Contribution to privacy-preserving cryptographic techniques

Man Ho Allen Au
University of Wollongong
NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.
Contribution to Privacy-Preserving Cryptographic Techniques

A thesis submitted in fulfillment of the requirements for the award of the degree

Doctor of Philosophy

from

UNIVERSITY OF WOLLONGONG

by

Man Ho Allen Au

School of Computer Science and Software Engineering

May 2009
Dedicated to
My mother and my father
Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

________________________________________________________________________
Man Ho Allen Au
May 8, 2009
Abstract

Digital signatures are fundamental cryptographic primitives. They are useful as a stand-alone application and building blocks of complex cryptographic systems. Accumulators are another useful cryptographic primitive which provide a way to combine a set of values into one short value. They are useful in improving efficiency of cryptographic systems. In particular, these two primitives are key components in privacy-preserving cryptographic systems.

In this thesis, we study the use of digital signatures and accumulators in cryptographic applications. We design digital signature schemes and accumulators with different features that are suitable for a wide range of applications. We are interested in privacy-preserving cryptographic applications including anonymous electronic cash systems, anonymous authentication schemes and anonymous credential systems.

We construct three different digital signature schemes, each with distinctive features. We also propose two novel constructions of accumulators. Based on our signature schemes and accumulators, we design two compact electronic cash schemes and a divisible electronic cash scheme. All our schemes are truly anonymous, meaning that privacy of the users is well-protected. We also explore other applications of our newly proposed signatures and accumulators. Specifically, we give a construction of \(k\)-times anonymous authentication schemes and attribute-based anonymous credential systems.

During the course of the development of the thesis, we generalise existing techniques of zero-knowledge proof-of-knowledge protocol of double-discrete logarithms into zero-knowledge proof-of-knowledge protocol of representation of a committed value. Our protocol is compatible with existing zero-knowledge proof-of-knowledge protocols that demonstrate relationship amongst discrete logarithms. We believe that this protocol, together with the newly introduced primitives, are of independent interest.
My experience as a graduate student in the University of Wollongong has been wonderful. I am grateful to my principal supervisor Willy Susilo for this opportunity. Willy has been an excellent supervisor who has offered me directions yet enough freedom for me to explore different areas. I would like to express my gratitude to my co-supervisor Yi Mu for his guidance. Yi has always been nice to me. I still remember the day when he brings me to the supermarket when I first arrived.

My sincere thanks to Dr. Duncan Wong for all his support and advice. The days when I was a research assistant in City University of Hong Kong are so valuable. Duncan and his research group, including Dr. Xiaojian Tian, Tommy Yang, Dennis Liu, Bessie Hu, Jing Chen, etc, are amazing. Special thanks to Qiong Wong with his valuable comments to many of my works. All the discussions are very helpful in the development of this thesis.

I would also like to thank Dr. Lucas Hui and Dr. Siu-Ming Yiu of the Hong Kong University for all the guidance and support. Many thanks to Pierre for her help during my stay as a research assistant in the Hong Kong University. My stay there allows me to build up my foundations before I start my doctorate degree programme. My first research project on e-cash started there and I am thankful to have worked with Sherman Chow.

I am fortunate to be here with a team of interesting people. Discussions are always stimulating and rewarding. I still miss Dr. Qianhong Wu and Dr. Shuhong Wang. The following people must be mentioned. Tsz Hon Yuen, who happened to be my classmate from undergraduate in Hong Kong to graduate student here. Cedric, who brought me around and offered me to stay at his place when I was homeless. I have enjoyed discussions with Xinyi Huang, Wei Wu, Shidi Xu, Mohammad Reza Reyhanitabar, Pairat Thorncharoenri, Liang Lu, Yi Gao, Shams Ud Din Qazi, Siamak Fayyaz Shahandashti, etc. The list goes on and on.

I wish to take this opportunity to give my heartfelt thanks to Prof. Victor Wei,
my supervisor when I was doing a master degree at the Chinese University of Hong Kong. It is Prof. Wei who brings me into the field of cryptography. I would also like to thank my peers at that time, Joseph Liu and Patrick Tsang. They have taught me a lot. Collaborations with them have been rewarding.

This thesis benefited immeasurably from the comments of the panel of knowledgeable thesis examination committee. My heartfelt thanks to Professor Masahiro Mambo and Professor Colin Boyd.

I am fortunate to have knowledgeable, and meticulous panel for my doctoral defense seminar, namely: Prof. Colin Boyd, Prof. Colin Fidge, and Dr. Greg Maitland. This thesis benefited immeasurably from their scrutiny, their relentless precision, and their impeccable taste.

I would like to thank Cindy Chan and Cindy Li for their emotional support. I am grateful to my roommates and friends Chris, Patrick, Emily, Grace, Wendy, Fran, Johnny, Christy, etc. for being with me during my stay in Australia. Finally, special thanks to Jenny Cheung for proof-reading this thesis.

My love and gratitude to my parents Sandra, David and my brother Albert for everything, everything and everything.
The following papers have been published or presented, and contain materials based on the content of this thesis.


I am thankful to have opportunities to collaborate with others in other areas of computer and communications security. The contributions are listed below and they are beyond the scope of this thesis.


# Contents

Abstract v  
Acknowledgement vi  
Publications viii  

1 Introduction 1  

2 Background 5  

2.1 Preliminaries 5  

2.1.1 Miscellaneous Notations 5  
2.1.2 Complexity Theory 6  
2.1.3 Abstract Algebra 7  
2.1.4 Bilinear Map 7  
2.1.5 Number-Theoretic Problems 8  

2.2 Cryptographic Tools 11  

2.2.1 Commitment Schemes 12  
2.2.2 Cryptographic Hash Functions 13  
2.2.3 Digital Signatures 14  
2.2.4 Encryption 15  

2.3 Zero-Knowledge Proof-of-Knowledge 16  

2.3.1 ZKPoK about Discrete Logarithms 18  
2.3.2 ZKPoK about Double Discrete Logarithms 23  

3 Signature Schemes with Efficient Protocols 25  

3.1 Syntax 27  
3.2 BBS+ Signature 30  

3.2.1 Relationship of BBS+ with the BBS Group Signature 30
7 Concluding Remarks

7.1 Summary of Our Contributions

7.1.1 New Construction of CL Signatures

7.1.2 New Construction of Accumulators with Extra Features

7.1.3 New Construction of Electronic Cash with Desirable Properties

7.1.4 Other Applications of the New Primitives

7.2 Open Problems

Bibliography
List of Tables

3.1 Summary of our Proposed CL-Signatures .......................... 53
4.1 Summary of Existing Accumulators ................................. 56
5.1 Time and Space Complexities of Our Scheme and CG07 .......... 121
5.2 Summary of our Proposed Electronic Cash Systems ............. 124
List of Figures

5.1 The life-cycle of an electronic coin .......................... 73
5.2 Construction of A Binary Tree (L=3) .......................... 109
5.3 Spending 4 coins ................................................. 111
5.4 Algorithm ComputeAllNodeKeys ............................... 114
5.5 Deposit Protocol - Computation of All Serial Numbers Associated
   with a Particular Coin ............................................. 116
5.6 RevokeDoubleSpender Algorithm - Computation of a Node Key from
   a Parent Serial Number ........................................... 116
Chapter 1

Introduction

In a broad sense, cryptography is the study and practice of hiding information against malicious adversaries. Today, it is regarded as a branch of both mathematics and computer science. Traditionally, cryptography is concerned solely with message confidentiality. It includes converting information from its comprehensible form into an incomprehensible format so that it is not readable by anyone without the secret knowledge. The process of converting a message into an incomprehensible format is called encryption, while the reverse process, which requires a certain secret knowledge, is called decryption.

Today, the field has expanded to include the study of integrity, authentication, and non-repudiation. Integrity protects a message from unauthorised modification. Authentication assures the originality of a message is from the anticipated sender. Non-repudiation prevents a sender from denying that he has sent a message.

Public Key Cryptography. In their seminal paper “New Directions in Cryptography” [DH76], Diffie and Hellman invented the concept of public key cryptography and showed that secret communication is possible without any prior establishment of a secret key via a secure channel, which is necessary previously. Their novel idea was to use two different keys, a public key for encryption and a private key for decryption. Based on this asymmetry, they further invented the concept of digital signatures which uses the private key to sign a message and generate a signature. The signature can then be verified using the public key. Public key cryptography has a distinct advantage over traditional cryptography: the communicating parties no longer need to exchange the secret key securely and only need to exchange their public key using an authentic mechanism. The concept of public key cryptography inspired many researchers and it has become a new and fast-growing research discipline. Since then, many constructions of digital signature schemes and public
key encryption schemes have been proposed. Based on these primitives, many complex systems such as the electronic cash systems or electronic voting schemes were devised.

**Cryptography For Privacy.** With the advancement of the Internet and the development of information technologies, e-commerce has boomed and opened up many new business opportunities. However, due to the complex and heterogenous nature, the Internet is also a perfect scenario for the adversary. As electronic messages pass through a number of routers on their way from sender to receiver, they might be intercepted, altered or eavesdropped without being noticed. Consequently, Internet today is constantly associated with fraud and privacy abuses.

Security and privacy have long been important issues of prime concerns. Indeed, people are concerned about when, what and how their personal information is being collected and how this information is being used. They also want to know whether the information is being sold or shared with others. If so, with whom and for what purposes. Ultimately, they want to have control over their privacy in the digital age.

Cryptography is a possible solution to the people’s concern. As mentioned earlier, cryptographic primitives such as encryption, authentication, digital signatures, etc. are useful in securing communications over the Internet. User privacy is, on the other hand, often neglected in comparison to security concerns. In fact, government is often concerned with allowing too much privacy as it may possibly be misused.

Anonymity is an important form of privacy protection. This is especially true in case of group-oriented cryptography, where a group of users are involved. In schemes where participation of one or a proper subset of members is required to complete a process, anonymity refers to whether participants are distinguishable from non-participants. Users may prefer perfect anonymity, meaning that it is not possible to distinguish participants from non-participants so as to maintain their privacy in participating the process. We refine levels of anonymity for group-oriented cryptography as follow, from highest level to lowest level (no anonymity).

1. **Full Anonymity.** It means that the identity of the participating user is indistinguishable from the non-participating users to any party. A prominent example is ring signature, formalised in [RST01].

2. **Linkable Anonymity.** Users can participate in the process anonymously but their participation are linked, that is, everybody can tell if the underlying
participant in two separate processes are the same.

3. **Revocable-iff-Linked Anonymity.** Similar to linkable anonymity, users enjoy full anonymity if they only participate once. However, if they participate twice, everybody can reveal their identity. A more general notion is $k$-times Revocable-iff-Linked anonymity, in which user identity is revealed if he participates for more than $k$ times.

4. **Revocable Anonymity.** Basically it means anonymity to everybody except an *Open Authority (OA).* From users’ standpoint, it can be regarded as a lower anonymity level than Revocable-iff-Linked anonymity since the user must trust the OA not to abuse his power in comparison with Revocable-iff-Linked anonymity, where users are anonymous unless they break the condition themselves. Group signatures [ACJT00] is a famous example.

5. **Linkable and Revocable Anonymity.** As its name suggests, users enjoy linkable anonymity towards everybody except OA, where OA can always revoke the anonymity of the user. Systems where users are identified by a pseudonym with an authority knowing the corresponding identity of the user for each pseudonym belong to this category.

6. **Revocable-iff-Linked and Revocable Anonymity.** Similarly, users enjoy Revocable-iff-Linked anonymity to everybody except OA.

7. **No Anonymity.** Identity-based signatures [Sha84] is an example of group-oriented cryptography with no anonymity.

Anonymity level 4-6 can be seen as the counterpart of level 1-3 with the addition of an *Open Authority.* Indeed, it is relatively straightforward to turn systems with anonymity level 1-3 into their counterparts (level 4-6) through the introduction of OA. While users may wish to enjoy full anonymity, some application requirements impose an upper bound on the level of anonymity offered. For instance, there is no anonymity bound for secret leaking. A user who leaks the secret might wish to enjoy full anonymity. On other hand, an e-voting scheme should at most offer linkable anonymity (or revocable-iff-linked anonymity) in order to detect double-votes. In e-voting, linkable anonymity may be acceptable since in the vote-counting stage, the party can disregard those double-vote. People who double-vote thus would not
gain any real benefits. Take electronic cash systems as another example. A double-sender must be caught and thus at most the scheme should offer is revocable-if-linked anonymity.

This thesis concerns with designing cryptographic systems with carefully adjusted level of anonymity suitable for different applications. We would like to strike a balance between user anonymity and accountability.

**Summary of this Thesis.** In this thesis, we consider the uses of signature schemes and accumulators in the context of privacy-preserving cryptographic systems. In this chapter, we have introduced what cryptography is, its history and its significance nowadays. In particular, we discuss a classification of anonymity level offered by cryptographic systems.

In Chapter 2, we introduce our notations and review background materials necessary for understanding the remainder of this thesis. We first give an introduction to the topics of complexity theory, abstract algebra and number theory. We then proceed to recall definitions of several cryptographic primitives, including commitment schemes, hash functions, digital signatures and encryption schemes. Finally we review zero-knowledge proof-of-knowledge protocols.

Next, we give our construction of useful primitives in Chapter 3 and 4. In Chapter 3 we present 3 constructions of digital signature schemes with various features. They are useful in constructing privacy-preserving systems with different requirements. In Chapter 4 we introduce two new features to accumulator, which is another important cryptographic primitive. We provide formal definitions and give new constructions.

In Chapter 5, we discuss electronic cash systems, an example of privacy-preserving cryptographic constructions. We review the background of electronic cash, discuss its requirements and desirable features. Based on the primitives introduced in Chapter 3 and 4, we provide three constructions of electronic cash system, each with its own features.

Finally, we show our newly-introduced primitives can be useful in other scenarios in Chapter 6. In particular, we review anonymous authentications and attribute-based anonymous credential systems.

We conclude this thesis in Chapter 7 with a summary of the thesis together with a discussion of open problems and possible research directions.
Chapter 2

Background

In this chapter, we cover the notations and definitions that will be used throughout this thesis. Background materials on the topic of number theory, cryptographic protocols, bilinear maps, will be presented. The aim of this chapter is to make this thesis self-contained. Readers who are familiar with these topics may skip this chapter. In-depth treatment on theory of cryptography can be found in the books from Menezes et al. [MvOV97] and Goldreich [Gol01, Gol04], while readers interested in elliptic curves and bilinear maps will find the book from Blake et al. [BSS05] or the thesis of Lynn [Lyn08] helpful.

2.1 Preliminaries

2.1.1 Miscellaneous Notations

If $S$ is a set, then $|S|$ is its cardinality. If $S$ is a non-empty set and $a \in_R S$, then $a$ is an element in $S$ drawn uniformly at random from $S$. We denote by $\mathbb{N}$ the set of natural numbers $\{1, 2, \ldots\}$ and by $\mathbb{Z}$ the set of integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$. By $\mathbb{Z}_p$ we denote the set $\{0, \ldots, p-1\}$ and $\mathbb{Z}_p^*$ the set of positive integers smaller than $p$ and relatively prime to $p$. That is,

$$\mathbb{Z}_p^* = \{ k \mid 1 \leq k \leq p \text{ and } \gcd(k, p) = 1 \}.$$  

If $s_1, s_2 \in \{0, 1\}^*$ are strings, $s_1 || s_2 \in \{0, 1\}^*$ is the concatenation of binary strings $s_1$ and $s_2$. By $s_1[i]$ we denote the $i$-th bit of $s_1$, counting from the right-hand end.

Let $f : X_1 \to X_2$ and $g : X_2 \to X_3$ be some functions. We use $g \circ f$ to denote the composition of functions $g$ and $f$. That is, $g \circ f : X_1 \to X_3$ and is defined as $g \circ f : x \mapsto g(f(x))$ for all $x \in X_1$.  

5
2.1.2 Complexity Theory

An algorithm is a computational procedure that takes an input and halts uttering an output. Let $A$ be an algorithm. Throughout this thesis, we use $A(\cdot)$ to denote that $A$ has one input. Similarly, by $A(\cdot, \ldots, \cdot)$ we denote that $A$ has several inputs. By $y \leftarrow A(x)$ we denote that $y$ was the output of algorithm $A$ on input $x$. Efficiency of an algorithm is measured base on the resources needed to produce the output. The term time complexity (resp. space complexity) of an algorithm is used to describe the number of primitive steps (resp. memory) required.

To describe the running time of algorithms, standard asymptotic notation is used. The expression $f(n) = O(g(n))$ means that there exist some positive constant $c$ and a positive integer $n_0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$. Broadly speaking, it means that $g$ is the upper bound of $f$. If $f$ is bounded below by $g$, that is, $g(n) = O((f(n))$, we write $f(n) = \Omega(g(n))$. An algorithm $A$ is a polynomial-time (PPT) if the worst-case running time of $A$ is $O(n^c)$ for some constant $c$, where $n$ is the size of the input. On the other hand, $A$ is an exponential-time algorithm if it has a worst-case running time upper-bounded by $O(c^n)$ for some constant $c > 1$.

We say that a function $\text{negl}(k)$ is a negligible function [Bel02], if for all polynomials $f(k)$, for all sufficiently large $k$, $\text{negl}(k) < 1/f(k)$.

An interactive protocol, or protocol for short, is a pair of algorithms $A$ and $B$ for two communicating players, often denoted as Alice and Bob in cryptography. These players send and receive message from the others and perform some computations according to the specification of the protocol. At some stage the protocol ends and each player obtains an output. A protocol transcript is the sequence of messages exchanged during the entire protocol run.

In cryptography, the security parameter, $\lambda$, is important as negligibility of functions and complexity of algorithms are often parameterised by $\lambda$. For instance, the size of the public parameter, as well as running time of the algorithm should be bounded by $O(\lambda)$. On the other hand, ideally, success probability of an adversary in any cryptographic systems should be negligible in terms of $\lambda$. That is, the success probability of any adversary should be bounded by $O(\text{negl}(\lambda))$. Security parameter is often represented in unary notation as $1^\lambda$ and all algorithms in this thesis receive this value as input.
2.1.3 Abstract Algebra

Let \( G \) be a non-empty set and \( * \) be a binary operation on elements in \( G \) such that \( * : G \times G \rightarrow G \).

**Definition 2.1** A group is a set \( G \) together with a binary operation \( * \) with the following properties:

- If \( a, b \in G \), \( a * b \in G \).
- (Associative.) \( a * (b * c) = (a * b) * c \) for all \( a, b, c \in G \).
- (Existence of identity.) There exists an element in \( G \), called the identity element (denoted as \( 1_G \)), such that for all \( a \in G \), \( 1_G * a = a * 1_G = a \).
- (Existence of inverse.) For any element \( a \in G \), there exists an element called inverse (denoted as \( a^{-1} \) or \( 1/G \)), such that \( a * a^{-1} = a^{-1} * a = 1_G \).

Often, a group is denoted by \( \langle G, * \rangle \) or simply \( G \). A group \( \langle G, * \rangle \) is said to be abelian or commutative if \( a * b = b * a \) for all \( a, b \in G \). The number of elements in \( G \), denoted by \( |G| \), is called the order of \( G \). A group \( G \) is finite if \( |G| \) is finite. We use the multiplicative notation throughout this thesis. Thus, for any positive integer \( k \), \( a^k \) means that \( a \) is multiplied \( m \)-times by itself while \( a^{-k} \) denotes \( (a^{-1})^k \).

A group \( G \) is called cyclic if there exists an element \( g \in G \) such that every element \( a \in G \) can be written in the form of \( g^i \) for some integer \( i \). Such an element \( g \) is called a generator of \( G \) and we use \( \langle g \rangle = G \) to indicate \( g \) generates \( G \). We abuse the notation and use \( |a| \) to denote the order of an element \( a \in G \), that is, the smallest positive integer \( n \) such that \( a^n = 1_G \).

A subset \( G' \subseteq G \) is called a subgroup of \( G \) if \( G' \) is also a group under the operation of \( G \). In this case, \( G' \) contains \( 1_G \) and if \( a, b \in G' \) then \( ab, a^{-1} \in G' \). Moreover, \( |G'| \) divides \( |G| \). An important class of subgroups of \( G \) is the groups generated by an element \( a \) of \( G \), that is, \( \langle a \rangle \).

2.1.4 Bilinear Map

A pairing is a bilinear mapping from a pair of group elements to a group element. Specifically, let \( G_1, G_2, G_T \) be cyclic groups of order \( p \). Let \( g \) and \( g \) be generators of \( G_1 \) and \( G_2 \) respectively. A function \( \hat{e} : G_1 \times G_2 \rightarrow G_T \) is said to be a pairing if it satisfies the following properties:
• (Bilinearity.) \( \hat{e}(u^x, v^y) = \hat{e}(u, v)^{xy} \) for all \( u \in G_1, v \in G_2 \) and \( x, y \in \mathbb{Z}_p \).

• (Non-Degeneracy.) \( \hat{e}(g, g) \neq 1_{G_T} \), where \( 1_{G_T} \) is the identity element in \( G_T \).

• (Efficient Computability.) \( \hat{e}(u, v) \) can be computed efficiently (that is, in polynomial time) for all \( u \in G_1 \) and \( v \in G_2 \).

• (Unique Representation.) All elements in \( G_1, G_2 \) and \( G_T \) have unique binary representation.

Galbraith et al. [GPS06] classify pairing into three types. Specifically, if \( G_2 = G_1 \) (or there exists efficiently computable isomorphism between the two groups), the pairing \( \hat{e} \) is classified as type 1. If \( G_1 \neq G_2 \) and there is no efficiently computable homomorphism from \( G_1 \) to \( G_2 \) but there exists an efficiently computable homomorphism \( \psi : G_2 \rightarrow G_1 \), the pairing \( \hat{e} \) is classified as type 2. Finally, if \( G_1 \neq G_2 \) and there are no efficiently computable homomorphisms between \( G_1 \) and \( G_2 \), the pairing \( \hat{e} \) belongs to type 3. Broadly speaking, type 1 is sometimes known as symmetric pairing while type 2 and type 3 are known as asymmetric pairing.

Typically, \( G_1 \) and \( G_2 \) are subgroups of the group of points on an elliptic curve over a finite field, \( G_T \) will be a subgroup of the multiplicative group of a related finite field and the map \( \hat{e} \) will be derived from either the Weil or Tate pairing on the elliptic curve.

2.1.5 Number-Theoretic Problems

The security of many cryptosystems relies on the intractability of solving some hard problems. By intractable we mean that there is no efficient algorithm that solves the problem. To be more precise, a problem is intractable if there is no polynomial time algorithm that solves it. Most often, it cannot be proven that no such algorithm exists and one has to assume that no such algorithm exists. We present below the problems that are relevant to this thesis. The respective assumptions state that no probabilistic, polynomial time algorithm has non-negligible advantage in security parameter in solving the corresponding problems described below.

The Discrete Logarithm Problem

The discrete logarithm problem (DLP) [BL96] forms the basis in the security of many cryptosystems. We restrict ourselves to DLP in cyclic group in this thesis.
Shoup [Sho97] derived a lower bound on any algorithms that solve DLP without exploiting any special properties of the encoding of the group element. Such algorithms are known as generic algorithms. Specifically, the lower bound is $\Omega(\sqrt{d})$, where $d$ is the largest prime dividing the order of the group. Indeed, such bound is met by the well-known Pollard’s rho algorithm [Pol78] that works in arbitrary groups.

**Definition 2.2** The Discrete Logarithm Problem in $\mathbb{G} = \langle g \rangle$ is defined as follows: On input a tuple $(g, Y) \in \mathbb{G}^2$, output $x$ such that $Y = g^x$.

**Representation Problem**

The representation problem (RP) [Bra93a] is a generalisation of DLP with respect to a number of bases.

**Definition 2.3** The Representation Problem in a cyclic group $\mathbb{G} = \langle g_1 \rangle = \cdots = \langle g_k \rangle$ such that $|\mathbb{G}| = p$, is defined as follows: On input a tuple $(g_1, g_2, \ldots, g_k, Y) \in \mathbb{G}^{k+1}$, output a $k$-tuple $(x_1, \ldots, x_k) \in \mathbb{Z}_p^k$ such that $Y = g_1^{x_1} \cdots g_k^{x_k}$.

The $k$-tuple $(x_1, \ldots, x_k)$ is called a representation of element $Y$. For any element $Y \in \mathbb{G}$, there exists $p^{k-1}$ representations. If the generators $g_1, \ldots, g_k$ are chosen randomly, finding two representations of an element is as hard as DLP.

**Computational Diffie-Hellman Problem**

If we can solve DLP in $\mathbb{G}$, we can also solve the computational Diffie-Hellman (CDH) [DH76] problem although whether the converse is true or not is still an open problem.

**Definition 2.4** The Computational Diffie-Hellman Problem in $\mathbb{G} = \langle g \rangle$ such that $|\mathbb{G}| = p$ is defined as follows: On input a tuple $(g, g^x, g^y) \in \mathbb{G}^3$, output $g^{xy}$.

**Decisional Diffie-Hellman Problem**

The decisional Diffie-Hellman (DDH) problem is the decisional version of the CDH problem. It was first formally introduced in [Bra93a].

**Definition 2.5** The Decisional Diffie-Hellman Problem in $\mathbb{G} = \langle g \rangle$ such that $|\mathbb{G}| = p$ is defined as follows: On input a tuple $(g, g^x, g^y, g^z) \in \mathbb{G}^4$, decide if $g^z = g^{xy}$.
A group equipped with symmetric pairing is an interesting example that separates the difficulties of the CDH problem and the DDH problem. Consider the case of symmetric pairing \( \hat{e} : G_1 \times G_1 \to G_T \). The DDH problem is easy in \( G_1 \) even though the CDH problem is conjectured to be hard. For given \((g, g^x, g^y, g^z) \in G_1^4\), one can test if \( \hat{e}(g^x, g^y) = \hat{e}(g^z, g) \). Groups with such property are called gap groups.

For asymmetric pairing \( \hat{e} : G_1 \times G_2 \to G_T \), the assumption that the DDH problem is hard in \( G_1 \) is called the eXternal Diffie-Hellman (XDH) assumption while the assumption that the DDH problems in both \( G_1 \) and \( G_2 \) are hard is called the Symmetric External Diffie-Hellman (SXDH) assumption [ACdM05]. The former assumption implies that there is no efficient computable isomorphism from \( G_1 \) to \( G_2 \) while the later implies that there is no efficient computable isomorphism in both directions. Those assumptions are shown to be false in groups of points for supersingular curves while it might hold in groups of points over MNT curves. See [BBS04] for a more throughout discussion. Nonetheless, they are viewed as strong assumptions.

\textit{q-Strong Diffie-Hellman Problem}

Boneh and Boyen [BB04] proposed a short signature scheme with the introduction of the \( q \)-strong Diffie-Hellman (\( q \)-SDH) problem. They derived a lower bound on any generic algorithms that solve the \( q \)-SDH problem, in the sense of Shoup [Sho97]. However, Cheon [Che06] pointed out that if \( d \) is a maximum of the largest divisor of \( p - 1 \) (resp. \( p + 1 \)) not exceeding \( q \) or \( \sqrt{p} \) (resp. \( q/2 \) or \( p^{1/3}(\log p)^{2/3} \))), the \( q \)-SDH problem can be solved in \( O(\log p(\sqrt{p/d} + \sqrt{d})) \) (resp. \( O(\log p(\sqrt{p/d} + d)) \)) steps. Cheon suggested that to achieve comparable security with systems related to DLP, systems based on the \( q \)-SDH (or related) problems make use of bilinear map and there is no known efficient algorithm to generate groups of the form recommended by Cheon.

\textbf{Definition 2.6} The \textit{q-Strong Diffie-Hellman problem} in \( \mathbb{G} = \langle g \rangle \) such that \(|\mathbb{G}| = p\), is defined as follows: On input a \((q + 1)\)-tuple \((g, g^x, g^{x^2}, \ldots, g^{x^q}) \in \mathbb{G}^{q+1}\), output a pair \((A, c) \in (\mathbb{G} \times \mathbb{Z}_p)\) such that \( A^{(x+c)} = g \).

The corresponding version of \( q \)-SDH problem in a bilinear group equipped with pairing \( \hat{e} : G_1 \times G_2 \to G_T \) with isomorphism \( \psi : G_2 \to G_1 \) is defined in [BBS04].
and is re-stated below.

**Definition 2.7** The $q$-Strong Diffie-Hellman ($q$-SDH) problem in a bilinear group pair $(G_1, G_2)$ with trace map $\psi$ is defined as follow: On input a $(q + 2)$-tuple $(g, g_1, g^{x_1}, \ldots, g^{x_q}) \in G_1 \times G_2^{q+1}$ such that there exists a trace map $\psi$ from $G_2$ to $G_1$ with $g = \psi(g)$, output a pair $(g^{\frac{1}{x_1}}, c) \in (G_1 \times \mathbb{Z}_p)$.

**y-Decisional Diffie-Hellman Inversion Problem**

The $y$-decisional Diffie-Hellman Inversion ($y$-DDHI) problem \cite{DY05, CHL05} is a decisional problem related to the $q$-SDH problem. It should be noted that Cheon’s analysis on the $q$-SDH problem is also applicable to the $y$-DDHI problem.

**Definition 2.8** The $y$-Decisional Diffie-Hellman Inversion problem in $G = \langle g \rangle$ such that $|G| = p$, is defined as follow: On input a $(y + 2)$-tuple $(g, g^x, g^{x_2}, \ldots, g^{x_y}, g^z) \in G^{y+2}$, decide if $z = 1/x$.

**LRSW Problem**

Lysyanskaya et al. \cite{LRSW99} introduced the oracle-based LRSW problem to construct a credential system. They justified its hardness by deriving a lower bound on any generic algorithms that solve the problem.

**Definition 2.9** Let $G = \langle g \rangle$, such that $|G| = p$, be a cyclic group. Let $u = g^x, v = g^y$ for some $x, y \in \mathbb{Z}_p$. Define $O_{u,v}(\cdot)$ as an oracle such that on input a value $m \in \mathbb{Z}_p$, output $(a, a^y, a^{x+mx})$ for a randomly chosen $a \in G$. The LRSW problem is defined as on input $g, u, v$, and the oracle $O_{u,v}(\cdot)$, output $(m, a, b, c) \in (\mathbb{Z}_p, G^3)$ such that $(m \neq 0 \land a \in G \land b = a^y \land c = a^{x+mx})$ and $m$ has not been input to $O_{u,v}(\cdot)$.

**2.2 Cryptographic Tools**

In their seminal paper “New Directions in Cryptography” \cite{DH76}, Diffie and Hellman invented the notion of public-key cryptography in which every player in the system is equipped with two keys, namely, public key and private key. The public key represents the identity of a user and is publicly available. The user holds the corresponding private key which is only available to himself. In the case of public key encryption, user Alice encrypts a message $m$ under the public key of Bob to
produce a ciphertext $\text{ctxt}$. Only Bob, who is in possession of the corresponding 
private key, can decrypt $\text{ctxt}$ and gets back the original message $m$. We present a 
brief review on the cryptographic tools needed in this thesis as follows.

2.2.1 Commitment Schemes

A commitment scheme is a protocol between two parties, namely, committer Alice 
and receiver Bob. It consists of two stages: the Commit stage and the Reveal stage. 
In the Commit stage, Alice receives a value $x$ as input, which is revealed to Bob 
at the Reveal stage. A commitment scheme is secure if at the end of the Commit 
stage, Bob cannot compute anything about the committed value while at the Reveal 
stage, Alice can only reveal one value, that is $x$.

In this thesis, we restrict ourselves to non-interactive commitment scheme al-
though in general a commitment scheme can be interactive. A commitment scheme 
is a tuple $(\text{Gen}, \text{Commit})$ such that on input security parameter $1^\lambda$, $\text{Gen}$ outputs 
some parameters $\text{param}$ for the commitment scheme. $\text{Commit}$ is a determinist al-
algorithm that on input $\text{param}$, a value $x$ and a random number $r$, outputs $C = 
\text{Commit}(\text{param}, x; r)$. $C$ is called the commitment of $x$ and $r$ is sometimes called an 
opening. A commitment scheme should be hiding and binding. Informally speaking, a commitment scheme is unconditionally hiding if given $C$, no adversary, even 
a computationally unbounded one, can tell anything about the value committed. It 
is computationally binding if no polynomial-time adversary can open a commitment 
in two different ways. Below we give a more formal definition.

**Definition 2.10** A commitment scheme $(\text{Gen}, \text{Commit})$ is secure if holding the fol-
lowing two properties:

1. (Perfect Hiding.) We employ the definition of semantic security from [GM84]. 
The adversary chooses $x_0$ and $x_1$ and receives a commitment to a random one 
of them. The adversary cannot tell, better than random guessing, a commit-
tment to which one of them it received. Specifically, for all algorithm $\mathcal{A}$ (even 
computationally unbounded one), we require that

$$
\Pr \left[ \begin{array}{c}
\text{param } \leftarrow \text{Gen}(1^\lambda); (x_0, x_1) \leftarrow \mathcal{A}(\text{param}); \\
 b \in_R \{0, 1\}; r \in_R \{0, 1\}^\lambda; \\
 C = \text{Commit}(\text{param}, x_b; r); b' \leftarrow \mathcal{A}(C); \\
 b' = b
\end{array} \right] \leq 1/2 + \text{negl}(\lambda)
$$
2. (Binding.) No PPT adversary $A$ can open a commitment in two different ways. Specifically,

$$
Pr \left[ \begin{array}{l}
\text{param} \leftarrow \text{Gen}(1^\lambda); (x_0, x_1, r_0, r_1) \leftarrow A(\text{param}) : \\
x_0 \neq x_1 \land \text{Commit(\text{param}, x_0; r_0)} = \text{Commit(\text{param}, x_1; r_1)}
\end{array} \right] = \text{negl}(\lambda)
$$

We describe below a commonly used commitment scheme from Pedersen [Ped91] in our setting.

**Gen.** Let $G$ be a cyclic group of prime order $p$. The $\text{Gen}$ algorithm randomly chooses two generators $g, h \in_R G$ and outputs $\text{param} = (G, g, h)$.

**Commit.** On input a value $x \in \mathbb{Z}_p$, the algorithm randomly chooses $r \in \mathbb{Z}_p$, computes and outputs commitment $C = g^x h^r$.

**Reveal.** To reveal the commitment $C$, the committer outputs $(x, r)$. Everyone can test if $C = g^x h^r$.

The scheme is unconditionally hiding and computationally binding if DLP is hard in $G$. One can easily extend the scheme to allow commitment of a block of values, say, $x_1, \ldots, x_k$ by setting the commitment $C = g^{x_1} \cdots g^{x_k} h^r$ with additional random generators $g_1, \ldots, g_k$ of $G$.

Trapdoor commitment is a special class of commitment schemes useful in many cryptographic systems. In addition to $\text{param}$, $\text{Gen}$ also outputs some trapdoor information, denoted as $\text{tk}$. The knowledge of $\text{tk}$ allows one to open a commitment in any way. In practice, $\text{Gen}$ is carried out so that the committer will not know $\text{tk}$ and thus a trapdoor commitment can be a secure commitment scheme.

### 2.2.2 Cryptographic Hash Functions

A hash function, $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$, is an efficient algorithm that maps an input of an arbitrary-length bit-string $x$ to an output $H(x)$ of fixed length $\lambda$ [MvOV97, Chapter 9]. In other words, hash functions allow one to reduce messages of arbitrary length to fixed length values, and are often employed by cryptographic schemes. Naturally, we require hash function to be efficiently computable. For cryptographic purpose in our thesis, we require that hash functions have to be collision resistant [Dam87]. That is, it is computationally hard to find a pair $x_0, x_1$ with $x_0 \neq x_1$, such that $H(x_0) = H(x_1)$. 
The random oracle model (ROM), first introduced by Fiat and Shamir \cite{FS86} and formalised by Bellare and Rogaway \cite{BR93}, represents an idealised view of hash functions. In the random oracle model, hash functions are treated as oracles which answer queries with random strings, except that the same query always yields the same answer. Some critics argue that in practice those deterministic polynomial-time hash functions do not resemble random oracles. For instance, Canetti, Goldreich and Halevi \cite{CGH04} constructed examples of cryptographic schemes that are secure in the random oracle model but are insecure under any instantiations with real-world implementation of hash function. While some argue that those examples were artificially constructed and practical system would not be designed that way, some believe that a proof in the random oracle model is more of a heuristic proof than an actual one. Nonetheless, the model is still widely accepted in the cryptographic community although recently researchers are more interested in designing systems that are provably secure without relying on the random oracle model.

2.2.3 Digital Signatures

Digital signature schemes, invented in \cite{DH76} and further formalised in \cite{GMR88}, are a fundamental cryptographic primitive as well as an important building block of many cryptographic applications. They are the digital counterpart of paper-based signatures.

A digital signature scheme consists of three algorithms, namely, KeyGen, Sign and Verify for generating keys, signing and verifying signatures, respectively.

The standard notion of security for a digital signature scheme is called existential unforgeability under chosen message attack (EU-CMA) \cite{GMR88}, which can be defined formally by the following game between a challenger $C$ and a PPT adversary $A$.

**Definition 2.11 (Game EU-CMA)** Setup: A challenger $C$ runs algorithm KeyGen to obtain a public key $PK$ and a private key $SK$. $C$ gives $PK$ to $A$.

Queries: $A$ requests signatures on $q$ messages, $M_1, \ldots, M_q$, adaptively. $C$ responds to each query with a signature $\sigma_i \leftarrow \text{Sign}_{SK}(M_i)$.

Output: $A$ outputs a pair $(M^*, \sigma^*)$. $A$ wins the game if $M^* \notin \{M_1, \ldots, M_q\}$ and valid $\leftarrow \text{Verify}_{PK}(M^*, \sigma^*)$. 
A digital signature scheme is secure if no PPT adversary can win in Game EU-CMA with non-negligible probability.

For various applications, a stronger notion of security called strong unforgeability [ADR02] is needed. Roughly speaking, a signature scheme with strong unforgeability means that an adversary cannot produce a new signature even for a previously signed message.

Formally, GAME S-EU-CMA can be defined to capture strong unforgeability. GAME S-EU-CMA is the same as GAME EU-CMA except the output stage is replaced by the following.

Output: $A$ outputs a pair $(M^*, \sigma^*)$. $A$ wins the game if $(M^*, \sigma^*) \neq (M_i, \sigma_i)$ for all $i = 1$ to $q$ and valid $\leftarrow$ $\text{Verify}_{PK}(M^*, \sigma^*)$.

We say a signature scheme is S-EU-CMA if no PPT adversary can win in GAME S-EU-CMA with non-negligible probability. Note that any signature scheme that is S-EU-CMA is EU-CMA.

We describe below the short signature from Boneh and Boyen [BB04], which is S-EU-CMA under the $q$-SDH assumption. Hereafter, we shall refer to this scheme as BB-signature.

**KeyGen.** Let $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ be a bilinear map as discussed. Note that $|\mathbb{G}_1| = |\mathbb{G}_T| = p$ for some prime $p$. Let $g$ be a generator of $\mathbb{G}_1$. Let $x, y \in \mathbb{Z}_p^*$ and $u = g^x$ and $v = g^y$. The secret key $SK$ is $(x, y)$ while the public key $PK$ is $(g, u, v)$.

**Sign.** Given $SK = (x, y)$ and message $m \in \mathbb{Z}_p^*$, pick a random $r \in \mathbb{Z}_p^*$ and compute $A = g^{x+m+yr}$. The term $x + m + yr$ is computed modulo $p$. In case $x + m + yr = 0 \mod p$, choose another $r$. The signature $\sigma$ on $m$ is $(A, r)$.

**Verify.** Given a public key $(g, u, v)$, a message $m \in \mathbb{Z}_p^*$, a signature $\sigma = (A, r)$, verify that

$$\tilde{e}(A, ug^m v^r) = \tilde{e}(g, g)$$

If the equality holds, output valid. Otherwise, output invalid.

### 2.2.4 Encryption

Encryption is another fundamental primitive in cryptography. The goal of a secure encryption scheme is to preserve the privacy of data. It allows one party to
send a message to another such that the contents of the messages remain hidden even if someone intercepts the communication. Encryption schemes can be divided into two categories, namely, private key or symmetric encryption and public key or asymmetric encryption. In a private key scheme, the same key (or keys that can be derived from each other easily) is used for both encryption and decryption. This implies the two communicating parties need to exchange a private key prior to the communication. In a public key encryption scheme, the public key can be published while the secret key used for decryption must be kept secret. Thus, in contrast to symmetric encryption where the key must be exchanged securely, the public key in asymmetric encryption needs to be exchanged only authentically, which is a much weaker requirement.

While symmetric encryption schemes were already known to Julius Caesar, public key encryption was only invented in 1977 by Diffie and Hellman in their seminal paper “New Directions in Cryptography” [DH76]. The standard definition for the security of encryption schemes was first formalised in [GM84].

It is now known that public-key methodologies were first discovered by the Communications Electronics Security Group (CESG) in the early 1970s. CESG was a branch of the British Government Communications Headquarters (GCHQ) whose function is to ensure information security for the British Government.

According to the information released by CESG in 1997\footnote{Available at http://www.cesg.gov.uk/publications/index.htm#nsecret}, J. Ellis put forth the idea of public key cryptography under the name of non-secret encryption (NSE) in the year 1970. In 1973, C. Cocks described an encryption scheme which is virtually the same as that of the RSA cryptosystem. In 1974, Williamson described what we now call the Diffie-Hellman key exchange.

\section{2.3 Zero-Knowledge Proof-of-Knowledge}

In a zero-knowledge proof protocol, introduced by [GMR85] in their seminal paper “The Knowledge Complexity of Interactive Proof-Systems”, a prover convinces a verifier that a statement is true, while the verifier learns nothing except the validity of the assertion. A proof-of-knowledge [BG92] is a protocol such that the verifier is convinced that the prover knows a certain quantity $w$ satisfying some kinds of relation $R$ with respect to a commonly known string $x$. That is, the prover convinces
the verifier that he knows some $w$ such that $(w, x) \in R$. If it can be done in such a way that the verifier learns nothing about $w$, this protocol is called a zero-knowledge proof-of-knowledge (ZKPoK) protocol.

In this thesis, we restrict ourselves to a special class of ZKPoK protocol called $\Sigma$-protocol which is defined below.

**Definition 2.12** A $\Sigma$-protocol for a binary relation $R$ is a 3-round ZKPoK protocol between two parties, namely, a prover $P$ and a verifier $V$. For every input $(w, x) \in R$ to $P$ and $x$ to $V$, the first round of the protocol consists of $P$ sending a commitment $t$ to $V$. $V$ then replies with a challenge $c$ in the second round and $P$ concludes by sending a response $z$ in the last round. At the end of the protocol, $V$ outputs accept or reject. We say a protocol transcript $(t, c, z)$ is valid if the output of an honest verifier $V$ is accept. A $\Sigma$-protocol has to satisfy the following two properties:

- *(Special Soundness.)* A cheating prover can at most answer one of the many possible challenges. Specifically, there exists an efficient algorithm $KE$, called knowledge extractor, that on input $x$, a pair of valid transcripts $(t, c, z)$ and $(t, c', z')$ with $c \neq c'$, outputs $w$ such that $(w, x) \in R$.

- *(Honest-Verifier Zero-Knowledge (HVZK).)* There exists an efficient algorithm $KS$, called zero-knowledge simulator, that on input $x$ and a challenge $c$, outputs a pair $(t, z)$ such that $(t, c, z)$ is a valid transcript having the same distribution as a real protocol transcript resulted from the interaction between a prover $P$ with input $(w, x) \in R$ and an honest verifier $V$.

$\Sigma$-protocols can be transformed to 4-move perfect zero-knowledge ZKPoK protocol [CDM00]. They can also be transformed to 3-move concurrent zero-knowledge protocol in the auxiliary string model using trapdoor commitment schemes [Dam00].

The remaining part of this section is devoted to various known techniques about efficient ZKPoK on relationships amongst discrete logarithms, which is an important basis of which this thesis is built on. A far better treatment on this kind of proof systems can be found in Camenisch [Cam98].

**Signature of Knowledge**

Any $\Sigma$-protocol can be turned into non-interactive form, called signature of knowledge [CS97], by setting the challenge to the hash value of the commitment together
with the message to be signed [FS86]. Pointcheval and Stern [PS96] showed that any
signature scheme obtained this way is EU-CMA in the random oracle model [BR93]
using Fiat-Shamir heuristic. As an example, the Schnorr signature scheme [Sch91]
is a signature of knowledge of discrete logarithm.

2.3.1 ZKPoK about Discrete Logarithms

We employ the following algebraic setting. Let $G$ be a finite cyclic group such that
$|G| = p$ for some prime $p$. Let $g, g_0, g_1, \ldots, g_k \in R \ G$ be generators of $G$ such that
DLP in $G$ is hard. Note that the generators are chosen randomly so that relative
discrete logarithms amongst any of them are unknown. In practice, $G$ is often chosen
as a cyclic subgroup of $\mathbb{Z}_p^*$ generated by some element of order $p$ such that prime
$p' = rp + 1$ for some integer $r$, or group of points of an elliptic curve over some finite
field. Let $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ by a cryptographic hash function for some security
parameter $\lambda$.

Let $G_q = \langle h \rangle$, with $|G_q| = q$ which is also a prime, be a subgroup of $\mathbb{Z}_p^*$. This
can be defined if $q$ is a prime factor of $p - 1$ and $G_q$ can be generated by an element
in $\mathbb{Z}_p^*$ with order $q$.

Knowledge of Discrete Logarithm

Following the notation introduced by Camenisch and Stadler [CS97], we use $PK\{(x) : Y = g^x\}$ to denote a $\Sigma$-Protocol that allows prover $P$ to prove the knowledge of
$x \in \mathbb{Z}_p$ such that $Y = g^x$ for some $Y \in G$ to verifier $V$. Instantiation of the protocol
(denoted as $PK_1$), introduced by Schnorr in his Identification Scheme [Sch91], is
reviewed below.

(Commitment.) $P$ randomly chooses $\rho \in R \ \mathbb{Z}_p$, computes $T = g^\rho$ and sends $T$ to $V$.

(Challenge.) $V$ randomly chooses $c \in R \ \{0, 1\}^\lambda$ and sends $c$ back to $P$.

(Response.) $P$ computes $z = \rho - cx \mod p$ and returns $z$ to $V$.

(Verify.) $V$ outputs accept if and only if $T = Y^c g^z$.

As an example, we show the corresponding signature of knowledge, which is
taken from [Cam98, Section 3.2].
2.3. Zero-Knowledge Proof-of-Knowledge

**Definition 2.13** A pair \((c, z) \in \{0, 1\}^\lambda \times \mathbb{Z}_p\) satisfying:

\[
c = H(g \parallel Y \parallel Y^c g^z \parallel m)
\]

is a signature of knowledge of the discrete logarithm of \(Y\) to base \(g\) for message \(m \in \{0, 1\}^*\) and is denoted as

\[
\text{SPK}_1\{(x) : Y = g^x\}(m).
\]

\(\text{SPK}_1\) can be computed by \(\mathcal{P}\) by replacing the challenge phase with the hash of the commitment in \(\text{PK}_1\). Specifically,

(Commitment.) \(\mathcal{P}\) randomly chooses \(\rho \in R\mathbb{Z}_p\), computes \(T = g^\rho\).

(Challenge.) \(\mathcal{P}\) computes \(c = H(g \parallel Y \parallel T \parallel m)\).

(Response.) \(\mathcal{P}\) computes \(z = \rho - cx \mod p\) and outputs \((c, z)\) as \(\text{SPK}_1\).

(Verify.) Anyone can verify \(\text{SPK}_1\) by testing if \(c \overset{?}{=} H(g \parallel Y \parallel Y^c g^z \parallel m)\).

Note that if the context is clear, sometimes we will simply set the challenge \(c\) to be the hash of the commitment \(T\) and the message \(m\) to be signed.

Below we review several existing \(\text{SPK}\) related to discrete logarithms that are useful in this thesis. Note that for each \(\text{SPK}\) presented, there is a corresponding \(\Sigma\)-protocol.

**Knowledge of Representation of An Element**

The next \(\text{SPK}\) we are going to review is the representation of an element:

\[
\text{SPK}_2\{(x_1, \ldots, x_k) : Y = g_1^{x_1} \cdots g_k^{x_k}\}(m)
\]

(Commitment.) \(\mathcal{P}\) randomly chooses \(\rho_1, \ldots, \rho_k \in R\mathbb{Z}_p\), computes \(T = g_1^{\rho_1} \cdots g_k^{\rho_k}\).

(Challenge.) \(\mathcal{P}\) computes \(c = H(T \parallel m)\).

(Response.) \(\mathcal{P}\) computes, for \(i = 1\) to \(k\), \(z_i = \rho_i - cx_i \mod p\) and outputs \((c, z_1, \ldots, z_k)\) as \(\text{SPK}_2\).

