SMS-based Certificate Authority

Semester Thesis

Imene Batata

July 4, 2011

Advisors: **Sacha Trifunovic**
Supervisor: **Prof. Dr. Bernhard Plattner**

Computer Engineering and Networks Laboratory, ETH Zurich

Abstract

In the context of mobile wireless networks, and more particularly in challenged networking environments where communication frequently breaks, mobile devices directly exchange content when they are within communication range. We refer to this type of networking as opportunistic networks. Considering the increase in popularity of wireless mobile devices, opportunistic distribution of content appears very beneficial to maintain mobile communication in case of infrastructure shut-downs. Indeed, even when the infrastructure is absent, unreliable, expensive or censored, the opportunistic content dissemination can still proceed since it does not rely on infrastructure.
Nevertheless, malicious users can use these networks to distribute illegitimate content such as spam. To prevent undesired content from spreading and enable applications to run on opportunistic networks, we propose to hold users accountable for their action by using an identity management. We design an SMS based protocol to enable mobile devices to register and obtain certificates from a central infrastructure. The certificates bind the public key of the user with his phone number. A proof of concept was implemented using Android technologies and resulted in a maximum number of 10 SMSs to be sent per request.
In this report, we will elucidate the background of the project, describe the protocol, show an implementation of the infrastructure, analyze the results and suggest for future improvements.

Contents

Chapter 1

Introduction

Recent statistics [1] show that already 80% of the U.S. consumers own a smartphone, or plan to buy one in the next six months for web browsing, email, and navigation services. One can imagine that this percentage will only keep growing and maybe reach its maximum in the near future. This means that there will be more and more people creating content like photos, videos, and willing to share that information using their smartphones. When these wireless devices contact each other, it gives birth to opportunistic networks which enable epidemic spreading of content. In opportunistic networks, users cooperate to distribute content of interest in a peer-to-peer fashion over wireless opportunistic contacts (e.g. one-hop ad hoc WiFi). Nevertheless, the content that is widely distributed in such networks can be illegitimate content -such as spam or virus sent by malicious users. One way to handle this issue is to build a reputation system. A reputation system [1, 2] assigns reputation scores for a set of objects such as services, goods or entities within a particular community. The scores are computed based on the opinions that other entities give about the named objects. Typically entities belonging to a community use reputation scores to make decisions, for instance to decide whether to buy a specific service or not. Naturally, the way it works is that an object associated with a high score is better considered than an object that has lesser reputation score. More precisely, it is the collection of ratings accumulated by the system that will determine, for each object, if its behaviour should be praised (high score) or sanctioned (low score).
So the idea is that a user who behaves well would be given high ratings, whereas someone who tries to send illegitimate content would be badly rated. But a malicious user could circumvent his bad ratings by creating a new identity everytime he would get a bad score. This is the so-called Sybil attack, where an attacker gains a disproportionately large influence over the system by creating a large number of identities; to prevent such malicious users from running away from their bad reputation using Sybils, we need to use an identity management that will hold users accountable for their action. More precisely, we bind a public key with a phone number in a certificate that will be delivered by a trusted certification authority (CA). A certificate[3] is a data structure that binds a public key to an identity -here the phone number. In this scheme, only a user owning a valid certificate would be considered as legitimate, and the certificate should be included in all content that is distributed.
In this project, we will exploit the fact that cell phones have SMS capabilities that will allow to contact the authority by SMS. We choose SMS messaging rather than any other mobile telecommunication service because the infrastructure necessary to send or receive an sms is available in all countries except for the ones where mobile phones

[1] http://www.marketforce.com/2011/02/consumers-now-more-likely-to-buy-androids-than-iphones/

are banned. Hence the protocol we design, in order to make applications running on opportunistic networks usable, is based on SMS.
As a starting point, this chapter describes the objective of the project, depicts the implementation procedure and outlines the whole report.

1.1 Objective

The purpose of this project is to enable mobile devices to register and obtain certificates from a central infrastructure. To be more specific, the goal is to perform the following tasks:

- design an SMS based Certificate Management Protocol
- implement the corresponding prototype
- evaluate the efficiency and level of security of the overall system

The system should also include a recover mechanism in case of compromised certificates and should be able to handle possible attacks. We furthermore take into account that SMS are costly and not guaranteed to arrive.

