ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Short, Undeniable Signatures for Android 2.x

Sebastien Duc

School of Computer and Communication Sciences

Master Semester Project

June 2012

Responsible	**Supervisor**
Prof. Serge Vaudenay	Dr. Ioana Boureanu
EPFL / LASEC	EPFL / LASEC

LASEC

Contents

Chapter 1

Introduction

Undeniable signature schemes were first introduced by Chaum and van Antwerpen [4]. Unlike classical signatures, undeniable signatures need the verifier to run an interactive protocol with the signer to verify a signature. MOVA is an undeniable signature scheme advanced by J. Monnerat and S. Vaudenay [5–9]. Classical signatures are already widely used and the applications are numerous. This is not necessarily the case for undeniable signatures, this is one of the reason why we tried to exhibit a realistic application for MOVA.

Nowadays, most of the people have a smartphone. This progress gave a lot of opportunities in the development of new mobile applications. Security in the mobile environment can be very challenging and MOVA can bring something. Indeed, MOVA has the property to provide very short signature. In this project, we studied MOVA and tried to find a relevant application on Android [1] that uses MOVA. The final application we came to is a university contest on Android phones. In this application, universities are challenging each other. Each university provides challenges or quizzes to the other. Students form teams to answer to the quizzes and send them back to the university who designed it. The aim of each university is to cumulate the maximum score, where a certain amount of score is won depending on how successfully the team answered to the challenge. In this application, a client-server architecture is used. MOVA is used to sign the challenges that are sent over the network.

In Chapter 2, we will first cover some basic notions of cryptography. We will recall some cryptographic primitives that will be used in the application. In Chapter 3, we will first describe undeniable signatures. Then we will present the MOVA signature scheme and we will show how it works in theory. In Chapter 4, we will describe the application we developed. In Section 4.1 a general view on it will be given and in Section 4.2 a general view on the security. In Section 4.3 the threat model will be described, followed by Section 4.4 where we describe some problems we encountered and finally in Section 4.5 we will justify why to use MOVA instead of classical signature. In Chapter 5, we will describe how the application was designed. In Section 5.1, we will present the architecture of the application. Then in Section 5.2 we will discuss some implementation choices.

Chapter 2

Preliminaries

In this chapter, we explain the notions essential for the remainder of this report. Several basic cryptographic notions are presented.

2.1 Hash Function

A hash function is a function that maps an arbitrary length input to a fixed length output. More formally, a hash function is a function

$$h\colon \{0,1\}^* \ \rightarrow \ \{0,1\}^n, \text{for } n \in \mathbb{N} \text{ fixed.}$$

A cryptographic hash function h has to satisfy some security properties. Especially it has to resist to the following attacks:

Collision attack Find x and x' in $\{0,1\}^*$ such that $h(x) = h(x')$ and $x \neq x'$.

First preimage attack Given the hash $y \in \{0,1\}^n$, find $x \in \{0,1\}^*$ such that $h(x) = y$.

Second preimage attack Given to input x and x' in $\{0,1\}^*$ such that $x \neq x'$ and $h(x) = h(x')$.

2.2 Pseudorandom Generator

A pseudorandom generator is a function taking a seed as input and outputs a sequence of bits. More formally, it is a function $F : \{0,1\}^k \rightarrow \{0,1\}^n$, for $k, n \in \mathbb{N}$ where $k < n$ usually. To be secure, a pseudorandom generator must be unpredictable which informally means that given a partial sequence of bits, it is hard to guess the next bits.

