# Advances in Signatures, Encryption, and E-Cash from Bilinear Groups

by

Susan Hohenberger

B.S., The Ohio State University, 2000
S.M., Massachusetts Institute of Technology, 2003

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2006
[June 2006]

© Massachusetts Institute of Technology 2006. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
May 25, 2006

Certified by .....
Ronald L. Rivest
Viterbi Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
NOV 0 2 2006
LIBRARIES

ARCHIVES

# Advances in Signatures, Encryption, and E-Cash from Bilinear Groups

by

Susan Hohenberger

Submitted to the Department of Electrical Engineering and Computer Science
on May 25, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

## Abstract

We present new formal definitions, algorithms, and motivating applications for three natural cryptographic constructions. Our constructions are based on a special type of algebraic group called *bilinear groups*.

1. **Re-Signatures:** We present the first public key signature scheme where a semi-trusted proxy, given special information, can translate Alice's signature on a message into Bob's signature on the same message. The special information, however, allows *nothing* else, i.e., the proxy cannot translate from Bob to Alice, nor can it sign on behalf of either Alice or Bob. We show that a path through a graph can be cheaply authenticated using this scheme, with applications to electronic passports.

2. **Re-Encryption:** We present the first public key cryptosystem where a semi-trusted proxy, given special information, can translate an encryption of a message under Alice's key into an encryption of the same message under Bob's key. Again, the special information allows *nothing* else, i.e. the proxy cannot translate from Bob to Alice, decrypt on behalf of either Alice or Bob, or learn anything else about the message. We apply this scheme to create a new mechanism for secure distributed storage.

3. **Compact E-Cash with Tracing and Bounded-Anonymity:** We present an offline e-cash system where $2^\ell$ coins can be stored in $O(\ell + k)$ bits and withdrawn or spent in $O(\ell + k)$ time, where $k$ is the security parameter. The best previously known schemes required at least one of these complexities to be $O(2^\ell \cdot k)$. In our system, a user's transactions are anonymous and unlinkable, unless she performs a forbidden action, such as double-spending a coin. Performing a forbidden action reveals the identity of the user, and optionally allows to trace *all* of her past transactions. We provide solutions without using a trusted party. We argue why features of our system are likely to be crucial to the adoption of any e-cash system.

Thesis Supervisor: Ronald L. Rivest
Title: Viterbi Professor of Electrical Engineering and Computer Science

# Acknowledgments

It was the best of times.
It was the worst of times.
It was the age of wisdom.
It was the age of foolishness.

---

Charles Dickens, "A Tale of Two Cities"

I gratefully acknowledge aid from the following wonderful sources.

First, I deeply thank Ron Rivest for being a patient, kind, challenging, insightful and supportive advisor for five years. I am honored that Ran Canetti and Silvio Micali joined Ron to serve on my PhD Committee. It was a pleasure to interact with all of the MIT Theory faculty. In particular, I would like to thank Nancy Lynch for teaching me about the art of teaching and Erik Demaine for starting me on my first research publication.

Many past or present MIT students greatly influenced me. Anna Lysyanskaya served as an excellent role model and has generously offered her time, advice, and support. Thanks to the best officemates imaginable: Ben Adida, Seth Gilbert, Nicole Immorlica, David Liben-Nowell, Moses Liskov and Steve Weis. I am also grateful to Adi Akavia, Jay Dryer, Kevin Fu, Jon Herzog, Sonia Jain, Yael Tauman Kalai, Jon Kelner, Matt Lepinski, Jen Mulligan, Claire Monteleoni, Chudi Ndubaku, Rafael Pass, Chris Peikert, Guy Rothblum, abhi shelat, Travis Simpkins, Adam Smith, Nati Srebro, Vinod Vaikuntanathan, Grant Wang and John Wuestneck for keeping me in graduate school. I think.

Thanks for years of candy and support to Be Blackburn and Joanne Talbot Hanley.

Thanks for years of fixing my computer to Matt McKinnon and Greg Shomo.

I benefited from wonderful contacts outside of MIT as well. Special thanks to Jan Camenisch for being a terrific mentor at IBM Zurich Research Lab. I am also enjoyed discussions with Nenita Angeles, Giuseppe Ateniese, Matt Green, David Molnar, Tyler Neylon and Alon Rosen. Additionally, I gratefully acknowledge The Ohio State University CSE Department, especially my mentors Bruce Weide and Paolo Bucci, for ten years of encouragement.

I was generously financially supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship and a Google Anita Borg Memorial Scholarship.

My family and friends gave large amounts of unconditional aid, especially Ray and Beth Hohenberger, Megan and Todd Nussbaum, Barbara and Andy Lorson, Dick and Dee Zarle, Gloria Hohenberger and Joe Carter. Finally, thank you, God.

# Patent Information

Two contributions of this thesis have patents pending. They are:

1. Description: the proxy re-encryption schemes $\Sigma_1, \Sigma_2$, and $\Sigma_{temp}$ in Chapter 5.3.3
   Inventors: Giuseppe Ateniese, Kevin Fu, Matthew Green, Susan Hohenberger
   Held by: Massachusetts Institute of Technology and Johns Hopkins University (equally)

2. Description: the techniques for detecting and preventing money laundering in electronic cash without a trusted third party in Chapter 7.3
   Inventors: Jan Camenisch, Susan Hohenberger
   Held by: IBM

# Contents


**1 Introduction** **17**
1.1 Bilinear Groups . . . 18
1.2 Proxy Re-Cryptography . . . 21
1.2.1 Proxy Re-Signatures . . . 23
1.2.2 Proxy Re-Encryption . . . 25
1.3 Electronic Cash . . . 26

**2 Bilinear Groups** **29**
2.1 Elliptic Curves . . . 29
2.2 Isomorphisms . . . 31
2.3 The DDH-CDH Gap . . . 32
2.4 Security and Efficiency . . . 32
2.4.1 Space Requirements for Elliptic Curve Keys and Elements . . . 33
2.4.2 Speed of Elliptic Curve Operations . . . 34

**3 Complexity Assumptions** **35**
3.1 Discussion of Assumptions . . . 37

**4 Proxy Re-Signatures** **39**
4.1 Introduction . . . 39
4.1.1 Related Work . . . 42
4.2 Definitions . . . 43
4.2.1 Re-Signatures and Program Obfuscation . . . 47
4.3 Proxy Re-Signature Schemes . . . 48
4.3.1 Additional Properties We Need and Want . . . 48
4.3.2 Remarks on the BBS Scheme . . . 50
4.3.3 $\Omega_{bi}$: A Multi-Use Bidirectional Scheme . . . 51
4.3.4 $\Omega_{uni}$ and $\Omega^*_{uni}$: Single-Use Unidirectional Schemes . . . 55
4.4 Applications . . . 61
4.4.1 Exploring BBS Key Management . . . 62
4.4.2 New Applications . . . 63
4.5 Contributions . . . 64
4.6 Open Problems . . . 65

**5 Proxy Re-Encryption** **67**
5.1 Introduction . . . 67
5.1.1 Related Work . . . 70
5.2 Definitions . . . 71
5.2.1 Proxy Re-Encryption and Program Obfuscation . . . 74
5.3 Improved Proxy Re-Encryption Schemes . . . 75
5.3.1 Additional Properties We Need and Want . . . 75
5.3.2 $\Sigma_0$: Paillier-Based Proxy Encryption (Not Re-Encryption) . . . 78
5.3.3 $\Sigma_1$ and $\Sigma_2$: New Unidirectional Re-Encryption Schemes . . . 79
5.3.4 $\Sigma_{temp}$: Temporary Unidirectional Proxy Re-Encryption . . . 85
5.4 Applications . . . 89
5.4.1 Secure Distributed Storage . . . 89
5.5 Contributions . . . 92
5.6 Open Problems . . . 92

**6 Compact E-Cash** **95**
6.1 Introduction . . . 95
6.2 Definition of Security . . . 100
6.2.1 Formal Definitions . . . 103
6.3 Preliminaries . . . 109
6.3.1 Known Discrete-Logarithm-Based, Zero-Knowledge Proofs . . . 109
6.3.2 Pseudorandom Functions . . . 110
6.3.3 Pedersen Commitments . . . 110
6.3.4 CL Signatures . . . 110
6.3.5 Verifiable Encryption . . . 111
6.3.6 Bilinear El Gamal Encryption . . . 111
6.4 System One: Compact E-Cash . . . 112
6.4.1 System Zero: A Simpler E-Cash System . . . 118
6.5 System Two: Compact E-Cash with Tracing . . . 118
6.5.1 Solution #1: Use DY PRF, make XDH Assumption. . . . 119
6.5.2 A New PRF for DDH-Easy Groups . . . 119
6.5.3 Solution #2: Use CHL PRF, make Sum-Free DDH Assumption. . . . 120
6.6 On Removing Random Oracles . . . 126
6.7 Contributions . . . 127
6.8 Open Problems . . . 127

**7 Compact E-Cash in the Bounded-Anonymity Model** **129**
7.1 Introduction . . . 129
7.2 Definition of Security . . . 131
7.3 Compact E-Cash in the Bounded-Anonymity Model . . . 133
7.3.1 Roadmap to Understanding the Construction . . . 133
7.3.2 Our Construction . . . 134
7.4 Scaling Back the Punishment for System Violators . . . 141

7.5 On Removing Random Oracles . . . . . 141
7.6 Contributions . . . . . 141
7.7 Open Problems . . . . . 142

**A Full Protocols to Spend Coins** **143**
A.1 System One: Full Protocol . . . . . 143
A.2 System Two: Partial Protocol . . . . . 145

# List of Figures


2-1 The elliptic curve defined by $y^2 = x^3 - x$. . . . . 30

4-1 A high-level view of (unidirectional) proxy re-signatures. Here Bob is the delegator and Alice is the delegatee. The proxy obtains a re-signature key $rk_{A \to B}$ from Bob, and can then translate signatures from Alice, denoted $\sigma_A(m)$, into ones from Bob, denoted $\sigma_B(m)$. . . . . 43

4-2 Unidirectional chains with multi-use proxy re-signature. Each node (e.g., $C$) holds only a proxy re-signature key (e.g., $rk_{C \to D}$) and *not* a signing key (e.g., $sk_C$). Thus, an adversary cannot inject a signed message into the chain by corrupting intermediate nodes. . . . . 63

5-1 A high-level view of (unidirectional) proxy re-encryption. Here Alice is the delegator and Bob is the delegatee. The proxy obtains a re-encryption key $rk_{A \to B}$ from Alice, and can then translate ciphertexts for Alice, denoted $\mathsf{Enc}_A(m)$, into ciphertexts for Bob, denoted $\mathsf{Enc}_B(m)$. . . . . 71

5-2 Operation of the proxy re-encryption file system. The user's client machine fetches encrypted files from the block store. Each encrypted file has two components: (1) the file content encrypted under a symmetric key, and (2) the symmetric key encrypted under a master public key, called the *lockbox*. The client then transmits the lockbox to the access control server for re-encryption under the user's public key. If the access control server possesses the necessary re-encryption key, it re-encrypts the lockbox and returns the new ciphertext. The client can decrypt the lockbox using the user's secret key, recover the symmetric key, and then decrypt the file content. . . . . 90

6-1 A summary of the System One Spend protocol. . . . . 113

6-2 A summary of the System Two (Solution #2) Spend protocol. . . . . 122

# List of Tables


1.1 We present the known proxy re-cryptography results. As discussed subsequently, the BBS re-signature scheme has some limitations. . . . . 23

2.1 NIST estimated key sizes for an equivalent level of security. . . . . 34

2.2 Due to the shorter key sizes, standard arithmetic operations in elliptic curve groups run more efficiently than comparable operations in Diffie-Hellman groups. 34

4.1 We compare the properties of several proxy re-signature schemes discussed in this work. The symbol * denotes that the property can be provably partially obtained or obtained by adding additional overhead. For ♯. we provide some insight on how this might be achieved. . . . . 49

5.1 We compare known proxy encryption (DI, $\Sigma_0$) and re-encryption (BBS, $\Sigma_2$) schemes based on the advantages described above. No schemes achieve properties five and eleven. We refer to the unidirectional schemes of Dodis-Ivan. The symbol $*$ indicates master secret key only. For †, the property is possible to achieve by doubling the overhead, as in scheme $\Sigma_{temp}$. . . . . 77

5.2 Average operation times for 100 runs of the $\Sigma_2$ proxy re-encryption scheme on our client and server. All operations refer to re-encryptable second-level ($\mathsf{Enc}_2$) ciphertexts. . . . . 92

# Chapter 1

# Introduction

Cryptography is an ancient art [129], historically used for securely communicating. In the days of Julius Caesar (100 BC), the Roman army scrambled the letters in their communications, according to a secret code, to prevent enemies from intercepting and reading the messages. During World War II, the Allies' successful interception and break of the German secret codes is credited with helping win the war. Today, cryptography has many uses beyond military ones. Indeed, it is employed by millions of users daily to purchase goods over the Internet. Cryptography even has applications beyond secure communication, such as electronic cash.

There are two main categories of cryptographic communication protocols: *symmetric* (also called *private key*) and *asymmetric* (also called *public key*). Before 1976, it was generally believed that for two parties, say Alice and Bob, to securely communicate it was necessary that Alice and Bob share some common secret. Such protocols are called symmetric, because both parties must know the same secret. In 1976, Diffie and Hellman [88] authored a seminal publication called "New Directions in Cryptography," in which they established the foundation of asymmetric protocols. Diffie and Hellman described a method by which Alice and Bob could agree on a common secret, even if an eavesdropper overheard all the messages passing between them! Such a protocol was dubbed a *key exchange* protocol.

In 1978, Rivest, Shamir and Adelman [165] proposed the first asymmetric encryption algorithm, named RSA after its authors. The conceptual breakthrough in RSA is that Alice can privately send Bob a message *without* the two of them ever agreeing on a common secret! Instead, to receive private messages, Bob first generates two values called a *public key* and a *secret key*. He then broadcasts his public key to everyone, e.g., posts it on the Internet, and keeps the secret key for himself. Now, using Bob's public key, Alice can encode a message in such a way that *only* Bob with his secret key will be able to read it.

Both the Diffie-Hellman and RSA results came from key observations about the properties of certain algebraic groups. A mathematical group has three components: a set of elements, a group operator (such as multiplication), and an identity element (such as one, where for any element $x$ in the group, $x$ multiplied by one is $x$.). In the case of Diffie-Hellman, the group elements are the set of integers modulo a large prime. In the case of RSA, the group elements are the set of integers modulo the product of two large primes. (In both cases,

the group operator is multiplication and the identity is one.) The key idea is that different groups offer different mathematical properties which, with human ingenuity, can be leveraged to construct new cryptographic functionalities.

In this thesis, we advance the state-of-the-art for three natural cryptographic functionalities: re-signatures, re-encryption, and electronic cash. Our constructions are based on an algebraic group called *bilinear groups*, where the group elements are (typically) the set of points on an elliptic curve. For each of the functionalities we consider, currently no constructions, comparable in terms of either security or efficiency, are known beyond the ones we present using bilinear groups. An interesting open question is whether or not these functionalities can be achieved using other algebraic groups.

For the remainder of this introduction, let us first discuss bilinear groups in more detail. We will then cover the definition, the technical advance, and the practical application of each of our works in re-signatures, re-encryption, and electronic cash.

## 1.1 Bilinear Groups

Bilinear groups are a set of three abstract algebraic groups, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, together with a deterministic function $\mathsf{e}$, called a *bilinear map*, that takes as input one element from $\mathbb{G}_1$ and one element from $\mathbb{G}_2$ and outputs an element in $\mathbb{G}_T$. Suppose all three groups have order $Q$, element $g_1$ generates group $\mathbb{G}_1$, and element $g_2$ generates group $\mathbb{G}_2$. Then, one special property called *bilinearity* of the map $\mathsf{e}$ is that for all $a, b < Q$, we have that $\mathsf{e}(g_1^a, g_2^b) = \mathsf{e}(g_1, g_2)^{ab}$. This new element, $\mathsf{e}(g_1, g_2)^{ab}$, is in group $\mathbb{G}_T$. The key observation is what happens to the exponents, $a$ and $b$, during the mapping: they are multiplied. The group $\mathbb{G}_T$ is distinct from both $\mathbb{G}_1$ or $\mathbb{G}_2$; thus the output of the mapping cannot be fed back into the map $\mathsf{e}$ as input. At first glance, this may seem like a simple algebraic trick between three groups, however, it has proven to be a powerful cryptographic building block.

We said above that bilinear groups are *abstract* groups, meaning that any three algebraic groups, whatever their set of group elements, etc., which satisfy the abstract definition can be called bilinear. The best known realization of bilinear groups come from elliptic curve groups, where the group elements are the set of points on an elliptic curve (plus one additional point). That is the set of $(x, y)$ solutions to the equation $y^2 = x^3 + ax + b$, plus one additional point at infinity. Here the group operation is defined geometrically. (We will discuss more technical details of bilinear groups and elliptic curves in more detail in Chapter 2.)

Elliptic curve groups were introduced into cryptography long before the potential of their bilinear properties was recognized. In 1985, Miller first mentioned using elliptic curve groups in cryptography [146] to implement the Diffie-Hellman key exchange protocol with the objective of obtaining better security and efficiency. To see this, consider the discrete logarithm (DL) problem for a group $\mathbb{G}$ of order $Q$ with generator $g$, which is: given $(g, g^a)$, compute $a < Q$. The DL problem is believed to be *hard* in many algebraic groups, meaning that any adversary that solves the problem must run exponentially in the bit length of $Q$.

The security of the Diffie-Hellman key exchange protocol rests in part on the hardness of the DL problem in the group used to implement it. Although, this protocol was (and still

is) believed to be secure when implemented using the groups originally proposed by Diffie and Hellman, some attacks against the DL problem in these groups run super-polynomially, but sub-exponentially in the bit length of $Q$. Thus, larger groups (i.e., more bits to comprise a larger $Q$) must be used to guarantee a desired level of security with respect to the best known attacks.

Miller [146] argued that because these DL attacks were not (and still are not) generally effective against elliptic curve groups, an elliptic curve implementation could offer a better security guarantee while costing fewer bits per group element. A similar proposal was made in 1987 by Koblitz [135]. By 1998, an elliptic curve signature scheme, ECDSA, was adopted as a US standard [4]. Indeed, elliptic curve groups appear, so far, to have withstood the test of time against more effective DL attacks against them. In Chapter 2, we discuss in more detail the particular time and space savings that they offer compared to other common cryptographic groups.

Up to this point, we have only discussed the introduction of elliptic curve groups in cryptography. The entrance of bilinear groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ joined by a mapping function $\mathsf{e}$, however, appears to be more revolutionary. Interestingly, bilinear groups have a similar history to the advent of public key cryptography in general. In 2000, Joux [126] sparked interest in the bilinear properties of elliptic curve groups by using them to implement an optimized version of the Diffie-Hellman key exchange protocol, where three people could agree on a key instead of only two. However, it was the 2001 identity-based encryption (IBE) scheme of Boneh and Franklin [30, 29] that triggered an explosion of bilinear schemes in cryptography.

The reason was largely that constructing a polynomial-time IBE scheme had been an open problem since proposed by Shamir in 1984 [172]. The IBE scheme of Boneh and Franklin was not only polynomial-time, but competitive with popular encryption schemes, such as El Gamal. (Shortly after the Boneh and Franklin result was published, Cocks [82] showed how to build a polynomial-time IBE scheme based on quadratic residues, i.e., no bilinear groups needed. While this was a theoretical breakthrough, the Cocks scheme is not efficient enough to be used in practice.)

Thus, Boneh and Franklin re-energized the cryptography community with the hope of realizing desirable protocols that had been open problems for years and the possibility of realizing previously unthought of functionalities. Since the Boneh and Franklin result [30] appeared in 2001, over 300 academic publications covering a wide range of applications of bilinear groups in cryptography emerged. It would be a daunting task indeed to survey all of these results.

To add perspective to the contributions of this thesis, let us sample *some* of the recent exciting applications of bilinear groups (in chronological order of publication within the categories). Results of the author are in italics. Random oracles are a heuristic often used in proofs of security, which are known to be unsound theoretically [66, 115].

**Signatures:**

1. Short signatures [35, 34]

2. Unique signatures [140]
3. Aggregate and verifiably encrypted signatures [31]
4. Efficient signatures without random oracles [25, 54]
5. *Proxy re-signatures* [10], Chapter 4
6. Logarithmic-size group signatures without random oracles [39]
7. Aggregate and multi-signatures without random oracles [138]
8. *Constant-size group signatures without random oracles* [6]

**Encryption:**

1. Identity-based encryption (IBE) [30, 29]
2. Forward-secure public-key encryption [67]
3. Encryption with keyword search [28]
4. *Unidirectional proxy re-encryption* [8, 9], Chapter 5
5. IBE without random oracles [180]
6. Fuzzy IBE [167]
7. Traitor tracing schemes [69]
8. (Almost) doubly homomorphic encryption [33]
9. Broadcast encryption [32]

**Other Algorithms and Protocols:**

1. Pseudorandom and verifiable random functions [140, 92, 95]
2. *Electronic cash* [49, 50], Chapters 6 and 7
3. Perfect non-interactive zero knowledge for NP [120]

Boneh and Silverberg [36] also explored applications of *multilinear groups*. For instance, suppose there are four groups $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_3$, $\mathbb{G}_T$ and a mapping $\mathsf{e}$ that takes one element from each of the first three groups and maps it to an element of the fourth group such that the exponents multiply. Boneh and Silverberg amply motivate the utility of multilinear groups in cryptography; unfortunately, no such candidate implementations are known. Thus, we will restrict our focus to bilinear groups only.

Our above sample of applications of bilinear groups in cryptography utilize these groups in four major ways: (1) realize a new functionality for which no non-bilinear constructions, however inefficient, are known; (2) achieve better time and/or space efficiency; (3) remove trusted third parties; and (4) remove random oracles.

The first application is the most exciting one! Indeed, our re-signature and re-encryption constructions in Chapters 4 and 5 fall into this first category. The second two applications are also of great practical significance, and our advances in electronic cash in Chapters 6 and 7 will highlight how bilinear maps can be used to exponentially improve efficiency and remove (for the first time) trusted third parties. Recall that since one of the goals of cryptography is to eliminate the *need* to trust others, removing "trusted" parties from systems is a fundamental goal.

The final application of bilinear groups, to remove random oracles, is one of its most prevalent uses, including a work of the author [6]. The current trend is to replace random

oracles by bilinear groups *and* a new (typically untested) complexity assumption. The best current method for providing some confidence in these new assumptions is to show that they hold in the generic group model of Shoup [152, 173]. Oddly enough this generic group model is subject to some of the same theoretical flaws as random oracles [87]! There is certainly value in this final application of bilinear groups, but a further discussion on this topic is both warranted and beyond the scope of this thesis.

Now that we have a better understanding of bilinear groups and the methods in which they are often applied, let us overview the main contributions of this thesis. We present new advances in the areas of re-signatures, re-encryption, and electronic cash.

## 1.2 Proxy Re-Cryptography

We begin with some basic definitions:

**Definition 1.1 (Signature Scheme).** *A* signature scheme *is a tuple of algorithms* (KeyGen, Sign, Verify), *where on input the security parameter* $1^k$, *the* KeyGen *algorithm outputs a public and secret key pair* $(pk, sk)$. *On input the secret key* $sk$ *and a message* $m$, *the* Sign *algorithm outputs a signature* $\sigma$. *On input the public key* $pk$, *the message* $m$, *and a purported signature* $\sigma$, *the* Verify *algorithm accepts if* $\sigma$ *is a valid signature and rejects otherwise.*

A signature scheme is the digital equivalent of physically signing a paper document with a pen. The idea is that Alice (a user) generates a key pair for herself, publicizing $pk$ and keeping $sk$ secret. Then, only Alice can sign messages that will verify against her public key $pk$. Goldwasser, Micali, and Rivest provided the standard definition of security for digital signatures [118]. The intuition is that even if an adversary is allowed to ask Alice to sign messages of *his* choice, he is still unable to forge her signature on a new message.