1.2 Implementation tool

The project has been built and tested for the Android platform. The user interface is designed in a way that it displays the main registration events - along with their respective relevant details - as well as the continually updated lists of certified identities distributed by the central infrastructure. It also allows users to request new certificates or revoke compromised ones by clicking on the corresponding buttons. The system will save the records into the Android database.

1.3 Outline

In this report, we will review the key steps in developing the whole project. We firstly introduce the concepts behind Certification Authorities, the SMS protocol and PodNet requirements in Chapter 2. Subsequently, in Chapter 3, we present the design part by describing the protocols for both acquiring and revoking certificates, and we take a look at the basic procedures in the implementation of the software in Chapter 4. Then we will evaluate the results, presenting the communication overhead and security aspects in Chapter 5. As a conclusion, we will discuss the methodologies for improving the software performance in Chapter 6.

Chapter 2

Related Work

In this chapter, we will depict the main concepts that lie behind the project. Beginning with the way certification authorities operate, we will then investigate the SMS protocol. We will furthermore argue that our certification system can be useful to the PodNet project.

2.1 Certification authorities

As its name indicates, a Certification Authority (CA) is responsible for issuing digital certificates. The latter serve to bind the ownership of a public key to the owner of the certificate, which allows other entities to verify the validity of signatures made by the private key. In communication models using certificates to establish trustworthy interactions like public key infrastructures, we often require a CA to act as a third party that will be trusted by both the owner of the certificate and the parties relying upon the certificate.
Certificates[3] are data structures that bind public key values to subjects, hence they contain a public key and the identity of the owner. The corresponding private key is kept secret by the user who generated the key pair and the CA issuing the certificate is responsible for verifying the credentials of the applicants before digitally signing the certificates. The CA may verify this for instance by challenging the applicant for the private key.
The attributes contained in the certificate are chosen depending on the goal of the infrastructure that is involved in the system. A certificate that is used on the internet usually consists of a sequence of three parts:

- CertificateDetails: It contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information.
- SignatureAlgorithm: The signatureAlgorithm part contains the identifier for the cryptographic algorithm - such as RSA or EC - used by the CA to sign the certificate.
- SignatureValue: signatureValue contains a digital signature that the CA uses to certify the validity of the information in the CertificateDetails part. Notably, the CA certifies the binding between the public key and the subject of the certificate.

Note that a certificate has a limited validity period which should be present in its contents.

2.2 SMS Protocol

The Short Message Service or so-called SMS[4] is a globally accepted wireless service dedicated to the transmission of alphanumeric messages. SMS provides a mechanism for transmitting short messages to and from wireless devices, using an SMS Center (SMSC), which functions in a store-and-forward way. The wireless network provides the mechanisms required to find the destination station and transports short messages between the SMSCs and wireless stations. For most providers, the service elements are designed to provide information about the delivery of text messages - in particular non-delivery reports when something goes wrong during the delivery process. One of the convincing features of the SMS service is that an active mobile device is able to receive or send a short message at any time, independent of whether a voice or data call is in progress. If a receiving station is not available, the short message will be stored in the SMSC until the device becomes available again.

2.3 PodNet

The PodNet [1] project is destined to ad hoc podcasting; in other words, it achieves mobile distribution of user-generated content. PodNet contrasts with the sharing from traditional Internet based platforms in the sense that it carrys out epidemic spreading of the content. The system assumes that a group of peer-to-peer users interact with each other and distribute content over opportunistic links created between wireless devices carried by the users. This specific type of information transfer can actually happen between people having common interests during any social gathering event. But PodNet could be subject to malicious behaviour and used to send spam for instance. Thus, for such a content distribution model to work properly, it requires that the users can be trusted or accurately recognized within the opportunistic networks. This can be achieved using an identity management that would rely on a certification authority (CA). Since PodNet is supposed to be used on handheld phones which are equipped with the SMS protocol, it further motivates us to run the CA over SMS.

[1] http://www.podnet.ee.ethz.ch/

Chapter 3

Design

This chapter will first outline the project's general assumptions, then it will present the protocol for the acquisition of a certificate along with the possible scenarios.