2.3 Commitment Scheme

Commitment schemes are cryptographic primitives used when a party in a protocol wants to hide a value temporarily but accepts to be bound to this value from the beginning to the end of the protocol. A sender chooses a value, sends another value to the receiver related to his chosen value without revealing his initial value, i.e. the sender commits to his value. Then both the sender and receiver exchange some messages and at some point, the sender reveals some material to the received so that the receiver knows the value and can check that it is correct.

$$\begin{tikzcd}
\text{Sender} & & & \text{Receiver} \\
X & & & \\
(c,k) = \mathrm{Commit}(X;r) & {} \arrow[r, "c"] & {} & c \\
& {} \arrow[r] & {} & \\
& \vdots & & \\
& {} & {} \arrow[l] & \\
& {} \arrow[r, "k"] & {} & X = \mathrm{open}(c,k)
\end{tikzcd}$$

Figure 2.1: Commitment Scheme

The properties we want for a commitment scheme is to be hiding (when given $\mathrm{Commit}(X;r)$, it should be impossible to recover X) and binding (given that $c = \mathrm{Commit}(X;r)$ it is should be impossible X' such that $\mathrm{Commit}(X';r') = c$).

Example 1. *We can build a commitment scheme using hash functions h. Let $\mathrm{Commit}(X;r) = h(X||r)$ where $\cdot||\cdot$ denotes the concatenation. To open the commitment, the sender just sends X and r. The commitment scheme is hiding and binding when the hash function is collision resistant and resistant to first and second preimage attacks.*

2.4 Digital Signature

A digital signature scheme is a public-key cryptosystem that is aimed to sign messages so that the identity of the sender of the message is binded with the message.

Given a security parameter n, a signature scheme is defined by three algorithms:

KeyGen An algorithm that generates the public and the secret key. Let $(k_p, k_s) \leftarrow \text{KeyGen}(1^n)$, where k_p is the public and k_s is the secret key.

Sign An algorithm to sign a message. Let $\sigma \leftarrow \text{Sign}(m, k_s)$, where m is a message from the message space.

Verify An algorithm to verify a signature. $\mathrm{Verify(m, \sigma, k_p)}$ outputs a bit that determines whether (m, σ) is either valid (when the output is 1) or invalid (when the output is 0).

One security property one would want to have is existential unforgeability under chosen message attack. This means that an adversary without the secret key should not be able to forge a valid signature for any message, given that he can query some messages to an oracle some messages and get their valid, respective signature[1].

More precisely, $\forall$ probabilistic polynomial-time algorithm A that has access to the signing oracle and that outputs a valid message signature pair (m, σ') that was not queried to the oracle,

$$\Pr\left\{1 \leftarrow \text{Verify}(m, \sigma', k_p) | (m, \sigma') \leftarrow A\right\} \in O(n^{-k}) \quad \forall k \in \mathbb{N},$$

where the probability is taken in the random choices of A.

In classical signature scheme that were just described, anyone can verify a message-signature pair without communicating with the signer. The verifier only needs to know the public key.

[1]the adversary cannot query the message that he wants to forge the signature of to the oracle

Chapter 3

MOVA Signature Scheme

In this chapter, we will first present undeniable schemes. Then MOVA signature scheme will be described herein.

3.1 Undeniable Signatures

Undeniable signature scheme were invented by Chaum and van Antwerpen [4]. The aim of an undeniable signatures scheme is roughly the same that a classical signature scheme (see section 2.4) which is to bind an identity to a message. The only difference is that the verifier must run an interactive protocol with the signer to verify the validity of the message-signature pair.

As in classical signatures, let n be a security parameter. Let S and V be the signer and the verifier respectively. To define an undeniable scheme, the following algorithms must be defined:

Setup It consists of two algorithms that generate the key pair for S and V. Let $(k_p^S, k_s^S) \leftarrow \text{Setup}^S(1^n)$ and $(k_p^V, k_s^V) \leftarrow \text{Setup}^V(1^n)$.

Sign An algorithm to sign a message in the message space using the secret key of S. Let $\sigma \leftarrow \text{Sign}(m, k_s^S)$,where m is the message.

Confirm It consists of an interactive protocol between S and V to confirm the validity of a message signature pair.

Deny It consists of an interactive protocol between S and V to deny the validity of a message signature pair.

3.2 MOVA

The MOVA[1] signature scheme was invented by J. Monnerat and S. Vaudenay [5–9]. It is an undeniable signature scheme.