Signatures are a fundamental building block of secure communication. For example, a stockbroker needs to know that an order to "sell 10,000 shares" really came from his client. Of course, the client and stockbroker also want to keep the contents of their messages private, which brings us to the second fundamental building block: encryption.

**Definition 1.2 (Encryption Scheme).** *An* encryption scheme *is a tuple of algorithms* (KeyGen, Enc, Dec), *where on input the security parameter* $1^k$, *the* KeyGen *algorithm outputs a public and secret key pair* $(pk, sk)$. *On input the public key* $pk$ *and a message* $m$, *the* Enc *algorithm outputs a ciphertext* $\pi$. *On input the secret key* $sk$ *and a valid ciphertext* $\pi$, *the* Dec *algorithm outputs the message* $m$.

An encryption scheme is the digital equivalent of Bob locking a paper document in one of Alice's lead-lined safes, which only she can open, and then putting the safe on a public train headed in Alice's direction. The idea is that Alice generates a key pair for herself, publicizing $pk$ and keeping $sk$ secret. Then, anyone can encrypt a message for Alice using her public key $pk$ in such a way that only Alice (with her secret key $sk$) can recover the message. Goldwasser and Micali provided the first standard definition of security for encryption [117], with stronger definitions due to Dolev, Dwork, Naor and Yung [150, 96]. The intuition is

that an adversary cannot distinguish encryptions of two messages, even if he gets to choose those messages himself!

**Proxy Cryptography.** Signatures and encryption are two of the most widely used cryptographic tools, and depending on how they are used in practice, it is often useful to consider variations on the basic schemes. In 1996, Mambo, Usuda, and Okamoto [142] introduced the concept of *proxy signatures*. In a proxy signature, Alice (a user) delegates her signing rights to Bob (another user) but only if a special party called, the proxy, cooperates. Thus, Bob and the proxy can jointly sign arbitrary messages on Alice's behalf– if they work together –but neither one can sign for Alice on his own. This is usually, but not necessarily, accomplished using threshold techniques; that is, by dividing Alice's secret key $sk$ into two shares which are distributed to Bob and the proxy (each gets only one share). A signature from Alice on a message is generated by combining two partial signatures on the same message computed by Bob and the proxy under their own shares, respectively.

Thus, if Alice needs to go on vacation, she can have her friend Bob and her company's proxy sign messages on her behalf while she is away. Many proxy signature schemes are known, based on Schnorr, El Gamal, and RSA signatures, with the best recent work due to Dodis and Ivan [92].

In 1997, Mambo and Okamoto [141] also explored the concept of *proxy encryption*. In a proxy encryption scheme, Alice (a user) delegates her decryption rights to Bob (another user) but only if the proxy cooperates. Thus, Bob and the proxy can jointly decrypt ciphertexts intended for Alice– if they work together –but neither one can read her messages on his own. This is usually accomplished using the same ideas as proxy signatures, where Alice divides her secret key $sk$ into two shares which are distributed to Bob and the proxy (each gets only one share). A ciphertext intended for Alice is first partially decrypted by the proxy and then fully decrypted by Bob.

Again, while Alice is on vacation, she can have her friend Bob read her encrypted email, so long as the proxy cooperates. (Imagine that Alice gives the proxy instructions only to help Bob decrypt messages tagged "urgent.") Many proxy encryption schemes are known, based on El Gamal, RSA, and IBE techniques, with the best recent work again due to Dodis and Ivan [92].

**Proxy Re-Cryptography: A Strict Subset of Proxy Cryptography.** In 1997 and 1998, Blaze, Bleumer, and Strauss [22, 21] (BBS) proposed proxy signature and encryption schemes that have a special property: namely that the proxy's role is to *translate* between two user's keys.

This is best understood by example. Suppose Alice and Bob are two users with two independent encryption key pairs $(pk_A, sk_A)$ and $(pk_B, sk_B)$, respectively. BBS proposed a *proxy re-encryption* scheme where the proxy is given special information that allows to translate a ciphertext encrypted under Alice's public key, $\mathsf{Enc}(pk_A, m)$, into a ciphertext that can be decrypted using Bob's secret key, $\mathsf{Enc}(pk_B, m)$, containing the same message. The proxy cannot, however, decrypt and read the messages of either party.

| | Bidirectional | Unidirectional |
|---|---|---|
| **Re-Signatures** | AH [10], Chapter 4<br>multi-use | AH [10], Chapter 4<br>single-use |
| **Re-Encryption** | BBS [22, 21]<br>multi-use | AFGH [8, 9], Chapter 5<br>single-use |

Table 1.1: We present the known proxy re-cryptography results. As discussed subsequently, the BBS re-signature scheme has some limitations.

Likewise, in a *proxy re-signature*, the proxy is given special information that allows to translate a perfectly-valid and publicly-verifiable signature from Alice on a certain message, $\mathsf{Sign}(sk_A, m)$, into one from Bob on the same message, $\mathsf{Sign}(sk_B, m)$. These signatures can coexist and both can be publicly verified as being two signatures from two distinct people on the same message. Notice that, in the case of re-signatures it is Bob who is delegating signing rights to Alice, whereas in re-encryption it was Alice delegating to Bob.

Any proxy re-signature can be used to build a proxy signature, but the reverse is not necessarily true. For instance, it is possible to build a proxy signature based on RSA (as Dodis and Ivan do [92], by splitting Alice's secret $d$ into $d_1$ and $d_2$ such that $d = d_1 + d_2$) but it is *not* possible to have a proxy re-signature scheme that translates between two publicly-verifiable RSA signatures sharing the same modulus (because of the *common modulus* problem/attack). The same reasoning shows that any proxy re-encryption scheme can be used to build a proxy encryption scheme, but the reverse is not necessarily true. Thus, proxy re-cryptography is a strict subset of proxy cryptography.

Despite the ample work on proxy cryptography, the only schemes, prior to our work, that had this special translation property were those proposed by BBS [22, 21]. BBS proposed one re-encryption and one re-signature scheme, both of which were *bidirectional*, which means that the proxy can translate from Alice to Bob and from Bob to Alice. BBS left as an open problem how to realize schemes with the elusive *unidirectional* property, where the proxy's information only allows it to translate in one direction.

Thus, building unidirectional schemes remained open for seven years, and with the aid of bilinear groups, this is where our contributions begin. We summarize the known constructions in Table 1.1. One other important feature to note is whether a re-signed message or a re-encrypted ciphertext can be re-signed or re-encrypted again. For example, can the proxy translate from Alice to Bob and then from Bob to Carol and so on. We call such schemes where this is possible *multi-use*, and otherwise refer to them as *single-use*.

### 1.2.1 Proxy Re-Signatures

In the area of proxy re-signatures, Ateniese and Hohenberger [10] made five distinct contributions which we will expand on in Chapter 4. Roughly, these are:

1. **Expose Weakness in BBS Scheme.** In the BBS re-signature scheme [22, 21], anyone

given an original signature and its re-signed value can compute the proxy key, the special information that allows the proxy to translate. (We show this attack in Section 4.3.2.) Moreover, from a proxy key between Alice and Bob, in the BBS scheme, Bob can compute Alice's secret key, and vice versa. Thus, after only *one* re-signature, everyone is a proxy, and Alice and Bob have exchanged secret keys!

2. **Provide Formal Definitions and Security Model.** We formalize a security model for proxy re-signatures that incorporates the desirable property of having the proxy key safely stored at the proxy. In our unidirectional scheme in Section 4.3.4, we realize a strong notion of security: Suppose Alice delegates to Bob, via a proxy, the ability to change his signatures into hers. As one might expect, even when the proxy and Alice collude, they cannot recover any information about Bob except his public key. More surprisingly, we show that even when the proxy and Bob collude, they can only recover a weak version of Alice's secret key.

3. **Provide Two Proxy Re-Signature Constructions.** We present two different constructions based on bilinear groups. The first is a *multi-use, bidirectional* proxy re-signature, which is very simple and efficient. Unlike the bidirectional BBS scheme, here the proxy can keep his proxy keys private. The security of the scheme is based on the Computational Diffie-Hellman (CDH) assumption, i.e., given $(g, g^x, g^y)$, it is hard to compute $g^{xy}$, in the random oracle model.

   The second scheme is *single-use*, but *unidirectional.* This is the first construction of a unidirectional scheme since it was proposed as an open problem by BBS seven years ago [21]. (In BBS, they refer to such schemes as *asymmetric.*) In this scheme, the proxy key is leaked, just as in BBS. However, in the unidirectional setting, some important security features can still be salvaged, and we provide natural applications for this scheme. We also discuss how the proxy key can be kept private at an additional cost. The security of this scheme is based on the CDH and 2-Discrete Logarithm (2-DL), i.e., given $(g, g^x, g^{x^2})$, it is hard to compute $x$, assumptions in the random oracle model.

4. **Apply to Create Ultra-Efficient Proof of Flow.** We propose exciting new applications of proxy re-signatures in Section 4.4. One application of the bidirectional scheme is a proof of flow through a set of nodes. Imagine that the government wants people to visit three checkpoints $A$, $B$, and $C$, in that order, before obtaining a driver's license. The government could generate keypairs for each node, and then provide $A$ with a signing key, but only provide the subsequent nodes with a re-signing key. Then, node $C$ only has to verify a *single* signature under a *single* public key from $B$ even if several nodes (not just $A$) precede $B$ in the path.

   Our technique requires some pre-configuration but provides several benefits: (1) all (but the first) nodes do not store any signing key so that, even if a node is compromised, no new messages can be injected into the flow, (2) only a single signature must be stored at any one time, i.e., there is no need to accumulate signatures and public keys while moving through the checkpoints, (3) no information about prior nodes is collected so

that, if applications require it, the actual flow traversed by a message could remain *private*, i.e., the last node of the path knows that the message followed a legitimate path through the checkpoints but does not know which one (this property is optional).

We also discuss applications to certificate sharing and a weak form of group signatures.

5. **Provide New Signature Scheme with Two Trapdoors.** Finally, we note that our unidirectional scheme introduces a new signature algorithm that may be of independent interest, because it allows the signer to use *two secrets* (strong and weak) for a *single* public key (more details in Section 4.3.4).

### 1.2.2 Proxy Re-Encryption

In the area of proxy re-encryption, Ateniese, Fu, Green, and Hohenberger [8, 9] made four distinct contributions which we will expand on in Chapter 5. Roughly, these are:

1. **Formal Definitions and Security Model.** We formalize the security model for unidirectional proxy re-encryption, where the proxy key remains private. We require only chosen-plaintext (CPA) security in our definitions [117]. Obviously, CCA2-security [150, 96] would be very interesting, but we do not know of any scheme that meets such a definition. In the CPA context, however, we are able to make strong security guarantees. In particular, suppose Alice delegates to Bob, via a proxy, the ability to decrypt her ciphertexts. Even when the proxy and Alice collude, they cannot recover any information about Bob except his public key. More surprisingly, we show that even when the proxy and Bob collude, they can only recover a weak version of Alice's secret key.

2. **Provide Three Proxy Re-Encryption Constructions.** We present three *single-use, unidirectional* proxy re-encryption schemes where the proxy key is kept private, based on different complexity assumptions. The security of our first two schemes is implied by the (bilinear) $y$-DDHI assumption [25, 95], i.e., given $(g, g^x, g^{x^2}, \ldots, g^{x^y}, Q)$, decide if $Q = \mathsf{e}(g,g)^{1/x}$. We can and do, however, prove their security under weaker assumptions.

   Our third scheme is designed to allow for temporary delegations. That is, at the beginning of each time period, a trusted party broadcasts a value that invalidates all re-encryption keys for the previous time period. This scheme is less efficient than the previous two, but its security is based on the mild DBDH assumption, i.e., given $(g, g^a, g^b, g^c, Q)$, decide if $Q = \mathsf{e}(g,g)^{abc}$.

   Unlike re-signatures, here we will not require random oracles in our proofs of security.

3. **Apply Scheme to Secure Distributed Storage.** In many secure distributed file systems, such as Cepheus [102], all of the files are stored encrypted under a single master key. When a user requests a file, the access control server decrypts the file with the master secret key and then encrypts the file under the user's public key. This requires complete trust in the control server, since it sees the contents of every requested file.

Unfortunately, this makes the control server a prime target of hackers. Indeed, as we'll discuss in Chapter 5, in the recent break of Apple's iTunes DRM, hackers managed to steal the DRM-less versions of the songs during this decrypt-and-encrypt step [174].

We show that by using our proxy re-encryption technology, the access server need never see the content of *any* of the files that it processes. This offers a large gain in security for relatively little performance cost. We provide the first empirical performance measurements of proxy re-encryption, and show that our algorithms take roughly double the time of the decrypt-and-encrypt approach.

4. **Provide New Encryption Scheme with Two Trapdoors.** Finally, the base of our unidirectional scheme is a cryptosystem that may be of independent interest, because it allows to decrypt using *two secrets* (strong and weak) for a *single* public key. We use this observation as a critical part of our electronic cash system in Chapter 6 (specifically, enabling the efficient tracing feature).

## 1.3 Electronic Cash

Now, let us turn our attention to another exciting cryptographic system. We leverage the power of bilinear groups build an efficient electronic cash system without trusted third parties.

The concept of *electronic cash* was proposed by Chaum [73, 74] in 1982, and has been extensively studied since [77, 100, 79, 40, 57, 41, 175, 99, 178, 19]. The main idea is that, even though the same party (a bank) is responsible for giving out electronic coins, and for later accepting them for deposit, the withdrawal and the spending protocols are designed in such a way that it is impossible to identify when a particular coin was spent. I.e., the withdrawal protocol does not reveal any information to the bank that would later enable it to trace how a coin was spent.

As a coin is represented by data, and it is easy to duplicate data, an electronic cash scheme requires a mechanism that prevents a user from spending the same coin twice (double-spending). There are two scenarios. In the *on-line* scenario [74, 75, 76], the bank is on-line in each transaction to ensure that no coin is spent twice, and each merchant must consult the bank before accepting a payment. In the *off-line* [77] scenario, the merchant accepts a payment autonomously, and later submits the payment to the bank; the merchant is guaranteed that such a payment will be either honored by the bank, or will lead to the identification (and therefore punishment) of the double-spender.

In the area of off-line electronic cash (e-cash), Camenisch, Hohenberger, and Lysyanskaya [49, 50] made four distinct contributions which we will expand on in Chapters 6 and 7. Some, but not all, require bilinear groups as we will discuss. Roughly, these are:

1. **Formal Definitions for Electronic Cash.** In Section 6.2, we formalize the security model for electronic cash, building on the intuition of many previous works. We capture the intuition that, unless a user misbehaves, her transactions remain anonymous

and unlinkable, while at the same time the bank will never have to accept more coins for deposit than were withdrew. Our definitions are property-based, however, they would imply Universal Composability (UC) security [61, 63, 64] whenever the extractors and simulators are constructed appropriately. It remains open to model security for electronic cash in the UC framework, however, we provide some intuition in this direction.

2. **Compact E-Cash System.** We present an efficient off-line e-cash scheme where a user can withdraw a wallet containing $2^\ell$ coins each of which she can spend anonymously and unlinkably. The complexity of the withdrawal and spend operations is $O(\ell+k)$ and the user's wallet can be stored using $O(\ell + k)$ bits, where $k$ is a security parameter. The best previously known schemes require at least one of these complexities to be $O(2^\ell \cdot k)$! In fact, compared to previous e-cash schemes, our whole wallet of $2^\ell$ coins has about the same size as *one* coin in these schemes. If a user double-spends a coin, the bank can recover her identity. Our scheme also offers exculpability of users, that is, the bank can prove to third parties that a user has double-spent. Our scheme is secure under the Strong RSA and the $y$-DDHI assumptions in the random oracle model. In some cases, e.g., the bank and the merchant are the same entity, our system is secure in the standard model as we discuss in Section 6.6. We point out that this system does not *require* bilinear groups, although they can be employed to achieve a constant factor optimization in the system complexities.

3. **Add Tracing without a Trusted Third Party.** We then extend our scheme to our second system, the first e-cash system that provides traceable coins without a trusted third party. That is, once a user has double spent one of the $2^\ell$ coins in her wallet, *all* her spendings of from *all* of her wallets can be efficiently traced. This strikes a balance between the needs of law enforcement and private citizens. If a user abuses her anonymity by double-spending a coin, law enforcement officers can see what else she has bought. At the same time, users who do not double-spend remain anonymous–for the first time –without fearing that a "trusted" third party is able to track them.

   We present two alternate constructions, both requiring bilinear groups. One construction shares the same complexities with our first result but requires the XDH assumption for bilinear groups (which is only conjectured to hold for ordinary elliptic curves). The second construction works on more general types of elliptic curves, but the price for this is that the complexity of the spending and of the withdrawal protocols become $O(\ell \cdot k)$ and $O(\ell \cdot k + k^2)$, respectively, and wallets take $O(\ell \cdot k)$ bits of storage. This scheme is also secure in the random oracle model.

4. **Add Bounded-Anonymity without a Trusted Third Party.** We then extend the two results above to our third system, the first e-cash scheme that supports bounded-anonymity without a trusted third party. In our previous schemes, a user can spend each of her coins anonymously and unlinkably unless she abuses her anonymity by double-spending a coin. In Chapter 7, however, we ask: what other abuses of the system should be forbidden?

For some applications, it is desirable to set a limit on the dollar amounts of anonymous transactions. For example, governments require that large transactions be reported for tax purposes. In this work, we present the first e-cash system that makes this possible without a trusted party. In our system, a user's anonymity is guaranteed so long as she does not: (1) double-spend a coin, or (2) exceed the publicly-known spending limit with any merchant. The spending limit may vary with the merchant. Violation of either condition can be detected, and can (optionally) lead to identification of the user and tracing of her past transactions. While it is possible to balance accountability and privacy this way using e-cash, this is impossible to do using regular cash!

To achieve this functionality, we do not add any complexity assumptions beyond the ones employed in the previous result. We maintain the same asymptotic complexities as the original system, although the constant roughly doubles.

# Chapter 2

# Bilinear Groups

**Notation:** we write $\mathbb{G} = \langle g \rangle$ to denote that element $g$ generates group $\mathbb{G}$.

We being with an abstract definition of bilinear groups and the map that connects them. Galbraith, Paterson, and Smart [109] recently presented guidelines for cryptographers using bilinear groups in an abstract fashion, which we will follow.

**Definition 2.1 (Bilinear Map).** *Let algorithm* Bilinear_Setup, *on input the security parameter* $1^k$, *output parameters* $(\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ *for a* ***bilinear map*** *where:*

1. $\mathbb{G}_1, \mathbb{G}_T$ *are both (multiplicative) cyclic groups of prime order* $q = \Theta(2^k)$;
2. *each element of* $\mathbb{G}_1$, $\mathbb{G}_2$, *and* $\mathbb{G}_T$ *has a unique binary representation;*
3. *the parameters include a generator for the domain groups* $\mathbb{G}_1 = \langle g_1 \rangle$ *and* $\mathbb{G}_2 = \langle g_2 \rangle$.
4. $\mathsf{e}$ *is an efficiently computable bilinear map* $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ *such that:*
   - *(Bilinear) for all* $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, *and all* $a, b \in \mathbb{Z}_q^2$, $\mathsf{e}(g_1^a, g_2^b) = \mathsf{e}(g_1, g_2)^{ab}$;
   - *(Non-degenerate) if* $g_1$ *is a generator of* $\mathbb{G}_1$ *and* $g_2$ *is a generator of* $\mathbb{G}_2$, *then* $\mathsf{e}(g_1, g_2)$ *generates* $\mathbb{G}_T$.

Bilinear maps are also referred to as *pairings*, after the Weil and Tate [35, 108] pairing typically used to implement them.

**Definition 2.2 (Bilinear Groups).** *Let algorithm* Bilinear_Setup, *on input the security parameter* $1^k$, *output parameters* $(\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ *for a bilinear map. Then we call groups* $\mathbb{G}_1$, $\mathbb{G}_2$, *and* $\mathbb{G}_T$ ***bilinear groups***.

## 2.1 Elliptic Curves

The existence of an efficient bilinear mapping $\mathsf{e}$ is what distinguishes bilinear groups from other algebraic groups commonly used in cryptography. The current candidate implementations for bilinear groups are constructed using elliptic curves.

**Definition 2.3 (Elliptic Curve).** *An* ***elliptic curve*** *is a set of solutions* $(x, y)$ *to the equation* $y^2 = x^3 + ax + b$ *together with an extra point at infinity.*

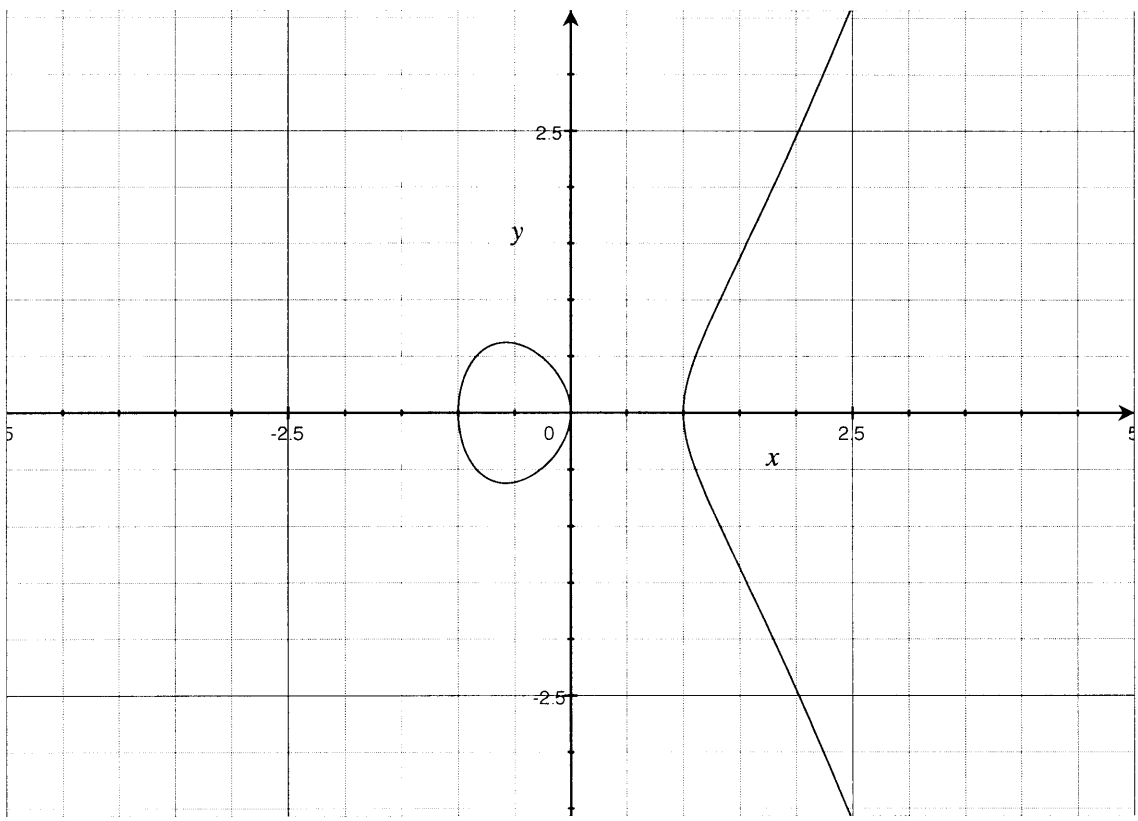


Figure 2-1: The elliptic curve defined by $y^2 = x^3 - x$.

In Figure 2-1, we illustrate a sample elliptic curve where $a = -1$ and $b = 0$.