(Verify.) Anyone can verify \(\text{SPK}_2\) by testing if \(c \overset{?}{=} H(Y^c g_1^{z_1} \cdots g_k^{z_k} \parallel m)\).
Knowledge of Equality of Discrete Logarithms

The next SPK we are going to review is the equality of discrete logarithm:

\[ \text{SPK}_3\{(x) : Y_1 = g_1^x \land Y_2 = g_2^x\}(m) \]

(Commitment.) \( \mathcal{P} \) randomly chooses \( \rho \in_R \mathbb{Z}_p \), computes \( T_1 = g_1^{\rho_1} \) and \( T_2 = g_2^{\rho_2} \).

(Challenge.) \( \mathcal{P} \) computes \( c = H(T_1||T_2||m) \).

(Response.) \( \mathcal{P} \) computes \( z = \rho - cx \mod p \) and outputs \((c, z)\) as \( \text{SPK}_3 \).

(Verify.) Anyone can verify \( \text{SPK}_3 \) by testing if \( c \overset{?}{=} H(Y_1^c g_1^z || Y_2^c g_2^z || m) \).

Statements about Knowledge of Discrete Logarithms

Camenisch [Cam98] demonstrates how the above proof systems can be combined together. They are useful in demonstrating relationship amongst the discrete logarithms. As an example, let \( Y_1 = g_1^{x_1} g_2^{x_2} \), \( Y_2 = g_3^{x_3} g_4^{x_4} g_5^{x_5} \). A prover \( \mathcal{P} \) wishes to demonstrate the knowledge of a representation of \( Y_1 \) to base \( g_1, g_2 \) and a representation of \( Y_2 \) to base \( g_3, g_4, g_5 \), together with the fact that the first index in the representation of \( Y_1 \) is equal to the second index in the representation of \( Y_2 \).

\[ \text{SPK}_4\{(x_1, x_2, x_3, x_4) : Y_1 = g_1^{x_1} g_2^{x_2} \land Y_2 = g_3^{x_3} g_4^{x_4} g_5^{x_5}\}(m) \]

(Commitment.) \( \mathcal{P} \) randomly chooses \( \rho_1, \rho_2, \rho_3, \rho_4 \in_R \mathbb{Z}_p \), computes \( T_1 = g_1^{\rho_1} g_2^{\rho_2} \) and \( T_2 = g_3^{\rho_3} g_4^{\rho_4} g_5^{\rho_5} \).

(Challenge.) \( \mathcal{P} \) computes \( c = H(T_1||T_2||m) \).

(Response.) \( \mathcal{P} \) computes \( z_1 = \rho_1 - cx_1 \mod p \), \( z_2 = \rho_2 - cx_2 \mod p \), \( z_3 = \rho_3 - cx_3 \mod p \), \( z_4 = \rho_4 - cx_4 \mod p \) and outputs \((c, z_1, z_2, z_3, z_4)\) as \( \text{SPK}_4 \).

(Verify.) Anyone can verify \( \text{SPK}_4 \) by testing if \( c \overset{?}{=} H(Y_1^c g_1^{z_1} g_2^{z_2} || Y_2^c g_3^{z_3} g_4^{z_4} g_5^{z_5} || m) \).

Consider another example. Let \( Y_1 = g^{x_1} \), \( Y_2 = g^{x_2} \) and \( Y_3 = g^{x_3} \). One could also use a combination of the above techniques to demonstrate that \( x_3 = x_1 x_2 \mod p \). The trick is to prove that, in addition to the knowledge of discrete logarithm of \( Y_1 \), \( Y_2 \) and \( Y_3 \) to base \( g \), the prover also proves the knowledge of \( Y_3 \) to the base \( Y_2 \) and that this discrete logarithm is equal to that of \( Y_1 \) to base \( g \). Specifically,

\[ \text{SPK}_5\{(x_1, x_2, x_3) : Y_1 = g^{x_1} \land Y_2 = g^{x_2} \land Y_3 = g^{x_3} \land Y_3 = Y_2^{x_1}\}(m) \]
2.3. Zero-Knowledge Proof-of-Knowledge

(Commitment.) \( \mathcal{P} \) randomly chooses \( \rho_1, \rho_2, \rho_3 \in_R \mathbb{Z}_p \), computes \( T_1 = g^{\rho_1}, T_2 = g^{\rho_2}, T_3 = g^{\rho_3} \) and \( T_4 = Y_2^{\rho_1} \).

(Challenge.) \( \mathcal{P} \) computes \( c = H(T_1||T_2||T_3||T_4||m) \).

(Response.) \( \mathcal{P} \) computes \( z_1 = \rho_1 - cx_1 \mod p, z_2 = \rho_2 - cx_2 \mod p, z_3 = \rho_3 - cx_3 \mod p \) and outputs \( c, z_1, z_2, z_3 \) as \( \text{SPK}_5 \).

(Verify.) Anyone can verify \( \text{SPK}_5 \) by testing if \( c = H(Y_1^z Y_2^{-z_1} || Y_1^z Y_2^{-z_2} || Y_1^z Y_2^{-z_3} || Y_1^z Y_2^{-z_1} || m) \).

Inequality of Discrete Logarithm

Camenisch and Shoup [CS03] constructed an efficient proof for inequality of discrete logarithms. Let \( Y_1 = g_1^x \) and the prover wishes to prove that \( Y_2 \) is not equal to \( g_2^x \). Of course it can be done trivially by computing a value \( Y = g_2^x \), conducting a proof-of-correctness of \( Y \) with respect to \( Y_1 \) (that is, discrete logarithm of \( Y_1 \) to base \( g_1 \) is the same as the discrete logarithm of \( Y \) to base \( g_2 \)). If \( Y \) is not equal to \( Y_2 \), it implies \( Y_2 \neq g_2^x \). This however, is not exactly zero-knowledge as it leaks some information about \( x \), that is, \( g_2^x \).

\[
\text{SPK}_6\{(x) : Y_1 = g_1^x \land Y_2 \neq g_2^x\}(m)
\]

The actual construction of \( \text{SPK}_6 \) is a bit more involved. Let \( \mathfrak{g}_1, \mathfrak{g}_2 \) be some additional random generators of \( \mathbb{G} \). \( \mathcal{P} \) first randomly generates \( r_1, r_2 \in_R \mathbb{Z}_p \), computes \( \mathfrak{A}_1 = \mathfrak{g}_1^{r_1}, \mathfrak{g}_2^{r_2} \) and \( \mathfrak{A}_2 = (Y_2/g_2^x)^{r_1} \). The prover then conducts the following signature of knowledge, denoted as \( \text{SPK}_7 \) which can be thought of as the instantiation of \( \text{SPK}_6 \).

\[
\text{SPK}_7\{(x, r_1, r_2, \beta_1, \beta_2) : Y_1 = g_1^x \land \mathfrak{A}_1 = \mathfrak{g}_1^{r_1}, \mathfrak{g}_2^{r_2} \land \mathfrak{A}_2 = Y_2^{r_1} g_2^{-\beta_1}\}(m)
\]

(Commitment.) \( \mathcal{P} \) randomly chooses \( \rho_x, \rho_{r_1}, \rho_{r_2}, \rho_{\beta_1}, \rho_{\beta_2} \in_R \mathbb{Z}_p \), computes \( T_1 = g_1^{\rho_x}, T_2 = \mathfrak{g}_1^{\rho_{r_1}}, \mathfrak{g}_2^{\rho_{r_2}}, T_3 = \mathfrak{A}_1^{\rho_{\beta_1}}, \mathfrak{g}_1^{\rho_{\beta_2}} \) and \( T_4 = Y_2^{\rho_{r_1}} g_2^{-\rho_{\beta_1}} \).

(Challenge.) \( \mathcal{P} \) computes \( c = H(\mathfrak{A}_1||\mathfrak{A}_2||T_1||T_2||T_3||T_4||m) \).

(Response.) \( \mathcal{P} \) computes \( z_x = \rho_x - cx \mod p, z_{r_1} = \rho_{r_1} - cr_1 \mod p, z_{r_2} = \rho_{r_2} - cr_2 \mod p, z_{\beta_1} = \rho_{\beta_1} - cwr_1 \mod p, z_{\beta_2} = \rho_{\beta_2} - cwr_2 \mod p \) and outputs \( c, z_x, z_{r_1}, z_{r_2}, z_{\beta_1}, z_{\beta_2} \) as \( \text{SPK}_7 \).

(Verify.) Anyone can verify \( \text{SPK}_7 \) by testing if

\[
c = H(\mathfrak{A}_1||\mathfrak{A}_2||Y_1^z g_1^{z_x} || \mathfrak{A}_1^{z_{r_1}}, \mathfrak{g}_2^{z_{r_2}} || \mathfrak{A}_1^{z_{\beta_1}}, \mathfrak{g}_1^{z_{\beta_2}} g_2^{-z_{\beta_2}} || \mathfrak{A}_2 Y_2^{-z_{r_1}} Y_2^{-z_{\beta_1}} || m)
\]

Any one can verify \( \text{SPK}_6 \) if \( \text{SPK}_7 \) is valid and \( \mathfrak{A}_2 \neq 1_G \).
2.3. Zero-Knowledge Proof-of-Knowledge

The idea of the proof is to compute an auxiliary value $A_2$ which is the quotient of $Y_2$ divided by $g_2^x$ raised to a random number $r_1$. If $Y_2 \neq g_2^x$, the value $Y_2/g_2^x$ will not be $1_G$, the identity element in $G$. On the other hand, the value will be $1_G$ if they are equal. The value $r_1$ acts as a mask to the quotient so that it does not leak anything about $x$. If $A_2$ is not $1_G$, it can be concluded that $Y_2 \neq g_2^x$. The rest of the proof is to demonstrate that $A_2$ is formed correctly. In particular, the use of $A_1$ is to help showing that $\beta_1 = x r_1$. Note also that the goal of the random number $r_2$ is to mask the random number $r_1$ in $A_1$. We call $A_1, A_2$ auxiliary commitments in this thesis as their goals are to help presenting the main idea in a $\Sigma$-protocol. Throughout this thesis, auxiliary commitments shall be denoted using Fraktur font.

Knowledge of $d$-Out-of-$n$ Statements About Discrete Logarithms

Cramer et al. [CDS94] proposed a technique of constructing $\Sigma$-protocol that allows one to prove $d$-out-of-$n$ statement about discrete logarithms are true, without revealing which of them is true, with $O(n)$ complexity. As an example, consider a set $S = \{Y_1, \ldots, Y_n\}$ of $n$ elements $G$ and a subset $S' \subset S$ such that $|S'| = d$. A prover $P$, who knows the discrete logarithm for all $Y_i \in S'$, wishes to demonstrate the knowledge of discrete logarithm for $d$ elements in set $S$ without revealing the elements in $S'$. Specifically,

$$\text{SPK}_8\{(x_1, \ldots, x_n) : \forall s^* \in S', |s^*| = d (\land i \in S^* Y_i = g^{x_i})\}(m)$$

(Commitment.) Let $I$ be a set $\{1, \ldots, n\}$. Define a set $I'$ such that $I' = \{i | Y_i \in S'\}$. Note that $I' \subset I$ and $|I'| = d$. For all $i \in I'$, $P$ randomly chooses $\rho_i \in R \mathbb{Z}_p$, computes $T_i = g^{\rho_i}$. For all $i \in I \setminus I'$, randomly chooses $c_i \in R \{0, 1\}^\lambda$, $z_i \in R \mathbb{Z}_p$ and computes $T_i = Y_i^{c_i} g^{z_i}$.

(Challenge.) $P$ computes $c_0 = H(T_1 || \cdots || T_n || m)$.

(Response.) $P$ generates a polynomial $f$, in $\mathbb{Z}_p$, of degree at most $n - d$ such that $c_0 = f(0)$ and $c_i = f(i)$ for all $i \in I \setminus I'$. For all $i \in I'$, $P$ computes $c_i = f(i)$ and computes $z_i = \rho_i - cx_i \mod p$. $P$ outputs $(f, z_1, \ldots, z_n)$ as $\text{SPK}_8$.

(Verify.) Anyone can verify $\text{SPK}_8$ by testing if $f(0) \overset?= H(Y_1^{f(1)} g^{z_1} || \cdots || Y_n^{f(n)} g^{z_n} || m)$. 

2.3. Zero-Knowledge Proof-of-Knowledge

Example: Conjunction of Several Statements about Discrete Logarithms

We conclude the discussion of ZKPoK about discrete logarithms with an example of constructing a signature of knowledge of some complex statements. The purpose is to illustrate the power of combining the above techniques. Specifically, let $Y_1, Y_2, Y_3$ be elements in $\mathbb{G}$. Assume a prover knows a representation of $Y_3$ to base $g_1$ and $g_2$ such that $Y_3 = g_1^{x_1}g_2^{x_2}$ and would like to construct a signature of knowledge of a pair $(x_1, x_2)$ on the disjunction of the following statements.

1. $Y_1 = g_1^{x_1} \land Y_2 = g_2^{x_2}$; or
2. $Y_1 \neq g_1^{x_1}$; or
3. $Y_3 = g_1^{x_1}g_2^{x_2}$.

The corresponding signature of knowledge can be constructed as follows.

(Auxiliary Commitment.) The prover randomly generates $A_1, A_2 \in_R \mathbb{G}_1$ such that $A_2 \neq 1$.

(Commitment.) Prover randomly chooses $c_1, z_{1,x_1}, z_{1,x_2}, c_2, z_{2,r_1}, z_{2,r_2}, z_{2,x_1}, z_{2,b_1}, z_{2,b_2}, \rho_{3,x_1}, \rho_{3,x_2} \in_R \mathbb{Z}_p$, computes $T_{1,1} = Y_1^{c_1}g_1^{z_{1,x_1}}, T_{1,2} = Y_2^{c_1}g_2^{z_{1,x_2}}, T_{2,1} = A_1^{c_2}g_1^{z_{2,r_1}}g_2^{z_{2,r_2}}, T_{2,2} = A_1^{-z_{2,x_1}}g_1^{z_{2,b_1}}g_2^{z_{2,b_2}}$ and $T_{3,1} = g_1^{\rho_{3,x_1}}g_2^{\rho_{3,x_2}}$.

(Challenge.) Prover computes $c_0 = H(T_{1,1}||T_{1,2}||T_{2,1}||T_{2,2}||T_{2,3}||T_{3,1}||m)$.

(Response.) Prover generates a polynomial $f$, in $\mathbb{Z}_p$, of degree 2 such that $c_0 = f(0)$, $c_1 = f(1)$ and $c_2 = f(2)$. Prover then computes $c_3 = f(3)$ and $z_3,x_1 = \rho_{3,x_1} - c_3x_1 \mod p$ and $z_3,x_2 = \rho_{3,x_2} - c_3x_2 \mod p$. Prover outputs $(f, z_{1,x_1}, z_{1,x_2}, z_{2,r_1}, z_{2,r_2}, z_{2,b_1}, z_{2,b_2}, z_{3,x_1}, z_{3,x_2})$ as the signature of knowledge.

(Verify.) Anyone can verify this signature of knowledge by testing if $f(0) = H(Y_1^{f(1)}g_1^{z_{1,x_1}}||Y_2^{f(1)}g_2^{z_{1,x_2}}||A_1^{f(2)}g_1^{z_{2,r_1}}g_2^{z_{2,r_2}}||A_1^{-z_{2,x_1}}g_1^{z_{2,b_1}}g_2^{z_{2,b_2}}||A_2^{f(2)}Y_1^{z_{2,x_1}}g_3^{z_{2,b_1}}||Y_3^{f(3)}g_1^{z_{3,x_1}}g_2^{z_{3,x_2}}||m)$, $f$ is a degree 2 polynomial and that $A_2 \neq 1$.

2.3.2 ZKPoK about Double Discrete Logarithms

A double discrete logarithm of $Y \in \mathbb{G}$ to bases $g$ and $h$ is the integer $y \in \mathbb{Z}_q$ such that $Y = g^{hy}$, if such an $y$ exists. The following signature of knowledge demonstrates the knowledge of a double discrete logarithm was due to [CS97].

$$\text{SPK}_g\{(y) : Y = g^{hy}\}(m)$$
2.3. Zero-Knowledge Proof-of-Knowledge

(Commitment.) For $i = 1$ to $\lambda$, the prover $P$ randomly chooses $\rho_i \in_R \mathbb{Z}_q$, computes $T_i = g^{h^{\rho_i}}$.

(Challenge.) $P$ computes $c = H(T_1 || \cdots || T_\lambda || m)$.

(Response.) For $i = 1$ to $\lambda$, $P$ computes $z_i = \rho_i - c[i]x \mod q$ and outputs $(c, z_1, \ldots, z_\lambda)$ as $\text{SPK}_9$.

(Verify.) Anyone can verify $\text{SPK}_9$ by testing, for $i = 1$ to $\lambda$,

\[
T_i \overset{?}{=} g^{h^{z_i}} \text{ if } c[i] = 0 \\
T_i \overset{?}{=} Y^{h^{z_i}} \text{ if } c[i] = 1
\]

Note that $\text{SPK}$ for double discrete logarithms is less efficient than that of relations amongst discrete logarithms. In particular, the complexity is linear in the security parameter $\lambda$. The soundness of the protocol is in fact $2^{-\lambda}$. Moreover, there is no known ZKPoK protocols for demonstrating the relationship amongst double discrete logarithms.
Chapter 3

Signature Schemes with Efficient Protocols

Digital signature schemes are a fundamental cryptographic primitive and building blocks in cryptographic protocol design. In a series of papers, Camenisch and Lysyanskaya [CL01, CL02a, CL04] identified a special class of digital signature schemes, commonly referred to as a “CL-signature”, as a key building block for privacy-preserving cryptographic systems.

A CL-signature comprises of a digital signature scheme together with two protocols, namely, Issue and Prove. Issue is a protocol between a signer and a user. The user wishes to obtain a signature on a value $M$ from the signer with public key $PK$. The user forms a commitment $C$ to the value $M$ and gives $C$ to the signer. Upon successful completion of the protocol, the user obtains a signature $\sigma$ on $M$ while the signer learns nothing about $M$ except he has signed the value that the user has committed to. Prove is a ZKPoK protocol of a signature on a committed value. The prover, also known as the user, is in possession of a message-signature pair $(M,\sigma)$. The prover formed a commitment $C$ to value $M$ and, proves, in zero-knowledge, that he knows a pair $(M,\sigma)$ such that $C$ is a commitment of $M$ and $(M,\sigma)$ is a valid message-signature pair.

CL-signatures are useful building blocks for privacy-preserving cryptographic systems. For instance, they are used directly to construct anonymous credentials [CL01, CL04], $k$-times anonymous authentications ($k$-TAA) [TF04, TS06], electronic cash systems [CHL05, CG07], direct anonymous attestation [BL07], to name a few.
Related Work

There are several constructions of CL-signatures in literature, including the original CL-signature [CL02a], the LRSW assumption based CL-signature [CL04] (sometimes known as CL+) and very recently, P-signatures [BCKL08]. P-signatures address a shortcoming of CL-signatures, the lack of efficient non-interactive Prove protocol without relying on the random oracle model. Their scheme makes an extensive use of the non-interactive proof techniques by Groth and Sahai [GS08]. Roughly speaking, a P-signature is a CL-signature with an efficient non-interactive Prove protocol. A shortcoming common to existing CL-signatures is that they are secure only when the Issue protocol is executed sequentially.

Our Contribution

We present three CL-signatures, namely, BBS+ [ASM06], ESS+ [ASM08b] and C-Signature [ASM08a], all of which are capable of signing block of messages. The first one, BBS+, is not conceptually new. Indeed, [CL04] outlined how the short group signature scheme due to Boneh, Boyen and Shacham [BBS04] can be modified to become a $q$-SDH variant of the original CL-signature. We supply the details of the modification with the protocols and prove that BBS+ is strongly existentially unforgeable under chosen message attack in the standard model under the $q$-SDH assumption. Besides, Prove of our construction is different from [BBS04] in which the modified protocol achieves perfect zero-knowledge, while the original protocol is only computational due to the identity-escrow.

The second scheme, ESS+, is new and is based on a new complexity assumption called AWSM [AWSM07]. It was given the name ESS+ because the original scheme proposed was called extended special signature (ESS). ESS is a hybrid scheme in which the signer learns part of the messages being signed during the Issue protocol and is thus not considered a construction of CL-signatures. Nonetheless, ESS+ has an interesting property in which the message space is an element in a cyclic group $G$ together with a tuple in the group $Z_p$. It turns out that this feature is useful in a new approach for constructing electronic cash scheme, described in Chapter 5.

The last scheme, C-Signature, is CL-signature that supports concurrent execution of the Issue protocol efficiently. Indeed, Issue in our construction only involves two short communication flows between the signer and user. This addresses the shortcoming of existing CL-signatures that they are secure only when the Issue protocol
is executed *sequentially*. The realisation is based on a *new technique* for proving the knowledge of representation of a committed value. This technique allows demonstration of relationship amongst double discrete logarithms and may be of independent interest.

## 3.1 Syntax

In this thesis, a signature scheme with efficient protocols, also known as a CL-signature, is a tuple \( \text{CL Sig} = (\text{KeyGen, Sign, Verify, Issue, Prove}) \) of five PPT algorithms/protocols described below.

**KeyGen.** On input security parameter \( 1^\lambda \), output a key pair \((PK, SK)\). In particular, \( PK \) includes a message space \( \mathcal{M} \).

**Sign.** On input a private key \( SK \) and a message \( M \) (which may be a block of messages), output a signature \( \sigma \). We use \( \sigma \leftarrow \text{Sign}_{SK}(M) \) to denote \( \sigma \) is the output of the \( \text{Sign} \) algorithm on message \( M \).

**Issue.** This is a protocol between a signer with input \((PK, SK)\) and a user with input \( PK \) and a message \( M \). Upon successful completion of the protocol, the user obtains a signature \( \sigma \).

**Verify.** On input \( PK \), a message \( M \) and a signature \( \sigma \), output valid or invalid. Similarly, we use \( \text{valid} \leftarrow \text{Verify}_PK(M, \sigma) \) (resp. \( \text{invalid} \leftarrow \text{Verify}_PK(M, \sigma) \)) to denote the output of the \( \text{Verify} \) algorithm on input \( PK \) and \( M \) and \( \sigma \).

**Prove.** This is a protocol which allows a prover with input \( PK \), message \( M \) and the corresponding signature \( \sigma \), to prove to a verifier, with input \( PK \), that he is in possession of the pair \((M, \sigma)\) such that \( \text{valid} \leftarrow \text{Verify}_PK(M, \sigma) \).

A \( \text{CL Sig} \) should possess *correctness, unforgeablility, Issue-privacy* and *Prove-privacy* described below.

### Correctness

*Correctness* requires that if \((PK, SK) \leftarrow \text{KeyGen}(1^\lambda)\), for all messages \( M \) from the message space \( \mathcal{M} \) and \( \sigma \leftarrow \text{Sign}_{SK}(M) \), \( \text{Verify}_PK(M, \sigma) \) must output valid. Besides, we require that any signature \( \sigma \) obtained by the user upon successful termination
of an Issue protocol is indistinguishable with the signature generated using the Sign algorithm.

Unforgeability

Unforgeability of CL Sig is similar to that of standard signature. For the purpose of this thesis, we are interested in (resp. strong) existential unforgeability under chosen message attack, denoted by EU-CMA (resp. Strong EU-CMA). The following game, GAME CL-EU-CMA (resp. CL-S-EU-CMA), is a modification of GAME EU-CMA (resp. GAME S-EU-CMA) to accommodate the introduction of Issue and Prove in CL Sig. The rationale is that, after obtaining $q_1$ signatures from the Sign algorithm and $q_2$ signatures from the Issue protocol, no PPT adversaries should be able to come up with $q_1 + q_2 + 1$ signatures.

Definition 3.1 (GAME CL-EU-CMA (resp. CL-S-EU-CMA))

Setup: A challenger $C$ runs KeyGen to obtain a public key $PK$ and a private key $SK$. $C$ gives $PK$ to adversary $A$.

Sign Query: Let $q_1$ be the total number of this type of query. For each sign query, $A$ requests a signature on message $M_i$. $C$ responds to each query with a signature $\sigma_i \leftarrow \text{Sign}_{SK}(M_i)$.

Issue Query: Let $q_2$ be the total number of this type of query. For each Issue query, $A$ acts as a user and $C$ acts as a signer in the Issue protocol.

Output: At some stage, $A$ outputs $q_1 + q_2 + 1$ pairs $(M_i^*, \sigma_i^*)$. $A$ wins the game if for all pairs, $\text{valid} \leftarrow \text{Verify}_{PK}(M_i^*, \sigma_i^*)$ and $M_i^* \neq M_j^*$ if $i \neq j$ (resp. $(M_i^*, \sigma_i^*) \neq (M_j^*, \sigma_j^*)$ if $i \neq j$).

We said CL Sig is (strongly) existentially unforgeable under concurrent chosen message attack (C-EU-CMA) (resp. strong C-EU-CMA) if no polynomial-time adversary $A$ can win in GAME CL-EU-CMA (resp. CL-S-EU-CMA) with non-negligible probability. If the adversary is restricted and that queries cannot be issued in parallel, the corresponding CL Sig is said to be (strong) existentially unforgeable under chosen message attack (EU-CMA) (resp. S-EU-CMA).
Issue-Privacy

Issue-privacy protects the privacy of a user during the Issue protocol. In particular, it guarantees that after the protocol, the signer learns no information about the message $M$ being signed. Note that we only require the message is hidden from the signer, but not necessarily the output signature $\sigma$. The reason is that in most cases, CL Sig is used in applications where $\sigma$ is never shown in clear. The user will just use the Prove protocol to demonstrate he knows $\sigma$. Nevertheless, the following game, GAME CP-Issue, formally defines Issue-Privacy.

Definition 3.2 (GAME CP-Issue)

Setup: A challenger $C$ runs KeyGen to obtain a public key $PK$ and a private key $SK$. $C$ gives both $(PK, SK)$ to adversary $A$. $A$ randomly generates two messages $M_0, M_1$ within the message space and gives $M_0, M_1$ to challenger.

Challenge: $C$ flips a fair coin $b \in \{0, 1\}$ and runs Issue protocol as a user with input $M_b$, with $A$ acting as the signer.

Output: $A$ outputs a guess bit $b' \in \{0, 1\}$, indicating whether $M_0$ or $M_1$ was used during the challenge phase.

CL Sig is said to possess Issue-privacy if no PPT adversary $A$ can win in GAME CP-Issue with non-negligible probability over random guessing. That is, the probability of $A$ guessing correctly minus half should be negligible.

Prove-Privacy

Prove-privacy ensures that the verifier, even with the help of the signer, learns nothing about the underlying message-signature pair in a Prove protocol run. For simplicity, we required that Prove protocol of CL Sig to be a $\Sigma$-protocol on the knowledge of a message-signature pair.

We say a CL Sig is secure if it possesses the above four properties. It is concurrently-secure if, in addition, it is C-EU-CMA. Using the notation of [ASM08a], a concurrently-secure CL Sig is called a C-Signature.
3.2 BBS+ Signature

BBS+ signature is built from bilinear pairing. In the following we describe BBS+ in the general setting of asymmetric pairing \( \hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T \). Note that, however, the security proof requires an efficient isomorphism \( \psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1 \) although such mapping is not needed in the scheme itself.

3.2.1 Relationship of BBS+ with the BBS Group Signature

As discussed, \([CL04]\) outlined how the BBS group signature \([BBS04]\) can be modified to become a \( q \)-SDH variant of the original CL-signature. In BBS group signature, group manager issues a membership certificate to a user equipped with a secret value. To generate a group signature, the user produces a signature of knowledge of the membership certificate as well as his secret value. The idea in \([CL04]\) is to treat the membership certificate on the user secret value as a signature on a message.

3.2.2 Construction of BBS+

**KeyGen.** On input security parameter \( 1^\lambda \), generate a bilinear map \( \hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T \) such that \( |\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T| = p \) for some \( \lambda \)-bit prime \( p \). Let \( g, g \) be generators of \( \mathbb{G}_1 \) and \( \mathbb{G}_2 \) respectively, with \( g = \psi(g) \) such that \( \psi \) is an efficiently computable isomorphism from \( \mathbb{G}_2 \) to \( \mathbb{G}_1 \). Let \( \mu \in R \mathbb{Z}_p \) and \( Z = g^\mu \in \mathbb{G}_2 \). The message space is \( M = \mathbb{Z}_p^L \) for some (possibly small) positive integer \( L \). Let \( g_0, g_1, \ldots, g_L \in R \mathbb{G}_1 \) be additional random generators of \( \mathbb{G}_1 \). The public key \( PK \) is \( (\hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, g_0, g_1, \ldots, g_L, g, Z) \) and the secret key \( SK \) is \( (\mu) \).

**Sign.** Given \( SK = (\mu) \) and message \( M = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L \), pick some random \( e, s \in R \mathbb{Z}_p \). Compute \( \zeta = (g_0 g_1^m \ldots g_L^m)^{1/\mu} \in \mathbb{G}_1 \). Signature \( \sigma \) on \( M \) is \( (\zeta, e, s) \in (\mathbb{G}_1 \times \mathbb{Z}_p \times \mathbb{Z}_p) \).

**Issue.** This is a protocol between a signer with input \( SK = (\mu) \) and user with input \( M = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L \). Both parties have the common input \( PK \).

1. User randomly chooses \( s' \in R \mathbb{Z}_p \), computes and sends \( C_M = g_0^{s'} g_1^{m_1} \ldots \)
3.2. **BBS+ Signature**

\[ g_L^m \in \mathbb{G}_1, \text{ along with the following proof}^{1} \text{ to the signer.} \]

\[ \text{SPK}_{10}\{(s', m_1, \ldots, m_L) : C_M = g_0^{s'} g_1^{m_1} \cdots g_L^{m_L}\}(R) \]

2. After verifying \( \text{SPK}_{10} \), the signer chooses \( s'', e \in \mathbb{Z}_p \), computes \( \zeta = (gg_0^{s''}C_M)^{\frac{1}{e}} \in \mathbb{G}_1 \) and sends \((\zeta, e, s'')\) back to the user.

3. The user computes \( s = s' + s'' \mod p \) and stores the signature \( \sigma \) on \( M \) as \((\zeta, e, s)\).

**Verify.** Given a public key \( PK \), a message \( M = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L \), a signature \( \sigma = (\zeta, e, s) \), verify that

\[ \hat{e}(\zeta, Zg^e) = \hat{e}(g_0^{s} g_1^{m_1} \cdots g_L^{m_L}) \]

The equation is also known as the verification equation. If the equality holds, output valid. Otherwise, output invalid.

**Prove.** This is a \( \Sigma \)-Protocol of the knowledge of a message-signature pair \((M = (m_1, \ldots, m_L), \sigma = (\zeta, e, s))\), between a prover (user) and a verifier, such that the pair satisfies the verification equation. For the ease of representation, we let \( C_M = g_0^{r} g_1^{m_1} \cdots g_L^{m_L} \in \mathbb{G}_1 \), which is a commitment of \( M \) using randomness \( r \). We assume the prover wishes to prove he is in possession of a signature on a valued \( M \) committed to \( C_M \). The protocol shall be abstracted as

\[ \text{PK}_{11}\{ (\zeta, e, s, m_1, \ldots, m_L, r) : \]

\[ \hat{e}(\zeta, Zg^e) = \hat{e}(g_0^{s} g_1^{m_1} \cdots g_L^{m_L}, g) \land \]

\[ C_M = g_0^{r} g_1^{m_1} \cdots g_L^{m_L} \}

\( \text{PK}_{11} \) cannot be directly constructed using the techniques discussed in Section 2.3. Below we describe how it can be transformed so that those techniques apply. Throughout this thesis, we say the proof-of-knowledge requires instantiation to describe this situation. Specifically, the prover first computes \( \mathcal{A}_1 = g_1^{r_1} g_2^{r_2} \in \mathbb{G}_1, \mathcal{A}_2 = \zeta g_2^{r_1} \in \mathbb{G}_1 \) for some randomly generated \( r_1, r_2 \in \mathbb{Z}_p \). Then he sends \((\mathcal{A}_1, \mathcal{A}_2)\) to the verifier and conducts the following protocol \( \text{PK}_{12} \). \( \text{PK}_{12} \) is a direct application of zero-knowledge proof-of-knowledge of

---

1This proof is a direct application of proof-of-knowledge of representation of an element, discussed in Section 2.3.1. This proof guarantee that \( C_M \) is correctly formed to the signer. Here \( R \) is a random challenge issued by the signer. One could use \( \text{SPK} \) or \( \text{PK} \) here, and a subtle difference shall be discussed during the security analysis.
3.2. BBS+ Signature

As an example, the actual steps are given below.

\[
\text{PK}_{12}\left\{ \left( e, s, m_1, \ldots, m_L, r, r_1, r_2, \beta_1, \beta_2 \right) : \right. \\
\mathcal{A}_1 = g_1^{r_1} g_2^{r_2} \quad \wedge \\
1_{G_1} = \mathcal{A}_1^{-e} g_1^{\beta_1} g_2^{\beta_2} \quad \wedge \\
\mathcal{C}_M = g_0 g_1^{m_1} \cdots g_L^{m_L} \quad \wedge \\
\frac{\hat{e}(\mathcal{A}_2, Z)}{\hat{e}(g, g)} = \hat{e}(\mathcal{A}_2, g)^{-e} \hat{e}(g_2, Z)^{r_1} \hat{e}(g_2, g)^{\beta_1} \hat{e}(g_0, g)^s \quad \hat{e}(g_1, g)^{m_1} \cdots \hat{e}(g_L, g)^{m_L} \\
\left. \right\} 
\]

(Commitment.) Prover randomly generates \( \rho_{r_1}, \rho_{r_2}, \rho_{\beta_1}, \rho_{\beta_2}, \rho_r, \rho_{m_1}, \ldots, \rho_{m_L}, \rho_e, \rho_s \in \mathbb{Z}_p \), computes \( T_1 = g_1^{\rho_{r_1}} g_2^{\rho_{r_2}}, T_2 = \mathcal{A}_1^{-\rho_e} g_1^{\rho_{\beta_1}} g_2^{\rho_{\beta_2}}, T_3 = g_0^{\rho_r} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} \in \mathbb{G}_1 \) and \( T_4 = \hat{e}(\mathcal{A}_2, g)^{-\rho_e} \hat{e}(g_2, Z)^{\rho_{r_1}} \hat{e}(g_2, g)^{\rho_{\beta_1}} \hat{e}(g_0, g)^{\rho_s} \hat{e}(g_1, g)^{\rho_{m_1}} \cdots \hat{e}(g_L, g)^{\rho_{m_L}} \in \mathbb{G}_T \). Prover sends \( (T_1, T_2, T_3, T_4) \) to the verifier.

(Challenge.) Verifier chooses a random challenge \( c \in \mathbb{Z}_p \{0, 1\}^\lambda \) and sends \( c \) to prover.

(Response.) Prover computes \( z_{r_1} = \rho_{r_1} - cr_1, z_{r_2} = \rho_{r_2} - cr_2, z_{\beta_1} = \rho_{\beta_1} - c r_1, z_{\beta_2} = \rho_{\beta_2} - c r_2, z_r = \rho_r - cr, z_{m_1} = \rho_{m_1} - cm_1, \ldots, z_{m_L} = \rho_{m_L} - cm_L, z_e = \rho_e - ce, z_s = \rho_s - cs \) and sends \( (z_{r_1}, z_{r_2}, z_{\beta_1}, z_{\beta_2}, z_r, z_{m_1}, \ldots, z_{m_L}, z_e, z_s) \) to verifier.

(Verify.) Verifier outputs \textbf{accept} if and only if the following four equations hold.

\[
T_1 = \mathcal{A}_1^c g_1^{z_{r_1}} g_2^{z_{r_2}} \\
T_2 = \mathcal{A}_1^{-c} g_1^{z_{\beta_1}} g_2^{z_{\beta_2}} \\
T_3 = \mathcal{C}_M g_0^c g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} \\
T_4 = \left( \frac{\hat{e}(\mathcal{A}_2, g)}{\hat{e}(g, g)} \right)^c \hat{e}(\mathcal{A}_2, g)^{-zc} \hat{e}(g_2, Z)^{z_{r_1}} \hat{e}(g_2, g)^{z_{\beta_1}} \hat{e}(g_0, g)^{zs} \hat{e}(g_1, g)^{zm_1} \cdots \hat{e}(g_L, g)^{zm_L} 
\]

\( \text{PK}_{12} \) can be best understood as an instantiation of \( \text{PK}_{11} \) as follows. The first two equalities together prove that \( \beta_1 = r_1 e \) and the third equality is just used to prove the correctness of \( \mathcal{C}_M \). Now if we put \( \beta_1 = r_1 e \) into the last equality, we are in fact proving \( \hat{e}(\mathcal{A}_2 g_2^{-r_1}, Z) \hat{e}(\mathcal{A}_2 g_2^{-r_1}, g^e) = \hat{e}(g_0 g_1^{m_1} \cdots g_L^{m_L}, g) \). That
is, $\hat{e}(A_2g_2^{-r_1}, Zg') = \hat{e}(gg_0^{s_1} \cdots g_L^{s_L}, g)$ and the signature held by the prover is $(A_2g_2^{-r_1}, e, s)$.

### 3.2.3 Security Analysis of BBS+

It is trivial to show that BBS+ possesses correctness. Below, we investigate the unforgeability, Issue-Privacy and Prove-Privacy.

#### Unforgeability

We show that BBS+ is strongly unforgeable under the assumption that $q$-SDH problem in bilinear group (definition 2.7) is hard by reduction. Suppose there exists a PPT adversary $A$ which could win GAME CL-S-EU-CMA with non-negligible advantage and assume it makes less than or equal to $q$ signature queries. We construct a simulator $S$ which solves the $q$-SDH problem in bilinear group pair. The proof falls along the line of that in [BB08]. In particular, the major technique is the same as in Lemma 9 of [BB08] such that given a $q$-SDH problem instance, one can construct a new 1-SDH problem instance with $q-1$ solutions and that any new solution reveals a solution to the original problem instance. The proof is presented below.

**Proof:** $S$ is given an instance of the $q$-SDH problem in $(g', g', g'^0, \ldots, g'^L) \in (G_1 \times G_2 + 1)$ such that there exists a pairing $\hat{e} : G_1 \times G_2 \rightarrow G_T$ and an isomorphism $\psi : G_2 \rightarrow G_1$ with $g' = \psi(g')$. Note that $|G_1| = |G_2| = |G_T| = p$ for some prime $p$. The goal of $S$ is to output a pair $(A', e') \in (G_1 \times \mathbb{Z}_p)$ such that $A'^{e' + \mu} = g'$. This pair satisfies $\hat{e}(A', e', g'^0) = \hat{e}(g', g')$.

**Simulating the public parameters.** $S$ first randomly chooses, for $i = 1$ to $q - 1$, $e_i \in_R \mathbb{Z}_p$ and denotes the $q-1$ degree polynomial $f(x) = \prod_{i=1}^{q-1}(x + e_i)$. $S$ also randomly chooses $e^*, k^*, a^* \in_R \mathbb{Z}_p$ and computes $g = g'^{f(\mu)} \in G_2$, $Z = g^\mu = g'^{\mu f(\mu)} \in G_2$ and $g_0 = g^{(e^* + \mu)k^*-1} \in G_2$. Next $S$ randomly chooses $\mu_j \in_R \mathbb{Z}_p^*$ and sets $g_j = g_0^{\mu_j}$ for $j = 1$ to $L$. Finally, $S$ computes $g = \psi(g)$ and $g_j = \psi(g_j)$ for $j = 0$ to $L$ and gives $(\hat{e}, G_1, G_2, G_T, p, g, g_0, \ldots, g_L, g, Z)$ to $A$ as the public key of the BBS+ signature. Note that due to the setting of the $q$-SDH assumption, $g'^{f(\mu)}$ for any degree-$q$ polynomial $f$ in $\mu$ with known coefficients are computable.

---

$^2$A signature query is either a **Sign** query or a **Issue** query.
3.2. BBS+ Signature

Handling queries. \( A \) is allowed to issue up to \( q \) signature queries. If it is a \texttt{Issue} protocol query, \( S \) needs to rewind \( A \) and uses the knowledge extractor of \( \text{SPK}_{10} \) to obtain the underlying block of messages \((s', m_1, \ldots, m_L)\) in the commitment \( C_M \). If \( \text{SPK} \) is used, \( S \) is given access to the random oracle and the scheme shall be secure under the random oracle model. After running the knowledge extractor, the query can be handled in the same way as the normal \texttt{Sign} query. Note that, due to the need of rewinding, BBS+ signature cannot be proven C-EU-CMA. See Section 3.2.3 for a more detailed discussion. Nonetheless, below we describe how a normal \texttt{Sign} query is handled.

For the \( i \)-th query, denote the block of messages to be signed as \((m_{1,i}, \ldots, m_{L,i})\).

For each query, \( S \) computes \( M_i = \sum_{j=1}^{L} m_{j,i} \mu_j \).

Out of these \( q \) queries, \( S \) randomly chooses one, which we shall called query \( * \) and shall be handled differently. For the rest of the \( q - 1 \) queries, \( S \) randomly picks \( s_i \in_R \mathbb{Z}_p \), denotes \( S_i = s_i + M_i \) computes \( \varsigma_i = \left( gg_{0}^{s_i} \right)^{1/(e_i + \mu)} \). Note that

\[
\varsigma_i = \left( gg_{0}^{S_i} \right)^{1/(e_i + \mu)} = \left( g^{(e^* + \mu)k^* + a^* - 1} \right)^{1/(e_i + \mu)} = \left( g^{f(\mu)/e_i + \mu} \right)^{a^* - 1} \left( g^{g'(\mu)(e^* + \mu)} \right)^{k^*}
\]

and is computable by \( S \) even though \( \mu \) is unknown since \((e_i + \mu)\) divides \( f(\mu) \) and \((e^* + \mu)(f(\mu)) \) is a degree \( q \) polynomial. \( S \) returns \((\varsigma_i, e_i, s_i)\) as the answer of the \( i \)-th \texttt{Sign} query. In case it is a \texttt{Issue} query, \( S \) returns \((\varsigma_i, e_i, s_i - s')\) where \( s' \) is the value obtained using the knowledge extractor of \( \text{PK}_{10} \).

For query \( * \), \( S \) computes \( s^* = a^* - M_\ast \) and returns \((g^{k^*}, e^*, s^*)\) as the answer. Similarly, in case it is a \texttt{Issue} query, \( S \) returns \((g^{k^*}, e^*, s^* - s')\) where \( s' \) is the value obtained using the knowledge extractor of \( \text{PK}_{10} \). Recall that \( g_0 = g^{(e^* + \mu)k^* - 1} \in \mathbb{G}_2 \). Thus,

\[
\psi((g^{k^*})^{e^* + \mu}) = \psi(g^{a^*}_0) \\
(g^{k^*})^{e^* + \mu} = g^{a^*_0} \\
= g^{a^*_0} g_{0}^{M_\ast} \\
= g^{a^*_0} g_{1}^{m_{1,\ast}} \cdots g_{L}^{m_{L,\ast}}
\]

and thus \((g^{k^*}, e^*, s^*)\) is a valid signature.

The reduction. Finally, \( A \) outputs \( q + 1 \) distinct messages-signature pairs. At least one of them is different from the \( q \) message-signature pairs obtained during the
signing query phase.

Let this signature-message pair be \((s', e', s')\), \((m'_1, \ldots, m'_L)\). Denote \(S' = s' + \sum_{j=1}^{L} m_j \mu_j\). There are three possibilities.

- **Case I** \([e' \notin \{e_i, e^*\}]:

\[
\begin{align*}
\hat{e}(s', Z g^{e'}) &= \hat{e}(g g_0^{s'}, g) \\
\zeta^{e' + \mu} &= g g_0^{s'} \\
\zeta^{s' + \mu} &= g^{-\frac{k^* s'}{a^*}} g^{(\mu + e^*) k^* s'}
\end{align*}
\]

Since \(e' \notin \{e_i, e^*\}\), \((e' + \mu)\) does not divide \(f(\mu)\) (resp. \(f(\mu)(\mu + e^*)\)) and \(S\) computes a \(q - 2\) (resp. \(q - 1\)) degree polynomial \(Q(\mu)\) (resp. \(Q^*(\mu)\)) and constant \(Q\) (resp. \(Q^*\)) such that \(f(\mu) = Q(\mu)(\mu + e') + Q\) (resp. \(f(\mu)(\mu + e^*) = Q^*(\mu)(\mu + e') + Q^*\)). Thus,

\[
\zeta' = \left( g' \frac{g^e}{a^e} \right)^{Q(\mu)} + \frac{Q}{e + \mu} \left( g' \frac{(e + \mu) k^* s'}{a^e} \right)^{Q(\mu)} + \frac{Q^*}{\mu + e^*}
\]

\[
g'^{\frac{1}{\mu + e^*}} = \left( \zeta' \left( g'^{Q(\mu)} \frac{g^e}{a^e} \right) \left( g'^{Q^*(\mu)} \frac{k^* s'}{a^e} \right) \right)^{\frac{1}{Q + Q^*}}
\]

Thus, \((g'^{\frac{1}{\mu + e^*}}, e')\) is the solution to the \(q\)-SDH problem.

- **Case II**: \([e' = e_i\) and \(\zeta' = \zeta_i\): This happens with negligible probability unless \(A\) solves the relative discrete logarithm amongst two of the \(g_i\)'s.

- **Case III**: \([e' \in \{e_i, e^*\}\) and \(\zeta' \neq \zeta_i\): With probability, \(1/q\), \(e' = e^*\).

\[
\begin{align*}
\hat{e}(s', Z g^{e^*}) &= \hat{e}(g g_0^{s'}, g) \\
\zeta^{e^* + \mu} &= g g_0^{s'} \\
\zeta^{s^* + \mu} &= g^{-\frac{k^* s'}{a^*}} g^{(e^* + \mu) k^* s'} \\
\zeta' &= g^{-\frac{k^* s'}{a^*}} g^{(e^* + \mu) k^* s'}
\end{align*}
\]

Since \((e^* + \mu)\) does not divide \(f(\mu)\), \(S\) computes \(q - 2\) degree polynomial \(Q(\mu)\) and constant \(Q\) such that \(f(\mu) = (e^* + \mu) Q(\mu) + Q\). Thus,

\[
\zeta' = g'^{\frac{e^* s'}{a^*}} \left( g'^{Q(\mu)} + \frac{Q}{e + \mu} g^{Q(\mu) k^* s'} \right)^{\frac{e^* s'}{a^*}}
\]

\[
g'^{\frac{1}{\mu + e^*}} = \left( \zeta' \frac{g^{-\frac{k^* s'}{a^*}} g^{s^*}}{a^*} \right)^{\frac{1}{Q}}
\]

Thus, \((g'^{\frac{1}{\mu + e^*}}, e')\) is the solution to the \(q\)-SDH problem.
If the success probability of $A$ is $\epsilon$, then in the worst case, success probability of $S$ is $\epsilon/q$. Consequently, if there exists a PPT adversary $A$ which wins GAME CL-S-EU-CMA with non-negligible probability, there exists a PPT simulator $S$ that solves the $q$-SDH problem with non-negligible probability and this completes our proof. \(\square\)

**On Number of Rewinding Required**

In the above proof, the simulator rewinds the adversary for each `issue` query to extract the underlying block of messages. The number of rewinding required is equal to the number of `issue` query if the adversary only issue the query sequentially. However, it becomes much more complicated if the adversary is allowed to conduct `issue` queries concurrently. Specifically, rewinding cannot be employed and a “straight-line” approach needs to be followed that makes the complexity of the simulator to become exponential.

**Issue-Privacy**

`Issue-Privacy` of BBS+ signature is straightforward. Note that during `issue` protocol, the signer only obtains the commitment of the block of messages $C_M = g_0^{s'} g_1^{m_1} \cdots g_L^{m_L}$ together with the $\text{SPK}_{10}$. Due to the unconditional hiding property of the commitment scheme and the zero-knowledgeness of the $\text{SPK}$, the signer learns nothing about the message being signed.

Indeed, there is a subtlety worth some discussions. While $C_M$ leaks no information on the messages because for any $C_M$ and any block of messages $(m_1^*, \ldots, m_L^*)$, there exists an $s'$ such that $C_M = g_0^{s'} g_1^{m_1^*} \cdots g_L^{m_L^*}$, the zero-knowledgeness of $\text{SPK}_{10}$ requires some attention. In the actual proof, the simulator is required to invoke the zero-knowledge simulator $\text{KS}$ and if the non-interactive $\text{SPK}$ is used, the simulator is given control over the random oracles. That is, the scheme is secure only under the random oracle model. On the other hand, if the interactive $\text{PK}_{10}$ is employed, the zero-knowledge version instead of the 3-move $\Sigma$-Protocol has to be used. The reason is that in this case, the verifier (that is, the signer) is no longer trusted to be honest. Luckily, one could always transform a $\Sigma$-protocol into a ZKPoK protocol in the auxiliary string model or with trapdoor commitment, as discussed in Section 2.3.
3.3 ESS+ Signature

Prove-Privacy of BBS+ signature is also straightforward. The proof that \( \text{PK}_{12} \) is a \( \Sigma \)-Protocol is omitted as it is trivial. Moreover, the values \( C_M, A_1 \) and \( A_2 \) leak no information about the message nor the signature. Thus, Prove protocol leaks no information about the message nor the signature.