[1]MOVA stands for **Mo**nnerat **Va**udenay

3.2.1 Definition

Consider a signer S and a verifier V. Assume that they can use two pseudorandom generators, GenK and GenS.

The **Setup** algorithm is defined as follows. S chooses two Abelian groups Xgroup and Ygroup. Then he chooses **hom** : Xgroup $\rightarrow$ Ygroup. Then using GenK on seed seedK he generates Lkey elements of Xgroup:

$$\mathrm{Xkey}_1, ..., \mathrm{Xkey}_{\mathrm{Lkey}} \leftarrow \mathrm{GenK}(\mathrm{seedK}).$$

Finally, he computes $\mathrm{Ykey}_1, ..., \mathrm{Ykey}_{\mathrm{Lkey}}$, where $\mathrm{Ykey}_i = \mathbf{hom}(\mathrm{Xkey}_i)$.

With this Setup algorithm, the public key

$$K_p^S = (\mathrm{Xgroup}, \mathrm{Ygroup}, d, \mathrm{seedK}, (\mathrm{Ykey}_1, ..., \mathrm{Ykey}_{\mathrm{Lkey}})),$$

where $d = |\mathrm{Ygroup}|$ and the secret key

$$K_s^S = \mathbf{hom}.$$

The **Sign** algorithm is defined as follows. Let $m \in \{0,1\}^*$ be the message to sign. First GenS is seeded with m to obtain $\mathrm{Xsig}_1, ..., \mathrm{Xsig}_{\mathrm{Lsig}} \in \mathrm{Xgroup}$. The signature is

$$\sigma = (\mathrm{Ysig}_1, ..., \mathrm{Ysig}_{\mathrm{Lsig}}),$$

where $\mathrm{Ysig}_i = \mathbf{hom}(\mathrm{Xsig}_i)$.

The **Confirm** interactive protocol is defined as follows. It takes (m, σ) as input. First S and V retrieve values $\mathrm{Xkey}_i, \mathrm{Xsig}_j$ for $i = 1, ..., \mathrm{Lkey}$ and $j = 1, ..., \mathrm{Lsig}$ by using GenK and GenS. Then they run $\mathrm{GHIproof}_{\mathrm{Icon}}(S)$ with

$$S = \{(\mathrm{Xkey}_i, \mathrm{Ykey}_i) | i = 1, ..., \mathrm{Lkey}\} \cup \{(\mathrm{Xsig}_i, \mathrm{Ysig}_i) | i = 1, ..., \mathrm{Lsig}\}$$

and S is playing the role of the prover. $\mathrm{GHIproof}_l(S)$ is described in figure 3.1.

The **Deny** interactive protocol is defined as follows. It takes a supposedly invalid message-signature pair (m, σ') as input with $\sigma' = (\mathrm{Zsig}_1, ..., \mathrm{Zsig}_{\mathrm{Lsig}})$ First, S and V retrieve values $\mathrm{Xkey}_i, \mathrm{Xsig}_j$ for $i = 1, ..., \mathrm{Lkey}$ and $j = 1, ..., \mathrm{Lsig}$ by using GenK and GenS. Then they run $\mathrm{coGHIproof}_{\mathrm{Iden}}(S, T)$ with

$$S = \{(\mathrm{Xkey}_i, \mathrm{Ykey}_i) | i = 1, ..., \mathrm{Lkey}\}$$

and

$$T = \{(\mathrm{Xsig}_i, \mathrm{Zsig}_i) | i = 1, ..., \mathrm{Lsig}\}.$$

$\mathrm{coGHIproof}_{\mathrm{Iden}}(S, T)$ is described in figure 3.2.

3.2.2 Interactive Proofs

In this section, we will describe the two interactive proofs used in MOVA (namely GHIproof and coGHIproof). In MOVA the secret key is a group homomorphism. So let us start with a small definition.