**Definition 2.4 (Elliptic Curve Group).** *An **elliptic curve group** is the set of points described above, together with:*

1. *an identity element $\mathcal{O}$ defined as the point at infinity.*

2. *an (additive) group operation defined geometrically. The definition of this operation is dependent on the field over which the curve is implemented (e.g., real numbers, the finite field modulo p). For cryptographic purposes, we often work over $\mathbb{F}_p$, the finite field modulo p, where the group operation for adding two points $S = (x_s, y_s)$ and $T = (x_t, y_t)$ is defined as follows* [68]. *The negative of a point $Z = (x_z, y_z)$ is $-Z = (x_z, -y_z \bmod p)$.*

   (a) *if $S = -T$, then output the identity $\mathcal{O}$.*
   (b) *if $S = T$ and $y_s \neq 0$, let $z = (3x_s^2 + a)/2y_s \bmod p$ be the tangent line to the curve at this point, where a is a parameter of the elliptic curve $y^2 = x^3 + ax + b$. Then we have $2S = R$, where $x_r = z^2 - 2x_s \bmod p$ and $y_r = -y_s + z(x_s - x_r) \bmod p$.*
   (c) *if $S = T$ and $y_s = 0$, then the tangent line does not intersect any other point on the curve, and the output is the point at infinity $\mathcal{O}$.*
   (d) *if $S \neq T$ and $(x_s - x_t) \neq 0$, let $z = (y_s - y_t)/(x_s - x_t) \bmod p$ be the slope of the line between the two points. Then we have $S + T = R$, where $x_r = z^2 - x_s - x_t \bmod p$ and $y_r = -y_s + z(x_s - x_r) \bmod p$.*
   (e) *if $S \neq T$ and $(x_s - x_t) = 0$, then the slope of the line between the two points is undefined, and the output is the point at infinity $\mathcal{O}$.*

Although the group operation is an additive one, we will follow the standard notation [35, 34] and express these groups using multiplicative notation. The reason this notation was adopted is that, given a bilinear mapping $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, the group $\mathbb{G}_T$ is always a multiplicative group. Thus, it is often easier to make sense of cryptographic protocols using bilinear groups when all groups are expressed in the same notation.

## 2.2 Isomorphisms

We classify bilinear mappings based on the *efficiently-computable* isomorphisms existing between the groups $\mathbb{G}_1$ and $\mathbb{G}_2$. For our purposes, we are interested in the following two cases:

1. **Double Isomorphism:** There exists a pair of *distortion maps* $(\psi, \psi')$ such that $\psi$ is an efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$, with $\psi(g_2) = g_1$, and $\psi'$ is an efficiently computable is isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$, with $\psi'(g_1) = g_2$, and $\psi' = \psi^{-1}$.

   Candidate implementations: *supersingular* elliptic curves [179, 30, 29].

2. **Single-or-None Isomorphisms:** There does not exist an efficiently computable isomorphism $\psi'$ from $\mathbb{G}_1$ to $\mathbb{G}_2$; whereas an efficiently computable isomorphism $\psi$ from $\mathbb{G}_2$ to $\mathbb{G}_1$ may or may not exist.

Candidate implementations: *ordinary* elliptic curves: Barreto-Naehrig [16], Brezing-Weng [45], Cocks-Pinch [83], Freeman [101], Miyaji-Nakabayashi-Takano (MNT) [147].

It is well known that efficient isomorphisms exist for supersingular curves [179]. However, it is conjectured that bilinear groups without at least one efficient isomorphism can be realized using any of the ordinary curves listed above [71].

## 2.3 The DDH-CDH Gap

In 2001, bilinear groups first gained enormous popularity in cryptography due to the breakthrough result of Boneh and Franklin in the area of identity-based encryption [30]. At that time, Boneh and Franklin considered only double-isomorphism bilinear mappings, i.e., those implemented using supersingular curves, where efficient double isomorphisms $(\psi, \psi')$ between $\mathbb{G}_1$ and $\mathbb{G}_2$ are known.

Given the existence of $(\psi, \psi')$, the (distinct) groups $\mathbb{G}_1$ and $\mathbb{G}_2$ were often treated abstractly as a *single* group, with mappings being expressed as $\mathsf{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Using this simplified notation, it becomes clearer why the frenzy over bilinear groups began in cryptography: the famous *Decisional* Diffie-Hellman (DDH) problem in $\mathbb{G}$ is now easy to solve, while the *Computational* Diffie-Hellman (CDH) problem in $\mathbb{G}$ appears to remain hard. This gap was first explicitly observed by Joux and Nguyen in 2001 [127].

Let us informally recall these problems. Let $g$ be a random generator of group $\mathbb{G}$.

- **DDH:** Given $(g, g^a, g^b, Q)$, decide if $Q = g^{ab}$ or not.
- **CDH:** Given $(g, g^a, g^b)$, compute $g^{ab}$.

So, why is DDH easy in $\mathbb{G}$? Recall that a bilinear map, by itself, is not sufficient to decide DDH in either $\mathbb{G}_1$ or $\mathbb{G}_2$. Instead this property is obtained via the isomorphisms. When such an isomorphism $\psi : \mathbb{G}_2 \to \mathbb{G}_1$ is available, it can be used to decide DDH instances in $\mathbb{G}_2$ as: on input $(g_2, g_2^a, g_2^b, Q)$, test if $\mathsf{e}(\psi(g_2), Q) = \mathsf{e}(\psi(g_2^a), g_2^b)$. For bilinear groups where such isomorphisms are not known, it is open whether the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are DDH-easy.

In Chapter 6, we will show how to optimize our e-cash system by assuming that DDH is hard in $\mathbb{G}_1$ for curves where the isomorphism $\psi' : \mathbb{G}_1 \to \mathbb{G}_2$ is not known.

## 2.4 Security and Efficiency

There are two main reasons why elliptic curve groups are used in cryptography. The first reason, which we covered above, is that they are the best known candidate implementations of bilinear maps, a powerful cryptographic building block. This is the main focus of our thesis.

The second reason is time and space efficiency, from which our protocols also benefit. Indeed, the US National Security Agency (NSA) makes the following strong statement about elliptic curve cryptography on their website [151]:

*Elliptic Curve Cryptography provides greater security and more efficient performance than the first generation public key techniques (RSA and Diffie-Hellman) now in use.*

Indeed, Miller proposed using elliptic curves to improve the time and space efficiency of cryptographic protocols in 1985 [146] and in 1998, the Elliptic Curve Digital Signature Algorithm (ECDSA) was adopted as a US standard [4], well before the explosion of bilinear-based protocols in 2001. The observation motivating this interest in elliptic curves was that, unlike other popular cryptographic groups like $\mathbb{Z}_n^*$ where $n$ is an RSA modulus, the best known methods for taking discrete logarithms (DL) in *general* elliptic curve groups were the *generic* algorithms. By *generic*, we mean algorithms that do not take advantage of the representation of the group and thus work in *any* finite algebraic group.

Indeed, since the 1985 introduction of elliptic curves in cryptography by Miller [146], less progress has been made in attacking these groups than those algebraic groups proposed by Diffie and Hellman [88] in 1976 and Rivest, Shamir, and Adelman [165] in 1978 [151].

According to Galbraith [106], there are over a half dozen algorithms for computing discrete logarithms in elliptic curve groups, including Pohlig-Hellman, Shank's baby-step-giant-step, and Pollard's method (especially the parallel Pollard method of van Oorschot and Wiener). Due to the Pohlig-Hellman algorithm, one must always choose a generator $g_1$ for group $\mathbb{G}_1$ (or $g_2$ for group $\mathbb{G}_2$) of large prime order. Using this precaution, however, the only DL algorithms that apply generally are Shank's baby-step-giant-step and Pollard's [106]. Both algorithms take exponential time; although, there are a few sub-exponential (but super-polynomial) algorithms for specific classes of curves.

For example, while supersingular curves can implement the bilinear map functionality, they are *not* used for, say, obtaining short signatures, because they are a special class of curves on which Coppersmith's algorithm works effectively [35, 34]. Instead, ordinary curves, such as MNT, are used for this purpose [35, 34].

### 2.4.1 Space Requirements for Elliptic Curve Keys and Elements

In Table 2.1, we list the National Institute of Standards and Technology (NIST) estimated key sizes, as taken from the NSA webpage [151], for: (1) the symmetric ciphers, Data Encryption Standard (DES) and Advanced Encryption Standard (AES), (2) RSA and Diffie-Hellman, and (3) elliptic curve cryptosystems. An example Diffie-Hellman group is the cyclic subgroup of $\mathbb{Z}_p^*$ of prime order $q$, where $p = 2q + 1$ and $p, q$ are large primes.

NIST recommends that by December 31, 2008 all encryption algorithms use keys providing the equivalent security of 112 bit symmetric keys [154]. RSA Security makes the same recommendation with a deadline of 2010 [130]. Thus, we see that using elliptic curves could save an order of magnitude in data transmission in the coming years.

These key sizes can also serve as rough approximations of the size of group elements in RSA and elliptic curve groups [151]. More accurate approximations would need to fix a curve, take into account the embedding degree, and other considerations beyond the scope of this work. As one example, in their work on *short* signatures, Boneh, Lynn, and Shacham [35, 34]

| Symmetric Key Size (bits) | RSA and Diffie-Hellman Key Size (bits) | Elliptic Curve Key Size (bits) |
|---|---|---|
| 80 | 1024 | 160 |
| 112 | 2048 | 224 |
| 128 | 3072 | 256 |
| 192 | 7680 | 384 |
| 256 | 15360 | 521 |

Table 2.1: NIST estimated key sizes for an equivalent level of security.

| Security Level (bits) | Ratio of Diffie-Hellman Cost : Elliptic Curve Cost |
|---|---|
| 80 | 3:1 |
| 112 | 6:1 |
| 128 | 10:1 |
| 192 | 32:1 |
| 256 | 64:1 |

Table 2.2: Due to the shorter key sizes, standard arithmetic operations in elliptic curve groups run more efficiently than comparable operations in Diffie-Hellman groups.

estimate that using MNT curves the security of their $n$-bit signature is roughly equivalent to the discrete logarithm problem in a finite field of size $2^{6 \cdot n}$. Thus, to achieve 1024-bit security, their signatures would be of length $1024/6 = 171$ bits. Again, this represents an order of magnitude space savings over RSA signatures! Fortunately, the signatures we present in Chapter 4 inherit the Boneh, Lynn, and Shacham space efficiency.

### 2.4.2 Speed of Elliptic Curve Operations

Let us now discuss how fast elliptic curve operations are in practice. In Table 2.2, we list an estimation of the relative cost of arithmetic operations (e.g., modular exponentiations, modular multiplications) in traditional Diffie-Hellman and elliptic curve groups, as taken from the NSA webpage [151].

Table 2.2 deals with standard group operations such as modular exponentiation and multiplication. A separate issue of great interest is how efficiently one can compute the bilinear mapping e. Several researchers have argued that an optimized version of the Tate pairing over ordinary curves provides the fastest mapping [108, 15].

In Chapter 5 (see Section 5.4), we present performance measurements for various bilinear operations using the Tate pairings over supersingular curves. Our mapping estimates confirm a rumor known to us before we started this work that the cost of computing one (supersingular) mapping $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is approximately equal to eight modular exponentiations in $\mathbb{G}$ for a security level of 256 bits. In fact, in our tests, a mapping cost between six to eight exponentiations and roughly five mappings for a 256 bit security level could be computed per second on a 1.8Ghz computer.

# Chapter 3

# Complexity Assumptions

**Notation:** we write $\mathbb{G} = \langle g \rangle$ to denote that element $g$ generates group $\mathbb{G}$. Let $x \xleftarrow{R} S$ denote that the element $x$ is chosen uniformly at random from the set $S$.

We now review the complexity assumptions made in this thesis. We begin with a classic assumption that will only be used for our e-cash system in Chapter 6.

**Assumption 3.1 (Strong RSA [14, 105]).** *Let $n = pq$ be an RSA modulus, where $p = 2p' + 1$ and $q = 2q' + 1$ are $k$-bit primes, and $p'$ and $q'$ are also primes. Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[x \xleftarrow{R} \mathbb{Z}_n^*;\ (y, e) \leftarrow \mathsf{Adv}(n, x)\ :\ (e > 1) \wedge (y^e \equiv x \bmod n)] < 1/\mathrm{poly}(k).$$

The next three assumptions appeared at the earliest days of public key cryptography. Here the algebraic group $\mathbb{G}$ may or may not be a bilinear group.

**Assumption 3.2 ($y$-Discrete Logarithm ($y$-DL)).** *Let $\mathbb{G} = \langle g \rangle$ be of arbitrary order $q \in \Theta(2^k)$. Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[a \xleftarrow{R} \mathbb{Z}_q\ :\ a \leftarrow \mathsf{Adv}(\mathbb{G}, g, g^a, g^{(a^2)}, \ldots, g^{(a^y)})] < 1/\mathrm{poly}(k).$$

Note that when $y = 1$, the above assumption is the classic discrete logarithm assumption.

**Assumption 3.3 (Computational Diffie-Hellman (CDH) [88]).** *Let $\mathbb{G} = \langle g \rangle$ be of arbitrary order $q \in \Theta(2^k)$. Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[a, b \xleftarrow{R} \mathbb{Z}_q^2\ :\ g^{ab} \leftarrow \mathsf{Adv}(\mathbb{G}, g, g^a, g^b)] < 1/\mathrm{poly}(k).$$

The **Co-CDH** variant on the above involves two groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$, and states that given $(g_1, g_1^a, g_2, g_2^b)$, it is hard to compute $g_2^{ab}$.

**Assumption 3.4 (Decisional Diffie-Hellman (DDH) [97]).** *Let $\mathbb{G} = \langle g \rangle$ be of arbitrary order $q \in \Theta(2^k)$. Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[a, b \xleftarrow{R} \mathbb{Z}_q^2;\ x_0 = g^{ab};\ x_1 \xleftarrow{R} \mathbb{G};\ d \xleftarrow{R} \{0, 1\};\ d' \leftarrow \mathsf{Adv}(\mathbb{G}, g, g^a, g^b, x_d)\ :\ d = d'] < 1/2 + 1/\mathrm{poly}(k).$$

The **Co-DDH** variant on the above involves two groups $\mathbb{G}_1 = \langle g_1 \rangle$ and $\mathbb{G}_2 = \langle g_2 \rangle$, and states that given $(g_1, g_1^a, g_2, g_2^b, Q)$, it is hard to decide if $Q = g_2^{ab}$.

The following assumption was introduced by Boneh and Boyen [25]. To be well-defined, it is set in a group of prime order.

**Assumption 3.5 ($y$-Decisional Diffie-Hellman Inversion ($y$-DDHI) [25, 95]).** *Let $\mathbb{G} = \langle g \rangle$ be of prime order $q \in \Theta(2^k)$. Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[a \xleftarrow{R} \mathbb{Z}_q;\ x_0 = g^{1/a};\ x_1 \xleftarrow{R} \mathbb{G};\ d \xleftarrow{R} \{0,1\};\ d' \leftarrow \mathsf{Adv}(\mathbb{G}, g, g^a, g^{(a^2)}, \ldots, g^{(a^y)}, x_d)\ : \\ d = d'] < 1/2 + 1/\mathrm{poly}(k).$$

Alternatively, when in a bilinear setting, we have $x_0 = \mathsf{e}(g,g)^{1/a}$ and $x_1$ drawn from the target group [25, 95]. This variant is called the $y$-Decisional Bilinear Diffie-Hellman Inversion ($y$-DBDHI) assumption.

This next assumption was introduced by Dodis [89], and will only be used for our e-cash system in Chapter 6.

**Assumption 3.6 (Sum-Free Decisional Diffie-Hellman (SF-DDH) [89]).** *Let $\mathbb{G} = \langle g \rangle$ be of prime order $q \in \Theta(2^k)$. Let $L$ be any polynomial function of $k$. Let $O_{\vec{a}}(\cdot)$ be an oracle that, on input a subset $I \subseteq \{1, \ldots, L\}$, outputs the value $g^{\beta_I}$ where $\beta_I = \prod_{i \in I} a_i$ for some $\vec{a} = (a_1, \ldots, a_L) \in \mathbb{Z}_q^L$. Further, let $R$ be a predicate such that $R(J, I_1, \ldots, I_t) = 1$ if and only if $J \subseteq \{1, \ldots, L\}$ is DDH-independent from the $I_i$'s; that is, when $v(I_i)$ is the $L$-length vector with a one in position $j$ if and only if $j \in I_i$ and zero otherwise, then there are no three sets $I_a, I_b, I_c$ such that $v(J) + v(I_a) = v(I_b) + v(I_c)$ (where addition is bitwise over the integers). Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}^{(\cdot)}$,

$$\Pr[\vec{a} = (a_1, \ldots, a_L) \xleftarrow{R} \mathbb{Z}_q^L;\ (J, \alpha) \leftarrow \mathsf{Adv}^{O_{\vec{a}}}(1^k);\ y_0 = g^{\prod_{i \in J} a_i};\ y_1 \xleftarrow{R} \mathbb{G}; \\ d \xleftarrow{R} \{0,1\};\ d' \leftarrow \mathsf{Adv}^{O_{\vec{a}}}(1^k, y_b, \alpha)\ :\ d = d' \wedge R(J, Q) = 1] < 1/2 + 1/\mathrm{poly}(k),$$

*where $Q$ is the set of queries that* $\mathsf{Adv}$ *made to $O_{\vec{a}}(\cdot)$.*

We now focus our attention on decisional assumptions purely in the context of bilinear groups.

**Assumption 3.7 (Decisional Bilinear Diffie-Hellman (DBDH) [30]).** *Let algorithm* $\mathsf{Bilinear_Setup}(1^k)$ *output the bilinear map parameters* $(\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$, *where there is an efficiently computable isomorphism $\psi$ from $\mathbb{G}_2$ to $\mathbb{G}_1$. Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[a, b, c \xleftarrow{R} \mathbb{Z}_q^3;\ x_0 = \mathsf{e}(g_1, g_2)^{abc};\ x_1 \xleftarrow{R} \mathbb{G}_T;\ d \xleftarrow{R} \{0,1\}; \\ d' \leftarrow \mathsf{Adv}(g_1, g_2, g_2^a, g_2^b, g_2^c, x_d)\ :\ d = d'] < 1/2 + 1/\mathrm{poly}(k),$$

**Assumption 3.8 (Extended Decisional Bilinear Diffie-Hellman (eDBDH) [8]).** *Let* $\mathsf{Bilinear_Setup}(1^k)$ *output the bilinear map parameters* $(\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$. *where there is an efficiently computable isomorphism* $\psi$ *from* $\mathbb{G}_2$ *to* $\mathbb{G}_1$. *Then, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$,

$$\Pr[a, b, c \xleftarrow{R} \mathbb{Z}_q^3;\ x_0 = \mathsf{e}(g_1, g_2)^{abc};\ x_1 \xleftarrow{R} \mathbb{G}_T;\ d \xleftarrow{R} \{0,1\};$$
$$d' \leftarrow \mathsf{Adv}(g_1, g_2, g_2^a, g_2^b, g_2^c, \mathsf{e}(g_1, g_2)^{bc^2}, x_d)\ :\ d = d'] < 1/2 + 1/\mathrm{poly}(k),$$

This final assumption– requiring a single-isomorphism bilinear mapping –will only be required for an *optimization* of our e-cash system in Chapter 6.

**Assumption 3.9 (External Diffie-Hellman (XDH) [107, 171, 143, 27, 12]).** *Let* $\mathsf{Bilinear_Setup}(1^k)$ *output the parameters for a bilinear mapping* $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. *The XDH assumption states that, for all probabilistic polynomial time adversaries* $\mathsf{Adv}$, *the DDH problem is hard in* $\mathbb{G}_1$. *This implies that there does* not *exist an efficiently computable isomorphism* $\psi' : \mathbb{G}_1 \rightarrow \mathbb{G}_2$.

This concludes the list of complexity assumptions on which our results are based.

## 3.1 Discussion of Assumptions

A cryptosystem is only as secure as the complexity assumptions on which it is based. If one of these assumptions proves to be false, all confidence in the system is shattered. In this thesis, we do *not* introduce any new assumptions. All of our schemes depend on complexity assumptions previously used to build other cryptographic systems. We made an effort to build our schemes from the simplest and longest-standing complexity assumptions possible. For some of our schemes, one assumption may be substituted for another (perhaps at some loss of efficiency); whenever substitutions are available, we will discuss the options.

Previously, we listed nine main assumptions. The first four assumptions– Strong RSA, DL, CDH, and DDH –are long-standing, foundation assumptions in cryptography. The DBDH and eDBDH assumptions are two very mild assumptions for bilinear groups. There are some relations among these assumptions and the remaining others. It is easy to see that DDH implies CDH, SF-DDH implies DBDH and eDBDH, and all five assumptions imply DL (i.e., $y$-DL for $y = 1$). Boneh and Boyen [24] also showed that $y$-DDHI implies a generalized form of DH problems which includes DDH, DBDH, and eDBDH. The relation between $y$-DDHI and Sum-Free DDH is unknown.

As a method of evaluating and comparing the hardness of various assumptions, it has become standard practice to provide a lower bound on the complexity of Diffie-Hellman based assumptions in generic groups using Shoup's method [152, 173]. Shoup originally showed that for groups of order $p$ where the adversary is allowed to perform $q_G$ group operations an adversary's probability of solving CDH is at most $O(q_G^2/p)$ and DDH is at most $1/2 + O(q_G^2/p)$ [173]. Boneh and Franklin showed a bound of $O(q_G^2/p)$ for DBDH [29], which can be easily extended to provide the same bound for eDBDH. Boneh, Boyen, and

Goh also showed a bound of $O((q_G^2 \cdot y + y^3)/p)$ for $y$-DDHI [26]. We could not find a generic group proof for Sum-Free DDH, although we conjecture that the bound is $O((g_G^2 \cdot y + y^3)/p)$ where the adversary is allowed to make $y$ calls to oracle $O_{\vec{a}}$.

Finally, as mentioned earlier, Galbraith, Paterson, and Smart recently published an excellent guide for which assumptions are plausible for bilinear groups [109] to which we point the more interested reader.

# Chapter 4

# Proxy Re-Signatures

This chapter is an extended version of the joint work with Giuseppe Ateniese (Johns Hopkins University) [10].

## 4.1 Introduction

In a *proxy re-signature scheme*, a semi-trusted proxy is given some information which allows it to transform Alice's signature on a message $m$ into Bob's signature on $m$, but the proxy cannot, on its own, generate signatures for either Alice or Bob.

This primitive was introduced at Eurocrypt 1998 by Blaze, Bleumer, and Strauss [21] and yet very little follow up work has been done, to our knowledge. One explanation is that the BBS original construction [21] is inefficient and has limited features. Moreover, the definition of proxy re-signature in the BBS paper [21] is informal and has created some confusion between the notion of proxy re-signature and the distinct one of *proxy signature* as introduced by Mambo, Usuda, and Okamoto [142].

Proxy signatures [142] allow Alice to delegate her signing rights to Bob but only if the proxy cooperates in the signing. This is usually accomplished by dividing Alice's secret into two shares which are distributed to Bob and the proxy (each gets only one share). A signature from Alice on a message is generated by combining two partial signatures on the same message computed by Bob and the proxy under their own shares, respectively.

By contrast, in proxy re-signatures [21], a proxy "translates" a perfectly-valid and publicly-verifiable signature from Alice on a certain message, which we'll denote as $\sigma_A(m)$, into one from Bob on the same message, denoted as $\sigma_B(m)$. Notice that, in a proxy re-signature, the original and re-signed signature as generated by the proxy, can coexist and both can be publicly verified as being two signatures from two distinct parties on the same message. Moreover, with additional setup, the proxy can convert a single signature into multiple signatures of several and distinct signers, and vice-versa!

The relationship between these primitives is that proxy re-signatures are a strict subset of proxy signatures. Any proxy re-signature scheme can be used to build a proxy signature trivially: Bob delegates to Alice by giving the proxy the ability to turn her signatures into

his. However, the reverse is not necessarily true. For instance, it is possible to build a proxy signature based on RSA (as Dodis and Ivan did [92], by splitting Alice's secret $d$ into $d_1$ and $d_2$ such that $d = d_1 + d_2$) but it is not possible to have a proxy re-signature scheme that translates between two publicly-verifiable RSA signatures sharing the same modulus (because of the *common modulus* problem/attack).