3.1 Overview and assumptions

The project relies on mobile users willing to get certificates. The users are implemented on the client side of the infrastructure, whereas the certification authority lies on the server side. In order to distribute the certificates in an efficient way and to protect the system from possible attacks, the server keeps track of every new client it receives a request from, and it saves all relevant information - phone number, public key, issue and validity dates - in its database. Since SMS are not free, we want to minimize charge, for both client and server, hence minimize the number of SMSs to be sent.

3.2 Certificate Acquisition Protocol

In this setting, each new client who wants to register to the authority is supposed to generate a cryptographic key pair and request a new certificate by sending an SMS to the server. The request SMS must only contain the public key of the client. More specifically, the SMS should be composed of the public key. What follows the client's request is a challenge-response protocol to check the validity of the phone number as well as to challenge the ownership of the private key. The server sends a challenge, encrypted with the user's public key, and upon reception of that SMS, the client will proceed to the decryption using its private key. Then it will send the decrypted challenge to the server. Upon verification of the challenge, the server will let the client know about the time it will have to wait before receiving the certificate. Finally the server will proceed to send a message [1] consisting of the signed certificate, i.e. the signed sequence of the timestamp, followed by a space, followed by the end of validity date, followed by another space and then the public key. Figure 3.1 shows the protocol steps involved in the certificate request. We choose not to include the phone number in the certificate for privacy reasons.
In case the private key is lost or stolen from a client, the client will need to revoke his current certificate and replace his key pair by a new one.
To discourage the client from creating too many new identities - i.e. certificates -, we examine two alternative approaches. We can make the client pay for each revocation

[1] Note that all messages end with a hash symbol to indicate the end of their content.

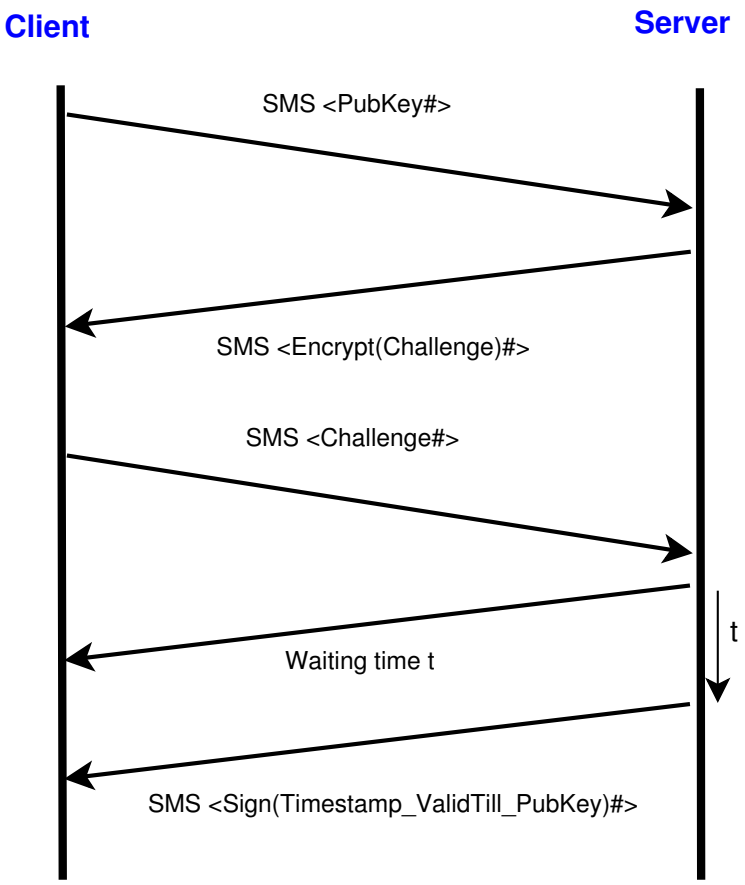

Figure 3.1: Certificate Acquisition Protocol

he needs. This will prevent the client from doing infinite revocations - especially in case of malicious client - but might not be that effective if the amount to pay is too low. On the other hand, if the amount is too high, we might be unfair towards genuine users that simply lost their keys. We also would like to avoid disparities among rich and poor users.
In this protocol, we decided to make the client wait exponentially more time for each new request of certificate happening before the expiration date. Basically, losing your key one or two times is forgiven, but then, because of the exponential growth, each new loss becomes more and more tedious. This seems to be a good trade-off between locking a malicious client and still give a chance to a honest user. One possibility we might end up considering is to use a hybrid approach, where it would be possible for a client to cancel his waiting time by paying a certain amount of money. This would come in handy in case of an emergency need for the new certificate.
Since the binding between the public keys and phone numbers is only saved on the server side, it is worth well protecting the server. If the server was compromised for instance, all the relevant information would be lost without any mean to recover it.