3.3 ESS+ Signature

ESS+ signature is built from bilinear pairing in groups such that the SXDH assumption holds. Admittedly, SXDH is a strong assumption. We are unable to reduce the security of ESS+ to any existing number theoretic hard problems even under the SXDH assumption. ESS+ works in the setting of cyclic groups \((G_1, G_2, G_T)\) of prime order \(p\) with pairing \( \hat{e} : G_1 \times G_2 \rightarrow G_T \). An interesting feature of ESS+ is that the message space is \( M = (G_1 \times \mathbb{Z}_p^L) \) for some (small) positive integer \( L \). With this message space, we need a new commitment scheme as well. The commitment scheme, together with our construction of ESS+ and its security analysis are given below.

3.3.1 A Commitment Scheme

Gen. On input security parameter \( 1^\lambda \), output a cyclic group \( G \) of prime order \( p \).

Output generators \( h_1, h_2, g, g_0, g_1, \ldots, g_L \in_R G \) of \( G \). The parameter \text{param} is \((G, p, h_1, h_2, g, g_0, g_1, \ldots, g_L)\).

Commit. To commit \((M, m_1, \ldots, m_L) \in (G \times \mathbb{Z}_p^L)\), the committer randomly generates \( s, t \in_R \mathbb{Z}_p \), outputs \((C_M, C_m) \in (G \times G) =, (Mh_1^s, h_2^s g_0^t g_1^{m_1} \cdots g_L^{m_L})\).

Reveal. To reveal the commitment \((C_M, C_m)\), the committer outputs \((M, m_1, \ldots, m_L, s, t)\). Everyone can test if \( C_M = Mh_1^s \) and \( C_m = h_2^s g_0^t g_1^{m_1} \cdots g_L^{m_L} \).

Hiding. The scheme is unconditionally hiding. For any \((C_M, C_m) \in (G \times G)\) and any \((M, m_1, \ldots, m_L) \in (G \times \mathbb{Z}_p^L)\), there exists \( s, t \in \mathbb{Z}_p \) such that \( C_M = Mh_1^s \) and \( C_m = h_2^s g_0^t g_1^{m_1} \cdots g_L^{m_L} \). Thus, the scheme is hiding even for computationally unbounded adversary.
3.3. ESS+ Signature

Binding. Suppose some PPT $\mathcal{A}$ is able to open a commitment $(C_M, C_m)$ in two ways, say, $(M, m_1, \ldots, m_L, s, t)$ and $(M', m'_1, \ldots, m'_L, s', t')$. Consider the following cases $M \neq M'$ and $M = M'$. For the first case, it implies $s \neq s'$. Consequently, $\mathcal{A}$ has come up with two different representation of $C_m$ to bases $h_2, g_0, g_1, \ldots, g_L$. Thus $\mathcal{A}$ has solved the representation problem, which is equivalent to DLP in $\mathbb{G}$.

The second case implies $s = s'$. Since $(m_1, \ldots, m_L, s, t) \neq (m'_1, \ldots, m'_L, s', t')$, this again implies $\mathcal{A}$ is able to come up with two representation of $C_m$.

3.3.2 Construction of ESS+

KeyGen. On input security parameter $1^\lambda$, generate a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that $\left| \mathbb{G}_1 \right| = \left| \mathbb{G}_2 \right| = \left| \mathbb{G}_T \right| = p$ for some $\lambda$-bit prime $p$. Let $g, h_1, h_2, g_0, \ldots, g_L \in_{R} \mathbb{G}_1$ be generators of $\mathbb{G}_1$, $g, g_1 \in_{R} \mathbb{G}_2$ be generators of $\mathbb{G}_2$. The secret key $SK$ is a pair $(X, \mu) \in (\mathbb{G}_1, \mathbb{Z}_p)$. Let $Z = g^\mu \in \mathbb{G}_2$ and $Z = \hat{e}(X, g) \in \mathbb{G}_T$. The public key $PK$ is $(\hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, g_0, g_1, \ldots, g_L, h_1, h_2, g, g_1, Z, Z)$ and the secret key $SK$ is $(X, \mu)$. The message space $\mathcal{M}$ is $(\mathbb{G}_1 \times \mathbb{Z}_p^L)$.

Sign. Given $SK = (X, \mu)$ and message $M = (M, m_1, \ldots, m_L) \in \mathcal{M}$, pick some random $e, s, t \in_{R} \mathbb{Z}_p$. Compute $\zeta_1 = X(Mh_1^s) \in \mathbb{G}_1$, $\zeta_2 = (g h_2^e g_0^{m_1} \cdots g_L^{m_L})^{\frac{1}{p+e}} \in \mathbb{G}_1$ and $\zeta_3 = g^e \in \mathbb{G}_2$. Signature $\sigma$ on $M$ is $(\zeta_1, \zeta_2, \zeta_3, s, t) \in (\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^2)$.

Issue. This is a protocol between a signer with input $SK = (X, \mu)$ and user with input $M = (M, m_1, \ldots, m_L) \in \mathcal{M}$. Both parties have the common input $PK$.

1. User randomly chooses $s, t' \in_{R} \mathbb{Z}_p$, computes and sends $C_M = Mh_1^s \in \mathbb{G}_1$, $C_m = h_2^s g_0^{m_1} \cdots g_L^{m_L} \in \mathbb{G}_1$, along with the following proof3 to the signer.

$$\text{SPK}_{13}\{(s, t', m_1, \ldots, m_L) : C_m = h_2^s g_0^{m_1} \cdots g_L^{m_L}\}(R)$$

2. After verifying $\text{SPK}_{13}$, the signer chooses $t'' \in_{R} \mathbb{Z}_p$, computes $\zeta_1 = X(C_M)^e \in \mathbb{G}_1$, $\zeta_2 = (g g_0^{t''} C_m)^{\frac{1}{p+e}} \in \mathbb{G}_1$, $\zeta_3 = g^e \in \mathbb{G}_2$ and sends $(\zeta_1, \zeta_2, \zeta_3, t'')$ back to the user.

3This proof guarantees that $(C_M, C_m)$ is correctly formed to the signer. In fact $C_M$ is always formed correctly to the view of the signer as for any $s$, there exists an $M$ such that $C_M = Mh_1^s$. Here $R$ is a random challenge issued by the signer. Similar to that of BBS+, one could use $PK$ or $\text{SPK}$ here.
3. The user computes \( t = t' + t'' \mod p \) and stores the signature \( \sigma \) on \( M \) as \((s_1, s_2, s_3, s, t)\).

**Verify.** Given a public key \( PK \), a message \( M = (M, m_1, \ldots, m_L) \in \mathcal{M} \), a signature \( \sigma = (s_1, s_2, s_3, s, t) \), verify that
\[
\hat{e}(s_1, g) \overset{?}{=} \hat{e}(Mh_1^s, s_3) \\
\hat{e}(s_2, s_3Z) \overset{?}{=} \hat{e}(g^{s_2}h_0^t g_1^{m_1} \cdots g_L^{m_L}, g)
\]
The set of equations is also known as the verification equations. If the equality holds, output **valid**. Otherwise, output **invalid**.

**Prove.** This is a \( \Sigma \)-Protocol of the knowledge of a message-signature pair \((M = (M, m_1, \ldots, m_L), \sigma = (s_1, s_2, s_3, s, t))\), between a prover (user) and a verifier, such that the pair satisfies the set of verification equations. For the ease of representation, we let \( C_M = Mh_1^{r_1} \in \mathbb{G}_1 \), \( C_m = h_2^{r_2}g_0^{r_1}g_1^{m_1} \cdots g_L^{m_L} \in \mathbb{G}_1 \), which is a commitment of \( M \) using randomness \( r_1, r_2 \). We assume the prover wishes to prove he is in possession of a signature on a valued \( M \) committed to \((C_M, C_m)\). The protocol shall be abstracted as
\[
\text{PK}_{14}\left\{ \left( s_1, s_2, s_3, s, t, r_1, r_2, M, m_1, \ldots, m_L \right) : \\
\hat{e}(s_1, g) = \hat{e}(Mh_1^s, s_3) \land \\
\hat{e}(s_2, s_3Z) = \hat{e}(g^{s_2}h_0^t g_1^{m_1} \cdots g_L^{m_L}, g) \land \\
C_M = Mh_1^{r_1} \land \\
C_m = h_2^{r_2}g_0^{r_1}g_1^{m_1} \cdots g_L^{m_L} \right\}
\]
\( \text{PK}_{14} \) requires instantiation. The prover first computes \( \mathcal{A}_1 = s_1g_1^{r_3} \in \mathbb{G}_1 \), \( \mathcal{A}_2 = s_2g_1^{r_4} \in \mathbb{G}_1 \), \( \mathcal{A}_3 = s_3g_1^{r_5} \in \mathbb{G}_2 \), \( \mathcal{A}_4 = g_1^{r_6}g_2^{r_5} \in \mathbb{G}_1 \) for some randomly generated \( r_3, r_4, r_5, r_6 \in_R \mathbb{Z}_p \). Then he sends \((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4)\) to the verifier.
and conducts the following protocol $\text{PK}_{15}$.

\[
\text{PK}_{15}\left\{ (s, t, m_1, \ldots, m_L, r_1, r_2, r_3, r_4, r_5, r_6, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6 ) : \\
\begin{align*}
\mathcal{A}_4 &= g_1^{r_4}g_2^{r_5} \\
1_{G_1} &= \mathcal{A}_4^{-r_1}g_1^{\beta_1}g_2^{\beta_2} \\
1_{G_1} &= \mathcal{A}_4^{-s}g_1^{\beta_3}g_2^{\beta_4} \\
1_{G_1} &= \mathcal{A}_4^{-t}g_1^{\beta_5}g_2^{\beta_6} \\
\mathcal{C}_m &= h_2^{r_1}g_0^{r_1}g_1^{m_1} \ldots g_L^{m_L} \\
\frac{\mathcal{Z}e(\mathcal{C}_M, \mathcal{A}_3)}{\hat{e}(\mathcal{A}_1, g)} &= \hat{e}(\mathcal{G}_1, g_1)^{r_5} \hat{e}(h_1, g_1)^{\beta_1-\beta_2} \hat{e}(h_1, \mathcal{A}_3)^{r_1} \hat{e}(g_1, g)^{-r_3} \\
\frac{\hat{e}(\mathcal{A}_2, g_1)}{\hat{e}(g, g)} &= \hat{e}(\mathcal{A}_2, g_1)^{r_5} \hat{e}(g_1, \mathcal{A}_3)^{r_4} \hat{e}(g_1, g)^{-\beta_6} \hat{e}(g_0, g)^t
\end{align*}
\right\}
\]

(Commitment.) Prover randomly generates $\rho_s, \rho_t, \rho_m_1, \ldots, \rho_m_L, \rho_r_1, \rho_r_2, \\
\rho_r_3, \rho_r_4, \rho_r_5, \rho_r_6, \rho_\beta_1, \rho_\beta_2, \rho_\beta_3, \rho_\beta_4, \rho_\beta_5, \rho_\beta_6 \in R \mathbb{Z}_p$, computes $T_1 = \\
g_1^{r_6}g_2^{r_5}, \ T_2 = \mathcal{A}_4^{-r_1}g_1^{\beta_3}g_2^{\beta_4}, \ T_3 = \mathcal{A}_4^{-s}g_1^{\beta_4}g_2^{\beta_5}, \ T_4 = \mathcal{A}_4^{-t}g_1^{\beta_5}g_2^{\beta_6}, \\
T_5 = h_2^{r_1}g_0^{r_2}g_1^{m_1} \ldots g_L^{m_L} \in G_1$, and $T_6 = \hat{e}(\mathcal{C}_M, g_1)^{r_5} \hat{e}(h_1, g_1)^{\beta_1-\beta_2} \hat{e}(h_1, \mathcal{A}_3)^{r_1} \hat{e}(g_1, g)^{-r_3}$.

(Challenge.) Verifier chooses a random challenge $c \in R \{0, 1\}^l$ and sends $c$ to prover.

(Response.) Prover computes, in $\mathbb{Z}_p$, $z_s = \rho_s - cs, z_t = \rho_t - ct, z_m_1 = \\
\rho_m_1 - cm_1, \ldots, z_m_L = \rho_m_L - cm_L, z_r_1 = \rho_r_1 - cr_1, z_r_2 = \rho_r_2 - cr_2, z_r_3 = \\
\rho_r_3 - cr_3, z_r_4 = \rho_r_4 - cr_4, z_r_5 = \rho_r_5 - cr_5, z_r_6 = \rho_r_6 - cr_6, z_\beta_1 = \rho_\beta_1 - cr_1 r_6, \\
z_\beta_2 = \rho_\beta_2 - cr_1 r_5, z_\beta_3 = \rho_\beta_3 - c \beta r_6, z_\beta_4 = \rho_\beta_4 - c \beta r_5, z_\beta_5 = \rho_\beta_5 - c \beta r_4 r_6, \\
z_\beta_6 = \rho_\beta_6 - c \beta r_4 r_5$ and sends $(z_s, z_t, z_m_1, \ldots, z_m_L, z_r_1, z_r_2, z_r_3, z_r_4, z_r_5, z_r_6, \\
z_\beta_1, z_\beta_2, z_\beta_3, z_\beta_4, z_\beta_5, z_\beta_6)$ to verifier.

(Verify.) Verifier outputs accept if and only if the following seven equations
The AWSM problem is defined as follows: Let Definition 3.3 below that ESS+ is strongly unforgeable if the AWSM problem is hard.

More explicit, we define of the scheme to any existing hard problems. To make the security of our scheme and thus it only works in asymmetric pairing such that the SXDH assumption unforgeability. It is trivial to show that ESS+ possesses correctness. Below, we investigate the correctness of \( C \).

We put \( \hat{e}(g_1, g_2) = \hat{e}(g_1, g_2) \). The signature held by the prover is \((A_1, A_2, A_3) \). If we put \( \beta_6 = r_4 \), into the last equality, we are proving \( \hat{e}(A_2, g_1 r_4, Z A_3 g_1 r_3) = \hat{e}(A_2, g_1 r_4, Z A_3 g_1 r_3) \). The signature held by the prover is \((A_1, A_2, A_3) \). The signature held by the prover is \((A_1, A_2, A_3) \).

### 3.3.3 Security Analysis of ESS+

It is trivial to show that ESS+ possesses correctness. Below, we investigate the unforgeability, Issue-Privacy and Prove-Privacy.

**Unforgeability**

Firstly, ESS+ is forgeable if there exists any mapping from \( G_2 \) to \( G_1 \) (or vice versa) and thus it only works in asymmetric pairing such that the SXDH assumption holds. Even under this strong assumption, we are unable to reduce the security of the scheme to any existing hard problems. To make the security of our scheme more explicit, we define the following problem, called AWSM, and show in the proof below that ESS+ is strongly unforgeable if the AWSM problem is hard.

**Definition 3.3** The AWSM problem is defined as follows: Let \( G_1, G_2, G_T \) be some cyclic groups of prime order \( p \) with efficiently computable pairing \( \hat{e} : G_1 \times G_2 \rightarrow \).

\[
\begin{align*}
T_1 &= \mathcal{A}_1 g_1 \hat{e}(g_1, g_2) \\
T_2 &= \mathcal{A}_4 g_1 \hat{e}(g_1, g_2) \\
T_3 &= \mathcal{A}_4 g_1 \hat{e}(g_1, g_2) \\
T_4 &= \mathcal{A}_4 g_1 \hat{e}(g_1, g_2) \\
T_5 &= \mathcal{C}_M h_2 \hat{e}(g_1, g_2) \\
T_6 &= \left( \frac{Z \hat{e}(\mathcal{A}_M, \mathcal{A}_3)}{\hat{e}(\mathcal{A}_1, g)} \right)^g \hat{e}(\mathcal{C}_M, g_1)^z \hat{e}(h_1, g_1)^z \hat{e}(h_1, \mathcal{A}_3)^z \hat{e}(g_1, g)^z \hat{e}(g_0, g)^z \\
T_7 &= \left( \frac{\hat{e}(\mathcal{A}_2, Z \mathcal{A}_3)}{\hat{e}(g, g)} \right)^g \hat{e}(\mathcal{A}_2, g_1)^z \hat{e}(g_1, Z \mathcal{A}_3)^z \hat{e}(g_1, g_1)^z \hat{e}(h_2, g)^z \hat{e}(g_0, g)^z \\
\hat{e}(g_1, g)^z \hat{e}(g_2, g)^z \hat{e}(g_3, g)^z \cdots \hat{e}(g_m, g)^z.
\end{align*}
\]
3.3. ESS+ Signature

Let $Z \in_R \mathbb{G}_T$, $Z \in_R \mathbb{G}_2$, $g \in_R \mathbb{G}_1$, $g \in_R \mathbb{G}_2$ and an oracle $O$ that on input $(M_i) \in \mathbb{G}_1$, outputs $(a_{1,i}, a_{2,i}, a_{3,i}) \in \mathbb{G}_1^2 \times \mathbb{G}_2$ such that $\hat{e}(a_{1,i}, g) = Z \hat{e}(M_i, a_{3,i})$ and $\hat{e}(a_{2,i}, a_{3,i}Z) = \hat{e}(g, g)$.

The problem is on input $(G_1, G_2, G_3, \hat{e}, p, g, Z, Z, O)$, making at most $q$ queries to $O$, outputs $(M, a_1, a_2, a_3) \in \mathbb{G}_1^3 \times \mathbb{G}_2$ such that $\hat{e}(a_1, g) = Z \hat{e}(M, a_3)$ and $\hat{e}(a_2, a_3Z) = \hat{e}(g, g)$ with the restriction that $(M, a_1, a_2, a_3) \neq (M_i, a_{1,i}, a_{2,i}, a_{3,i})$ for $i = 1$ to $q$.

Proof: The proof is quite straightforward. Assume there exists a PPT adversary $A$ that could win in GAME CL-S-EU-CMA, we show how to construct a PPT simulator $S$ that solves the AWSM problem. $S$ is given the AWSM problem instance defined above (definition 3.3) and is required to output a tuple $(M, a_1, a_2, a_3)$.

Simulating the public parameters. $S$ first randomly chooses $\mu_j \in R \mathbb{Z}_p$ and sets $g_j = g^{\mu_j}$ for $j = 0$ to $L$. $S$ also randomly chooses $\beta_1, \beta_2 \in R \mathbb{Z}_p$ and set $h_1 = g^{\beta_1}$, $h_2 = g^{\beta_2}$. $S$ randomly generates an element $g_1 \in R \mathbb{G}_2$. The message space $\mathcal{M}$ is set to $G_1 \times Z_p^L$. Finally, $S$ gives $(\hat{e}, G_1, G_2, G_T, p, g, q_0, \ldots, q_L, h_1, h_2, g, g_1, Z, Z)$ to $A$ as the public key of the ESS+ signature.

Handling queries. $A$ is allowed to issue up to $q$ signature queries. If it is a Issue protocol query, $S$ needs to rewind $A$ and uses the knowledge extractor of $SPK_{13}$ to obtain the underlying block of messages $(s, t', m_1, \ldots, m_L)$ in the commitment $C_m$. If $SPK$ is used, $S$ is given access to the random oracle and the scheme shall be secure under the random oracle model. After running the knowledge extractor, invoke oracle $O$ on input $C_M$ and receives output $(a_{1,i}, a_{2,i}, a_{3,i})$. Compute $\varsigma_{2,i} = a_{2,i}^{1+\beta_2 s+\beta_1 t+\sum_{j=1}^L \mu_j m_j}$ for some randomly generated $t \in R \mathbb{Z}_p$. Return $(a_{1,i}, \varsigma_{2,i}, a_{3,i}, t-t')$ if it is in the Issue protocol.

In a normal Sign query, $A$ gives $S$ a message denoted as $(M_i, m_{1,i}, \ldots, m_{L,i})$. $S$ randomly generates $s_i$ and invoke oracle $O$ with input $M_i h_1^{s_i}$ and receives the output $(a_{1,i}, a_{2,i}, a_{3,i})$. Randomly generate $s_i, t_i \in R \mathbb{Z}_p$ and compute $\varsigma_{2,i} = a_{2,i}^{1+\beta_2 s_i+\beta_1 t_i+\sum_{j=1}^L \mu_j m_{j,i}}$. Return the signature to $A$ as $(a_{1,i}, \varsigma_{2,i}, a_{3,i}, s_i, t_i)$.

The reduction. Finally, $A$ outputs $q + 1$ signatures, $(\varsigma_{1,i}^*, \varsigma_{2,i}^*, \varsigma_{3,i}^*, s_i^*, t_i^*)$ and their corresponding messages $(M_i^*, m_{1,i}^*, \ldots, m_{L,i}^*)$, for $i = 1$ to $q + 1$. Compute $\bar{M}_i = M_i h_1^{s_i}$, $\bar{a}_{2,i} = (\varsigma_{2,i}^*)^{1+\beta_2 s_i^*+\beta_1 t_i^*+\sum_{j=1}^L \mu_j m_{j,i}^*}$. Set $\bar{A}_{1,i} = \varsigma_{1,i}^*$ and $\bar{A}_{3,i} = \varsigma_{3,i}^*$. At least one of
3.4 C-Signature

the \((\bar{M}_i, \bar{a}_{1,i}, \bar{a}_{2,i}, \bar{a}_{3,i})\) is different from the input-output pair of oracle \(O\) and thus \(S\) has solved the AWSM problem. \(\square\)

Issue-Privacy

Issue-Privacy of ESS+ signature is straightforward. Note that during Issue protocol, the signer only obtains the commitment of the block of messages \((C_M, C_m)\) along with zero-knowledge proof-of-knowledge \(\text{SPK}_{13}\). Due to the unconditional hiding property of the commitment scheme and the zero-knowledgeness of the \(\text{SPK}\), the signer learns nothing about the message being signed.

Prove-Privacy

Prove-Privacy of ESS+ signature is also straightforward. The proof that \(\text{PK}_{15}\) is a \(\Sigma\)-Protocol is omitted as it is trivial. Moreover, the values \(C_M, C_m, A_1, A_2, A_3\) and \(A_4\) leak no information about the message nor the signature. Thus, Prove protocol leaks no information about the message nor the signature.

3.4 C-Signature

Existing CL-signatures are not proven secure under concurrent execution of the Issue protocol. Consequently, systems built on CL-signatures typically require a sequential operation in some stage (normally the joining stage). For example, clients in the e-cash systems are required to withdraw money one after another. Group manager in anonymous credential systems, \(k\)-TAA can only register users one by one. This is sometimes undesirable as the sequential stage can easily become the bottleneck and it also makes the system vulnerable to denial-of-service attacks. C-Signature is an attempt to address this shortcoming.

One possible reason for the insecurity of Issue protocol under concurrent execution in existing CL-signatures is that it is hard to simulate the protocol without knowing the message to be signed. Thus, existing schemes incorporate proof-of-knowledge of the commitment during the Issue protocol (say, \(\text{SPK}_{10}\) for BBS+ and \(\text{SPK}_{13}\) for ESS+) and security proofs rely on the knowledge extractor of the corresponding proof-of-knowledge protocols which involves rewinding the adversary to extract the message to be signed. After the message is known, the simulator simulates as if it is a normal signing query. As each Issue query requires one rewind,
the number of rewinding will be exponential to the number of queries if they are allowed to be executed concurrently.

**Overview of Our C-Signature Construction.** We employ a totally different approach. Let \((\text{Gen}, \text{Commit})\) be some commitment scheme. Let \(\text{Sig} = (\text{KeyGen}, \text{Sign}, \text{Verify})\) be a secure digital signature scheme. If the space of the commitment \(C = \text{Commit}(\text{param}, x; r)\) is always within the message space of \(\text{Sig}\), we can define a new signature scheme \(\hat{\text{Sig}}\) as follows.

\text{KeyGen}: Invoke \text{KeyGen} to obtain \((PK, SK)\). Invoke \text{Gen} of the commitment scheme to obtain \text{param}. Output \(\hat{PK} = (PK, \text{param})\) and \(\hat{SK} = SK\).

\text{Sign}: On input message \(m\), randomly generate \(r\) and compute \(C = \text{Commit}(\text{param}, m; r)\). Invoke \(\sigma \leftarrow \text{Sign}_{SK}(C)\). Output \(\hat{\sigma} = (\sigma, r)\).

\text{Verify}: On input \((m, \hat{\sigma} = (\sigma, r))\), output \text{valid} if and only if

\[
\text{valid} \leftarrow \text{Verify}_{PK}(\text{Commit}(\text{param}, m; r), \sigma)
\]

We can easily define a two-move \text{Issue} protocol for \(\hat{\text{Sig}}\). Upon receiving a commitment \(C\) from the user, the signer returns \(\sigma \leftarrow \text{Sign}_{SK}(C)\). The user parse \(\hat{\sigma}\) as \((\sigma, r)\) where \(r\) is the randomness used to compute commitment \(C\). If \(S\) is a secure digital signature scheme and \text{Commit} is a secure commitment scheme, \(\hat{\text{Sig}}\) will be a secure digital signature scheme with concurrently-secure \text{Issue} protocol.

Next we are going to select a suitable signature scheme \(\text{Sig}\) and commitment scheme. In particular, we are required to select schemes so that efficient zero-knowledge proof-of-knowledge of message-signature pair can be constructed for the resulting scheme. This naturally eliminates a lot of efficient signature schemes that involve the use of hash function.

We choose Pedersen Commitment \cite{Ped91} which commits values from \(\mathbb{Z}_q^*\) for some prime \(q\). Its space is \(\mathbb{Z}_p^*\) for some prime \(p\) such that \(q\) divides \(p - 1\). There are different choices for the signature scheme \(\text{Sig}\) with message space \(\mathbb{Z}_p^*\). For instance, CL+ signature, BBS+ signature and the \(q\)-SDH assumption based Boneh-Boyen short signature \cite{BB04} (BB-signature). We choose BB-signature as it gives the shortest signature. We would like to stress that CL+ or BBS+ signature could also be used. However, it seems a bit tricky to claim achieving concurrently-secure \text{Issue}
when the resulting scheme depends on an interactive oracle based assumption in the case of CL+ (the LRSW assumption).

The choice of signature scheme and commitment scheme gives rise to a challenge. Specifically, we are required to conduct zero-knowledge proof-of-knowledge of representation of a committed element. We develop a zero-knowledge protocol for our purpose and the resulting protocol is presented below. This protocol can be thought of as an extension of the zero-knowledge protocol for double-discrete logarithm [CS97] and is compatible with common zero-knowledge protocol for demonstrating relationship amongst discrete logarithms, thus answering the question whether one could demonstrate relationships amongst double discrete logarithms. We believe this protocol could be of independent interest.

### 3.4.1 Proof-of-Knowledge of Representation of a Committed Value

We employ the following notation throughout this section. Let $G_1$ be a cyclic group of prime order $p$. Let $G_q \subset \mathbb{Z}_p^*$ be a cyclic group of prime order $q$. Let $g, g_0, g_1, g_2 \in R$ $G_1$ be random elements of $G_1$ and $h, h_0, h_1, \ldots, h_L \in R$ $G_q$ be random elements of $G_q$. A representation of an element $M \in G_q$ is an ordered set $(m_1, \ldots, m_L) \in (\mathbb{Z}_q)^L$ such that $M = h^{m_1} \cdots h_L^{m_L}$. As discussed, it is hard to come up with two different representations of an element in $G_q$ if relative discrete logarithms amongst any of $h, h_0, \ldots, h_L$ are not known.

Let $(m_1, \ldots, m_L)$ be a representation of $M$ to bases $h_1, \ldots, h_L$. That is, $M = h_1^{m_1} \cdots h_L^{m_L}$. Let $C = g_0^M g^r \in G_1$ be the commitment of $M$. Let $D = h^s h_1^{m_1} \cdots h_L^{m_L} \in G_q$ be the commitment of $(m_1, \ldots, m_L)$. We would like to derive a Σ-Protocol that the prover knows a representation, $(m_1, \ldots, m_L)$, committed to $D$ of a value, $M$, committed to $C$, without revealing anything about $M$ or $(m_1, \ldots, m_L)$.

We call such a protocol a ZKPoK of Representation of Committed Value (RCV Protocol). The prover is to prove to a verifier, in zero-knowledge manner, the following, in abstract level:

$$\text{PK}_{\text{RCV}} \left\{ \left( M, r, s, m_1, \ldots, m_L \right) : 
\begin{align*}
C &= g_0^M g^r \\
D &= h_1^{m_1} \cdots h_L^{m_L} h^s \\
M &= h_1^{m_1} \cdots h_L^{m_L}
\end{align*}
\right\}$$
Actual Construction of $PK_{RCV}$. The actual construction of RCV Protocol consists of two parts. Looking ahead, the role of D is for compatibility with the common ZKPoK of relations amongst discrete logarithm in $G_q$. If the goal is to demonstrate the knowledge of representation of $M$ committed to $C$ alone, the parts related to D could be removed.

\[
PK_{RVC}\left\{\begin{array}{l}
PK_{16}\left\{(M, r_1) : C = g_0^M g^{r_1}\right\} \\
PK_{17}\left\{(s, r_2, m_1, \ldots, m_L) : C = g_0^{h_1^{m_1} \cdots h_L^{m_L}} g^{r_2} \land D = h^s h_1^{m_1} \cdots h_L^{m_L}\right\}
\end{array}\right.
\]

We use the notation $r_1$ and $r_2$ to represent the same variable $r$ because in $PK_{16}$ and $PK_{17}$, no explicit checks are made to ensure they are the same. Indeed, such check is not necessary.

Roughly speaking, the relationship $M = h_1^{m_1} \cdots h_L^{m_L}$ is implied by the proof that $C = g_0^M g^{r_1}$ in $PK_{16}$ and $C = g_0^{h_1^{m_1} \cdots h_L^{m_L}} g^{r_2}$ in $PK_{17}$. The actual argument will be presented when we prove the soundness of the RCV Protocol.

Let $\lambda_k$ be a security parameter. In practice, we suggest $\lambda_k$ should be at least 80. $PK_{16}$ can be constructed using standard techniques. For completeness, it is stated below.

(Commitment.) The prover randomly generates $\rho_M, \rho_r \in \mathbb{Z}_p$, computes and sends
\[
T = g_0^{\rho_M} g^{\rho_r}
\]
to the verifier.

(Challenge.) The verifier returns a random challenge $c \in \mathbb{R} \{0, 1\}^{\lambda_k}$.

(Response.) The prover, treating $c$ as an element in $\mathbb{Z}_q$, computes $z_M = \rho_M - cM \in \mathbb{Z}_p$, $z_r = \rho_r - cr \in \mathbb{Z}_p$ and returns $(z_M, z_r)$ to the verifier.

(Verify.) Verifier accepts if and only if $T = C^c g_0^{z_M} g^{z_r}$.

$PK_{17}$ is more involved and can be thought of as the extension of the ZKPoK of double-discrete logarithm in combination with ZKPoK of equality of discrete logarithm.

(Commitment.) For $i = 1$ to $\lambda_k$, the prover randomly generates $\rho_{m_1, i}, \ldots, \rho_{m_L, i}$, $\rho_{s, i} \in \mathbb{Z}_q$ and $\rho_{r, i} \in \mathbb{Z}_p$. Then the prover computes
\[
T_{1,i} = g_0^{h_1^{\rho_{m_1, i}} \cdots h_L^{\rho_{m_L, i}}} g^{\rho_{r, i}} \in \mathbb{G}_1 \quad \text{and} \quad T_{2,i} = h_1^{\rho_{m_1, i}} \cdots h_L^{\rho_{m_L, i}} h^{\rho_{s, i}} \in \mathbb{G}_q.
\]
After that, the prover sends $(T_{1,i}, T_{2,i})_{i=1}^{\lambda_k}$ to the verifier.

\footnote{Consequently, the bit-length of $p$ should be longer than $\lambda_k$.}
(Challenge.) The verifier returns a random challenge $c \in \mathbb{R} \{0,1\}^{\lambda k}$.

(Response.) Denote $c[i]$ as the $i$-th bit of $c$. That is, $c[i] \in \{0,1\}$. For $i = 1$ to $\lambda k$, the prover computes $z_{m_1,i} = \rho_{m_1,i} - c[i]m_1 \in \mathbb{Z}_q$, $\ldots$, $z_{m_L,i} = \rho_{m_L,i} - c[i]m_L \in \mathbb{Z}_q$, $z_{s,i} = \rho_{s,i} - c[i]s \in \mathbb{Z}_q$ and $z_{r,i} = \rho_{r,i} - c[i]h_1^{z_{m_1,i}} \cdots h_L^{z_{m_L,i}}r \in \mathbb{Z}_p$. The prover sends $(z_{m_1,i}, \ldots, z_{m_L,i}, z_{s,i}, z_{r,i})^{\lambda k}_{i=1}$ to the verifier.

(Verify.) The verifier accepts if the following equations hold for $i = 1$ to $\lambda k$.

$$T_{2,i} = g^{{z_{m_1,i}}_{\lambda k} \cdots {z_{m_L,i}}_{\lambda k} h_{z_{s,i}}}$$
$$T_{1,i} = g_{c[i]}^{h_{0}^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{z_{r,i}}}$$
$$T_{1,i} = g_{c[i]}^{h_{1}^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{z_{r,i}}}$$

Note that $\text{PK}_{16}$ and $\text{PK}_{17}$ can be executed in parallel using the same challenge. Since $\text{PK}_{\text{RCV}}$ is non-standard, we prove it is an $\Sigma$-Protocol by constructing an efficient knowledge extractor $\text{KE}$ and zero-knowledge simulator $\text{KS}$ below.

**Soundness of RCV Protocol**

We construct a knowledge extractor $\text{KE}$ for RCV Protocol. Upon receiving $T$, $(T_{1,i}, T_{2,i})^{\lambda k}_{i=1}$, $\text{KE}$ generates $c \in \mathbb{R} \{0,1\}^{\lambda k}$ and sends it to the prover. The prover returns with $z_M$, $z_r$, $(z_{m_1,i}, \ldots, z_{m_L,i}, z_{r,i}, z_{s,i})^{\lambda k}_{i=1}$. The extractor rewinds the prover, issues another challenge $\hat{c} \in \mathbb{R} \{0,1\}^{\lambda k}$ and receives another set of responses $\hat{z}_M$, $\hat{z}_r$, $(\hat{z}_{m_1,i}, \ldots, \hat{z}_{m_L,i}, \hat{z}_{r,i}, \hat{z}_{s,i})^{\lambda k}_{i=1}$.

Since $T = C^{c[t]}g_0^{z_M}g^{\hat{z}_r}$ and $T = C^{\hat{c}[t]}g_0^{z_M}g^{\hat{z}_r}$, we have $C^{c[t]} - C^{\hat{c}[t]} = g_0^{z_M - \hat{z}_M}g^{\hat{z}_r - \hat{z}_r}$. Denote $\delta_c$ as $c - \hat{c}$, $\delta_M = z_M - \hat{z}_M$ and $\delta_r = z_r - \hat{z}_r$. The simulator obtains a relation $C = g_0^{M} g^{\tilde{r}}$, where $\tilde{M} = -\delta_M/\delta_c$ and $\tilde{r} = -\delta_r/\delta_c$.

On the other hand, as $c \neq \hat{c}$, there exists a position $i$ such that $c[i] \neq \hat{c}[i]$. Without loss of generality, assume $c[i] = 0$ and $\hat{c}[i] = 1$. We have $T_{2,i} = h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{z_{s,i}}$ and $T_{2,i} = D h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{z_{s,i}}$. Thus, $D = h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{\delta s}$, where $\delta_{m_1} = z_{m_1,i} - \hat{z}_{m_1,i}$, $\ldots$, $\delta_{m_L} = z_{m_L,i} - \hat{z}_{m_L,i}$ and $\delta_{s} = z_{s,i} - \hat{z}_{s,i}$.

We also have $T_{1,i} = g_0^{h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{z_{r,i}}}$ and $T_{1,i} = C h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} g^{\hat{z}_{r,i}}$. Substituting $C = g_0^{M} g^{\tilde{r}}$ into the equation, we have $g_0^{h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}} h_{z_{r,i}}} = g_0^{h_1^{\delta_{m_1} \cdots h_{L}^{\delta_{m_L}}} h_{\delta_{s}}} g^{\hat{z}_{r,i}}$, where $\delta_{r,i}$ is defined as $z_{r,i} - \hat{z}_{r,i}$. Under the discrete logarithm assumption in $\mathbb{G}_1$, $\tilde{M} = h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}}$ and $\tilde{r} = \delta_{r,i}/(h_1^{z_{m_1,i}} \cdots h_{L}^{z_{m_L,i}})$. 


Thus, the extractor KE has successfully extracted a value \( \tilde{M} \), whose representation is \( \delta_{m_1}, \ldots, \delta_{m_L} \) such that \( C \) is a commitment of \( M \) (with opening \( = \tilde{r} \)) and \( D \) is a commitment of \( (\delta_{m_1}, \ldots, \delta_{m_L}) \) (with opening \( = \delta_s \)). It implies that the RCV Protocol protocol is sound.

**Honest Verifier Zero-Knowledgeness of RCV Protocol**

We construct a zero-knowledge simulator KS for RCV Protocol that, on input a random challenge \( c \), outputs a transcript which is indistinguishable from the actual transcript of a real protocol run.

For a given commitments \( C \) and \( D \) and a random challenge \( c \in \{0,1\}^{\lambda_k} \), the simulator randomly generates \( z_M, z_r \in \mathbb{Z}_p \) and for \( i = 1 \) to \( \lambda_k \), \( z_r \in \mathbb{Z}_p \), \( z_{s,i} \in \mathbb{Z}_q \) and \( z_{m_1,i} \in \mathbb{Z}_q, \ldots, z_{m_L,i} \in \mathbb{Z}_q \). Next, it computes \( T = C^c g_0^{z_M} g^{z_r}, T_{2,i} = D^{c[i]} h_1^{z_{m_1,i}} \cdots h_L^{z_{m_L,i}} h_{z,s,i} \) and \( T_{1,i} = g_0^{h_1^{z_{m_1,i}} \cdots h_L^{z_{m_L,i}}} g^{z_r,i} \) if \( c[i] = 0 \) or \( T_{1,i} = C^{\gamma_1^{z_{m_1,i}} \cdots h_L^{z_{m_L,i}}} g^{z_r,i} \) if \( c[i] = 1 \). Finally, it outputs \((T, c[z_M, z_r])\) as a transcript of PK_{16} and \((T_{1,i}, T_{2,i}, c[i], z_{m_1,i}, \ldots, z_{m_L,i}, z_{s,i}, z_{r,i})\) as a transcript of PK_{17}.

It is straightforward to show that the distribution of the simulated transcript is indistinguishable from a real transcript.

### 3.4.2 Construction of a C-Signature

**KeyGen.** On input security parameter \( 1^\lambda \), generate a bilinear map \( \hat{e} : G_1 \times G_1 \rightarrow G_T \) such that \( |G_1| = |G_T| = p \) for some \( \lambda \)-bit prime \( p \). Let \( G_q \) be some cyclic group of prime order \( q \) such that \( G_q \subset G_1^* \). This can be done by setting \( p \) to be a prime of the form \( p = \gamma q + 1 \) for some integer \( \gamma \) and let \( G_q \) to be a cyclic group generated by an element of order \( q \) in \( G_1^* \).

Let \( g, g_1, g_2 \) be generators of \( G_1 \). Let \( h, h_0, h_1, \ldots, h_L \in R \mathbb{G}_q \) be generators of \( \mathbb{G}_q \). \( L \) is defined by the signer as the maximum size of the block of messages to be signed.

Let \( x, y \in R \mathbb{Z}_p \) and \( u = g^x \in G_1 \) and \( v = g^y \in G_1 \). The secret key \( SK \) is \((x, y)\) while the public key \( PK \) is \((\hat{e}, G_1, G_2, p, g, g_1, g_2, u, v, G_q, q, h, h_0, h_1, \ldots, h_L)\). The message space is \( M = \mathbb{Z}_q^L \) for some (possibly small) positive integer \( L \).

**Sign.** Given \( SK = (x, y) \) and message \( m = (m_1, \ldots, m_L) \in \mathbb{Z}_q^L \), pick a random \( s \in R \mathbb{Z}_q \). Compute \( M = h_0^{m_1} h_1^{m_2} \cdots h_L^{m_L} \in G_1 \). Then pick a random \( t \in R \mathbb{Z}_p \), compute \( \varsigma = g^{x + m_1 + \gamma y} \). The signature \( \sigma \) on \( m \) is \((\varsigma, s, t) \in (G_1 \times \mathbb{Z}_q \times \mathbb{Z}_p)\).
3.4. C-Signature

**Issue.** This is a protocol between a signer with input $SK = (x, y)$ and user with input $m = (m_1, \ldots, m_L) \in \mathbb{Z}_q^L$. Both parties have the common input $PK$.

1. User randomly chooses $s \in R \mathbb{Z}_q$, computes and sends $M = h_0^s h_1^{m_1} \cdots h_L^{m_L} \in \mathbb{G}_q$ to the signer.
2. Signer picks a random $t \in R \mathbb{Z}_p$, computes $\varsigma = g^{r + s \cdot f}$. Signer returns $(\varsigma, t) \in (\mathbb{G}_1 \times \mathbb{Z}_p)$ to the user.
3. The user parses the signature $\sigma$ on $m$ as $(\varsigma, s, t)$.

**Verify.** Given a public key $PK$, a message $m = (m_1, \ldots, m_L) \in \mathbb{Z}_q^L$, a signature $\sigma = (\varsigma, s, t)$, verifies that

$$\hat{e}(\varsigma, u g^{h_0 h_1^{m_1} \cdots h_L^{m_L} v^t}) \overset{?}{=} \hat{e}(g, g)$$

The equation is also known as the verification equation. If the equality holds, output valid. Otherwise, output invalid.

**Prove.** This is a $\Sigma$-Protocol of the knowledge of a message-signature pair $(m = (m_1, \ldots, m_L), \sigma = (\varsigma, s, t))$, between a prover (user) and a verifier, such that the pair satisfies the verification equation. For the ease of representation\footnote{One could say in the actual instantiation that $\mathcal{C}_m$ is not necessary as one of the auxiliary commitments contains the commitment of $m$. However, it is kept in the main text for the ease of representation.}, we let $\mathcal{C}_m = h_0^s h_1^{m_1} \cdots h_L^{m_L} \in \mathbb{G}_q$, which is a commitment of $m$ using randomness $r$. We assume the prover wishes to prove he is in possession of a signature on a valued $m$ committed to $\mathcal{C}_m$. The protocol shall be abstracted as

$$\text{PK}_{18}\left\{\left(\varsigma, s, t, m_1, \ldots, m_L, r\right) : \begin{array}{c}
\hat{e}(\varsigma, u g^{h_0 h_1^{m_1} \cdots h_L^{m_L} v^t}) = \hat{e}(g, g) \\
\mathcal{C}_m = h_0^s h_1^{m_1} \cdots h_L^{m_L} \end{array}\right\}$$

$\text{PK}_{18}$ requires instantiation. The prover first computes $\mathcal{C}_1 = g_1^{h_0 h_1^{m_1} \cdots h_L^{m_L} r_1} \in \mathbb{G}_1$ for some randomly generated $r_1 \in R \mathbb{Z}_p$, $\mathcal{C}_2 = h_2^{r_2 h_0 h_1^{m_1} \cdots h_L^{m_L}} \in \mathbb{G}_q$ for some randomly generated $r_2 \in R \mathbb{Z}_q$. Next he conducts the following $\Sigma$-Protocol $\text{PK}_{19}$, which is a parallel execution of a $\text{PK}_{RCV}$, $\text{PK}_{20}$ and $\text{PK}_{21}$. Here we notice that $\mathcal{C}_m$ is not necessary as its role is taken place by the auxiliary commitment $\mathcal{C}_2$. The inclusion of $\mathcal{C}_m$ helps preserve charity, while decreases efficiency as...
one has to execute $\mathsf{PK}_{20}$ to demonstrate that it is correctly formed with respect to $\mathcal{C}_2$.

$$\mathsf{PK}_{19} \left\{ \begin{array}{l}
\mathsf{PK}_{\text{RCV}} \left\{ (M, r_1, r_2, s, m_1, \ldots, m_L) : \\
\quad \mathcal{C}_1 = g_1^M g_2^{r_1} \land \mathcal{C}_2 = h^r h_0^s h_1^{m_1} \ldots h_L^{m_L} \land \\
\quad M = h_0^s h_1^{m_1} \ldots h_L^{m_L} \end{array} \right. \\
\mathsf{PK}_{20} \left\{ (r, s, r_2, m_1, \ldots, m_L) : \\
\quad \mathcal{C}_2 = h^r h_0^s h_1^{m_1} \ldots h_L^{m_L} \land \\
\quad \mathcal{C}_m = h_0^r h_1^{m_1} \ldots h_L^{m_L} \end{array} \right. \\
\mathsf{PK}_{21} \left\{ (\varsigma, M, r_1, t) : \\
\quad \mathcal{C}_1 = g_1^M g_2^{r_1} \land \hat{e}(\varsigma, u g^M v^t) = \hat{e}(g, g) \right. \}$$

$\mathsf{PK}_{\text{RCV}}$ has been discussed, and $\mathsf{PK}_{20}$ can be constructed using techniques of equality of discrete logarithm and is thus omitted. $\mathsf{PK}_{21}$, while can be constructed using similar techniques as $\mathsf{PK}_{11}$, requires instantiation. For completeness, its instantiation is shown below. The prover randomly generates $r_3, r_4 \in_R \mathbb{Z}_p$ and computes $\mathfrak{A}_1 = g_1^{r_3} g_2^{r_4}$, $\mathfrak{A}_2 = s g_2^{r_4}$. The prover sends ($\mathfrak{A}_1$, $\mathfrak{A}_2$) to the verifier, along with the following $\Sigma$-Protocol $\mathsf{PK}_{22}$.

$$\mathsf{PK}_{22} \left\{ \left( t, M, r_1, r_3, r_4, \beta_1, \beta_2, \beta_3, \beta_4 \right) : \\
\quad \mathfrak{A}_1 = g_1^{r_3} g_2^{r_4} \land \\
\quad 1_{G_1} = \mathfrak{A}_1^{-M} g_1^{\beta_1} g_2^{\beta_2} \land \\
\quad 1_{G_1} = \mathfrak{A}_1^{t} g_1^{\beta_3} g_2^{\beta_4} \land \\
\quad \mathcal{C}_1 = g_1^M g_2^{r_1} \land \\
\quad \hat{e}(\mathfrak{A}_{2, u}) \hat{e}(g, g) = \hat{e}(g_2, u)^{r_3} \hat{e}(g_2, g)^{\beta_3} \hat{e}(g_2, v)^{\beta_3} \hat{e}(\mathfrak{A}_2, g)^{−M} \hat{e}(\mathfrak{A}_2, v)^{−t} \right. \}$$

(Commitment.) The prover randomly generates $\rho_t$, $\rho_M$, $\rho_{r_1}$, $\rho_{r_3}$, $\rho_{r_4}$, $\rho_{\beta_1}$, $\rho_{\beta_2}$, $\rho_{\beta_3}$, $\rho_{\beta_4}$ $\in_R \mathbb{Z}_p$ and computes, in $G_1$, $T_1 = g_1^{\rho_{r_3}} g_2^{\rho_{r_4}}$, $T_2 = \mathfrak{A}_1^{−\rho_M} g_1^{\rho_{\beta_1}} g_2^{\rho_{\beta_2}}$, $T_3 = \mathfrak{A}_1^{−\rho_t} g_1^{\rho_{\beta_3}} g_2^{\rho_{\beta_4}}$, $T_4 = g_1^{\rho_M} g_2^{\rho_{r_1}}$, and $T_5 = \hat{e}(g_2, u)^{\rho_{r_3}} \hat{e}(g_2, g)^{\rho_{\beta_3}} \hat{e}(g_2, v)^{\rho_{\beta_3}} \hat{e}(\mathfrak{A}_2, g)^{−\rho_M} \hat{e}(\mathfrak{A}_2, v)^{−\rho_t} \in G_T$. The prover sends ($T_1$, $T_2$, $T_3$, $T_4$, $T_5$) to the verifier.

(Challenge.) The verifier returns a random challenge $c \in_R \{0, 1\}^{\lambda_k}$. 
(Response.) The prover computes, in $\mathbb{Z}_p$, $z_t = \rho_t - ct$, $z_M = \rho_M - ch_0 h_1^{m_1} \cdots h_L^{m_L}$, $z_{r_1} = \rho_{r_1} - cr_1$, $z_{r_3} = \rho_{r_3} - cr_3$, $z_{r_4} = \rho_{r_4} - cr_4$, $z_{r_4} = \rho_{r_4} - cr_4$, $z_{r_4} = \rho_{r_4} - cr_4$, $z_{r_4} = \rho_{r_4} - cr_4$, and returns $(z_t, z_M, z_{r_1}, z_{r_3}, z_{r_4}, z_{r_4}, z_{r_4}, z_{r_4}, z_{r_4})$ to the verifier.