Another unique property of proxy re-signatures is that the "translation" from one signature to another can be performed in sequence and multiple times by distinct proxies without even requiring the intervention of the signing entities. Thus, the valuable secret keys can remain *offline*. The signatures generated in this process are all valid and publicly-verifiable signatures on the same message from distinct entities.

We re-open the discussion of proxy re-signatures by providing four separate results:

1. We first motivate the need for improved schemes, by pointing out that the original BBS scheme [21], while satisfying their security notion, is unsuitable for most practical applications, including the ones proposed in their original paper.
2. We provide formal definitions and a security model.
3. We introduce provably secure proxy re-signature constructions from bilinear groups.
4. We present new applications and perform comparisons with other cryptographic primitives.

Incidentally, we also introduce a new signature scheme based on bilinear groups where two signing secrets are associated to a single public key. Let us briefly describe these results in more detail.

**1. Remarks on the BBS Scheme.** The authors of the BBS paper introduced several potential applications of proxy re-signatures. By taking a careful look at how one might wish to use proxy re-signatures in practice, we noticed that the BBS construction [21] has inherent limitations. In short, their scheme is actually "proxy-less" since it is possible to recover the information that would be stored at the proxy (the re-signature key) by looking at the original signature and its transformation. This precludes the possibility of having a secure proxy in the first place since anyone would be able to impersonate the proxy itself once a *single* re-signature is released. Moreover, the BBS scheme is what the authors called *symmetric* [21], which means that, from the re-signature key (which is public!), Alice can recover Bob's secret key or vice-versa. We review the BBS scheme and prove these claims in Section 4.3.2.

Finding secure schemes without these limitations turned out to be a very difficult task. Many proxy re-signature schemes based on standard signature algorithms that we investigated were found susceptible to the same type of problems. Several signature schemes based on bilinear groups also failed or it was not clear how to turn them in proxy re-signatures.

**2. Formal Definitions and Security Model.** There is no formal definition of proxy re-signatures in the BBS paper [21], which may have caused some of the problems mentioned earlier.

In this chapter, we formalize the notion of a proxy re-signature and provide a security model that incorporates the desirable property of having the re-signature key safely stored at the proxy (so that it is reasonable to talk about *proxies* in this context). First, this allows us to make meaningful claims about our schemes' security. In particular, in Section 4.3.3, we realize a strong notion of security: Suppose Bob delegates to Alice, via a proxy, the ability to change her signatures into his. As one might expect, even when the proxy and Bob collude, they cannot recover any information about Alice except her public key. More surprisingly, we show that even when the proxy and Alice collude, they can only recover a weak version of Bob's secret key – that only gives them the power to compute what Bob had already delegated to them. Secondly, our formal notion allows us to view the primitive abstractly, for easier reasoning about its applications in Section 4.4.

**3. Two Proxy Re-Signature Constructions.** We present two different constructions based on bilinear groups. The first is a *bidirectional* proxy re-signature in which the proxy can translate from Alice's signatures to Bob's and vice-versa using a single proxy key. The scheme is very attractive for its simplicity. Unlike the bidirectional BBS scheme, here the proxy can keep its proxy keys private. This scheme also allows for *multi-use*, meaning that a signature may be transformed from Alice to Bob, then from Bob to Carol, and so on. The security of the scheme is based on the Computational Diffie-Hellman (CDH) assumption, i.e., given $(g, g^x, g^y)$, it is hard to compute $g^{x \cdot y}$, in the random oracle model.

The second scheme is *unidirectional* in that the proxy can be given information that allows it to translate from Alice to Bob, but not from Bob to Alice. This is the first construction of a unidirectional scheme since it was proposed as an open problem by BBS seven years ago [21]. (In BBS, they refer to such schemes as *asymmetric.*) Here, we allow the re-signature key to be public so that anyone can act like a proxy but, at the same time, we ensure that certain important security properties are guaranteed based on its unidirectional nature. (We also provide some insight on how one might keep this proxy key private.) The security of this scheme is based on the CDH and 2-Discrete Logarithm (2-DL) assumptions in the random oracle model. Recall 2-DL: given $(g, g^x, g^{x^2})$, it is hard to compute $x$.

Finally, we note that our unidirectional scheme introduces a new signature algorithm that may be of independent interest, because it allows the signer to use *two secrets* (strong and weak) for a *single* public key (more details in Section 4.3.4).

**4. Applications.** We propose exciting new applications of proxy re-signatures in Section 4.4. In particular we show how to use proxy re-signatures to provide a proof that a certain flow through a directed graph on a set of nodes occurred. In the simplest case, the basic idea is that each node in the path (except the first) is only given a re-signature key which allows it to translate signatures from predecessor nodes, but which is *not* a signing key. For instance, given three nodes in a path $A \rightarrow B \rightarrow C$, we give the first node $A$'s signing key, while the second node is only able to translate signatures from $A$ into signatures from $B$, without using (or storing) $B$'s signing key. Then, node $C$ only has to verify a single signature from $B$ even if several nodes (not just $A$) precede $B$ in the path.

Our technique requires some pre-configuration but provides several benefits: (1) all (but

the first) nodes in the graph do not store any signing key so that, even if a node is compromised, no new messages can be injected in the path, (2) only a single signature must traverse the path, i.e., there is no need to accumulate signatures and public keys while traversing the path, (3) no information on the path itself is collected so that, if applications require it, the actual path traversed by a message could remain *private*, i.e., the last node of the path knows that the message followed a legitimate path but does not know which one (we stress that this property is optional). Indeed, in one of our constructions, scheme $\Omega_{bi}$ in Section 4.3.3, original signatures and their re-signed values cannot be linked, which allows for a strong privacy guarantee.

We will also introduce other interesting applications. For instance, we show how to use proxy re-signatures to share existing public-key certificates and save the cost of obtaining and distributing new ones. In particular, a signature $\sigma_1(m)$, that can only be verified with a new and/or uncertified public key $pk_1$, could be transformed into $\sigma_2(m)$ that can be verified with a trusted and certified public key $pk_2$. Depending on the application, translating between signatures could be more convenient then requesting a certificate for $pk_1$ and distributing it to several parties. Similarly, if a company wanted to set up a server that on input $\sigma_1(m)$ outputs $\sigma_2(m)$, it is much more desirable to give the server a proxy key rather than a signing key, as it makes the server a less attractive target for hackers.

We also show how to generate anonymous signatures that, much like group signatures, can be used to provide anonymity while also providing accountability. Indeed, a proxy could translate signatures from group members into signatures from the group manager. The advantage here is that the proxy does not store any signing key and the key of the group manager cannot get exposed even if the proxy is compromised.

### 4.1.1 Related Work

Proxy re-signatures [21] should not be confused with the similar sounding *proxy signatures* [142, 92] as previously discussed. In particular, definitions in Dodis et al. [92] are for proxy signatures: In their general construction, Bob's signature is seen as a *double signature* which includes a signature from Alice and one from the proxy. There is clearly no translation between valid signatures of Alice into valid signatures of Bob.

Proxy re-signatures share some properties with the transitive signatures as introduced by Micali and Rivest [145] and extended by Bellare and Neven [17, 18], Molnar [148], and Hohenberger [122]. In a transitive signature scheme, a single master signer signs edges of a graph in such a way that *anyone* (i.e., not only a proxy) in possession of signatures on edges $ab$ and $bc$ can compute the master's signature on edge $ac$. There is a symmetry between these schemes: in a transitive signature, the verification key stays constant while the message content changes, while in a proxy re-signature scheme the message content remains constant while the verification key changes. Transitive signatures may also be appropriate for *some* of our authenticated routing applications in Section 4.4.

Another signature variant that allows to transform one type of signature to another are convertible undeniable (also called invisible). Chaum and Van Antwerpen introduced the notion of *undeniable* signatures [80], where instead of having a public verification algorithm,

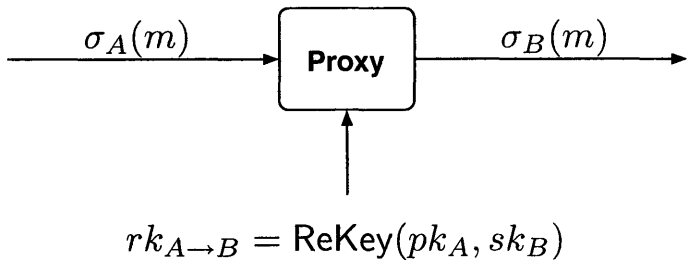


Figure 4-1: A high-level view of (unidirectional) proxy re-signatures. Here Bob is the delegator and Alice is the delegatee. The proxy obtains a re-signature key $rk_{A\to B}$ from Bob, and can then translate signatures from Alice, denoted $\sigma_A(m)$, into ones from Bob, denoted $\sigma_B(m)$.

a verifier runs a protocol with the signer to check the validity of a purported signature. Later, Boyar, Chaum, Damgård and Pedersen [38] proposed *convertible undeniable* signatures, where the signer can convert an undeniable signature, verifiable by one party at a time, into a universally verifiable signature.

Our work is also related to multi and aggregate signatures. Multi-signatures [144, 23, 157, 155] allow $n$ users, each with a public and private key pair $(pk_i, sk_i)$, to all sign the same message $m$ and have the result be just one single signature. Similarly, aggregate signatures, introduced by Boneh, Gentry, Lynn, and Shacham [31], allow each of the $n$ users to sign a distinct message such that the computed $n$ signatures, on $n$ *distinct* messages, can be aggregated into a single one. Our multi-use proxy re-signatures can be used as an alternative to multi-signatures, but not to aggregate signatures, given that we allow transformations to be made only between signatures on the same message. In certain applications (e.g., those requiring a chain of signers), multi and aggregate signatures have a potential drawback: the verifier must possess and trust $n$ public keys in order to verify the final *compressed* signature. In Section 4.4, we discuss how, when proxy re-signatures are used instead, the verifier need only possess and trust *one* public key. This cuts down on the storage needs and key distribution headaches of some practical applications.

We notice that our first proxy re-signature scheme, in Section 4.3.3, uses a construction similar to Boldyreva's multi-signature [23] and Dodis and Reyzin's "verifiably committed signature" [94] constructions which are both based on the short signature of Boneh, Lynn, and Shacham [35, 34]. However, the purpose, definition, and the security model of these cryptographic primitives are different and unrelated.

## 4.2 Definitions

Now, we explain the different components of a proxy re-signature scheme, as partially illustrated in Figure 4-1.

**Definition 4.1 (Proxy Re-Signature).** *A* proxy re-signature scheme *is a tuple of (possibly probabilistic) polynomial time algorithms* (SysGen, KeyGen, ReKey, Sign, ReSign, Verify), *where:*

- The $\mathsf{SysGen}(1^k)$ algorithm takes as input the security parameter $1^k$ and returns the global system parameters *params*.
- The $\mathsf{KeyGen}(params)$ algorithm outputs a user keypair $(pk, sk)$. Let $pk$ and $sk$ contain the system parameters *params* so that we do not have to explicitly give them to signers and verifiers in future.
- The $\mathsf{ReKey}(pk_A, sk_B)$ algorithm outputs a key $rk_{A \to B}$ for the proxy.

  (NOTE: for bidirectional schemes, ReKey is a *protocol* between the two users and the proxy, where the proxy obtains the key $rk_{A \leftrightarrow B}$.)
- The $\mathsf{Sign}(sk, m)$ algorithm outputs a signature $\sigma$.
- The $\mathsf{ReSign}(rk_{A \to B}, pk_A, \sigma, m)$ algorithm outputs $B$'s signature on message $m$ if $\mathsf{Verify}(pk_A, m, \sigma)$ accepts and $\perp$ otherwise.
- The $\mathsf{Verify}(pk, m, \sigma)$ algorithm accepts if $\sigma$ is a valid signature under key $pk$ on message $m$ and rejects otherwise.

**Correctness.** The correctness property has two requirements. For any message $m$ in the message space and any key pairs $(pk, sk), (pk', sk') \leftarrow \mathsf{KeyGen}(params)^2$ where $params \leftarrow \mathsf{SysGen}(1^k)$, let $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and $rk \leftarrow \mathsf{ReKey}(pk, sk')$. Then the following two conditions must hold:

$$\mathsf{Verify}(pk, m, \sigma) = 1 \quad \text{and} \quad \mathsf{Verify}(pk', m, \mathsf{ReSign}(rk, \sigma)) = 1.$$

That is, all signatures validly formed by either the signing or re-signing algorithms will pass verification.

**Internal and External Security.** Our security model protects users from two types of attacks: those launched from parties outside the system (*External Security*), and those launched from parties inside the system, such as the proxy, another delegation partner, or some collusion between them (*Internal Security*). We now provide both intuition and a formalization of these security notions.

**External Security:** Our first security notion protects a user from adversaries outside the system (i.e., excluding the proxy and any delegation partners). This is the proxy equivalent existential unforgeability against chosen message attack (where a forgery must be on a new message) [118]. However, our constructions are capable of satisfying the notion of *strong* existential unforgeability under adaptive chosen-message attack

(where an adversary cannot create a new signature even for a previously signed message) [3]. We point out how to strengthen our core definition meet strong existential unforgeability as well.

Formally, for any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathsf{Adv}$,

$$\begin{aligned}\Pr[&params \leftarrow \mathsf{SysGen}(1^k), \{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(params)\}_{i\in[1,n]},\\ &(t, m, \sigma) \leftarrow \mathsf{Adv}^{\mathcal{O}_{sign}(\cdot,\cdot),\mathcal{O}_{resign}(\cdot,\cdot,\cdot,\cdot)}(\{pk_i\}_{i\in[1,n]}) :\\ &\mathsf{Verify}(pk_t, m, \sigma) = 1 \wedge (1 \le t \le n) \wedge (t, m) \notin Q] < 1/\text{poly}(k)\end{aligned}$$

where the oracle $\mathcal{O}_{sign}$ takes as input an index $1 \le j \le n$ and a message $m \in M$, and produces the output of $\mathsf{Sign}(sk_j, m)$; the oracle $\mathcal{O}_{resign}$ takes as input two distinct indexes $1 \le i, j \le n$, a message $m$, and a signature $\sigma$, and produces the output of $\mathsf{ReSign}(\mathsf{ReKey}(pk_i, sk_j), pk_i, \sigma, m)$; and $Q$ denotes the set of (index, message) tuples $(t, m)$ where $\mathsf{Adv}$ obtained a signature on $m$ under public key $pk_t$ by querying $\mathcal{O}_{sign}$ on $(t, m)$ or $\mathcal{O}_{resign}(\cdot, t, m, \cdot)$.

For strong unforgeability, let $Q$ be the set of (index, message, signature) tuples $(t, m, \sigma)$ where $\mathsf{Adv}$ obtained the signature $\sigma$ on $m$ under public key $pk_t$ by querying $\mathcal{O}_{sign}$ on $(t, m)$ or $\mathcal{O}_{resign}(\cdot, t, m, \cdot)$.

In the above security notion, the proxy is *required* to keep the re-signature keys private (or it would be easy for an adversary to "win"). For some unidirectional schemes, however, one might want these values to be public (i.e., making all users proxies). When this is the case, there are no "external adversaries" to the system, and thus we look instead to the internal security guarantee below.

**Internal Security:** Our second security notion protects a user, as much as possible, when they are fooled into trusting a rogue proxy and/or delegation partner (who may be colluding with each other). Intuitively, there are three guarantees to make.

**1. Limited Proxy:** If the delegator and the delegatee are both honest, then: (1) the proxy cannot produce signatures for the delegator unless the message was first signed by one of her delegatees, and (2) the proxy cannot create *any* signatures for the delegatee. This is identical to the external security game, except that instead of a re-signing oracle $\mathcal{O}_{sign}$, $\mathsf{Adv}$ may directly obtain the re-signature keys via $\mathcal{O}_{rekey}$.

*Unidirectional:* For any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathsf{Adv}$,

$$\begin{aligned}\Pr[&params \leftarrow \mathsf{SysGen}(1^k), \{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(params)\}_{i\in[1,n]},\\ &(t, m, \sigma) \leftarrow \mathsf{Adv}^{\mathcal{O}_{sign}(\cdot,\cdot),\mathcal{O}_{rekey}(\cdot,\cdot)}(\{pk_i\}_{i\in[1,n]}) :\\ &\mathsf{Verify}(pk_t, m, \sigma) = 1 \wedge (1 \le t \le n) \wedge (t, m) \notin Q] < 1/\text{poly}(k)\end{aligned}$$

where the oracle $\mathcal{O}_{rekey}$ takes as input two distinct indexes $1 \le i, j \le n$ and returns the output of $\mathsf{ReKey}(pk_i, sk_j)$; and $Q$ denotes the set of pairs $(t, m)$ where $\mathsf{Adv}$ obtained a signature on $m$ under public key $pk_t$ or one of its delegatee keys by querying $\mathcal{O}_{sign}$.

*Bidirectional:* Since both parties mutually delegate, the set $Q$ includes all messages associated with *any* $\mathcal{O}_{sign}$ query.

**2. Delegatee Security:** If the delegatee is honest, then he is "safe" from a colluding delegator and proxy. That is, they cannot produce any signatures on his behalf. We associate the index 0 to the delegatee.

*Unidirectional:* For any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathsf{Adv}$,

$$\begin{aligned}\Pr[&params \leftarrow \mathsf{SysGen}(1^k), \{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(params)\}_{i\in[0,n]},\\ &(m,\sigma) \leftarrow \mathsf{Adv}^{\mathcal{O}_{sign}(0,\cdot),\mathcal{O}_{rekey}(\cdot,\star)}(pk_0, \{pk_i, sk_i\}_{i\in[1,n]}) :\\ &\qquad\qquad \mathsf{Verify}(pk_0, m, \sigma) = 1 \wedge m \notin Q] < 1/\text{poly}(k)\end{aligned}$$

where $\star \neq 0$ and $Q$ is the set of messages $m$ such that $\mathsf{Adv}$ queried $\mathcal{O}_{sign}(0, m)$.

For strong unforgeability, let $Q$ be the set of pairs $(m, \sigma)$ such that $\mathsf{Adv}$ queried $\mathcal{O}_{sign}(0, m)$ and obtained $\sigma$.

*Bidirectional:* Since both parties mutually delegate, this property does not apply.

**3. Delegator Security:** If the delegator is honest, then she is "safe" from a colluding delegatee and proxy. That is, there are two types of universally distinguishable signatures, called first and second level signatures, that a signer can create that verify with respect to the same public key. This property states that the colluding delegatee and proxy cannot produce a first-level signature on the delegator's behalf. We associate the index 0 to the delegator.

*Unidirectional:* For any non-zero $n \in \text{poly}(k)$ and all PPT algorithms $\mathsf{Adv}$,

$$\begin{aligned}\Pr[&params \leftarrow \mathsf{SysGen}(1^k), \{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(params)\}_{i\in[1,n]},\\ &(m,\sigma) \leftarrow \mathsf{Adv}^{\mathcal{O}_{sign}(0,\cdot),\mathcal{O}_{rekey}(\cdot,\cdot)}(pk_0, \{pk_i, sk_i\}_{i\in[1,n]}) :\\ &\qquad\qquad \mathsf{Verify}(pk_0, m, \sigma) = 1 \wedge m \notin Q] < 1/\text{poly}(k)\end{aligned}$$

where $\sigma$ is a first-level signature, and $Q$ is the set of messages $m$ where $\mathsf{Adv}$ queried $\mathcal{O}_{sign}(0, m)$. Since a signer can produce two types of signatures, we let the adversary tell $\mathcal{O}_{sign}$ whether it wants a first or second-level signature.

For strong unforgeability, let $Q$ be the set of pairs $(m, \sigma)$ where $\mathsf{Adv}$ queried $\mathcal{O}_{sign}(0, m)$ to obtain $\sigma$.

*Bidirectional:* This property is not required. In fact, we leave a proper formulation of bidirectional delegator security, if such a thing makes sense, as an open problem.

This ends our formal definition of proxy re-signatures. Two limitations of this definition are: (1) we consider only static adversaries; that is, $\mathsf{Adv}$ is not allowed to adaptively choose which users to corrupt, and (2) we require a trusted key issuing entity; that is, users receive their keypairs from an authority. We pursue schemes without these limitations as future work.

### 4.2.1 Re-Signatures and Program Obfuscation

One alternative method for formalizing (and potentially realizing) proxy re-signatures is to treat them as a special case of the more general problem of *program obfuscation* [13]. Informally, the goal in program obfuscation it to provide a (potentially malicious) party with a program it can execute, where an adversary learns nothing more from having the code of the program than she could have learned by having access to an oracle providing the same functionality. More formally, treating the program as a circuit, we want that for all adversaries $\mathsf{Adv}$, there exists a simulator $\mathcal{S}$, for all circuits $C$, such that the probability that $\mathsf{Adv}$, given an obfuscated version of $C$, outputs 1 is statistically indistinguishable from the probability that the simulator, with access to an oracle computing $C$, outputs 1.

This notion of program obfuscation was first formalized by Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan, and Yang [13]. While finding a general method for obfuscating an arbitrary program is impossible [13, 116], it is known how to successfully obfuscate some classes of programs, such as the point functions of Canetti [62] and Wee [181] and the mixnet of Adida and Wikström [1]. The question for us is whether or not re-signature programs can be obfuscated? This would allow us to define security for and reason about re-signatures in a more general framework.

The results of this chapter get us closer to an answer. Consider the following circuits that translate Alice's signature $\sigma$ on message $m$ into Bob's. Assume the circuit for the algorithms $(\mathsf{Sign}, \mathsf{ReSign}, \mathsf{Verify})$ and public keys are known to all.

1. *Circuit One:* hard-wire Bob's secret key $sk_B$ into the instructions: (1) run $\mathsf{Verify}(pk_A, m, \sigma)$ on Alice's purported signature, (2) if it verifies, return the output of $\mathsf{Sign}(sk_B, m)$.

2. *Circuit Two:* hard-wire the re-signature key $rk_{A\to B}$ into the instructions: (1) run $\mathsf{Verify}(pk_A, m, \sigma)$ on Alice's purported signature, (2) if it verifies, return the output of $\mathsf{ReSign}(rk_{A\to B}, \sigma)$.

Loosely speaking, circuit two is an obfuscation of circuit one, because circuit two does not leak information that allows to sign arbitrary messages on Bob's behalf (i.e., Bob's secret key). The question is: does circuit two meet the Barak et al. definition of obfuscation? Unfortunately, the answer appears to be no. Let us explain why. In our schemes in Section 4.3, $rk_{A\to B}$ is a deterministic function of $pk_A$ and $sk_B$ for unidirectional schemes, and of $sk_A$ and $sk_B$ for bidirectional schemes. Thus, in the obfuscation game of Barak et al. mentioned above, the probability distributions could be distinguished by $\mathsf{Adv}$ outputting a hardcore bit of $rk_{A\to B}$ (whereas $\mathcal{S}$ could only guess).

Indeed, we claim that our bidirectional scheme ($\Omega_{bi}$), as currently written, is *not* a valid obfuscation under the Barak et al. definition.

In our unidirectional scheme ($\Omega_{uni}$), the proxy key $rk_{A\to B}$ can be computed from an original signature and its re-signed value. Thus, by calling a re-signature oracle for this scheme, the simulator would be able to compute the key $rk_{A\to B}$ (note this is provably not the case for $\Omega_{bi}$.) Thus, $\Omega_{uni}$ *would* trivially meet the obfuscation definition.

Of course, a valid question to ask is: is the Barak et al. definition of program obfuscation the "right" one for re-signatures? If unforgeability against chosen message attack holds even

when proxy keys are known to the adversary, is that not enough? Perhaps the Barak et al. definition can be relaxed to capture the idea the an obfuscated program does not give the adversary any new abilities (such as the ability to sign arbitrary messages). We acknowledge Ran Canetti and Guy Rothblum for useful discussions on this topic. We intend to continue research in this direction, although we now wish to return to the focus of this thesis.