Definition 1. *Let G and H be two Abelian groups. Let us consider the set of points $S = \{(x_i, y_i)|i = 1, ..., s\} \subseteq G \times H$. We say that S interpolates in a group homomorphism if there exists* **hom** *: $G \to H$ st.* $\mathbf{hom}(x_i) = y_i$, *for $i = 1, ..., s$. Furthermore if we consider $T \subseteq G \times H$, we say that T interpolates in a group homomorphism with the set S if $S \cup T$ interpolates in a group homomorphism.*

We can now define two problems taken from [7,8] related to MOVA and the previous definition. We have the Group Interpolation Problem:

***n*-*S*-GHI Problem**
Parameters: Two abelian groups G, H, and $S \subseteq G \times H$, $n \in \mathbb{N}$.
Instance Generation: n elements $x_1, ..., x_n \in_U G$.
Problem: Find $y_1, ..., y_n \in H$ such that $\{(x_i, y_i)|i = 1, ..., n\}$ interpolates with S in a group homomorphism.

And we have the Group Interpolation Decisional Problem:

***n*-*S*-GHID Problem**
Parameters: Two abelian groups G, H and $S \subseteq G \times H$, $n \in \mathbb{N}$.
Instance Generation: The instance T is generated according to T_0 or T_1. T_0 is generated by picking $\{(x_i, y_n)|i = 1, ..., n\} \in (G \times H)^n$ uniformly at random such that it interpolates with S in a group homomorphism. T_1 is picked uniformly at random in $(G \times H)^n$.
Problem: Decide whether the instance T is of type T_0 or T_1.

For both problems, in practice and in particular in MOVA, we consider sets S that interpolates in a unique group homomorphism.

Now we can describe the interactive protocols that are used in MOVA.

GHID Problem interactive proof

We first describe the interactive proof for the GHID Problem. Consider G, H and S as defined above where $|H| = d$. Consider a security parameter l. In this protocol, a prover convinces a verifier that S interpolates in a group homomorphism **hom**. **hom** the secret witness known by the prover. The protocol is described in Figure 3.1.

GHIproof$_l(S)$

Parameters: $G, H, d = |H|$

Input: $l, S = \{(g_i, e_i) | i = 1, ..., s\} \subseteq G \times H$

Prover P		Verifier V
		$r_i \in_U G, a_{i,j} \in_U \mathbb{Z}/d\mathbb{Z}$ compute $u_i = dr_i + \sum_{j=1}^{s} a_{i,j} g_j$
	$\xleftarrow{u_i}$	$w_i = \sum_{j=1}^{s} a_{i,j} e_j$ $i = 1, ..., l$ and $j = 1, ..., s$
$v_i = \mathbf{hom}(u_i) \quad i = 1, ..., l$	$\xrightarrow{\text{Commit}(v_1,...,v_l)}$	
	$\xleftarrow{r_i, a_{i,j}}$	
check $u_i = dr_i + \sum_{j=1}^{s} a_{i,j} g_j$	$\xrightarrow{\text{open}(v_1,....,v_l)}$	
		check $v_i = w_i$

Figure 3.1: GHIproof$_l(S)$

coGHID Problem interactive proof

Let us describe the interactive proof for the coGHID Problem. Consider G, H,S and T as defined above where $|H| = d$ and p is the smallest prime factor of d. Consider also a security parameter l. In this protocol, the prover convinces the verifier that for at least one $(x_i, z_i) \in T$ we have that $z_i \neq \mathbf{hom}(x_i)$, where **hom** uniquely interpolates S. The protocol is depicted in Figure 3.2.

One might notice that at step 2 of the protocol, when the prover has to find λ_i, for $i = 1, ..., l$, this is equivalent to solve the discrete logarithm problem in H. But $\lambda_i \in \mathbb{Z}/p\mathbb{Z}$, so in practice we choose d such that its smallest prime factor is not too large.