(Verify.) Verifier accepts if and only if all the following holds.

\[
T_1 = g_1^{z_{r_3}} g_2^{z_{r_4}}
\]
\[
T_2 = g_1^{-z_M} g_1^{z_{r_1}} g_2^{z_{r_2}}
\]
\[
T_3 = g_1^{-z_t} g_1^{z_{r_3}} g_2^{z_{r_4}}
\]
\[
T_4 = g_1^{z_M} g_1^{z_{r_1}}
\]
\[
T_5 = [\frac{e(\mathcal{A}_2, u)}{e(g, g)}]^c g_2^{z_{r_3}} e(g_2, g)^{z_{r_1}} e(g_2, v)^{z_{r_4}} e(\mathcal{A}_2, g)^{-z_M} e(\mathcal{A}_2, v)^{-z_t}
\]

### 3.4.3 Security Analysis of C-Signature

It is trivial to show that our C-Signature possesses correctness. Below we investigate the unforgeability, Issue-Privacy and Prove-Privacy.

**Unforgeability**

Our C-signature is strongly unforgeable when the $q$-SDH problem in bilinear group is hard and the discrete logarithm in $\mathbb{G}_q$ is hard in the standard model. The proof is by reduction.

**Proof:** Suppose there exists an PPT adversary $\mathcal{A}$ which could win GAME CL-S-EU-CMA with non-negligible advantage and assume it makes less than or equal to $q$ signature queries. We construct a PPT simulator $\mathcal{S}$ that breaks the strong unforgeability of the BB signature or solves DLP in $\mathbb{G}_q$. Since BB signature is strongly unforgeable under the $q$-SDH assumption in the standard model, we can safely conclude that our C-Signature is S-EU-CMA.

**Simulating the public parameters.** We assume the common parameter are $\mathbb{G}_1$, $\mathbb{G}_T$, $\hat{e}$ such that $|\mathbb{G}_1| = |\mathbb{G}_T| = p$ for some prime $p$, $\mathbb{G}_q \subseteq \mathbb{Z}_q^*$ such that $|\mathbb{G}_q| = q$ for some prime $q$. $\mathcal{S}$ is given public key of a BB-signature $(g, u, v)$ and is given access to a BB-signature oracle $O_{BB}$ which, on input a message $M \in \mathbb{Z}_p^*$, outputs a BB-signature $\sigma = (A, r)$ such that $\hat{e}(A, u g^M v^r) = \hat{e}(g, g)$. The goal of the simulator
3.4. C-Signature

\( S \) is to output a new message-signature pair \( M^*, (A^*, r^*) \) which is not equal to any input-output pair of \( O_{BB} \).

Let the message space \( \mathcal{M} \) be \( \mathbb{Z}_q^L \). \( S \) randomly generates \( h, h_0, h_1, \ldots, h_L \in \mathbb{G}_q \) and \( g_1, g_2 \in \mathbb{G}_1 \). \( S \) gives \((\hat{e}, \mathbb{G}_1, \mathbb{G}_2, p, g, g_1, g_2, u, v, \mathbb{G}_q, q, h_0, h_1, \ldots, h_L)\) to \( A \) as the public key of the C-Signature.

Handling queries. \( A \) is allowed to issue up to \( q \) signature queries. If it is a Issue protocol query, \( S \) receives \( M \in \mathbb{G}_q^6 \) as a commitment from \( A \). \( S \) invokes \( O_{BB} \) and obtains \((A, r)\). Parse \( \varsigma = A \) and \( t = r \) and return \((\varsigma, t)\) to complete the protocol.

In a normal Sign query, \( A \) gives \( S \) a message denoted as \( m = (m_1, \ldots, m_L) \in \mathbb{Z}_q^L \). \( S \) randomly generates \( s \) and invokes oracle \( O_{BB} \) with input \( h_s^0 h_1^{m_1} \cdots h_L^{m_L} \in \mathbb{G}_q \) and receives the output \((A, r)\). Parse \( \varsigma = A \) and \( t = r \) and return \((\varsigma, s, t)\) to \( A \) as the signature on \( m \).

The reduction. Finally, \( A \) outputs \( q + 1 \) distinct messages-signature pairs, denoted as \((m_i, \sigma_i)\) with \( m_i = (m_{1,i}, \ldots, m_{L,i}) \) and \( \sigma_i = (\varsigma_i, s_i, t_i) \). Denote \( M_i = h_0^s h_1^{m_{1,i}} \cdots h_L^{m_{L,i}} \).

There are two possibilities. If \( M_i \neq M_j \) for any \( i \neq j \), then at least one of the message-signature pair consists of an input \( M_i \) which has never been input to oracle \( O_{BB} \). \( S \) outputs \((M_i, \varsigma_i, t_i)\) as a new message-signature pair as the forged BB-signature.

Now if \( M_i = M_j \) for some \( i = j \), there are two possibilities. If \( t_i \neq t_j \), it implies the breaking of strong unforgeability of BB-signature.

If \( t_i = t_j \), it implies \((s_i, m_{1,i}, \ldots, m_{L,i}) \neq (s_j, m_{1,j}, \ldots, m_{L,j})\) and thus \( S \) is able to come up with two different representations of the same element in \( \mathbb{G}_q \), which implies a solution of DLP in \( \mathbb{G}_q \).

Note that during the proof, the adversary is allowed to issue its query in concurrent manner. Thus, our C-Signature is concurrent strongly unforgeable under the chosen message attack. \( \square \)

Issue-Privacy

Issue-Privacy of our C-Signature is straightforward. Note that during Issue protocol, the signer only obtains the commitment of the block of messages. Due to the
unconditional hiding property of the commitment scheme, the signer learns nothing about the message being signed.

Prove-Privacy

Prove-Privacy of our C-Signature is also straightforward as $\mathsf{PK}_{19}$ consists of an RCV Protocol protocol and two standard $\Sigma$-Protocols. It has been proven that $\mathsf{PK}_{RCV}$ is a $\Sigma$-Protocol. Thus, $\mathsf{PK}_{19}$ is a $\Sigma$-Protocol too.

3.5 Chapter Summary

In this chapter, we proposed three secure CL-signatures, namely, BBS+, ESS+ and C-Signature, with different properties. Our proposals are capable of signing block of messages and are strongly unforgeable. That is, adversary cannot produce a new signature even for a previously signed message [ADR02]. The following table summarises the properties of our constructions, as well as other existing CL-signatures. In the table, Strongly UF means strongly unforgeable, “?” represents the scheme probably satisfies the property but no formal security analysis is given. Assume the block of message being signed is of length $L$. The size of the signature are taken so that they their security is roughly equal to that of a 1024-bit standard RSA signature.

<table>
<thead>
<tr>
<th></th>
<th>Strongly UF</th>
<th>Concurrent Issue?</th>
<th>Size(bit)</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBS+</td>
<td>✓</td>
<td>×</td>
<td>511</td>
<td>$q$-SDH</td>
</tr>
<tr>
<td>ESS+</td>
<td>✓</td>
<td>×</td>
<td>1192</td>
<td>AWSM</td>
</tr>
<tr>
<td>C-Signature</td>
<td>✓</td>
<td>✓</td>
<td>2218</td>
<td>$q$-SDH</td>
</tr>
<tr>
<td>CL</td>
<td>?</td>
<td>×</td>
<td>2532</td>
<td>SRSA</td>
</tr>
<tr>
<td>CL+</td>
<td>?</td>
<td>×</td>
<td>$1536 + 1024L$</td>
<td>LRSW</td>
</tr>
<tr>
<td>P-Signature</td>
<td>?</td>
<td>×</td>
<td>850</td>
<td>$q$-SDH</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of our Proposed CL-Signatures
Chapter 4

Accumulators

Introduced by Benaloh and de Mare [BdM93], accumulators allow the representation of a set of elements by a single value independent of the cardinality of the set. Given a value $v$, one can accumulate an element $y$ into $v$ by invoking the accumulating function $f$ as $v' \leftarrow f(v, y)$. Therefore, if $u$ is an initial value, $v = f(\cdots f(f(u, y_1), y_2) \cdots, y_n)$ represents a value $v$ into which elements $y_1, y_2, \ldots, y_n$ have been accumulated. The accumulating function $f$ is quasi-commutative. That is, $f(f(u, y_1), y_2) = f(f(u, y_2), y_1)$ for any $u, y_1, y_2$. Thus, for a set of elements $Y = \{y_1, y_2, \ldots, y_n\}$, we abuse the notation and uses $f(u, Y)$ to represent $f(\cdots f(f(u, y_1), y_2) \cdots, y_n)$. Furthermore, for any element $y$ and any value $v$, there exists a witness $w$ for $y$ w.r.t. $v$ if and only if $y$ has been accumulated into $v$. One can thus, by demonstrating (the knowledge of) a valid corresponding witness, prove that an element has been accumulated into a value.

Accumulators should be collision-resistant [BP97]: for any element set $Y$, if $v = f(u, Y)$, then it is (computationally) infeasible to compute a witness $w$ w.r.t $v$ for any element $\bar{y} \notin Y$. Collision-resistance has now become a standard feature of accumulators; we refer to collision-resistant accumulators simply as accumulators in the rest of this thesis.

Accumulators are mainly use as building blocks for more complex cryptographic systems. For instance, they are used to construct anonymous identification in ad hoc groups [DKNS04], and anonymous credentials with efficient revocation of credentials [CL02b].

Related Works.

Here we discuss in greater depth the existing accumulator constructions. In particular, we propose a classification of all existing accumulator constructions into two types, namely type-SRSA and type-qSDH, and explain some of the features and
limitations of each type.

**Type-SRSA Constructions** Several accumulator constructions[^BP97] have their accumulating function in the form of $v' = v^y \mod N$ for some safe-prime product $N$, and rely on the Strong RSA (SRSA) assumption[^BP97] for their security. They permit only primes of certain size (as determined by a security parameter) to be accumulated.

Dodis *et al.*[^DKNS04] constructed the first constant-sized ring signatures[^RST01], wherein the signer accumulates the public keys $\{y_i\}$ of all signers in the ring and then demonstrates (non-interactively) that the signer is in possession of a private key that corresponds to a public key in the accumulator value. Since they employed a type-SRSA accumulator construction, the public keys hence have to be primes so that they can be accumulated. Consequently, a rather special private/public key-pair was used: the private key is a pair of two primes $(p, q)$ of specific length such that the corresponding public key $y = 2pq + 1$ is also prime, so that the mapping $h_F$ from private keys to public keys is thus $h_F : (p, q) \mapsto y = 2pq + 1$, the one-wayness of which can be reduced to the hardness of the factorisation problem.

**Type-$q$SDH Constructions** One accumulator construction relies on the $q$-Strong Diffie-Hellman ($q$-SDH) assumption[^BB04] for its security, and is from Nguyen[^Ngu05]. To the best of the author’s knowledge, this is the only construction of accumulator of this type.

In his construction, the accumulating function has the form of $v' = v^{s+y}$ in a cyclic group equipped with a bilinear map, where $s$ is the master secret of the accumulator instance. The domain of accumulatable elements is $\mathbb{Z}_p \setminus \{-s\}$ for some prime $p$.

Besides operating in a different mathematical structure, Nguyen’s construction differs from type-SRSA accumulator constructions in at least two ways. First, its accumulating function has the form of $g \circ f$ for some function $g$ such that only $f$ needs to be quasi-commutative (as opposed to the accumulating function $g \circ f$). Second, while anyone can add elements into the accumulator value in type-SRSA accumulator constructions, doing the same requires the knowledge of the master secret in Nguyen’s construction.

To the authors’ best knowledge, no one-way mapping with efficient zero-knowledge...
A proof-of-knowledge protocol exists to map private keys onto the domain of accumulatable elements in Nguyen’s construction. Consequently, the problem of constructing a constant-size ring signature from Nguyen’s construction remains open.

**Features of Existing Accumulators.** Several uses of accumulators, e.g., revocation in anonymous credential systems, require them to be dynamic, as defined by Camenisch and Lysyanskaya [CL02b]: given an accumulator value, one can efficiently update—by adding elements to, and possibly later removing them from—the value. Furthermore, when a value is updated, e.g., from $v$ to $v'$, a witness $w$ for some element $y$ w.r.t. $v$ can also be efficiently updated to a new witness $w'$ for the same element $y$ w.r.t. the new value $v'$. They also gave the first construction of dynamic accumulator [CL02b]. The other dynamic accumulator, the type-$q$SDH accumulator due to [Ngu05] was later proposed.

Li et al. [LLX07] presented a technique to augment universality to the above type-SRSA dynamic accumulator construction. As an extension to dynamic accumulators, they proposed and constructed dynamic universal accumulators, in which there exists a non-membership witness $\bar{w}$ for $\bar{y}$ w.r.t. value $v = f(u,Y)$, for any element set $Y$ and any element $\bar{y} \notin Y$. A corresponding non-membership witness (or the demonstration of its knowledge) can thus serve as a proof that an element is not in an accumulator.

Table 4 summarised the properties of existing accumulators.

<table>
<thead>
<tr>
<th>Accumulator</th>
<th>Type</th>
<th>Dynamic</th>
<th>Universal</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BP97]</td>
<td>SRSA</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>[CL02b]</td>
<td>SRSA</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>[Ngu05]</td>
<td>$q$SDH</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>[LLX07]</td>
<td>$q$SDH</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of Existing Accumulators

**Our Contribution**

We formally introduce two features to accumulators, namely, *Bound* and *multiversality*. Recall that an accumulator allows the accumulation of a set of values into a short value. We say an accumulator is bounded if there is a upper limit of the size of the set of values to be accumulated. Informally speaking, an accumulator is *bounded*, if and only if there exists a positive integer $k$, called the *bound* of the accumulator,
such that at most $k$ elements can be accumulated into the accumulator. We then make the observation that the type $q$SDH accumulator due to Nguyen \cite{Ngu05} is in fact a bounded accumulator. Formal definition of bounded accumulator shall be presented in Section \ref{sec:boundedaccumulator}. Bounded accumulator is an important building block of our electronic cash systems, to be presented in the next chapter.

*Dynamic Multiversal Accumulator* (DMA) is an extension to dynamic universal accumulator. Informally speaking, a DMA is a collision-resistant, dynamic and universal accumulator with the following additional features:

1. There exists a family of one-way pseudorandom functions $\mathcal{H} = \{h_1, h_2, h_3, \ldots\}$ that map onto (a subset of) the domain of accumulatable elements.

2. There exist efficient and secure commitment schemes and zero-knowledge protocols for various relationship among accumulated values, (the commitment of) accumulatable elements, and (the commitment of) functional preimages, including, but not necessarily limited to, such protocols for proving that:

   (a) A commitment $C$ opens to an accumulatable element that is in an accumulator value $v$,
   
   $$
   PK \{(y) : v = f(u, Y) \land y \in Y \land c = \text{Commit}(y)\}.
   $$

   (b) A commitment $C$ opens to an accumulatable element that is *not* in an accumulator value $v$, i.e.,
   
   $$
   PK \{(y) : v = f(u, Y) \land y \notin Y \land c = \text{Commit}(y)\}.
   $$

   (c) A commitment $C$ opens to an accumulatable element that is the output of a function $h_i$ on input a preimage, to which another commitment $c_0$ opens, i.e.,
   
   $$
   PK \{(y, x) : y = h_i(x) \land h_i \in \mathcal{H} \land c = \text{Commit}(y) \land c_0 = \text{Commit}_0(x)\}.
   $$

Formal definition of DMA is given in Section \ref{sec:dynamicmultiversalaccumulator}. Looking ahead, our construction of DMA is based on Nguyen’s type-$q$SDH accumulator construction\cite{Ngu05}. We are going to build an attribute-based anonymous authentication system from our DMA, to be presented in the next chapter.

\footnote{We are aware of the attacks on Nguyen’s proposal due to \cite{TW06} and \cite{ZC05}. Specifically, \cite{TW06} pointed out a flaw in Nguyen’s definition in which the auxiliary information is exposed to the adversary. Looking ahead, we avoid the flaw by giving $g \circ f$, instead of $f$ and $g$ to the adversary. The attack in \cite{ZC05} is on the ID-based ad-hoc anonymous identification scheme based on Nguyen’s accumulator instead of the accumulator itself.}
4.1 Syntax

4.1.1 Dynamic Multiversal Accumulator

We incrementally define Dynamic Multiversal Accumulators (DMAs). First, we recall the definition of type-qSDH Accumulators. Then we define universal accumulators (UAs) by adapting Li et al.’s definition for the universality property to type-qSDH accumulators. We then extend the definition for (dynamic) multiversal accumulators. Recalled that in type-qSDH accumulator, the accumulating function has the form of \( f \circ g \), and that only \( g \) needs to be quasi-commutative.

Definition 4.1 (Accumulators) An accumulator is a scheme with the following properties:

- (Efficient generation.) There exists a PPT algorithm \( \text{AGen} \) that, on input security parameter \( 1^\lambda \), outputs a tuple \( (f, g, Y_f, u, t_f) \), where \( f \) is a function \( U_f \times Y'_f \rightarrow U_f \) and \( g \) is another function \( U_f \rightarrow U_g \) for some domains \( Y_f, U_f, U_g \); \( Y_f \subseteq Y'_f \) is the domain for accumulatable elements; \( t_f \) is some optional auxiliary information about \( f \); and \( u \) is an element in \( U_f \). We assume the tuple \( (f, g) \) is drawn uniformly at random from its domain.

- (Quasi-commutativity.) For all \( (f, g, Y_f, \cdot) \leftarrow \text{AGen}(1^\lambda) \), \( v \in U_f \) and \( y_1, y_2 \in Y'_f \), we have \( f( f(v, y_1), y_2) = f( f(v, y_2), y_1) \). Hence, if \( Y = \{y_1, \ldots, y_k\} \subset Y'_f \), then we can denote \( f( \cdots f( f(v, y_1), y_2) \cdots, y_k) \) by \( f(v, Y) \) unambiguously.

- (Efficient evaluation.) For all \( (f, g, Y_f, t_f, u) \leftarrow \text{AGen}(1^\lambda) \), \( v \in U_f \), and \( Y \subseteq Y_f \) so that \( |Y| \) is polynomial in \( \lambda \), the function \( g \circ f(v, Y) \) is computable in time polynomial in \( \lambda \). \( v = g \circ f(u, Y) \) represents the set \( Y \). We call \( v \) the accumulated value for \( Y \) and say that \( y \) has been accumulated into \( v \) (or \( y \) is “in” \( v \)), for all \( y \in Y \).

- (Membership witnesses.) For all \( (f, g, Y_f, \cdot) \leftarrow \text{AGen}(1^\lambda) \), there exists a relation \( \Omega \) that defines membership witnesses: a value \( w \) is a valid membership witness for element \( y \in Y_f \) w.r.t. accumulated value \( v \in U_f \) if and only if \( \Omega(w, y, v) = 1 \). Membership witness should be efficiently computable (in polynomial-time in \( \lambda \)) without \( t_f \).

Definition 4.2 (Universal Accumulators (UAs)) An universal accumulator is an accumulator with the following additional properties:
4.1. Syntax

- (Non-membership witnesses.) For all \((f, g, \mathcal{Y}_f, \cdot) \leftarrow \text{AGen}(1^\lambda)\), there exists a relation \(\overline{\Omega}\) that defines non-membership witnesses: a value \(w\) is a valid non-membership witness for element \(y \in \mathcal{Y}_f\) w.r.t. accumulated value \(v \in \mathcal{U}_f\) if and only if \(\overline{\Omega}(w, y, v) = 1\). Non-membership witness should be efficiently computable (in polynomial-time in \(\lambda\)) without \(t_f\).

The security of (universal) accumulators requires the difficulty of finding a valid membership (resp. non-membership) witness for an element that is not in (resp. is indeed in) an accumulated value w.r.t. that accumulated value. We employ a strong definition in which the adversary is considered successful even if he presents an element that is outside the intended domain of the accumulator (\(\mathcal{Y}_f'\) instead of \(\mathcal{Y}_f\)). Accumulators with this stronger sense of security improve efficiency of systems on which it is based because users within this system need not conduct any proof to demonstrate the elements presented are inside the intended domain of the accumulator. Below we give a precise definition of security for accumulators and universal accumulators.

**Definition 4.3 (Security of Accumulators)** An accumulator is secure if, for any PPT algorithm \(A\), \(P_1\) is negligible in \(\lambda\), where:

\[
P_1 = \Pr \left[ (f, g, \mathcal{Y}_f, u, \cdot) \leftarrow \text{AGen}(1^\lambda); \ (y, w, Y) \leftarrow A(g \circ f, g, \mathcal{Y}_f, u) : Y \subset \mathcal{Y}_f' \wedge y \in \mathcal{Y}_f' \setminus Y \wedge \Omega(w, y, g \circ f(u, Y)) = 1 \right].
\]

**Definition 4.4 (Security of Universal Accumulators (UAs))** An universal accumulator is secure if, for any PPT algorithm \(A\), both \(P_1\) and \(P_2\) are negligible in \(\lambda\), where:

\[
P_1 = \Pr \left[ (f, g, \mathcal{Y}_f, u, \cdot) \leftarrow \text{AGen}(1^\lambda); \ (y, w, Y) \leftarrow A(g \circ f, g, \mathcal{Y}_f, u) : Y \subset \mathcal{Y}_f' \wedge y \in \mathcal{Y}_f' \setminus Y \wedge \Omega(w, y, g \circ f(u, Y)) = 1 \right],
\]

\[
P_2 = \Pr \left[ (f, g, \mathcal{Y}_f, u) \leftarrow \text{AGen}(1^\lambda); \ (y, w, Y) \leftarrow A(g \circ f, g, \mathcal{Y}_f, u) : Y \subset \mathcal{Y}_f' \wedge y \in Y \wedge \overline{\Omega}(w, y, g \circ f(u, Y)) = 1 \right].
\]

Next, we define **Multiversal Accumulators (MAs)**.

**Definition 4.5 (Multiversal Accumulators (MAs))** A multiversal accumulator is an universal accumulator with an additional property: there exists an efficient
4.1. Syntax

PPT algorithm $\text{AGen}'$ that, on input $(1^\lambda, f, Y_f)$, outputs an element in $h \in \mathcal{H}_f$, where $\mathcal{H}_f$ is a set of efficiently computable (in polynomial-time in $\lambda$) functions $\{h : X_f \rightarrow Y_f\}$ for some domain $X_f$.

The security of multiversal accumulators requires that the output of any function generated by $\text{AGen}'$ is indistinguishable from an element drawn uniformly at random from its domain. Below is a precise definition.

**Definition 4.6 (Security of Multiversal Accumulators (MAs))** A multiversal accumulator is secure if it is a secure universal accumulator and, for all PPT algorithm $\mathcal{A}$, $P_3$ is negligible in $\lambda$, where:

$$P_3 = \Pr \left[ \left( f, g, Y_f, u, t_f \right) \leftarrow \text{AGen}(1^\lambda); h, h_1 \leftarrow \text{AGen}'(1^\lambda, f, Y_f); x \in_R X_f; y \leftarrow h(x); y_0 \in_R Y_f; y_1 \leftarrow h_1(x); b \in_R \{0, 1\}; b' \leftarrow \mathcal{A}(g \circ f, g, Y_f, t_f, h, h_1, y, y_0) : b = b' \right] - \frac{1}{2}.$$ 

Finally, we are going to define DMAs.

**Definition 4.7 (Dynamic Multiversal Accumulators (DMAs))** A Dynamic Multiversal Accumulator (DMA) is a multiversal accumulator with the following additional properties:

- **(Efficient update of accumulator.)** There exists an efficient algorithm $\text{Dyn}_1$ such that for all $v = g \circ f(u, Y_f)$, $y \notin Y$ and $\hat{v} \leftarrow \text{Dyn}_1(t_f, v, y)$, we have $\hat{v} = g \circ f(u, Y_f \cup \{y\})$. If $y \in Y$ instead, then we have $\hat{v} = g \circ f(1, Y_f \setminus \{y\})$ instead.

- **(Efficient update of membership witnesses.)** Let $v$ and $\hat{v}$ be the original and updated accumulator values respectively and $\hat{y}$ be the newly added (or deleted) element. There exists an efficient algorithm $\text{Dyn}_2$ that, on input $y, w, v, \hat{v}$ with $y \neq \hat{y}$ and $\Omega(w, y, v) = 1$, outputs $\hat{w}$ such that $\Omega(\hat{w}, y, \hat{v}) = 1$.

- **(Efficient update of non-membership witnesses.)** Let $v$ and $\hat{v}$ be the original and updated accumulator values respectively and $\hat{y}$ be the newly added (or deleted) element. There exists an efficient algorithm $\text{Dyn}_3$ that, on input $y, \overline{w}, v, \hat{v}$ with $y \neq \hat{y}$ and $\overline{\Omega}(\overline{w}, y, v) = 1$, outputs $\overline{\hat{w}}$ such that $\overline{\Omega}(\overline{\hat{w}}, y, \hat{v}) = 1$.

It is obvious that dynamism of algorithms $\text{Dyn}_1, \text{Dyn}_2, \text{Dyn}_3$ can be constructed trivially using computations that are linear in the size of the accumulated set $Y$. 
That is, by computing everything from scratch. Thus, by “efficient” we mean the
time complexity for each of these functions is \textit{independent} of the size of the accumu-
lated set \( Y \). It is also worth noting that the update of membership and non-
membership witness does not require the auxiliary information.

While the security goal of DMAs is the same as that of multiversal accumulators
(namely, it must be hard to find a valid witness that proves an incorrect statement
about what is or is not in an accumulated value), in the case of DMAs, an adversary
has the additional capability of dynamically adding elements to and/or removing
elements from an accumulated value when attempting to violate the goal. Therefore,
to define the security for DMAs, we first formally describe such capability of the
adversary. An oracle \( O_{\text{Dyn}} \) is initialised with the tuple \(( f, g, Y_f, u, t_f )\) and it maintains
a list of values \( Y \), which is initially empty. \( O_{\text{Dyn}} \) responds to two types of queries,
namely “add \( y \)” and “delete \( y \).” It responds to a “add \( y \)” query by adding \( y \) to the
set \( Y \), modifying the accumulator value \( v \) using algorithm \( \text{Dyn}_1 \) and sending back
the updated accumulator value \( \hat{v} \). It responds to a “delete \( y \)” query by deleting it
from set \( Y \), modifying the accumulator value \( v \) using algorithm \( \text{Dyn}_1 \) and sending
back the updated accumulator value \( \hat{v} \). In the end, \( O_{\text{Dyn}} \) outputs the current set \( Y \)
and accumulated value \( v \).

\textbf{Definition 4.8 (Security of Dynamic Multiversal Accumulators (DMAs))}

A dynamic multiversal accumulator is secure if, for any PPT algorithm \( A \), \( P_4 \), \( P_5 \) and \( P_5 \) are negligible in \( \lambda \), where:

\[
\begin{align*}
P_4 &= \Pr \left[ (f, g, Y_f, u, t_f) \leftarrow \text{AGen}(1^\lambda); (y, w, Y) \leftarrow A^{O_{\text{Dyn}}(f, g, Y_f, t_f)}(g \circ f, g, Y_f); \\
Y \subseteq Y_f' \land y \in Y_f' \setminus Y \land v = g \circ f(u, Y) \land \Omega(w, y, v) = 1 \right] \\
P_5 &= \Pr \left[ (f, g, Y_f, u, t_f) \leftarrow \text{AGen}(1^\lambda); (y, w, Y) \leftarrow A^{O_{\text{Dyn}}(f, g, Y_f, t_f)}(g \circ f, g, Y_f); \\
Y \subseteq Y_f' \land y \in Y \land v = g \circ f(u, Y) \land \Omega(w, y, v) = 1 \right]
\end{align*}
\]

Similar to the relationship between accumulator and dynamic accumulator [CL02b]
or UA and dynamic UA [LLX07], we have the following theorem, stating that any
dynamic multiversal accumulator is secure if the underlying multiversal accumulator
is secure.

\textbf{Theorem 4.1} A DMA is secure if the underlying multiversal accumulator is secure.

\textit{described above}
Proof: Let $\mathcal{A}$ be an adversary that breaks the security definition in Definition 4.7 for an dynamic multiversal accumulator, we show how to construct a simulator $S$ which breaks the security definition of the underlying multiversal accumulator.

The third probability $P_3$ in the definition regarding the security of the family of functions $\mathcal{H}$ for the dynamic multiversal accumulator is the same as that for the underlying multiversal accumulator. Thus, if $\mathcal{A}$ breaks the security of it, $S$ simply uses it to break the same property of the underlying multiversal accumulator.

Now we assume $\mathcal{A}$ breaks the security property by outputting a membership witness (resp. non-membership witness) for an element that is not inside (resp. inside) the accumulator. On input $(f, g)$, $S$ gives the value $g \circ f$ to $\mathcal{A}$. $S$ simulates the oracle $O_{\text{Dyn}}$ by computing the accumulation of the updated set. For instance, when $\mathcal{A}$ sends add $y$ query, $S$ inserts $y$ to the set $Y$, and computes $v = g \circ f(u, Y)$ from scratch. Similarly, when $\mathcal{A}$ sends delete $y$ query, $S$ removes $y$ from set $Y$ and computes $v = g \circ f(u, Y)$. Note that both operations do not require the auxiliary information. Finally, $\mathcal{A}$ outputs an element $y \notin Y$ (resp. $y \in Y$) and a membership witness $w$ (resp. non-membership witness $w$) for $y$. $S$ outputs $w, y, Y$ and breaks the security property of the underlying multiversal accumulator. \qed

4.1.2 Bounded Accumulator

The idea of bounded accumulator is simple. An accumulator (be it dynamic, universal or multiversal or not) is bounded if there exists a limit $k$, called the bound of the accumulator, such that at most $k$ elements can be accumulated without the auxiliary information. The following definition formally captures the idea of bounded accumulator.

**Definition 4.9 (Security of Bounded Accumulators)** An accumulator is bounded, if there exists a positive integer $k$ such that, for any PPT algorithm $\mathcal{A}$, $P_6$ is negligible in $\lambda$, where:

$$
P_6 = \Pr \left[ (f, g, Y_f, u, \cdot) \leftarrow \text{AGen}(1^\lambda); (y_1, w_1, \ldots, y_{k+1}, w_{k+1}, v) \leftarrow \mathcal{A}(g \circ f, g, Y_f, u) : v \in \mathcal{U}_g \wedge y_i \in \mathcal{Y}_f \wedge \Omega(w_i, y_i, v) = 1 \forall i \in \{1, \ldots, k + 1\} \right] \right]
$$
4.2 The Constructions

We first describe the type-qSDH accumulator due to \[\text{Ngu05}\] in our notation for two purposes. Firstly, we make the observation that this accumulator is bounded, thus providing an example of Bounded Accumulator. Secondly, this accumulator also serves as a starting point of our DMA. We then go on to present an universal accumulator, followed by the multiversal accumulator. We finally describe our construction of DMA by presenting the necessary algorithms for adding dynamism.

4.2.1 (Bounded) Accumulator

Generation. Let \(\lambda\) be a security parameter. Let \(\hat{e} : G_1 \times G_1 \rightarrow G_T\) be a bilinear pairing such that \(|G_1| = |G_T| = p\) for some \(\lambda\)-bit prime \(p\). Let \(g_0\) be a generator of \(G_1\). For simplicity, we describe the accumulator in the symmetric setting. It can be easily transformed into the general setting. Let \(G_q = \langle h \rangle \subset \mathbb{Z}_p^*\). The generation algorithm \(\text{AGen}\) randomly chooses \(\alpha \in \mathbb{Z}_p\). For simplicity, we always take the initial element \(u = 1\), the identity element in \(\mathbb{Z}_p^*\). Note that \(u\) is also the identity element of \(G_q\). The function \(f\) is defined as \(f : u, y \mapsto u(y + \alpha)\). The function \(g\) is defined as \(g : y \mapsto g_0^y\). The domain \(\mathcal{Y}_t\) of accumulatable elements is \(G_q \setminus \{-\alpha\}\).

The auxiliary information \(t\) is \(\alpha\). To define \(g \circ f\) with \textit{bound} \(k\), also output \(\text{param}_k = (g_0^\alpha, \ldots, g_0^{k\alpha})\) as public parameter.

Evaluation. Computing \(g \circ f(1,Y)\) efficiently is straightforward with the auxiliary information \(\alpha\). Note that in case one wishes to allow computation of \(g \circ f\) without \(\alpha\), one makes use of \(\text{param}_k\). If we denote the polynomial \(\prod_{y \in Y}(y + \alpha) = \sum_{i=0}^{\text{deg} k} u_i \alpha^i\) of maximum degree \(k\) as \(v(\alpha)\), one can efficiently compute \(g \circ f(1,Y)\) as \(g \circ f(1,Y) = g_0^{v(\alpha)} = \prod_{i=0}^{\text{deg} k} g_0^{u_i} \in G_1\) without the knowledge of \(\alpha\).

Membership witnesses. The relation \(\Omega\) is defined as \(\Omega(w, y, v) = 1\) if and only if \(\hat{e}(w, g_0^{y \alpha^i} g_0^{\alpha^i}) = \hat{e}(v, g_0)\). For a set of values \(Y = \{y_1, \ldots, y_k\} \in G_q\), a membership witness for the value \(y \in Y\) can be computed in either one of the following ways, depending on whether one knows the auxiliary information.

- (With auxiliary information.) Compute the witness as \(w = [g_0^{\prod_{i=1}^{k}(y_i + \alpha)}]_n\).  

\(^3\)If \(p = 2q + 1\), one can choose a random element in \(h \in R \mathbb{Z}_p^*\) with order \(q\) and set \(G_q\) as the subgroup generated by an element \(h \in \mathbb{Z}_p^*\) of order \(q\).
4.2. The Constructions

Next we describe how the accumulator described above can be extended to an universal accumulator by adding the ability to accommodate non-membership witness.

Non-membership witnesses. The relation $\overline{\Omega}$ for non-membership witnesses is defined as $\overline{\Omega}(w, y, v) = 1$ if and only if $w = (c, d)$ and $e(c, g_0^w g_0^d) e(g_0, g_0)^d = e(v, g_0)$. For a set of values $Y = \{y_1, \ldots, y_k\} \in \mathbb{G}_q$, a non-membership witness for $\hat{y} \notin Y$ can be computed in either one of the following ways, depending on whether one knows the auxiliary information:

- (With auxiliary information.) Compute $w = (c, d)$ according to $d = \prod_{i=1}^{k} (y_i + \alpha) \mod (\alpha + \hat{y}) \in \mathbb{Z}_p$ and $c = g_{0}^{\frac{w(y_i + \alpha) - d}{\alpha + \hat{y}}} \in \mathbb{G}_1$.

- (Without auxiliary information.) Denote the polynomial $v(\alpha)$ as $\prod_{i=1}^{k} (y_i + \alpha)$. Compute a polynomial division of $v(\alpha)$ by $(\alpha + \hat{y})$. Since $(\alpha + \hat{y})$ is a degree one polynomial and $\hat{y} \neq y_i$ for all $i$, there exists a degree $k-1$ polynomial $c(\alpha)$ and a constant $d$ such that $v(\alpha) = c(\alpha)(\alpha + \hat{y}) + d$. Expand $c(\alpha)$ and write it as $c(\alpha) = \sum_{i=0}^{k-1} (u_i \alpha^i)$. Compute $c = g_{0}^{c(\alpha)} = \prod_{i=0}^{k} g_i^{u_i} \in \mathbb{G}_1$. The non-membership witness of $\hat{y}$ is $w = (c, d)$.

The theorem below states the security of our UA.

**Theorem 4.2 (Security of our UA construction)** Under the $k$-SDH assumption in $\mathbb{G}_1$, the above construction is a secure universal accumulator.

**Proof:** Let $\mathcal{A}$ be a polynomial-time adversary to our UA such that the maximum number of elements to be accumulated is $k$, we show how to construct a PPT simulator $\mathcal{S}$ which solves the $k$-SDH problem in $\mathbb{G}_1$ by invoking $\mathcal{A}$.

We assume the problem instance is from a restricted class of the $q$-SDH problem (specifically, the $k$-SDH problem instance is in a group of safe-prime order equipped...
with a bilinear map. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ such that $\mathbb{G}_1 = \langle g_0 \rangle$ and $|g_0| = p$ such that $p = 2q + 1$ and both $p, q$ are primes. $S$ is given $\mathbb{G}_1, \mathbb{G}_2, \hat{e}, g_0, g_0^a, \ldots, g_0^{a^k}$ and its goal is to output $w^*, y^*$ such that $w^{a^i + y} = g_0$. Such pair satisfies the relation $\hat{e}(w^*, g_0^a g_0^d) = \hat{e}(g_0, g_0)$.

Let $k$ be the maximum number of elements to be accumulated in the accumulator. ($\mathbb{G}_1, \mathbb{G}_2, \hat{e}, p, q, g_0, g_0^a, \ldots, g_0^{a^k}$) is given to the adversary as the definition of $g \circ f$. That is, $f : (u, y) \mapsto u(\alpha + y)$ and $g : y \mapsto g_0^y$ such that $g \circ f(1, Y)$ is efficiently computable for any set $Y \subset \mathbb{G}_q$ with $|Y| \leq k$.

If $P_1$ is non-negligible, $A$ outputs a set $Y \subset \mathbb{Z}_p^*$, a value $y' \notin Y$, and a witness $w$ such that $\Omega(w, y', g \circ f(1, Y)) = 1$ with non-negligible probability. We have $\hat{e}(w', g_0^y g_0^d) = \hat{e}(g \circ f(1, Y), g_0)$. It is safe to assume $\alpha \notin Y$, otherwise $S$ solves the $k$-SDH problem immediately. Let $v(\alpha) = \prod_{y \in Y}(\alpha + y)$. Note that $g \circ f(1, Y) = g_0^{v(\alpha)}$. Since $y' \notin Y$, there exists a polynomial $q(\alpha)$ of degree less than $k$ such that $v(\alpha) = q(\alpha)(y' + \alpha) + d$ for a constant $d$. Let $w^* = [w' g_0^{-q(\alpha)}]^\frac{1}{2}$. $w^*$ is computable because $S$ can always express $q(\alpha)$ as $\sum_{i=1}^{i=k-1} u_i \alpha^i$. $S$ sets $y^* = y'$ and returns $(w^*, y^*)$ as the solution to the $k$-SDH problem.

If $P_2$ is non-negligible, $A$ outputs a set $Y \subset \mathbb{Z}_p^*$, a value $y' \in Y$, and a witness $\overline{w} = (c', d') \in \mathbb{G}_1 \times \mathbb{Z}_p^*$ such that $\overline{\Omega}(\overline{w}, y', g \circ f(1, Y)) = 1$ with non-negligible probability. That is, $\hat{e}(c', g_0^y g_0^d) \hat{e}(g_0, g_0)^{d'} = \hat{e}(g \circ f(1, Y), g_0)$. It is safe to assume $\alpha \notin Y$, otherwise $S$ solves the $k$-SDH problem immediately. Let $v(\alpha) = \prod_{y \in Y}(\alpha + y)$. Since $y' \in Y$, there exists a polynomial $q(\alpha)$ such that $v(\alpha) = q(\alpha)(y' + \alpha)$. $S$ compute $w^* = [c' g_0^{-q(\alpha)}]^\frac{1}{2}$, sets $y^* = y'$ and returns $(w^*, y^*)$ as the solution to the $k$-SDH problem.

\[\square\]

### 4.2.3 Multiversal Accumulator

We extend the above UA to multiversal accumulator by defining the function $AGen'$.

Formally,

$(AGen')$. For an intended accumulator domain $\mathcal{Y}_f = \mathbb{G}_q$, $AGen'$ randomly chooses a generator $h_i \in \mathbb{G}_q$. The function $h_i$ is defined as $h_i : \mathbb{Z}_q \rightarrow \mathbb{G}_q$ such that $h_i : x \mapsto h_i^x$.

The theorem below states the security of our MA construction.
4.2. The Constructions

Theorem 4.3 (Security of our MA construction) Under the $k$-SDH assumption in $\mathbb{G}_1$ and the DDH assumption in $\mathbb{G}_q$, the above construction is a secure multiversal accumulator.

Proof: $Pr_1$ and $Pr_2$ have been shown to be negligible under the $k$-SDH assumption in $\mathbb{G}_1$ in Theorem 4.2. Assume there exists an adversary $A$ such that $Pr_3$ is non-negligible. We show how to construct a PPT simulator $S$, having black-box access to $A$, that solves the DDH problem in $\mathbb{G}_q$ with non-negligible advantage.

Specifically, $S$ is given a problem instance $(h, h_1, h^x, y^*)$ and its goal is to tell if $y^* = h_1^x$ or $y^*$ is a random element in $\mathbb{G}_q$. $S$ generates a prime $p$ such that $\mathbb{G}_q \subset \mathbb{Z}_p^*$. $S$ then generates a bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ such that $|\mathbb{G}_1| = |\mathbb{G}_T| = p$. It then randomly chooses a generator $g_0$ of $\mathbb{G}_1$ and $\alpha \in \mathbb{RZ}_p$. $S$ then defines $f$ as $f : u, y \mapsto u(y + \alpha)$ and $g$ as $g : y \mapsto g_0^y$. Computes $\text{param}_k = (g_0^0, \ldots, g_0^k)$. $S$ sets $h : x \mapsto h^x \in \mathbb{G}_q$, $h_1 : x \mapsto h_1^x \in \mathbb{G}_q$, $y = h^x$.

$S$ then gives $(\hat{e}, \mathbb{G}_1, \mathbb{G}_T, g_0, \text{param}_k, h, h_1, y, y^*)$ to $A$. $A$ returns a guess bit $b' \in \{0, 1\}$. If $A$ returns 0, $S$ returns 0 which indicates that $y^*$ is a random element. On the other hand, if $A$ returns 1, $S$ returns 1 which indicates that $y^* = h_1^x$.

It is straightforward to see that $S$ solves the DDH problem as long as the guess bit output by $A$ is correct. Since $Pr_3$ is non-negligible for adversary $A$, the advantage of $S$ in solving the DDH problem is also non-negligible.

4.2.4 Dynamic Multiversal Accumulator

We present our construction of dynamic multiversal accumulator by adding the various dynamism algorithms $\text{Dyn}_1$, $\text{Dyn}_2$, $\text{Dyn}_3$ to our construction of MA. Due to Theorem 4.3 and Theorem 4.1, our DMA is secure under the $k$-SDH assumption and the DDH assumption in $\mathbb{G}_q$.

(Update of accumulator (algorithm $\text{Dyn}_1$).) Adding a value $\hat{y}$ to the accumulator $v$ can be done by computing $\hat{v} = v^{\hat{y} + \alpha}$. Similarly, deleting a value $\hat{y}$ in the accumulator $v$ can be done by computing $\hat{v} = v^{1/(\hat{y} + \alpha)}$. Both cases require the auxiliary information $\alpha$.

(Update of membership witnesses (algorithm $\text{Dyn}_2$).) Let $w$ be the original membership witness of $y$ w.r.t the accumulator value $v$. Let $\hat{v}$, $\hat{y}$ be the new accumulator value and the value added (resp. deleted). Suppose $\hat{y}$ has been
added, the new membership witness \( \hat{w} \) for \( y \) can be computed as \( vw^{\hat{g} - y} \). Suppose \( \hat{y} \neq y \) has been deleted, the new non-membership witness \( \hat{w} \) for \( y \) can be computed as \( w^{\frac{1}{y} \cdot \hat{v}^{\frac{1}{y}}} \).

(Update of non-membership witnesses (algorithm Dyn\(_3\)).) Let \((c, d)\) be the original non-membership witness of \( y \) w.r.t. accumulator value \( v \). Let \( \hat{v}, \hat{y} \) be the new accumulator values and the value added (resp. deleted).

- **(Addition.)** Suppose \( \hat{y} \neq y \) has been added, the new non-membership witness \( \hat{c}, \hat{d} \) of \( y \) can be computed as \( \hat{c} = vc^{\hat{g} - y} \in G_1 \) and \( \hat{d} = d(y - y) \in \mathbb{Z}_p^\ast \).

This can be verified as follows:

\[
\hat{v} = v^{\alpha + y} = v^{(\alpha + y) + (\hat{g} - y)} = v^{\alpha + y}v^{\hat{g} - y} = v^{\alpha + y}(c^{\alpha + y}g_0^d)\hat{g} - y
\]

\[
= [v^{g\hat{g} - y}]^{\alpha + y}g_0^d(\hat{g} - y) = \hat{c}^{\alpha + y}g_0^d
\]

- **(Deletion.)** Suppose \( \hat{y} \) has been deleted, the new non-membership witness \( \hat{c}, \hat{d} \) of \( y \) can be computed as \( \hat{c} = (c\hat{v}^{-1})^{\frac{1}{\hat{g} - y}} \in G_1 \) and \( \hat{d} = d\frac{y}{\hat{g} - y} \in \mathbb{Z}_p^\ast \).

Indeed,

\[
\hat{v} = \hat{v}^{(\alpha + \hat{g} - (\alpha + y))} = v^{\frac{1}{\hat{g} - y}}\hat{v}^{\alpha + y} = [c^{\alpha + y}g_0^d]^{\frac{1}{\hat{g} - y}}\hat{v}^{\alpha + y}
\]

\[
= [(c\hat{v}^{-1})^{\alpha + y}g_0^d]^{\frac{1}{\hat{g} - y}} = [(c\hat{v}^{-1})^{\frac{1}{\hat{g} - y}}]^{\alpha + y}g_0^d = \hat{c}^{\alpha + y}g_0^d
\]

### 4.3 Σ-Protocols for our DMA

In this section we assume the following notations. Let \( Y \subset \mathbb{Z}_p^\ast \) and \( v = g \circ f(1, Y) \). That is, \( v \) is the accumulation of the set \( Y \). Below we are going to describe three Σ-protocols. Let \( g_1, g_2 \in_R G_1 \) be generators of \( G_1 \). Let \( h_1, h_2 \in_R G_q \) be generators of \( G_q \). They are required for the construction of the protocols. For simplicity, we use \( V \) to denote \( g_0^\alpha \).

#### 4.3.1 Knowledge of a Committed Value Inside an Accumulator

Let \( y \in Y \) and \( w \) be its corresponding membership witness. That is, \( \hat{e}(w, g_0^yg_0^\alpha) = \hat{e}(v, g_0) \). Let \( \mathcal{C} = g_1^yg_2^\alpha \in G_1 \) be a commitment of \( y \) for some random number \( r \). A Σ-protocol of the knowledge of \( y \) inside an accumulator \( v \) can be abstracted below.

\[
\text{PK}_{22}\{(y, w, r) : \hat{e}(w, g_0^yV) = \hat{e}(v, g_0) \land \mathcal{C} = g_1^yg_2^\alpha\}
\]
PK_{22} requires instantiation. The prover first computes auxiliary commitments \( A_1 = g_1^{r_1} g_2^{r_2}, A_2 = w g_2^{r} \) for some randomly generated \( r_1, r_2 \in R \). Then the prover conducts the following \( \Sigma \)-protocol, PK_{23}, with the verifier.

\[
\begin{align*}
\text{PK}_{23} & \left\{ \begin{array}{l}
(y, r, r_1, r_2, \beta_1, \beta_2) : \\
A_1 = g_1^{r_1} g_2^{r_2} & \land \\
1_{G_1} = A_1^{-y} g_1^{\beta_1} g_2^{\beta_2} & \land \\
C = g_1^y g_2^r & \land \\
\hat{\epsilon}(A_2, V) = \hat{\epsilon}(A_2, g_0)^{-y} \hat{\epsilon}(g_2, V)^{r_1} \hat{\epsilon}(g_2, g_0)^{\beta_1}
\end{array} \right. \\
\end{align*}
\]

PK_{23} can be conducted using standard techniques and we shall omit its details here.

