Anonymous Payment Mechanisms for Electric Car Infrastructure

Author: Chao Li

Thesis submitted for the degree of Master of Engineering: Electrical Engineering

Promotors:
Prof. dr. ir. Ingrid Verbauwhede
Prof. dr. ir. Claudia Diaz
Prof. dr. ir. Johan Driesen

Assessors:
Prof. dr. ir. Bart Preneel
Prof. dr. ir. Geert Deconinck

Counsellors:
Ir. Alfredo Rial
Ir. Josep Balasch

Academic year 2010 – 2011
Without written permission of the promotors and the authors it is forbidden to reproduce or adapt in any form or by any means any part of this publication. Requests for obtaining the right to reproduce or utilize parts of this publication should be addressed to Departement Elektrotechniek, Kasteelpark Arenberg 10 postbus 2440, B-3001 Heverlee, +32-16-321130 or by email info@esat.kuleuven.be.

A written permission of the promoter is also required to use the methods, products, schematics and programs described in this work for industrial or commercial use, and for submitting this publication in scientific contests.
Preface

First and foremost, I would like to thank my daily supervisors Alfredo Rial and Josep Balasch, for their support and guidance during the development of this master thesis and also for their patience and kindness when answering questions. Without their help this thesis would not have been possible.

My sincere gratitude also goes to Prof. Ingrid Verbauwhede, prof. Claudia Diaz and Prof. Johan Driesen for giving me the opportunity to develop this thesis and for helping me whenever necessary. I sincerely thank them for their helpful advices and discussion.

I am also very grateful to Prof. Vincent Rijmen, Prof. Bart Preneel and Prof. Geert Deconinck for all of the time they have spent with me.

Last but not least, I want to thank my family and my classmates for their encouragement and support.

Author: Chao Li
## Contents

Preface ........................................... i
Abstract ........................................ iv
List of Figures .................................. v
List of Tables ................................... vi

1 Introduction ................................... 1
   1.1 Motivation .................................. 1
   1.2 Goal of the thesis ......................... 2
   1.3 Structure of the Thesis .................. 2

2 Preliminaries .................................. 3
   2.1 Notation ................................... 3
   2.2 Number Theoretic Problems .............. 4
   2.3 Cryptographic hash function .............. 4
   2.4 Commitment Schemes ...................... 5
   2.5 Zero-Knowledge Proofs ................... 6
   2.6 Anonymous Credentials .................. 9

3 E-cash Protocol ............................... 13
   3.1 Introduction of the E-Cash System ...... 13
   3.2 Structure and Specification of our scheme .................................................................. 15
   3.3 Efficiency Analysis ......................... 25

4 Implementation of the Anonymous Payment System ........................................ 27
   4.1 Overview ................................... 27
   4.2 General Structure ......................... 28
   4.3 Implementation .............................. 32
   4.4 Test of the Code ............................ 35

5 Performance Evaluation ...................... 37
   5.1 Evaluation Tool .............................. 37
   5.2 Environment ................................ 37
   5.3 Performance ................................. 39

6 Conclusions and Future Work .............. 43

Bibliography ...................................... 45
Bibliography

48
Abstract

Electric vehicles are receiving more and more support from both companies and governments as one of the most popular choices for the future green car systems. However, the need for an infrastructure of electric charging stations across the road network can threaten the driver’s privacy. In particular, information disclosed by electric payment mechanism can reveal the identity and driving patterns of users, thus threatening their location privacy.

One of the solutions to protect the drivers’ privacy is the use of E-cash. Many E-cash protocols have been proposed for the last 25 years. Nevertheless, the efficiency analysis of those protocols is hidden behind complex mathematical ideas which might limit its use in practical deployments. In this thesis, we have implemented a representative E-cash protocol and tested its efficiency.

Embedded ARM controllers are a good choice to deploy a network of charging poles due to their low cost and low power properties. In this thesis, we present the first implementation and evaluation of the merchant entity (corresponding to the charging pole) of the e-cash system in an ARM processor.

**Keywords:** Electric Car System, Recharging Pole, Privacy, E-cash, ARM
List of Figures

2.1 The Schnorr Protocol ........................................... 7
2.2 Proof of the length of a discrete logarithm .................. 8
2.3 Non-interactive proofs of knowledge ......................... 8
2.4 Proof of knowledge of a representation ..................... 9
2.5 Proof of equality of secret values ............................ 9
2.6 Proof of Polynomial Relations ............................... 10
2.7 Get Signature on a Committed value ....................... 11
2.8 Proof of Knowledge of a Signature ......................... 12
3.1 E-Cash System ................................................... 14
3.2 Overview of Our Scheme ....................................... 14
4.1 Entities of the Anonymous Electric Car Payment System ..... 28
4.2 Operation of Initialization phase ............................ 33
4.3 Operation of Withdraw phase .................................. 34
4.4 Operation of Spend phase .................................... 35
4.5 Operation of Deposit phase ................................... 35
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Data structure of rusage</td>
<td>38</td>
</tr>
<tr>
<td>5.2</td>
<td>Time of Basic Operation in PC</td>
<td>38</td>
</tr>
<tr>
<td>5.3</td>
<td>Time of Basic Operation in ARM</td>
<td>38</td>
</tr>
<tr>
<td>5.4</td>
<td>Time and Memory Cost in Initialization Phase in PC</td>
<td>40</td>
</tr>
<tr>
<td>5.5</td>
<td>Time and Memory Cost in Withdraw Phase in PC</td>
<td>40</td>
</tr>
<tr>
<td>5.6</td>
<td>Time and Memory Cost in Spend Phase in PC</td>
<td>41</td>
</tr>
<tr>
<td>5.7</td>
<td>Time and Memory Cost in Deposit Phase in PC</td>
<td>42</td>
</tr>
<tr>
<td>5.8</td>
<td>Running Time in ARM</td>
<td>42</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

Electric vehicles, regarded as one of the most popular choices for future green transportation, are attracting support from more and more governments. One big challenge for the use of electric cars is the battery: a full charged battery can only support a car to drive continuously for about only 4 or 5 hours. It means that we require a large number of infrastructures near the companies, residential areas, and amusement places for recharging our cars [3].

However, paying for recharging in those infrastructures may disclose the users’ private information such as their location privacies. Those location privacies include the drivers’ living places, working companies, the amusement places they usually go, and so on. Privacy is regarded as a fundamental human right and leaking them is possible to identify at least three key negative effects according to the recent literature [1]. The first negative effect is Location-based "spam", which means that the location information could be used by malicious businesses to bombard an individual with unsolicited marketing for products or services related to that individuals location. Another negative effect is that the location can be used to infer an individuals political views, state of health, or personal preferences. Furthermore, the disclosure of location privacy may also result in safety problems. For example, it may be used by unscrupulous persons such as the robbers for stalking or physical attacks.

In addition, the progress of information technologies has provided tools that allow to collect, store and share those data automatically, and thus help the malicious people to process those data easily. Those tools also increase the probability that various organizations cross-correlate their databases to analyze user’s privacy mutually. Due to this, the problem in privacy protection is attracting the concern from more and more users.

One possible way to achieve privacy payment is to use E-cash systems. E-cash is a kind of electronic money that tries to emulate its paper money counterpart in terms of functionality. E-cash systems are firstly proposed by David Chaum [13]. They were consist of three entities: a user, a bank and a merchant. E-cash are designed to provide users all of the important properties paper money possesses,
such as transferability, balance, and privacy. These properties are also required by our payment system.

Over the past 25 years, a great deal of research has been undertaken to improve Chaum’s original e-cash protocol, for instance [29, 7, 15, 28]. Many of those protocols are claimed to be efficient. Nevertheless, their analysis is based on complex mathematical ideas and are lack the support of real performance evaluation. Some of these systems are implemented and analyzed in practice recently. For example, A. Hajabbasgholi implemented Okamoto-Ohta’s e-Cash protocol in [23], Cuervo and Henriquez implemented an off-line e-cash protocol for mobile environment in [16], and Sarker implemented a divisible transferable E-cash protocol in [31].

However, most of the implementations focus on the user part, and no one has ever implemented the merchant entity in a resource constrained device.

1.2 Goal of the thesis

In this thesis, we have use the compact E-cash protocol [7] to build an anonymous electric car recharging system. This E-cash protocol satisfies the basic payment system requirements such as security, anonymity, unlikeliness, exculpability, balance, and double spending detection. We have implemented this protocol and carefully analyzed its performance with various lengths of secret values in PC.

We have further presented the first implement of the merchant entity (corresponding to the charging pole) in an embedded ARM controller. This processor is the best candidate for the charging poles in the electric car system because of its low power and low cost properties. We have analyzed the performance in the ARM to offer a reference for its possible implementation in industry.

Our design focus on reducing the memory consumption and enhancing the speed of the system. We achieve these mainly by using a compact E-cash protocol, a fast cryptographic library, and certain fast algorithms. Besides, we have tested the important characteristics, such as the balance and double spend detection properties, of our system.

1.3 Structure of the Thesis

The structure of this thesis can be divided into two parts: theoretical part and practical part. The theoretical part includes Chapter 2 and Chapter 3. Chapter 4 and Chapter 5 are the practical part. In Chapter 2, some necessary theoretical background is provided, in order to understand the protocols in our system. In Chapter 3, we describe the e-cash protocol used by our system in detail. Chapter 4 gives an overview of our implementation, including how to install and run our system, the general structure of it, and how have we tested this system. In Chapter 5, we present the performance analysis of our program in both PC and ARM. The thesis finishes with Chapter 6, in which possible improvements and a future work plan are given.
Chapter 2
Preliminaries

In this chapter we briefly present the basic concepts used in our schema. We start by introducing the notation together with the number theory. These are basic concepts used in many cryptographic algorithms. After that, we describe some cryptographic building blocks: the cryptographic hash function, the commitment scheme and zero-knowledge proof scheme. Finally, we depict the anonymous credential schemes and the CL signature. We did not describe the above concepts in detail. Interested readers can refer to [26] and [22].

2.1 Notation

Here we list the notations that will be used in our system, further notations please refer to [26].

- $\mathbb{Z}$ denotes the set of integers: \ldots,-2,-1,0,1,2,\ldots;
- $[a,b]$ denotes the integers $x$ satisfying $a \leq x \leq b$
- $a \in \mathbb{Z}$ means that element $a$ is a member of the set $\mathbb{Z}$
- $\sum_{i=1}^{n} a_i$ denotes the sum $a_1 + a_2 + \cdots + a_n$
- $\prod_{i=1}^{n} a_i$ denotes the product $a_1 \cdot a_2 \cdot \cdots \cdot a_n$
- $f: A \rightarrow B$ means each element $a \in A$ is precisely assigned to one element $b \in B$

Let $G$ be a non-empty set, and let $\cdot$ be a binary operation on $G$. We say that $G$ is a group if it has the following four properties: 1. Closure: $\forall a, b \in G, a \cdot b \in G$; 2. Associativity: $\forall a, b, c \in G, (a \cdot b) \cdot c = a \cdot (b \cdot c)$; 3. Identity: $\exists$ an element $\infty \in G$ such that $\forall a \in G, a \cdot \infty = \infty \cdot a = a$; 4. Invertibility: $\forall a \in G, \exists$ and only $\exists$ one $b \in G$ such that $a \cdot b = b \cdot a = 1$.

Let $p$ be a positive integer, then group $\mathbb{Z}_p$ denotes the set $0,1,\ldots,p-1$ and group $\mathbb{Z}_p^*$ denotes the set $\{ i \in \mathbb{Z} : 1 \leq i \leq p-1 \text{ and } \gcd(i, p) = 1 \}$. $\mathbb{Z}_p$ is a group under addition modulo $p$, while $\mathbb{Z}_p^*$ is a group under multiplication modulo $N$. 

