charge and less than 20 %. This time is due to the **key pair generation** in the client that does all the safe prime number generation, public and private keys creation. It could be improved **if prime numbers are used instead of safe primes** but that would undermine the efficiency of the proposal. The reason for this claim is that creation of safe primes takes at least twice the time and computations as creation of primes.
2. When the client creates a signature or the server is creating a certificate, it is **important not to overload the same thread with other tasks**. Otherwise, it would take much time for the client or server to create the certificate or signature. In this POC, this is carried out by displaying progress bars in the client and disabling the buttons in the home page as shown below:

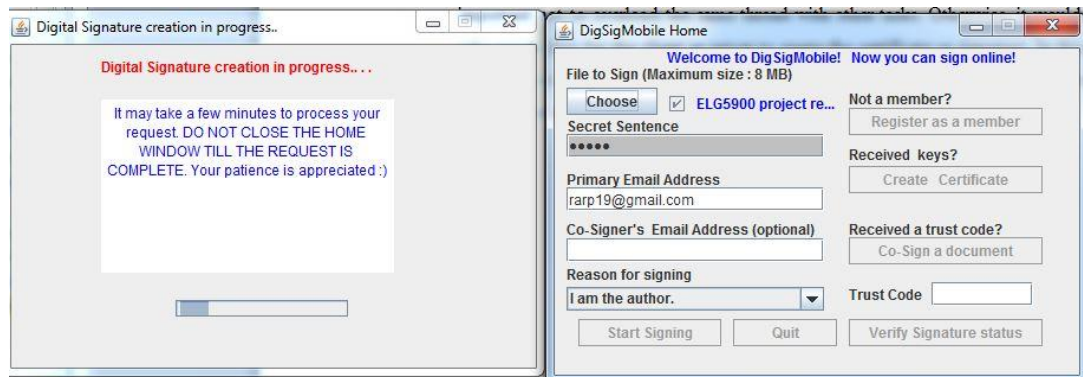


Figure 11 Progress window and the corresponding home screen

3. The finger print sensors used [3] could not give a unique string per finger of a user. So it was difficult to use the sensor for biometric integration of the POC. The 3<sup>rd</sup> party sensor had masked the string that was produced for any finger with various headers and footers and gave arbitrary bits each time. So application of Hamming distance or any other error correction techniques was seemingly infeasible. Suitable fuzzy extractors as explained in [2] would be useful had the input string been unmasked.

## 7. FUTURE WORK

The POC could be extended in the future, including enhancements to the existing POC and new features, as explained in this section. Some of them could also be viewed as limitations of the existing POC.

1. Functionalities for **certificate revocations** are to be included in the server. Currently the server only checks the validity of the certificate but not keeps track of CRLs (Certificate Revocation List) that are an essential part of a CA.
2. Let signature creations be allowed for **modified version of the document** signed by the signature initiator. Currently no modification is allowed for the document that is to be co-signed.
3. **SMS sending** happens through an android application (see section 4) that utilizes the network carrier for the same. This could be replaced by a suitable application say web-based.
4. The documents that need to be signed by co-signers be attached as e-mails to the co-signee. Currently only trust code and the signer information is sent in the e-mail and not the document.
5. **Some other independent channel** needs to be chosen for users who do not own a mobile. This POC currently uses mobile and e-mail as the channels. The same holds true for e-mail addresses. Care should be taken for e-mail addresses that are no longer valid.

6. **Suitable fuzzy extractors** [2] could be used for biometric integration. Currently secret sentences are used for key generations and signature creations.
7. **Web-based applications** could be developed for the same POC. Note that the current desktop version is built using JAVA and therefore it works in all platforms as mentioned in the section 4.
8. **Mobile version** of this POC exists now for Android. Other platforms could be considered.
9. Applications for other devices, say tablet can be designed.
10. A UI could be designed to show the end user her created signature. Currently she just gets e-mails and dialogs on the validity of her created signature.

## 8. CONCLUSION

Working of the developed POC for desktop version of the given proposal had been explained with flow charts for the server and screen shots for the client. Sufficient background on the algorithms used, discussion of the problems encountered and some suggestions in terms of future work of the existing POC had also been pointed out.

## 9. ACKNOWLEDGEMENTS

I gratefully acknowledge my peers, University of Ottawa undergraduate students, who worked for the POC during the four month period of summer 2014 from May to August. We had a status meeting every two days per week and it gave the opportunity to work as a team, analyze the requirements and distribute the tasks. Special thanks are to **Bernado Augusto Pereira de Macedo** ([Bpere016@uottawa.ca](mailto:Bpere016@uottawa.ca)), whose contribution by suggesting and modifying the Bouncy Castle provider is appreciable. Also I acknowledge **John Merkowsky** ([johnmerkowsky@gmail.com](mailto:johnmerkowsky@gmail.com)) for his initial works in the GUI of SWING framework.

## 10. REFERENCES

- [1] Adams, C. and Jourdan, G.-V, “Digital Signatures for Mobile Users”, 27th IEEE CCECE 2014: Symposium on Computers, Software and Applications, Toronto, Canada, May 2014.
- [2] Dodis, Y., R Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy Extractors: How to Generate Strong Keys From Biometrics and Other Noisy Data”, SIAM Journal on Computing, vol. 38, iss. 1, March 2008, pp. 97-139.
- [3] Digital persona U.are.U 4500, U.are.U 5160 developer guide, Digital Persona Inc.,