### 4.3.2 Knowledge of a Committed Value Not Inside an Accumulator

Let \( y' \notin Y \) such that \( y' \in \mathbb{Z}_p \) and \((c, d)\) be its corresponding non-membership witness. That is, \( \hat{\epsilon}(c, g_0^y V) \hat{\epsilon}(g_0, g_0)^d = \hat{\epsilon}(v, g_0) \). Let \( C = g_1^y g_2^r \in G_1 \) be a commitment of \( y' \) for some random number \( r \). A \( \Sigma \)-protocol of the knowledge of \( y' \) which is not inside an accumulator \( v \) can be abstracted below.

\[
\begin{align*}
\text{PK}_{24} & \left\{ (y', c, d, r) : \hat{\epsilon}(c, g_0^y V) \hat{\epsilon}(g_0, g_0)^d = \hat{\epsilon}(v, g_0) \land d \neq 0 \land C = g_1^y g_2^r \right\}
\end{align*}
\]

PK_{24} requires instantiation. The prover first computes auxiliary commitments \( A_1 = g_1^{r_1} g_2^{r_2}, A_2 = c g_1^{r_1}, A_3 = g_1^{r_3} g_2^{r_4} \) and \( A_4 = g_1^{d r} \) for some randomly generated \( r_1, r_2, r_3, r_4 \in R \). The prover conducts the following \( \Sigma \)-protocol, PK_{25}, with the verifier.

\[
\begin{align*}
\text{PK}_{25} & \left\{ \begin{array}{l}
(y', d, r, r_1, r_2, r_3, r_4, \beta_1, \beta_2, \beta_3, \beta_4) : \\
A_1 = g_1^{r_1} g_2^{r_2} & \land \\
1_{G_1} = A_1^{-y} g_1^{\beta_1} g_2^{\beta_2} & \land \\
A_3 = g_1^{r_3} g_2^{r_4} & \land \\
1_{G_1} = A_3^{-d} g_1^{\beta_3} g_2^{\beta_4} & \land \\
A_4 = g_1^{\beta_4} & \land \\
C = g_1^y g_2^r & \land \\
\hat{\epsilon}(A_2, V) = \hat{\epsilon}(A_2, g_0)^{-y} \hat{\epsilon}g_0, g_0^{-d} \hat{\epsilon}(g_2, V)^{r_1} \hat{\epsilon}(g_2, g_0)^{\beta_1}
\end{array} \right. \\
\end{align*}
\]

PK_{25} can be conducted using standard techniques and we shall omit its details here. Note that, the verifier also needs to check \( A_4 \neq 1 \) to make sure \( d \neq 0 \).
Knowledge of a Committed Value and its Pre-Image

Let $\mathcal{C}_y = g_1^y g_2^{r_1} \in \mathbb{G}_1$ be a commitment of $y$ with randomness $r \in_R \mathbb{Z}_p$. Let $\mathcal{C}_x = h_1^x h_2^{r_2} \in \mathbb{G}_q$ be a commitment of $x \in \mathbb{Z}_q$ for some randomness $r_2 \in \mathbb{Z}_q$. The following protocol is useful in demonstrating the knowledge of $x$ and $y$ such that $x$ is the pre-image of $y$ under the function $h : x \mapsto h^x$. This is abstracted as $\text{PK}_{26}$.

$$\text{PK}_{26}\{(y, x, r_1, r_2) : \mathcal{C}_y = g_1^y g_2^{r_1} \land \mathcal{C}_x = h_1^x h_2^{r_2} \land y = h^x\}$$

This can simply be done by using the RCV protocol discussed in Section 3.4.1 (in its simplest case).

$$\text{PK}_{RCV}\{ (y, r_1, r_2, x) : \\
\mathcal{C}_y = g_1^y g_2^{r_1} \land \\
\mathcal{C}_x = h_1^x h_2^{r_2} \land \\
y = h^x \}$$

4.4 Chapter Summary

In this chapter, we formally introduced two features to accumulators, namely, Bound and multi-versality. We also provided construction of each type. As a concluding remark, recalled that any dynamic multiversal accumulator is also an universal accumulator. In this sense, we have proposed the first type-$q$SDH dynamic universal accumulator. Indeed, we present our construction of DMA as a dynamic universal accumulator in [ATSM09].
Chapter 5

Application to Electronic Cash Systems

This chapter focuses on the application of the primitives introduced in previous chapters to electronic cash systems. In particular, we discuss electronic cash with different features.

5.1 Background

With the advance of the Internet and the development of information technologies, e-commerce has boomed and opened up many new business opportunities. Today, traditional credit cards is virtually the only choice of Internet payment, despite its numerous shortcomings/limitations, e.g. lack of security, lack of privacy, large overhead and inefficiency in processing small payment amounts.

5.1.1 Electronic Payment Methods

There are various ways to pay electronically. our focus in this thesis is offline anonymous electronic cash. We first review different electronic payment methods, serving as a background of our discussion in the sequel. Closely mimicking their physical counterparts, payment methods in the electronic world can be divided into three categories: credit cards, electronic cheques and electronic cash.

Credit Cards. The most popular electronic payment on the Internet today is by credit cards because of their ease of use and ubiquitousness. A customer does not need to be registered for a new payment service, as long as he/she already has a credit card. Using the existing networks and a computer as a payment terminal, there is no need for creating new hardware or infrastructure, which means a high scalability. Credit cards provide a large customer base for merchants, thus acceptability is very
5.1. Background

However, credit card payments offer no anonymity, and do not allow small payments:

1. **High costs and inability to allow small payments.** Each credit card payment has a fixed cost of 20 to 40 cents originated from the cost of performing a transaction, plus a variable cost of 2 to 20%, which serves as a compensation for the credit card issuers’ lost from fraud. The variable cost is especially high for Internet purchases due to its higher security risks. As a result, small payments cannot be made with credit cards with a reasonable profit being made by the merchants.

2. **All purchases are traceable.** While the fact that credit card issuers have all the users’ spending information which enables easy resolution of payment disputes, the issuer’s possession of such information poses serious privacy concerns to the customers. For example, the information can be sold to advertisers to target advertisements to their audiences. In general, customers want to keep private about their whereabouts, their spending patterns, and their personal preferences.

An example of electronic payment systems using credit cards is SET [MS98]. It is VISA and MasterCard’s analog to a credit-card setting which uses digital signatures as a tool for authenticating users, merchants, and banks.

**Electronic Cheques.** Electronic cheques resemble closely physical cheques we are using in the real world. They are bit strings that encode a value, signed using digital signatures for distinguishing between valid and invalid bit strings. Though some methods have indeed been put to practice (e.g. NetBill [ST95] and NetCheque [NM95]), there has been no large-scale adoption to date.

**Electronic Cash.** In the physical world, credit cards, cheques and cash together dominate the ways people pay. The fact that each of them has its own pros and cons accounts for their co-existence. Electronic cash (E-cash), invented by Chaum [Cha82] in 1982, is the digital counterpart of cash in the electronic world. A practical electronic cash system should at least be offline and provide anonymity.
Depending on implementation, it may or may not require some proprietary hardware. Some successful stories include Hong Kong’s Octopus cards, Singapore’s EZLink cards, both of them require some proprietary hardware.

1. **Anonymity.** A distinctive feature of cash payments is the anonymity (also referred to as privacy) a customer can enjoy. Anonymity means that payments do not leak the customers’ whereabouts, spending patterns or personal preferences. Nevertheless, too much privacy may cause problems in the regulatory and legal levels. For example, it makes crimes such as money laundering, illegal goods purchasing and blackmailing relatively easy. To prevent against these, some electronic cash systems allow an administrative party to revoke the consumer’s anonymity under certain circumstances, such as a court order.

2. **Offline Transactions.** An electronic cash system is *online* (resp. *offline*) if it (resp. does not) requires the bank’s participation during payment between the customer and the merchant. An online system is less desirable since being available for helping out in every transaction is possibly a heavy burden to the bank and such systems are more prone to denial of service attacks. This undermines scalability of the system too. In fact, being offline is a prominent reason why physical cash is so widely used.

3. **Hardware Requirements.** Some electronic cash schemes derive their security entirely from the proprietary hardware used (e.g. the tamper-resistant chips in smart cards). Users carry hardware, in which the balance is kept. When a transaction is performed, this balance is altered correspondingly. In schemes that do not require proprietary hardware, the security is derived from some cryptographic tools. Users cannot generate money by themselves (without being detected). The “cash” can live and travel in, e.g., users’ hard disks. Relying on proprietary hardware puts an extra cost in infrastructure setup, management and maintenance and thus results in higher deployment cost of electronic cash systems.

### 5.1.2 More on Electronic Cash

**Architecture.** In its simplest form, an *e-cash* system consists of four main procedures/protocols, namely, *Account Establishment*, *Withdrawal Protocol*, *Spend Protocol* and *Deposit Protocol* amongst three parties, namely, the bank B, the user U and the
5.1. Background

merchant $M$. The user $U$ first performs Account Establishment with the bank $B$. The currency circulating around is quantised as coins. $U$ obtains a coin by performing Withdrawal Protocol with $B$ and spends the coin by participating in Spend Protocol with $M$. To deposit a coin, $M$ performs a Deposit Protocol with $B$. The above description is depicted in Figure 5.1.

Security Requirements. We consider three security aspects of offline e-cash, namely, balance, anonymity and exculpability. In the electronic world, all objects are represented by data; e-cash is by no means an exception. Without propriety hardware, it is hard to prevent malicious user from duplicating an electronic coin. That act of spending the same coin twice is called double-spending. The malicious user would probably spend the duplicated e-coin in two different shops. Thus, it is difficult for the shops to check for double-spending on their own if the system is offline. Instead, the bank checks for double-spending when the shops deposit the coins. The shops will get the real payment, or the bank will identify the double-spender. What we require is that no collusion of users and merchants together can deposit more than they withdraw without being detected. We call this property...
5.1. Background

balance and is the most important point of view from the bank. On the other hand, honest spenders cannot be slandered to have double-spent. This property is called exculpability. Finally, when the shops deposit the money from the payee, the bank should not be able to trace who the actual spender is even if the shop and bank cooperate. This is known as anonymity of user.

Desirable features. Depending on application, some add-on properties like divisibility and transferability may be desirable. High efficiency is also of key importance for practical e-cash systems. We explain these features as follows:

1. Revocability. Certain administrative party is capable of revoking the consumer’s anonymity under certain circumstances, such as a court order. This greatly reduces the incentive to commit crimes such as blackmailing, money laundering and obtaining a ransom safely [vSN92].

2. Coin Traceability. In addition to double-spending detection, sometimes it is beneficial to have the coin-traceability, such that all the coins withdrawn by a particular payee can be traced if he had double-spent any one of his coins. This serve as a more severe punishment for double-spending.

3. Divisibility. Divisibility supports exact payment in an efficient way without the involvement of the bank B. In systems that are indivisible, computation, communication and storage requirements for exact payments are excessive since users must keep a multitude of coins, similar to the way multiple coins are needed for exact physical payments. On the other hand change-making simply transfers the problem of keeping exact change from users to shops.

4. Transferability. Transferability is the ability to transfer a coin from one user to another without contacting the bank B, just like what we can do with physical cash. Transferability has not been applied in most electronic cash systems mainly due to the danger involved. For example, an overspending user is only identified when the coins he/she spent return to the bank, and a single coin can be over-spent by all users from whose hands it has passed.

5. Efficiency. Considering efficiency, we look at: (1) time and bandwidth needed for the withdrawal, payment and deposit protocols, (2) the size of an electronic
coin, (3) the size of the bank’s database. If the system also supports revocability, divisibility or transferability, we also need to look into the time and space complexities of the respective protocols.

5.1.3 Related Works

As mentioned before, secure e-cash systems require that no one except the bank can produce an electronic coin. Thus, a secure digital signature, being unforgeable, is a good candidate for implementing e-cash. Indeed, Chaum [Cha82] introduced the idea of blind signature, a special class of digital signature, to build the first electronic cash system. The bank can sign on the information associated with the electronic coin in a blinded way without knowing the information about an individual’s whereabouts and lifestyle. Besides, blind signature ensures unlinkability: even the bank is given the message/signature pair at later stage, it is impossible to recollect the corresponding invocation of signing protocol. However, the property that user can ask the bank to blindly sign any messages is undesirable. Cut-and-choose methodology was applied in [CFN88] such that the bank can ensure by statistical probability that the user has not presented a malformed message. But it is very inefficient by nature. Alternatively, later research work proposed using variations of blind signature scheme, such as restrictive blind signature [Bra93b] and partially blind signature [AF96], to prove a user has not breached security.

The digital counterpart of cash offers a possibility over traditional paper cash, namely, divisibility. While it is hard to divide a physical coin into separate coins of smaller momentary values, the same restriction does not apply to its electronic counterpart. Divisibility in e-cash is attractive for improving the bandwidth efficiency for the system tremendously. We use the following notation through this paper for divisible e-cash. The user withdraws a wallet of $K = 2^L$ coins in a single execution of Withdrawal Protocol. Spends $2^\ell$, $\ell \leq L$, coins together can be done more efficiently than repeating Spend Protocol for $2^\ell$ times. Indeed, extensive research has been done regarding divisible e-cash [OO91, Pai92, DC94, EO94, Oka95, CFT98, NS00, CG07]. Nonetheless, with the exception of [CG07], which we shall analyse in greater detail soon, none of the above divisible e-cash systems is truly anonymous. For instance, everyone can tell whether the spending in [Oka95, CFT98] is from the same wallet (i.e., linkable). In [NS00], there exists a trusted party who can revoke the identity of every spender (also known as fair e-cash [CT03]). Moreover, which part of the
wallet that is being used is known. That is, if the payee of transaction one and the payee of transaction two are using the same part of a wallet, everyone can conclude that these two transactions are indeed performed with different wallets.

Compact e-cash system, introduced in [CHL05], also aims at improving bandwidth efficiency. Same as divisible e-cash, users can withdraw efficiently a wallet containing $K$ coins. These coins, however, must be spent one by one.

5.1.4 Our Contributions

We are interested in electronic cash which is truly anonymous. Indeed, in all our constructions, no collusion of any parties together can learn anything about the spending of an honest user. We do not discuss revocability because it is straightforward to add this feature to our system. For instance, one can always require the spender to verifiably encrypt his identity under the public key of the administrative party. Nonetheless, the contributions we made is threefold. Firstly, we follow the idea of compact e-cash and propose some improvements in the sense that how Spend Protocol can be optimised in several ways. The result is described in Section 5.3. Secondly, we make use of bounded accumulator described in Section 4.1.2 and ESS+ described in Section 3.3 to propose a new way of constructing compact e-cash which supports more efficient double-spender detection. There is a slight flaw in the original result that appeared in [AWSM07]. We outline the flaw and provide a revised version. They are described in Section 5.4. Finally, we point out some subtleties about the first divisible e-cash that is truly anonymous [CG07]. We then build a practical divisible e-cash from bounded accumulator and ESS+ signature. These results are presented in Section 5.5. To commence with, we present the formal security model for an electronic cash scheme in the next section.

5.2 Syntax

An electronic cash system is a tuple $(\text{BGen}, \text{UGen}, \text{Withdrawal Protocol}, \text{Spend Protocol}, \text{Deposit Protocol}, \text{Revoke}, \text{VerifyGuilt})$ of seven polynomial-time algorithms/protocols between three entities, namely, the bank $\text{B}$, the merchant $\text{M}$ and the user $\text{U}$.

(BGen.) On input a security parameter $1^\lambda$, the algorithm outputs the bank $\text{B}$’s key pair $(bpk, bsk)$, which includes wallet size $K$. 

(UGen.) On input $bpk$, the algorithm outputs a key pair $(pk_U, sk_U)$ (resp. $(pk_M, sk_M)$) for user $U$ (resp. merchant $M$). Note that we assume every user (resp. merchant) in the system is equipped with a public key which uniquely identifies the user (resp. merchant).

(Withdrawal Protocol.) $U$ with input $(pk_U, sk_U)$ wishes to withdraw a wallet $W$ of $K$ coins from $B$ (with input $(bpk, bsk)$). Upon successful termination of the protocol, $U$’s output is a wallet $W$ while the bank might possibly retain certain information $b_W$. In particular, $pk_U$ is known to the bank in this protocol since the bank needs to deduct the user from his account. Normal e-cash is just a special case under this definition when $K = 1$ while for all our purpose we assume $K = 2^L$ for some integer $L$ for divisible e-cash. Wallet $W$ is associated with a counter balance, which is $K$ at the beginning to indicate there are $K$ coins left in the wallet. Sometimes we simply say balance of $W$ to refer to the value of this counter.

(Spend Protocol.) This is the protocol when $U$ (with input $(W, bpk, pk_U, sk_U, pk_M)$) spends $k$ coins (such that balance of $W$ is not less than $k$), to merchant $M$ (with input $(bpk, pk_M, sk_M)$). Upon successful completion of the protocol, $M$ obtains a proof of validity $\pi$ and possibly some auxiliary information $aux$, and outputs accept/reject, depending on whether the payment is accepted. $U$’s output is an updated wallet $W'$ whose counter balance is decreased by $k$. Normal e-cash or compact e-cash is a special case under this definition when $k$ is always 1 while for all our purpose, we let $k = 2^\ell$ for some integer $\ell$ for divisible e-cash.

(Deposit Protocol.) This is the protocol when $M$ (with input $(bpk, pk_M, sk_M, \pi_k, aux)$) deposits the money received from $U$ to $B$ (with input $(bpk, bsk, pk_M)$). $M$ submits $(\pi_k, aux)$ to $B$ who recovers from $(\pi_k, aux)$ the $k$ serial numbers $S_i$. Each serial number uniquely identifies one electronic coin. The bank checks if any duplicated serial number exists in its database. If not, it deposits $M$ directly. Otherwise, it invokes Revoke described below to find out the identity of the double-spender. If $M$ is not the double-spender, $B$ still credits $M$ while charges back the actual double-spender.

(Revoke.) On input two Deposit Protocol protocol transcripts $(\pi_k, aux)$ and $(\pi'_2k', aux')$ which involves a duplicated serial number $S$, this algorithm outputs $pk$, ...
the identity of the double-spender, and a proof-of-correctness $\pi_{pk}$.

(VerifyGuilt.) On input two Deposit Protocol protocol transcripts $(\pi_k, \text{aux})$, $(\pi'_k, \text{aux}')$, a public key $pk$ and a proof of correctness $\pi_{pk}$, this algorithm outputs valid/invalid which indicates whether $pk$ is the identity of the double-spender or $B$ is cheating.

The definition is a bit restricted in the sense that we assume every coin is associated with one unique serial number for which the bank uses to identify double-spending. However, it seems that all existing scheme fits into this definition.

Correctness of an e-cash scheme is defined as follows.

(Correctness for User.) It is required whenever an honest user obtains $W$ from the bank who might be dishonest, an honest merchant shall output accept when the user engages with the merchant in Spend Protocol.

(Correctness for Merchant.) It is required whenever an honest merchant obtains $(\pi_k, \text{aux})$ from some execution of Spend Protocol with some user who might be dishonest, there is a guarantee that this transaction will be accepted by the honest bank.

### 5.2.1 Security Model

As discussed, security requirements of an e-cash scheme include Balance, Anonymity and exculpability. We use a game-based approach to define the security formally. The adversary’s capabilities are modeled by arbitrary and adaptive queries to oracles. The oracles are stateful and together they maintain several sets $U_A$, $U_H$, $D$, $U_W$ which are all initialised to $\emptyset$ and counter $i$, $\text{bal}_A$ are initialised to 0. Looking ahead, set $U_A$ and $U_H$ represent the sets of adversary-controlled users and honest users respectively. Set $U_W$ represents the wallets possessed by honest users while set $D$ represents the electronic coins spent to honest merchants. Counter $i$ indexes all users in the system while counter $\text{bal}_A$ is the balance held by the adversary. The oracles are defined as follows.

- $O_U$. This oracle allows the adversary $A$ to add a user into the system in two ways. If a public $pk_A$ is supplied, the oracle added $pk_A$ to set $U_A$. If no public

---

1It can be seen that it is the bank’s responsibility to identify the double-spender. The rationale behind is that a user can always spend the same coin to different merchants in an offline e-cash system and the merchant has no way to detect such a double-spending.
5.2. Syntax

key is supplied, the oracle runs UGen to obtain \((pk_U, sk_U)\). \(pk_U\) is added to set \(U_H\) and is also returned to \(\mathcal{A}\).

- \(\mathcal{O}_{WH}\). This oracle allows the adversary \(\mathcal{A}\) to instruct an execution of Withdrawal Protocol. \(\mathcal{A}\) supplied a public key \(pk \in U_A \cup U_H\). If \(pk \in U_H\), the oracle simulates a protocol run between honest user \(pk\) and an honest bank. An entry \((i, W_i, pk, K)\) is added to the set \(U_W\). Counter \(i\) is then incremented by 1. The index \(i\) is given to the adversary. On the other hand, if \(pk \in U_A\), the oracle performs the role of an honest bank and conducts Withdrawal Protocol with the adversary. Upon successful termination of the protocol, the counter \(\text{bal}_A\) is increased by \(K\).

- \(\mathcal{O}_{WA}\). This oracle allows the adversary \(\mathcal{A}\) to instruct an execution of Withdrawal Protocol. \(\mathcal{A}\) supplied a public key \(pk \in U_H\). The oracle engages in the protocol as an honest user \(pk\) with the adversary-controlled bank. Upon successful termination of the protocol, an entry \((i, W_i, pk, K)\) is added to set \(U_W\). Counter \(i\) is then incremented by 1.

- \(\mathcal{O}_S\). This oracle allows the adversary \(\mathcal{A}\) to instruct an execution of Spend Protocol. \(\mathcal{A}\) supplied an index \(i\) or a public key \(pk_U \in U_A\), a public key \(pk_M \in U_A \cup U_H\) and an amount of transaction \(k\). The oracle handles three cases.

1. If \(pk_M \in U_H\) and there exists an entry \((i, W_i, pk, \text{bal}_i) \in U_W\), the oracle simulates the Spend Protocol between an honest user \(pk\) (using wallet \(W_i\)) and an honest merchant \(pk_M\). Of course, the protocol will be executed only if \(\text{bal}_i\) is not less than \(k\). The resulting protocol transcript is given to the adversary. The entry \((i, W_i, pk, \text{bal}_i) \in U_W\) is updated to \((i, W_i', pk, \text{bal}_i - k)\), where \(W_i'\) is the updated wallet after execution of the protocol.

2. If \(pk_M \in U_A\) and there exists an entry \((i, W_i, pk, \text{bal}_i) \in U_W\), the oracle acts as an honest user \(pk\) (using wallet \(W_i\)) engages in Spend Protocol with the adversary-controlled merchant \(pk_M\). Again, the protocol will be executed only if \(\text{bal}_i \geq k\). Upon successful termination of the protocol, the counter \(\text{bal}_A\) is increased by \(k\) while the entry \((i, W_i, pk, \text{bal}_i)\) is updated to \((i, W_i', pk, \text{bal}_i - k)\).
3. If $pk_U \in U_A$ and $pk_M \in U_H$, the oracle acts as an honest merchant engages in the Spend Protocol with the adversary-controlled user. Upon successful termination of the protocol, the counter $\text{bal}_A$ is decreased by $k$. The tuple, $(\pi_k, \text{aux}, pk_M, sk_M)$, where the first two entries is the output of the merchant from the protocol, is added to set $D$.

The following game formally defines the property balance for an e-cash.

**Definition 5.1 (GAME Balance)**

(Initialisation Phase.) The challenger $C$ takes a sufficiently large security parameter $\lambda$ and runs $\text{BGen}$ to generate $(bpk, bsk)$. $C$ keeps $bsk$ to itself and sends $bpk$ to adversary $A$.

(Probing Phase.) The adversary $A$ perform a polynomially bounded number of queries to the oracles except $O_{WA}$ in an adaptive manner.

(End Game Phase.) When $A$ decides it is in the End Game Phase, $C$ first runs the Deposit Protocol using the entries in set $D$. $A$ wins the game if it can then execute the Deposit Protocol for a total of more than $\text{bal}_A$ dollar to $C$ such that $\text{Revoke}$ does not output any $pk \in U_A$.

The advantage of $A$ is defined as the probability that $A$ wins.

The following game between a challenger $C$ and an adversary $A$ formally defines anonymity.

**Definition 5.2 (GAME Anonymity)**

(Initialisation Phase.) The challenger $C$ takes a sufficiently large security parameter $\lambda$ and runs $\text{BGen}$ to generate $(bpk, bsk)$. $C$ sends both $(bpk, bsk)$ to adversary $A$.

(Probing Phase.) The adversary $A$ can perform a polynomially bounded number of queries to the oracles except $O_{WH}$ in an adaptive manner.

(Challenge Phase.) $A$ chooses two different indices $i_0, i_1 \in U_W$, a public key $pk_M \in U_A \cup U_H$ and a value $k$ such that $(i_0, W_0, pk_0, \text{bal}_0), (i_1, W_1, pk_1, \text{bal}_1) \in U_W$.\footnote{It is chosen in such a way that the advantage of any computationally bounded adversary is negligible in $\lambda$.}
with the constrict that \( \text{bal}_0, \text{bal}_1 \geq k \). Note that \( pk_0 = pk_1 \) is allowed. \( C \) flips a fair coin \( b \in \{0, 1\} \) and engages in Spend Protocol as user \( pk_b \) (using wallet \( W_b \)) with merchant \( pk_M \) on a transaction that worths \( k \) dollars.

(End Game Phase.) The adversary \( \mathcal{A} \) outputs a guess bit \( b' \).

\( \mathcal{A} \) wins the above game if \( b = b' \). The advantage of \( \mathcal{A} \) is defined as the probability that \( \mathcal{A} \) wins minus \( \frac{1}{2} \).

The following game between a challenger \( C \) and an adversary \( \mathcal{A} \) formally defines exculpability.

**Definition 5.3 (GAME Exculpability)**

(Initialisation Phase.) The challenger \( C \) takes a sufficiently large security parameter \( \lambda \) and runs \( BGen \) to generate \( (bpk, bsk) \). \( C \) sends both \( (bpk, bsk) \) to adversary \( \mathcal{A} \).

(Probing Phase.) The adversary \( \mathcal{A} \) can perform a polynomially bounded number of queries to the oracles except \( O_{WA} \) in an adaptive manner.

(End Game Phase.) When \( \mathcal{A} \) decides it is in the End Game Phase, \( C \) first runs Deposit Protocol using the entries in set \( D \) with \( \mathcal{A} \) acting as bank. \( \mathcal{A} \) then outputs \( (pk, \pi_{pk}) \). \( \mathcal{A} \) wins the game if \( \text{VerifyGuilt} \) with input \( (pk, \pi_{pk}) \) outputs valid and \( pk \in U_H \).

The advantage of \( \mathcal{A} \) is defined as the probability that \( \mathcal{A} \) wins.

An e-cash scheme is secure if no PPT adversary can win in GAME Balance, GAME Anonymity and GAME Exculpability with non-negligible advantage.

### 5.3 Practical Compact E-Cash

Camenisch, Hohenberger and Lysyanskaya [CHL05] proposed a secure e-cash scheme (which we shall refer to as CHL scheme from now on) which is compact to address the efficiency issue. In their scheme, a wallet containing \( K \) coins can be withdrawn and stored in complexity \( O(\lambda + \log(K)) \) for a security parameter \( \lambda \), where each coin can be spent unlinkably with complexity \( O(\lambda \log(K)) \). We shall briefly review their scheme, and present how Spend Protocol can be optimised in several ways.
5.3.1 An Overview of CHL Compact E-Cash

Roughly speaking, CHL scheme is built from a CL-Signatures CL Sig and a pseudorandom function $vrf$ \cite{DY05} with efficient proof of correctness of its output. For simplicity, we assume the key pairs of a user is of the form $(g^u, u) \in (G, \mathbb{Z}_p)$ for some cyclic group $G = \langle g \rangle$ such that $|G| = p$.

When a user $U$ withdraws a wallet $W$ containing $K$ coins from bank $B$, $U$ obtains a CL Sig $\sigma$ from the bank on the three values $(u, s, t)$ using the Issue protocol. In particular, $B$ learns nothing on $(u, s, t)$ such that $s$ and $t$ are just random numbers. The wallet of $U$ is thus $W = (\sigma, u, s, t, J)$ where $J$ is the balance of the wallet and is $K$ initially.

Every coin is uniquely identified with a serial number $S = vrf(s, J)$. Each time when a coin is spent, the balance of the wallet is decreased by 1 and when it reaches zero, there are no coins left. Thus, each wallet contains exactly $K$ coins.

To spend a coin to merchant $M$ with public key $pk_M$, both $U$ and $M$ first agree on certain transaction information $I$ which includes $pk_M$ and a random nonce from the merchant. $U$ uses $pk_M$ and those information to generate a random challenge $R^3$. $U$ computes serial number $S = vrf(s, J)$ of the coin being spent and also a value $T$ called a tracing tag. The value of $T$ is defined as $T = g^u vrf(t, J)^R$. Finally, $U$ produces a commitment $C = \text{Commit}(u, s, t, J; r)$. Note that it is not necessary in actual implementation. However, it is easier to present the scheme this way. Finally, the user produces an SPK $\pi_S$ which serves as a proof of the following:

1. $C$ opens to values $(u, s, t, J)$.

2. Serial number $S$ is correctly formed ($S = vrf(s, J)$).

3. Tracing tag $T$ is correctly formed ($T = g^u vrf(t, J)^R$).

4. Knowledge of $\sigma$ which is a valid CL Sig on values $(u, s, t)$.

5. Wallet contains sufficient balance, $(K \geq J \geq 1)$.

$M$ verifies the $\pi_S$ and accepts the payment if it is valid. $M$ stores $(\pi_S, S, T, R, I)$. The Deposit Protocol is simple. $M$ submits $(\pi_S, S, T, R, I)$ to $B$ who verifies $\pi_S$ and credits the merchant if it is valid, $R$ is fresh and $pk_M$ included in $I$ is the correct public key of $M$. Item 5, sometimes denoted as exact range proof, is the most

\footnote{The straightforward way is to set $R$ to be the output of some cryptographic hash function on $pk_M$ and $I$.}
expensive operation in CHL. CHL employs the technique from [Bou00] which has a complexity of $O(\lambda \log(K))$, where $\lambda$ is the security parameter, and works in group with unknown order. Under current practice when RSA modulus is taken to be 1024-bit, one exact range proof takes about 2 kB of communication cost.

$B$ stores all the serial numbers in its database. For any two Deposit Protocol requests, say $(\pi_S, S, T, R)$ and $(\pi'_S, S', T', R')$, if it happens that $S = S'$, they constitute a double-spending. The bank then computes $(T'R')^{1/R-R}$, which is the public key of the double-spender.

Note that for the balance counter $J$ taking valid values $K$ to 1, $U$ can engage in a Spend Protocol for $K$ times. In case $U$ tries to double-spend, he has to reuse some of the serial number $S$. The revocation of double-spender does not work if $R = R'$. Thus, it is important to ensure $R$ is fresh during Deposit Protocol. Recall that $R$ is the output of some hash function on input the public key of $M$ and transaction information which includes a random nonce from $M$. If $R$ is not fresh, it is safe to conclude that the $M$ is cheating and consequently, the bank can reject any deposit request having the same $R$.

### 5.3.2 Generic Construction of Practical Compact E-Cash

Basically, our practical compact e-cash is built following the idea from CHL. In particular, we optimise protocol of spending $k$ coins together. When $k = K$, we achieve a complexity of $O(\lambda)$. For other cases, the actual implementation is much more efficient than $k$ parallel executions of Spend Protocol of 1 coin. We also incorporate the technique due to [TS06] to achieve complexity of $O(\lambda)$ instead of $O(\lambda \log(K))$ in Spend Protocol. The tradeoff is that public key size of the bank becomes $O(\lambda K)$.

(BGen.) Let $\text{vrf}(\cdot)$ be a pseudorandom function with efficient proof of correctness of its output as discussed. Let $\text{Sig}_1$, $\text{Sig}_2$ be two CL Sig’s. In practice, they might be same or different signature. The bank invokes $\text{Sig}_1.\text{KeyGen}$ to obtain $(\text{Sig}_1.pk, \text{Sig}_1.sk)$. The bank also invokes $\text{Sig}_2.\text{KeyGen}$ to obtain $(\text{Sig}_2.pk, \text{Sig}_2.sk)$. Define the value $K$ which is the momentary value of a wallet. The bank also chooses a cryptographic hash function $H$. The bank computes $\sigma_i \leftarrow \text{Sig}_2.\text{Sign}_{\text{Sig}_2.sk}(i)$ for $i = 1$ to $K$. The bank also chooses a cyclic group $G = \langle g \rangle$ of order $p$ such that $\mathbb{Z}_p$ is within the message space of $\text{Sig}_1$. Further chooses two random generators $g_1, g_2 \in_R G$. The public key $b_{pk}$
is \((\text{Sig}_1.pk, \text{Sig}_2.pk, \text{vrf}(\cdot), H, K, \sigma_1, \ldots, \sigma_K, g, g_1, g_2)\) and the private key \(\text{bsk}\) is \((\text{Sig}_1.sk)\).

The choice of \(\text{Sig}_2\) worths some discussion. Security requirement for \(\text{Sig}_2\) is weaker than that of \(\text{Sig}_1\). Specifically, \(\text{Sig}_2\) is used to issue signature on \(1 \) to \(K\). Thus, it only needs to be unforgeable under fixed message attack. It is possible to employ certain weaker signature schemes to improve performance.

Another issue is that \(\text{Sig}_2.sk\) is no longer needed after this stage and the bank should safely delete it. The reason is that, the knowledge of \(\text{Sig}_2.sk\) allows the adversary to forge new message, which in turn allow forging of an electronic coin and breaking the balance property. On the other hand, the presence of \(\text{Sig}_2.sk\) does not help breaking anonymity nor exculpability. Thus, it is against the bank’s interest to keep \(\text{Sig}_2.sk\).

\((\text{UGen.})\) We employ the common discrete logarithm type key pairs. Specially, the algorithm randomly generates \(u \in_R \mathbb{Z}_p\) and computes \(g^u\). It outputs \((pk_U, sk_U)\) as \((g^u, u)\).

\((\text{Withdrawal Protocol.})\) User \(U\), with input \((pk_U, sk_U, bpk)\), obtains a signature from the bank \(B\), with input \((pk_U, bsk)\), \(\sigma_W \leftarrow \text{Sig}_1.\text{Sign}_{\text{Sig}_1.sk}(s, t, u, y)\) using the \text{Issue} of \(\text{Sig}_1\) so that \(B\) learns nothing about \((s, t, u, y)\). \((s, t, y)\) are random numbers generated by \(U\) and during the course \(U\) is required to prove to \(B\) that \(u\) is indeed his private key. \(U\) parses his wallet \(W\) as \((s, t, u, y, J)\) where \(J = K\) is the initial balance of the wallet.

\((\text{Spend Protocol.})\) \(U\) with input \((W = (\sigma_W, s, t, u, y, J), pk_M, bpk)\) engages with merchant \(M\) with input \((pk_M, sk_M, bpk)\) in the spend protocol. The parties first agree on the transaction amount \(k\) (such that \(k \leq J\)), random challenge \(R = H(pk_M||\text{nonce}||\text{info})\) which includes a random nonce from the merchant and some other information \(\text{info}\) related to the transaction.

Depending on the value \(k\), \(U\) uses one of the following sub-protocols.

- \((k = 1.)\) \(U\) computes serial number \(S = \text{vrf}(s, J)\), tracing tag \(T =\)
5.3. Practical Compact E-Cash

g^u \text{vrf}(t, J)^R \text{ and computes the following } \text{SPK}_{26} \text{ (denoted by } \pi_S).\

\text{SPK}_{26}\left\{ \begin{array}{l}
(\sigma_{W, s, t, u, y, \hat{\sigma}, J}) : \\
\text{valid } \leftarrow \text{ Sig}_1.\text{Verify}_{\text{Sig}_1.\text{pk}}(\sigma_{W, s, t, u, y}) \land \\
\text{valid } \leftarrow \text{ Sig}_2.\text{Verify}_{\text{Sig}_2.\text{pk}}(\hat{\sigma}, J) \land \\
S = \text{ vrf}(s, J) \land \\
T = g^u \text{vrf}(t, J)^R \\
\end{array}\right\}(R)

U sends \((S, T, \pi_S)\) to M. If \(\pi_S\) is valid, M accepts the payment and stores the coin as \((\pi_S, S, T, R, \text{nonce, info})\). U decrements J by 1.

• \((k = K.)\) U computes a special tracing tag \(T_y = g^u \text{vrf}(y, 0)^R\), a special value \(C = g_1^s g_2^t\). U computes the following \text{SPK}_{27} \text{ (denoted by } \pi_C).

\text{SPK}_{27}\left\{ \begin{array}{l}
(\sigma_{W, s, t, u, y, \hat{\sigma}, J}) : \\
\text{valid } \leftarrow \text{ Sig}_1.\text{Verify}_{\text{Sig}_1.\text{pk}}(\sigma_{W, s, t, u, y}) \land \\
C = g_1^s g_2^t \land \\
T_y = g^u \text{vrf}(y, 0)^R \\
\end{array}\right\}(R)

U sends \((T_y, C, s, t, \pi_C)\) to M. If \(\pi_C\) is valid and \(C = g_1^s g_2^t\), M accepts the payment and stores the \(K\) coins as \((\pi_C, \pi_T, \pi_{C}, \pi_{T_y}, \text{nonce, info})\). U decrements J by \(K\). That is, the wallet is used up.

• \((K > k > 1.)\) U computes serial numbers \(S_i = \text{vrf}(s, J - i)\), tracing tag \(T = g^u \text{vrf}(t, J - i)^R\), for \(i = 0\) to \(k - 1\). and computes the following \text{SPK}_{28} \text{ (denoted by } \pi_B).

\text{SPK}_{28}\left\{ \begin{array}{l}
(\sigma_{W, s, t, u, y, \hat{\sigma}, J}) : \\
\text{valid } \leftarrow \text{ Sig}_1.\text{Verify}_{\text{Sig}_1.\text{pk}}(\sigma_{W, s, t, u, y}) \land \\
\text{valid } \leftarrow \text{ Sig}_2.\text{Verify}_{\text{Sig}_2.\text{pk}}(\hat{\sigma}, J) \land \\
S_0 = \text{vrf}(s, J) \land \\
T_0 = g^u \text{vrf}(t, J)^R \land \\
\vdots \\
S_{J-k+1} = \text{vrf}(s, J - k + 1) \land \\
T_{J-k+1} = g^u \text{vrf}(t, J - k + 1)^R \land \\
\text{valid } \leftarrow \text{ Sig}_2.\text{Verify}_{\text{Sig}_2.\text{pk}}(\hat{\sigma}, J-k+1, J-l+1) \\
\end{array}\right\}(R)

U sends \(\{S_i, T_i\}_{i=0}^{k-1}, \pi_B\) to M. If \(\pi_B\) is valid, M accepts the payment and stores the \(k\) coins as \((\pi_B, \{S_i, T_i\}_{i=0}^{k-1}, R, \text{nonce, info})\). U decrements J by \(k\).
(Deposit Protocol.) M submits \((\pi_S, S, T, R, \text{nonce}, info), (\pi_C, T_y, C, s, t, R, \text{nonce}, info), (\pi_B, \{S_i, T_i\}_{i=0}^{k-1}, R, \text{nonce}, info)\) to bank B for a deposit of 1, K or \(k\) coins respectively. B first checks if \(R \overset{?}{=} H(pk_M||\text{nonce}||info)\), \(pk_M\) is indeed the public key of the \(M\) and nonce is fresh. Then B verifies \(\pi_S, \pi_C\) or \(\pi_B\). If the checks are valid, B credits M. If it is a deposit for \(K\) coins, B computes \(S_i = \text{vrf}(s, i)\) for \(i = 1\) to \(K\).

Finally, the bank checks if the serial number(s) is in its database. If not, stores these serial number(s). Otherwise, B invokes Revoke with the input of the two deposit requests that produces the duplicated serial number.

(Revoke.) Considered the two deposit requests that involve the same serial number. In the case the deposit request is of \(k \neq k\) coins, the number \((S, T, R)\) are well-defined. On the other hand, if it is from deposit request of \(K\) coins, while serial number \(S\) can be computed from \(s\), the tracing tag \(T\) cannot be computed using \(t\). Thus, there are three cases and B has to deal with them separately.

1. \((S, T, R)\) is well-defined for both deposit requests. Let \((S, T, R)\) and \((S, T', R')\) be the two entries. B computes \(pk_U\) by \(\left(\frac{T'R}{TR}\right)^{\frac{1}{2}}\).

2. \((S, T, R)\) is well-defined for one of the deposit requests. Let \((S, T, R)\) be one of the entries. The other entry will be \((\pi_C, T_y, C, s, t, R', \text{nonce}, info)\) such that there exists an index \(i\) with \(S = \text{vrf}(s, i)\). B computes \(pk_U\) by \(\frac{T}{\text{vrf}(t, i)^{\frac{1}{2}}}\).

3. \((S, T, R)\) is not well-defined for both requests. The two entries shall be \((\pi_C, T_y, C, s, t, R, \text{nonce}, info)\) and \((\pi'_C, T'_y, C', s', t', R', \text{nonce}', info')\) such that there exist two indices \(i\) and \(i'\) such that \(\text{vrf}(s, i) = \text{vrf}(s', i')\). B computes \(pk_U\) by \(\left(\frac{T'R'}{T'_y}\right)^{\frac{1}{2}}\).

After obtaining \(pk_U\), B checks if the public key indeed belongs to some valid users. If such user does not exist, it is a false linking. This is possible when there exists \(J, J' \leq K\) such that \(J + s = J' + s'\) for two different wallets. Specifically, if the random number \(s\) chosen by different users during the withdrawal protocol is within \(K\), a false linking occurs. This, however, happens only with negligible probability when \(s\) is randomly chosen. The proof-of-correctness

\[^{4}\text{We assume some kind of authentication is done.}\]
for the bank in \texttt{Revoke} is the two deposit requests that generate the repeated serial number.

\textit{(VerifyGuilty.)} Every one can check if B is honestly running \texttt{Revoke} as it does not require any of the secrets from B.

\subsection*{5.3.3 Security Analysis of Our Generic Construction}

We give an outline of why our generic practical compact e-cash is secure. It is only a sketch because a complete formal proof is extremely tedious and at the same time, the techniques involved are straightforward. This has often been the case for complex payment schemes in the literature. The same applies to the security analysis in Section 5.4.4 and Section 5.5.5.

\textit{Proof:} (Sketch.)

\textbf{Balance.}

Let \( A \) be an adversary that win GAME Balance with non-negligible advantage. We outline why the success probability of \( A \) is negligible due to the unforgeability of \( \text{CL Sig} \; \text{Sig}_1 \) and \( \text{Sig}_2 \), by constructing a simulator \( S \) acting as challenger \( C \). \( S \) is given two signing oracles for \( \text{Sig}_1 \) and \( \text{Sig}_2 \) respectively and its goal is to forged a signature for \( \text{Sig}_1 \) or \( \text{Sig}_2 \). Firstly, \( S \) invokes the signing oracle of \( \text{Sig}_2 \) on input messages 1, \ldots, \( K \) to produce \( \hat{\sigma}_i \)'s in the public key of the bank.

\textbf{Oracle Simulations.}

\( \mathcal{O}_U \). If a public key is supplied, stored it in \( \mathcal{U}_A \). Otherwise, randomly generate \( u \) and set \( pk_U \) to \( g^u \). Add \( pk_U \) to \( \mathcal{U}_H \) and returns \( pk_U \) to adversary.

\( \mathcal{O}_{WH} \). If \( pk_U \in \mathcal{U}_A \), simulate the protocol by invoking the signing oracle of \( \text{Sig}_1 \) with \texttt{issue}. Increase \( \text{bal}_A \) by \( K \). Note that while the messages being signed is unknown to \( S \), it is of the form \((s,t,u,y)\).

If \( pk_U \in \mathcal{U}_H \), randomly generate \((s,t,y)\) and balance \( J \) as if he has obtained a signature from the bank on \((s,t,u,y)\).

\( \mathcal{O}_S \). If it involves an honest user (regardless of whether merchant is adversary-controlled or not), the oracle simply uses the knowledge simulator to finish the view of the protocol. More specially, for user \((g^u,u)\) with sufficient balance, simulate the protocol as if he is in possession of a wallet \( \mathcal{W} = (\sigma_W, s, t, u, y, J) \).
This requires the knowledge simulator of $\sigma_Y$ and its presence is due to the Prove privacy of Sig$_1$ and Sig$_2$.

**Outputs.** Finally, $A$ runs deposits $\text{bal}_A + 1$ to $S$. $S$ uses $s$ to calculate the $K$ serial numbers corresponding to that single deposit operation in case it is a deposit of $K$ coins.

$A$ wins the game either by (1) all the $\text{bal}_A + 1$ serial numbers in Deposit Protocol are unique or (2) some of the serial numbers are duplicated but Revoke on the corresponding Deposit Protocol transcripts does not output any $pk \in U_A$. Now we analyse these two cases separately.

Case (1): Due to the security of Sig$_1$, each $O_{WH}$ query only gives $A$ one single $s$ to work with. Due to the setting of the game, the total “valid” serial numbers obtained by $A$ from $S$, minus the serial numbers that has been presented to $S$ during $O_S$ query is $\text{bal}_A$.

$A$ can only win in Case (1) by convincing $S$ to accept a serial number $S$ that is not one of these $\text{bal}_A$ serial numbers. Then $A$ must have conducted a false proof as part of the signature of knowledge, $\text{SPK}_{26}$, such that one of the following is fake:

1. Possession of a Sig$_1$ signature on block of messages $(s, t, u, y, J)$.
2. $S = \text{vrf}(s, J)$.
3. Possession of Sig$_2$ signature on $J$.

If it is $\text{SPK}_{27}$, $A$ has conducted a false proof so that one of the following is fake:

1. Possession of a Sig$_1$ signature on block of messages $(s, t)$.
2. $C = g^{s \cdot t}$
3. $(s, t)$ is an opening of $C$.

If it is $\text{SPK}_{28}$ for $k$ coins, one of the following is fake:

1. Possession of Sig$_1$ signature on block of messages $(s, t, u, y)$.
2. $S_i = \text{vrf}(s, i)$ for $i = J$ to $J - k + 1$.
3. Possession of Sig$_2$ signatures on $J$ and $J - k + 1$. 
Fake proof of possession of \textbf{CL Sig} happens with negligible probability. Fake proof of $S$ (and $T$) are well-formed happens with negligible probability under the discrete logarithm assumption. Thus, $A$ can only win if he could generate a new valid $\text{Sig}_1$ signature on $(s, t, u, y)$ or a new valid $\text{Sig}_2$ signature on $J \notin \{1, \ldots, K\}$. Thus $S$ can use $A$ to break the security of $\text{Sig}_1$ or $\text{Sig}_2$.

Case (2): We have shown in case (1) that $A$ cannot convince an $S$ to accept an invalid serial number with non-negligible probability. We now suppose duplicated $S$ or $s$ are accepted.

It remains to show the associated $T$, or $(t, T_y)$ is bounded by specification except with negligible probability so that the correctness of the Revoke implies the recovering of a $pk \in U_A$. Due to the soundness of the proof of knowledge protocol (SPK$_{26}$, SPK$_{28}$), $T = g^{u \text{vrf}(t, J)}^R$ is the only tracing tag to accompany serial number $S = \text{vrf}(s, J)$. For SPK$_{27}$, special tracing tag $T_y = g^{u \text{vrf}(y, 0)}^R$ is the only valid tag accompany the value $C = g_1^s g_2^t$ with respect to $\text{Sig}_1$ signature on $(s, t, u, y)$. Since $R$ is chosen by the random oracle, the two $R$’s shall be different in the two transactions. To deviate from these valid tags, $A$ must fake one of the proofs which we have already shown to happen with negligible probability only. Thus, $A$’s success probability is negligible.

\textbf{Anonymity.} Anonymity is straightforward. Due to Issue and Prove privacy, Withdrawal Protocol, Spend Protocol and Deposit Protocol do not contain any information about the underlying secret except the values $S$, $T$ and $pk_u$. More specifically, recall that for a wallet $W = (\sigma_W, s, t, u, y)$, the numbers $(s, t, y)$ are not known to anybody except the underlying user. Some information about $u$ is known since $g^u$ is the public key. The only information leaked about a user or wallet during Withdrawal Protocol is $g^u$, $\sigma_W$ (since Issue privacy of \textbf{CL Sig} does not guarantee privacy about the signature issued) while the only information leaked about a user or wallet during Spend Protocol is $S$ and $T$ or $(s, t, T_y)$ in case of spending $K$ coins. The randomness of $S$ and $T$ is guaranteed by the security property of the underlying \text{vrf}. Finally, $s, t, T_y$ do leak information on $s$ and $t$, but they will only be used once since the wallet has been used up once the protocol of spending $K$ coins is being executed.

\textbf{Exculpability.}  In Revoke, either one or both transcript contains the proof of correctness of $T$ or $T_y$, which involves proving knowledge of the user secret $u$. To
slander an honest user, adversary without knowledge of user secret $u$ has to fake the knowledge which involves knowledge of $u$ to base $g$. This happens with negligible probability under the discrete logarithm assumption.

\[ \square \]

5.3.4 Actual Construction of Practical Compact E-Cash

Following the generic construction, practical compact e-cash can be constructed readily by choosing a suitable CL Sig and vrf. One additional criterion is that $PK\{(t, x, J) : T = g^x vrf(t, J)^R\}$ can be efficiently done since this may not be efficient for any vrf. Below we present an instantiation following the generic construction based on the BBS+ signature, together with the vrf due to [DY05] described below. The same vrf was used in the original CHL scheme. Since the goal of our constructions is, as its name suggested, practical, we also choose the weak version of BB signature as $\text{Sig}_2$ in the generic construction. Finally, protocols presented below are optimised for efficiency. While the scheme employs BBS+ signature on values $(t, u, y, r)$, the role of $s$ in the generic construction is performed by $s$ in a BBS+ signature $(\varsigma, e, s)$. The advantage of this is that it disallows deliberate false linking (which does not break the security requirement but is a hassle) by requiring $s$ to be co-generated by the user and the bank. Details shall be clear when we present the actual construction.