3
2. Preliminaries

We call a set $S \subseteq G$ subgroup if $S$ is a group in its own right, under the same operation as that under group $G$. If $g \in \mathbb{Z}_p$ is any member of the group, we denote $\langle g \rangle = \{ g^i : i \in \mathbb{Z}_p \}$ a subgroup of $G$. If a subgroup $\langle g \rangle$ has the same order as group $\mathbb{Z}_p$, then we say $g$ is a generator, and $\mathbb{Z}_p$ is a cyclic group. We can see that if $p$ is a prime, then $2 \mathbb{Z}_p$ is a cyclic group.

An element $a$ of a group $G$ is called a square or quadratic residue, if $\exists b \in G$ such that $b^2 = a$ in $G$. We let $\text{QR}(G) = \{ g \in G : g \text{ is quadratic residue in } G \}$.

2.2 Number Theoretic Problems

The security of our payment system is based on the following number theoretic problems [7]:

Definition 1 (Strong RSA). Given an RSA modulus $n$ and a random element $g \in \mathbb{Z}_n^*$, compute $h \in \mathbb{Z}_n^*$ and integer $e > 1$ such that $h^e \equiv g \mod n$. The modulus $n$ is of a special form $pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.

Definition 2 (y-DDH). Given a random generator $g \in G$, where $G$ has prime order $q$, the values $(g, g^x, \ldots, g^{(x)^y})$ for a random $x \in \mathbb{Z}_q$, and a value $R \in G$, decide if $R = g^{1/x}$ or not.

2.3 Cryptographic hash function

A cryptographic hash function $h$ maps arbitrary finite length of input messages to fixed length of strings: $D \rightarrow R$ and $|D| > |R|$. Hash functions are typically used for data integrity as follows. The hash value corresponding to a particular message $x$ is computed at a time $T_1$. This value is protected in certain manner. At another time $T_2$, one can determine whether the message has been altered or not by comparing the new hash result of this message with the protected result [26]. This property is used in our scheme to provide the integrity of the commitments, and thus to offer a non-interactive proof. Hash functions are many-to-one, implying the existence of collisions. But in practical, the hash value should be uniquely identifiable with a single input, and collisions should be computationally difficult to find. So the hash functions are also viewed as public random oracles to justify the security of some cryptographic schemes such as the signature scheme proposed by Leighton and Micali in the paper 'Provably fast and secure digital signature algoritms based on secure hash functions' [5].

A random oracle $R$ is a map from $\{0,1\}^*$ to a truly random group $\{0,1\}^\infty$ ($\infty$ means sufficiently long so that no collision happens), chosen $y$ selecting each bit of $R(x)$ uniformly and independently for every $x$ [5, 22]. Random oracles model provides a bridge between cryptographic theory and cryptographic practice.

The Random Oracle Methodology normly consists of two steps: First, one designs an ideal system in which all parties (including the adversary $A$) have oracle access to a truly random function, and proves this ideal system to be secure (in which case,
one says that the system is secure in the random oracle model). Next, one replaces
the random oracle with a "good cryptographic hashing function" [34].

2.4 Commitment Schemes

Commitment schemes are digital analogs of sealed envelopes. They are important for
the construction of modern cryptographic protocols such as zero-knowledge proofs
and coin flipping. Making a commitment means that a player is able to choose a
value from some (finite) set and commit to this value such that he can never change
it. The player can choose to hide this value or reveal it later [33]. A commitment
scheme can be constructed based on any one-way 1-1 function. It consists of the
following three algorithms:

\[
\text{Setup}(1^k): \text{Output parameters } \text{params} \text{ on input the security parameter } k.
\]
\[
\text{Commit}(\text{params}, m, x): \text{Output a value } C \text{ on input a message } m \text{ and a}
\]
\[
\text{random string } x.
\]
\[
\text{Open}(\text{params}, C, m', x'): \text{Compute } C' = \text{commit}(\text{params}, m', x'), \text{and}
\]
\[
\text{check } C' = C, \text{ output accept or reject.}
\]

Commitment schemes have two basic properties: a binding property and a hiding
property. We give an informer description of these two properties. For formal
definitions refer to [19]

**Binding property:** A player cannot change the value anymore after it
is committed. The binding property comes in two flavors unconditional and computational. Roughly speaking, unconditional means
that a player cannot change her committed value even with infinite
computing power, while computational means that a player cannot
change her committed value even he has "very large" computing
resources.

**Hiding property:** The adversary cannot learn any information about the
committed value without the random string \( x \). It also comes in two
flavors computational and unconditional.

Pedersen proposed an efficient commitment scheme 'Pedersen Commitment' that
possesses above two properties in [30]. In this scheme, the public parameters are
one cyclic group \( \mathbb{Z}_q \) and two generators \( g, h \in \mathbb{Z}_q \), such that no one knows \( \log_g(h) \).
The player chooses a random string \( \text{open} \in \mathbb{Z}_q \) and commits to a value \( m \in \mathbb{Z}_q \) by
computing \( \text{PedCom}(m, \text{open}) = g^m h^{\text{open}} \).

Now we will briefly analyze the property of the Pedersen commitment. Since
\( x \) is selected randomly, the committed value is uniformly distributed in group \( \mathbb{Z}_q \).
Therefore, this scheme satisfies hiding and semantic property. The difficulty of
computing discrete logarithms ensures the Pedersen commitment to be an one-way
function and thus computational binding exists. It is proved that a player cannot make the receiver open the commitment to a different value \( x' \) unless he know \( \log_g(h) \).

In Pedersen commitment, prover opens a commitment in group \( p \) by demonstrating the relation \( ab = c \mod p \) among the committed members \( a, b \) and \( c \), since the group order \( p \) is known. However, we usually need to commit values under groups with hidden order such as group \( QR_n \). In this case, the prover has to prove the relation \( ab = c \) holds over the integers. Ivan Damgard and Eiichiro Fujisaki proposed such a commitment scheme in [20], which is also used in our payment system. In this scheme, the public values are one hidden order group \( G \) and two generators \( g, h \in G \). The user also commits a value \( m \in G \) by computing \( \text{PedCom}(m, x) = g^m h^{\text{open}} \). But different from the Pedersen Commitment, the random string open in this scheme should have a length at least \( k + \log_2(G) \). \( K \) is additional bit numbers to make the committed value \( c \) statistically close to uniform in the group \( G \). Since \( c \) is statistically close to uniform, the hiding property is satisfied. The binding property is also offered. The detail analysis of these two properties are shown in [20].

\section{Zero-Knowledge Proofs}

A proof of knowledge is a two-party protocol between a prover and a verifier by means of which the verifier can be convinced of the validity of an assertion. Proofs of knowledge have two properties: completeness and soundness (chapter 10 of [26]).

\textbf{Definition 3 (Completeness).} Given an honest prover and an honest verifier, the protocol succeeds with probability 1.

\textbf{Definition 4 (Soundness).} If a dishonest prover \( P \) can with non-negligible probability successfully execute the protocol with the verifier \( V \), then there exists an expected polynomial time algorithm \( M \) that can be used to extract from this prover knowledge, which with overwhelming probability allows successful subsequent protocol executions.

We call a proof of knowledge zero-knowledge proof if it allows this: the verifier cannot learn any information about the assertion itself other than one bit truth or false.

\textbf{Definition 5 (Zero-Knowledge property).} There exists an expected polynomial-time algorithm (simulator) which can produce, upon input of the assertion(s) to be proven but without interacting with the real prover, transcripts indistinguishable from those resulting from interaction with the real prover [26].

\subsection{Proofs of Knowledge About Discrete Logarithms}

In this subsection we explain how to construct proofs of knowledge about discrete logarithms. Firstly, we give the Schnorr’s identification protocol as an example. Next we talk about how to prove the length of a discrete logarithm in group of hidden order. After that we evolve the non-interactive proofs of knowledge, and introduce
Zero-Knowledge Proofs

\[
P K \{ (x) : y = g^x \} \]

<table>
<thead>
<tr>
<th>Prover(y,x)</th>
<th>Verifier(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \in \mathbb{Z}_q ), ( t = g^r )</td>
<td>( t )</td>
</tr>
<tr>
<td>( c \in \mathbb{R} { 0,1 } )</td>
<td>( c )</td>
</tr>
<tr>
<td>( s = (r - cx) \mod (q) )</td>
<td>( s )</td>
</tr>
<tr>
<td>( t' = g^s y^c = t )</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2.1:** The Schnorr Protocol

How to do non-interactive proofs of knowledge, a representation, equality of secret values, and polynomial relations.

**Schnorr protocol**

Schnorr protocol is a proof of knowledge described in Figure 2.1.

In this protocol, the prover has the knowledge of secret value \( x \), and the public generator \( g \in \mathbb{Z}_q \). She computes and publishes \( y = g^x \). To prove that she has knowledge of the secret value \( x \), the prover firstly commits to a random value \( r \in \mathbb{R} \mathbb{Z}_q \) by computing and sending \( t = g^r \). In the second round, the verifier sends a challenge \( c \) to the prover. Finally, the prover computes and sends the response \( s = r - cx \) (mod \( q \)), and the verifier is able to verify the prover’s knowledge of the value \( x \) by computing \( t' = g^s y^c \) and comparing it with the previously received value \( t \).

**Proof of the Length of a Discrete Logarithm**

In our scheme, we also use proofs of knowledge in the group of quadratic residues \( QR_n \). The value \( n \) is calculated by \( n = pq = (2p' + 1)(2q' + 1) \), where \( p, q, p' \) and \( q' \) are all primes. Two features of this group are that finding the order \( (p'q') \) of this group \( QR_n \) is as hard as factoring \( n \) and that almost all members in this group are generators. A problem of the Schnorr protocol, when used in quadratic residue group \( QR_n \), is that the prover can compute response not over modular \( s = r - cx \) (mod \( \text{ord}(QR_n) \)) but over integer \( s = r - cx \), since she does not know the order of \( QR_n \). So this protocol may disclose some information of the secret value \( x \) such as its length. To avoid the disclosure of the information, the prover should choose the random value from a quite larger range than \( x \). For example, if we choose \( x \in (0,1)^{l_x} \), then we should choose \( r \in (0,1)^{l_x+l_r+l_c} \). \( l_x \) is the length of the challenge, and \( l_r \) is the additional bit numbers to provide the hiding property.

Verifier should check the length of secret values, say \( x \), satisfy \(-2^{l_x+l_r+l_c} < x < 2^{l_x+l_r+l_c}\) by running the protocol

\[
P K \{ (x, r) : y_1 = g^{x h^{r_x}} \land -2^{l_x+l_r+l_c} < x < 2^{l_x+l_r+l_c} \} \]

This protocol is shown is Figure 2.2. The prover chooses the random number \( r_x \) and \( r_1 \) with length of \( l_x + l_s + l_c \) and \( l_n + l_s + l_c \) separately. When verifier received \( s_x := r_x - cx \), she can check \( s_x \in \{ 0,1 \}^{l_x+l_s+l_c} \) to make sure that \( x \) has a length of no more than \( l_x + l_s + l_c \). The analysis of why this protocol proves that \(-2^{l_x+l_r+l_c} < \log_g y < 2^{l_x+l_r+l_c}\) refers to \([10]\).
2. Preliminaries

\[
PK \{ (x, r_h) : y = g^{x h} \wedge -2^{l_x + l_k + l_c} < x < 2^{l_x + l_k + l_c} \}
\]

Prover(y,x) \hspace{1cm} Verifier(y)

\[
\begin{align*}
& r_x \in_R \{0, 1\}^{l_x + l_k + l_c}, \\
& r \in_R \{0, 1\}^{l_n + l_k + l_c}, \\
& t = g^{r_x h^r}, \\
& s_x = (r - c r x), \\
& s_h = (r - c r h), \\
& c \in_R \{0, 1\}^{l_c}, \\
& c' = Hash(g||t||m) & & verify \ c' = c
\end{align*}
\]

Figure 2.2: Proof of the length of a discrete logarithm

\[
PK \{ x : y = g^x \}
\]

\[
\begin{align*}
& r \in_R Z_q, \\
& t = g^r \mod n, \\
& c = Hash(g||t||m), \\
& s = (r - c r) \mod (q), \\
& c' = Hash(g||t'||m) & & verify \ c' = c
\end{align*}
\]

Figure 2.3: Non-interactive proofs of knowledge

Non-interactive proofs of knowledge about discrete logarithms.
Zero-knowledge proofs can also be non-interactive. Blum, Feldman and Micali introduced this notation in [6]; the prover and the verifier share a random string and the communication is restricted to one-way. This protocol leads to some interesting advantages and applications. We used the Fiat-Shamir heuristic to compute non-interactive proofs as described in [21]. Different from the Schnorr protocol the challenge \( c \) is generated by the hash function \( c = hash(g||t||m) \), of the concatenation of the witness \( g \) and \( t \), and the message \( m \). The construction is shown in Figure 2.3.