Chapter 4

Implementation

Now we detail the implementation of the system. We first present an overview of the system and technologies involved, then we look more precisely at the actual implementation, going from a general overview of the main components to a precise description of both the client and the server. We conclude with a discussion about the choices made in the implementation.

4.1 Overview

We decided to implement a prototype to serve as a proof of concept of the protocol. The phone clients are based on the open source Android framework. The server is also running on an Android device.
Android provides an API to access phone's capabilities, like SMS messaging. Both the client and the server use this API to send and receive SMS from each other. Since SMSs are text based and we wish to send binary data (cryptography public key), we need to encode sequences of bytes into a string. We do this using the Base64 encoding algorithm.
The protocol requires each client to generate a key pair and then register itself to the server by sending its public key. Java provides support for cryptography, and in particular for RSA. We use Java's API to generate a key pair for both the client and the server, and also to sign the certificate for the server.

4.2 Architecture

The implementation is organized in a modular system of classes. Since both the client and the server have to do very similar tasks, it is important to reuse as much code as possible.
In particular, we are able to factorize the cryptography algorithms and the database access classes, so that the client and server can use them in a similar fashion. The encoding and decoding of SMSs are also shared among the client and the server.
Here is a quick overview of the main modules of the implementation.

`Crypto` is a class that provides methods for RSA cryptography. It provides several useful methods used throughout the rest of the system. One of its most important features is the generation of random key pairs. We chose to use keys of fixed length 1024 bits.

Of course, it also provides encryption and decryption methods that take as parameters respectively the public and private key. Although it is the exact

same algorithms, but with reversed public and private key, it also offers signing and signature-verification methods for convenience.

The other capabilities of the `Crypto` class is to provide encoding of public and private key to a sequence of bytes and to an ASCII string. Encoding a key to a string is an essential step to be able to send through an SMS. Additionnaly, the methods to read the key from a sequence of bytes or an ASCII string are also available. We rely on an external library to encode stream of bytes into an ASCII string.

`KeyPairDB` and `ServerDB` are a database abstraction layer that gives access to persistent storage on the Android devices. It builds on top of the Android library that gives full support for SQLite.

The `KeyPairDB` class offers method to store and load key pairs into a SQLite database. This is a convenient object to use for both the client and the server, that need to store their public and private keys across several executions.

The `ServerDB` is, on the other hand, only used by the server. It has the role to store the important information about each client. It records the public key of the client, his phone number, a timestamp indicating when the certificate has been released as well as another timestamp indicating until when the certificate is valid

`ClientActivity` and `clientservice` module contains the main code for the execution of a client. It relies on other modules, like `Crypto` and `KeyPairDB`.

`ServerActivity` and `Serverservice` module contains the main code for the execution of a server. It relies on most of the other modules, including `Crypto`, `KeyPairDB` and `ServerDB`.

4.3 Client

The most important part of the client code is contained in the `ClientService` class. It implements the protocol presented in the previous chapter. It inherits from the `Service` class and run in the background once launched by the entry point of the client. We provided `Activity` as a way to start the client: the `ClientActivity`.
One important role of the client is to gain access to the phone SMS capabilities. In order to receive SMSs, the client uses a `BroadcastReceiver` that is set to intercept the reception of SMS via the utilization of an `IntentFilter`. We programmed it such that the client has a higher priority over received SMS than the default SMS application, which allows to filter SMSs before they can reach the user.
To be able to send SMSs, we make use of the `SMSManager` class services. Any application can require the activation of another `Activity` or `Service` present on the device via the use of message that takes the form of `Intent`. This is how we get access to the `SMSManager` from the `Client`.
The client application exposes a minimal graphical user interface, defined in `ClientActivity`, in order to test the protocol, and also to display status information. We offer a button to register to the central authority The application also display some information about the current status of the protocol.
The current key pair used by the client is stored in a local database using the `KeyPairDB` module.
Once all of the previous libraries and modules are working and available, implementing the protocol becomes relatively straightforward. Basically we gather the required informations for one specific message, form the message and then encode it and send

it via the SMS managing functions. The `BroadcastReceiver` then notices us when we receive a response and we can parse it and interpret it accordingly.