## 4.3 Proxy Re-Signature Schemes

We begin our discussion of proxy re-signature schemes by discussing the properties of these schemes that are either necessary or simply desirable for our applications in Section 4.4. Next, we motivate the need for improved schemes by detailing certain inherent limitations of the original BBS [21] scheme. We then present two secure proxy re-signature schemes: *bidirectional* and *unidirectional*. The bidirectional scheme $\Omega_{bi}$ is based on the short signatures of Boneh et al. [35, 34]. The unidirectional scheme $\Omega_{uni}$ is a novel El Gamal-type algorithm over bilinear groups. We also suggest an extension to the unidirectional scheme $\Omega^*_{uni}$.

### 4.3.1 Additional Properties We Need and Want

Let us first understand what properties we expect out of a proxy re-signature scheme in addition to correctness and security from Section 4.2. We now informally list what are, in our opinion, the most desirable properties of a proxy re-signature scheme. In Table 4.1, we show which properties we are currently able to realize.

1. *Unidirectional:* The re-signature key $rk_{A\to B}$ allows the proxy to turn Alice's signatures into Bob's, but not Bob's into Alice's. This property allows for applications where the trust relationship between two parties is not necessarily mutual. Schemes that do not have this property are called *bidirectional*.

2. *Multi-use:* A message can be re-signed a polynomial number of times. That is, signatures generated by either the Sign or ReSign algorithms can be taken as input to ReSign. In contrast, one might imagine weaker, *single-use* schemes where only signatures generated by Sign can be inputs to ReSign.

3. *Private Proxy:* In a *private proxy* scheme, the re-signature keys can be kept secret by an honest proxy. Thus, the proxy controls which signatures get translated. In *public proxy* schemes, the re-signature keys can be recomputed by an adversary passively observing the proxy.

4. *Transparent:* The proxy is transparent in the scheme, meaning that a user may not even know that a proxy exists. More formally, we mean that the signatures generated by Alice on a message $m$ using the Sign algorithm are computationally indistinguishable from her signatures on $m$ generated by the proxy as the output of ReSign. Notice that this implies that the input and the corresponding output of the ReSign algorithm cannot be linked to each other. It is plausible that transparency could be revoked using trapdoor information. For example, signatures could be original and re-signed signatures could be indistinguishable

| Property | BBS [21] | $\Omega_{bi}$ (Section 4.3.3) | $\Omega_{uni}$ (Section 4.3.4) |
|---|---|---|---|
| 1. Unidirectional | **No** | No | **Yes** |
| 2. Multi-use | **Yes** | Yes | **No** |
| 3. Private Proxy | **No** | **Yes** | **No**$^\sharp$ |
| 4. Transparent | Yes | Yes | Yes* |
| 5. Unlinkable | **No** | **Yes** | No |
| 6. Key Optimal | Yes | Yes | Yes |
| 7. Non-interactive | **No** | No | **Yes** |
| 8. Non-transitive | **No** | No | **Yes** |
| 9. Temporary | **No** | No | **Yes*** |

Table 4.1: We compare the properties of several proxy re-signature schemes discussed in this work. The symbol * denotes that the property can be provably partially obtained or obtained by adding additional overhead. For $\sharp$, we provide some insight on how this might be achieved.

to everyone except parties with knowledge of the signing key. We return to this point at the end of this section.

5. *Unlinkable:* In an *unlinkable* scheme, a re-signature cannot be linked to the original signature from which it was generated. This property should hold both for original signatures generated under two different keys as well as original signatures under the same key using different randomness (if the scheme is nondeterministic). For an example of a linkable scheme, see the "Discussion of Schemes $\Omega_{uni}$ and $\Omega^*_{uni}$" in Section 4.3.4.

6. *Key Optimal:* Alice is only required to protect and store a small constant amount of secret data (i.e., secret keys) regardless of how many signature delegations she gives or accepts. Here, we want to minimize the safe storage cost for each user. One might also wish to consider the size and number of keys that a proxy is required to safeguard.

7. *Non-interactive:* Bob (the delegator) can create the re-signature key $rk_{A\to B}$ from his secret key $sk_B$ and Alice's *public key* $pk_A$, i.e., the delegatee does not act in the delegation process.

8. *Non-transitive:* The proxy alone cannot re-delegate signing rights. For example, from $rk_{A\to B}$ and $rk_{B\to C}$, he cannot produce $rk_{A\to C}$.

9. *Temporary:* Whenever a party delegates some of her rights to another party, there is always the chance that she will either need or want to revoke those rights later on. Since it may not always be feasible for a delegator to change her public key after every revocation, we are interested in schemes that minimize revocation overhead even when the proxy is not fully trusted. Of course, if the re-signature proxy is trusted, then we can realize temporary delegations for any re-signature scheme by issuing the appropriate instructions to the proxy.

Of course, one would want to formalize a definition with the right combination of these

additional properties for a given application. Our purpose here is start a discussion of what additional properties may be desired and achieved.

*Fail-Stop Proxy Re-Signatures.* An interesting possible extension is to the basic schemes we've been discussing is to consider fail-stop proxy re-signatures. In regular *fail-stop* signatures as introduced by Pfitzmann and Waidner [162], the signer can prove to third parties that a forgery is in fact a forgery and not a signature that she created. Similarly, in some applications, it may be desirable for a signer to be able to distinguish (and prove) which signatures she created from those created by the proxy. This is especially useful if the proxy is suspected of abusing its authority. It is conceivable that the signer's secret information could be used to distinguish signature types, and thus, the transparency property might be simultaneously realizable.

### 4.3.2 Remarks on the BBS Scheme

In the BBS scheme, the re-signature key is necessarily public since it is not possible for the proxy to keep it secure. Indeed, it is enough to just observe a valid signature and its transformation to be able to retrieve the re-signature key and then impersonate the proxy itself. Moreover, both parties are forced to mutually share their secret keys as these can be easily computed from the (public or exposed) re-signature key and one of the parties' secret key. Let's have a look, next, at the details of the BBS scheme so that it is easier to discuss its limitations.

**BBS Re-Signatures.** Recall the BBS proxy re-signature scheme [21].

- **System Parameter Generation** (SysGen): On input the security parameter $1^k$, output *params* containing $(q, g, \mathbb{G}, H)$, where $g$ is a generator of a group $\mathbb{G}$ of order $q = \Theta(2^k)$ and $H$ is a hash function mapping strings in $\{0,1\}^*$ to elements in $\mathbb{Z}_q$.

- **Key Generation** (KeyGen): On input the parameters *params*, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = g^a$ and $sk = a$. (Assume *params* are also included.)

- **Re-Signature Key Generation** (ReKey): On input two secret keys $sk_A = a, sk_B = b$, output the re-signature key $rk_{A \to B} = a/b \pmod q$. (BBS did not describe a protocol for this, but we will in Section 4.3.3.)

- **Sign** (Sign): On input a secret key $sk = a$ and a message $m$, select random elements $x_1, \ldots, x_k \in \mathbb{Z}_q^k$. Then, compute $r = (g^{x_1}, \ldots, g^{x_k})$ and extract $k$ pseudorandom bits $b_1, \ldots, b_k$ from the output of $H(r)$. Finally, output the signature $\sigma = (r, s)$, where $s = (s_1, \ldots, s_k)$ and each $s_i = (x_i - m \cdot b_i)/a \pmod q$.

- **Re-Sign** (ReSign): On input a re-signature key $rk_{A \to B}$, a public key $pk_A$, a signature $\sigma$, and a message $m$, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If $\sigma$ verifies, set $r' = r$ and $s'_i = s_i \cdot rk_{A \to B} \pmod q$, and output the signature $\sigma_B = (r', s')$, where $s' = (s'_1, \ldots, s'_k)$; otherwise, output the error message $\perp$.

- **Verify** (Verify): On input a public key $pk_A$, a message $m$, and a purported signature $\sigma = (r, s)$, compute $H(r)$ and extract pseudorandom bits $b_1, \ldots, b_k$. For each $g^{x_i} \in r$ and $s_i \in s$, check that $(pk_A)^{s_i} = g^{x_i}/g^{m \cdot b_i}$. If all check pass, output 1; otherwise output 0.

Given any pair of signatures $(\sigma_A, \sigma_B)$, where $\sigma_A$ was created by the Sign algorithm and $\sigma_B$ is the result of the ReSign algorithm on $\sigma_A$, anyone can compute the re-signature key $rk_{A \to B}$ as follows: Let $\sigma_A = (r, s)$ and $\sigma_B = (r, s')$ be signatures as described above, where $s = (s_1, \ldots, s_k)$ and $s' = (s'_1, \ldots, s'_k)$. Then, $s'_1/s_1 = a/b = rk_{A \to B} \pmod q$ and thus, anyone can become a rogue proxy. Moreover, from $rk_{A \to B} = a/b$, Alice (resp., Bob) can compute Bob's (resp., Alice's) secret key as $(a/b)^{-1} \cdot a \bmod q = b$.

Although the BBS scheme satisfies their security definition (the scheme is called *symmetric* in [21]), it is clearly inadequate for many interesting applications, including those suggested in the original BBS paper [21].

**Alternatives?** Finding suitable and secure proxy re-signature schemes required a substantial effort. Natural extensions of several standard signatures were susceptible to the sort of problems above. To illustrate the intuition behind this, consider a naive proxy re-signature construction based on the popular Schnorr [169] signature. Let $||$ denote concatenation. Recall Schnorr signatures with key pairs of the form $(pk_A, sk_A) = (g^a, a)$ and signatures of the form $(r, s) = (g^k, a \cdot H(m||r) + k)$. One might think to give the proxy $rk_{A \to B} = (b - a)$, so that it can re-sign messages as $(r', s') = (r, s + rk_{A \to B} \cdot H(m||r))$. However, as in the BBS scheme, anyone can compute $rk_{A \to B} = (s' - s)/H(m||r) = (b - a)$ from a signature $(r, s)$ and its re-signature $(r, s')$ on a message $m$.

We also considered a few new schemes (e.g. [54]), but it was not obvious how to turn them into proxy re-signatures that provided a satisfying subset of the features described in Section 4.3.1.

### 4.3.3 $\Omega_{bi}$: A Multi-Use Bidirectional Scheme

We now present a new proxy re-signature scheme, denoted $\Omega_{bi}$, using the signatures due to Boneh, Lynn, and Shacham [35, 34].

This scheme requires a bilinear map, as discussed in Chapter 2. For simplicity, we will present this scheme for a *double isomorphism* bilinear map $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ (e.g., implemented using a supersingular curve) where we treat $\mathbb{G}_1 \equiv \mathbb{G}_2$ and, as before, $\langle g_1 \rangle = \mathbb{G}_1$ has prime order $q$. We can also use a *single-or-none isomorphism* bilinear map to obtain short signatures, as we discuss at the close of this section. (Recall that by "short" we mean that signatures of 171 bits have roughly the same security as 1024 bit RSA signatures [35].)

- **System Parameter Generation** (SysGen): On input the security parameter $1^k$, run Bilinear_Setup$(1^k) \to (\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ to obtain parameters for the bilinear map. In addition, the global parameters *params* include a hash function $H$ from arbitrary strings to elements in $\mathbb{G}_1$ as defined by Boneh and Franklin [29]. Output *params*.

- **Key Generation** (KeyGen): On input the parameters *params*, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = g_1^a$ and $sk = a$. (Assume *params* are included too.)

- **Re-Signature Key Generation** (ReKey): On input two secret keys $sk_A = a$, $sk_B = b$, output the re-signature key $rk_{A \to B} = b/a \pmod q$. The three-party protocol between Alice with $sk_A$, Bob with $sk_B$, and the proxy operates as follows:

  1. the proxy sends a random $r \in \mathbb{Z}_q$ to Alice,
  2. Alice sends $r/a$ to Bob,
  3. Bob sends $b \cdot (r/a)$ to the proxy,
  4. the proxy computes $(b \cdot r)/(a \cdot r) = b/a$.

  We are clearly assuming private and authenticated channels and no collusion. Bidirectional schemes make no security guarantees in the event of collusion.

- **Sign** (Sign): On input a secret key $sk = a$ and a message $m$, output $\sigma = H(m)^a$.

- **Re-Sign** (ReSign): On input a re-signature key $rk_{A \to B}$, a public key $pk_A$, a signature $\sigma$, and a message $m$, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If $\sigma$ does not verify, output $\perp$; otherwise, output $\sigma' = \sigma^{rk_{A \to B}}$.

- **Verify** (Verify): On input a public key $pk_A$, a message $m$, and a purported signature $\sigma$, output 1 if $\mathsf{e}(g_1, \sigma) = \mathsf{e}(pk_A, H(m))$ and 0 otherwise.

Notice that the re-signature keys for this scheme are the same as those in the BBS scheme: a ratio of the secret keys. In this scheme, however, the re-signature key *cannot* be computed from an original signature and its re-signed value.. Although this scheme is very simple, proving its security takes some work. We will first prove the following theorem and then discuss some of the nice properties of this scheme.

**Theorem 4.2 (Security of $\Omega_{bi}$).** *In the random oracle model, any adversary that $\varepsilon$-breaks the bidirectional proxy re-signature scheme $\Omega_{bi}$ requesting at most $q_H$ direct hash queries, $q_S$ signature queries, and $q_K$ public keys can be used to*

$$\left(\varepsilon \cdot \frac{1}{q_K} \cdot \frac{1}{q_H} \cdot \left(1 - \frac{1}{q - q_H - q_S}\right)\right) \text{-break}$$

*the Computational Diffie-Hellman (CDH) assumption in $\mathbb{G}_1$; that is, for random $g_1 \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_q^2$, given $(g_1, g_1^a, g_1^b)$, it is hard to compute $g_1^{a \cdot b}$.*

*Proof.* The correctness of the scheme is easily observable. We show security in two parts.

**External Security:** For security, we show that any adversary $\mathsf{Adv}$ that can break the security of the above proxy re-signature scheme with non-negligible probability $\varepsilon$ after making at most $q_H$ direct hash queries, $q_S$ $\mathcal{O}_{sign}$ queries, and requesting $q_K$ public keys can be used to build an adversary $\mathsf{Adv}'$ that solves the CDH problem in $\mathbb{G}_1$ with probability roughly

$\varepsilon/(q_H \cdot q_K)$. On input $(g_1, g_1^a, g_1^b)$, the CDH adversary $\mathsf{Adv}'$ simulates a proxy re-signature security game for $\mathsf{Adv}$ as follows:

*Public keys:* As $\mathsf{Adv}$ requests the creation of system users, $\mathsf{Adv}'$ guesses which one $\mathsf{Adv}$ will attempt a forgery against. Without loss of generality, we denote the target public key as $pk_t$ and set it as $pk_t = g_1^a$, for all other public keys requested set $pk_i = g_1^{x_i}$ for a random $x_i \in \mathbb{Z}_q$. Let the total number of public keys requested be $q_K$.

*Oracle Queries:* There are three types of queries that $\mathsf{Adv}'$ must answer: the hash function $H$, the signature oracle $\mathcal{O}_{sign}$, and the re-signature oracle $\mathcal{O}_{resign}$.

- For each query to $H$ on input $m_i$, check if there is an entry in $T_H$. If so, output the corresponding value, otherwise guess if $m_i$ is the message $m^*$ that $\mathsf{Adv}$ will attempt to use in a forgery. If $m_i = m^*$, output $g^b$; otherwise, select a random $y_i \in \mathbb{Z}_q$ and output $g^{y_i}$. Record the pair $(m_i, y_i)$ in table $T_H$ for each $m_i \neq m^*$.

- For each query to $\mathcal{O}_{sign}$ on input $(j, m_i)$, if $j \neq t$, check if there is an entry for $m_i$ in $T_H$. If so retrieve $y_i$ from $T_H$; otherwise make a new entry in $T_H$ for $(m_i, y_i)$ for random $y_i$. Then, return the signature $g_1^{y_i \cdot x_j}$.

  If $j = t$ and $m_i \neq m^*$, return the signature $(g_1^a)^{y_i}$; otherwise, abort.

- For each query to $\mathcal{O}_{resign}$ on input $(i, j, m_k, \sigma)$, if $\mathsf{Verify}(pk_i, \sigma, m_k) \neq 1$, output $\perp$. Otherwise, output $\mathcal{O}_{sign}(j, m_k)$ (via calling oracle $\mathcal{O}_{sign}$).

*Forgery:* At some point $\mathsf{Adv}$ must output a purported forgery $(j, m, \sigma)$. If $t \neq j$, then $\mathsf{Adv}'$ guessed the wrong target user and must abort. If $\mathsf{Verify}(pk_j, \sigma, m) \neq 1$ or $(m, \sigma)$ is the result of any $\mathcal{O}_{sign}$ or $\mathcal{O}_{resign}$ query, adversary $\mathsf{Adv}$ has failed, so $\mathsf{Adv}'$ also aborts. Otherwise, $\mathsf{Adv}'$ outputs $\sigma = H(m^*)^a = g_1^{a \cdot b}$ as the proposed CDH solution.

First, we analyze how well $\mathsf{Adv}'$ simulates the world for $\mathsf{Adv}$. The responses from the hash oracle $H$ are uniformly distributed. The responses of signing oracles $\mathcal{O}_{sign}$ and $\mathcal{O}_{resign}$ are correct, except when $\mathsf{Adv}'$ incorrectly guesses the target user $t$ or the message $m^*$ on which $\mathsf{Adv}$ will forge.

The probability that $\mathsf{Adv}'$ will guess the target user correctly is $1/q_K$. The probability that $\mathsf{Adv}'$ will guess the forged message $m^*$ is $1/q_H$ conditioned on the fact that $\mathsf{Adv}$ queries the hash function for $m^*$. The probability that $\mathsf{Adv}$ can correctly predict $H(m^*)$ without querying $H$ is $1/(q - q_H - q_S)$, which is negligible ($q$ is the order of $\mathbb{G}_1$). Recall that since the scheme is deterministic, it trivially satisfies the property that $\mathsf{Adv}$ cannot produce a *new* signature on a previously signed message (i.e., $\mathcal{O}_{sign}$ will *not* be forced to abort when the simulator correctly guesses the target user and message.) Thus, we conclude that if $\mathsf{Adv}$ forges with probability $\varepsilon$, then $\mathsf{Adv}'$ solves CDH with probability $\varepsilon \cdot (1/q_K) \cdot (1 - 1/(q - q_H - q_S))/q_H$.

**(Bidirectional) Internal Security:** For bidirectional schemes, internal security refers only to *Limited Proxy* security: that is, a guarantee that the proxy cannot use its re-signature keys to sign on behalf of honest users. We now show that a rogue proxy $\mathsf{Adv}$ that can forge with probability $\varepsilon$ can be used to build an adversary $\mathsf{Adv}'$ that solves the CDH problem with

probability roughly $\varepsilon/q_H$. On input $(g_1, g_1^a, g_1^b)$, the CDH adversary $\mathsf{Adv}'$ simulates a proxy re-signature security game for $\mathsf{Adv}$ as follows:

*Public keys:* For each key, choose a random $x_i \in \mathbb{Z}_q$, and set $pk_i = (g_1^a)^{x_i}$.

*Oracle Queries:* There are three types of queries that $\mathsf{Adv}'$ must answer: the hash function $H$, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key generation oracle $\mathcal{O}_{rekey}$.

- For each query to $H$ on input $m_i$, check if there is an entry in $T_H$. If so, output the corresponding value, otherwise guess if $m_i$ is the message $m^*$ that $\mathsf{Adv}$ will attempt to use in a forgery. If $m_i = m^*$, output $g_1^b$; otherwise, select a random $y_i \in \mathbb{Z}_q$ and output $g_1^{y_i}$. Record the pair $(m_i, y_i)$ in table $T_H$ for each $m_i \neq m^*$.
- For each query to $\mathcal{O}_{sign}$ on input $(j, m_i)$, if $m_i \neq m^*$, return the signature $(pk_j)^{y_i}$ where $(m_i, y_i) \in T_H$; else, abort. (If $m_i$ does not appear in $T_H$, choose a random $y_i$ and enter the pair into $T_H$.)
- For each query to $\mathcal{O}_{rekey}$ on input $(i, j)$, if $i = t$ or $j = t$, abort; else, return $rk_{i \to j} = (x_j/x_i)$.

*Forgery:* At some point $\mathsf{Adv}$ must output a purported forgery $(j, m, \sigma)$. If $\mathsf{Verify}(pk_j, \sigma, m)$ does not accept or $(m, \sigma)$ is the result of any $\mathcal{O}_{sign}$ or $\mathcal{O}_{resign}$ query, adversary $\mathsf{Adv}$ has failed, so $\mathsf{Adv}'$ aborts. Otherwise, $\mathsf{Adv}'$ outputs $\sigma^{1/x_j} = H(m^*)^a = g_1^{a \cdot b}$ as the proposed CDH solution.

The final analysis is simpler than before. In the case that $\mathsf{Adv}'$ correctly guesses the target user $t$ and the message to forge $m^*$, $\mathsf{Adv}'$ perfectly simulates the world for $\mathsf{Adv}$. Thus, CDH is solved with probability $\varepsilon \cdot (1 - 1/(q - q_H - q_S))/q_H$. □

**Discussion of Scheme $\Omega_{bi}$.** This scheme is simple and highly efficient. Basing it on the BLS construction [35, 34], the signature is under 200 bits while only one exponentiation is needed to sign or resign. This efficiency, combined with its multi-use functionality, makes it highly attractive for many network authentication applications because it allows for long signing chains. It is also bidirectional, which means that the re-signing key $rk_{A \to B}$ can be used to transform Alice's signatures into Bob's or vice versa. Bidirectionality is desirable for some applications, but a potential security risk in others. (The construction of a scheme that is both multi-use and unidirectional remains an open problem.) Transparency is guaranteed by the fact that the signature algorithm is deterministic and, since each user just stores one signing key, the scheme is also key optimal.

**Using General Bilinear Maps for $\Omega_{bi}$.** In our presentation of $\Omega_{bi}$, we assumed a double-isomorphism bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, which, based on current knowledge, would limit us to implementations using supersingular curves. We could allow for more candidate implementations using ordinary curves and *shorter* signatures (see Chapter 2) by assuming a bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with arbitrary isomorphisms; that is, we will not use any of the isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ in our scheme or its proof, but neither will our scheme or proof break if efficient isomorphisms exist. To use a general bilinear map, the following must occur:

1. the signatures and the range of hash function $H$ are set in $\mathbb{G}_2$,
2. the public keys are set in $\mathbb{G}_1$.
3. the complexity assumption changes from CDH in $\mathbb{G}_1$ to Co-CDH in $(\mathbb{G}_1, \mathbb{G}_2)$; that is, for random $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_q^2$, given $(g_1, g_1^a, g_2, g_2^b)$, it is hard to compute $g_2^{a \cdot b}$.

### 4.3.4 $\Omega_{uni}$ and $\Omega^*_{uni}$: Single-Use Unidirectional Schemes

We now present two proxy re-signature schemes, denoted $\Omega_{uni}$ and $\Omega^*_{uni}$ respectively. These schemes are unidirectional since the re-signature key $rk_{A \to B}$ can be used to change Alice's signatures into Bob's, but *not* vice versa. The schemes $\Omega_{uni}$ and $\Omega^*_{uni}$ differ in a single feature: In $\Omega_{uni}$, the re-signature key is made public or is easily computable by anyone, while in $\Omega^*_{uni}$, this key is secret and stored at the proxy. Applications of unidirectional schemes with both public and private re-signature keys will be provided in Section 4.4.

Each signer has a strong and weak secret key associated with their single public key. The intuition behind the unidirectional schemes is to use the re-signature key to transform Alice's signatures computed under her strong secret into signatures computed under Bob's weak secret. Signatures under any "weak secret" cannot be converted, which makes the schemes single-use. Notice that we must deal with scenarios where signatures from several users are converted into signatures from a single user (and vice-versa). This rules out trivial solutions based on bidirectional schemes with multiple public keys per user.