Proof of knowledge of a representation.
If we want to prove the knowledge of a vector of values \( x_1, \ldots, x_n \) and have the same number of bases \( g_1, \ldots, g_n \), we can perform the protocol as Figure 2.4.

Proof knowledge of equality.
We can also extend the above protocol to prove the equality relations between secret values (Figure 2.5). To prove knowledge of equality we have a vector of generators \( g_1, \ldots, g_n \), public keys \( y_1, \ldots, y_m \) and secret values \( x_1, \ldots, x_l \). The difference of this protocol with the proof of knowledge of a representation is that one secret value \( x_i \) may be used in more than one representation. We can use the notation \( e_{ij} = k \) to represent that a secret value \( x_k \) is used in public value \( y_i \) with a generator \( g_j \).
Anonymous Credentials

\[ PK\{(x_1, \ldots, x_n) : \prod_{i=0}^{n} y = g_x^i\} \]

<table>
<thead>
<tr>
<th>Prover(y,x)</th>
<th>Verifier(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_i \in {0, 1}^{t_i+t_i+e_i}, (i = 1, \ldots, n) ),</td>
<td>( t = (\prod_{i=0}^{n} g_i^i) \mod n )</td>
</tr>
<tr>
<td>( c = Hash(g_1</td>
<td></td>
</tr>
<tr>
<td>( c, s_1, \ldots, s_n \rightarrow t' = (\prod_{i=0}^{n} g_i^i y^c) \mod n )</td>
<td>( c' = Hash(g_1</td>
</tr>
</tbody>
</table>

**Figure 2.4:** Proof of knowledge of a representation

\[ PK\{(x_1, \ldots, x_l) : \land_{i=0}^{n} y_i = \prod_{j \in J_i} g_j^{x_i j} \} \]

<table>
<thead>
<tr>
<th>Prover(y_1, \ldots, y_m, x_1, \ldots, x_k)</th>
<th>Verifier(y_1, \ldots, y_m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_k \in {0, 1}^{t_i+t_i+e_i}, (k = 1, \ldots, n) )</td>
<td>( t_i = (\prod_{j \in I_i} g_j^i) \mod n ), ( (i = 1, \ldots, m) )</td>
</tr>
<tr>
<td>( c = Hash(g_1</td>
<td></td>
</tr>
<tr>
<td>( c, s_1, \ldots, s_n \rightarrow t'<em>i = (\prod</em>{j \in J_i} y_j^{x_i j} y^c) \mod n )</td>
<td>( c' = Hash(g_1</td>
</tr>
</tbody>
</table>

**Figure 2.5:** Proof of equality of secret values

The prover picks one random value \( r_k \) and computes response one \( s_k = r_k - cx_k \) for every secret value \( x_k \). The verifier will use the response values in the place where \( x_k \) is committed, similar as the proof of knowledge of a representation. However, one response \( s_k \) here may be used in different representations which satisfy \( e_{ij} = k \).

**Proof knowledge of Polynomial Relations.**

We briefly introduce two polynomial relations: linear relations and multiplicative relations. For instance, we can prove a linear relation \( 2x_1 + 7x_2 = 5 \mod n \) by using the proof:

\[ PK\{(x_1, x_2) : y_1 = g^{x_1} \land y_2 = g^{x_2} \land g^5 = y_1^2 y_2^7\} \]

And we can prove a multiplicative relation by:

\[ PK\{(x_1, x_2, x_3) : y_1 = g^{x_1} \land y_2 = g^{x_2} \land 1 = y_1^{x_3} (1/g)^{x_1}\} \]

The proof of the multiplicative relation is described in Figure 2.6.

**2.6 Anonymous Credentials**

Anonymous credentials are important methods to certify a user while protecting her privacy. In this section, we describe anonymous credentials and their properties.
2. Preliminaries

\[ PK \{ (x_1, x_2, x_3, R_1, R_2, R_3) : y_1 = g^{x_1}h^{R_1}, \]
\[ y_2 = g^{x_2}h^{R_2} \land y_3 = g^{x_3}h^{R_3}, \]
\[ 1 = y_1^{x_2}(1/g)^{x_3}(1/h)^{R_1R_2} \}\]

Prover\((y_1, \ldots, y_3, x_1, \ldots, x_3)\) Verifier\((y_1, \ldots, y_3)\)

\[ r_{x_1}, \ldots, r_{x_3} \in R \{0, 1\}^{l_s+lc+ln}, \]
\[ r_1, \ldots, r_3 \in R \{0, 1\}^{l_s+l_c+l_n}, \]
\[ t_1 = g^{r_{x_1}h^{R_1}}, \ldots, \]
\[ t_3 = g^{r_{x_3}h^{R_3}} \]
\[ t_m = y_1^{r_{x_2}}(1/g)^{r_{x_3}}(1/h)^{r_h} \]
\[ c = Hash(g||h|| t_1|| t_2|| t_3|| t_m||m) \]
\[ s_{x_1} = r_{x_1} - cx_1 \]
\[ s_1 = r_1 - cR_1 \]
\[ s_{x_2} = r_{x_2} - cx_2 \]
\[ s_2 = r_2 - cR_2 \]
\[ s_{x_3} = r_{x_3} - cx_3 \]
\[ s_3 = r_3 - cR_3 \]
\[ s_h = (r_h - cR_1R_2) \]
\[ \frac{c, s_1, s_2, s_3, s_{h_1}, s_h}{t'_1 = g^{s_{x_1}h^{s_1}y_1}} \]
\[ \frac{c, s_1, s_2, s_3, s_{h_1}, s_h}{t'_2 = g^{s_{x_2}h^{s_2}y_2}} \]
\[ \frac{c, s_1, s_2, s_3, s_{h_1}, s_h}{t'_3 = g^{s_{x_3}h^{s_3}y_3}} \]
\[ \frac{c, s_1, s_2, s_3, s_{h_1}, s_h}{t_m = y_1^{s_{x_2}}(1/g)^{s_{x_3}}(1/h)^{s_h}} \]
\[ \frac{c, s_1, s_2, s_3, s_{h_1}, s_h}{c' = Hash(g||h|| t'_1|| \ldots || t'_m||m)} \]
\[ t_m = y_1^{r_{x_2}}(1/g)^{r_{x_3}}(1/h)^{r_h} \]
\[ c' = c \]

Figure 2.6: Proof of Polynomial Relations

Then we further describe a credential scheme which is used in our system.

2.6.1 Definition

A credential is a set of certified statements that can be used by owners to demonstrate that they are authorized to access a service. It consists of data items, provided by a user, and a digital signature issued by a credential issuer. There are three entities in a credential system: owners, issuers and verifiers. An owner can obtain a signature on a set of statements from an issuer and prove possession of this signature to the verifiers in a way that this owner can choose the amount of information she wants to disclose [8].

Anonymity means that the verifiers and the issuers cannot link the user’s request for a certificate to her proof that she possesses a signature. In addition, if user proves possession of a signature multiple times, these proofs cannot be linked to each other, which is called unlinkability. These two properties make Anonymous credentials a good privacy-preserving authentication mechanism.
Anonymous Credentials

1, User ← Issuer The user forms commitment $y_n = a^x b^r \pmod{n}$ and runs the protocol

$$PK\{(x, r_c, r) : y = g^x h^{r_p} \land y_n = a^x b^r \land -2^{l_x+l_s+l_c} < x < 2^{l_x+l_s+l_c} \land -2^{l_r+l_s+l_c} < r < 2^{l_r+l_s+l_c}\}$$

to prove issuer knowledge of $x$ and $r$, and $-2^{l_x+l_s+l_c} < x < 2^{l_x+l_s+l_c}$.

2, User ← Issuer The issuer chooses a random number $r'$ of length $l_r+l_s+l_c$, and a prime number $e$ of length $l_e$. Computes $v = (y_n b^{r'} c)^{1/e}$ and sends $(r', e, v)$ to the user.

3, User computes $s = r + r'$, the $(s, e, v)$ is a valid signature.

**Figure 2.7:** Get Signature on a Committed value

### 2.6.2 Camenisch-Lysyanskaya (CL) Signature

An anonymous credential scheme is usually a signature scheme provided with two protocols: (1) a protocol to obtain a signature by an issuer on a set of committed values; (2) a protocol to prove possession of the signature to a verifier.

In [8] Camenisch and Lysyanskaya proposed a signature scheme which is widely used to construct anonymous credential systems together with commitment schemes. They introduced an anonymous credential system based on this signature. This anonymous credential system contains three phases: (1) proving equality of two committed values; (2) getting a signature on a committed value; and (3) proving knowledge of a signature on a committed value. We briefly describe this anonymous credential system here.

**Initial:** First of all, the issuer generates two groups $QR_n$ and $\mathbb{Z}_p^*$, one for commitment and signature the other for identification. Then the Issuer generates and publics several generators $(a, b, c ...)$ and $(g, h, ...)$ in these two groups. User generates secret values $x$ and $r_p$ and commits them as $y = g^x h^{r_p} \pmod{p}$.

$l_x$, $l_c$, and $l_e$ are the length of the secret values, the challenge, and the prime exponent $e$ of the signature. $l_e$ should be larger than $l_x + 2$ (usually choose $l_e = l_x + 3$). $l_r$ controls the statistical zero-knowledge property, and the typical value is 80 bits.

**Obtain a signature.** The protocol to get a signature on the committed value is shown in Figure 2.7.

**Prove knowledge of a signature** To prove the knowledge of secret data items
2. Preliminaries

1. User → Issuer The user chooses random values \( r_v \) and \( r_w \) of length \( l_n \), computes \( v' = v g^{r_v} \), \( y_w = g^{r_v h^{r_w}} \). (Notice that \((s + er_v, s, v')\) is also a valid signature), and sends \( v' \) and \( y_w \) to the issuer.

2. User ↔ Issuer The user and the verifier run following zero-knowledge proofs of knowledge:

\[
PK\{(x, e, s, r_v, r_w, \phi, \beta) : \\
c = v'^e (1/a)^x (q/b)^s (1/g)^\phi \wedge y = g^x h^\phi \\
y_w = g^{r_v h^{r_w}} \wedge 1 = y_w^e (g/q)^\phi (1/h)^\beta \wedge \\
2^{l_e-1} < e < 2^{l_e} \wedge \\
2^{l_m-1} < x < 2^{l_m} \}
\]

Figure 2.8: Proof of Knowledge of a Signature in a signature, a protocol is shown in Figure 2.8.

We used the CL signature in our system because it is an efficient way to prove knowledge of a signature.
Chapter 3

E-cash Protocol

3.1 Introduction of the E-Cash System

As mentioned in the chapter 1, achieving location privacy is a big problem for electric car systems. We solved this problems by using an anonymous e-cash system.

E-cash is one of the electronic payment systems. The first scheme proposed was David Chaum’s fully anonymous digital cash protocol [14]. We chose e-cash, because it provides user anonymity against both the bank and the merchant during the spending phase, even though the bank is responsible for giving out electronic coins and for accepting them for deposit [7]. In addition, it offers unlikability. These two properties ensure that a malicious merchant is unable to relate the spending record to one user to harm her privacy.