4.4 Server

The basic building blocks of the server are very similar to the ones of the client. The server also has its own class containing the implementation of the protocol from the server side. The `ServerService` class also inherits from the `Service` class.
Similarly to the client, the server needs an access to the phone's SMS capabilities. For this purpose, we use the same kind of tools as we used in the case of the client.
One difference with the client's management of SMSs is that when the server sends the certificate to the client, it needs to sign it with its private key. This will transform the message into a sequence of bytes, that then needs to be encoded in a similar fashion as was the public key sent by the client. For this, `Server` relies on the `Base64` module to do the encoding to a valid ASCII string that can then be sent with the SMS protocol.
The first time the server is ran, it will generate a new pair of keys and store it into the `KeyPairDB` database. Then at restart, it can load its key pair into the main memory and use it for the current session.
The server uses a second database to store information about the clients. Access to this database is ensured via the `ServerDB` class that provides methods to store, update and get client's essential data. The information stored includes the public key, the phone number and the dates of issue and expiration of the certificate. These details are essential since they represent the unique valid mapping between phone numbers and public keys.
When a client requires to change its certificate before the official expiration date, we impose a waiting time before issuing the new certificate. We use the following formula (in seconds):

$$2 \cdot 2^n$$

where n represents the number of times the client has requested a new public key. This is mostly for demonstration purposes, in order to have a relatively small waiting time but still long enough to notice it. Optimal waiting time formula in a deployed server is still an open question.
To maintain a certain fairness, we proceed to a counter reset - clearing all records - when a certificate request arises after the certificate has naturally expired.

The interface of the server once it has started is relatively simple and is implemented in both `ServerActivity` and `ServerDetailActivity`. The former activity lists the phone numbers for which the server has issued certificates. The latter furnishes the detailed certificates for a selected phone number.

4.5 Implementation concerns

In this section, we cover a few issues encountered across the implementation. These are some of the reasons we ended up with the previous design. We discuss the technology used for the server side with respect to the other possibilities, we compare different compression schemes for the SMS messaging and we mention the cryptography support from Java.

4.5.1 Server Technology

The server has to be able to receive SMSs. So we initially decided to use gnokii [1], which is a set of tools and drivers for Nokia mobile phones on Linux. In theory, it could be used to connect a mobile phone to a computer in which a server would be running. Then the server could send and receive SMSs from the connected mobile phone.
However, even though the sending of SMSs through the phone was successfully implemented, we were not able to let the server access the received SMSs of the device using gnokii. It seems that none of the various phone models we tried are supported by the current version of gnokii.
In order to build our prototype, we then opted for the alternative of using an Android mobile phone as the server. In this setting, access to the SMS system is basically given for free.
Another option would have been to use a kind of proxy Android phone together with a server on a Linux machine. The phone would forward SMSs to the server, and the server would send request to the phone to send an SMS.

4.5.2 SMS Encoding

SMS messages are text based. Thus it is challenging to send binary data, like cryptographic keys over SMS. Additionally, an SMS can contain a maximum of 140 characters. We thus need to encode binary data to strings of characters.
We first consider a simple and straightforward approach: URL-encoding. URL encoding keeps the basic alphabet `[a-zA-Z0-9]` untouched and transform every other bytes to a string consisting of a percentage followed by its hexadecimal code.
The main drawback of this technique is that it will generate a very long string (up to 3 times the original size) for some sequence of bytes. This has to do with the fact that URL encoding was designed to encode some characters from a URL, and URL were mostly using standard alphanumeric values. Unfortunately, the sequence of bytes representing a key is uniformly distributed, unlike URLs.
We opted for the base64 encoding. Basically, the encoding picks 64 characters that are used as the base alphabet, and encoded strings are over this alphabet. The standard way to pick this alphabet is to choose both the lowercase and uppercase 26 latin alphabet characters, the ten digits and two additional characters, like '/' and '-' for example.
We could also have used hex encoding - which expands one byte to two - or base85 - which expands four bytes to five. Base64 expands three bytes to four, so it is less verbose than hex encoding, but a bit more than base 85. However, we chose base64 since it seemed to be the most widely used hence the best supported.