3.2.3 Advantages of MOVA

MOVA has some advantages, when comparing it to classical signature schemes. To begin, MOVA scheme is an undeniable scheme. Therefore verification can be done only with the help of the signer whereas classical signature schemes

coGHIproof$_l(S, T)$

Parameters: $G, H, d = |H|, p$ smallest prime factor of d

Input: $l, S = \{(g_i, e_i)|i = 1, ..., s\} \subseteq G \times H$,
$T = \{(x_i, z_i)|i = 1, ..., t\} \subseteq G \times H$

Prover P		Verifier V
		$r_{i,k} \in_U G, a_{i,j,k} \in_U \mathbb{Z}/d\mathbb{Z}$ and $\lambda_i \in_U \mathbb{Z}/p\mathbb{Z}$ compute $u_{i,k} = dr_{i,k} + \sum_{j=1}^{s} a_{i,j,k} g_j + \lambda_i x_i$
	$\xleftarrow{u_{i,k}, w_{i,k}}$	$w_{i,k} = \sum_{j=1}^{s} a_{i,j,k} e_j + \lambda_i z_i$ for $i = 1, ..., l$, $j = 1, ..., s$ and $k = 1, ..., t$
$v_{i,k} = \mathbf{hom}(u_{i,k}), y_k = \mathbf{hom}(x_k)$ for $i = 1, ..., l$ and $k = 1, ..., t$		
find λ_i st $w_{i,k} - v_{i,k} = \lambda_i(z_k - y_k)$	$\xrightarrow{\text{Commit}(\lambda_1, ..., \lambda_l)}$	
	$\xleftarrow{r_{i,k}, a_{i,j,k}}$	
check $u_{i,k} = dr_{i,k} + \sum_{j=1}^{s} a_{i,j,k} g_j + \lambda_i x_i$		
$w_{i,k} = \sum_{j=1}^{s} a_{i,j,k} e_j + \lambda_i z_i$	$\xrightarrow{\text{open}(\lambda_1, ..., \lambda_l)}$	
		check that λ_i are correct

Figure 3.2: coGHIproof$_l(S)$

are universally verifiable (everyone can verify the signature). In some application where privacy is an issue, this property can be useful. For example we can think of an application where someone wants to sign a document, but he does not want that anyone can see that he signed it.

The other main advantage of MOVA is that we can have very short signatures (for example 20 bits as suggested in [5]). Short signature might be very useful when dealing with mobile or resource limited devices. For example, we could encode some information with his signature in a QR-code.

Chapter 4

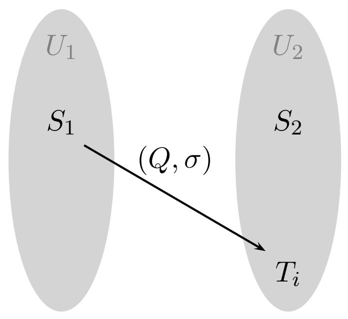
The Application

Before starting the development, the first step was to find an application on mobile devices that has to use MOVA without being too artificial. Different applications were considered and compared to choose the most adapted. The application we came to is named **University Contest**. The shortness of the signature and batch verification were the two main benefits of MOVA. In this chapter, the developed application will be presented. Then we will establish a threat model and say why MOVA is more appropriated than classical signature in our mobile application.

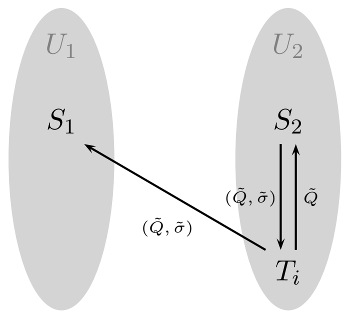
4.1 Overview

We consider an application where several universities are participating to a tournament. We can imagine that for the first round of the tournament, universities are organized into pairs. Let us consider one pair, and let us denote the two universities by U_1 and U_2. U_1 and U_2 are in competition. In this application, U_1 provides quizzes/challenges/riddles to students in U_2. Similarly U_2 provides quizzes to students in U_1. Students in a university are forming teams. We assume that teams have a certain amount of time (for example one week) to complete the current quiz/challenge/riddle. Each team in a university have a different quiz[1]. When completed they send it to the server of both universities and get a score. Then the university who provided the quiz also provides the solution of the quiz and the score the team did, to the students. At the end of the first round, all the best of the two universities (i.e. the ones with the highest cumulated score of all teams in the university) in the pairs are promoted to the second round. The game goes on the same way until the final round, where only two universities are remaining. The best of the two wins the game.