Building Blocks

A Pseudorandom Function. We employ a particular construction of pseudorandom function due to Dodis and Yampolskiy [DY05] (DY-PRF). DY-PRF is defined by a tuple $(G, p, g, s)$, where $G = \langle g \rangle$ is a cyclic group of prime order $p$. On input $s, x$, $vrf(s, x)$ is defined as $vrf(s, x) : s, x \mapsto g^{\frac{1}{s+x}}$. The output of vrf is indistinguishable from random elements in $G$, provided that the $q$-DDHI assumption holds, see [CHL05] for details\footnote{The aim of [DY05] is to propose a verifiable random function which has non-interactively verifiable proof by itself, e.g. without Fiat-Shamir heuristic.}

Short BB Signature under weak-chosen message attack. Recall that in the generic construction, $\text{Sig}_2$ is only required to sign “messages” from 1 to $K$. Indeed, the security requirement for $\text{Sig}_2$ is that after seeing the signatures from 1 to $K$,
no one shall be able to output a new signature on “message” other than 1 to K. With this in mind, we are interested in searching for an efficient signature that satisfies this reduced security requirement. The weakly-secure BB Short signature in [BB04] does satisfy our need and is briefly described here. For a secret key \( x \) and public key \( g^x \) in groups equipped with a bilinear map, the signature \( \sigma \) on message \( m \) is just a single group element, \( g^{x+m} \). The corresponding verification equation is \( \epsilon(\sigma, g^x g^m) = \epsilon(g, g) \). The scheme is secure in this reduced security requirement under the \( q \)-SDH assumption.

Our Construction

Common Parameter\(^6\) Let \( \lambda \) be a security parameter. Let \( G_1, G_2, G_T, G_p \) be cyclic groups of prime order \( p \) such that \( p \) is of \( \lambda \)-bit. Let \( \epsilon : G_1 \times G_2 \to G_T \) be a bilinear map and \( \psi : G_2 \to G_1 \) be an isomorphism from \( G_2 \) to \( G_1 \).

(BGen.) The bank \( B \) randomly generates \( g, g_0, g_1, g_2, g_3, g_4 \in_R G_2 \), computes \( g = \psi(g) \) and \( g_i = \psi(g_i) \) for \( i = 0 \) to \( 4 \). \( B \) also generates \( h, h_0 \in_R G_p \). We assume relative discrete logarithms are unknown. This can be done by setting them to be the output of some cryptographic hash function of some publicly known seed (such as the identity of the bank). \( B \) randomly chooses \( \gamma, \gamma_r \in_R Z_p \), computes \( w = g^\gamma \in G_2 \) and \( v = g^{\gamma_r} \in G_2 \). \( B \) then computes \( \hat{\sigma}_i = g^{\frac{1}{\gamma r+i}} \) for \( i = 1 \) to \( K \). Note that \( K \) has to be much smaller than \( 2^\lambda \).

The public key \( bpk \) is \( (G_1, G_2, G_T, G_p, p, \epsilon, \psi, g, g_0, g_1, g_2, g_3, g_4, g, g_0, g_1, g_2, g_3, g_4, w, v, K, \hat{\sigma}_1, \ldots, \hat{\sigma}_K) \) and the secret key \( bsk \) is \( (\gamma) \).

(UGen.) User \( U \) randomly chooses his secret key \( u \in_R Z_p \) and computes his public key as \( pk_u = h^u \in G_p \).

(Withdrawal Protocol.) \( U \) randomly generates \( s', t, y, r \in_R Z_p \), computes \( C' = g_0^s g_1^t g_2^y g_3^u g_4^r \in G_1 \), and sends \( C', pk_u \) to \( B \), and conducts the following protocol.

\[
\text{PK}_{29} \{(s', t, u, y, r) : C' = g_0^s g_1^t g_2^y g_3^u g_4^r \land pk_u = h^u \}
\]

\( B \) verifies \( \text{PK}_{29} \), randomly selects \( s'', e \in_R Z_p \), computes \( \zeta = (C' g_0^{s''}) \frac{1}{\gamma r} \) and returns \( (\zeta, e, s'') \) to \( U \). \( U \) computes \( s = s' + s'' \), checks if \( \epsilon(\zeta, wg^s) = \epsilon(gg_0^s g_1^t g_2^y g_3^u g_4^r, g) \), sets \( J = K \) and stores \( (\zeta, e, s, t, y, u, r, J) \) as his wallet \( W \).

\(^6\)We assume these common parameters are generated by some trusted parties.

\(^7\)\( \gamma_r \) is discarded.
5.3. Practical Compact E-Cash

(Spend Protocol.) U with input \((s, e, t, y, u, r, J)\) engages with merchant M with public key \(pk_M\) for a transaction involving \(k\) coins such that \(J \geq k\). M generates a random number \textit{nonce}, both parities agree on the transaction information \textit{info} and compute \(R = H(pk_M||\text{nonce}||\text{info})\) locally. We assume \(pk_M\) is authentic, that is, U is indeed communicating with merchant with public key \(pk_M\). Depending on the values of \(k\), U and M proceed as follows.

- \((k = 1.)\) U computes serial number \(S = h_0^{\frac{1}{R + k}}\), tracing tag \(T = h_0^{\frac{R}{R + k}}\), auxiliary commitments \(\mathbf{A}_1 = \mathbf{g}_1^{r_1} \mathbf{g}_2^{r_2}, \mathbf{A}_2 = \varsigma \mathbf{g}_2^{r_1}, \mathbf{A}_3 = \mathbf{g}_1^{r_3} \mathbf{g}_4^{r_4}, \mathbf{A}_4 = \tilde{\sigma} f \mathbf{g}_2^{r_3}, \mathbf{A}_5 = \mathbf{g}_1^{r+J} \mathbf{g}_2^{r_5}\) for some randomly generated \(r_1, r_2, r_3, r_4, r_5 \in R \mathbb{Z}_p\) and computes the following \(\text{SPK}_{30}\). Denote \((\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4, \mathbf{A}_5, \text{SPK}_{30})\) by \(\pi_S\).

\[
\begin{align*}
\text{SPK}_{30} & \left\{ \begin{array}{l}
\mathbf{A}_1 = \mathbf{g}_1^{r_1} \mathbf{g}_2^{r_2} \\
1_{G_1} = 1_{G_1}^{-e} \mathbf{g}_1^{\beta_2} \\
\tilde{e}(\mathbf{A}_2, \mathbf{w}) = \tilde{e}(g_0, g)^s \tilde{e}(g_1, g)^t \tilde{e}(g_2, g)^u \tilde{e}(g_3, g)^v \tilde{e}(g_4, g)^w \\
\tilde{e}(\mathbf{g}_2, \mathbf{w}) = \tilde{e}(g_2, g)^w \tilde{e}(\mathbf{g}_2, \mathbf{g}) \tilde{e}(\mathbf{A}_2, \mathbf{g})^{-e} \\
\mathbf{A}_3 = \mathbf{g}_1^{r_3} \mathbf{g}_4^{r_4} \\
1_{G_1} = 1_{G_1}^{-J} \mathbf{g}_1^{\beta_3} \mathbf{g}_4^{\beta_4} \\
\tilde{e}(\mathbf{A}_3, \mathbf{v}) = \tilde{e}(g_2, \mathbf{v})^r \tilde{e}(g_2, \mathbf{g}) \tilde{e}(\mathbf{A}_4, \mathbf{g})^{-J} \\
h_0 = S^s S^J \\
\mathbf{A}_5 = \mathbf{g}_1^{r_5} \mathbf{g}_2^{r_5} \\
1_{G_1} = 1_{G_1}^{-u} \mathbf{g}_1^{\beta_5} \mathbf{g}_2^{\beta_6} \\
h_0^R = T^u T^J h^{-\beta_5}
\end{array} \right\} (R)
\end{align*}
\]

\(\text{SPK}_{30}\) can be produced using standard techniques and we shall omit its details here. Note that U uses \(\beta_1 = r_1 e, \beta_3 = r_3 J\) and \(\beta_5 = u(t + J)\) in the SPK. U sends \((S, T, \pi_S)\) to M. If \(\pi_S\) is valid, M accepts the payment and stores the coin as \((\pi_S, S, T, R, \text{nonce}, \text{info})\). U decrements \(J\) by 1.

- \((k = K.)\) U computes a special tracing tag \(T_y = h_0^{\frac{R}{R + k}}\), a special value \(C = g_1^s g_2^t\), auxiliary commitments \(\mathbf{A}_1 = \mathbf{g}_1^{r_1} \mathbf{g}_2^{r_2}, \mathbf{A}_2 = \varsigma \mathbf{g}_2^{r_1}, \mathbf{A}_3 = \mathbf{g}_1^{y} \mathbf{g}_2^{r_3}\) for some randomly generated \(r_1, r_2, r_3 \in R \mathbb{Z}_p\) and computes the following
5.3. Practical Compact E-Cash

SPK_{31}. Denote \((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \text{SPK}_{31})\) by \(\pi_C\).

\[
\text{SPK}_{31} \left\{ \left( e, s, t, u, y, r, r_1, r_2, r_3, \beta_1, \beta_2, \beta_3, \beta_4 \right) : \right.
\]
\[
\begin{align*}
\mathcal{A}_1 &= g_1^{r_1} g_2^{r_2} \\
1_G &= \mathcal{A}_1^{-e} g_1^{\beta_1} g_2^{\beta_2} \\
\hat{e}(\mathcal{A}_2, w) &= \hat{e}(g_0, g)^{s} \hat{e}(g_1, g)^{t} \hat{e}(g_2, g)^{y} \hat{e}(g_3, g)^{u} \hat{e}(g_4, g)^{r} \\
\hat{e}(g_2, w)^{r_1} \hat{e}(g_2, g)^{\beta_1} \hat{e}(\mathcal{A}_2, g)^{-e} \\
C &= g_1^{y} g_2^{l} \\
\mathcal{A}_3 &= g_1^{u} g_2^{r_3} \\
1_G &= \mathcal{A}_5^{-u} g_1^{\beta_3} g_2^{\beta_4} \\
h_0^R &= T_y y h^{-\beta_3} \\
\left. \right\} (R)
\]

\text{SPK}_{31} \text{ can be produced using standard techniques and we shall omit its details here.} \ U \text{ sends } (T_y, C, s, t, \pi_C) \text{ to } \mathcal{M}. \text{ If } \pi_C \text{ is valid and } C = g_1^{s} g_2^{l}, \mathcal{M} \text{ accepts the payment and stores the } K \text{ coins as } (\pi_C, T_y, C, s, t, R, \text{nonce, info}). \ U \text{ decrements } J \text{ by } K.

\bullet (K > k > 1.) \ U \text{ computes serial numbers } S_i = h_0^{\frac{1}{R} i - R}, \text{ tracing tag } T_i = h_i^{u} h_0^{\frac{1}{R} i - R}, \text{ for } i = 0 \text{ to } k - 1. \ U \text{ also computes auxiliary commitments } \mathcal{A}_1 = g_1^{r_1} g_2^{r_2}, \mathcal{A}_2 = \hat{g}_2^{r_2}, \mathcal{A}_3 = g_1^{r_3} g_2^{r_4}, \mathcal{A}_4 = \hat{g}_2^{r_3}, \mathcal{A}_5 = g_1^{r_5} g_2^{r_6}, \mathcal{A}_6 = \hat{g}_1^{r_4 + 1} g_2^{r_5}, \mathcal{A}_7 = g_1^{r_4 + J} g_2^{r_7} \text{ for some randomly generated } r_1, r_2, r_3, r_4, r_5, r_6, r_7 \in \mathbb{Z}_p \text{ and computes the following SPK}_{32}. \ Denote
5.3. Practical Compact E-Cash

\((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7, \text{SPK}_{32})\) by \(\pi_B\).

\[
\text{SPK}_{32} \left\{ \left( e, s, t, u, y, r, J, r_1, r_2, r_3, r_4, r_5, r_6, r_7, \right) \right. \\
\left. \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8 \right\}
\]

\begin{align*}
\mathcal{A}_1 &= g_1^r v_2^r \\
1_{G_1} &= \mathcal{A}_1^e g_1^{\beta_1} g_2^{\beta_2} \\
\hat{\mathcal{e}}(\mathcal{A}_2, v) &= \hat{\mathcal{e}}(g_0, g)^y \hat{\mathcal{e}}(g_1, g)^t \hat{\mathcal{e}}(g_2, g)^y \hat{\mathcal{e}}(g_3, g)^u \\
\mathcal{A}_3 &= g_1^r v_2^r \\
1_{G_1} &= \mathcal{A}_3^{-y} g_1^{\beta_3} g_2^{\beta_4} \\
\hat{\mathcal{e}}(\mathcal{A}_4, v) &= \hat{\mathcal{e}}(g_2, v)^{y_s} \hat{\mathcal{e}}(g_2, g)^{\beta_3} \hat{\mathcal{e}}(\mathcal{A}_1, g)^{-J} \\
\mathcal{A}_5 &= g_1^r v_2^r \\
1_{G_1} &= \mathcal{A}_5^{-y} g_1^{\beta_5} g_2^{\beta_6} \\
\hat{\mathcal{e}}(\mathcal{A}_6, v) &= \hat{\mathcal{e}}(g_2, v)^{y_s} \hat{\mathcal{e}}(g_2, g)^{\beta_3} \hat{\mathcal{e}}(\mathcal{A}_6, g)^{-J} \\
\mathcal{A}_7 &= g_1^r y_v g_2^{y_v} \\
1_{G_1} &= \mathcal{A}_7^{-y} g_1^{\beta_7} g_2^{\beta_8} \\
\Lambda_{i=0}^{k-1} h_0^t n_i^{t_i} &= S_i^t \mathcal{J}_i \\
\Lambda_{i=0}^{k-1} h_0^{R_i} T_i^{t_i} &= T_i^{q_i} T_i^{J_i} h^{-\beta_7} (h^i)^u \\
\right\}(R)
\]

SPK\(_{32}\) can be produced using standard techniques and we shall omit its details here. \(U\) sends \(\{\{S_i, T_i\}_{i=0}^{k-1}, \pi_B\}\) to \(M\). If \(\pi_B\) is valid, \(M\) accepts the payment and stores the \(k\) coins as \(\{\pi_B, \{S_i, T_i\}_{i=0}^{k-1}, R, \text{nonce}, \text{info}\}\). \(U\) decrements \(J\) by \(k\).

Deposit Protocol, Revoke, VerifyGuilt have been discussed in the generic construction.

5.3.5 Arbitrary Wallet Size for Compact E-Cash

We outline how to modify our system so that wallet size \(K\) is to be chosen by user arbitrarily during Withdrawal Protocol while coins from wallet of different size are indistinguishable in Spend Protocol. The idea is that during the withdrawal protocol, the value \(K\) chosen by the user is also signed by the bank. During spending, the user proves, in zero-knowledge manner, that the counter \(J\) is within 1 to \(k\) where \(k\) is hidden from the merchant. However, inefficient exact range proof [Bou00] has to be employed. This requires setting up another group with unknown order. Specifically,
the change to make is that the user obtains $\sigma_W = \text{Sign}(s, t, u, y, K)$ during the withdrawal protocol while change of $\text{SPK}_{26}$ is shown as follow.

$$\text{SPK}\{ \left( \sigma_W, s, t, u, y, K, J \right) :$$

$$1 \leftarrow \text{Sig}_1.\text{Verify}_{\text{Sig}_1.\text{pk}}(\sigma_W, s, t, u, y, K) \land$$

$$0 \leq K - J \land$$

$$1 \leq J \land$$

$$S = \text{vrf}(s, J) \land$$

$$T = g^u\text{vrf}(t, J)^R \land \} (R)$$

5.4 Compact E-Cash from Bounded Accumulator

In the last section we outlined how existing compact e-cash are constructed from $\text{CL Sig, vrf}$ with the tracing mechanism based on the so-called serial number and tracing tag approach. In this section, we consider e-cash built from another tracing mechanism. Recall that in compact e-cash scheme, the tracing tag approach outputs the double-spender’s identity in the form of $g^u$, while this new approach would simply output $u$, the secret key of the cheating user efficiently.

Later, we discuss extension to e-cash schemes to support full coin-tracing, where this feature is possibly more beneficial. Below, we first outline the idea of a normal e-cash scheme, called short e-cash in [ACS05], which serves as an overview of this new approach in constructing compact e-cash.

5.4.1 Short E-Cash

The version presented here is different from what appeared in [ACS05]. The reason is that this version is more suitable serving as an overview to our compact e-cash from bounded accumulator. Nonetheless, the central idea remains the same. That is, a user with key pairs $(g^u, u)$ obtains a CL signature from the bank on random numbers $(s, t, u)$. Compute a one-time base $h$ as $g^t$, two values $T = h^s$ and $U = h^u$. Then the user uses $s$ as the randomness in a zero-knowledge proof-of-knowledge of the DL of $U$ to base $h$, that is, $u$. For each withdrawal request, the user only obtains one $s$ and if he tries to produce the proof of $U$ to base $h$ twice using the same $s$, the secret key $u$ could be computed. The value $t$ is used to compute a random one-time base $h$ so that with the same secret key $u$, the value $U$ would not be linkable for each withdrawn coin under the DDH assumption.
Compact E-Cash from Bounded Accumulator

(BGen.) Let $\text{Sig} = (\text{KeyGen}, \text{Sign}, \text{Verify})$ be a CL $\text{Sig}$. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of order $p$ such that $\mathbb{Z}_p$ is within the message space of $\text{Sig}_1$. Further let $g_1, g_2 \in_r \mathbb{G}$ be random generators of $\mathbb{G}$. The bank $B$ chooses some cryptographic hash function $H$ and invokes $\text{Sig.Enums}$. Let $\text{Sig}.\text{KeyGen}$ and obtains $(\text{Sig}.pk, \text{Sig}.sk)$. The public key $bpk$ of $B$ is $(\mathbb{G}, p, g, g_1, g_2, \text{Sig}.pk, H)$ and the corresponding secret key $bsk$ is $(\text{Sig}.sk)$.

(UGen.) We employ the common discrete logarithm type key pairs. Specifically, user $U$ randomly generates $u \in_r \mathbb{Z}_p$ and computes $pk_U = g^u$. The key pair is $(g^u, u)$.

(WWithdrawal Protocol.) $U$ randomly generates two random numbers $s, t \in_r \mathbb{Z}_p$ and obtains a signature $\sigma_W \leftarrow \text{Sig}.\text{Sign}_{\text{Sig}.sk}(s, t, u)$ from $B$ using Issue of $\text{Sig}$ so that $B$ learns nothing on $s, t, u$. Of course, $U$ is required to prove to $B$ that $u$ is indeed his secret key corresponds to the public key $pk_U$. $U$ parses the electronic coin, or the wallet $W$ of 1 coin, as $W = (\sigma_W, s, t, u)$.

(S Spend Protocol.) $U$ with $W = (\sigma_W, s, t, u)$ engages with merchant $M$ (with public key $pk_M$). They first agree on a random challenge $R = info || nonce || pk_M$ where $nonce$ is a random number issued by $M$ while $info$ is the transaction information. $U$ computes $h = g^t, U = h^u, T = h^s$ and the following $\text{SPK}_{32}$ (denoted as $\pi_S$). The triple $(T, U, h)$ is called the serial number of the coin.

$$\text{SPK}_{32}\{\left( \sigma_W, s, t, u \right):$$

$$\text{valid} \leftarrow \text{Sig}.\text{Verify}_{\text{Sig}.pk}(\sigma_W, s, t, u) \land$$

$$h = g^t \land$$

$$U = h^u \land$$

$$T = h^s \} (R)$$

Finally, $U$ computes $c = H(R || h || U || T || \pi_S)$ and $z = s - cu$. $U$ sends $(\pi_S, h, U, T, z)$ to $M$. $M$ computes $c = H(R || h || U || T || \pi_S)$, accepts the payment if $\pi_S$ is a valid $\text{SPK}$ and $T = U^ch^z$. $M$ stores the coin as $(\pi_S, h, U, T, c, z, info, nonce)$.

(D Deposit Protocol.) $M$ submits $(\pi_S, h, U, T, c, z, info, nonce)$ to bank $B$ for a deposit of 1 coin. $B$ checks if $nonce$ is fresh and $M$ is the owner of public key $pk_M$. Next $B$ verifies $\pi_S$ and checks if $T = U^ch^z$. If the checks pass, $B$ credits $M$. Finally, $B$ checks if the serial number $(T, U, h)$ is already inside its database.
If not, B stores \((T, U, h, c, z)\). Otherwise, B invokes \texttt{Revoke} with the input of the two deposit requests that involve the same serial number.

\texttt{(Revoke).} Assume \((\pi_S, h, U, T, c, z, \text{info}, \text{nonce})\) and \((\pi'_S, h, U, T, c', z', \text{info}', \text{nonce}')\) are two valid transcripts. It implies \(T = U^c h^z\) and \(T = U^{c'} h^{z'}\). B computes \(u = \frac{z' - z}{c' - c}\) and sets \(pk_u = g^u\) of the double-spender. The two transcript serves as a proof-of-correctness for the bank.

\texttt{(VerifyGuilt.)} Every one can check if B is honestly running \texttt{Revoke} as it does not require any secret of B.

5.4.2 The Original Version and Its Flaw

We first describe the original compact e-cash from bounded accumulator that appeared in [AWSM07]. However, there is a flaw that leads to an attack in anonymity. Specially, spending of the same wallet can be linked. Nonetheless, we outline below the original scheme, followed by the attack. Finally, we propose a fix followed by security proof.

The idea of the compact e-cash from bounded accumulator pretty-much falls along the line of the short e-cash described previously. Recall that the bank signs a single random number \(s\) in short e-cash so that the user could use this \(s\) as the randomness for conducting the zero-knowledge proof-of-knowledge of \(h^u\) to base \(h\) once. Now we wish to modify the scheme so that the bank signs \(K\) random numbers in one execution. That is where bounded accumulator is useful. User prepares \(K\) random numbers, accumulates them into the bounded accumulator and sends the accumulator value to the bank. The bank does not need to do any inspections because at most \(K\) random values can be accumulated into it due to the nature of the bounded accumulator\(^8\). We describe the scheme in greater detail below.

The Construction.

\texttt{(BGen.)} Let \(\text{Sig} = (\text{KeyGen}, \text{Sign}, \text{Verify})\) be a \texttt{CL Sig}. Let \(\mathbb{G} = \langle g \rangle\) be a cyclic group of order \(p\). Let \(g_1, g_2, g_3 \in_R \mathbb{G}\) be random generators of \(\mathbb{G}\). The bank B invokes \(\text{Sig.KeyGen}\) to obtain \((\text{Sig.pk}, \text{Sig.sk})\). B also chooses a bounded accumulator

\(^8\)Of course, use could accumulate less than \(K\) random numbers. However, it is against the interest of the user since the bank will deduct his account for \(K\) dollars anyway.
5.4. Compact E-Cash from Bounded Accumulator

\[ (g \circ f : \mathcal{U}_t \rightarrow \mathcal{U}_g, u, \Omega, K) \]

such that the bound of this accumulator is \( K \). The primitives are chosen such that the message space of \( \text{Sig} \) includes \((\mathcal{U}_g \times \mathbb{Z}_p)\) and that \( \mathbb{Z}_p \subset \mathcal{U}_t \). The public key \( \text{bpk} \) is \((G, p, g, g_1, g_2, g_3, \text{Sig}.pk, g \circ f, u, \Omega, K)\) and the secret key \( \text{bsk} \) is \((\text{Sig}.sk)\).

(UGen.) We employ the common discrete logarithm type key pairs. Specifically, user \( U \) randomly generates \( u \in \mathbb{Z}_p \) and computes \( pk_U = g^u \). The key pair is \((g^u, u)\).

(Withdrawal Protocol.) User \( U \) (with input \((pk_U, u)\)) first generates a set of \( K \) random numbers \( \{s_i\}_{i=1}^K \), computes \( V = g \circ f(u, \{s_i\}_{i=1}^K) \), which is the accumulation of the \( K \) random numbers. \( U \) obtains a signature \( \sigma_W \) from \( B \) on values \((V, u)\) using \text{Issue} so that \( B \) learns nothing about \((V, u)\). As usual, \( U \) is required to prove to \( B \) that \( u \) is the secret key of \( pk_U \). \( U \) computes the set of \( K \) witnesses \( \{w_i\}_{i=1}^K \) such that \( \Omega(w_i, s_i, V) = 1 \) for all \( i \). \( U \) parses his wallet \( W \) as \((\sigma_W, \{s_i\}, \{w_i\}, u)\).

(Spend Protocol.) \( U \) with \( W = (\sigma_W, \{s_i\}, \{w_i\}, u) \) engages with merchant \( M \) (with public key \( pk_M \)). They first agree on a random challenge \( R = \text{info}||\text{nonce}||pk_M \) where \text{nonce} is a random number issued by \( M \) while \text{info} \( I \) is the transaction information. \( U \) randomly chooses one of the \( s_i \in \mathbb{Z}_p \) \( \{w_i\} \) (and the corresponding \( w_i \in \{w_i\} \)), computes \( U = g_1^w g_1^{r_1}, T = g_2^{s_i} g_2^{r_2} \) for some randomly generated \( r_1, r_2 \in \mathbb{Z}_p \), \( S = g_3^{s_i} \), and the following \( \text{SPK}_{33} \) (denoted as \( \pi_S \)). \( S \) is called the serial number of the coin.

\[
\text{SPK}_{33}\{ \left( \sigma_W, V, s_i, w_i, u, r_1, r_2 \right) : \\
\text{valid} \leftarrow \text{Sig}.\text{Verify}_{\text{Sig}.pk}(\sigma_W, V, u) \land \\
1 = \Omega(w_i, s_i, V) \land \\
T = g_1^{s_i} g_2^{r_2} \land \\
U = g_1^w g_2^{r_1} \land \\
S = g_3^{s_i} \right) (R) \}
\]

Finally, \( U \) computes \( c = H(R||S||T||U||\pi_S) \) and \( z = s_i - cu, z_r = r_1 - cr_2 \). \( U \) sends \((\pi_S, S, T, U, z, z_r)\) to \( M \). \( M \) computes \( c = H(R||S||T||U||\pi_S) \), accepts

---

\(^9\)Recall that \( g \circ f \) is the accumulating function, \( u \) is the initial element and \( \Omega \) is the membership relation.

\(^{10}\)In fact, the original scheme employed a weaker primitive in which \( B \) knows \( V \) but not anything about \( u \).
the payment if $\pi_S$ is a valid SPK and $T = U^c g_1^z g_2^{z'}$. $M$ stores the coin as $(\pi_S, S, T, U, c, z, z_r, info, nonce)$. $U$ removes $s_i, w_i$ from sets $\{s_i\}$ and $\{w_i\}$ in his wallet $W$.

(Deposit Protocol.) $M$ submits $(\pi_S, S, T, U, c, z, z_r, info, nonce)$ to bank $B$ for a deposit of 1 coin. $B$ checks if $nonce$ is fresh and $M$ is the owner of public key $pk_M$. Next $B$ verifies $\pi_S$ and checks if $T = U^c g_1^z g_2^{z'}$. If the checks pass, $B$ credits $M$. Finally, $B$ checks if the serial number $S$ is already inside its database. If not, $B$ stores $(S, c, z)$. Otherwise, $B$ invokes Revoke with the input of the two deposit requests that involve the same serial number.

(Revoke.) Assume $(\pi_S, S, T, U, c, z, z_r, info, nonce)$ and $(\pi'_S, S', T', U', c', z', z'_r, info', nonce')$ are two valid transcripts. $B$ computes $u = \frac{z - z'}{c - c'}$ and sets $pk_u = g^u$ of the double-spender. The two transcripts serve as a proof-of-correctness for the bank.

(VerifyGuilt.) Every one can check if $B$ is honestly running Revoke as it does not require any secret of $B$.

The Flaw.

Recall that the bank issued a signature on $K$ random numbers $\{s_1, \ldots, s_K\}$, together with secret key $u$ for the user. Each spending reveals certain information, namely, serial number $S_i = g_3^{s_i}$ and $z_i = s_i - c_i u \mod p$. In the original paper, we falsely claimed that since each $s_i$ is random and under the DDH assumption, $S_i$ leaks no useful information. In fact, $S_i$ does leak information which is sufficient for an adversary to link the spending from the same wallet.

Specifically, consider two spendings $(S, c, z)$ and $(S', c', z')$. If they are from the same wallet, we have $S = g_3^s, z = s - cu$ and $S' = g_3^{s'}$ and $z' = s' - c'u$. An adversary can easily derive the relation $c's + cz' = cs' + c'z$ if they are from the same wallet. Consequently, one could test if $S' g_3^{cz'} = S g_3^{cz}$ and confirm that they are from the same wallet (and consequently, the same user) if the equality holds.

5.4.3 The Revised Version

The fix is somehow simple. We just need to change the serial number $S$ from $g^s$ to $g_3^{s'}$ to prevent the above attack. The anonymity can then be proven under the
y-DDHI assumption. Below we present the revised version as an instantiation using ESS+ signature and the bounded accumulator discussed in Chapter 3. We then provide a security proof for our instantiation.

The Actual Construction.

Common Parameter Let λ be a security parameter. Let \( G_1, G_2, G_T \) be cyclic groups of prime order \( p \) such that \( p \) is of \( \lambda \)-bit. Let \( \hat{e} : G_1 \times G_2 \rightarrow G_T \) be a bilinear map. We assume the SXDH assumption holds, that is, DDH assumption holds in both \( G_1 \) and \( G_2 \).

(BGen.) Bank \( B \) randomly generates \( g, g_1 \in_R G_2, h, h_1, h_2, g, g_0, g_1, g_2, g_3, g_4 \in_R G_1 \). \( B \) randomly generates \( X \in G_1, \alpha, \mu \in_R \mathbb{Z}_p \), computes \( Z = g^\mu \in G_2, Z = \hat{e}(X, g) \in G_T, v = g^\alpha \in G_2, v_1 = g^\alpha \in G_1 \), \( \ldots, v_K = g^{\alpha K} \in G_1 \). \( B \) also chooses two more random generators \( g_1, g_2 \in_R G_1 \) and a cryptographic secure hash function \( H \). The public key \( bpk \) is \( (G_1, G_2, G_T, p, \hat{e}, H, h, h_1, h_2, g, g_0, g_1, g_2, g_3, g_4, g, g_1, Z, v, v_1, \ldots, v_K, Z, K, g_1, g_2) \) and the secret key \( bsk \) is \( (X, \mu) \). Note that \( \alpha \) is no longer needed and can be safely deleted. The presence of \( \alpha \) does not help breaking anonymity nor exculpability while it is useful in breaching balance. Thus, it is against the bank’s interest to keep \( \alpha \).

(UGen.) User \( U \) randomly chooses his secret key \( u \in_R \mathbb{Z}_p \) and computes his public key as \( pk_u = h^u \in G_2 \).

Withdrawal Protocol. \( U \) randomly generates \( s_1, \ldots, s_K, x, y' \in_R \mathbb{Z}_p \), computes \( V = g^{\Pi_{i=1}^K (\alpha + s_i)}, C_M = VH_1^x, C_m = h_2^x g_0^y g_1^u \). The \( U \) sends \( pk_U, C_M, C_m \) to bank \( B \), along with the following proof,

\[
\text{PK}_{34}\{(x, y', u) : C_m = h_2^x g_0^y g_1^u \wedge pk_U = h^u \}
\]

\( B \) verifies \( \text{PK}_{34} \), randomly selects \( y'', e \in_R \mathbb{Z}_p \), computes \( \varsigma_1 = (X(C_M))^e \in G_1, \varsigma_2 = (gg_0^y C_m)^{\frac{1}{e + e}} \in G_1, \varsigma_3 = g^e \in G_2 \) and returns \((\varsigma_1, \varsigma_2, \varsigma_3, y'') \) to \( U \). \( U \) computes \( w_i = g^{\Pi_{j=1,j\neq i}^K (\alpha + s_i)} \) for \( i = 1 \) to \( K \), \( y = y' + y'' \) mod \( p \) and stores \((\varsigma_1, \varsigma_2, \varsigma_3, x, y, \{s_i\}, \{w_i\}, u) \) as his wallet \( W \).

(Spend Protocol.) \( U \) with \( W = (\varsigma_1, \varsigma_2, \varsigma_3, x, y, \{s_i\}, \{w_i\}, u) \) engages with merchant \( M \) (with public key \( pk_M \)). They first agree on a random challenge

\[\text{We assume that these common parameters are generated by some trusted parties.}\]
\[ R = info \| nonce \| pk_M \] where \( nonce \) is a random number issued by \( M \) while \( info \) is the transaction information. \( U \) randomly chooses one of the \( s_i \in R \{ s_i \} \) (and the corresponding \( w_i \in \{ w_i \} \)), computes \( U = g_2^u g_3^{r_1}, T = g_2^s g_3^{r_2} \) for some randomly generated \( r_1, r_2 \in R Z_p \), \( S = g_4^{1/s_i} \), and the following \( SPK_{35} \).

\[
\begin{align*}
SPK_{35} \left\{ \left( \varsigma_1, \varsigma_2, \varsigma_3, x, y, V, s_i, w_i, u, r_1, r_2 \right) \right. & : \\
\hat{e}(\varsigma_1, g) &= Z \hat{e}(V h_1^x, \varsigma_3) \quad \land \\
\hat{e}(\varsigma_2, \varsigma_3 Z) &= \hat{e}(g h_2^x g_0^y g_1^u, g) \quad \land \\
\hat{e}(w_i, g^{s_i} v) &= \hat{e}(g, g) \quad \land \\
T &= g_2^s g_3^{r_1} \quad \land \\
U &= g_2^u g_3^{r_2} \quad \land \\
S &= g_4^{1/s_i} \right\} (R)
\]

Finally, \( U \) computes \( c = H(R||S||T||U||\pi_S) \) and \( z = s_i - cu \), \( z_r = r_1 - cr_2 \). \( U \) sends \((\pi_S, S, T, U, z, z_r)\) to \( M \). \( M \) computes \( c = H(R||S||T||U||\pi_S) \), accepts the payment if \( \pi_S \) is a valid \( SPK \) and \( T = U^c g_2^s g_3^{r_r} \). \( M \) stores the coin as \((\pi_S, S, T, U, c, z, z_r, info, nonce)\). \( U \) removes \( s_i, w_i \) from sets \( \{ s_i \} \) and \( \{ w_i \} \) in his wallet \( \mathcal{W} \).

\( SPK_{35} \) requires instantiation. \( U \) randomly generates \( r_3, r_4, r_5, r_6, r_7, r_8, r_9 \in R Z_p \), computes \( \mathfrak{A}_1 = \varsigma_1 g_2^{r_3}, \mathfrak{A}_2 = V g_2^{r_4}, \mathfrak{A}_3 = \varsigma_2 g_2^{r_5}, \mathfrak{A}_4 = \varsigma_3 g_1^{r_6}, \mathfrak{A}_5 = w_i g_2^{r_7}, \)
\[ \mathcal{A}_6 = g_1^{r_6} g_2^{r_8}, \quad \mathcal{A}_7 = g_1^{r_7} g_2^{r_9} \]

and computes the following SPK_{36}.

\[
\text{SPK}_{36} \left\{ \left( x, y, s_i, u, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, \beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8, \right) \right. \\
\left. \mathcal{A}_6 = g_1^{r_6} g_2^{r_8}, \quad \mathcal{A}_7 = g_1^{r_7} g_2^{r_9} \right\}
\]

( Deposit Protocol.) \( M \) submits \((\pi_S, S, T, U, c, z, z_r, info, nonce)\) to bank \( B \) for a deposit of 1 coin. \( B \) checks if \( nonce \) is fresh and \( M \) is the owner of public key \( pk_M \).

Next \( B \) verifies \( \pi_S \) and checks if \( T = U^c g_2^z g_3^{z_r} \). If the checks pass, \( B \) credits \( M \). Finally, \( B \) checks if the serial number \( S \) is already inside its database. If not, \( B \) stores \((S, c, z)\). Otherwise, \( B \) invokes \texttt{Revoke} with the input of the two deposit requests that involve the same serial number.

( Revoke.) Assume \((\pi_S, S, T, U, c, z, z_r, info, nonce)\) and \((\pi_S', S, T', U', c', z', z_r', \text{info}', nonce')\) are two valid transcripts. \( B \) computes \( u = g_3^{z-r'z} \) and sets \( pk_u = g^u \) of the double-spender. The two transcripts serve as a proof-of-correctness for the bank.

(VerifyGuilt.) Every one can check if \( B \) is honestly running \texttt{Revoke} as it does not require any secret of \( B \).

### 5.4.4 Security Analysis of Our Revised Version

We give a proof sketch to show why the revised compact e-cash from accumulator is secure.
Proof: (Sketch.)

Balance.

Let $A$ be an adversary that wins GAME Balance with non-negligible advantage. We outline why the success probability of $A$ is negligible due to the unforgeability of $ESS+$ and the security of the bounded accumulator by constructing a simulator $S$ acting as challenger $C$. $S$ is given a signing oracle for $ESS+$ and its goal is to forge a $ESS+$ signature or output $K + 1$ value-witness pair for the pairing-based bounded accumulator with bound $K$.

Oracle Simulations.

$O_U$. If a public key is supplied, the oracle stores it in $U_A$. Otherwise, randomly generate $u$ and set $pk_U$ to $g^u$. Add $pk_U$ to $U_H$ and return $pk_U$ to adversary.

$O_{W,H}$. If $pk_U \in U_A$, simulate the protocol by invoking the signing oracle of $ESS+$.

Note that it involves rewinding $PK_{34}$. $S$ obtains the values $(x, y', u)$. Define the value $V$ as $C_M/h_x^z$. During the process, $S$ gets the whole wallet $W = (\varsigma_1, \varsigma_2, \varsigma_3, x, y, V, u)$ and it stores it in a set $W$.

If $pk_U \in U_H$, randomly generate $\{s_i\}$ and compute the corresponding $V$ as the accumulator value of set $\{s_i\}$. $S$ also computes the corresponding witness set $\{w_i\}$. $S$ stores the wallet as $W = (\cdot, V, \{s_i\}, \{w_i\}, u)$. The first term refers to the $ESS+$ signature which is unknown to the simulator.

$O_S$. If it involves an honest user (regardless of whether the merchant is adversary-controlled or not), $S$ simply uses the knowledge simulator to finish the view of the protocol. More specially, for user $(h^u, u)$ with wallet, simulates the protocol as if he is in possession of a wallet $W = (\sigma_W, V, \{s_i\}, \{w_i\}, u)$. This requires the knowledge simulator of $\sigma_W$ and its presence is due to the Prove privacy of $ESS+$. Remove the used $s_i$ and $w_i$ from their corresponding sets.

If it involves an adversary-controlled user and an honest merchant, $S$ rewinds $SPK_{35}$ to obtain the underlying $(V, s_i, w_i, u)$ used.

Outputs. Finally, $A$ deposits $\text{bal}_A + 1$ coins to $S$. $A$ wins the game either by (1) all the $\text{bal}_A + 1$ serial numbers in Deposit Protocol are unique or (2) some of the serial numbers are duplicated but Revoke on the corresponding Deposit Protocol transcripts does not output any $pk \in U_A$. Now we analyse these two cases separately.
5.4. Compact E-Cash from Bounded Accumulator

Case (1): Due to the security of ESS+, each \( O_{WH} \) query only gives \( A \) one single \( V \) to work with. Due to the setting of the game, the total “valid” serial numbers obtained by \( A \) from \( S \), minus the serial numbers that has been presented to \( S \) during \( O_S \) query is \( \text{bal}_A \).

\( A \) can only win in Case (1) by convincing \( S \) to accept a serial number \( S \) that is not one of these \( \text{bal}_A \) serial numbers or for any \( V \) that it obtains an ESS+ signature, he can generate \( K + 1 \) value-witness pairs. If this is the case, \( S \) rewinds all \( \text{SPK}_{35} \) and obtains \( K + 1 \) value-witness pairs, thus breaking the security of the bounded accumulator. Otherwise, \( A \) must have conducted a false proof as part of the signature of knowledge, \( \text{SPK}_{35} \), such that one of the following is fake:

1. Possession of a ESS+ signature on block of messages \( (V, u) \)
2. Possession of a pair \( (w, s) \) such that \( \Omega(w, s, V) = 1 \)
3. \( S = g_4^{1/s} \).

Fake proof of possession of ESS+ happens with negligible probability. Fake proof of \( S \) is well-formed happens with negligible probability under the discrete logarithm assumption. Thus, \( A \) can only win if he could generate a new ESS+ signature and \( S \) can use it to break the unforgeability of ESS+.

Case (2): We have shown in case (1) that \( A \) cannot convince \( S \) to accept an invalid serial number with non-negligible probability. We now suppose that duplicated \( S \) are accepted.

It remains to show that the associated \( (T, U, z, c) \) is bounded by specification except with negligible probability so that the correctness of Revoke implies the recovering of a \( pk \in U_A \). Due to the soundness of the proof of knowledge protocol \( \text{SPK}_{35} \), \( T = g_2^s g_3^{r_1} \) and \( U = g_2^u g_3^{r_2} \) and since \( T = U^c g_2^z g_3^{z_r} \), we have \( z = s - cu \) unless the adversary has solved the discrete logarithm of \( g_2 \) to base \( g_3 \). Thus, \( A \)’s success probability is negligible.

Anonymity. We construct a simulator \( S \) which solves the \( q \)-DDHI problem in \( G_1 \) if the adversary can win in GAME Anonymity. \( S \)’s goal is on input \( g, g^\beta, g^{\beta^2}, g^{\beta^q}, Y \) in \( G_1 \), to decide if \( Y = g^\beta \). \( S \) randomly generates \( \gamma_1, \gamma_2 \in R \mathbb{Z}_p \), denotes \( u = \gamma_1 \beta + \gamma_2 \) and computes \( g, g^u, \ldots, g^{u^q} \) for \( i = 1 \) to \( q^{12} \). Then, for \( i = 1 \) to \( q \), \( S \) random generates \( c_i, z_i \in R \mathbb{Z}_p \) and \( \gamma \in R \mathbb{Z}_p \). Denote \( F(u) \) as the \( q \) degree polynomial \( \prod_{i=1}^{q} (c_i u + z_i) \).

\(^{12}\text{This is straightforward using Binomial Theorem.}\)
Compute $g_4 = g^{F(u)}$ and set $h = g^\gamma$. Denote the set $W_S = \{c_i u + z_i\}_{i=1}^q$. $S$ then generates $(bpk, bsk)$ honestly with the exception of $h$ and $g_4$. $(bpk, bsk)$ are given to adversary $A$. To prove that the secret information of the bounded accumulator does not help, $A$ is also given $\alpha$.

$S$ chooses one honest user $U^*$ and handles all queries related to $U^*$ differently. The public key of $U^*$ is $h^u = g^{\gamma u}$ and the corresponding secret key is $u$ and is unknown to $S$. Withdrawal query of $U$ is treated as if it is obtaining a signature on $V, u$ such that $V$ is the accumulation of a set, indexed by counter $j$, $W_j \subseteq W_S$ with $|W_j| = K$. After that set $W_S$ as $W_S \setminus W_j$. Note that the total number of withdrawal of $U^*$ is thus $\lfloor q/K \rfloor$. Even though $S$ does not know $V$ or $u$, it can be handled with the help of the knowledge simulator. Spend Protocol of $U$ is handled as follow. Assume the value of $s$ to be used is $c_i u + z_i \in W_S$, $S$ randomly generates $z_r \in R \mathbb{Z}_p, U \in R \mathbb{G}_1$ and computes $T = U^{c_i g_2^z} g_3^{z_r}$. Set $S$ as $g_4^{1/s} = g^{F(u)/(c_i u + z_i)}$.

Finally, back patch $c_i$ to be the hash output.

All other queries are handled honestly.

If a wallet of $U$ is chosen to be the users in the challenge phase, the challenge spending is set in such a way that the underlying $s$ being used is randomly chosen. Firstly, randomly generate $z_r \in R \mathbb{Z}_p, U \in R \mathbb{G}_1$ and computes $T = U^{c_i g_2^z} g_3^{z_r}$. Now the $s$ being used is $cu + z$, which is unknown to the simulator. The simulator has to set the corresponding hash output to $c$. Specifically, the value $S = g_4^{1/s}$ will be $g^{F(u)/(cu+z)}$. There exists a $q-1$ degree polynomial $Q(u)$ such that $F(u) = Q(u)(cu+z) + R$ such that $R$ is a constant. Now $S = g_4^{1/(cu+z)}$ is $g^{Q(u) + R/(cu+z)}$ which is $g^{Q(u)}(g^R)^{\frac{1}{q}}$. $S$ sets $S$ as $g^{Q(u)Y^R}$ and it is correctly formed if and only if $Y = g^\frac{1}{q}$. If $Y$ is not equal to $g^\frac{1}{q}$, the transcript does not contain information about any user.

Using the answer from $A$, $S$ knows if $Y$ is a $q$-DDHI tuple or not.

**Exculpability.** In Revoke, both transcripts contain the proof of correctness of $U$, which involves proving knowledge of the user secret $u$. To slander an honest user, adversary without knowledge of $u$ has to fake SPK$_{36}$ and this happens with negligible probability under the discrete logarithm assumption.

\[\square\]
5.5 Divisible E-Cash

The first practical divisible e-cash was due to Okamoto [Oka95] and improved by Chan et al. [CFT98]. However, until very recently with the proposal from [CG07] (which we shall refer to as CG07 hereafter), none of the divisible e-cash in the literature is truly anonymous. In this section, we shall investigate the practicality of the CG07, present our construction from ESS+ signature and bounded accumulator, followed by security analysis.

5.5.1 On Practicality of the CG07 Scheme

We analyse CG07 in greater detail. To allow efficient withdrawal of $2^L$ coins, they require a series of $L + 2$ cyclic groups ($\mathbb{G} = \langle g \rangle, \mathbb{G}_1 = \langle g_1 \rangle, \ldots, \mathbb{G}_{L+1} = \langle g_{L+1} \rangle$) such that $\mathbb{G}_i \subset \mathbb{Z}_{|\mathbb{G}_i|}^*$ for $i = 1$ to $L + 1$ and $\mathbb{G} \subset \mathbb{Z}_{|\mathbb{G}_1|}^{13}$. and the decisional discrete logarithm assumption (DDH) holds in all $\mathbb{G}_i$. However, whether such series of groups exist, for moderate $L$ (say, $L = 10$), is unknown. The authors suggest using the same setting of groups as in [NS00] which proposes to set $|\mathbb{G}_i|$, for $i = 1$ to $L + 1$, to be of prime order and assume $|\mathbb{G}_{i+1}| = 2|\mathbb{G}_i| + 1$ for $i = 1$ to $L + 1$. This implies finding a series of primes $p_1, \ldots, p_{L+1}$ such that $p_{i+1} = 2p_i + 1$. Again, whether such series of primes exist, for moderate $L$, is unknown and it is also unknown how these series of primes can be efficiently generated. The authors in [NS00] propose using a brute-force approach. That is, randomly generate an odd number $n$ (equals to order of group $\mathbb{G}$) and test if $p_1 := 2n + 1$ is a prime. If yes, compute and test if $p_2 := 2p_1 + 1$ is prime. Continue until $p_{L+1} := 2p_L + 1$ is also a prime. A well-known result, the prime number theorem, states that the number of primes not exceeding $m$ is approximately $\frac{m}{\ln(m)}$. Thus, the probability that a $k$-bit odd number is a prime is about $\frac{2}{k \ln 2}$. For a randomly generated $k$-bit odd number $n$, the probability that $(p_1, \ldots, p_{L+1})$ are primes such that $p_{i+1} := 2p_i + 1$ and $p_1 := 2n + 1$ is approximately $\frac{k^{10L}}{(k+L+1)!(\ln(2)^L)^{k+L}}$. Taking $k = 170$ and $L = 10$, the probability of obtaining such series of prime numbers on a given $k$-bit odd number $n$ is about $2^{-66}$. In fact, in [NS00], $n$ is taken to be an RSA-modulus (which is normally of 1024-bit), and the corresponding probability is $2^{-94}$. Therefore, it is questionable whether the systems in [NS00] or CG07 are in fact implementable.

\[13\text{In their paper, it was written as } \mathbb{G}_1 \subset \mathbb{Z}_{|\mathbb{G}_1|}^*. \text{ However, according to their construction(as it involves computation of } g_1^s \text{ for some } s \text{ in } \mathbb{Z}_{|\mathbb{G}_1|}, \mathbb{G} \subset \mathbb{Z}_{|\mathbb{G}_1|} \text{ should be the case.}\]
5.5. Divisible E-Cash

The spending operation in CG07 is also quite inefficient. As mentioned in the same paper, the authors regard spending a single coin as quite an expensive operation. It is due to the need of $L$ “1-out-of-2 zero-knowledge proof-of-knowledge of double discrete logarithm”. For a cheating probability of $2^{-t}$, with $t$ being a security parameter, a single zero-knowledge proof-of-knowledge of double discrete logarithms requires $t$ exponentiations. For a cheating probability of $2^{-80}$ and a moderate $L$ (say 10), spending a single coin requires $2 \times 80 \times 10 = 1600$ exponentiations. Moreover, it requires a communication cost of more than 1600 group elements (each group element shall be of size greater than 1 kB). Detailed analysis in the cost of each protocol will be outlined later. Nonetheless, while CG07 provides an affirmative answer to whether divisible e-cash can be truly anonymous, it is fair to say that constructing a practical divisible e-cash which is truly anonymous is not that easy.