4.5.3 Handling Out of Order SMSs

SMSs are not guarantee to arrive in the exact order in which they were sent. In that way, they are similar to IP packets that can both get lost or arrive out of order.
In fact it seems that those problems are very likely to happen whenever we send several SMSs in a very short time. Unfortunately we are doing exactly that each time we need to cut a long message into several smaller SMSs.
In order to handle this case, we add a sequence number to each SMS we send. This sequence number can be used to reconstruct the correct message. The sequence numbers also help to implement a resend of the lost SMS: the peer that did not received the SMS simply asks for a resend of a message with the corresponding sequence.

[1] http://www.gnokii.org/

4.5.4 Cryptographic Aspects

In order to sign and decrypt certificates, we need a public-key cryptography algorithm. The two most famous algorithm for this task are Elliptic Curve Cryptography(ECC)[5] and RSA[6]. In theory, EC is more efficient[7] than RSA, because it needs smaller keys. For instance, ECC-160 has a six times smaller key-size than RSA-1024 and it can generate a signature 12 times faster.
Since we want to minimize the number of SMSs, and since keys must be sent through SMS, this is an important feature.
Unfortunately the standard Java distribution does not seem to support ECC cryptography. The Java API architecture for cryptography follows a very dynamic design pattern, using Factories classes to build actual instances. Even though most of the interfaces for most of the algorithm are present in the API, you need to request the actual algorithm you wish to use at runtime through a string request. The supported algorithms depend on the provider you use, which is also dynamically specified. Unfortunately, none of the standard providers on Java or Android were supporting ECC cryptography.
We finally decided to use RSA, since it does not induce any fundamental change to the system.

Chapter 5

Evaluation

In this chapter dedicated to the evaluation of the system, we will elucidate the average cost of the SMS based infrastructure and try to assert how much security can be achieved within our prototype.

5.1 Communication overhead

Considering that we want to minimize the number of SMSs to be sent from both sides of the infrastructure, it seems worth evaluating the cost coming from the SMS charge. We chose to use keys of length 1024 bits. Due to the fixed length of the cryptographic keys, the number of SMSs sent by both client and server is fixed for each kind of request too, and it never exceeds 4 SMSs - 2 SMSs for a client's message and 4 SMSs for the server's corresponding answer -, which is a quite small number hence inducing a small cost.
In order to request a certificate, only the encoded public key should be sent, which takes 2 SMSs. But for the server's answer, the encoded public key needs to be appended to other (text) data, then it should be signed by the CA, and finally the whole sequence of bytes should be encoded into text characters again. The signing and encoding procedures both add to the length of the data, resulting in a total of 4 SMSs. Note that the challenge-response handshake needs 3 SMSs, the first message takes up 2 SMSs because it contains the server's challenge encrypted with the client's public key, whereas the client's response only needs 1 SMS - the decrypted challenge being sent in clear. One additional SMS needs to be sent to let the client know the delay in receiving a new certificate -in case of a revocation. This leads to 10 SMSs to send in total for the whole protocol.
In a service model where the server provides the service, the client should bear or contribute to the cost of the server, here induced by the sent SMSs.

5.2 Security aspects

Now let us present the main attacks that our system can actually avoid :
The first one that springs to mind when we think about identity management in a distributed scenario, is the Sybil attack. In this attack scenario, an attacker creates a big number of identities in order to gain a disproportionately large influence in a peer-to-peer network. Since our prototype renders a phone number equivalent to an identity, and considering one can get as many phone numbers as one can afford, we can assume that only a very wealthy malicious user could create a big number of identities; but we expect that the cost will be considerable enough to limit the number of

identities being forged in a potential Sybil attack.