There are two flavors: on-line and off-line. Figure 3.1 shows an on-line e-cash system. This system contains three entities: the user, the bank and the merchant. The user must open an account in the bank firstly. Then she can use that account to withdraw coins from the bank. After that she is able to use those coins to purchase goods or services from a merchant. The merchant sends the coins to the bank for verification and deposit, and the bank will inform the merchant that the coin is valid and deposit those coins into the merchant’s account if the verification passes. The bank is also responsible for identifying double spenders, since it is easy to duplicate the electronic data. In order to do this, the bank should maintain a database of all used coins. In order to do this, the bank should maintain a database of all used coins. In an off-line scenario, the merchant accepts a payment autonomously, and later submits the payment to the bank. This requires the merchant’s ability to verify the coin.

Our electric car payment system is shown in Figure 3.2. The bank is the same as the bank in the e-cash system. The electric infrastructure (or charging pole) and the driver are the counterparts of the merchant and the user in the e-cash system. It works as the e-cash system. The drivers firstly open accounts in the bank and withdraw coins from their accounts. After that they can spend those coins to recharge their cars in the charging poles. During the spend, the charging poles verify the payment and store the coins if they are valid. The charging poles deposit the stored
3. E-cash Protocol

 coins regularly (say everyday in the morning) to the bank. If those coins are valid, the bank stores them to the merchant’s account.

There are many e-cash schemes such as [29, 7, 15, 28, 27]. We chose the first scheme in [7] mainly for four reasons. Firstly, it achieves all of the basic security requirements in the electric car system like anonymity and unlinkability, and offers exculpability of users (the bank can prove to a third party that a user has double-spent). Secondly, it is a compact system. The user only uses $\mathcal{O}(l+k)$ (k is the security parameter) bits to store $2^l$ coins. Thirdly, the communication complexity of withdrawal and spend operations is $\mathcal{O}(l+k)$. Finally, it is an off-line system.

14
3.2 Structure and Specification of our scheme

In this section, we briefly introduce the structure of our scheme, how it works, and the specification of it. The scheme consists of six procedures: initialization, withdrawal, spending, deposit, identify and verify guilt.

3.2.1 Initialization Phase \( (A(1^k, \mathcal{L}), U(1^k, \mathcal{L}), M(1^k, \mathcal{L})) \)

The initialization phase is used to generate the public values for commitment and signatures, and secret keys and public keys of users and merchants. In this phase, the certification authority \( A \) receives as input a security parameter \( 1^k \) and outputs key pair \((pk_B, sk_B)\); user and merchant input security parameters and output \((pk_U, sk_U)\) and \((pk_M, sk_M)\) respectively. The bank also records \((pk_U, \text{user info})\) in a database for validating users and identifying double spenders.

First, the certification authority generates a CL signature key pair \((pk_B, sk_B)\) and publishes \(pk_B\). Second, users and merchants generate their ELGamal key pair \((pk_U, sk_U)\) and \((pk_M, sk_M)\), and send their public keys to the certification authority. The detail operation is shown below.

certification authority

The certification authority generates a CL signature [9] key pair \((pk_B, sk_B)\):

1. Find two safe primes (as explained in Section 7 of [18]), i.e.:

\[
p = 2p' + 1, \quad q = 2q' + 1.
\]
3. E-cash Protocol

a) Generate a random, odd number $p'$ of desired length, say k bits
b) Test if either $p'$ or $p = 2p' + 1$ are divisible by any primes up to the bound $B$ ($B = (k/2)^4$), if so go back to Step 1.
c) Test if 2 is a Miller-Rabin witness of the compositeness of $p'$, if so go back to Step 1.

- Express $p' - 1$ as $2^d$ ($d$ is odd)
- $x \leftarrow 2^d \pmod{p'}$
- If $x = 1$ or $x = p' - 1$ then return probably prime
- For $r = 1$ to $s - 1$
  - $x \leftarrow x^2 \pmod{p'}$
  - If $x = p' - 1$ then return probably prime
  - Return composite
d) Set $p = 2p' + 1$, and test if $2^p \equiv \pm 1 \pmod{p}$. If not, go back to Step 1.
e) Apply the Miller-Rabin test to $p'$ $t$ times using randomly selected bases.
  (The error probability of $\epsilon$ is lower than $2^{-80}$ if choose $t = 6$).

2. Calculate the special RSA modulus as:

$$n = pq.$$ 

3. Find a random generator $Q$ in the quadratic residues group $QR_n$ (as described in Section 4.3 of [4]).

a) Generate a random number $a \in R Z_n^*$ satisfying $gcd(a \pm 1, n) = 1$
b) $Q = a^2 \pmod{n}$ is a generator of group $QR_n$

4. Calculate the bases $V_1, V_2, V_3, V_4, V_5$ as:

$$v_1, v_2, v_3, v_4, v_5 \in R \left[1, p'q\right],$$

$$V_1 = Q^{v_1} \pmod{n}, \quad V_2 = Q^{v_2} \pmod{n},$$

$$V_3 = Q^{v_3} \pmod{n}, \quad V_4 = Q^{v_4} \pmod{n},$$

$$V_5 = Q^{v_5} \pmod{n}.$$ 

5. Pick a random generator $Z \in R QR_n$.

6. Store the secret key $sk_B = p$, and output the public key parameters $pk_B = (n, V_1, V_2, V_3, V_4, V_5, Q, Z)$.
The certification authority generates generators \([21]\) for the public key or user and merchant, coin serial numbers and security tags

1. Find a Sophie-Germain primes \(q\) such that \((p - 1) = 2q\), \(p\) is also a prime

2. Generate a random generator \(g\) of the subgroup \(G\) of order \(q\) of \(\mathbb{Z}_p^\ast\):
   a) Find a random generator of \(\mathbb{Z}_p^\ast\): \(g' \in R\mathbb{Z}_p^\ast\) with \(g' \neq \pm 1\) and \(g'^q \neq 1\) (mod \(p\))
   b) Then \(g = g'^2\) (mod \(p\)) is a generator of the subgroup \(G\) of \(\mathbb{Z}_p^\ast\)

3. Calculate the bases \((g_U, g_M)\) as:
   \[
   r_U, r_M, r_0, r_1, r_2, r_3 \in R\mathbb{Z}_p^\ast
   g_U = g^{r_U} \pmod{p}, g_M = g^{r_M} \pmod{p}
   \]

4. Public \((g_U, g_M)\)

**User**

The user \(U\) generates an EL Gamal key pair \((pk_U, sk_U)\)

1. Generate random number \(u \in R\mathbb{Z}_q\)

2. Store the secret key \(sk_U = (u)\), and send the public key \(pk_U = g^u_U \pmod{p}\) to the certification authority.

**Merchant**

The merchant \(M\) generates a EL Gamal key pair \((pk_M, sk_M)\) as the user, and send her public key to the certification authority.

### 3.2.2 Withdraw Phase \((U(pk_B, sk_U, 2^l), B(pk_U, sk_B, 2^l))\)

In this phase, the user runs the withdraw protocol with the bank to get a wallet of \(2^l\) coins. The bank inputs her secret key \(sk_B\) and the user’s public key, and outputs success or failure; the user inputs her secret key \(sk_U\) and bank’s public key \(pk_B\), and outputs wallet \(W = (u, s, t, \sigma, J)\). \(\sigma = (sQ, e, d)\) is the bank’s signature on \((u, s, t)\), and \(J\) is the wallet size \(2^l\).

This protocol contains five steps. In the first step, the user identifies herself to the bank by proving knowledge of her secret key \(u\). In the second step, the user and the bank contribute randomness to the wallet secret \(s\): the user picks random values \(s', t \in R\mathbb{Z}_q\), and send a commitment \(A' = \text{commit}(u, s', t)\) to the bank; the bank reply a random \(r' \in R\mathbb{Z}_q\) to the user; then the user and the bank can compute a commitment \(A\) to the value \((u, s, t)\), the user also computes the wallet secret \(s = s' + r'\). In the third step, the user and the bank run the CL signature for obtaining the bank’s signature \(\sigma\) on the committed value \((u, s, t)\). Finally, the user store the wallet \(W = (u, s, t, \sigma = (sQ, e, d), J)\) (Notice: From now on, all the exponentiations below are under moduler, so we omits the notation mod \(n\) or mod \(p\)
3. E-cash Protocol

1. The user identifies himself to the bank by sending $pk_U = y_U$

2. The bank contacts the certification authority to make sure $y_U$ exists.

3. The user commits the secret values $(u, s', t)$.

   **User:**
   
   a) Generates random numbers $t, s' \in R \mathbb{Z}_q$ and $r'_Q \in R (1, \lceil \frac{n}{4} \rceil)$ ($\frac{n}{4}$ is an approximation of the group length of $QR_n$).
   
   b) Calculates
   
   $$A' = V_1^u V_2^s V_3^t Q^{r'_Q}$$
   
   c) Sends bank $A'$.

4. The bank contributes randomness to the serial number $s$ of user’s wallet:

   **User:**
   
   a) Bank sends a random $r' \in \mathbb{Z}_q$ to user.
   
   b) User computes $s = s' + r'$ and computes a new commitment $A = V_1^u V_2^s V_3^t Q^{r'_Q}$
   
   c) Bank computes $A = V_2^{r'} A'$

5. The user identifies himself to the bank by proving knowledge of secret key $U$ and run the CL protocol [11] with bank to obtain a signature on the committed values by providing the proof:

   $$SPK7\{ (u, s, t, r'_Q) : y_U = g_U^u \land A = V_1^u V_2^s V_3^t Q^{r'_Q} \land u, s, t, r'_Q \in \{0, 1\}^{l_u + l_s + l_t} \}$$
Structure and Specification of our scheme

User:

a) Generates random number

\[ R_1, R_2, R_3, R_Q \in \{0, 1\}^{l_c + l_n + l_r} \]

\( l_n \): group length : maximum(\(\text{ord}(QR_n)\), \(\text{ord}(\mathbb{Z}_q)\)),

\( l_c \): length of hash function,

\( l_r \): additional bits for masking the information of secret exponents, e.g. the length

b) Computes commitment

\[ T_U = g_{U}^{R_1}, \quad T_A = Q^R \prod_{i=1}^{3} V_i^{R_i} \]

c) Generate challenge \( C = H(S || T_U || T_A) \)

\( (S = g_U || y_U || V_1 || ... || V_3 || Q || A) \)

d) Calculates

\[ s_1 = R_1 - Cu, \quad s_2 = R_2 - Ct, \]
\[ s_3 = R_3 - Cs, \quad s_4 = R_Q - Cr'_Q \]

e) Sends bank the tuple \((C, s_1, ..., s_4)\)

Bank:

a) Computes

\[ T'_U = y_{U}^{s_1}, \quad T'_A = A^{C} Q^{s_4} \prod_{i=1}^{3} V_i^{s_i} \]

b) Computes \( C' = H(S || T'_U || T'_A || m) \)

c) Verifies \( C' = C \)

d) Verifies

\[ s_1, s_2, s_3, s_4 \in (-2^{l_c + l_n + l_r}, 2^{l_c + l_n + l_r}) \]

Using public key \( y_U \) to calculate \( T'_U \) proves the user knows the secret key \( u \) and thus authenticates the user. Using same \( s_1 \) in \( T'_U \) and \( T'_A \) proves the exponent value of \( V_1 \) in the signature is also \( u \).

6. The user obtains bank’s signature by:
### 3. E-cash Protocol

**Bank:**

a) Chooses a random number $e'$ of length $l_e$ bits so that $e := 2^{l_n} + e'$ is a random prime number ($l_e = l_m + 3$)

b) Generates a random number $S_Q'$ of length $l_n$

c) Computes $d = (\frac{Z}{A Q^{s_Q'}})^{1/e} = (\frac{Z}{Q^{s_Q'+V_1 V_2 V_3}})^{1/e}$

d) Sends user $(S_Q', e, d)$

**User:**

a) Verifies $e$ is a prime number

b) Calculates: $s_Q = S_Q' + r_Q'$

c) Verifies $Q^{s_Q V_1 V_2 V_3 d^e} = Z$

d) If all proved, then $(s_Q, e, d)$ is a valid signature

7. The user stores the wallet $(u, s, t, (s_Q, e, d), J)$, $(J = J_0 + J_1)$

   $(J$ lies in a odd interval $J_0 - J_1 \leq J \leq J_0 + J_1$ (e.g. $J_1 = 2^{l-1}$, $J_0 = 2^l$) so that coin serial number $S$ and double spending equation $T$ can be computed as $S = g^{1/(J+s)}$, $T = pk_t F^{DY}_{g,t}(J)^R = g^{u g^{R/(J+t)}}$ to simplify the proof (appendix of [7])).