#### $\Omega_{uni}$ with Public Re-Signature Key

This scheme requires a bilinear map, just as in the previous section. For simplicity, we will present this scheme for a *double isomorphism* bilinear map $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ (e.g., implemented using a supersingular curve) where we treat $\mathbb{G}_1 \equiv \mathbb{G}_2$ and, as before, $\langle g_1 \rangle = \mathbb{G}_1$ has prime order $q$. We can also use a *single isomorphism* bilinear map, which we will discuss at the close of this section.

- **System Parameter Generation** (SysGen): On input the security parameter $1^k$, run $\mathsf{Bilinear_Setup}(1^k) \to (\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ to obtain parameters for the bilinear map. In addition, let the global parameters *params* include a hash function $H$ from arbitrary strings to elements in $\mathbb{Z}_q$ and a random generator $h_1 \in \mathbb{G}_1$. Output *params*.
- **Key Generation** (KeyGen): On input the parameters *params*, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = (g_1^a, h_1^{1/a})$ and $sk = a$. We think of $sk = a$ as the "strong" secret, and the value $h_1^a$ as the "weak" secret.

  (Note: a user need only output the second component $h_1^{1/a}$ of her public key if she wishes to receive delegations; it does not play a role in signature verification. Also, the second component can be verified against the first as $\mathsf{e}(g_1^a, h_1^{1/a}) = \mathsf{e}(g_1, h_1)$, so only the $g_1^a$ portion need be certified.)

- **Re-Signature Key Generation** (ReKey): On input a public key $pk_A = (g_1^a, h_1^{1/a})$ and a secret key $sk_B = b$, Bob outputs the re-signature key $rk_{A\to B} = h_1^{b/a}$. Let $rk_{A\to B}$ be public.

- **Sign** (Sign): On input a secret key $sk = a$ and a message $m$, select a random $k \in \mathbb{Z}_q$, set $r = h_1^k$, $s = a \cdot (H(m||r) + k) \pmod q$; output the pair $\sigma = (r, s)$. We call a signature of this form a *first-level* signature. It should be tagged as such.

  Optionally, the signer could choose to output a signature that could not be re-signed, where the last element of $\sigma$ is set to $h_1^{a\cdot H(m||r)+a\cdot k}$ instead. We call this a *second-level* signature. It should be tagged as such.

- **Re-Sign** (ReSign): On input a re-signature key $rk_{A\to B}$, a public key $pk_A$, a (first-level) signature $\sigma$, and a message $m$, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If $\sigma = (r, s)$ does not verify, output $\perp$; otherwise, set $r' = r$ and $s' = (rk_{A\to B})^s$, and output $\sigma' = (r', s')$. The proxy always tags this as a first-level signature.

- **Verify** (Verify): On input a public key $pk = (g_1^a, h_1^{1/a})$, a message $m$, and a purported signature $\sigma = (r, s)$, if $\sigma$ is tagged as a first-level signature, set $s' = h_1^s$; otherwise, set $s' = s$. Output 1 if $\mathsf{e}(g_1, s') = \mathsf{e}(g_1^a, r \cdot h_1^{H(m||r)})$ and 0 otherwise.

**Theorem 4.3 (Security of $\Omega_{uni}$).** *In the random oracle model, any adversary that $\varepsilon$-breaks the unidirectional proxy re-signature scheme $\Omega_{uni}$ requesting at most $q_H$ hash queries can be used to*

$$\left(\varepsilon - \frac{\varepsilon \cdot q_H + 1}{q}\right)^2 \text{-break}$$

*the CDH or 2-DL assumptions in $\mathbb{G}_1$; the latter being that given $(g_1, g_1^a, g_1^{a^2})$, for random $g_1 \in \mathbb{G}_1$ and $a \in \mathbb{Z}_q$, it is hard to compute $a$.*

*Proof.* We argue security in two parts. Here, for clarity of exposition, we will use the well-known result [166] that CDH is equivalent to the *Square Diffie-Hellman* (sq-DH) problem in the same group. The sq-DH problem in $\mathbb{G}_1$ is to compute $g_1^{x^2}$ given $(g_1, g_1^x)$ for a random $g_1 \in \mathbb{G}_1$ and $x \in \mathbb{Z}_q$. Showing that sq-DH implies CDH is trivial. To see the other direction, suppose you are given CDH input $(g_1, g_1^x, g_1^y)$ and an sq-DH solver. Use the sq-DH solver to compute $A = g_1^{x^2}$, $B = g_1^{y^2}$, and $C = g_1^{(x+y)^2} = g_1^{x^2+2\cdot x\cdot y+y^2}$. From these values, it is easy to compute $\sqrt{C/(A \cdot B)} = g_1^{x\cdot y}$ for this group of known prime order.

**External Security:** In $\Omega_{uni}$, the re-signature keys are public by design. Thus, all users are considered proxies and there are no "external adversaries" to the system. Therefore, the external security property holds trivially.

**(Unidirectional) Internal Security:** For unidirectional schemes, internal security refers to:

1. *Limited Proxy Security:* protecting an honest delegator and delegatee from a rogue proxy,

2. *Delegatee Security:* protecting an honest delegatee against delegators colluding with the proxy,
3. *Delegator Security:* offering limited protection to an honest delegator against delegatees colluding with the proxy.

1. *Limited Proxy Security:* For security, we show that any proxy $\mathsf{Adv}$ that can break this security guarantee with non-negligible probability $\varepsilon$ can be used to build an adversary $\mathsf{Adv}'$ that solves the CDH (equivalently, sq-DH) problem in $\mathbb{G}_1$ with probability roughly $\varepsilon^2$.

*System Parameters:* On a sq-DH challenge $(g_1, g_1^x)$, the simulator $\mathsf{Adv}'$ outputs the system parameters as $g_1$ and $h = g_1^x$.

*Public keys:* As $\mathsf{Adv}$ requests the creation of system user $i$, $\mathsf{Adv}'$ chooses a random $c_i \in \mathbb{Z}_q$ and outputs $pk_i = (g_1^{x \cdot c_i}, g_1^{1/c_i}) = (g_1^{x \cdot c_i}, h^{1/(x \cdot c_i)})$. The virtual $sk_i$ is $(x \cdot c_i)$. The pair $(i, c_i)$ is saved.

*Oracle Queries:* There are three types of queries that $\mathsf{Adv}'$ must answer: the hash function $H$, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key generation oracle $\mathcal{O}_{rekey}$.

- For each query to $H$ on input $a_i$, check if there is an entry in $T_H$. If so, output the corresponding value, otherwise output a random value $y_i \in \mathbb{Z}_q$. Record the pair $(a_i, y_i)$ in table $T_H$. Let $q_H$ be the total number of queries to $H$.

- For each query to $\mathcal{O}_{sign}$ on input $(j, m_i)$, concoct the signature using control over the output of $H$ as follows. Select random values $s, y \in \mathbb{Z}_q$. Parse user $j$'s public key as $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Compute $r = (pk_j^{(2)})^s / h^y$. Check if $(m_i || r, ?)$ is already in $T_H$, if so abort; otherwise, output the first-level signature $(r, s)$. Record $(m_i || r, y)$ in table $T_H$.

- For each query to $\mathcal{O}_{rekey}$ on input $(i, j)$, generate the re-signature key $rk_{i \to j}$ by computing $(g_1^x)^{c_j / c_i}$. (This is equal to $h^{(x \cdot c_j)/(x \cdot c_i)} = h^{sk_j / sk_i}$.)

In the above, $\mathsf{Adv}'$ *almost* perfectly simulates the world for adversary $\mathsf{Adv}$, except for the possibility of aborting in oracle $H$. Aborting happens on collisions in $H$ which occur with probability at most $q_H/q$, where $q$ is the order of $\mathbb{G}_1$. Thus a simulation will end with $\mathsf{Adv}$ successfully forging some message with probability $\varepsilon \cdot (1 - q_H/q)$.

We will be applying the Reset Lemma due to Bellare and Palacio [19]. The Reset Lemma applies to any canonical three round proof protocol; that is, any transcript consisting of a prover's *commitment*, a verifier's *challenge*, and a prover's *response*. Our signature scheme $\Omega_{uni}$ is the result of applying the Fiat-Shamir heuristic [98] to such a three round protocol. This lemma upper-bounds the probability that a cheating prover can trick an honest verifier into accepting.

**Lemma 4.4 (Reset Lemma [19]).** *Let $P$ be a canonical three round proof protocol, $\mathcal{P}$ be the prover with state $p$, and $\mathcal{V}$ be the verifier with state $v$. Then the probability* $\mathsf{acc}(p, v)$ *that an honest $\mathcal{V}$ accepts after executing protocol $P$ with $\mathcal{P}$ is*

$$\mathsf{acc}(p, v) \leq \frac{1}{|\mathrm{ChSet}|} + \sqrt{\mathsf{reset}(p, v)}$$

*where* ChSet *is the verifier's challenge set and* $\mathsf{reset}(p, v)$ *is the probability that for the same random tape and commitment of the prover, the verifier accepts on two random challenges from* ChSet.

Applying the Reset Lemma where we already argued that $\mathsf{acc}(p, v) = \varepsilon \cdot (1 - q_H/q)$ and $|\text{ChSet}| = q$, we have $\mathsf{reset}(p, v) = (\varepsilon - (\varepsilon \cdot q_H + 1)/q)^2$. Thus, with this probability, Adv can produce two valid signature transcripts $(r, d_1, s_1)$ and $(r, d_2, s_2)$ for user $t$, and where $d_1$ and $d_2$ are two different random responses from $H$ on input $(m||r)$ for some $m$. When this occurs, Adv can solve sq-DH by computing and outputting:

$$\left(\frac{s_1}{s_2}\right)^{1/(c_i \cdot (d_1 - d_2))} = \left(\frac{h^{x \cdot c_i \cdot (d_1 + k)}}{h^{x \cdot c_i \cdot (d_2 + k)}}\right)^{1/(c_i (d_1 - d_2))} = h^{\frac{x \cdot c_i \cdot (d_1 + k - d_2 - k)}{c_i \cdot (d_1 - d_2)}} = h^x = (g_1^x)^x = g_1^{x^2}.$$

2. *Delegatee Security:* Recall that $\Omega_{uni}$ is a non-interactive scheme, meaning that Bob (the delegator) can compute a re-signature key $rk_{A \to B}$ from Alice's public key. Thus, intuitively "nothing is learned about Alice's secrets" from Bob and the proxy both seeing $rk_{A \to B}$.

More formally, Adv′ must be able to provide Adv with the secret keys of all delegators of a target user $pk_0$. Here Adv need only produce a second-level signature to "win" in the delegatee security game.

*System Parameters:* On a sq-DH challenge $(g_1, g_1^x)$, the simulator Adv′ outputs the system parameters as $g_1$ and $h = (g_1^x)^z$ for random $z \in \mathbb{Z}_q$.

*Public and Secret Keys:* Adv′ generates the following keys for Adv.

- For the delegatee, set the public key as $pk_0 = (g_1^x, h^{1/x} = g_1^z)$. For all other users, set as $pk_i = (g_1^{y_i}, h^{1/y_i} = g_1^{z/y_i})$ for a random $y_i \in \mathbb{Z}_q$.
- Output each delegator's secret key $sk_i$ as $y_i$.

*Oracle Queries:* There are three types of queries that Adv′ must answer: the hash function $H$, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key oracle $\mathcal{O}_{rekey}$.

- For each query to $H$ on input $x_i$, check if there is an entry in $T_H$. If so, output the corresponding value, otherwise output a random value $c_i \in \mathbb{Z}_q$. Record the pair $(x_i, c_i)$ in table $T_H$.
- For each query to $\mathcal{O}_{sign}$ on input $(j, m_i)$, select random values $s, c \in \mathbb{Z}_q$. Parse user $j$'s public key as $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Compute $r = (pk_j^{(2)})^s / h^c$. Check if $(m_i||r, ?)$ is already in $T_H$, if so abort; otherwise, output the first-level signature $(r, s)$. Record $(m_i||r, c)$ in table $T_H$.
- For each query to $\mathcal{O}_{rekey}$ on input $(0, i)$, compute each re-signature key $rk_{0 \to i}$ as $(h^{1/x})^{y_i}$. All other allowed keys $rk_{i \to j}$ (i.e., $j \neq 0$) Adv can compute himself given the secret keys.

Adversary $\mathsf{Adv}'$ simulation of the world for $\mathsf{Adv}$ succeeds with the same probability as before, thus we apply the Reset Lemma [19], and from the two second-level signature transcripts compute $h^x = g_1^{x^2}$. (Note that $\mathsf{Adv}'$ only aborts during $\mathcal{O}_{sign}$ in the unlikely event that a collision occurs; unlike the proof of Theorem 4.2, it does not depend on *which* message is being signed. Thus, $\mathsf{Adv}'$ succeeds even when $\mathsf{Adv}$ produces a new forgery for a message $\mathsf{Adv}'$ was already asked to sign.)

3. *Delegator Security:* $\mathsf{Adv}'$ must now be able to provide $\mathsf{Adv}$ with the secret keys of all delegatees of a target user $pk_0$. Since $\mathsf{Adv}$ must produce a first-level signature to "win" in the delegator security game, we actually base this property on the 2-DL assumption that given $(g_1, g_1^x, g_1^{x^2})$, for random $g_1 \in \mathbb{G}_1$ and $x \in \mathbb{Z}_q$, it is hard to compute $x$.

*System Parameters:* On a 2-DL challenge $(g_1, g_1^x, g_1^{x^2})$, the simulator $\mathsf{Adv}'$ outputs the system parameters as $g_1$ and $h = (g^x)^z$ for random $z \in \mathbb{Z}_q$.

*Public and Secret Keys:* $\mathsf{Adv}'$ generates the following keys for $\mathsf{Adv}$.

- For the delegator, set the public key as $pk_0 = (g_1^x, g_1^z)$. For all other users, set as $pk_i = (g_1^{c_i}, h^{1/c_i} = g_1^{z/c_i})$ for a random $c_i \in \mathbb{Z}_q$.
- Output each delegatee's secret key $sk_i$ as $c_i$.

*Oracle Queries:* There are three types of queries that $\mathsf{Adv}'$ must answer: the hash function $H$, the signature oracle $\mathcal{O}_{sign}$, and the re-signature key oracle $\mathcal{O}_{rekey}$.

- For each query to $H$ on input $d_i$ (this represents the message concatenated with the randomness), check if there is an entry in $T_H$. If so, output the corresponding value, otherwise output a random value $y_i \in \mathbb{Z}_q$. Record the pair $(d_i, y_i)$ in table $T_H$.
- For each query to $\mathcal{O}_{sign}$ on input $(j, m_i)$, select random values $s, y \in \mathbb{Z}_q$. Parse user $j$'s public key as $pk_j = (pk_j^{(1)}, pk_j^{(2)})$. Compute $r = (pk_j^{(2)})^s / h^y$. Check if $(m_i || r, ?)$ is already in $T_H$, if so abort; otherwise, output the first-level signature $(r, s)$. Record $(m_i || r, y)$ in table $T_H$.
- For each query to $\mathcal{O}_{rekey}$ on input $(i, 0)$, compute each re-signature key $rk_{i \to 0}$ as $(g_1^{x^2})^{z/c_i} = h^{x/c_i}$. All other keys $rk_{i \to j}$ the adversary $\mathsf{Adv}$ can compute himself given the secret keys.

The simulation succeeds with the same probability as before, thus we apply the Reset Lemma [19], and from the two first-level signature transcripts compute $sk_0 = x$.

□

### $\Omega^*_{uni}$ with Private Re-Signature Key

In $\Omega_{uni}$, re-signature keys are *public*. However, this does not render the system as vulnerable as the BBS scheme, since at least the delegatee remains secure. For many of the applications we will shortly discuss in Section 4.4, our schemes $\Omega_{bi}$ and $\Omega_{uni}$ will be sufficient. However,

it would also be desirable to have a unidirectional scheme where the proxy key can be kept private. We briefly propose how one might consider naturally modifying $\Omega_{uni}$ into a new scheme $\Omega^*_{uni}$ to achieve these properties. The setup and global parameters hold from $\Omega_{uni}$. The following algorithms change:

- **Re-Signature Key Generation** (ReKey*): The re-signature key is $rk_{A\to B} = h_1^{b/a}$ as before, plus the proxy stores $pk_B$. The proxy keeps $rk_{A\to B}$ private.

- **Re-Sign** (ReSign*): On input a re-signature key $rk_{A\to B}$, a public key $pk_A$, a (first-level) signature $\sigma$, and a message $m$, check that $\mathsf{Verify}(pk_A, m, \sigma) = 1$. If $\sigma = (r, s)$ does not verify, output $\perp$; otherwise choose a random $w \in \mathbb{Z}_q$, set $r' = r$, $s' = (rk_{A\to B})^{s\cdot w}$, $t' = (pk_B^{(1)})^w$, and generate a signature proof of knowledge $u'$ of the discrete logarithm of $t'$ for base $pk_B^{(1)}$ (i.e., the first part of Bob's public key) on message $(r', s', t')$ using a new global hash function $\hat{H} : \{0,1\}^* \to \mathbb{Z}_q$. (This last step can be efficiently done using Schnorr's technique [169].) Output $\sigma' = (r', s', t', u')$.

- **Verify** (Verify*): The verifier now checks: (1) the proof $u'$, and (2) $\mathsf{e}(g_1, s') = \mathsf{e}(t', r' \cdot h_1^{H(m||r')})$.

The scheme $\Omega^*_{uni}$ is a natural extension of $\Omega_{uni}$ and we conjecture its security is based on the same assumptions. We leave a formal analysis of it as a subject for future work.

**Discussion of Schemes $\Omega_{uni}$ and $\Omega^*_{uni}$.** The only conceptional difference between these two schemes is that in $\Omega_{uni}$ the re-signature key is necessarily public (i.e., it is prey to the Section 4.3.2 attack), while in $\Omega^*_{uni}$ the proxy can keep the re-signature key private. (The re-randomization added to ReSign* thwarts this attack.)

Even though the re-signature key $rk_{A\to B}$ in $\Omega_{uni}$ is public, which allows anyone to translate between signatures, it does not reveal any information about Alice's (delegatee) signing keys to *anyone* (because it was computed by Bob with her public key). This is important since the delegatee should not be taking on any security risk. Furthermore, no third party can use this key to sign arbitrary messages for Bob (delegator) – and Alice can only recover Bob's *weak* secret $h_1^b$. This does not give Alice any new signing capability that she didn't have before: Alice could sign on behalf of Bob anyway, either by herself ($\Omega_{uni}$) or jointly with the proxy ($\Omega^*_{uni}$). (We stress that Alice won't be able to generate Bob's first-level signatures in any case.)

Bob does run the risk, however, that Alice may publish $h_1^b$, allowing anyone to produce second-level signatures for Bob. Yet, such a risk for the delegator seems inevitable.

Both of these schemes are exclusively for single-use applications (i.e., a signature translated from Alice to Bob cannot be again translated from Bob to Carol). One such application is a company mail server turning employee's signatures into a group signature before public release. An interesting open problem is designing a scheme that is simultaneously unidirectional and multi-use.

Happily, these schemes are non-interactive since Bob only needs Alice's public key to delegate to her (i.e., $rk_{A\to B}$). One potential drawback is that the original and re-signed

values can be linked; that is, given a first-level signature pair $(r, s)$, the ReSign algorithm produces a new second-level signature pair $(r', s')$ (or $(r', s', t', u')$) with $r = r'$. Nevertheless, weak transparency is achieved because the delegator can also produce second-level signatures from first-level ones due to the fact that he knows the re-signature key.

**Using General Bilinear Maps for $\Omega_{uni}$ and $\Omega^*_{uni}$.** In our presentations of $\Omega_{uni}$ and $\Omega^*_{uni}$, we assumed a double-isomorphism bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, which, based on current knowledge, would limit us to implementations using supersingular curves. We could allow for more candidate implementations using ordinary curves by assuming a bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with arbitrary isomorphisms; that is, we will not use any of the isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ in our scheme or its proof, but neither will our scheme or proof break if efficient isomorphisms exist. To use a general bilinear map, the following must occur:

1. the global parameter $g_1$ and the certified part (first half) of the public key $g_1^a$ are set in $\mathbb{G}_1$,

2. the global parameter $h_2$, the second half of the public key $h_2^{1/a}$, and both halves $(r, s)$ of second-level signatures are set in $\mathbb{G}_2$ (for first-level signatures, $s$ is in $\mathbb{Z}_q$),

3. the complexity assumption changes from CDH and 2-DL in $\mathbb{G}_1$ to Co-CDH and 2-DL in $(\mathbb{G}_1, \mathbb{G}_2)$; where Co-CDH is equivalent to: for random $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a \in \mathbb{Z}_q$, given $(g_1, g_1^a, g_2, g_2^a)$, it is hard to compute $g_2^{a^2}$.

**Temporary Delegations (and Revocations) for $\Omega_{uni}$ and $\Omega^*_{uni}$.**

What if Bob only wishes to allow Alice's signatures to be turned into his, via giving $rk_{A \rightarrow B}$ to the proxy, for the span of one week? If the proxy is honest, Bob may simply issue it these instructions. However, if Bob does not fully trust the proxy, is his only option to change his public key? Fortunately, the answer is no. We can modify the scheme as follows. At each time period $i$, a trusted party broadcasts a new global parameter $h_{1,i} \in G_1$, which replaces $h_{1,(i-1)}$ in the signing, verification, and re-signature key generation algorithms. This links each signature of the form $(h_{1,i}^k, a \cdot (H(m||r + k)))$ and each re-encryption key of the form $h_{1,i}^{b/a}$ with a time period $i$. This method effectively revokes *all* re-signature keys at each new time period, but takes only a single broadcast value and leaves all *certified* public keys of the form $g_1^a$ valid. (One would, of course, have to update the second portion to accept delegations with each new time period.) This simple revocation technique was first proposed in the setting of proxy re-encryption [8] and we will discuss it in more detail in Chapter 5.

## 4.4 Applications

Blaze, Bleumer, and Strauss [21] suggested several interesting applications of proxy re-signatures relating to key management. We begin by taking a closer look at how proxy

re-signatures can help relieve some of the common key management headaches. Next, we explore a broader set of applications and show that proxy re-signatures can be used to form weak (but easy to manage) group signatures, a space-efficient proof that a certain path was taken in a graph (e.g., a packet followed a prescribed path through the network), and more.

### 4.4.1 Exploring BBS Key Management

BBS [21] pointed out that proxy re-signatures can be used to change or add new public key pairs to a system without obtaining new certificates, notably simplifying key management. Let us explore this idea in more detail.

**Certifying Keys is Expensive, Can We Share?** Since certification of new public keys is a procedure that can be expensive and time consuming, using proxy re-signatures is a way to *share* existing certificates. Signatures under new keys can be transformed into ones that can be verified with public keys that are already certified. Consider also that distribution of certificates may be difficult or impossible in certain environments. Proxy re-signatures could be used to mitigate (at least temporarily) this issue by transforming signatures into ones that can be verified with public keys already trusted by the verifier. We now present an example of certificate sharing.

**A Time to Share (using $\Omega_{uni}$ or $\Omega^*_{uni}$).** Consider the case where a set of public keys is embedded into a software product or a limited-storage device, such as a smartcard. In many cases, it would be convenient if operating systems came installed with the certified public keys of major companies. The drawback, however, is that it might then be difficult or cumbersome for a company to change or add new keys. For example, a large company may want to begin signing documents at a department, rather than a company-wide, level even though software has been shipped with only one company-wide verification key. To solve this dilemma, the company could set up a proxy which could translate between old and new keys or from a large set of keys to a smaller one, etc.

To see this, suppose that software was shipped with a single company verification key $pk_A$. The company could still create new verification keys for each department $pk_B, pk_C, pk_D$ and include these certified keys in the next software release. However, to temporarily remain backwards compatible with the old software, the company could also publish (or setup a semi-trusted proxy) with the re-signature keys $rk_{B\to A}$, $rk_{C\to A}$, $rk_{D\to A}$; thus the proxy could change any signature generated by departments $B$, $C$, or $D$ (which the old software would not recognize) and turn it into a company-wide signature under $A$ (which the old software will recognize).