Although our system helps preventing the harm of malicious behaviour, it can nevertheless not compensate for the lack of security originating from the SMS protocol itself. Indeed, SMS messaging comprises security vulnerabilities due to some of its features, and is therefore exposed to various attacks.
One of them is the so-called SMS Spoofing. It occurs when an attacker tries to impersonate a user by submitting messages with his fake number. Fortunately, this attack is harmless since the certificate will be sent to the real device owning the forged number. The attacker could send a request for a certificate message with a forged number, but the server will in turn send the signed certificate to the legitimate owner of the public key binded to the certificate, namely the victim of the spoofing who would consequently stand uninjured. However, in case a malicious user were to spoof a number and send a revocation request, the client would receive a new certificate even though his current certificate was still valid, and would not even be able to use it since the corresponding key pair was generated by the malicious user. The victim would then have to reissue a certificate, which means he would be unfairly penalized for issuing two certificates before their expiration dates are reached. This can be avoided thanks to our four-way protocol. Indeed, the challenge-response protocol following any request provides the CA a way to authenticate the phone number - by challenging the claiming owner of the number with an SMS - and it also allows to confirm the client's request - if the user answers the challenge as expected.

Chapter 6

Conclusion and Future work

This project enables mobile devices to register and obtain certificates from an implemented central infrastructure by using a simple communication protocol over SMSs. The implemented system uses base64 encoding with fixed size keys to limit the maximum numbers of SMSs required for certificate operations to 10. It helps in reducing the communication overhead and operating cost for the system. The prototype can handle the main potential security attacks: namely the Sybil attack and the SMS spoofing. Addendum, it is also capable of handling of sporadic SMS losses by using a retransmission mechanism.
In this section we will highlight some potential directions where the application can be improved.

The currently implemented prototype is not complete. It does not support any revocation scheme due to the lack of potential applications using our infrastructure. However a simple revocation scheme can be implemented by keeping track of the latest valid certificate and applications that use it. In case of certificate compromise or loss, a user has to actively notify or update his certificate for all the listed applications. Although simple and effective in a closed controlled environment, this scheme will break if a new unlisted application uses the old certificate to perform malicious operations on behalf of the user. A node can let others know that its certificate is no longer valid either by flooding its new certificate - which would be very costly -, or by having each application check the validity of the certificates with the server. The distributed nature of the system makes the revocation of certificates even more challenging. Therefore, depending upon the security demands of the applications using our system, it will require a more comprehensive and thorough certificate revocation scheme.

We are also looking into reducing the number of SMSs required by using lossless data compression schemes (such as LZW compression or Huffman coding etc.) on the text sequence, before the CA signs it. It can potentially reduce the number of SMSs to be sent but nevertheless requires more computation power -due to the additional compression and decompression routines. Although a client low on battery should be able to switch off compression to save power. This is a classical trade-off between network usage and CPU time. Nevertheless, compression can not reduce space considerably when applied on random sequences such as cryptographic keys.
A better way to minimize the amount of data sent is to change the cryptographic scheme. For instance, ECC would seem like a better option than RSA since it has smaller keys and therefore allows to send the keys over less SMSs.

The design and principles of our system are equally applicable to other messaging technologies as well. A future revision of the system can also use high-bandwidth 3G technologies to communicate to the server using MMS or direct data connection, rather than SMS messaging.

Bibliography

[1] J. Liu and V. Issarny. Enhanced reputation mechanism for mobile ad hoc networks. *Trust Management*, pages 48–62, 2004.

[2] K. Walsh and E.G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *Proceedings of the 3rd conference on Networked Systems Design & Implementation-Volume 3*, pages 1–1. USENIX Association, 2006.

[3] D. Solo, R. Housley, W. Ford, and T. Polk. Internet x. 509 public key infrastructure certificate and crl profile. Technical report, IETF RFC 2459, 1999.

[4] 3GPP Specification detail at `http://www.3gpp.org/ftp/Specs/html-info/23040.htm`.

[5] Certicom Research. Sec1: Elliptic curve cryptography. 2000.

[6] A. Shamir R. L. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[7] N. Gura, A. Patel, A. Wander, H. Eberle, and S.C. Shantz. Comparing elliptic curve cryptography and rsa on 8-bit cpus. *Cryptographic Hardware and Embedded Systems-CHES 2004*, pages 925–943, 2004.