[1]If several teams in a university have the same quiz, then they can copy on each other to increase the final score.



(a) S_1 sending quizzes to teams in U_2



(b) T_i sending back the filled in quiz

Figure 4.1: Application Overview

4.2 Security Overview

For this part let us consider one pair with universities U_1 and U_2 as before. Let us denote the server of U_1 by S_1 and the server of U_2 by S_2. We can assume that communication between S_1 and S_2 are confidential, authenticated and preserve integrity. When U_1 creates a quiz/challenge for students of U_2 they sign it with MOVA and then send it to them. In that way, the students are ensured that the quiz comes indeed from U_1 and that it is not a fake quiz from other cheating students who wants to make them lose time, hence lose score. Furthermore, only the designated team can verify the signature. Then, when the students are sending back the quiz to S_2, S_2 can sign the filled in quiz with MOVA and send it to back to the team. The team can verify that the signature is correct[2], and send it to S_1. Finally S_1

[2]In that model, we consider that the students do not trust the server and want to be sure that answer were not changed or deleted.

receives the filled in quiz and verify both signatures. For signatures coming from S_2, S_1 can run a batch verification with S_2. Then when S_1 provides the solutions with score the team, he signs everything and sends it across. Therefore, students are ensured that the solution and the score are correct.

When a team is starting to communicate with a server, the team is authenticated using a preshared secret. We can assume that when a team registers, it has to choose a password.

4.3 Threat Model

Let us now describe the threat model, by showing what an adversary could do or what he might want to do if he is also a participant for example.

Fake Quizzes

The first thing that an adversary might do is to forge fake quizzes. Hence the use of the signature when sending new quizzes. A reason why an adversary might do that is to make the teams fill in useless quizzes. In that case, the adversary might be another team from the opposing university. Another reason might be to make university servers receive useless filled in quizzes.

Fill Another Team's Quiz

An adversary might want to send a quiz assigned to a team he is not belonging to, so that the team gets a zero score on that quiz. But by construction of the application, he cannot steal a quiz and make the score count for his team. Even if he is also participating to the contest because quizzes are unique and associated to their respective teams.

Modify the Answer in a Filled in Quiz

Another type of attack one could do is when the team sends the filled in quiz, the adversary could modify what is sent so that the team loses score. Therefore, as stated previously, the team sends first the filled in quiz to his university server so that it is signed. Then the team can verify that nothing was modified.

4.4 Problems

We need to authenticate the students to their university server. Because if this is not the case, then anyone can forge a valid signature. When sending the filled in quiz to the university to get the signature, if no authentication is provided then anyone can make the server sign anything[3].

4.5 Using Classical Signatures

In this application we could try to use classical signature. For the signature of the unfilled quiz (the signature signed by S_1 when S_1 is the quiz provider), MOVA could bring something compared to classical signatures, due to the following reasons:

- We use that MOVA is not universally verifiable, because only the designated team must verify the signature.
- For the signature of the filled in quiz (signed by S_2, when S_1 is the quiz provider), we exploit the fact that S_1 runs batch verification. For example at the end of the day, the server has received one quiz per team, and the signatures have to be verified. So all signatures are verified in one shot. We can assume that verification is scheduled at the end of each day or the end of a round.
- The fact that the signatures are very short is always an advantage when dealing with mobile devices. Furthermore since the server has to store some signature to run the batch verification, we gain some storage-space by having shorter signature. We could schedule the batch verification at the end of each month for example.