**Where Previous Schemes Fail in These Applications.** Although (some form of) the applications presented above were proposed for the BBS proxy re-signature scheme [21], that scheme is bidirectional (among its other limitations). A unidirectional scheme, such as $\Omega_{uni}$ or $\Omega^*_{uni}$, is much better suited to sharing certificates. For example, one might allow new certificates to be converted into old ones for backwards compatibility, but not old ones into new!

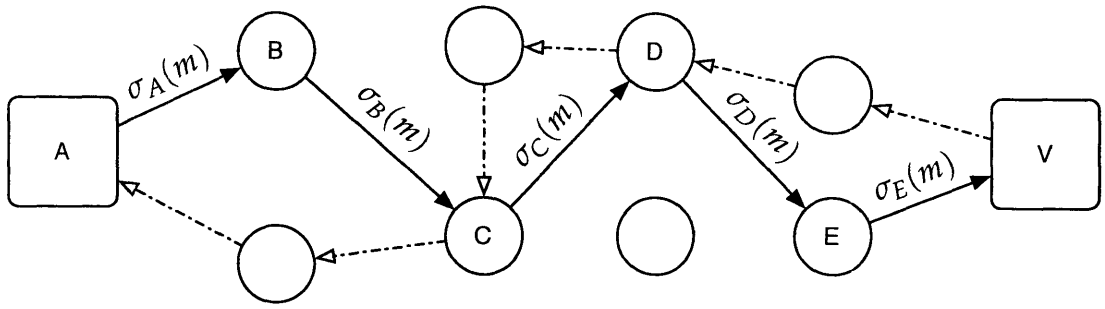


Figure 4-2: Unidirectional chains with multi-use proxy re-signature. Each node (e.g., $C$) holds only a proxy re-signature key (e.g., $rk_{C \to D}$) and *not* a signing key (e.g., $sk_C$). Thus, an adversary cannot inject a signed message into the chain by corrupting intermediate nodes.

### 4.4.2 New Applications

Armed with our new proxy re-signature schemes, we now show that they can be used as a space-efficient "proof" that a path was taken in a graph and as an easy-to-manage group signature scheme.

**Space-Efficient Proof that a Path was Taken (using $\Omega_{bi}$).** Proxy re-signatures are particularly useful when deployed with their multi-use capability, such as $\Omega_{bi}$. In particular, signatures can be converted in series as shown in Figure 4-2. Here, the signer $A$ generates the first signature on the message $m$, $\sigma_A(m)$, and the intermediate proxies convert it into the final signature $\sigma_E(m)$ through a series of transformations repeated in sequence. Such a structure can be used to prove that a certain item followed a specific path without taking any shortcuts. Indeed, we predict that this application will be the most general and useful application of proxy re-signatures.

The United States is currently in the process of adopting *E-passports* [128] – traditional passport documents capable of storing a limited number of digital signatures. Suppose Eve arrives in New York from her home country of Eden and shows US border patrol a signature $\sigma_A(m)$ from Eden that she is a citizen in good standing. The border patrol officer checks this signature and translates it into $\sigma_B(m)$, stating that Eve has passed the border patrol check. Eve next takes her passport to the customs officer. The customs officer need only verify Eve's passport against one public key – that of border patrol – and if it checks out and she passes customs, he can translate the signature into $\sigma_C(m)$, etc.

This system has many benefits. First, keeping only one signature around at a time reduces the space requirements on the limited memory passport – and also reduces the number of verification keys that checkpoints down the chain must store. Second, by only giving each checkpoint a re-signature key (e.g., giving customs $rk_{B \to C}$ instead of $sk_C$), corrupting a customs officer only allows Eve to skip the customs check – but she must still have gone through the initial checks by Eden and border patrol. Thus, Eve can – at best – skip one stop for each checkpoint officer that she compromises, and only for a message that Eden already authenticated.

Notice that although we use our bidirectional scheme (because it is "multi-use"), the chain in Figure 4-2 is actually *unidirectional* as long as the secret $sk_E$ corresponding to the last public key is kept secret. Thus, we can design only one way for Eve to get through the airport checks. Obviously, a scheme that is both multi-use and unidirectional would be ideal for this application, but no such scheme currently exists.

With a proper release of keys, proxy re-signatures can be used to non-repudiably prove that a message has traversed a graph via a legitimate path. Of course, one could employ *multi-signatures* [144] or *aggregate signatures* [31] to get a proof that certain nodes were visited, but both of these solutions require the verifier to have the verification keys of *all* of these nodes. Using proxy re-signatures, each node (including the final one) need only store and trust a *single* public key (the public key of the node proceeding it in the chain) – and yet each node has some real confidence that it is validating the *entire* path. Thus, we see another savings in key management.

Additionally in some cases, users may want the *privacy* that proxy re-signatures provide; that is, these signatures could simultaneously *authenticate and yet hide* the path traversed by the message in the network. This is not the case for multi or aggregate signatures.

**Easy to Manage Group Signatures (using $\Omega^*_{uni}$).** Proxy re-signatures can also be used to conceal identities or details of the structure of an organization. For instance, a corporate proxy sitting on a company's outgoing mail server could translate the individual signatures of its employees, which are perfectly valid signatures inside the organization, into signatures that can be verified with a single *corporate* public key. The proxy could (optionally) log which employee signed the message for internal auditing, but choose to keep that information company confidential. The interesting feature here is that even if the proxy is compromised *from the outside*, no signing keys are ever revealed which makes the proxy a less appealing target. The actual corporate secret key could be kept securely in a lock-box, and then the proxy, with only re-signing information, could sit out on the mail server. For accountability purposes, it is also advisable to employ unidirectional schemes (with private re-signature key) so that the proxy will not be able to generate members' signatures from existing corporate signatures.

## 4.5 Contributions

In this chapter, we formalized the proxy re-signature primitive of Blaze, Bleumer, and Strauss [21]. We pointed out several limitations of the BBS scheme and we provided new improved constructions. One of our schemes ($\Omega_{uni}$) allows the proxy to translate from Alice to Bob, but not vice versa. This is the only known construction to have this property since BBS proposed the concept in 1998 [21]. Our schemes are efficient and based on standard assumptions in the random oracle model, although they offer slightly different properties. Finally, we presented exciting applications of proxy re-signatures, including key management, (weak) group signatures, and short proofs that a valid path was taken in a graph. We are confident that proxy re-signatures have many additional applications beyond those mentioned here.

## 4.6 Open Problems

The holy grail for proxy re-signature schemes– a unidirectional *and* multi-use scheme –is not yet realized. Moreover, all schemes presented in this chapter require the random oracle model in their proof of security. Finding schemes of *similar efficiency* in the plain model is also an open problem.

The only proxy re-signatures that we are aware of are based on bilinear groups. (Some unpublished follow-up results to the work in this chapter is known to us, but those constructions are also bilinear-based, and less efficient.) It would be very interesting to know if this functionality can be efficiently realized without using bilinear maps.

There are also many additional variants of proxy re-signatures to consider. In Section 4.3.1, we proposed the idea of fail-stop re-signatures, where a signer could distinguish (and prove) signatures she created from those created for her by the proxy. Let us now consider two other variants.

One open question is whether or not proxy re-signature schemes can be built that translate from one type of signature scheme to another. For example, a scheme that translates Alice's Schnorr signatures into Bob's RSA-based ones. Kissner and Molnar [134] recently provided applications of such schemes, if indeed they exist.

A second variant would be to create *conjunctive* re-signature schemes where Alice can create a proxy key that allows the proxy, when given *both* a signature from Bob on a message $m$ and a signature from Carol on $m$, to output a signature from Alice on $m$.

Finally, on a more theoretical note, the relationship between proxy re-signatures and program obfuscation should be explored. Can re-signatures (and its variations) be formalized and realized as obfuscated programs?

# Chapter 5

# Proxy Re-Encryption

This chapter is an extended and modified version of the joint work with Giuseppe Ateniese (Johns Hopkins University), Kevin Fu (University of Massachusetts at Amherst), and Matthew Green (Johns Hopkins University) [8, 9]. In particular, the proxy encryption scheme $\Sigma_0$ in Section 5.3.2 is due to Giuseppe Ateniese and the implementation and performance evaluation in Section 5.4 were conducted by Kevin Fu and Matthew Green.

## 5.1 Introduction

In a *proxy re-encryption scheme*, a semi-trusted proxy is given some information which allows it to transform a ciphertext encrypted under Alice's public key into one that can be decrypted by Bob's secret key on the same message, but the proxy cannot decrypt ciphertexts for either Alice or Bob. That is, the proxy never sees the underlying plaintext.

There are many important applications of this primitive. In practice, it *frequently* happens that data encrypted under one key needs to be encrypted under a different key. Without care, the security of the system can be compromised during this transfer. For example, in March 2005, Apple's iTunes DRM (Digital Rights Management) was cracked by programmers who managed to steal the plaintext (song) made available during a translation from a ciphertext encrypted under a global key into a ciphertext encrypted under a key unique to each iPod [174]. If Apple had used the proxy re-encryption scheme described in this section, then the plaintext of the song would not have been available to steal. (Apple iTunes DRM used symmetric encryption, whereas we will be presenting an asymmetric scheme. Although, some hybrid combination would be feasible.)

Another real-life application is as simple as Alice wanting to forward her encrypted email to Bob while she is on vacation without revealing her secret key to either Bob or her mail server (acting as a proxy).

The first step in designing re-encryption schemes was taken by Mambo and Okamoto in 1997 [141], however their scheme was purely an efficiency improvement over traditional decrypt-and-then-encrypt approaches. No security was offered; that is, the proxy learned both the plaintext and Alice's secret key.

In 1998, Blaze, Bleumer, and Strauss [21] proposed the first proxy re-encryption scheme, where the plaintext and secret keys were kept hidden from the proxy. Let us briefly cover their El Gamal-based scheme. Suppose $g$ generates a group of prime order $q$. Let Alice own the public-secret key pair $(g^a, a)$ and Bob own the key pair $(g^b, b)$. An encryption of a message $m$ under Alice's public key has the form $(m \cdot g^k, g^{a \cdot k})$ for random $k \in \mathbb{Z}_q$. Then, the proxy is entrusted with the re-encryption key $(b/a \bmod q)$ for the purpose of diverting ciphertexts from Alice to Bob via computing $(m \cdot g^k, (g^{a \cdot k})^{b/a}) = (m \cdot g^k, g^{b \cdot k})$. This scheme is efficient and, on its own, the proxy does not learn message $m$, or secret keys $a$ or $b$.

The authors noted, however, that this scheme contains an inherent restriction: it is *bidirectional*; that is, the value $b/a$ can be used to divert ciphertexts from Alice to Bob and vice versa. Thus, this scheme is only useful when the trust relationship between Alice and Bob is mutual. This problem can be solved, in theory, by generating an additional, otherwise unused, key pair for the delegatee, but this introduces a suffocating amount of overhead for our practical applications in Section 5.4.

Indeed, BBS proposed finding a *unidirectional* proxy re-encryption as an open problem in their 1998. In this chapter, we present the first (and only known) unidirectional scheme.

The BBS scheme leaves open several other issues as well. Delegation in the BBS scheme is *transitive*, which means that the proxy alone can create delegation rights between two entities that have never agreed on this. For example, from the values $a/b$ and $b/c$, the proxy can re-encrypt messages from Alice to Carol. Another drawback to this scheme is that if the proxy and Bob collude, they can recover her secret key as $(a/b) \cdot b = a$!

In our scheme, delegation is non-transitive. Furthermore, the delegatee's secret key remains private even if the proxy and the delegator collude. We re-open the discussion of proxy re-encryption by providing three separate results:

1. We provide formal definitions and a security model that takes into account possible collusions overlooked by previous definitions.
2. We present the first unidirectional proxy re-encryption scheme.
3. We present a new application of proxy re-encryption to secure distributed storage including a brief discussion of an experimental implementation.

Incidentally, we also introduce a new cryptosystem based on bilinear maps where two decryption secrets are associated to a single public key. (We will use this cryptosystem again in Chapter 6.)

Let now us describe the main results of this chapter in more detail. These are very similar to the results we presented in Chapter 4 for re-signatures. Recall our summary of both of these suites of results in Chapter 1, Table 1.1.

**1. Formal Definitions and Security Model.** There is no formal definition of proxy re-encryption in the BBS paper [21]. In this chapter, we formalize the notion of a proxy re-encryption and provide a security model that captures the unidirectionality of the scheme. This definition also incorporates the lesson learned in Chapter 4 of having the re-encryption key safely stored at the proxy. We require only chosen-plaintext (CPA) security in our definitions, although they can be extended to chosen-ciphertext (CCA2) security. Obviously,

a CCA2-secure scheme would be very interesting, but we do not know of any scheme that meets such a definition.

In the CPA context, however, we are able to make some strong security guarantees. In particular, suppose Alice delegates to Bob, via a proxy, the ability to decrypt her ciphertexts. Even when the proxy and Alice collude, they cannot recover any information about Bob except his public key. More surprisingly, we show that even when the proxy and Bob collude, they can only recover a weak version of Alice's secret key – that only gives them the power to compute what Alice had already delegated to them. Secondly, our formal notion allows us to view the primitive abstractly, for easier reasoning about its applications in Section 5.4.

**2. Proxy Re-Encryption Constructions.** We present several efficient proxy re-encryption schemes, based on different complexity assumptions, that offer security improvements over earlier approaches. The primary advantage of our schemes is that they are unidirectional (solving the open problem of BBS from 1998) and do not require delegators to reveal all of their secret key to anyone – or even interact with the delegatee – in order to allow a proxy to re-encrypt their ciphertexts. In our schemes, only a limited amount of trust is placed in the proxy. The proxy is not able to decrypt the ciphertexts it re-encrypts, and we prove our schemes secure even when the proxy publishes all the re-encryption information it knows. This enables a number of applications that would not be practical if the proxy needed to be fully trusted.

Our El Gamal-based schemes require standard assumptions in bilinear groups (see Chapter 3). Unlike our re-signature constructions, here we will not require random oracles in our proofs of security.

**3. Applications.** Proxy re-encryption has many exciting applications in addition to the previous proposals [21, 92, 123, 183] for email forwarding, law enforcement, and performing cryptographic operations on storage-limited devices. In particular, proxy cryptography has natural applications to secure network file storage.

We provide the first empirical performance measurements of applications using proxy re-encryption. To demonstrate the practical utility of our proxy re-encryption schemes, we measure an implementation of proxy re-encryption used in a secure file system. Our system uses a centralized *access control server* to manage access to encrypted content stored on distributed, untrusted replicas. We use proxy re-encryption to allow for centrally-managed access control without granting full decryption rights to the access control server.

A secure file system is a natural application of proxy re-encryption because the system often assumes a model of untrusted storage.

A number of file systems build confidential storage out of untrusted components by using cryptographic storage [2, 20, 111, 131]. Confidentiality is obtained by encrypting the contents of stored files. These encrypted files can then be stored on untrusted file servers. The server operators can distribute encrypted files without having access to the plaintext files themselves.

In a single-user cryptographic file system, access control is straightforward. The user creates and remembers all the keys protecting content. Thus, there is no key distribution

problem. With group sharing in cryptographic storage, group members must rendezvous with content owners to obtain decryption keys for accessing files.

Systems supporting cryptographic storage such as the SWALLOW object store [164] or CNFS [121] assume an out-of-band mechanism for distributing keys for access control. Other systems such as Cepheus [102] use a trusted access control server to distribute keys.

The access control server model requires a great deal of trust in the server operator. Should the operator prove unworthy of this trust, he or she could abuse the server's key material to decrypt any data stored on the system. Furthermore, even if the access control server operator is trustworthy, placing so much critical key data in a single location makes for an inviting target.

In contrast, our system makes use of a semi-trusted access control server. We propose a significant security improvement to the access control in cryptographic storage, using proxy cryptography to reduce the amount of trust in the access control server. In our approach, keys protecting files, called "data keys", are stored encrypted under a master public key, using one of the schemes in Section 5.3. When a user requests a data key, the access control server uses proxy cryptography to directly re-encrypt the appropriate data key to the user without learning the data key in the process. Because the access control server does not itself possess the master secret, it cannot decrypt the data keys it stores. The master secret key can be stored offline, by a content owner who uses it only to generate the re-encryption keys used by the access control server. In Section 5.4, we describe our implementation and provide a performance evaluation of our constructions.

### 5.1.1 Related Work

This work should not be confused with *universal re-encryption* [119], often used in voting mix-nets, which re-randomizes ciphertexts instead of changing the public key under which they are encrypted. Proxy re-encryption [21] should also not be confused with the similar sounding *proxy encryption* [141, 92], where a proxy can change ciphertexts for Alice into ciphertexts that Bob can decrypt, but allows Alice to provide Bob with a new secret key instead of just translating to encryptions under his public key. As was the case for signatures in Chapter 4, proxy re-encryption forms a strict subset of proxy encryption schemes.

Recently, Dodis and Ivan [92] realized unidirectional *proxy encryption* for El Gamal, RSA, and an IBE scheme by sharing the user's secret key between two parties. They also solved the problem of the proxy alone assigning new delegation rights. In their unidirectional El Gamal scheme, Alice's secret key $s$ is divided into two shares $s_1$ and $s_2$, where $s = s_1 + s_2$, and distributed to the proxy and Bob. On receiving ciphertexts of the form $(m \cdot g^{s \cdot k}, g^k)$, the proxy first computes $(m \cdot g^{s \cdot k}/(g^k)^{s_1})$, which Bob can decrypt as $(m \cdot g^{s_2 \cdot k}/(g^k)^{s_2}) = m$. Although this scheme offers some advantages over the BBS approach, it introduces new drawbacks as well. These "secret-sharing" schemes do not change ciphertexts for Alice into ciphertexts for Bob in the purest sense (i.e., so that Bob can decrypt them with *his* own secret key), they delegate decryption by requiring Bob to store additional secrets that may in practice be difficult for him to manage. For example, in our file system in Section 5.4, the number of secrets a user must manage should remain constant regardless of the number

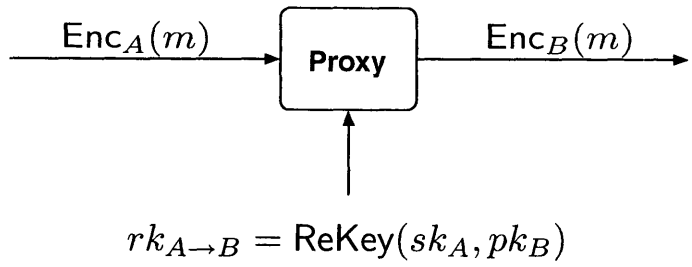


Figure 5-1: A high-level view of (unidirectional) proxy re-encryption. Here Alice is the delegator and Bob is the delegatee. The proxy obtains a re-encryption key $rk_{A\to B}$ from Alice, and can then translate ciphertexts for Alice, denoted $\mathsf{Enc}_A(m)$, into ciphertexts for Bob, denoted $\mathsf{Enc}_B(m)$.

of files it accesses. Using either the BBS or our proxy re-encryption scheme, the number of secrets is one; while using the Dodis-Ivan approach the secrets grows linearly with the number of files accessed. One exception is the Dodis-Ivan IBE scheme [92] where the global secret that decrypts *all* ciphertexts is shared between the proxy and the delegatee. Thus, the delegatee need only store a single secret, but an obvious security problem is that when the proxy and any delegatee in the system collude, they can decrypt *everyone* else's messages.

The proxy encryption security model of Dodis and Ivan [92], however, can be applied to our algorithm with some modifications which we will discuss in Section 5.2.

Jakobsson [123] developed a quorum-based protocol where the proxy is divided into sub-components, each controlling a share of the re-encryption key; here, the keys of the delegator are safe so long as some of the proxies are honest. A similar approach was considered by Zhou, Mars, Schneider and Redz [183].

Our results can be viewed as contributing both to the set of key-insulated [90, 91, 93] and signcryption [3, 11, 182] schemes, where Alice may expose her secret key without needing to change her public key and/or use the same public key for encryption and signing purposes.

## 5.2 Definitions

Now, we define what a unidirectional proxy re-encryption scheme is, as partially illustrated in Figure 5-1, and what minimum security properties it should have. We compare our definition to a similar definition due to Dodis and Ivan [92], although their definition is for proxy encryption (not *re*-encryption). Following our definition, we discuss some of the short-comings and benefits of our definition in a series of remarks.

**Definition 5.1 (Unidirectional Proxy Re-encryption).** *A* unidirectional proxy re-encryption scheme *is a tuple of (possibly probabilistic) polynomial time algorithms* (SysGen, KeyGen, ReKey, {Enc}, ReEnc, {Dec}), *where the components are defined as follows:*

- The $\mathsf{SysGen}(1^k)$ algorithm takes as input the security parameter $1^k$ and returns the global system parameters *params*.

- The $\mathsf{KeyGen}(params)$ algorithm outputs a user keypair $(pk, sk)$. Let $pk$ and $sk$ contain the system parameters *params* so that we do not have to explicitly give them out again.

- The $\mathsf{ReKey}(sk_A^\dagger, pk_B)$ algorithm outputs a key $rk_{A\to B}$ for the proxy. The second input marked with a '†' may in some cases be replaced by the tuple $(rk_{A\to C}, sk_C)$; see Remark 5.2 for more.

- For all $\mathsf{Enc}_i \in \{\mathsf{Enc}\}$, which is possibly a singleton set, the $\mathsf{Enc}_i(pk, m)$ algorithm ouputs a ciphertext $C$.

- The $\mathsf{ReEnc}(rk_{A\to B}, C_A)$ algorithm outputs a ciphertext $C_B$.

- There exists a $\mathsf{Dec}_i \in \{\mathsf{Dec}\}$ such that when $C$ is a valid encryption on $m \in M$ under the public key corresponding to $sk$, then the $\mathsf{Dec}_i(sk, C)$ algorithm outputs a message $m$.

**Correctness.** Informally, a party holding a secret key $sk_A$ should always be able to decrypt ciphertexts encrypted under $pk_A$; while a party holding a secret key $sk_B$ should be able to decrypt $\mathsf{ReEnc}(rk_{A\to B}, C_A)$. $\{\mathsf{Enc}\}$ may contain multiple encryption algorithms; for example, having *first-level* encryptions that cannot be re-encrypted by the proxy; while *second-level* encryptions can be re-encrypted by the proxy and then decrypted by delegatees. This provides the sender with a choice *given the same public key* whether to encrypt a message only to Alice or to Alice and, say, her secretary. Whenever a re-encryption does take place, however, we require that the underlying plaintext remain *consistent* – i.e., Bob should get exactly what Alice was supposed to receive.[1]

More formally, let key pairs $(pk_A, sk_A)$ and $(pk_B, sk_B)$, generated according to $\mathsf{KeyGen}$, belong to parties $A$ and $B$, respectively, and let $rk_{A\to B}$ be generated according to $\mathsf{ReKey}$. Then, for all messages $m$ in the message space $M$, the following equations hold with probability one:

$$\forall\ \mathsf{Enc}_i \in \{\mathsf{Enc}\},\ \exists\ \mathsf{Dec}_j \in \{\mathsf{Dec}\},\quad \mathsf{Dec}_j(sk_A, \mathsf{Enc}_i(pk_A, m)) = m,$$
$$\exists\ \mathsf{Enc}_i \in \{\mathsf{Enc}\},\ \exists\ \mathsf{Dec}_j \in \{\mathsf{Dec}\},\quad \mathsf{Dec}_j(sk_B, \mathsf{ReEnc}(rk_{A\to B}, \mathsf{Enc}_i(pk_A, m))) = m.$$

**Security.** We provide a security definition similar to that of Dodis and Ivan [92]. Although the Dodis-Ivan definition is for proxy encryption (not *re*-encryption) the difference between these schemes is in the semantics and the number of secret keys per user, not in their security models.