5.5.2 Overview of Our Construction

The construction of our divisible e-cash is derived from the classical binary tree approach in combination with the use of a bounded accumulator. We make use of the bounded accumulator to make a trade-off between computational cost during the withdrawal protocol and the spend protocol. The cost (computational and bandwidth) of our withdrawal protocol and spend protocol is $O(L)$ and $O(1)$, respectively, while the corresponding figures for [CG07] are $O(1)$ and $O(Lt)$. Since the spending protocol is executed much more frequently than the withdrawal protocol, our system is much more desirable in practice.

The trade-off is achieved with the use of accumulators. During the withdrawal protocol, the user computes the accumulation of a binary tree into $L+1$ accumulator values ($V_1, \ldots, V_{L+1}$) and obtains $L+1$ signatures. In the spending protocol, if a node of level $\ell$ is to be used, the user only needs to compute a ZKPoK such that the node he is about to use is inside the accumulator $V_{\ell}$. In this way, our spend protocol achieves a complexity of $O(1)$.

An obvious way to ensure that the user is honestly accumulating node values that form a binary tree, while maintaining anonymity, is to require the user to produce ZKPoK such that the set of accumulator values ($V_1, \ldots, V_{L+1}$) are correctly formed. This approach, however, is inefficient. Another approach is to apply the cut-and-choose method in a straightforward manner. Specifically, the user prepares $k$ sets of value and submits them all to the bank who requires the user to reveal $k - 1$ of
them at random. The bank checks if these $k - 1$ sets of value are honestly generated and signs the remaining one if the check is successful. To ensure that a user cannot cheat, $k$ has to be large. Thus, this approach is inefficient as well.

Luckily, bounded accumulator gives us the possibility of a third solution, which is a modification of the cut-and-choose method. Our approach is statistical, that is, a cheating user might spend more than what he withdraws for a particular withdrawal protocol but in a long run, the bank is guaranteed that users cannot spend more than they withdraw on average. The idea is derived from the following fact: since the accumulator we use is bounded, the user can only accumulate a predefined number of values regardless of whether they are cheating or not. Naturally, there is an upper bound for which a cheating user might gain. In our scheme, the cheating user can get at most a monetary value of $L2^L$, compared with a value of $2^L$ for an honest user. If the bank inspects the withdrawal protocol every two withdrawal requests and imposes a fine of monetary value $2L2^L$ if a user is found cheating, the bank is guaranteed it will not lose money on average. Later, we will formally define the security model for e-cash schemes that employs this kind of statistical approach. Note the gain of a cheater cannot be large; since if the gain is large, a cheater might not be able to pay the fine if he is caught. Secondly, a large gain gives extra incentive for people to cheat.

5.5.3 High Level Description of our Construction

Since the actual construction is quite involved, we provide a high level description together with an illustrating example of a simple scenario of our construction.

(BGen.) Let $\text{Sig} = (\text{KeyGen, Sign, Verify})$ be a CL $\text{Sig}$. Let $G = \langle g \rangle$ be a cyclic group of order $p$. Let $h, g_1, g_2, g_3 \in_R G$ be random generators of $G$. The bank $B$ invokes $\text{Sig.KeyGen}$ to obtain $(\text{Sig.pk, Sig.sk})$. $B$ also chooses an $L + 1$ bounded accumulator $(g \circ f_i : \mathcal{U}_f \rightarrow \mathcal{U}_g, u_i, \Omega_i, 2^i)$, for $i = 0$ to $L$, such that the bounded of the $i$ accumulator is $2^i$. The primitives are chosen such that the message space of $\text{Sig}$ includes $(\mathcal{U}_g \times \mathbb{Z}_p)$ and that $\mathbb{Z}_p \subset \mathcal{U}_f$. Let $LF, RF$ be two secure cryptographic hash functions. Let $H$ be another secure cryptographic hash function. The public key $bpk$ is $(G, p, g, g_1, g_2, g_3, h, \text{Sig.pk, } (g \circ f_i, u_i, \Omega_i, 2^i)_{i=0}^L, L, H, LF, RF)$ and the secret key $bsk$ is $(\text{Sig.sk})$.

(UGen.) We employ the common discrete logarithm type key pairs. Specifically,
user \( U \) randomly generates \( h \in_R \mathbb{Z}_p \) and computes \( pk_U = h^u \). The key pair is \((h^u, u)\).

(Withdrawal Protocol.) User \( U \) (with input \((pk_U, u)\)) first generates a random number \( x \in_R \mathbb{Z}_p \), the wallet secret. \( U \) then computes a binary tree of \( L + 1 \) levels as follows. The root node \( N_{0,0} \) is assigned the node key value \( k_{0,0} = x \).

For all nodes \( N_{i,j} \), the left children, \( N_{i+1,2j} \), is assigned a node key value \( k_{i+1,2j} := LF(g^{k_{i,j}}) \). Similarly, the right children, \( N_{i+1,2j+1} \), is assigned a node key value \( k_{i+1,2j+1} := RF(g^{k_{i,j}}) \). Let \( \text{Tree}_x \) be the resulting binary tree computed by \( U \). Fig 5.2 illustrates a construction of a binary tree with \( L = 3 \).

\[
\begin{align*}
N_{0,0} &= x \\
N_{1,0} &= LF(g^{k_{0,0}}) \\
N_{1,1} &= RF(g^{k_{0,0}}) \\
N_{2,0} &= LF(g^{k_{1,0}}) \\
N_{2,1} &= RF(g^{k_{1,0}}) \\
N_{2,2} &= LF(g^{k_{1,1}}) \\
N_{2,3} &= RF(g^{k_{1,1}}) \\
N_{3,0} &= LF(g^{k_{2,0}}) = RF(g^{k_{2,1}}) = LF(g^{k_{2,2}}) = RF(g^{k_{2,3}}) = LF(g^{k_{3,0}}) = RF(g^{k_{3,1}})
\end{align*}
\]

Figure 5.2: Construction of A Binary Tree (L=3)

then computes, for \( i = 0 \) to \( L \), \( V_i = g \circ f_i(u, \{k_{i,j}\}_{j=0}^{2^i-1}) \), which are the accumulation of the node key values of level \( i \) of \( \text{Tree}_x \). \( U \) tries to obtain \( L + 1 \) signatures \( \sigma_i \) from \( B \) on values \((V_i, u)\) using \text{Issue} so that \( B \) learns nothing about \((V_i, u)\). As usual, \( U \) is required to prove to \( B \) that \( u \) is the secret key of \( pk_U \).

With probability \( 1/L \), \( B \) does not issue the signatures. Instead, \( B \) asks the user to reveal \( x \). \( B \) computes from \( x \) the binary tree \( \text{Tree}_x \) and tests if the
5.5. Divisible E-Cash

V_i’s are correctly formed. That is, to check whether V_i is the accumulation of \{k_{i,0}, \ldots, k_{i,2^\ell-1}\}. If yes, B asks U to restart the withdrawal procedure. Otherwise, B imposes a fine \$L^22^L$ to U since U is cheating.

Otherwise, B completes Issue and issues the L + 1 signatures \(\sigma_i\) on \((V_i, u)\). U parses \(\sigma_W\) as \((\sigma_0, \ldots, \sigma_L)\). Then, U parses his wallet \(W\) as \((\sigma_W, Tree_x, u)\). To save space, U could store \(Tree_x\) as \(x\).

(Spend Protocol.) U with \(W = (\sigma_W, Tree_x, u)\) engages with merchant M (with public key \(pk_M\)) first agrees on a random challenge \(R = H(\text{info}||\text{nonce}||pk_M||2^\ell)\) where \(\text{nonce}\) is a random number issued by M, \(\text{info}\) is the transaction information, \(2^\ell\) is the transaction amount.

U first chooses a node from the \(Tree_x\) at level \(L - \ell\) which has not been marked as used. Let \(N_{i,j}\) be the node chosen (that is, \(i = L - \ell\)). Compute serial number \(S = g_2^{k_{i,j}}\) and tracing tag \(T = h^u g_3^{k_{i,j}R}\) such that \(k_{i,j}\) is the node key value of node \(N_{i,j}\). U computes the corresponding witness \(w_{i,j}\) such that the value \(k_{i,j}\) is in the accumulator \(V_i\). Finally U computes the following SPK37, denoted as \(\pi_S\).

\[
\text{SPK}_{37} \{ \left( \sigma_i, V_i, k_{i,j}, w_{i,j}, u \right) : \begin{align*}
\text{valid} & \leftarrow \text{Sig.Verify}_{\text{Sig}_{\text{pk}}} (\sigma_i, V_i, u) \land \\
1 & = \Omega_i(w_{i,j}, k_{i,j}, V_i) \land \\
T & = h^u (g_3^R)^{k_{i,j}} \land \\
S & = g_2^{k_{i,j}} \end{align*} \right\} (R)
\]

M accepts the payment if \(\pi_S\) is a valid SPK and stores the coin as \((\pi_S, S, T, \text{info}, \text{nonce}, 2^\ell)\). If M accepts, U marks \(N_{i,j}\) and all its children, as well as ancestors, from \(Tree_x\) as used node.

Example: Spending 4 dollars. We use the binary tree in Fig 5.2 as an example. U’s wallet includes \(W = (\sigma_0, \sigma_1, \sigma_2, \sigma_3, V_0, V_1, V_2, V_3, Tree_x, u)\) and wishes to pay 4 dollars to merchant M.

Suppose the node \(N_{1,0}\) is being chosen. The corresponding node key \(k_{1,0}\) is \(LF(g^{(x)})\). U computes the serial number \(S = g_2^{LF(g^x)}\) and the security tag \(T = h^u g_3^{LF(g^x)R}\) where \(R\) is a random challenge discussed above. U computes \(w_{1,0}\) which is the witness of the value of \(k_{1,0}\) such that \(k_{1,0}\) is inside the accumulator.
of value $V_1$. $U$ sends $(\pi_S, S, T, 2^2)$ to $M$, where $\pi_S$ is an SPK of the following:

$$\text{SPK}\{ (\sigma_1, V_1, k_{1,0}, w_{1,0}, u) :$$

$$\text{valid } \leftarrow \text{Sig. Verify}_{\text{Sig.pk}}(\sigma_1, V_1, u) \land$$

$$1 = \Omega_1(w_{1,0}, k_{1,0}, V_1) \land$$

$$T = h^u(g_3^{R_1})^{k_{1,0}} \land$$

$$S = g_2^{k_{1,0}} \} \mod (R)$$

$M$ accepts the payment if $\pi_S$ is a valid proof. $U$ marks $N_{0,0}$, $N_{1,0}$, $N_{2,0}$, $N_{2,1}$, $N_{3,0}$, $N_{3,1}$, $N_{3,2}$, $N_{3,3}$ as used nodes, as in Fig 5.3.

---

**Figure 5.3: Spending 4 coins**

(Deposit Protocol.) $M$ submits $(\pi_S, S, T, info, nonce, 2^\ell)$ to bank $B$ for a deposit of $2^\ell$ coins. $B$ checks if $nonce$ is fresh and $M$ is the owner of public key $pk_M$. Next, $B$ verifies $\pi_S$. If the checks pass, $B$ credits $M$. $B$ then tries to detect if the coin with serial number $S$ has been double-spent. Let $S$ be the serial number of a coin of monetary value $2^\ell$. Let $N_{i,j}$ be the corresponding node of the binary

\[14\] The exact position $j$ is unknown.
5.5. Divisible E-Cash

From $S$, the bank computes the $2^\ell$ serial numbers corresponding to the leaves of subtree of node $N_{i,j}$ by repeatedly applying the functions $LF(\cdot)$, $RF(\cdot)$ and $g^z(\cdot)$. If not, it stores the $2^\ell$ entries $(S_i, S_T, R, \pi_S)$ in its database. Suppose there exists another entry in the database $(S'_i, S'_T, T', R', \pi')$, B invokes Revoke.

**Example: Deposit a coin of 4 dollars.** Continuing our example, M submits $(\pi_S, S, T, info, nonce, 2^2)$ to bank B for deposit after getting 4 dollars from U. B verifies $\pi_S$ is a valid proof and nonce is fresh and credits M. From $S$, the bank computes $S_0, \ldots, S_3$ as follows. Compute intermediate value $\tilde{S}_0 = g^{LF(S)}$ and $\tilde{S}_1 = g^{RF(S)}$. Compute $S_0 = g^{LF(\tilde{S}_0)}$, $S_1 = g^{RF(\tilde{S}_0)}$, $S_2 = g^{LF(\tilde{S}_1)}$, $S_3 = g^{LF(\tilde{S}_1)}$. The bank checks if $S_0, S_1, S_2, S_3$ exists in its database of spent-coins. If not, it stores $(S_i, S, T, \pi_S, R)$, for $i = 0$ to 3 in its database.

(Revoke.) Assume the input entries are $(S_i, S, T, R, \pi_S)$ and $(S_i', S', T', R', \pi'_S)$, bank B computes the identity of the double-spender as follows. If $S$ and $S'$ are the same, compute $pk_U := (T'_R)^i \frac{1}{P^i -\pi}$. On the other hand, if $S$ and $S'$ are different, $S$ and $S'$ must be of different monetary value. Without loss of generality, assume the monetary value of coin with serial number $S$ is greater than that of $S'$. The bank can compute the node key $k_{i,j}$ such that $S' = g^{k_{i,j}}$ from $S$ by repeatedly applying the $LF(\cdot)$, $RF(\cdot)$, $g^z(\cdot)$ in suitable order. From $k_{i,j}$, the bank computes $pk_U = \frac{T'_R}{g^{R k_{i,j}}}$ and obtains identity of the double-spender. The two transcripts serve as a proof-of-correctness for the bank.

**Example: Catching Double-Spender.** Continuing our example. Assuming U spends another 2 dollars to merchant M, this time using the node $N_{2,1}$ in an attempt to over-spend his wallet. Assume the resulting transcript is $(S'_i, T', \pi'_S, R', 2^1)$. When M submits this transcript for deposit, the bank will identify it as a double-spent coin since $S'_i = g^{LF(S')}$ will be equal to $S_3$ in the above example. The bank can then compute, from $S$ in the above example, $k_{2,1} = RF(S)$. From $k_{2,1}$, B computes $PK_U = \frac{T'_R}{g^{R k_{2,1}}}$. 
5.5. Divisible E-Cash

113

The actual construction is based on \textit{ESS+} signature and the pairing-based bounded accumulator.

**Common Parameter**\footnote{We assume that these common parameters are generated by some trusted parties.} Let $\lambda$ be a security parameter. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be cyclic groups of prime order $p$ such that $p$ is of $\lambda$-bit. Let $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map. We assume the SXDH assumption holds, that is, DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.

**BGen.** Bank $B$ randomly generates $g, g_1 \in \mathbb{G}_1, h, h_1, h_2, g, g_0, g_1, g_2, g_3 \in \mathbb{G}_2$. $B$ randomly generates $X \in \mathbb{G}_1, \alpha_0, \ldots, \alpha_L, \mu \in \mathbb{Z}_p$, computes $Z = g_1^\mu \in \mathbb{G}_2$, $Z = \hat{e}(X, g) \in \mathbb{G}_T$, $v_i = g_i^\alpha \in \mathbb{G}_2, v_{1,i} = g^\alpha \in \mathbb{G}_1, \ldots, v_{2^i,i} = g_1^{\alpha^k} \in \mathbb{G}_1$ for $i = 0$ to $L$. $B$ also chooses two more random generators $g_1, g_2 \in \mathbb{G}_1$ and three collision-resistant hash functions $H, H_0, H_1$\footnote{$H_0$ and $H_1$ take the role of LF and RF in the high level description.}. The public key $bpk$ is $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, \hat{e}, H, H_0, H_1, h, h_1, h_2, g, g_0, g_1, g_2, g_3, g, g_1, Z, (v_i), (v_{1,i}), (v_{2^i,i}), L, g_1, g_2)$ and the secret key $bsk$ is $(X, \mu)$. Note that $(\alpha_i)_{i=0}^L$ is no longer needed and can be safely deleted. The presence of these $\alpha$'s does not help breaking \textit{anonymity} nor \textit{exculpability} while it is useful in breaching \textit{balance}. Thus, it is against the bank’s interest to keep these $\alpha$’s.

**UGen.** User $U$ randomly chooses his secret key $u \in \mathbb{Z}_p$, and computes his public key as $pk_u = h_u^u \in \mathbb{G}_p$.

**Withdrawal Protocol.** $U$ randomly generates $x \in \mathbb{Z}_p$, computes a binary tree $\text{Tree}_x$ of $L + 1$ level by executing algorithm \textit{ComputeAllNodeKeys}, as shown in Fig 5.4 with input $(x, L)$. For $i = 0$ to $L$, $U$ computes $V_i = g_1^{\prod_{j=0}^{i-1} (\alpha_i + k_{i,j})}$. Note that $V_i$ is the accumulation of node keys of the $i$-th level of $\text{Tree}_x$. Next, $U$ computes, for $i = 0$ to $L$, $C_{M,i} = V_i h_1^{s_i}, C_{m,i} = h_2^{t_i} g_0^u g_1^u$ for some randomly generated $s_i, t_i \in \mathbb{Z}_p$. Then $U$ sends $pk_U, (C_{M,i}, C_{m,i})_{i=0}^L$ to bank $B$, along with the following proof,

\[
\text{PK}_{38} \{ (s_0, t_{0}', \ldots, s_L, t_{L}', u) : \bigwedge_{i=0}^L C_{m,i} = h_2^{s_i} g_0^{t_i} g_1^u \land pk_U = h_u^u \}\]
5.5. Divisible E-Cash

\[
\text{Algorithm ComputeAllNodeKeys}
\]
\[
\begin{array}{ll}
\text{Input: } x, L \\
\text{Output: } k_{0,0}, k_{1,0}, k_{1,1}, \ldots, k_{L,0}, \ldots, k_{L,2^L-1}
\end{array}
\]
\[
k_{0,0} := x
\]
\[
\text{For } i = 1 \text{ to } L
\]
\[
\text{For } j = 0 \text{ to } 2^i - 1
\]
\[
\text{if } j \text{ mod } 2 = 0 \quad \text{Left children of parent node}
\]
\[
k_{i,j} := H_0(g^{k_{i-1,j/2}}) \text{ for left children}
\]
\[
\text{else} \quad \text{Right children of parent node}
\]
\[
k_{i,j} := H_1(g^{k_{i-1,j/2}}) \text{ for right children}
\]

Figure 5.4: Algorithm ComputeAllNodeKeys

With probability $1/L$, $B$ asks the user to reveal ($\{s_i, t'_i\}, x$). $U$ sends $\{s_i, t'_i\}, x$ to $B$ who checks if $V_i$ is computed correctly. If the check fails, impose a fine of $L2^L$ to $U$’s account. Otherwise, ask $U$ to restart the protocol.

Otherwise, $B$ verifies $\text{PK}_{38}$, randomly selects $t''_i, e_i \in R \mathbb{Z}_p$, computes $s_{1,i} = (X(C_M))^{e_i} \in G_1$, $s_{2,i} = (gg_0^{t''_i}C_{m,i})^{\frac{1}{e_i}} \in G_1$, $s_{3,i} = g^{e_i} \in G_2$ for $i = 0$ to $L$ and returns $\{s_{1,i}, s_{2,i}, s_{3,i}, t''_i\}_{i=0}^L$ to $U$. $U$ computes $t_i = t'_i + t''_i \mod p$ for $i = 0$ to $L$ and stores $\{\{s_{1,i}, s_{2,i}, s_{3,i}, s_i, t_i\}_{i=0}^L, x, u\}$ as his wallet $W$.

(Spend Protocol.) $U$ with $W = (\{s_{1,i}, s_{2,i}, s_{3,i}, s_i, t_i\}_{i=0}^L, x, u)$ engages with merchant $M$ (with public key $pk_M$) first agrees on a random challenge $R = H(\text{info}|\text{nonce}|pk_M|2^\ell)$ where nonce is a random number issued by $M$, info is the transaction information and $2^\ell$ is the transaction amount. $U$ chooses an unused node in the binary tree $\text{Tree}_x$ of level $i := L - \ell$. Let $k_{i,j}$ be the node key being chosen. $U$ computes $w_{i,j} = g^{\prod_{k=0,k \neq j}^{i-1}(a_i+k_{i,j})}$, which is the corresponding witness of $k_{i,j}$ in the accumulator of value $V_i$. $U$ also computes $S = g_2^{k_{i,j}}, T = h^ug_3^{Rk_{i,j}}$ and the following $\text{SPK}_{39}$ (denoted as $\pi_S$).

\[
\text{SPK}_{39}\left\{
\begin{array}{l}
\hat{e}(s_{1,i}, g) = Z\hat{e}(V_i h_1^{s_i}, s_{3,i}) \land \\
\hat{e}(s_{2,i}, s_{3,i}Z) = \hat{e}(gh_2^x g_1^{t_i} g_{g_1}^{t_i}, g) \land \\
\hat{e}(w_{i,j}, g^{k_{i,j}v_i}) = \hat{e}(g, g) \land \\
T = h^ug_3^{Rk_{i,j}} \land \\
S = g_2^{k_{i,j}}
\end{array}\right\}(R)
\]

$U$ sends $(\pi_S, S, T)$ to $M$. $M$ accepts the payment if $\pi_S$ is a valid $\text{SPK}$ and stores the coin as $(\pi_S, S, T, \text{info}, \text{nonce}, 2^\ell)$. $U$ marks the node $N_{i,j}$, its ancestors
and all its children in $Tree_2$ as used nodes. $SPK_{39}$ requires instantiation. $U$ randomly generates $r_1, r_2, r_3, r_4, r_5, r_6, r_7 \in R \mathbb{Z}_p$, computes $A_1 = s_1, g_1^r$, $A_2 = V_1 g_2^r$, $A_3 = s_2, g_2^r$, $A_4 = s_3, g_3^r$, $A_5 = w_{i,j} g_2^r$, $A_6 = g_1^{r_5} g_6^r$, $A_7 = g_1^{r_5} g_7^r$ and computes the following $SPK_{39}$.

$$
SPK_{39}\left\{ \begin{array}{l}
(s, t, k_{i,j}, u, r_1, r_2, r_3, r_4, r_5, r_6, r_7, \\
\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8, \\
A_6 = g_1^{r_4} g_2^{r_6}, \\
1_{G_1} = A_6^{-r_5} g_1^{\beta_3} g_2^{\beta_8}, \\
1_{G_1} = A_6^{-r_3} g_1^{\beta_1} g_2^{\beta_9}, \\
A_7 = g_1^{r_5} g_2^{r_7}, \\
1_{G_1} = A_7^{-r_1} g_1^{\beta_3} g_2^{\beta_8} \end{array} \right. \wedge \left( R \right)
$$

(Deposit Protocol.) $M$ submits $(\pi_S, S, T, info, nonce, 2^\ell)$ to bank $B$ for a deposit of $2^\ell$ coins. $B$ checks if $nonce$ is fresh and $M$ is the owner of public key $pk_M$. Next $B$ verifies $\pi_S$ and credits $M$ if $\pi_S$ is valid.

Finally, $B$ checks if the serial number $S$ is already inside its database. Specially, let $i = L - \ell$ and $B$ executes algorithm $ComputeAllSerialNumbers$, as shown in Fig. 5.5, on input $S, L, \ell$ and obtains the $2^\ell$ serial numbers, $(S_{L,0}, \ldots, S_{L,2^\ell})$, associated with the coin $S$ of value $2^\ell$. $B$ then checks if $(S_{L,0}, \ldots, S_{L,2^\ell})$ is in its database. If yes, $B$ invokes Revoke. Otherwise, it stores the $2^\ell$ entries $(S_i, S, T, R, \pi_S)$, for $i = 0$ to $2^\ell - 1$ in its database.

(Revoke.) Let $(S_{L,i}, S, T, R, \pi_S)$ and $(S'_{L,i'}, S', T', R', \pi_{S'})$ be two entries such that one of the outputs from algorithm $ComputeAllSerialNumbers$ ($S_{L,i} = S'_{L,i'}$ in this case) is the same. If both coins are of the same value (which implies $S = S'$), output $pk = (\frac{T R'}{T' R})^{1/p}$ as the identity of the double-spender.
5.5. Divisible E-Cash

Algorithm ComputeAllSerialNumbers
Input: $S$, $L$, $\ell$
Output: $S_{L,0}, \ldots, S_{L,2^\ell-1}$

For $j = 0$ to $2^\ell - 1$
   $Temp := S; index := j$;
   For $i = \ell + 1$ to $L$
      if $index \mod 2 == 0$ \ Left children of parent node
         $Temp := g^{H_0(Temp)}$ \ $H_0$ for left children
      else \ Right children of parent node
         $Temp := g^{H_1(Temp)}$ \ $H_1$ for right children
      $index := index/2$;
   $S_{L,j} := Temp$;

Figure 5.5: Deposit Protocol - Computation of All Serial Numbers Associated with a Particular Coin

Assume $S \neq S'$, this implies they are of different values. Without loss of generality, assume value of coin $S$ is $2^\ell$ and value of coin $S'$ is $2^{\ell'}$ such that $\ell > \ell'$. Let $S_{L,i}, 0 \leq i \leq 2^\ell - 1$, be the output from ComputeAllSerialNumbers on $(S, L, \ell)$ such that $S_{L,i}$ equals to one of the output serial numbers ($S'_{L,i'}$) from ComputeAllSerialNumbers on input $(S', L, \ell')$. Execute algorithm GetNodeKey, as shown in Fig 5.6, with input $S, i, L, \ell, \ell'$ and obtain $k$. Output $pk = T_{\text{rev}}^{k}$ as the identity of the double-spender.

Algorithm GetNodeKey
Input: $S, i, L, \ell, \ell'$
Output: $k$

$Temp := S; index := i$;
For $j = 1$ to $\ell - \ell'$
   if $index \mod 2 == 0$ \ Left children of parent node
      $k := H_1(Temp, 0)$ \ $0$ for left children
   else \ Right children of parent node
      $k := H_1(Temp, 1)$ \ $1$ for right children
   $index := index/2$;
$Temp := g^k$;

Figure 5.6: RevokeDoubleSpender Algorithm - Computation of a Node Key from a Parent Serial Number

(VerifyGuilt.) Every one can check if $B$ is honestly running Revoke as it does not require any secret of $B$. 
5.5. Divisible E-Cash

5.5.5 Security Analysis of Our Divisible E-Cash

As mentioned before, we employ a statistical approach to enhance the efficiency of our e-cash. We first formally define statistical balance and give a proof sketch for our scheme. We would like to restate the security guarantee of our divisible e-cash as follows. Cheating user can get at most an amount of $L 2^L$ instead of $2^L$ in our construction. Consequently, if bank $B$ inspects the withdrawal requests with probability $1/L$ and imposes a fine of $L^2 2^L$, the expected gain of a cheating user will be the same as an honest user.

A Weaker Model: Statistical Balance

Roughly speaking, what we wish to model in statistical balance is that the maximum gain of an adversary $A$ is $\text{bal}_G$ for a cheating withdrawal query. A cheating withdrawal query is an attempt to run Withdrawal Protocol with the bank $B$ such that if $B$ asks $A$ to reveal the random numbers, $A$ will be found cheating. Specifically, we wish to model the fact that when the adversary runs $q_1$ honest Withdrawal Protocol and $q_2$ cheating Withdrawal Protocol, he can only get $2^L q_1 + P q_2$ dollars. Formally, the GAME S-Balance is the same as GAME Balance except oracle $\mathcal{O}_{WH}$ is replaced by $\mathcal{O}'_{WH}$. And, there exists a moderator $M$ whose sole purpose is to decide if an withdrawal request is cheating or not.

$\mathcal{O}'_{WH}$. This oracle allows the adversary $A$ to instruct an execution of Withdrawal Protocol. $A$ supplies a public key $pk \in \mathcal{U}_A \cup \mathcal{U}_H$. If $pk \in \mathcal{U}_H$, the oracle simulates a protocol run between honest user $pk$ and an honest bank. An entry $(i, W_i, pk, K)$ is added to the set $\mathcal{U}_W$. Counter $i$ is then incremented by 1. The index $i$ is given to the adversary.

On the other hand, if $pk \in \mathcal{U}_A$, the oracle first makes a copy of the adversary $A$ as $A'$. The oracle itself performs the role of an honest bank and conducts Withdrawal Protocol with the adversary $A$ with the exception that the oracle never asks $A$ to reveal his randomness. The moderator $M$ will act as an honest bank who always asks $A'$ to reveal his randomness. If $A'$ is found cheating, $M$ outputs a bit 1 (otherwise $M$ outputs 0). Finally, if the protocol between the oracle and $A$ terminates successfully and $M$ outputs 0, $\text{bal}_A$ is increased by $K$. If $M$ outputs 1, $\text{bal}_A$ is increased by $\text{bal}_G$. 
The Proof

We give a proof sketch to show why our divisible e-cash scheme is secure.

Proof: (Sketch.)

Statistical Balance.

Let $\mathcal{A}$ be an adversary that wins GAME S-Balance with non-negligible advantage. We outline why the success probability of $\mathcal{A}$ is negligible due to the unforgeability of $\text{ESS}^+$ and the security of the bounded accumulator by constructing a simulator $\mathcal{S}$ acting as challenger $\mathcal{C}$. $\mathcal{S}$ is given a signing oracle for $\text{ESS}^+$ and its goal is to forge a $\text{ESS}^+$ signature or output $2^L + 1$ value-witness pair for the pairing-based bounded accumulator with bound $2^L$.

Oracle Simulations.

$\mathcal{O}_U$. If a public key is supplied, store it in $\mathcal{U}_A$. Otherwise, randomly generate $u$ and set $pk_u$ to $g^u$. Add $pk_u$ to $\mathcal{U}_H$ and return $pk_u$ to the adversary.

$\mathcal{O}_{WH}$. If $pk_u \in \mathcal{U}_A$, simulate the protocol by invoking the signing oracle of $\text{ESS}^+$. Note that it involves rewinding PK$_3$. $\mathcal{S}$ obtains the values $(s_i, t'_i, u)_i$. Define the value $V_i$ as $C_{M_i}/h_{i}^s$. During the process, $\mathcal{S}$ gets the whole wallet $\mathcal{W} = \{(s_1, s_2, s_3, s_i, t_i, V_i), u\}$ and it stores it in a set $\mathcal{W}$.

$\mathcal{S}$ also plays the role of $\mathcal{M}$, who asks the copy of the adversary $\mathcal{A'}$ to reveal $x, \{V_i, s_i, t_i\}$. If $\mathcal{A'}$ refuses, or $V_i$ is not formed correctly with respect to the binary tree $\text{Tree}_x$ using $x$ as root node key value, $\mathcal{S}$ increases $\text{bal}_A$ by $L2^L$. Otherwise, increase $\text{bal}_A$ by $2^L$.

If $pk_u \in \mathcal{U}_H$, randomly generate $x$ and compute the corresponding $V_i$’s from the binary tree $\text{Tree}_x$ formed by using $x$ as root node key. Stores the wallet as $\mathcal{W} = (\cdot, \{V_i\}, x, u)$. The first term refers to the set of $L + 1 \text{ESS}^+$ signatures which is unknown to the simulator.

$\mathcal{O}_S$. If it involves an honest user (regardless of whether merchant is adversary-controlled or not) simply uses the knowledge simulator to finish the view of the protocol. More specially, for user $(h^u, u)$ with wallet, simulates the protocol as if he is in possession of a wallet $\mathcal{W} = (\{\sigma_i, V_i\}, x, u)$. This requires the knowledge simulator of $\sigma_i$ and its presence is due to the Prove privacy of $\text{ESS}^+$. Mark the used nodes, its ancestors and children as used.
If it involves adversary-controlled user to an honest merchant, $S$ rewinds $\text{SPK}_{39}$ to obtain the underlying $(V_i, k_{i,j}, w_{i,j}, u)$ used.

**Outputs.** Finally, $A$ runs deposits of $\text{bal}_A + 1$ coins to $S$. If the coin of serial number $S$ corresponds to an amount of $2^i$, $S$ generates the $2^i$ serial numbers with it as an honest bank would do. $A$ wins the game either by (1) all the $\text{bal}_A + 1$ serial numbers in Deposit Protocol are unique or (2) some of the serial numbers are duplicated but $\text{Revoke}$ on the corresponding Deposit Protocol transcripts does not output any $\text{pk} \in U_A$. Now we are to analyse these two cases separately.

Case (1): Due to the security of $\text{ESS}^+$, each $O'_W \text{H}$ query only gives $A L + 1 V$’s to work with. Due to the setting of the game, the total “valid” serial numbers obtained by $A$ from $S$, minus the serial numbers that have been presented to $S$ during $O_S$ query is $\text{bal}_A$.

Note that if it is cheating, the root and the children might be independent. Thus, a cheating withdrawal gives at most $L2^L$ serial numbers. Due to the setting of the game, the counter $\text{bal}_A$ will be increased by $L2^L$.

$A$ can only win in Case (1) by convincing $S$ to accept a serial number $S$ that is not one of these $\text{bal}_A$ serial numbers or for any $V_i$, $i = 0$ to $L$, that it obtains an $\text{ESS}^+$ signature, he can generate $2^i + 1$ value-witness pairs. If this is the case, $S$ rewinds all $\text{SPK}_{39}$ and obtains $2^i + 1$ value-witness pairs, thus breaking the security of the bounded accumulator. Otherwise, $A$ must have conducted a false proof as part of the signature of knowledge, $\text{SPK}_{39}$, such that one of the following is fake:

1. Possession of a $\text{ESS}^+$ signature on block of messages $(V_i, u)$
2. Possession of a pair $(w_{i,j}, k_{i,j})$ such that $\Omega(w_{i,j}, k_{i,j}, V_i) = 1$
3. $S = g_2^{k_{i,j}}$.

Fake proof of possession of $\text{ESS}^+$ happens with negligible probability. Fake proof that $S$ is well-formed happens with negligible probability under the discrete logarithm assumption. Thus, $A$ can only win if he could generate a new $\text{ESS}^+$ signature and $S$ can use it to break the unforgeability of $\text{ESS}^+$.

Case (2): We have shown in case (1) that $A$ cannot convince an $S$ to accept an invalid serial number with non-negligible probability. We now suppose duplicated $S$ are accepted.
It remains to show the associated \((T)\) is bounded by specification except with negligible probability so that the correctness of the Revoke implies the recovering of a \(pk \in U_A\). Due to the soundness of the proof of knowledge protocol \(SPK_{39}\), \(T = h^u g_3^{Rk_{i,j}}\). Thus, \(A\)'s success probability is negligible unless he could solve the DL of \(g_3\) to base \(h\).

**Anonymity.** Anonymity is straightforward. Due to Issue and Prove privacy, Withdrawal Protocol, Spend Protocol and Deposit Protocol do not contain any information about the underlying secret except the values \(S, T\) and \(pk_u\). More specifically, recall that for a wallet \(W = (\{\sigma_i, V_i\}, x, u)\), the values \((V_i, x, u)\) are not known to anybody except the underlying user. Some information about \(u\) is known since \(h^u\) is the public key. The only information leaked about a user or wallet during Withdrawal Protocol is \(h^u, \{\sigma_i\}\) (since Issue privacy of ESS+ does not guarantee privacy about the signature issued) while the only information leaked about a user or wallet during Spend Protocol is \(S\) and \(T\). The randomness of \(S\) and \(T\) are guaranteed due to the DDH assumption. In particular, due to the security of the hash functions \(H_0\) and \(H_1\), the node keys \(k_{i,j}\) and \(k_{i,j'}\) cannot be linked together. Indeed, the only linkable part of a binary tree in our construction is from a parent node to its children which will not appear simultaneously unless the wallet is overspend.

**Exculpability.** In Revoke, both transcripts contain the proof of correctness of \(U\), which involves proving knowledge of the user secret \(u\). To slander an honest user, adversary without knowledge of \(u\) has to produce a fake proof of \(SPK_{37}\) and this happens with negligible probability under the discrete logarithm assumption. □

### 5.5.6 Efficiency Analysis

Table 5.1 summarises the complexities of different protocols of our scheme and CG07. The cost of the protocol with pre-processing of our scheme is shown as well. It is somehow hard to quantify the exact cost of the spend protocol of CG07 as the instantiation of its ZKPoK is very complex. Furthermore, it involves \(L + 1\) cyclic groups of different orders. We simplify the comparison by stating the total number of group elements needed. If the original CL signature \([CL02a]\) is used, as stated in CG07, the group \(G\) in their scheme would be the group of quadratic residue modulo a safe-prime product \(n\), which would be of 1024-bit. \(t\) is the security parameter controlling
the cheating probability of the proof-of-knowledge of double-discrete logarithm. For example, \( t = 80 \) would give the protocol a cheating probability of \( 2^{-80} \).

For a moderate value \( L = 10 \) and \( t = 40 \)\(^{17}\), spending a coin of monetary value 1 in CG07 requires 816 and 857 multi-based exponentiations from the user and the merchant respectively, and a total bandwidth of 981 elements in \( \mathbb{Z}_{|G|}^* \) and 28 elements in \( G \). If the base group is of order \( n \) which is 1024-bit, each of the above elements is at least 1024-bit in size. On the contrary, spending a coin of any monetary value in our scheme requires a constant cost of 21 and 13 multi-based exponentiations from the user and the merchant respectively, and a total bandwidth of 9 elements in \( G \) and 21 elements in \( \mathbb{Z}_{|G|}^* \) is needed.

A final note is that, we assume the bank requires the user to reveal its random numbers with probability \( 1/2 \). That is, on average, the user would need to carry out two withdraw protocols with the bank for withdrawing a single wallet of \( 2^L \) coins. Since our scheme is only statistical, the comparison favors our scheme in this regard.

<table>
<thead>
<tr>
<th>Withdrawal Protocol</th>
<th>Time Complexities</th>
<th>CG07</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>User</td>
<td>Bank</td>
</tr>
<tr>
<td>multi-EXP</td>
<td>( 2^L + 1 + 9L + 5 )</td>
<td>( 2^L + 1 + 8L + 6 )</td>
</tr>
<tr>
<td>Pairing</td>
<td>( 2L + 2 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>Spend Protocol</td>
<td>User</td>
<td>Merchant</td>
</tr>
<tr>
<td>(coin of value ( 2^{L-i} ))</td>
<td>w/o Preproc.</td>
<td>w/ Preproc.</td>
</tr>
<tr>
<td>multi-EXP</td>
<td>21</td>
<td>6</td>
</tr>
<tr>
<td>Pairing</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.1: Time and Space Complexities of Our Scheme and CG07.

### 5.6 Extension to Coin Tracing and Revocability

We outline how to achieve coin tracing for our e-cash systems. We consider two types of coin-tracing, namely, coin-tracing for double-spender only and universal coin-tracing. Coin-tracing for double-spender means that all coins from the double-spender can be traced (and identified), regardless it has been spent honestly or not.

\(^{17}\)The corresponding cheating probability is \( 2^{-40} \), which favors CG07.
Universal coin-tracing means that there exists an external party, called judge, who is able to output some tracing information $tr_U$ for a user $U$. With $tr_U$, everyone can trace the coins from $U$.

We also consider revocability, that is, for every transaction, there exists a trusted party, called open manager, who is able to output the public key of the spender.

Indeed, one reason that we are inclined to design e-cash systems that are truly anonymous is that one can easily add external parties for the above features.

### 5.6.1 Tools - Verifiable Encryption

Roughly speaking, a verifiable encryption is an encryption scheme with a proof of correctness of the ciphertext. The encrypter encrypts a message $m$ under the public key $pk_O$ to produce a ciphertext $CT_m$. Only the owner of $sk_O$ corresponding to the public key $pk_O$ can decrypt $CT_m$ and recover $m$. Verifiable encryption allows the encrypter to prove, in zero-knowledge, that he knows the message $m$, which is the encrypted to $CT_m$ and that $m$ satisfies certain relationship. For example, $m$ might be the opening of some other commitments, or that $m$ is within certain range. The first verifiable encryption system that provides chosen ciphertext security without using inefficient cut-and-choose proofs was due to [CS03].

### 5.6.2 Revocability

Revocability can be trivially added. Let $(pk_O, sk_O)$ be the public/private key of the open manager of a verifiable encryption scheme. For every spending protocol, the user $U$ also verifiably encrypts his public key $pk_U$ or secret key $u$ under the public key of the open manager. This gives the open manager the power of revoking the identity of every spender.

### 5.6.3 Coin Tracing of Double-Spender

We have introduced three e-cash schemes, namely, practical compact e-cash (Scheme 1), compact e-cash from bounded accumulator (Scheme 2) and divisible e-cash from bounded accumulator (Scheme 3). Scheme 1 and scheme 3 extract the public key $h^u$ of the double-spender while scheme 2 extracts the secret key $u$ (with corresponding public key $pk_U = h^u$).
5.6. Extension to Coin Tracing and Revocability

The idea of coin tracing of spender is to have the user verifiably encrypt a number tr under his own public key during the withdrawal protocol which would allow tracing of his coins. If the user’s secret key is revoked when he double-spends, B can decrypt and obtain tr, which will allow everyone to trace the spending of this user. We borrow the idea from traceable signatures [KTY04]. Recall that in our three schemes, user U obtains a signature from B on his secret key u. U and B also need to generate together a random number tr. Now, B’s signature includes tr. User verifiably encrypts tr during the withdrawal protocol under his own public key. During the spend procedure, the user computes a one time tag \( h_0 = g^\beta \) for some randomly generated \( \beta \in_R \mathbb{Z}_p \), and a tracing tag \( T_t = h_0^{tr} \). Thus, when the secret key of the user is exposed when he double-spends, B decrypts tr and anyone who knows tr could test if the coin is from this double-spending by testing if \( T_t \overset?= h_0^{tr} \). There is a simplification for our Scheme 1 (resp. Scheme 3). That is, to set \( tr \) to be s (resp. x). With s (resp. x) in Scheme 1 (resp. Scheme 3), spending of the user can be linked.

The remaining problem is that the value \( h^u \) outputted from scheme 1 and 3 for double-spender is a public key. Luckily, we can make use of the method due to [CHL05] in which \( h^u \) is the secret key which corresponds to a public key \( \hat{e}(h^u, g) \). This key structure is employed in bilinear encryption. One can turn any encryption schemes into a verifiable encryption scheme using the cut-and-choose method, as shown in [CD00]. However, the resulting verifiable encryption scheme is not very efficient.

5.6.4 Universal Coin-Tracing

Universal coin-tracing can be added to our scheme in a similar way as coin-tracing for double-spender. Instead of verifiably encrypting tr under the user U’s own public key, U verifiably encrypts it under the public key of the tracing manager. The tracing manager can trace the spending of the same user by decrypting all the tr of U in all withdrawal procedures. With tr, everyone can test if \( T_t \overset?= h_0^{tr} \), which allows efficient tracing.
5.7 Chapter Summary

In this thesis, we are interested in electronic cash which is truly anonymous. In this Chapter, we constructed 3 different electronic cash schemes based on our primitives. In all our constructions, no collusion of any parties together can learn anything about the spending of an honest user. The following table summaries our contributions to electronic cash. In the following table, BA stands for bounded accumulator.

<table>
<thead>
<tr>
<th>Scheme 1</th>
<th>Anonymous?</th>
<th>Compact?</th>
<th>Divisible?</th>
<th>Assumption</th>
<th>Building Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>q-SDH</td>
<td>BBS+</td>
</tr>
<tr>
<td>Scheme 2</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>AWSM</td>
<td>ESS+, BA</td>
</tr>
<tr>
<td>Scheme 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>AWSM</td>
<td>ESS+, BA</td>
</tr>
</tbody>
</table>

Table 5.2: Summary of our Proposed Electronic Cash Systems
Chapter 6

Other Privacy-Preserving Applications

This chapter focuses on the application of the primitives introduced in previous chapters to other privacy-preserving cryptographic systems. In particular, we discuss the application of the primitives to \(k\)-times anonymous authentication (\(k\)-TAA) and attribute-based anonymous credential system (ABACS).

6.1 Dynamic \(k\) Times Anonymous Authentication

Teranishi, Furukawa and Sako [TFS04] proposed \(k\)-times anonymous authentication (\(k\)-TAA) system (TFS04) so that users of a group can access applications anonymously while application providers can decide the number of times users can access their applications. In its simplest form, there are three entities, namely, a group manager (GM), several application providers (AP’s) and a bunch of users (U’s). Each AP, denoted as AP\(_i\), announces independently a bound \(K_i\), which is the allowable number of times a valid user can access its application (or services). A user U is required to register to GM first before he is able to enjoy services from any AP’s. After registration, U can authenticate himself to the AP’s anonymously, up to the allowable number of times for each AP. Anyone can trace a dishonest user who tries to access an AP, say, AP\(_i\) for more than the allowable number of times (\(K_i\)). Application of \(k\)-TAA includes trial browsing.

Requirements. A basic \(k\)-TAA has to satisfy three requirements, namely, anonymity, accountability and exculpability, describe below.

1. **Anonymity.** Users in a \(k\)-TAA enjoy high level of anonymity (or privacy). Anonymity means that the habit of an honest user remain anonymous. No parties, even with the collusion of GM and AP’s, could tell anything about the
underlying user in an authentication. This implies authentications of an honest user cannot be linked. More formally, the requirement is that no collusion of parties could distinguish authentication executions between two honest users who are eligible to access the AP.

2. **Accountability.** A subset of colluding users cannot perform the authentication procedure with the same honest application provider for more than the allowed number of times without being detected.

3. **Exculpability.** No collusion of parties, including GM and AP can produce a *false* proof that an honest user has authenticated himself for more than the allowable number of times.

**Dynamic $k$-TAA** In the original $k$-TAA, an AP has no control over whom can access its application. Nguyen and Safavi-Naini \cite{NSN05} (NS05) proposed dynamic $k$-TAA to address this issue. In dynamic $k$-TAA, user U is required to register with AP, if U wishes to access the service of AP. Moreover, AP could later revoke the access of U even if U has not authenticated himself to AP for up to $K$ times.

### 6.1.1 Related Works

TFS04 and NS05 are $k$-TAA that employ the so-called tag-base mechanism to provide anonymity and accountability. In particular, the complexity of user registration and authentication is $O(\lambda)$ and $O(k\lambda)$ respectively where $\lambda$ is the security parameter and $k$ is the allowable number of access. Later, Teranishi and Sako \cite{TS06} (TS06) proposed a $k$-TAA scheme with constant proving cost. Their scheme is based on some techniques from compact e-cash.

Indeed, as pointed out in TS06, $k$-TAA schemes share a lot of similarities with compact e-cash schemes. In particular, the role of GM is similar to bank B. The user U registering with GM is analogous to U withdrawing a wallet of $k$ coins from B. A wallet which can only be spent $k$-times is analogous to a registered user who can only access an application for $k$ times. The requirements of balance, anonymity and exculpability in compact e-cash are also very similar to the requirement of accountability, anonymity and exculpability in $k$-TAA. Again, there are certain differences. For compact e-cash, a user can only spend (c.f. authenticate) up to $k$ times to all merchants (c.f. application providers) combined; while for $k$-TAA, each application
provider may choose its own allowable number of access and the number of accesses to different applications by a user are not related. For instance, a user could authenticate himself $k_1$ (resp. $k_2$) times to $\text{AP}_1$ (resp. $\text{AP}_2$), provided that $k_1 \leq K_1$ (resp. $k_2 \leq K_2$), where $K_1$ (resp. $K_2$) is the limit imposed by the respective AP’s. Despite these differences, similar techniques can be used to build $k$-TAA and compact e-cash.

Another closely related notion is event-oriented $k$-times revocable-iff-linked group signatures ($k$-RiffLGS), introduced by Au, Susilo and Yiu [ASY06]. $k$-RiffLGS is a group signature scheme that every user can sign on behalf of the group anonymously for up to $k$ times per event, represented by a bit string. No one, not even the group manager, can revoke the identity of the signers. Signatures of the same user for different events cannot be linked. If the user signs for more than $k$ times for any events, his identity is revealed. Thus, every legitimate user can sign on behalf of the group for up to $k$ times per event and there are no limits for the number of events. In fact, as mentioned in [ASY06], $k$-RiffLGS can be viewed as a non-interactive version of $k$-TAA. It turns out that the $k$-RiffLGS by Au, Susilo and Yiu is similar to TS06. While both schemes achieve complexity $O(\lambda)$ and $O(\lambda \log(k))$ respectively for user registration and authentication respectively, TS06 offers a trade-off that lowers the authentication complexity to $O(\lambda)$ by increasing the public key size to $O(k)$.

Our construction can be seen as an extension of TS06 to dynamic $k$-TAA with the use of dynamic accumulator.