[3]This server only sign. He does not do any verification because all verifications are done on the other server (the one which will correct the quiz).

Chapter 5

Application Design

The University Contest application is an Android application where students form teams to answer to quizzes from other universities and score points for their own university. The application uses MOVA to sign unfilled/filled quizzes. The architecture of the application will be described in this chapter. We will also discuss some implementation choices.

5.1 Architecture

The application has the following architecture. It is composed of three participant: the Android application on which the students mainly fill the quizzes, the server of the first university and the opposing university.

5.1.1 The Android Application

The Android application on the phone has three main activities [1]:

ChallengeActivity The activity allowing the user to fill the current quiz.

TeamScoreActivity The activity used to get the score of the team and the result/correction of the quizzes.

UniversityScoreActivity The activity allowing the user to obtain the score of all universities participating the contest.

The `ChallengeActivity` is the most useful activity. When starting, it first checks if the phone already knows the MOVA public keys/parameters of both servers. If not it retrieves them from the respective server. Then it checks if there is an ongoing challenge already available on the phone. If not it tries to get a new one from the server of the opposing university. Then the student can fill the quiz and has to send it before a deadline. When sending the quiz, it is first sent to the respective university of the students which will sign it and send it back to the user. Then it automatically check the signature and sends it to the opposing university which will grade it.

The `TeamScoreActivity` is an activity containing the list of all quizzes submitted by the team with the score and the responses. It also contains the cumulated score of the team. This part is not implemented yet.

The `UniversityScoreActivity` is an activity containing the list of all universities participating the contest with their respective score. This part is not implemented yet.

5.1.2 The Servers

The two servers run the same custom protocol because of the symmetry of the application. To make simple, the client can

- Ask for the servers MOVA public key and MOVA parameters.
- Ask a new challenge (to the opposing university).
- Ask for his team score with the result of all quizzes his team submitted (to the opposing university).
- Ask for the score of all universities (to all servers, so by default it will ask his university server).
- Ask to sign his filled-in quiz (to his university server).
- Ask for the verification/denial of a message signed using MOVA.
- Send a filled-in quiz so that the server signs it (to his university server).
- Send a filled-in signed quiz so that the server grades it (to the opposing server).

The servers also communicate together to verify the signature of the filled quizzes they receive and to update the scores of the universities.

Let us discuss now how the server are implemented. All the code was made in Java [10].

Client Query Handler

This is a process that handles all queries from the client cited above. It is listening on port 12345 for new clients. To start this process just run `UniversityContestServer.class`.

Database

It is a MySQL [11] database. The name of the database is `unicontest`. It is composed of three tables:

`team` The team table contains the following informations. The team ID, the score of the team, the university of the team, and the ID of the current challenge for the team.

`challenge` The challenge table contains the following entries: the challenge ID, the associated team ID, the score that the team did for that challenge, a boolean value telling whether the filled-in challenge was received by the server, a boolean value telling whether the challenge was corrected and the name of the challenge.

`university` The university table contains the following entries: the university name and the university cumulated score.

To initialize the database, `DataBaseInit.class` can be used. But before a `java` user has to be created with the appropriated password. The other processes access the database using class `DataBaseHandler`.

Server Manager

The server manager is used by the administrator of the server. It can be used to add a new team, a new university or a new challenge. It can also be used to manage the challenges, correct them and assign scores. For more comfort, a GUI is provided. The latter can be started by running `ServerManagerGUI.class`.

5.2 Implementation

In this section we will discuss some implementation choices. First we will discuss how MOVA was implemented. Then we will discuss what choices where made for the commitment scheme and the pseudorandom generator (PRNG).

5.2.1 MOVA

Let us discuss how MOVA was implemented for the application. Given that MOVA is quiet theoretical, some implementation choices had to be made leading to some loss of generality in the scheme. For the sake of simplicity and for efficiency reasons, the following choices were made.