3.2.3 **Spend phase** $(U(W, pk_M), M(sk_M, pk_B, 2^l))$

The user spends coins to the merchant by providing a valid signature. The user inputs the wallet, and output success or failure, and a new wallet size $J$. The merchant inputs her secret key for identification, bank’s public key for verification, and the wallet size $2^l$ for checking balance, and outputs an e-coin or failure.

In the first step, the merchant sends the user a random number $r_M \in_R \mathbb{Z}_p^*$ to compute $R = H(pk_M || r_M)$ as an unique identifier of the spending. In the second step, the user provides the merchant with a coin $(S, R, T, \phi)$, where the $S$ is coin serial number, $T$ is double-spending tag and $\phi$ is a proof of knowledge computed in the Fiat-Shamir heuristic. In the third step, the merchant verify the validity of the coin, and provides the user the goods or services she wants to buy if the proof is correct. Finally, the user update her wallet size.

1. $M$ (Merchant) sends $pk_M$ and string info $\{0,1\}^*$ that containing transaction information and random string to User, and run protocol to prove knowledge of $sk_M$

2. The user computes $R = H(pk_M || r_M)$
3. The user computes $w_i$ so that $\sum_{i=1}^{4} w_i^2 = J_1^2 - (J - J_0)^2$ as described in [25]

\begin{itemize}
  \item[a)] Find $t$ and $k$ such that $u = J_1^2 - (J - J_0)^2 = 2^t(2k + 1)$
  \item[b)] Assume $t=1$:
    \begin{itemize}
      \item[i)] Pick random values $w_1 \leq \sqrt{u}$ and $w_2 \leq \sqrt{u - w_1^2}$, such that exactly one of them is even. Let $p = u - (w_1^2 + w_2^2)$, now $p \equiv 1 \pmod{4}$
      \item[ii)] Assume $p$ is prime, try to express $p = w_3^2 + w_4^2$ as follows:
        \begin{enumerate}
          \item find a solution $u$ to the equation $u^2 \equiv -1 \pmod{p}$
          \item apply the Euclidean algorithm to $(u, p)$, and take the first two remainders that are less than $\sqrt{p}$ to be $w_3$ and $w_4$;
          \item verify $p = w_3^2 + w_4^2$. If not, $p$ is not a prime, so go back to step b).
        \end{enumerate}
      \item[iii)] Return $(w_1, ..., w_4)$ as a representation.
    \end{itemize}
  \item[c)] If $t$ is odd but not 1, calculate $s = 2^{(t-1)/2}$ and return $(sw_1, ..., sw_4)$.
  \item[d)] If $t$ is even:
    \begin{itemize}
      \item[i)] Regroup the representation so that $w_1 \equiv w_2 \pmod{2}$ and $w_3 \equiv w_4 \pmod{2}$;
      \item[ii)] Calculate $s = 2^{t/2-1}$
      \item[iii)] Return $(s(w_1 + w_2), s(w_1 - w_2), s(w_3 + w_4), s(w_3 - w_4))$
    \end{itemize}
\end{itemize}

4. The user calculates:

\begin{align*}
A &= V_4^J V_5^{r_A}, \quad B = V_4^J V_5^{r_B}, \quad W_i = V_4^{w_i} V_5^{r_{wi}} (i = 1, 2, 3, 4), \\
Double - spending equation : T &= p q_d F_4^{DY} (J) R = g_u g_v R^{/(J+1)}, \\
Coin serial number : S &= F_4^{DY} (J) = g_1^{(J+1)}
\end{align*}

5. The user sends to the merchant Zero-Knowledge proof:

$\Phi = SPK \{(\alpha, U, s, t, J, e, r_A, r_B, w_1, ..., w_4, r_{w_1}, ..., r_{w_4}, h_5, h_7) :$

\begin{align*}
Z &= q_d^4 V_1^u V_2^u V_3^d d^e \land A = V_4^J V_5^{r_A} \land \\
    g_u &= S^{(J+s)} \land W_i = V_4^{w_i} V_5^{r_{wi}} \land \\
    V_4^{(J_1^2 - J_2^2)} &= (AV_4^{2-J_0})^4 \prod_{i=1}^{4} W_i^{w_i} V_5^{r_{wi}} \land \\
    B &= V_4^u V_5^{r_B} \land 1 = B_1 B_1^{/(V_4^a V_5^{r_B})} \land \\
    b_d^R &= T^{(J+t)} (1/g_u)^\alpha (u = (J + t)) \land \\
    u, s, t \in \{0, 1\}^{l_n+l_e+l_r} \land e \in \{0, 1\}^{l_e+l_n}
\end{align*}
3. E-cash Protocol

\{(Z, V_1, V_2, V_3, V_4, Q, d', g_{Si}, S, T, A, B, W_1, W_2, W_3, W_4, info)\}
a) Generates random numbers

\[ r_A, r_B, r_1, \ldots, r_4, r_5, r_w, \in \{0, 1\}^{l_0}, \]
\[ r_u, r_s, r_t \in \{0, 1\}^{l_0 + l_c + l_r}, \]
\[ r_j \in \{0, 1\}^{l_0 + l_c + l_r}, \]
\[ r_{h1}, \ldots, r_{h4}, r_Q, r_{h5}, r_h7 \in \{0, 1\}^{l_0 + l_c}, \]
\[ r_{d'} \in \{0, 1\}^{l_0 + l_c + l_r} \]

b) Computes \( d = d'Q^r \) (note \((s_Q + e \ast r, e, d')\) is also a valid certificate)

c) Calculates

\[ S_c = Q||V_1||\ldots||V_5||d'||g_{\ell_e}||S||W_1||\ldots||W_4||A||B||T \]
\[ T_Z = Q^{r_Q}V_1^{r_1}V_2^{r_2}V_3^{r_3}d'^{r_\ell_e}, \quad T_A = V_4^{r_{j_1}}V_5^{r_{h1}}, \]
\[ T_{g\ell_e} = S^{r_{j_1}}S^{r_{s}}, \quad T_{W_1} = V_4^{r_{j_1}}V_5^{r_{h1}}, \]
\[ T_{jV_4} = (AV_4^{−2h_0})^{r_{j_1}}W_1^{r_{w_1}}\ldots W_4^{r_{w_2}}V_3^{r_{h5}}, \]
\[ T_B = V_4^{r_{j_1}}V_5^{r_{h6}}, \]
\[ T_1 = B^{r_{j_1}}B^{r_{s}}(1/V_4)^{r_{h_7}}V_5^{r_{h_7}}, \quad T_{gR} = T^{(r_{j_1} + r_{s})}/g_{\ell_e}^{r_{s}}, \]
\[ T_e = T_Z||T_A||T_{g\ell_e}||T_{W_1}||T_{jV_4}||T_B||T_1||T_{gR} \]

d) Computes \( c = H(S_c||T_e) \)

e) Computes the merchant:

\[ s_j = r_j - cJ, \quad s_{u_1} = r_{u_1} - cU, \quad s_s = r_{s} - cs, \quad s_t = r_t - ct, \]
\[ s_{w_1} = r_{w_1} - cw_{w_1}, \quad s_\alpha = r_\alpha - c\alpha, \quad s_{hi} = r_{hi} - cr_{w_1}, \]
\[ s_{s_Q} = r_Q - cs_Q, \quad s_d = r_{d'} - ce, \]
\[ s_{h_5} = r_{h_5} - ch_5, \quad s_{h_6} = r_{h_6} - r_B, \quad s_{h_7} = r_{h_7} - ch_7, \]
\[ (s'_Q = s_Q + er, \quad h_5 = -Jr_A - \sum_{i=1}^{4} r_iw_i, \quad h_7 = -(J + t)r_B) \]

f) Sends the following values to the merchant:

\( d', \text{ commitments} : (A, B, W_1, \ldots, W_4, T, S), \)

\( \text{responses} : (s_j, s_{u_1}, s_{s}, s_{t}, s_{w_1}, \ldots, s_{w_4}, s_\alpha, s_{h_1}, \ldots, s_{h_7}, s_{s_Q}, s'_d), \)

and the challenge \( c \)
3. E-cash Protocol

The merchant can verify the zero-knowledge proof by:

a) Computing \( R = H(pk_M || r_M) \)

b) Calculating

\[
S'_c = Q || V_1 || \ldots || V_5 || d'|| g_{ld} || S || W_1 || \ldots || W_4 || A || B || T
\]

\[
T'_Z = Q^{s_j} V_1^{s_M} V_2^{s_M} d' || s_d' || Z', \quad T'_A = V_4^{s_j} V_5^{s_M} A^{c'},
\]

\[
T'_{g_{ld}} = S^{s_j} S^{s_M} g_{ld}, \quad T'_W = V_4^{s_j} V_5^{s_M} W^{c'},
\]

\[
T'_{J_{V_4}} = (AV_4^{-2j_0})^{s_j} W_1^{s_{W_1}} \ldots W_4^{s_{W_4}} V_5^{s_{V_5}} g_{ld}^{(R^2 - J_0^2)^{sc}},
\]

\[
T'_B = V_4^{s_j} V_5^{s_{V_5}} B^{c'}, \quad T'_1 = B^{s_j} B^{s_1} (1/V_4)^{s_5} V_5^{s_{V_5}},
\]

\[
T'_{g_R} = T^{(s_j + s_t)} (1/g_{ld})^{s_{g_{ld}}} g_{ld}^{(R^2)},
\]

\[
T'_c = T'_Z || T'_A || T'_{g_{ld}} || T'_W || T'_{J_{V_4}} || T'_B || T'_1 || T'_{g_R}
\]

c) Calculating \( c' = H(S'_c || T'_c || M) \)

d) Verifying

\[
c' = c, \quad s_u, s_s, s_t \in \{0, 1\}^{l_u + l_s + l_r}, \quad
\]

\[
and \quad s_e \in \{0, 1\}^{l_e + l_u + l_r}
\]

6. If the proof is correct, the merchant saves the coin

\[
(S, \pi = (R, T, \phi = (d', A, B, W_1, W_2, W_3, W_4, r_M)),
\]

responses : \((s_j, s_U, s_s, s_t, s_{w_1}, \ldots, s_{w_4}, s_a, s_{h_1}, \ldots, s_{h_7}, s_{s_Q}, s_d'),\)

and the challenge \( c \)

7. User updates her counter \( J = J - 1 \)

3.2.4 Deposit phase \((M(sk_M, (S, \pi), pk_B), B(pk_M, sk_B, 2^l))\)

The merchant inputs her secret key, the stored coins \((S, \pi))\), and bank’s public key, outputs success or fail. The bank inputs her secret key, merchant’s public key and the wallet parameter \(2^l\), and outputs success or fail, and user’s identity if she found double spending.

There are three steps in this phase. In the first step, the merchant sends the coins \((S, R, T, \phi)\) to the bank. In the second step, the bank checks the freshness of R and S. If R is not fresh, it means the merchant double deposits coins; if only S is not fresh, it means it is very likely that the user double spends the coins, then trigger the Identification. If both R and s are fresh. In the final step, the bank
verify the signature of knowledge $\phi$, and store the coins to the merchant’s account if verification passes.

1. The merchant $\mathcal{M}$ sends to bank $\mathcal{B}$ a coin $(S, \pi = (R, T, \phi))$
2. The bank verifies if $R$ is fresh
3. The bank verifies the proof $\phi$
4. If $R$ is fresh and the proof is correct, the bank will accept the coin and save $(S, R, T)$ in database, otherwise sends an error to $\mathcal{M}$.
5. The bank searches every $(S_1, R_1, T_1)$ in the database, and compares $S$ with $S_1$. If $S$ appeared before, trigger the Identification algorithm to check who used the same coin twice.