### 6.1.2 Syntax

We briefly review the syntax of dynamic $k$-TAA. A dynamic $k$-TAA system is a tuple $(\text{GMGen}, \text{APGen}, \text{Join Protocol}, \text{Grant Access}, \text{Revoke Access}, \text{Auth}, \text{Trace})$ of seven polynomial-time algorithms/protocols between three entities, namely, the group manager $\text{GM}$, application provider $\text{AP}$ and user $\text{U}$.

($\text{GMGen}$.) On input a security parameter $1^\lambda$, the algorithm outputs $\text{GM}$’s key pair $(\text{gpk}, \text{gsk})$. $\text{gpk}$ includes an membership achieve $\text{mList}$ which is empty initially.

($\text{APGen}$.) On input $\text{gpk}$, the algorithm outputs a key pair $(\text{apk}_i, \text{ask}_i)$ for user $\text{AP}_i$. $\text{apk}_i$ includes a bound $K_i$ representing the allowable number of access to the application provided by $\text{AP}_i$. $\text{AP}_i$ also maintains a membership list $\text{apList}_i$ which is empty initially.
6.1. Dynamic \(k\) Times Anonymous Authentication

(Join Protocol.) U with input \((gpk)\) registers to GM with input \((gpk, gsk)\) in this protocol. Upon successful termination of the protocol, U’s output is a membership key pair \(pk_U, sk_U\). GM adds \(pk_U\) to its membership achieve \(mList\).

(Grant Access.) This might be an interactive protocol between U and AP\(_i\) or an algorithm for AP\(_i\) which allows AP\(_i\) to grant access of its application to U. If it is an interactive protocol, U might obtain certain secret \(sk_{U,i}\) from AP\(_i\). Finally, AP\(_i\) adds \(pk_U\) to its membership list \(apList_i\).

(Revoke Access.) This algorithm allows AP\(_i\) to revoke the access right of user U. After execution of the algorithm, \(pk_U\) is removed from membership list \(apList_i\).

(Auth.) This protocol allows U to authenticate himself to AP\(_i\). U keeps a counter \(k_i\) which is the number of times it has authenticated himself to AP\(_i\) and he could authenticate himself to AP\(_i\) if and only if \(k_i \leq K_i\) and that \(pk_U\) is in \(apList_i\). AP\(_i\) stores the authentication transcript \(\pi\) in the authentication log \(alog_i\).

(Trace.) On input \(alog_i\), the algorithm outputs \(pk_U\), GM or valid which indicates user “\(pk_U\) tries to access AP\(_i\) more than \(K_i\) times”, “GM cheated” and “there are no malicious entries in this authentication log”, respectively. This algorithm can be executed by anyone.

6.1.3 The Scheme

We show how \(k\)-TAA can be constructed using the primitives introduced previously. It should be noted that it is in fact very similar to our practical compact e-cash. We present an outline first, followed by the details of our construction.

An Overview of Our Dynamic \(k\)-TAA

(GMGen.) Let \(\text{Sig}_1, \text{Sig}_2\) be two CL Sig’s. GM invokes \(\text{Sig}_1.\text{KeyGen}\) to obtain \((\text{Sig}_1.pk, \text{Sig}_1.sk)\). GM also chooses a cyclic group \(G = \langle g \rangle\) of order \(p\) such that \(Z_p\) is within the message space of \(\text{Sig}_1\). GM also chooses a cryptographic hash function \(H\). The public key \(gpk\) is \((G, p, g, \text{Sig}_1.pk, H, mList)\) and the secret key \((gsk)\) is \(\text{Sig}_1.sk\). \(mList\) is empty initially.

(APGen.) AP\(_i\) chooses a dynamic accumulator \((g \circ f_i : \mathcal{U}_f \to \mathcal{U}_g, u_i, \Omega_i)\). The primitives are chosen such that the message space of \(\text{Sig}_1, \text{Sig}_2\) and \(\mathcal{U}_f\) are
all in \( \mathbb{Z}_p \). AP
\(_i\) invokes \( \text{Sig}_2.\text{KeyGen} \) to obtain \((\text{Sig}_2.pk_i, \text{Sig}_2.sk_i)\). Define the value \( K_i \) as the allowable number of access for AP
\(_i\). The algorithm computes \( \hat{\sigma}_{j,i} \leftarrow \text{Sig}_2.\text{Sign}_{\text{Sig}_2.sk_i}(j) \) for \( j = 1 \) to \( K_i \). AP
\(_i\) also chooses a verifiable random function \( \text{vrf}_i(\cdot) \). The public key \( apk_i \) is \((\text{Sig}_2.pk_i, \text{vrf}_i(\cdot), g \circ f_i, u_i, \Omega_i, K_i, \hat{\sigma}_{1,i}, \ldots, \hat{\sigma}_{K_i,i}, \text{apList})\) such that \text{apList} is empty initially. AP
\(_i\) safely deletes \( \text{Sig}_2.sk_i \) and sets \( \text{ask}_i \) as \((t_f), \text{auxiliary information of the accumulator}\).

(Join Protocol.) User \( U \) randomly generates \((s, t, u, e)\) and obtains a signature \( \sigma_u \) on \((s, t, u, e)\) using \( \text{Issue of Sig}_1\) so that GM learns nothing about \((s, t, u)\) while \( e \) is generated by GM. At the end of the protocol, the entry \((g^u, e)\) is added to \text{mList}\(^2\). U stores \((\sigma_u, s, t, u)\) as his membership secret key.

(Grant Access.) AP
\(_i\) grants access to user \((g^u, e)\) in \text{mList} by setting \text{apList} := \text{apList} \cup \{e\}. AP
\(_i\) also computes \( V_i = g \circ f_i(u_i, \text{apList}) \). If it is an interactive protocol, AP
\(_i\) returns \( w_i \) which is the witness of \( e \) to the user.

(Revoke Access.) AP
\(_i\) revokes the access right of user \((g^u, e)\) in \text{mList} such that \( e \in \text{apList} \) by setting \text{apList} := \text{apList} \setminus \{e\}.

(Auth.) U with input \( sk_U = (\sigma_u, s, t, u) \) such that there exists an entry \((g^u, e) \in \text{mList} \) and \( e \in \text{apList} \) authenticates to AP
\(_i\) in this protocol. Let \( k_i \leq K_i \) be the counter of U such that this is the \( k_i \)-th authentication for U to authenticate himself to AP
\(_i\). AP
\(_i\) first sends a random \( nonce \) to U and both compute random challenge \( R = H(apk_i||nonce) \) locally.

U computes serial number \( S = \text{vrf}_i(s, k_i) \), tracing tag \( T = g^u \text{vrf}_i(t, k_i)^R \), \( V_i = g \circ f_i(u_i, \text{apList}), w_i = g \circ f_i(u_i, \text{apList} \setminus \{e\}) \)\(^3\) and computes the following \( \text{SPK}_{40} \)

\(^1\)This has to be chosen randomly so that the outputs of two \( \text{vrf}_i, \text{vrf}_j \) for the same input cannot be linked. Later we shall see how this can be defined.

\(^2\)User computes \( g^u \) for GM and is required to prove its correctness.

\(^3\)\( w_i \) is the witness of \( e \) inside the accumulator formed by the accumulation of set \text{apList}.}
6.1. Dynamic $k$ Times Anonymous Authentication

(denoted by $\pi_S$).

$$\text{SPK}_{40}\{\left(\sigma_u, s, t, u, e, \hat{\sigma}_{k_i, i}, k_i, w_i\right):}
\begin{align*}
\text{valid} & \leftarrow \text{Sig}_1.\text{Verify}_{\text{Sig}_1, pk}(\sigma_u, s, t, u, e) \land \\
\text{valid} & \leftarrow \text{Sig}_2.\text{Verify}_{\text{Sig}_2, pk_i}(\hat{\sigma}_{k_i, i}, k_i) \land \\
1 & = \Omega_i(w_i, e, V_i) \land \\
S & = \text{vrf}_i(s, J) \land \\
T & = g^{u}\text{vrf}(t, J)^R
\end{align*}
\} (R)

U sends $(S, T, \pi_S)$ to AP$_i$. If $\pi_S$ is valid, AP$_i$ accepts the authentication and stores $(S, T, R, \pi_S, nonce)$ in $\text{alog}_i$. U increments $k_i$ by 1.

(Trace.) Consider the list of the first entry in an authentication log $\text{alog}_i$. If none of them is equal, output valid. Otherwise, there exist two entries $(S, T, R, \pi_S, nonce)$ and $(S, T', R', \pi'_S, nonce')$. If $R = R'$, AP$_i$ is cheating as $R$ is the output of some hash function and nonce should be fresh each time. Otherwise, compute $pk = (\frac{r_{T'}}{T})^{1-R}$. If there exists an entry $(pk, e)$ in $\text{mList}$, output that entry as the cheating user. Otherwise, output GM.

Actual Construction

The actual construction is due to $BBS+$, BB signature and a modified verifiable random function. The role of $e$ in the overview is taken by part of the $BBS+$ signature. It does not breach the security since $BBS+$ is strongly unforgeable.

Common Parameter[^4] Let $\lambda$ be a security parameter. Let $G_1, G_2, G_T, G_p$ be cyclic groups of prime order $p$ such that $p$ is of $\lambda$-bit. Let $\hat{e} : G_1 \times G_2 \rightarrow G_T$ be a bilinear map and $\psi : G_2 \rightarrow G_1$ be an isomorphism from $G_2$ to $G_1$.

(GMGen.) GM randomly generates $g, g_0, g_1, g_2, g_3, g_4 \in_R G_2$, computes $g = \psi(g)$ and $g_i = \psi(g_i) \in G_1$ for $i = 0$ to 4. GM also generates $h \in G_p, g_1, g_2 \in_R G_1$. We assume that relative discrete logarithms are unknown. This can be done by setting them to be outputs of some cryptographic hash function of some publicly known seed (such as the identity of the GM). GM randomly chooses $\gamma \in_R Z_p$, computes $w = g^\gamma \in G_2$. GM also chooses a cryptographic hash function $H$. The public key $gpk$ is $(G_1, G_2, G_T, G_p, \hat{e}, \psi, p, g, g_0, g_1, g_2,$

[^4]: We assume these common parameters are generated by some trusted parties.
6.1. Dynamic $k$ Times Anonymous Authentication

g_3, g_4, g_1, g_2, g_0, g_1, g_2, g_3, g_4, v, h, H, \text{mList},$ where mList is an empty list initially, and the corresponding secret key $gsk$ is $\gamma$.

(APGen.) $AP_i$ randomly chooses $\gamma_i \in R \mathbb{Z}_p$ and computes $v_i = g^{\gamma_i} \in \mathbb{G}_2$. $AP_i$ then computes $\hat{\sigma}_{j, i} = g^{\frac{1}{v_i^{j-1}}}$ for $j = 1$ to $K_i$, where $K_i$ is the maximum number of times any users could access the application of $AP_i$. $AP_i$ randomly chooses $\alpha_i \in R \mathbb{Z}_p$, computes $y_i = g^{\alpha_i} \in \mathbb{G}_2$, $y_{1,i} = g^{\alpha_i}, \ldots, y_{q_i,i} = g^{(\alpha_i)q_i} \in \mathbb{G}_1$ where $q_i$ is the maximum size of apList. Compute $h_i = H(AP_i) \in \mathbb{G}_p$. The public key $apk_i$ is $(v_i, \hat{\sigma}_1, \ldots, \hat{\sigma}_{K_i}, K_i, y_{1,i}, \ldots, y_{q_i}, y_i, h_i, \text{apList})$ such that apList is empty initially. $AP_i$ safely deletes $\gamma_i$ and sets $ask_i$ as $(\alpha_i)$, the auxiliary information of the accumulator.

(Join Protocol.) $U$ randomly generates $s', t, u, r \in R \mathbb{Z}_p$, computes $C' = g_0^{s'} g_1 g_2 g_3^r \in \mathbb{G}_1$, and sends $C', pk_u = h^u$ to $GM$, and conducts the following protocol:

$$PK_{41}\{(s', t, u, r) : C' = g_0^{s'} g_1 g_2 g_3^r \land pk_u = h^u\}$$

$GM$ verifies $PK_{41}$, randomly selects $s'', e \in R \mathbb{Z}_p$, computes $\zeta = (C' g_0^{s''})^\frac{1}{\gamma + e}$ and returns $(\zeta, e, s'')$ to $U$. $U$ computes $s = s' + s''$, checks if $\hat{e}(\zeta, wg^e) = \hat{e}(gg_0^{s'} g_1 g_2 g_3^s, g)$ and stores $(\zeta, e, s, t, u, r)$ as his membership secret key $sk_U$. $GM$ sets $\text{mList} := \text{mList} \cup \{(pk_u, e)\}$.

(Grant Access.) $AP_i$ grants access to user $(pk_u, e)$ in mList by setting apList := apList $\cup \{e\}$. $AP_i$ also computes $V_i = g^{\Pi_{e \in \text{apList}}(\alpha_i + e)}$. If it is an interactive protocol, $AP_i$ returns $w_i = V_i^{\frac{1}{\gamma + e}}$ which is the witness of $e$ in accumulator $V_i$ to the user.

(Revoke Access.) $AP_i$ revokes the access right of user $(pk_u, e)$ in mList such that $e \in \text{apList}$ by setting apList := apList \ $\{e\}$.

(Auth.) $U$ with input $sk_U = (\zeta, e, s, t, u, r)$ such that there exists an entry $(h^u, e) \in \text{mList}$ and $e \in \text{apList}$ authenticates to $AP_i$ in this protocol. Let $k_i \leq K_i$ be the counter of $U$ such that this is the $k_i$-th authentication for $U$ to authenticate himself to $AP_i$. $AP_i$ first sends a random nonce to $U$ and both compute random challenge $R = H(apk_i || \text{nonce})$ locally.

$U$ computes serial number $S = h_i^{\frac{1}{\gamma + k_i}}$, tracing tag $T = h^u h_i^{\frac{R}{\gamma + k_i}}$, $V_i = g^{\Pi_{e \in \text{apList}}(\alpha_i + e)}$, $w_i = g^{(\Pi_{e \in \text{apList}}(\alpha_i + e))/(\alpha + e)}$, auxiliary commitments $\mathfrak{A}_1 = g_1^s$, $\mathfrak{A}_2 = g_2^{s'}$,
\[ \mathcal{A}_2 = c g_2^{r_1}, \mathcal{A}_3 = g_1 g_2^{r_2}, \mathcal{A}_4 = \delta_{k_1} g_2^{r_3}, \mathcal{A}_5 = g_1^{t+k_1} g_2^{r_5}, \mathcal{A}_6 = u g_2^{r_6} \] for some randomly generated \( r_1, r_2, r_3, r_4, r_5, r_6 \in \mathbb{R} \mathbb{Z}_p \) and computes the following \( \text{SPK}_{42} \).

Denote \((\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6, \text{SPK}_{42})\) by \( \pi_S \).

\[ \text{SPK}_{42} \left\{ \begin{array}{l}
\mathcal{A}_1 = g_1^{t} g_2^{r_2} \\
1_{G_1} = g_1^{-r_1} g_2^{\beta_2} \\
\frac{\hat{e}(\mathcal{A}_2, w)}{\hat{e}(g, g)} = \hat{e}(g_0, g)^{s} \hat{e}(g_1, g)^{t} \hat{e}(g_2, g)^{u} \hat{e}(g_3, g)^{r} \\
\hat{e}(g_2, w)^{r_1} \hat{e}(g_2, g)^{\beta_1} \hat{e}(\mathcal{A}_2, g)^{-e} \\
\mathcal{A}_3 = g_1^{r_3} g_2^{r_4} \\
1_{G_1} = g_3^{-k_1} g_2^{\beta_3} \\
\frac{\hat{e}(\mathcal{A}_4, v)}{\hat{e}(g, g)} = \hat{e}(g_2, v)^{r_3} \hat{e}(g_2, g)^{\beta_4} \hat{e}(\mathcal{A}_4, g)^{-k_1} \\
1_{G_1} = g_1^{-r_6} g_2^{\beta_6} \\
\frac{\hat{e}(\mathcal{A}_4, y)}{\hat{e}(v, g)} = \hat{e}(g_2, y_1)^{r_6} \hat{e}(g_2, g)^{\beta_5} \hat{e}(\mathcal{A}_6, g)^{-e} \\
\hat{h}_i = S^s S^{k_1} \\
\mathcal{A}_5 = g_1^{k_1} g_2^{r_5} \\
1_{G_1} = g_5^{-u} g_2^{\beta_6} \\
\hat{h}_i^R = T^1 T^{k_1} h^{-\beta_5} \end{array} \right\} (R) \]

3. If \( \pi_S \) is valid, \( \text{AP}_i \) accepts the authentication and stores \((S, T, R, \pi_S, \text{nonce})\) in \( \text{alog}_i \). \( U \) increments \( k_i \) by 1.

(Trace.) Consider the list of the first entry in an authentication log \( \text{alog}_i \). If none of them is equal, output \text{valid}. Otherwise, there exist two entries \((S, T, R, \pi_S, \text{nonce})\) and \((S, T', R', \pi'_S, \text{nonce}'\) if \( R = R' \), \( \text{AP}_i \) is cheating as \( R \) is the output of some hash function and nonce should be fresh each time. Otherwise, compute \( pk = (\frac{T'}{T})^{1/k-1} \). If there exists an entry \((pk, e)\) in \( \text{mList} \), output that entry as the cheating user. Otherwise, output \text{GM}.

Security Analysis

Security analysis of our \( k \)-TAA is omitted as it is similar to our compact e-cash.
6.2 Attribute-Based Anonymous Credential Systems

In an Anonymous Credential System (ACS) [CL01], only those users who have registered to an organisation can authenticate their membership in the organisation to anyone, e.g., some other organisations, anonymously and unlinkably among the set of all members. In several existing ACS constructions such as Camenisch and Lysyanskaya’s [CL01], a user acquires from an organisation $O$ its signature on her pseudonym as her credential during registration; the credential enables her to authenticate her membership in $O$ later to any verifiers by proving that she has a signed pseudonym in zero-knowledge, i.e., without revealing the pseudonym or the signature.

To support the efficient revocation of credentials, Camenisch and Lysyanskaya [CL02b] proposed that each organisation maintains a dynamic accumulator as a “white-list” of credentials that have not been revoked by the organisation so far. At a high level (we defer the details to later in Section 6.2.1), the organisation adds the user’s credential to its accumulator when she registers and, when desired, removes the credential from it during revocation. The user must thus additionally prove in zero-knowledge that her credential is on the list during authentication. The use of dynamic accumulators enables the generation and verification of such proof in time independent of the size of the list.

Li et al. [LLX07] proposed the use of dynamic universal accumulators as the organisations’ “blacklists” of credentials that have been revoked. Users thus instead prove in zero-knowledge that their credential is not on the blacklist (i.e., in the accumulator) during authentication. Comparing to the “white-listing” approach, an organisation needs not update its accumulator. Hence, any existing registered users need not update their credential when a new user registers.

6.2.1 Overview of the Existing Constructions

Following Camenisch and Lysyanskaya’s original approach [CL01], sometimes known as the CL approach, a series of subsequent works [CL02a, CL02b, CL04, BCKL08] employ CL Sig as a key building block of anonymous credential systems. A brief account of the CL approach is given below. Assume PKI and let each user register a public key. User $U$ registers unlinkable pseudonym $y_{U,1}$ and $y_{U,2}$ with organisation
6.2. Attribute-Based Anonymous Credential Systems

$O_1$ and $O_2$. $y_{u,1}$ and $y_{u,2}$ are simply commitments of U’s secret key and they are unlinkable due to the hiding properties of the commitment scheme. Organisation $O_1$ grants a credential to U by issuing a CL signature $\sigma_{1,U}$ on U’s secret key using Issue protocol so that $O_1$ learns nothing about the secret key. To allow dynamic revocation of credential, $\sigma_{1,U}$ (actually, part of it) is put into the dynamic accumulator maintained by $O_1$. To demonstrate a credential from $O_1$ to verifier $V$, U conducts a ZKPoK that he is in possession of $\sigma_{1,U}$ on his secret key, and that $\sigma_{1,U}$ is inside the accumulator of $O_1$. $O_1$ revokes the credential of U by removing $\sigma_{1,U}$ from his accumulator.

With the introduction of universal accumulator, $O_1$ maintains an accumulator of revoked credential instead. U demonstrates a credential from $O_1$ to verifier $V$ by proving that she is in possession of $\sigma_{1,U}$ on his secret key and that $\sigma_{1,U}$ is not inside the accumulator of $O_1$. This approach can be more desirable as the list of revoked credentials is usually shorter than that of the issued credentials.

6.2.2 A Generalisation to ACS

We introduce attributed-based anonymous credential system (ABACS) as a generalisation of the more conventional anonymous credential system. In ABACS, users are in possession of one or more boolean attributes and would like to convince some verifier of the validity of some statement about the attributes they possess and/or lack, in such a way that no information other than the validity of the statement is leaked.

Attribute-based anonymous authentication is an immediate application of ABACS, wherein the server is willing to grant a user access to an object, e.g., a file or a service as long as the attributes the user possesses and/or lacks satisfy the server’s access control policy on the object, while the user desires to access the object by revealing as little information (e.g., her identity, the attributes she possesses and/or lacks) as possible for privacy reasons.

Consider the following example. The Biology department reserves a free parking area near its building for its students and any visitors from outside the department. (Staff in the department must pay to park.) Access control at the entrance hence enforces a policy of “$(\text{Student} \land \text{Bio}) \lor (\neg \text{Bio})$.” Although identifiable authentication such as waving a RFID card at the entrance could be used for access control, it would violate the desire of some users to keep their location traces (i.e., when their car
enters and leaves the parking area) private from others, including the access control system.

As another example, a pharmacist must ensure that \( \text{Fever} \land \neg \text{Asthma} \) holds for a patient before dispensing Aspirin, as many asthma sufferers are allergic to Aspirin, while patients may not want to disclose their entire medical record, e.g., when it contains an unrelated genetic disorder.

**Requirements.** ABACS provides a solution to the above challenges. Specifically, it achieves the following goals:

1. **Fine-grained access control policies.** The server can enforce any access control policy that can be expressed as a formula of boolean attributes in the Disjunctive Normal Form (DNF), i.e., one that is a disjunction of terms, where each term is a conjunction of possibly negated boolean attributes. Here, negation of an attribute represents the lack of that attribute.

2. **User Privacy.** The server knows only whether an authenticating user satisfies its access control policy. More precisely, authentication attempts by honest users who satisfy (resp. do not satisfy) the server’s access control policy are anonymous and unlinkable among the set of all users who also satisfy (resp. do not satisfy) the policy.

3. **Attribute-CA Autonomy.** A user’s possession of an attribute is certified by an authority called the Attribute Certification Authority (Attribute-CA). ABACS permits a separate and independently-operated attribute-CA for the certification of each attribute, which facilitates its support for a large number of attributes with diversified nature, e.g., certification procedures and trust implications.

4. **Practical Negation Support.** A user in ABACS can prove her lack of an attribute without any prior setup. If this was not true, users might have to contact the corresponding attribute-CA for each of the attributes that they lack. The number of such attributes often dominates the number of attributes any user actually possesses.
6.2.3 Syntax

An attribute-based anonymous credential system is a tuple \((\text{OGen}, \text{UGen}, \text{Nym Protocol}, \text{Certify Attribute}, \text{Revoke Attribute}, \text{Auth})\) of six polynomial-time algorithms/protocols between three entities, namely, attribute-CA \(O_j\), user \(U\) and verifier \(V\).

\((\text{OGen}.\)\) On input a security parameter \(1^\lambda\), the algorithm outputs \((pk_j, sk_j)\), the key pairs for attribute-CA \(O_j\), who is responsible for certifying attribute \(\text{attribute}_j\). In particular, \(pk_j\) includes an member list \(\text{list}_j\) which is empty initially.

\((\text{UGen}.\)\) On input \(1^\lambda\), the algorithm outputs a secret key \((x_U)\) for user \(U\).

\((\text{Nym Protocol}.\)\) \(U\) with input \((pk_j, x_U)\) generates a pseudonym \(y_{U,j}\) with attribute-CA \(O_j\) (with input \(pk_j, sk_j\)) in this protocol. A pseudonym is a piece of bit-string that \(O_j\) recognises \(U\).

\((\text{Certify Attribute}.\)\) This might be an interactive protocol between \(U\) and \(O_j\) or an algorithm for \(O_j\) which allows \(O_j\) to certify that a user \(U\) whom he recognises by pseudonym \(y_{U,j}\) possesses the attribute \(\text{attribute}_j\). We say \(U\) is a member of \(O_j\) or \(U\) possesses attribute \(\text{attribute}_j\) upon successful completion of the protocol (or algorithm). \(O_j\) adds \(y_{U,j}\) to its member list \(\text{list}_j\).

\((\text{Revoke Attribute}.\)\) This algorithm allows \(O_j\) to take away the certification of attribute \(\text{attribute}_j\) to user \(U\) whom he recognises by pseudonym \(y_{U,j}\). After execution of the algorithm, \(y_{U,j}\) is removed from the membership list \(\text{list}_j\).

\((\text{Auth}.\)\) This interactive protocol allows \(U\) to demonstrate to a verifier \(V\) that he is in possession of a set of attributes satisfying certain policy \(\text{policy}\). In particular, an ABACS should support \(\text{policy}\) in the form: \(\text{policy} = \bigvee (\text{policy}_k)_{k=1}^t\) such that \(\text{policy}_k\) is the conjunction of any number of the following statements.

1. \(U\) possesses attribute \(\text{attribute}_j\)
2. \(U\) is the owner of a pseudonym \(y_{U,j^*}\) with \(O_{j^*}\).
3. \((\text{Negation of 1}).\) \(U\) does not possess attribute \(\text{attribute}_j\)
4. \((\text{Negation of 2}).\) \(U\) is not the owner of a pseudonym with \(O_{j^*}\).
6.2.4 Our Construction

We first give an overview of how ABACS can be constructed from DMA, followed by details of our construction.

Overview of Our Approach

The CL approach does not lead to ABACS that supports practical negation of an attribute even with the help of universal accumulator. The difficulty is that a credential can only be put on the accumulator after it has been generated. Consequently, the user must have contacted the organisation first before he could claim non-possession of the credential. This violates the goal that user should be able to prove her lack of an attribute without any prior setup.

We build our ABACS from DMA instead of CL Sig. Assume PKI (in fact, CA in our system can be one of the attribute-CA, denoted as $O_{CA}$), $O_{CA}$ chooses an one-way pseudorandom function (OWPRF) $h_{CA}$. The public key of a user $U$ is $h_{CA}(x)$ for some secret key $x$. Each attribute-CA $O_j$, who is responsible for managing attribute $attribute_j$, chooses a DMA and an OWPRF $h_j$. To obtain a certification of attribute $attribute_j$, $U$ sends $y_{U,j} = h_j(x)$ to $O_j$, who simply adds $y_{U,j}$ to its DMA. $y_{U,j}$ is treated as the pseudonym between $U$ and $O_j$. Suppose $U$ has obtained attribute $attribute_{j_1}$ and $attribute_{j_2}$. $U$ is able to demonstrate possession of attributes $attribute_{j_1}, attribute_{j_2}$ and non-possession of attribute $attribute_{j_3}$ by proving in zero-knowledge, that he has the knowledge of a value $x$ such that $h_{j_1}(x)$ is in the accumulator of $O_{j_1}$ and $h_{j_2}(x_i)$ is in the accumulator of $O_{j_2}$ and $h_{j_3}(x)$ is not in the accumulator of $O_{j_3}$.

Note also that $U$ is not required to contact $O_{j_3}$ in order to demonstrate non-possession of attribute $attribute_{j_3}$. To ensure that pseudonyms of the same user to different organisations are unlinkable, we required that the functions $h_{j_1}, h_{j_2}, h_{j_3}$ are pseudorandom and are randomly chosen. In some sense, the anonymity guarantee of pseudonyms for our ABACS is weaker than that of CL systems. In our case pseudonyms are OWPRF on secret key while they are commitments of secret key in CL systems. The latter can be perfectly hiding while the former can only be computationally indistinguishable. Looking ahead, CL systems achieve perfect anonymity while ours is only computational under the DDH assumption.

As a comparison, one could say the anonymity offered by ACS (and ABACS) is

\footnote{Note that $h_{j_1}(x_i), h_{j_2}(x_i), h_{j_3}(x_i)$ are not revealed during the proof.}
better than that of dynamic k-TAA. The reason is that in dynamic k-TAA, everyone can tell, from the membership list of application providers, which set of application providers a user has been granted access to. On the other hand, the pseudonyms of the same user to different organisations cannot even be linked. Thus, the membership list for each organisation is just a bunch of random bit-strings.

Our Construction

It is straightforward to construct ABACS from DMA following the above description.

**Common Parameter.** We assume \( \hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2 \) such that \( |\mathbb{G}_1| = |\mathbb{G}_2| = p \) and \( \mathbb{G}_q \subset \mathbb{Z}^*_p \) with \( |\mathbb{G}_q| = q \) are system-wide parameter. Note that it does not breach the security.

**OGen.** In our system, each \( O_j \) maintains its own DMA \((g \circ f_j)\) working in the bilinear group pair \( \mathbb{G}_1, \mathbb{G}_2 \) and a random OWPRF \( h_j \). To ensure \( h_j \) is chosen randomly, we set \( h_j = H(\text{attribute}_j) \in \mathbb{G}_q \) for some cryptographic hash function \( H \) and define \( h_j : x \mapsto h_j^x \). \( O_j \) also maintains two public lists, \( \text{list}_j \) and \( \mathcal{L}_j \) which are initialised to empty list.

**UGen.** User \( U_i \) chooses his secret key \( x_i \in_R \mathbb{Z}_q \).

**Nym Protocol.** \( U_i \) sends \( y_{i,j} = h_j(x_i) \) to \( O_j \) as his pseudonym. Upon successful completion of the protocol, we say \( U_i \) is the owner of the pseudonym \( y_{i,j} \).

**Certify Attribute.** \( O_j \) certifies attribute \( \text{attribute}_j \) to \( U_i \) by setting \( \text{list}_j := \text{list}_j \cup \{y_{i,j}\} \). For efficiency reason, \( O_j \) returns to \( U_i \) the membership witness \( w_{i,j} \) such that \( y_{i,j} \) is in the accumulator \( v_j = g_j \circ f_j(1, \text{list}_j) \). \( O_j \) also appends the entry \((v_j, y_{i,j}, \text{ADD}) \) to \( \mathcal{L}_j \).

**Revoke Attribute.** \( O_j \) sets \( \text{list}_j := \text{list}_j \setminus \{y_{i,j}\} \). \( O_j \) computes \( v_j = g_j \circ f_j(1, \text{list}_j) \) and appends the entry \((v_j, y_{i,j}, \text{REMOVE}) \) to \( \mathcal{L}_j \). Users use the dynamism algorithms \( \text{Dyn}_2 \) (resp. \( \text{Dyn}_3 \)) of our DMA to update its membership witness (resp. non-membership witness) with the list \( \mathcal{L}_j \).

**Auth.** Consider a policy \( \text{policy} \) (in DNF) of the following form: \( \text{policy} = \bigvee (\text{policy}_k)_{k=1}^t \) such that \( \text{policy}_k \) is the conjunction of any number of the following statements.

\footnote{This is not necessary since the user can compute the witness by himself from \( \text{list}_j \) but \( O_j \) can compute the value more efficiently with the auxiliary information.}
1. $U_i$ possesses attribute $\text{attribute}_j$
2. $U_i$ is the owner of a pseudonym $y_{i,j^*}$ with $O_{j^*}$.
3. (Negation of 1). $U_i$ does not possess attribute $\text{attribute}_j$
4. (Negation of 2). $U_i$ is not the owner of a pseudonym with $O_{j^*}$.

Each statement above has a corresponding zero-knowledge proof-of-knowledge statement.

1. $h_j(x_i)$ is in accumulator $v_j$.
2. $y = h_{j^*}(x_i)$.
3. (Negation of 1). $h_j(x_i)$ is not in accumulator $v_j$.
4. (Negation of 2). $y \neq h_{j^*}(x_i)$.

Thus, to assert policy, one just needs to conduct a zero-knowledge proof-of-knowledge by the disjunction of conjunction of the $\Sigma$-Protocols discussed in Chapter 4, together with the proof that all $x_i$ used in those proofs are equal.

### 6.2.5 Security Analysis

We first give an informal description on the security requirements of ABACS. Then we formally define a security model to capture the security requirements using a simulation-based approach, in the sense of [CL01]. Firstly we define an attribute-based anonymous credential system that relies on a trusted party $T$ as an intermediate. We assume communication between $T$ and any other parties is secure. This is sometimes referred to as an ideal-world specification of an anonymous credential system. Next we will define what it means for a cryptographic authentication system to conform to an ideal-world specification.

**Informal Description**

**User Authenticity.** The server can specify any access policies, and a user can successfully authenticate if and only if the set of attributes he possesses satisfy the policy. In particular, user authenticity must be collusion-resistant: no collusion of users who do not individually satisfy a policy can successfully authenticate w.r.t. that policy.
User Anonymity. Authentication of users with respect to a policy is anonymous and unlinkable among the set of all users who satisfy the policy. This is in fact the minimum amount of information a user must leak to the server for a secure access control.

Attribute-CA Autonomy. Attribute-CAs operate autonomously: an attribute-CA can certify (and possibly later revoke) an attribute without the need to interact with other Attribute-CAs.

Attribute Dynamism. New attribute-CAs can be introduced without affecting existing users. Each attribute-CA can certify (and revoke) users’ attribute independently. Existing users do not need to contact the attribute-CA to reflect the changes.

Attribute privacy. Collusion of attribute-CAs together cannot find out what attributes a user possesses.

Support for attribute negation. Users can authenticate their non-possession of an attribute without having to contact (and hence be certified by) the respective attribute-CA in advance.

Formal Model

Ideal-World ABACS System. We first describe the ideal-world ABACS system (IAS) that relies on a trusted party $T$ as an intermediator. An IAS consists of a trusted party $T$ through which all transactions are carried out and a set of honest ideal players. In IAS, the ideal players are the users $U$, verifiers $V$ and attribute-CA’s $O$.

Initialisation: The system is initialised when every attribute-CA $O_j$ creates a list $\text{list}_j$. $\text{list}_j$ represents the member list of $O_j$, which is empty at this stage. We say an user is in possession of attribute $\text{attribute}_j$ if and only if he is in the member list of $O_j$.

Ideal communication: All communications are routed through $T$. The sender can request $T$ not to reveal his identity to the recipient if he wishes to be anonymous. The sender may also request to establish a session between him and the recipient. In particular, we assume some kind of authentication is implemented between $T$ and
6.2. Attribute-Based Anonymous Credential Systems

each player. For instance, user $U_i$ cannot contact $T$, claiming himself to be another user $U_j$.

Events in the system: Each transaction between players is an event in the system. Events can be triggered through external processes or may be controlled by an adversary. An external process can trigger some particular event between a particular user and an organisation; or may trigger a set of events; or may cause some probability distribution on the events.

Output of the players: In the end of the system’s lifetime, each user outputs a list of the transactions he participated in and the transaction outcomes.

Transactions: The system supports the following transactions.

- **Nym Protocol**($U_i, O_j$): This protocol is a session between $U_i$ and $O_j$. $U_i$ first contacts $T$, requesting to establish a pseudonym between himself and $O_j$. If $U_i$ and $O_j$ have already established a pseudonym, $T$ simply replies with the previous one. Otherwise, $T$ picks a pseudonym $y_{i,j}$ for them and informs both $U_i$ and $O_j$ the value $y_{i,j}$. $T$ stores $y_{i,j}$ as the pseudonym of user $U_i$ with $O_j$.

- **Certify Attribute**($U_i, O_j$): This protocol is a session between $U_i$ and $O_j$. $U_i$ first contacts $T$ with $y_{i,j}$. If $y_{i,j}$ is not the pseudonym of user $U_i$ with $O_j$, $T$ replies “Fail”. Otherwise $T$ contacts $O_j$ with a certification request for pseudonym $y_{i,j}$. If $O_j$ accepts the request, $T$ notifies $U_i$ that he is now a member of $O_j$.

- **Revoke Attribute**($O_j, U_i$): This algorithm allows $O_j$ to revoke the attribute of $U_i$. $O_j$ removes $y_{i,j}$ from $\text{list}_j$ and notifies $T$ the updated $\text{list}_j$. In fact, this ideal functionality is defined to give the adversary an extra power of influencing an honest $O_j$ to revoke attributes from some of its users.

- **Auth**($U_i, V, \text{policy}$): This protocol is a session between $U_i$ and $V$ regarding a policy $\text{policy}$ which is the conjunction and disjunction of any number of the following, in disjunctive normal form:

  1. $U_i$ possesses attribute $\text{attribute}_j$ (or its negation)
  2. $U_i$ is the owner of pseudonym $y$ with $O_j$ (or its negation).

For each $O_j$ that appears in $\text{policy}$ (concerning attribute $\text{attribute}_j$), $T$ contacts $O_j$ and obtains $\text{list}_j$. $T$ forwards the set of lists $\{\text{list}_j\}$ to $U_i$. Upon receiving the list, $U_i$ could decide if he wishes to continue the protocol and if that is
the case, \( T \) checks from the set of \( \{\text{list}_j\} \) and his list of pseudonym \( \{y_{i,j}\} \) and decides if \( U_i \) fulfills the policy \( \text{policy} \). \( T \) contacts \( V \) with \( \text{policy} \), along with the list \( \{\text{list}_j\} \) and a bit indicating if the underlying user fulfills \( \text{policy} \) or not. \( V \) replies with his response to \( T \). \( T \) forwards the response to \( U_i \).

Intuitively, this ideal-world system captures the security requirements of an ABACS.

**Cryptographic Attribute-Based Anonymous Credential System.** A cryptographic ABACS (CAS) consists of a set of honest cryptographic players. In CAS, these players are the users \( U \), verifiers \( V \) and attribute-CA’s \( O \).

**The ideal-world (resp. real-world) adversary.** The ideal-world (resp. real-world) adversary is a PPT that gets control over the corrupted parties in the ideal world (resp. real-world). He receives a number of honest users and organisations and public information of the system as input. The adversary can also trigger an event as described above.

**Definition of Secure Cryptographic Anonymous Credential System.** A CAS is secure if it conforms to an ideal-world specification. Roughly speaking, CAS is said to be conformed to an ideal-world specification if there exists a simulator \( S \) such that for any real-world adversary \( A \), \( S \), with black-box access to \( A \), acts as an ideal-world adversary so that the output of all honest players in IAS is computationally indistinguishable to the output of all honest players in CAS. \( S \) represents adversary-controlled parties in IAS (ideal-world), and honest parties in CAS (real-world). Informally speaking, \( S \) translates a real-world adversary into an ideal-world adversary. Existence of such simulator implies that for any PPT algorithm (real-world adversary) in CAS, there exists a PPT algorithm (ideal-world adversary) in the ideal-world such that the outputs of the corresponding honest parties in the two worlds are the same. Since IAS is secure (any PPT in IAS cannot do anything that breach the security requirements), CAS is secure. Specifically, we have the following definition.

**Definition 6.1** Let IAS be the ideal-world ABACS. Let CAS be a cryptographic ABACS. For a security parameter \( \lambda \), let the number of players in the system be polynomial in \( \lambda \). By IAS(\( 1^\lambda, E \)) (resp., CAS(\( 1^\lambda, E \))) we denote an ABACS with
security parameter $\lambda$ and event scheduler $E$ for the events that have taken place in the system. Let $A$ be the real-world adversary. Since $E$ schedules the event according to $A$’s wishes, we denote it as $E^A$. By $Z_i(1^\lambda)$ we denote the output of (honest) party $i$ in the system. If \{\{Z_1(1^\lambda), \ldots, Z_\ell(1^\lambda)\}\} is a list of players’ outputs, then we denote these players’ output by \{\{Z_1(1^\lambda), \ldots, Z_\ell(1^\lambda)\}\}^{AS(1^\lambda,E)} when all of them together exist within an ABACS $AS$. $CAS$ is secure if there exists a simulator $S$ such that the following holds, for all PPT $A$ and for all sufficiently large $\lambda$:

1. In $IAS$, $S$ controls the ideal-world players corresponding to the real-world players controlled by $A$.
2. For all event schedulers $E^A$

\[
\{\{Z_i(1^\lambda)\}_{i=1}^\ell, A(1^\lambda)\}^{CAS(1^\lambda,E)} \equiv \{\{Z_i(1^\lambda)\}_{i=1}^\ell, S^A(1^\lambda)\}^{IAS(1^\lambda,E)}
\]

where $S$ is given black-box access to $A$ and $D_1(1^\lambda) \equiv D_2(1^\lambda)$ denotes computational indistinguishability of the two distributions $D_1$ and $D_2$.

Proof of Security of Our ABACS

If $k$ is the maximum number of members for any membership CA, we have the following theorem regarding the security of our ABACS.

**Theorem 6.1 (Security of our ABACS construction)** Under the $k$-SDH assumption in $G_1$ and the DDH assumption in $G_q$, our ABACS is secure in the random oracle model. \hfill \square

*Proof:* In order to prove the security of our ABACS, we must present a simulator $S$ satisfying definition [6.1]. We describe such a simulator and give a sketch of a proof that this simulator satisfies the definition.

The simulator $S$ represents the adversary-controlled parties to the system in the ideal world and represents the honest parties to the adversary in the real world. In the random oracle model, $S$ also gets control over hash functions. Specifically, $S$ is described below.

**System Parameter and Hash Query.** We assume $\hat{e}: G_1 \times G_1 \rightarrow G_2$ such that $|G_1| = |G_2| = p$ and $G_q \subset \mathbb{Z}_p^*$ with $|G_q| = q$ are system-wide parameter. Let $h$ be a generator of $G_q$ and $H: \{0,1\}^* \rightarrow G_q$ be a hash function. Recall that
for each Attribute-CA $O_j$, the OWPRF $h_j$ is defined as $x \mapsto h_j^x$ such that $h_j = H(\text{attribute}_j)$, where $\text{attribute}_j$ is the attribute managed by $O_j$. For each hash query, say $H(\text{attribute}_j)$, $S$ randomly picks $\rho_j \in_R \mathbb{Z}_q$ and returns $h_j = H(\text{attribute}_j) = h_j^{\rho_j}$.

For each honest attribute-CA in the ideal world, $S$ setup the public keys (DMA) for each of them in the real world honestly.

Representing adversary-controlled users in the ideal world and honest players to adversary-controlled users in the real world:

(Nym Protocol) When an adversary-controlled user initiates Nym Protocol by presenting pseudonym $y_{i,j}$ to organisation $O_j$, $S$ searches through the history to locate user $i$. Specifically, $S$ maintains a list of $L_A$ of adversary-controlled user. The list is indexed by value $y_i$ defined as $(y_{i,j})^{1/\rho_j}$. If $i$ is not found, $S$ adds $y_i = (y_{i,j})^{1/\rho_j}$ to $L_A$ and contacts $T$ requesting a pseudonym with $O_j$ on behalf of a new user $U_i$. Otherwise, $S$ contacts $T$ as an existing user $U_i$. In both cases, $T$ chooses a random $\tilde{y}_{i,j}$ as the pseudonym. After that, $S$ appends $y_{i,j}, \tilde{y}_{i,j}$ to the row indexed by $y_i$ as the pseudonym between user $U_i$ and $y_{i,j}$ in the real world, and $\tilde{y}_{i,j}$ is the corresponding pseudonym in the ideal world.

(Certify Attribute) When an adversary-controlled user initiates Certify Attribute by presenting pseudonym $y_{i,j}$ to organisation $O_j$, $S$ searches through the history to locate user $i$. If $y_{i,j}$ is not found, return “fail”. Otherwise, $S$ contacts $T$ on behalf of $U_i$ to $O_j$. If $O_j$ in the ideal world replies with accept, $S$ returns accept to the adversary in the real world. $S$ updates the list $l_{i,j}$ in the real world as well.

(Auth) When an adversary-controlled user is involved in Auth with an honest verifier, $S$ first examines the policy $\text{policy}$ to see if it involves any possession of pseudonym. If yes, it locates user $U_i$ using the list $L_A$ and contacts $T$ for the corresponding Auth functionality on behalf of $U_i$. If it does not involve any pseudonyms, $S$ picks one $U_i$ from the list $L_A$ that satisfies the policy if the authentication in the real world is successful. The simulation fails if the authentication in the real world is successful yet it fails in the ideal world or $S$ cannot find out any users in $L_A$ that satisfies the policy.

Representing adversary-controlled attribute-CAs in the ideal world and honest players to attribute-CAs in the real world:
(Nym Protocol) When \( T \) contacts the adversary-controlled attribute-CA \( O_j \) in the ideal world with a pseudonym \( \tilde{y}_{i,j} \), \( S \) computes \( y_{i,j} = h^x \) for some random \( x \). Note that in our model, \( T \) will not contact \( O_j \) if a pseudonym has previously setup between the underlying user and \( O_j \). \( S \) stores \( \tilde{y}_{i,j} \) and \( y_{i,j} \) as a pair in another list \( L_H \).

(Certify Attribute) Using the list \( L_H \), \( S \) locates the corresponding user and represents it in the real world. If the adversary-controlled attribute-CA \( O_j \) returns \( \text{accept} \), \( S \) notifies \( T \) with \( \text{accept} \).

Representing adversary-controlled Verifiers in the ideal world and honest players to Verifiers in the real world:

(Auth) When \( T \) contacts \( S \) in the ideal world a policy \( \text{policy} \) and whether the underlying user satisfies it, \( S \) acts accordingly. In particular, if \( T \) states that the underlying user satisfies the policy, \( S \) uses the zero-knowledge simulator to simulate the Auth protocol. This requires controlling the random oracle. \( S \) then forwards the response of the adversary-controlled verifier in the real world back to \( T \). On the other hand, if \( T \) states that the underlying user does not satisfy the policy, \( S \) sends some incomplete authentication request to the adversary-controlled verifier and forwards its response back to \( T \).

Successful Simulation. Due to the zero-knowledge nature of our protocols, the simulated Auth protocols are perfect. Note that, however, the pseudonym formed by \( S \) for honest users in the real world is not correct due to the fact that \( S \) simply chooses \( x \) randomly each time. In particular, the same underlying user might use different \( x \) to different attribute-CAs. However, under the DDH assumption, adversary will not notice such difference. The simulation also fails if the authentication (of an adversary-controlled user) in the real world (to an honest verifier) is successful yet it fails in the ideal world. This only happens when the adversary is able to fake a proof during the Auth and happens with negligible probability provided that our DMA is secure. Thus, the simulator \( S \) satisfies definition [6.1] under the DDH assumption and the \( k \)-SDH assumption. \( \square \)
Chapter 7

Concluding Remarks

7.1 Summary of Our Contributions

We briefly reiterate our main contributions.

7.1.1 New Construction of CL Signatures.

In Chapter 3, we constructed two new CL signatures, namely, ESS+ and C-Signature. We also formalised a CL signature, called BBS+, from existing ideas. In particular, ESS+ allows signing group element of a cyclic group equipped with a bilinear map while C-signature supports efficient and concurrently-secure Prove protocol. CL signatures are useful building blocks in cryptographic systems.

7.1.2 New Construction of Accumulators with Extra Features

In Chapter 4, we proposed two new features for accumulators, namely, bound and multiversality. We also classified accumulators into two types according to their underlying mathematical structures, namely, type-qSDH and type-SRSA. We made the observation that an existing type-qSDH accumulator is bounded. From it, we constructed a multiversal accumulator.

7.1.3 New Construction of Electronic Cash with Desirable Properties

In Chapter 5, we presented three new electronic cash schemes. The first scheme, practical compact e-cash, is an optimisation of the existing CHL compact e-cash scheme. We provided a generic construction and provided an instantiation from
BBS+. We also developed another approach in constructing compact e-cash with a different double-spender revocation mechanism. Following this new approach, we constructed a compact e-cash scheme from bounded accumulator and ESS+. Finally, we provided a new way of constructing divisible e-cash by incorporating bounded accumulator into the classical binary tree approach. The resulting scheme, practical divisible e-cash, is based on ESS+ and bounded accumulator.

### 7.1.4 Other Applications of the New Primitives

In Chapter 6, we constructed dynamic $k$-times anonymous authentication scheme from BBS+ and the type-$q$SDH accumulator. We also presented a new primitive, attribute-based anonymous credential system, which is a generalisation of the traditional credential system. The scheme was based on dynamic multiversal accumulator.

### 7.2 Open Problems

The C-signature was constructed from the zero-knowledge proof-of-knowledge of representation of a committed value. It employs a cut-and-choose method and is quite inefficient. It is an interesting problem to construct efficient $\text{CL Sig}$ with concurrently-secure $\text{Issue}$. The same drawback is also present in our construction of DMA.

Besides, the security of ESS+ requires certain attention. One might hope to construct the schemes whose security is based on standard assumptions.


Stefano D’Amiano and Giovanni Di Crescenzo. Methodology for Digital Money Based on General Cryptographic Tools. In Santis [San95], pages 156–170.