> Xgroup $= (\mathbb{Z}/n\mathbb{Z})^*$ for some $n = p_1p_2$ which is the product of two large prime numbers.

Ygroup $= \{-1, 1\}$ leading to $d = p = 2$.

The homomorphisms are the Legendre symbols in G. Suppose without any loss of generality that it is $\left(\frac{\cdot}{p_1}\right)$

Therefore with these choices, the public key will be defined by:

$$K_p^S = (n, \text{seedK}, (\text{Ykey}_1, ..., \text{Ykey}_{\text{Lkey}}))$$

since d is always 2 and Ygroup is always $\{-1, 1\}$. The secret key will be defined as:

$$K_s^S = p_1$$

since p_1 totally defines the homomorphism.

5.2.2 Other Cryptographic Primitives

In MOVA confirmation/denial protocol, a commitment scheme is needed. In this project, the commitment scheme was implemented as in Example 1. In this implementation a hash function is needed and SHA-256 [12] is used.

For the PRNG, we used the implementation in GNU cryto 2.1.0 [3].

5.3 The Code

Everything was coded in Java 6 and Android 2 for the application. The code is divided into three Eclipse [2] projects.

Share This project contains all shared code between the client and the server. It consists of the protocol, the MOVA scheme and all classes related to the application (such as the `Quiz` class).

UniversityContest This project is the Android project containing the client part to be installed on the mobile devices

Server This project contains the server part, which is the query handler and the server manager.

The code is commented in a Java Doc style so it will not be further discussed in this report.

Chapter 6

Conclusion

To conclude this report, we managed to develop an application for MOVA signature scheme on Android but in a strange threat model. The application may seem a bit artificial, but still it seemed to be the best amongst the one that were considered. Indeed finding a meaningful application revealed to be much more complicated than expected. Nevertheless, the shortness of the signatures was really appropriated for the application. And we could also exploit the batch verification on the server side. Thus with this application, we really managed to exploit the advantages of MOVA.

The selected application involves multiple universities participating to a contest organised by themselves. Students download quizzes from the opposing universities, try to fill them and send them back. Score is attributed to each quiz giving more chance to the university to win the contest. It seemed to us that in this application, the use of MOVA was justified.

To finish this conclusion, it would be good for future work to consider other applications that are less artificial than the one we developed. Or maybe MOVA could be a bit modified to fit to more applications.

Bibliography

[1] Android developers, 2012. `http://developer.android.com`.

[2] Eclipse, 2012. `http://www.eclipse.org/`.

[3] Paulo Barretto, Keith Burdis, Edward Kmett, Olivier Louchart-Fletcher, Casey Marshall, Raif Naffah, Mark Wielaard, and Thomas Wu. The gnu crypto project. `http://www.gnu.org/software/gnu-crypto/`.

[4] David Chaum and Hans Van Antwerpen. Undeniable Signatures. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer, 1989.

[5] Jean Monnerat. *Short undeniable signatures: design, analysis, and applications*. PhD thesis, EPFL, 2006.

[6] Jean Monnerat, Yvonne Anne Oswald, and Serge Vaudenay. Optimization of the MOVA Undeniable Signature Scheme. In Ed Dawson and Serge Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 196–209. Springer, 2005.

[7] Jean Monnerat and Serge Vaudenay. Generic Homomorphic Undeniable Signatures. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 354–371. Springer, 2004.

[8] Jean Monnerat and Serge Vaudenay. Short 2-Move Undeniable Signatures. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2006.

[9] Jean Monnerat and Serge Vaudenay. Short Undeniable Signatures Based on Group Homomorphisms. *J. Cryptology*, 24(3):545–587, 2011.

[10] Oracle. Java doc, 2011. `http://docs.oracle.com/javase/6/docs/`.

[11] Oracle. Mysql, 2012. `http://www.mysql.com/`.

[12] Processing Standards. Secure hash standard (shs). *Processing*, FIPS PUB 1(October), 2008.