Notice that if the bank is satisfied with detecting/identifying double-spenders, without worrying about proving anything to a third party, she only needs to store three values $(S_1, R_1, T_1)$ for one coin. This is important because a bank needs to store the coins of all of its users for a extremely long time (several years or more).

### 3.2.5 Identification ($\mathcal{I}, S, \pi_1, \pi_2$)

Inputs the double spent coins $\pi_1$ and $\pi_2$. Outputs the double spender’s public key.

1. The bank computes $(T_{R_1}^{R_1})^{-1} = \left(\frac{g^{u(R_1+R_1J_1+1)}}{g^{uR_1}}\right)^{(R_1-R)^{-1}} = (g^u(R_1-R))(R_1-R)^{-1} = g^uU$

   (notice that in the case of double spending $S_1 = S$, and $J_1 = J$)

2. The bank searches the database to find out the user who’s public key is equal to $y_U$.

### 3.3 Efficiency Analysis

We only discuss the efficiency of withdraw, spend and deposit phase here, since initialization phase happens only once.

**Withdraw Phase**

The dominant computational cost in this phase are multi base modular exponentiations, and inversion. The user does two modular exponentiations ($A = g_2^uA'$, $T_u = g_2^{U_2}$) and two multi base modular exponentiations ($A' = V_1^iV_2^{e_1}V_3^{d_1}$, $T_A = Q^{RQ}\prod_{i=1}^{3}V_i^{R_i}$); the bank also needs to do two modular exponentiations ($A = g_2^uA'$, $AQ^{U_2}$), two multi base modular exponentiations ($T_u = g_2^{U_2}$, $T_A = Q^{RQ}\prod_{i=1}^{3}V_i^{R_i}$), and two inversions ($\frac{1}{AQ^{U_2}}, \frac{1}{R}$).

The user only stores the wallet secret $(u, s, t)$ and one signature with three values $(s_0, e, d)$ for one wallet regardless of the coin number. This solves the storage limitation issue of user’s smart card.
3. E-cash Protocol

Spending Phase
The dominant computational cost in this phase are multi base exponentiations and
lagrange representation calculation. The user does one lagrange representation cal-
culation, nine multi base exponentiations \((A, B, W_1, ..., W_4, T, S)\) for commitment and
eleven more \((T_Z, T_A, T_{gA}, T_{W_1}, ..., T_{W_4}, T_{JV}, T_B, T_1, T_{gR})\) for the proof. The merchant
does eleven multi base exponentiations \((T'_Z, T'_A, T'_{gA}, T'_{W_1}, ..., T'_{W_4}, T'_{JV}, T'_B, T'_1, T'_{gR})\)
for the proof.

The merchant needs to store the same number of signatures as spent coin. Each
signature contains 10 commitment \((S, \pi = (R, T, \phi = (d, A, B, W_1, W_2, W_3, W_4, r_M)))\)
and eleven responses. Fortunately, the merchant does not need to store those values
any more after depositing the coins.

Deposit Phase
The merchant only needs to send the coins to the bank.

The dominant computational cost in bank are multi base exponentiations, file
operations, and inversions. It requires eleven multi base exponentiations to verify one
signature for each coin, and one file searching operation to check the freshness of \(R\)
and \(S\). Also it may involve two inversions and one exponentiation for identification
if a double spender is found. The bank only needs to store three values \((S, R, T)\) for
one coin if she does not worry about proving anything to a third party. Usually, a
bank has to store the coins of all of its users for a long time to check double spenders
and the freshness of the signature. So storing less values helps the bank to reduce
storage cost a lot, and also to reduce the time for searching database.
Chapter 4

Implementation of the Anonymous Payment System

In this chapter, we describe the implementation of our anonymous payment system and the way we tested it. Firstly, we describe how did we construct the system and present the tools that were used. Secondly, we introduce the structure of our code, which includes the data structures, files and functions in our code. After that, we briefly describe how does our code work to simulate the system. This chapter finishes with a description of the way we tested the program.

4.1 Overview

The three entities of our payment system is depicted in Figure 4.1. We have implemented the bank and the merchant in a 2.13GHz Intel [Core(TM)2 6400] PC, running a Linux [semos 2.6.18] operating system. After that, we have further implemented the merchant entity in an 32-bits embedded ARM Gumstix Overo controller. The three entities communicate with each other by socket, which are endpoints of a bidirectional interprocess communication channel. In a practical deployment, a user would withdraw the coins from her account using online banking. So the use of socket best simulates our payment system.

We have developed our implementation using C language. In order to overcome limitations on the size of the variables and make our code portable, we have used GNU [4.3.3] with GMP [5.0.1] library for precision arithmetic operations. This library uses its own integer type mpz_t, larger than 32 bits and are fast for number calculations. Moreover, it can be easily compiled to different platforms such as ARM. Finally, it is designed specially for cryptography and Internet security applications, and thus has many cryptographic functions, like random number generation functions and prime test function, useful for our implementation. For further detail please refer to [24]. Besides the GMP library, we have used the RIPEMD-160 hash function code from [17].

We have defined some global parameters in a header file, such as the secret key length and the group length. Users can easily change these parameters according to
their requirements. Each of the three entities has a configuration file that controls the communications among them. We divided the operation of our payment system into four phases: initialization, withdraw, spend and deposit (the identification is in the deposit phase). Each phase can be executed independently.

4.2 General Structure

In this section we give an overview on the structure of our system. First, we describe the data structures, the major functions and the files that are used for storage. Then, we introduce the organization the code and the operation of the whole system.

4.2.1 Data Structures

We defined the following structures in our system to improve the readability of our code:

**publicparams.** It contains all public parameters that are required to run our anonymous payment system. These include:
- $v_1, ..., g_d$: generators in group $\mathbb{Z}_q$ and group $\mathbb{QR}_n$
- $Z$: public value for verification of the CL signature
- $J_0, J_1$: values used to define the bound of the coin number
- $N$: size of the group $\mathbb{Z}_q$ and the group $\mathbb{QR}_n$
- $l$: bit numbers of wallet size
- $l_{mr}, l_{nr}, l_e, l_n$: size of some secret values

**wallet.** It contains the values necessary to spend coins:
- $(u, s, t)$: secret values for user’s wallet
- $(s_Q, e, d)$: bank’s signature
- $J$: coin numbers left in the wallet
**General Structure**

bound. It contains the variables $w_1, ..., w_4$ for the bounded arithmetic calculation $J_1^2 - (J - J_0)^2 = w_1^2 + ... + w_4^2$

certificate. It contains the values $(S, R, \pi)$ for verifying a certificate.

### 4.2.2 Functions

In this section, we introduce the functions that are used in code.

Firstly, we introduce the functions for basic operations:

**Initialization functions.** We defined some initialization functions to: 1, allocate memory spaces for the mpz_t variables (the mpz_t variables in GMP library must be initialized before setting a value); 2, input some initial parameters like the group length. For example, the ini_PK function allocates memory spaces for the public parameters, and initializes the wallet length and group length to the value defined by user.

**Clear functions.** These functions are the counterparts of the Initialization functions. They are used to free the memory space of the mpz_t variables.

**Store and fetch functions.** We defined different functions for file operations, which include: (1) fetching or storing secret keys and public keys by the user, the bank, or the merchant; (2) fetching or storing the global public parameters; (3) fetching or storing secret values in wallet (4) fetching or storing the spent coins by the merchant. (5) Record spent coins by the bank.

**Send and receive functions.** We defined some functions to do the data transmissions between the three parts. These functions can do following operations: (1) receive data (make sure the received data has the same size as expected); (2) send and receive public parameters; (3) send and receive certificates (4) send and receive responses for the proofs of knowledge.

**initial_*(), withdraw_*(), spend_*(), & deposit_*().** These functions are used by the user, the bank and the merchant to include all of the operations and required functions in initial, withdraw, spend, or deposit phase. * can be u, b or m, means the user, the bank or the merchant.

**main().** Each of the user, the bank and the merchant has a main function that contains all of the operations in the four phases.

Now we introduce the cryptographic functions. Those functions devide our code into several modules and thus make our code easily to be understand:

**prime().** This function is used to generate Sophie-Germain primes $p'$ such that $p = 2p' + 1$ is also a prime, according to the method described in [18].
4. Implementation of the Anonymous Payment System

initial_QR(). This function uses function prime() and function params_QR to generate group $QR_n$ and the parameters in it. The group $QR_n$ is used for Pedersen commitment and CL signatures.

params_QR(). This function is used by initial_QR() to create parameters like generators in group $QR_n$

initial_P(). This function uses function prime() and params_P() to generate group $Z_q$ and the parameters in it. The group $Z_q$ is used to identify users and merchants, and to generate coin serial numbers and security tags.

params_P(). This function is used by initial_P() to create parameters in group $Z_q$

gen_cer(). This function is used by initial_m() or initial_u() to create secret key and public key of merchant or user, and send their public keys to the bank for identification.

Commit() & TowCom(). These two functions are used by the user, the bank or the merchant to generate commitments. The former one generates commitment for four values, while the later one generates commitment for two values.

calcus_si(). This function is used by the user to generate responses $s_i = r_i - c \ast committedvalue$ for the proof of knowledge about a signature in the spend phase.

creat_sig(). This function is used by the bank to create a valid signature, and send it to the user in the withdraw phase.

verify_sig(). This function is a counterpart of the above function. It is used by the user to receive a signature from the bank, and verify the validity of it in the withdraw phase.

calcu_w() & lagrange(). There are two functions generates variables $w_1, ..., w_4$ such as $u = J^2_1 - (J - J_0)^2 = w_1^2 + ... + w_4^2$ (described in [25]). During the spend phase, the user part firstly uses function calcu_w() to generate $w_i$ in the case that $J^2_1 - (J - J_0)^2 = 2(2k+1)$ ($k$ is an odd number), and then uses function lagrange() to deal with the case $J^2_1 - (J - J_0)^2 = 2d(2k + 1) (d \neq 1)$. This method helps to provide the balance property: the user cannot spend more coins than her wallet contains.

fresh_R(). This function is used by the bank to verify the freshness of a coin, and to identify the double spenders in the deposit phase. It searches R and S in the file coins.txt and compares them with the new coin. If found same R (the merchant deposits one coin twice), it rejects the deposit operation; if found same S (the user double spent a coin), it calls the function identify() to compute the public key of this user to indentify her.
identify(). This function is used by the above function fresh_R() to compute the public key of the double spender.

creat_coin(). This function is used by the user to create a valid signature for spending a soin.

verify_coin(). This function is used by the merchant and the bank to verify the validity of the signature the above function creat_coin() offers.

RMDstring(). It calls the RMD160 hash function to compute the hash result of a data string.

conversion(). This function follows the above function RMDstring(). It transfers string type from 'unsigned hex char' outputted by the RMDstring(), to 'ASCII char', which can be transferred to mpz_t variables by function mpz_set_str.

For each of the above functions, it returns 0 when success, and returns a negative value when fail. For function fresh_R(): success returns 0; if the signature (R) is not fresh, returns 2; else if find a same coin (Serial number: S), returns 3; otherwise return a negative number.

4.2.3 Files for storage

In this section, we introduce the files used for storage.

User:

user_PK.txt: Stores the public values in group $\mathbb{Z}_q$ and group $QR_n$
user_yu.txt: Stores user’s public key $y_U = g_U^u$
user_wallet.txt: Stores secret data items in user’s wallet

Bank:

bank_PK.txt: Stores the public values in group $\mathbb{Z}_q$ and group $QR_n$
bank_SK.txt: Stores bank’s secret key $p$
user_keys.txt: Stores different user’s public key $y_u$
coins.txt: Stores the values (R,S,T) for testing freshness of a signature and identifying double spenders

Merchant:

merchant_PK.txt: Stores the public values in group $\mathbb{Z}_q$ and group $QR_n$
merchant_SK.txt: Stores bank’s secret key $m$
coin_cer.txt: Stores the data items for verifying a signature
4. Implementation of the Anonymous Payment System

4.3 Implementation

In this section, we introduce how to install and run our code. Then we describe how does our program work, and what structures, files and functions are included in the initial, withdraw, spend and deposit phases.

4.3.1 Installation and Execution

It is necessary to install the GMP library with a version not lower than 5.0.1, in order to run our program. The GMP library can be downloaded free from [24]. After that, users should open the file "header.h" to configure parameters such as the IP address of the bank and the merchant. Then, run the "Makefile" to compile the code corresponding to the user, the merchant and the bank, by inputting "make user", "make bank", and "make merchant". After compilation, typing ".user", ".bank", and ".merchant" allows to execute each entity of the system. One can input the number "1" to "3" to control which phase to execute in user side: "1" for initialization, "2" for withdraw, and "3" for spend and deposit.

Users can clean the temporary data when finishing execution by typing "make clean".

In the following, we briefly introduce the operations in different phases:

4.3.2 Initial Phase

The operations of this phase depicted in Figure 4.2. In the first step, the bank uses the functions initial_QR() and initial_P() to create the public parameters, which include the groups $QR_n$ and $Z_q$, and the parameters such as generators in them. These two functions first call the function prime() to generate the Sophie-Germain primes, and then use the Sophie-Germain primes to create the groups $Z_q$ and $QR_n$.

In the second step, the functions initial_QR() and initial_P() create the parameters in these two group by using the functions params_QR() and params_P().

In the third step, the bank uses the function initial_b() to send these public parameters to the user and the merchant as well as receive the public keys from them. The function initial_b() calls send_PK() to send the public parameters to the user and the merchant. The user and the merchant receive these public parameters by using the function receive_PK(). The user and the merchant use function gen_cer() to generate their secret keys and public keys, and send the public keys to the bank for identification. Finally, the user, the bank and the merchant store the public parameters by using the function store_PK(), and their secret keys by using the function store_sk() (the user uses store_wallet() to save her secret key). The bank also store the user’s public key by means of store_userkey().
4.3.3 Withdraw Phase

Figure 4.3 illustrates the operations of the withdraw phase. First, the user uses the functions fetch_PK(), fetch_skey() and fetch_wallet() to read out the public parameters, her public key and her secret key. The bank uses fetch_PK() and fetch_skey() to read out the public parameters and her secret key.

Second, the withdraw_u() of user does following things: 1, generates secret values $s', t, r_c \in_R \{0, 1\}^n$; 2, calls Commit() to commit these three values together with her secret key $u$, 3, sends her public key $y_U$ and the committed value $A'$ to the bank.

The withdraw_b() of the bank firstly uses function search_userkey() to make sure the user exists, and then generate a random value $r' \in_R \{0, 1\}^n$ to contribute the randomness of the serial number $s$ of the user’s wallet.

Third, the withdraw_u() computes the new serial number $s = s' + r'$ and new commitment $A = y_g^{s'} A'$, and then calls the function prove_com() to send $c$ and $s_i$ to the bank to prove her knowledge about the secret values $(u, s, t, r_c)$. Withdraw_b() of the bank calls verify_com() to receive the values $c$ and $s_i$, and verify the validity of the commitment.

Next, the withdraw_b() runs creat_sig() to create a valid signature and send this signature to the user. The user calls verify_sig() to receive the signature and validate it.

Finally, the user runs store_wallet() to store the secret values $(u, s, t)$ and the signature into her wallet.

4.3.4 Spend Phase

There are four steps in this phase as depicted in Figure 4.3. In the first step, the user and the merchant uses fetch_PK() to read out the public parameters. The merchant
4. Implementation of the Anonymous Payment System

uses fetch_SK() to fetch her public key. The user uses fetch_wallet() to read the secret values in her wallet.

In the second step, the merchant uses the function spend_m() to send her public key together with a random number \( r_M \) to the user, and calls the hash function RMDstring() and the function conversion() to compute the hash result \( R = \text{Hash}(y_M||r_M) \). The spend_u() of the user receives \( y_M \) and \( r_M \), and calls the functions RMDstring() and conversion() to compute the hash result \( R \) as the merchant.

In the second step, the spend_u() calls the function lagrange() to do lagrange representation computation \( w_1^2 + ... + w_4^2 = J_2 - (J - J_0)^2 \).

In the third step, the user uses the variables \( R \) and \( w_i \) to create a signature and send it to the merchant for spending her coin, by using the function creat_coin(). The merchant runs verify_coin() to receive the signature and validate it.

In the fourth step, if the verification pass, the bank uses store_coin() to store the signature into a file for future deposit. The user reduce her wallet size by one, and store the new value into her wallet by store_wallet().

4.3.5 Deposit Phase

The operation of the deposit phase is shown in Figure 4.5. The first step is to read out the necessary data. The merchant read out the coins by fetch_coin(). The bank read out the public parameters by fetch_PK().

In the second step, the deposit_m() of the merchant calls send_coin() to send the signatures \( (S, T, \pi) \) of the coins to the bank. The deposit_b() of the bank calls read_coin() to receive the signature.

In the third step, the deposit_b() calls fresh_R to verify the freshness of the coins and to check double spenders. If the coin is not fresh (found same R), then the bank
rejected the coin; if the coin is double spent (found same $S$), then the $\text{deposit}_b()$ calls function identify() to compute the public key of the double spender.

In the final step, if there is no double spent coin, then the bank runs $\text{verify}_\text{cer}()$ to validate the coin. If the coin is valid, the bank runs $\text{record}_\text{coin}()$ to record three most important values ($R, S, T$) of the deposited coins.

### 4.4 Test of the Code

We did some basic functional test, such as testing the data transmission, file operations and hash result, to make sure our code works fine. Besides we also tested certain special properties of our scheme as follows.

**Balance.** We makes the coin numbers $J$ in the user’s wallet larger or smaller than the bound ($J_1, J_0$), then user fails to compute $w_i$ with function $\text{calc}_w()$, and thus is unable to provide a valid signature to the merchant. We also tried to compute $w_1^2 + \ldots + w_4^2$ in the bound
while use a different \( J \) (because only \( J \) are used by bank to check double spend), then signature is invalid in this case. In addition, we controls the merchant to deposit one coin twice, then the function \( \text{fresh}_R() \) found this coin is not fresh, and the bank rejected this coin.

**Identify double spender.** We stored several users’ public key in bank’s data base, and then used one user to spend one coin twice. The user can also offer merchant a valid signature without being recognized, but the bank found the secret number \( S \) has already been stored, and identified the user successively by running the function \( \text{identify}() \). This test proves that both the function \( () \) and the storage of user’s public key work find.
Chapter 5

Performance Evaluation

In this chapter, we discuss the performance evaluation of our system. We firstly briefly introduce the tool we used to evaluate the performance. After that, we present a short overview about the execution environment of our system. Next, we show the performance of the three entities in the four phases in both PC and ARM.

5.1 Evaluation Tool

There is no dedicated performance evaluation tool in GCC, so we used the function getrusage. Getrusage is a basic performance evaluation function in Linux to provide measures of the resources and the time used by the current process by using the structure rusage. The structure of rusage is shown in Table 5.1. We mainly used the ru_utime and ru_nivcsw data. ru_utime counts the total amount of time spent executing in user mode, while ru_nivcsw shows the maximum number of physical memory the processes used simultaneously in kilobytes.

5.2 Environment

We have firstly evaluated our system in implemented and evaluated our payment system in a 2.13GHz Intel [Core(TM)2 6400] PC, running a Linux [semos 2.6.18] operating system. In order to better analyze our system, we have tested the performance of the basic cryptography functions and list the running time of them in Table 5.2. From this table, we can say that the exponentiation and prime are the most time costly operations in our system. Inversion function also takes much more time than the other functions.

Then we ported the code of the merchant entity to an ARM Gumstix Overo, with processor ARMv7 rev3. The operation system is Linux 2.6.34. The performance of the basic cryptography functions in our ARM controller is shown in Table 5.3. We can observe that the running time of all functions is increased by a factor approximately 10 times to Table 5.2.
### 5. Performance Evaluation

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ru_utime</td>
<td>struct timeval</td>
<td>user time used</td>
</tr>
<tr>
<td>ru_stime</td>
<td>struct timeval</td>
<td>system time used</td>
</tr>
<tr>
<td>ru_maxrss</td>
<td>long</td>
<td>maximum resident set size</td>
</tr>
<tr>
<td>ru_ixrss</td>
<td>long</td>
<td>integral shared memory size</td>
</tr>
<tr>
<td>ru_idrss</td>
<td>long</td>
<td>integral unshared data size</td>
</tr>
<tr>
<td>ru_isrss</td>
<td>long</td>
<td>integral unshared stack size</td>
</tr>
<tr>
<td>ru_minflt</td>
<td>long</td>
<td>page reclaims</td>
</tr>
<tr>
<td>ru_majflt</td>
<td>long</td>
<td>page faults</td>
</tr>
<tr>
<td>ru_nswap</td>
<td>long</td>
<td>swaps</td>
</tr>
<tr>
<td>ru_inblock</td>
<td>long</td>
<td>block input operations</td>
</tr>
<tr>
<td>ru_oublock</td>
<td>long</td>
<td>block output operations</td>
</tr>
<tr>
<td>ru_msgsnd</td>
<td>long</td>
<td>messages sent</td>
</tr>
<tr>
<td>ru_msgrcv</td>
<td>long</td>
<td>messages received</td>
</tr>
<tr>
<td>ru_nsignals</td>
<td>long</td>
<td>signals received</td>
</tr>
<tr>
<td>ru_nvcsw</td>
<td>long</td>
<td>voluntary context switches</td>
</tr>
<tr>
<td>ru_nivcsw</td>
<td>long</td>
<td>involuntary context switches</td>
</tr>
</tbody>
</table>

Table 5.1: Data structure of rusage

<table>
<thead>
<tr>
<th>Operations</th>
<th>functions</th>
<th>time(1024bits)</th>
<th>time(2048bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random number generation</td>
<td>mpz_urandomb()</td>
<td>0.0003ms</td>
<td>0.0006ms</td>
</tr>
<tr>
<td>Modular exponentiation</td>
<td>mpz_powm()</td>
<td>3.927ms</td>
<td>30.526ms</td>
</tr>
<tr>
<td>Generate next prime</td>
<td>mpz_nextprime()</td>
<td>0.2596s</td>
<td>5.6139s</td>
</tr>
<tr>
<td>Convert variable into hex string</td>
<td>mpz_get_str()</td>
<td>0.0002ms</td>
<td>0.0007ms</td>
</tr>
<tr>
<td>Convert string into variable</td>
<td>mpz_set_str()</td>
<td>0.0002ms</td>
<td>0.0009ms</td>
</tr>
<tr>
<td>Modular inversion</td>
<td>mpz_invert()</td>
<td>0.055ms</td>
<td>0.11ms</td>
</tr>
<tr>
<td>Multiplication</td>
<td>mpz_mul()</td>
<td>0.003ms</td>
<td>0.007ms</td>
</tr>
</tbody>
</table>

Table 5.2: Time of Basic Operation in PC

<table>
<thead>
<tr>
<th>Operations</th>
<th>functions</th>
<th>time(1024bits)</th>
<th>time(2048bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate random number</td>
<td>mpz_urandomb()</td>
<td>0.019ms</td>
<td>0.029ms</td>
</tr>
<tr>
<td>Modular exponentiation</td>
<td>mpz_powm()</td>
<td>38.617ms</td>
<td>263ms</td>
</tr>
<tr>
<td>Generate next prime</td>
<td>mpz_nextprime()</td>
<td>2.7117s</td>
<td>33.821s</td>
</tr>
<tr>
<td>Convert variable into hex string</td>
<td>mpz_get_str()</td>
<td>0.029ms</td>
<td>0.031ms</td>
</tr>
<tr>
<td>Convert string into variable</td>
<td>mpz_set_str()</td>
<td>0.031ms</td>
<td>0.031ms</td>
</tr>
<tr>
<td>Modular inversion</td>
<td>mpz_invert()</td>
<td>0.406ms</td>
<td>0.906ms</td>
</tr>
<tr>
<td>Multiplication</td>
<td>mpz_mul()</td>
<td>0.011ms</td>
<td>0.042ms</td>
</tr>
</tbody>
</table>

Table 5.3: Time of Basic Operation in ARM

38
5.3 Performance

In this section, we firstly describe the performance of the three entities in the initialization, withdraw, spend and deposit phases in PC to provide a comparison to the performance in the ARM. Then we depict the performance of the merchant in the ARM. We tested the time and memory consumption with different group lengths and different secret key lengths, to offer the reader a complete overview of the performance of our system.

5.3.1 Performance in PC

Initialization phase.
The performance of this phase in the PC is shown in Figure 5.4. The user and the merchant only need to do one exponentiation for generating their public key. So their time consumption is extremely low. The most time consuming part of the bank in this phase is the safe prime generation. We used the method introduced in chapter 7 of [18], which is a fast way to generate safe primes. It takes about 20 minutes, 2 hour and 8 hours to generate a safe prime with a length of 512 bits, 1024 bits and 2048 bits. Group $\mathbb{Z}_q$ needs one safe prime with group length, while group $QR_n$ requires two safe primes with half of the group length. So it takes the bank about 3 hours to generate these two groups with the group length of 1024 bits, and about 12 hours with the group length of 2048 bits. This time is acceptable since the initialization phase happens only once.

About the memory consumption in this phase, we used 28 mpz_t variables for the user, 22 mpz_t variables for the bank and 21 mpz_t variables for the merchant. Besides, we used a char buffer with a length of fourth of the group length for each of the three entities. The GMP manual says that mpz_t are small, containing only a couple of sizes, and pointers to allocated data, and it is only allocated the minimum required space. We can see this from Figure 5.4. When we increase the group length, the memory increases about the same time.

Withdraw phase.
The performance of the withdraw phase is shown in Figure 5.5. The major time consumption of the user in this phase comes from modular exponentiations. The user needs to execute the mpz_powm() 12 times to do the withdraw. The Figure 5.2 shows that the 2048 bits exponentiation is about 10 times slower than the 1024 bits exponentiation. That is why the time consumption of the user in group length of 2048 bits is much larger than that in the group length of 1024 bits. However under 1 second is good enough for practical use. The major time consumption of the bank in this phase comes from the modular exponentiations and the prime generation. The bank needs to execute 10 times of mpz_powm() and one time of mpz_nextprime(). Result from the slowly calculation of the function mpz_nextprime(), it takes the bank about 5 seconds in the case of 2048 bits group length. However, considering the fact that the wallet is compact, the users will not withdraw coins too often, and thus this time may be acceptable.

The user in this phase requires at most 36 mpz_t variables simultaneously, and
5. Performance Evaluation

<table>
<thead>
<tr>
<th>group length</th>
<th>key length</th>
<th>time</th>
<th>RAM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>1024bits</td>
<td>160 bits</td>
<td>0.006s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512bits</td>
<td>0.008s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.010s</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>512bits</td>
<td>0.014s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.024s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048bits</td>
<td>0.041s</td>
</tr>
<tr>
<td>Bank</td>
<td>1024bits</td>
<td>160 bits</td>
<td>3hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512bits</td>
<td>3hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>3hours</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>512bits</td>
<td>12h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>12h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048bits</td>
<td>12h</td>
</tr>
<tr>
<td>Merchant</td>
<td>1024bits</td>
<td>160 bits</td>
<td>0.006s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512bits</td>
<td>0.008s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.011s</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>512bits</td>
<td>0.015s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.022s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048bits</td>
<td>0.042s</td>
</tr>
</tbody>
</table>

Table 5.4: Time and Memory Cost in Initialization Phase in PC

<table>
<thead>
<tr>
<th>group length</th>
<th>key length</th>
<th>time</th>
<th>RAM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>1024bits</td>
<td>160 bits</td>
<td>0.080s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512bits</td>
<td>0.096s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.114s</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>512bits</td>
<td>0.541s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.624s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048bits</td>
<td>0.762s</td>
</tr>
<tr>
<td>Bank</td>
<td>1024bits</td>
<td>160 bits</td>
<td>0.342s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512bits</td>
<td>0.383s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>0.500s</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>512bits</td>
<td>4.809s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024bits</td>
<td>5.854s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2048bits</td>
<td>4.903S</td>
</tr>
</tbody>
</table>

Table 5.5: Time and Memory Cost in Withdraw Phase in PC

three char buffers with total length of about two times of the group length. The bank uses at most 32 mpz_t variables simultaneously, and the same buffer size as the user. The memory consumption of both the bank and the user are less than 20 kb, which will not brings problems to the real implementation, even in resource constraint devices.

Spend phase.
The performance of the spend phase is shown in Figure 5.6. The most time consuming part in the user entity is the exponentiation calculation, which is performed 50 times. Besides the user also needs to compute an Lagrange representation. The speed of this computation depends on the wallet size. The bank has to do 41 times of exponentiation. We only count the time for spending one coin. In practical, the user may spend several coins once. In this case the spend time will be considerably increased.

The user uses in total 83 mpz_t variables, and three char buffers with a total length of about three times of the group length. The merchant uses 61 mpz_t variables, and also three char buffers with the same length as the user.

**Deposit phase.**

The performance of this phase is shown in Figure 5.6. The bank requires 41 modular exponentiations, as the merchant does in the spend phase. In practical, the bank needs to store the information of a great amount of users and all the coins they spend. So the file operations for verifying the freshness of the coin and for checking double spending may become the bottleneck. We did not analyze this situation in this thesis. The merchant in this phase does not do any computation, but only sends the signatures to the bank.

For the storage, both the merchant use at most 61 mpz_t variables and char buffers with a total length of about three times of the group length simultaneously. The bank takes 5 more mpz_t variables than the merchant.

**ARM Implementation**

The performance of the merchant is shown in Figure 5.8. Since the code is not changed, the memory cost in ARM does not change. So we only list the time consumption. We did not list the time of the bank and the user neither, since it only takes them less than 10 milliseconds more for communicating with the ARM in each phases.

When the group length is 1024 bits, it takes the merchant a little bit more than
5. Performance Evaluation

<table>
<thead>
<tr>
<th>group length</th>
<th>key length</th>
<th>time</th>
<th>RAM usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank</td>
<td>1024bits</td>
<td>0.126s</td>
<td>11.760KB</td>
</tr>
<tr>
<td></td>
<td>512bits</td>
<td>0.009s</td>
<td>12.288KB</td>
</tr>
<tr>
<td></td>
<td>1024bits</td>
<td>0.012s</td>
<td>13.056KB</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>0.847s</td>
<td>23.808KB</td>
</tr>
<tr>
<td></td>
<td>512bits</td>
<td>0.868s</td>
<td>24.576KB</td>
</tr>
<tr>
<td></td>
<td>1024bits</td>
<td>0.976s</td>
<td>26.112KB</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>0.009s</td>
<td>11.648KB</td>
</tr>
<tr>
<td></td>
<td>1024bits</td>
<td>0.011s</td>
<td>12.416KB</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>0.074s</td>
<td>22.528KB</td>
</tr>
<tr>
<td>Merchant</td>
<td>1024bits</td>
<td>0.016s</td>
<td>11.120KB</td>
</tr>
<tr>
<td></td>
<td>512bits</td>
<td>0.009s</td>
<td>11.648KB</td>
</tr>
<tr>
<td></td>
<td>1024bits</td>
<td>0.011s</td>
<td>12.416KB</td>
</tr>
<tr>
<td></td>
<td>2048bits</td>
<td>0.843s</td>
<td>23.296KB</td>
</tr>
<tr>
<td></td>
<td>1024bits</td>
<td>0.843s</td>
<td>24.832KB</td>
</tr>
</tbody>
</table>

Table 5.7: Time and Memory Cost in Deposit Phase in PC

<table>
<thead>
<tr>
<th>group length</th>
<th>key length</th>
<th>initial</th>
<th>spend</th>
<th>deposit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024bits</td>
<td>160 bits</td>
<td>0.042s</td>
<td>1.015s</td>
<td>0.309s</td>
</tr>
<tr>
<td></td>
<td>512 bits</td>
<td>0.055s</td>
<td>1.082s</td>
<td>0.316s</td>
</tr>
<tr>
<td></td>
<td>1024 bits</td>
<td>0.078s</td>
<td>1.290s</td>
<td>0.343s</td>
</tr>
<tr>
<td>2048bits</td>
<td>512 bits</td>
<td>0.102s</td>
<td>7.169s</td>
<td>0.683s</td>
</tr>
<tr>
<td></td>
<td>1024 bits</td>
<td>0.164s</td>
<td>7.978s</td>
<td>0.686s</td>
</tr>
<tr>
<td></td>
<td>2048 bits</td>
<td>0.281s</td>
<td>9.572s</td>
<td>0.701s</td>
</tr>
</tbody>
</table>

Table 5.8: Running Time in ARM

1 second to accept a coin. This can be a problem if the user wants to spend several coins once. Fortunately, in the electric car charging system, we may expect that the user spend at most 4 or 5 coins once. Nissan says a full charge of their Nissan Leaf electric car will cost $2.75 [2], which support the car to travel about 100 miles. Besides, the King County Council decided to establish charging stations with a maximum of $5 charging fee per use [1]. Under this case, the performance of the merchant in the ARM is acceptable. However, when the group length is 2048 bits, the spend time is too slow for real use. This problem may be solved by taking the improvements introduced in chapter 6.
Chapter 6

Conclusions and Future Work

In this thesis two main issues have been addressed: implementation of an anonymous payment system and the performance evaluation about this system.

For the former, we have presented a possible solution to protect the privacy in electric car recharging systems by using an E-cash mechanism. In addition, we have developed one such system in software to simulate a real implementation. In order to make our program portable and fast, we have used the GMP library.

For the latter, we have evaluated the performance of our system to provide an efficiency analysis about a representative E-cash protocol. We have further evaluated the performance of the recharging pole in an ARM processor, which is one of the best choices for the recharging pole implementation in industry.

There are still many things that could be done in order to improve the efficiency of our anonymous recharging system. One of the possible solution is add a hardware coprocessor to the ARM controller. A hardware coprocessor can be connected to an ARM using memory mapping method described in [32].

Another possible improvement is to implement a divisible E-cash system, which allows the user to spend several coins more efficiently. The first practical divisible e-cash system was proposed by Okamoto [27]. However it does not have the unlinkability property. Canard and Gouget proposed a divisible E-cash system in [12] that is stated to be unlinkable and truly anonymous.

Besides, in practice, the bank usually desires to have the capability to trace all of the coins a double spender has spent. The second scheme of [7] achieves this property, but at the cost of losing its compact property.
Bibliography


Bibliography


Master thesis filing card

*Student:* Author: Chao Li

*Title:* Anonymous Payment Mechanisms for Electric Car Infrastructure

*UDC:* 621.3

*Abstract:*

Electric car is receiving more and more supportance from both companies and governments as one of the most popular choices for the future green car system. However, recharging electric cars in the charging pole may disclose the drivers’ privacy, such as their location privacy. One of the solution to protect the drivers’ privacy is using anonymous E-cash system. A great number of anonymous E-cash protocols have been proposed for the last 25 yeas. Nevertheless, the efficiency analysis of those protocols is hidden behind complex mathematical ideas which does not real much about their practical efficiency. In this thesis, we have implemented an E-cash protocol, which contains the basic payment system requirements, and tested its efficiency. In addition, ARM may be a best choice for the charging pole for its cheap and low power properties. But no one has ever tried to implemented the E-cash protocols in an ARM. In this thesis, we have also implemented the merchant entity (the charging pole) of the E-cash system in an ARM and analyzed its efficiency on it.

Thesis submitted for the degree of Master of Engineering: Electrical Engineering

*Promotors:* Prof. dr. ir. Ingrid Verbauwhede  
Prof. dr. ir. Claudia Diaz  
Prof. dr. ir. Johan Driesen

*Assessors:* Prof. dr. ir. Bart Preneel  
Prof. dr. ir. Geert Deconinck

*Counsellors:* Ir. Alfredo Rial  
Ir. Josep Balasch