Although their definition was for CCA2 security, they instead used CPA security for the El Gamal, RSA, and IBE-based schemes; for simplicity, we focus directly on CPA security. The first main difference between our definitions is that we consider the security of a user against a *group* of colluding parties; for example, the security of a delegator against the proxy and many delegatees, whereas the Dodis-Ivan definition focused on a single delegatee. Secondly, we discuss the system's security for circular delegation where the adversary watches

[1]Note, this only applies to ciphertexts that were honestly generated by the sender; no guarantee is implied in the case of malformed ciphertexts.

Alice and Bob delegate to each other. Finally, we provide a new guarantee for the delegator – even if the proxy and all delegatees collude, they can only recover a "weak" version of his secret key. We discuss some benefits and desirable extensions of this last feature in Remark 5.3.

**Standard Security.** The underlying cryptosystem (KeyGen, {Enc}, {Dec}) is semantically-secure [117] against *anyone* who has not been delegated the right to decrypt. We use subscript $T$ to denote the target user. Formally, for any non-zero $n \in \text{poly}(k)$, for all PPT algorithms Adv, $\mathsf{Enc}_j \in \{\mathsf{Enc}\}$, and $m_0, m_1 \in M_k^2$,

$$\begin{aligned}
\Pr[&params \leftarrow \mathsf{SysGen}(1^k),\\
&(pk_T, sk_T) \leftarrow \mathsf{KeyGen}(params), \{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(params)\}_{i\in[1,n]},\\
&(m_0, m_1, \alpha) \leftarrow \mathsf{Adv}^{\mathcal{O}_{rekey}(\star,\cdot)}(pk_T, \{(pk_i, sk_i)\}_{i\in[1,n]}), b \xleftarrow{R} \{0,1\},\\
&b' \leftarrow \mathsf{Adv}^{\mathcal{O}_{rekey}(\star,\cdot)}(\alpha, \mathsf{Enc}_j(pk_T, m_b)) : b = b'] < 1/2 + 1/\text{poly}(k)
\end{aligned}$$

where oracle $\mathcal{O}_{rekey}$ takes as input two distinct indexes $1 \leq i, j \leq n$ or $j = T$ (but $\star \neq T$), and returns the output of $\mathsf{ReKey}(sk_i, pk_j)$.

The above definition captures $T$'s security, even when the proxy (with knowledge of all the re-encryption keys) and a group of adversarial users (with knowledge of their own secret keys) collude against $T$ – provided that $T$ never delegated decryption rights to any adversarial user. This simultaneously captures the Limited Proxy and Delegatee Security mirrored in Definition 4.1 of re-signatures. We combine them here for simplicity, because we are only dealing with unidirectional re-encryption schemes.

We now turn our attention to what security can be guaranteed in the case that $T$ *does* delegate decryption rights to an adversarial user. Obviously, in this case, the adversary can simply decrypt and trivially win the game above. However, suppose that there exist one-way functions $f_1$ and $f_2$ such that $pk = f_2(f_1(sk))$. Then, we define here (and later prove) that just because $T$ delegates decryption rights to another party does not mean that $T$ necessarily surrenders his entire digital identity $sk$ in the case that all parties collude against him. Perhaps he gives up $f_1(sk)$ instead.

**Digital-Identity Security:** Suppose there exist one-way functions such that for all key-pairs generated by KeyGen, we have $pk = f_2(f_1(sk))$. We use subscript $T$ to denote the target user. For any non-zero $n \in \text{poly}(n)$ and all PPT algorithms Adv,

$$\begin{aligned}
\Pr[&params \leftarrow \mathsf{SysGen}(1^k),\\
&(pk_T, sk_T) \leftarrow \mathsf{KeyGen}(params), \{(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(params)\}_{i\in[1,n]},\\
&\alpha \leftarrow \mathsf{Adv}^{\mathcal{O}_{rekey}(\cdot,\cdot)}(pk_T, \{(pk_i, sk_i)\}_{i\in[1,n]}) : \alpha = sk_T] < 1/\text{poly}(k).
\end{aligned}$$

**Remark 5.2 (Transfer of Delegation Rights).** Unfortunately, achieving security based on the definition of the re-encryption key generation function ReKey as originally stated

(i.e., without including †) is very difficult to realize. We do not know of any such scheme, including the prior work of Dodis and Ivan [92], that does not succumb to the follow attack: *transfer of delegation rights*, where, on input $sk_B$, $sk_C$ and $rk_{A\to C}$, one can compute $rk_{A\to B}$. That is, if Alice delegates to Carol, then Carol, Bob and the proxy can collude to delegate to Bob. Thus, we modify the definition of ReKey to be executed with *either* the secret key of the delegator Alice $sk_A$ or with both a re-encryption key from Alice to Carol $rk_{A\to C}$ and Carol's secret key $sk_C$. This implies that Carol is *allowed* to transfer Alice's decryption capability. Arguably, this relaxed definition is not so damaging since Alice is already trusting Carol enough to delegate decryption rights to her.

**Remark 5.3 (On Digital-Identity Security).** At first glance, digital-identity security may seem very weak. All it guarantees is that an adversary cannot output a delegator's secret key $sk$. One might ask why this is useful. One motivation, mentioned above, stems from the fact that some proxy re-encryption schemes define two or more types of ciphertext, some of which may only be decrypted using the secret $sk$ (and not $f_1(sk)$). A scheme which provides digital-identity security will protect those ciphertexts even in the event that the proxy and delegatee collude. A second motivation comes from the fact that most standard signature schemes, such as El Gamal [97] and Schnorr [169], are actually proofs of knowledge of a discrete logarithm value, such as $sk_A = a \in \mathbb{Z}_q$, turned into a signature using the Fiat-Shamir heuristic [98]. Intuitively, if an adversary cannot output Alice's secret key, then the adversary cannot prove knowledge of it either. Thus, using a proxy re-encryption scheme that always hides $sk$ means that a user *may* be able to safely delegate decryption rights (via releasing $f_1(a) = g_1^a$) without delegating signing rights for the *same public key* $pk = f_2(f_1(a)) = \mathsf{e}(g_1, g_1)^a$.

Of course, a stronger definition here might require that even a colluding proxy and delegatee could not distinguish first-level ciphertexts belonging to the delegator. This is a promising direction to pursue.

**Remark 5.4 (On Adaptive Adversaries and Trusted Parties).** Two other limitations of this definition are: (1) we consider only static adversaries; that is, Adv is not allowed to adaptively choose which users to corrupt, and (2) we require a trusted key issuing entity; that is, users receive their keypairs from an authority. We pursue schemes without these limitations as future work. This first appears to be easy, while the second seems much harder.

### 5.2.1 Proxy Re-Encryption and Program Obfuscation

At the end of the proxy re-signature definition in Chapter 4.2, we considered a formalization based on program obfuscation, where a program allowed the proxy to re-sign for users. We decided, for the case of signatures, that one of our schemes ($\Omega_{uni}$) might be considered an obfuscation because it leaked the proxy key, while the more interesting scheme ($\Omega_{bi}$) that keeps this key private would not.

The question for us in this chapter is whether or not re-encryption programs can be obfuscated? Consider the following circuits that translate Alice's ciphertext $E_A(m)$ into Bob's $E_B(m)$. Assume the circuits for the algorithms (Enc, ReEnc, Dec) and the public keys are known to all.

1. *Circuit One:* hard-wire Alice's secret key $sk_A$ into the instructions: (1) run $\mathsf{Dec}(sk_A, E_A(m))$ to obtain $m$, (2) return the output of $\mathsf{Enc}(pk_B, m)$.

2. *Circuit Two:* hard-wire the re-encryption key $rk_{A\to B}$ into a circuit that computes the algorithm $\mathsf{ReEnc}(rk_{A\to B}, E_A(m))$.

Again, loosely speaking, circuit two is an obfuscation of circuit one, because Alice's secret key is already obfuscated in circuit two. However, it does not appear to meet the Barak et al. [13] definition of obfuscation for the same reasons that re-signature scheme $\Omega_{bi}$ did not. Our unidirectional re-encryption schemes $(\Sigma_1, \Sigma_2, \Sigma_{temp})$ do *not* have extractable re-encryption keys. That is, from seeing an original encryption and a re-encrypted message, the re-encryption key cannot be recovered. Thus, our unidirectional re-encryption schemes do not meet the (trivial) notion of obfuscation that our unidirectional re-signature scheme appears to meet. Moreover, the re-encryption keys are a deterministic function of two parties keys, and thus knowing a re-encryption key it is likely that the adversary can output some bit function of this key that the simulator, who doesn't know the key, cannot. We leave the resolution of these issues an interesting open research and return to the focus of this thesis.

## 5.3 Improved Proxy Re-Encryption Schemes

We begin our discussion of proxy re-encryption schemes by covering the properties of these schemes that are either necessary or simply desirable for practical applications. We then present two unidirectional proxy re-encryption schemes, denoted $\Sigma_1$ and $\Sigma_2$. The second scheme is secure under a standard complexity assumption, whereas the first scheme is slightly more efficient, but requires a new assumption. We also present a third scheme, $\Sigma_{temp}$, that allows for temporary delegations (i.e., revocations).

### 5.3.1 Additional Properties We Need and Want

To talk about "improvements," we need to get a sense of the benefits and drawbacks of previous schemes. Here is a list of, in our opinion, the most useful properties of proxy re-encryption protocols in addition to correctness and security.

1. *Unidirectional:* Delegation from $A \to B$ does not allow re-encryption from $B \to A$.

2. *Multi-use:* A message can be re-encrypted a polynomial number of times. That is, ciphertexts generated by either the Enc or ReEnc algorithms can be taken as input to ReEnc. In contrast, one might imagine weaker, *single-use* schemes where only ciphertexts generated by Enc can be inputs to ReEnc. For our applications so far, we found the multi-use property to be much more important for signatures than for encryption.

3. *Non-interactive:* Re-encryption keys can be generated by Alice using Bob's public key; no trusted third party or interaction is required. (Such schemes were called *passive* in BBS [21].)

4. *Proxy invisibility:* This is an important feature offered by the original BBS scheme. The proxy in the BBS scheme is *transparent* in the sense that neither the sender of an encrypted message nor any of the delegatees have to be aware of the existence of the proxy. That is, a re-encrypted ciphertext is drawn from the same distribution as original ciphertexts. Clearly, transparency is very desirable but it is achieved in the BBS scheme at the price of allowing transitivity of delegations and recovery of the complete secret keys of the participants. Our bilinear-based schemes, to be described shortly, offer a weaker form of transparency which we call *proxy invisibility*. Here, first-level and second-level ciphertexts are distinguishable. Re-encryptions must be first-level ciphertexts, while original ciphertexts can be either first or second level. (Recall that first-level ciphertexts cannot be re-encrypted.) Thus, the recipient is not certain if a re-encryption has occurred or not, so we say that the proxy is invisible.

As another way of saying it, we allow the sender to generate an encryption that can be opened only by the intended recipient (*first-level encryption*) or by any of the recipient's delegatees as well (*second-level encryption*). At the same time, we can ensure that any delegatee will not be able to distinguish a first-level encryption (computed under his public key) from a re-encryption of a ciphertext intended for another party (we are assuming that the encrypted message does not reveal information that would help the delegatee to make this distinction).

5. *Unlinkable:* In an *unlinkable* scheme, a re-encryption cannot be linked to the original ciphertext from which it was generated. This property should hold both for original ciphertexts generated under two different keys as well as original ciphertexts under the same key using different randomness.

6. *Original-access:* Alice can decrypt re-encrypted ciphertexts that were originally sent to her. In some applications, it may be desirable to maintain access to her re-encrypted ciphertexts; that is, re-encryption allows both Alice and Bob to decrypt the resulting ciphertext instead of just Bob. This is an inherent feature of the Dodis-Ivan schemes (since the proxy key is a share of the original); the BBS scheme and the bilinear schemes presented here can achieve this feature by adding an additional term to the ciphertext: for example, in BBS a re-encrypted ciphertext with original access looks like $(m \cdot g^k, g^{a \cdot k}, (g^{a \cdot k})^{b/a})$. This may impact proxy invisibility.

7. *Key optimal:* The size of Bob's secret storage is independent of the number of delegations he accepts. We call this a *key optimal* scheme. In the previous RSA-based schemes [92], the storage of both Bob and the proxy grows linearly with the number of delegations Bob accepts. This is an important consideration, since the safeguarding and management of secret keys is often a huge obstacle in practice.

8. *Collusion-"safe":* One drawback of all previous schemes is that by colluding, Bob and the proxy can recover Alice's entire secret key: for Dodis-Ivan, $s = s_1 + s_2$; for BBS, $a = (a/b) \cdot b$. We will mitigate this problem – allowing recovery of a "weak" secret key only. In a bilinear setting, suppose Alice's public key is $\mathsf{e}(g_1, g_1)^a$ and her secret key is $a$; then we might allow

| | Property | DI [92] | $\Sigma_0$ (Section 5.3.2) | BBS [21] | $\Sigma_2$ (Section 5.3.3) |
|---|---|---|---|---|---|
| 1. | Unidirectional | Yes | Yes | **No** | **Yes** |
| 2. | Multi-use | No | No | **Yes** | **No** |
| 3. | Non-interactive | Yes | Yes | **No** | **Yes** |
| 4. | Proxy invisible | No | No | Yes | Yes |
| 5. | Unlinkable | No | No | No | No |
| 6. | Original-access | Yes | Yes | Yes$^\dagger$ | Yes$^\dagger$ |
| 7. | Key optimal | **No** | **Yes** | Yes | Yes |
| 8. | Collusion-"safe" | No | No | **No** | **Yes**$^*$ |
| 9. | Temporary | Yes$^\dagger$ | Yes$^\dagger$ | Yes$^\dagger$ | Yes$^\dagger$ |
| 10. | Non-transitive | Yes | Yes | **No** | **Yes** |
| 11. | Non-transferable | No | No | No | No |

Table 5.1: We compare known proxy encryption (DI, $\Sigma_0$) and re-encryption (BBS, $\Sigma_2$) schemes based on the advantages described above. No schemes achieve properties five and eleven. We refer to the unidirectional schemes of Dodis-Ivan. The symbol $*$ indicates master secret key only. For $\dagger$, the property is possible to achieve by doubling the overhead, as in scheme $\Sigma_{temp}$.

Bob and the proxy to recover the value $g_1^a$, but not $a$ itself. The property of collusion safety is extremely useful in our context since we allow the sender to generate first-level encryptions that cannot be opened with a weak form of the secret key.

Intuitively, collusion safety allows Alice to delegate decryption rights, while keeping signing rights for the same public key. In practice, a user can always use two public keys for encryption and signatures, but it is theoretically interesting that she doesn't *need* to do so. Prior work on *signcryption* explored this area (e.g., [182, 11, 3]). Our scheme could be the basis for the first *sign-re-encryption* scheme (although we will not be formally concerning ourselves with the security of the signatures in this work).

9. *Temporary:* Dodis and Ivan [92] suggested applying generic key-insulation techniques [93, 90, 91] to their constructions to form schemes where Bob is only able to decrypt messages intended for Alice that were authored during some specific time period $i$. Citing space considerations, they did not present any concrete constructions. In Section 5.3.4, we provide a bilinear construction designed specifically for this purpose. In our construction, a trusted server broadcasts a new random number at each time period, which each user can then use to update their delegated secret keys. This is an improvement over using current key-insulated schemes where the trusted server needs to individually interact with each user to help them update their secret keys.

10. *Non-transitive:* The proxy, alone, cannot re-delegate decryption rights. For example, from $rk_{a\to b}$ and $rk_{b\to c}$, he cannot produce $rk_{a\to c}$.

11. *Non-transferable:* The proxy and a set of colluding delegatees cannot re-delegate decryption rights. For example, from $rk_{a\to b}$, $sk_b$, and $sk_c$, they cannot produce $rk_{a\to c}$. We are not aware of any scheme that has this property, and it is a very desirable one. For instance, a

hospital may be held legally responsible for safeguarding the encrypted files of its patients; thus, if it chooses to delegate decryption capabilities to a local pharmacy, it may need some guarantee that this information "goes no further." First, we should ask ourselves: is *transferability* really preventable? The pharmacy can always decrypt and forward the plaintext files to a drug company. However, this approach requires that the pharmacy remain an active, online participant. What we want to prevent is the pharmacy (plus the proxy) providing the drug company with a secret value that it can use offline to decrypt the hospital's ciphertexts. Again, the pharmacy can trivially send its secret key to the drug company. But in doing so, it assumes a security risk that is as potentially injurious to itself as the hospital. Achieving a proxy scheme that is *non-transferable*, in the sense that the only way for Bob to transfer offline decryption capabilities to Carol is to expose his own secret key, seems to be one of the main open problem left for proxy re-encryption.

### 5.3.2 $\Sigma_0$: Paillier-Based Proxy Encryption (Not Re-Encryption)

In the beginning of this chapter and in Table 5.1, we reviewed the BBS proxy *re*-encryption scheme as well as Dodis-Ivan proxy encryption scheme. Before presenting our new proxy *re*-encryption schemes, we offer a new proxy encryption scheme.

As Dodis and Ivan point out [92], one method for delegating decryption rights is to create a cryptosystem that has a two-stage decryption procedure with two different secret keys. In practice, Alice's secret key $s$ is divided into two shares: $s_1$, given to the proxy, and $s_2$, given to Bob. A ciphertext intended for Alice can be partially decrypted by the proxy via $s_1$. Bob can complete the decryption process by using $s_2$ and then recover the message. We already noticed that this "secret sharing" approach does not exactly yield a proxy re-encryption scheme given that Bob must use secrets other than his own to recover the plaintext (i.e., there is no transformation of a ciphertext under Alice's public key into one under Bob's). In particular, previous secret-sharing schemes [92] are not key optimal, proxy invisible, or collusion-"safe."

We improve on this proxy encryption body of work by presenting a scheme, denoted $\Sigma_0$, that is collusion-"safe", although not key optimal or proxy invisible. Our design paradigm, which will be used again shortly, is to start with a cryptosystem with ciphertexts that can be fully decrypted using either of two distinct keys, often called *weak* and *strong*. In particular, we consider a variant of the Paillier cryptosystem [160] with two trapdoors proposed by Cramer and Shoup [85]. For simplicity, we will describe a version that is only semantically secure and we refer to the original work [85] for the full CCA2 secure scheme. This simplified scheme was described by Bresson, Catalano, and Pointcheval [44] where the authors also show a variant of the scheme in [85] that works in the cyclic group of quadratic residues modulo $n^2$.

The public key is $(n, g, h = g^x)$ with $g$ of order $\lambda(n) = 2p'q'$, the *strong* secret key is the factorization of $n = pq$ (where $p = 2p' + 1, q = 2q' + 1$ are safe primes), and the *weak* secret key is $x \in [1, n^2/2]$. (As remarked by Cramer and Shoup [85], such a $g$ can be easily found by selecting a random $a \in \mathbb{Z}^*_{n^2}$ and computing $g = -a^{2n}$.) To encrypt a message $m \in \mathbb{Z}_n$,

select a random $r \in [1, n/4]$ and output the ciphertext $(T_1, T_2)$ as:

$$T_1 = g^r, T_2 = h^r \cdot (1 + m \cdot n) \pmod{n^2}.$$

If $x$ is known, then the message can be recovered as: $m = L(T_2/T_1^x \bmod n^2)$, where $L(u) = \frac{u-1}{n}$, for all $u \in \{u < n^2 \mid u = 1 \bmod n\}$. If $(p, q)$ are known, then $m$ can be recovered from $T_2$ by noticing that $T_2^{\lambda(n)} = g^{\lambda(n)xr}(1 + m\lambda(n)n) = (1 + m\lambda(n)n)$. Thus, given that $\gcd(\lambda(n), n) = 1$, $m$ can be recovered as: $m = L(T_2^{\lambda(n)} \bmod n^2)[\lambda(n)]^{-1} \bmod n$.

Part of the cryptosystem above can be seen as a variation of El Gamal when working modulo a squared composite number. So, similarly to the Dodis-Ivan scheme, we can divide $x$ into two shares $x_1$ and $x_2$, such that $x = x_1 + x_2$. The share $x_1$ is given to the proxy while $x_2$ is stored by the delegatee. The scheme is collusion-safe since only the weak secret $x$ is exposed if the delegatee and the proxy collude, but the factors of $n$, $p$ and $q$, remain secret. Indeed, one could send only the value $T_2$, rather than the ciphertext pair $(T_1, T_2)$, to allow the delegator, and only her, to decrypt the message. (Remember that we are assuming that ciphertexts are generated correctly.) Although collusion-"safe," this scheme is not key optimal or proxy invisible but it remains theoretically interesting because it goes beyond previous work and is not based on bilinear groups. However, it is not a pure proxy re-encryption scheme since there is no transformation of ciphertexts computed under the delegator's key into ones under the delegatee's.

**Fixing Key Optimality for Proxy Encryption?** One way to address this, which also applies to the Dodis-Ivan schemes, is to let the proxy store the delegatee's shares encrypted under his own public key. For instance, in the case where Alice is the delegator, the proxy could store $x_1$ and $x_2$, the latter encrypted under Bob's public key. The encrypted share would then be sent to Bob along with the ciphertext partially decrypted by the proxy. This solution, however, is not satisfactory for our practical applications. It requires more bandwidth; it doubles the cost of decrypting; it forces Bob to perform distinct decryption procedures based on whether he receives ciphertexts intended for him or ciphertexts from the proxy, and it complicates the revocation of decryption rights.

### 5.3.3 $\Sigma_1$ and $\Sigma_2$: New Unidirectional Re-Encryption Schemes

We now present the main results of this chapter.

#### $\Sigma_1$: A First Attempt at a Unidirectional Scheme

We begin with the ideas from the BBS [21], El Gamal-based [97] scheme. Our scheme requires a bilinear map, as discussed in Chapter 2. For simplicity, we will present this scheme for a *double isomorphism* bilinear map $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ (e.g., implemented using a supersingular curve) where we treat $\mathbb{G}_1 \equiv \mathbb{G}_2$ and, as before, $\langle g_1 \rangle = \mathbb{G}_1$ has prime order $q$. We can also use other types of bilinear maps, which we will discuss at the close of this section.

We assume that all ciphertexts are accompanied by a tag that identifies: (1) the relevant public key and (2) the type of decryption algorithm needed.

- **System Parameter Generation** (SysGen): On input the security parameter $1^k$, run $\mathsf{Bilinear_Setup}(1^k) \rightarrow (\mathsf{e}, q, g_1, g_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ to obtain parameters for the bilinear map. Output these as *params*.

- **Key Generation** (KeyGen): On input the parameters *params*, select a random $a \in \mathbb{Z}_q$, and output the key pair $pk = g_1^a$ and $sk = a$. (Assume *params* are included too.)

- **Re-Encryption Key Generation** (ReKey). A user $A$ delegates to $B$ by publishing the re-encryption key $rk_{A \rightarrow B} = g_1^{b/a} \in \mathbb{G}_1$, computed from $B$'s public key.

- **First-Level Encryption** ($\mathsf{Enc}_1$). To encrypt a message $m \in \mathbb{G}_T$ under $pk_A$ in such a way that it can only be decrypted by the holder of $sk_A$, output $c_{A,1} = (\mathsf{e}(g_1, g_1)^{a \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$. Tag ciphertext as a first-level ciphertext for user $A$.

- **Second-Level Encryption** ($\mathsf{Enc}_2$). To encrypt a message $m \in \mathbb{G}_T$ under $pk_A$ in such a way that it can be decrypted by $A$ and her delegatees, output $c_{A,2} = (g_1^{a \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$. Tag ciphertext as a second-level ciphertext for user $A$.

- **Re-Encryption** (ReEnc). Anyone can change a second-level ciphertext for $A$ into a first-level ciphertext for $B$ with $rk_{A \rightarrow B} = g_1^{b/a}$. From $c_{A,2} = (g_1^{a \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$, compute $\mathsf{e}(g_1^{a \cdot k}, g_1^{b/a}) = \mathsf{e}(g_1, g_1)^{b \cdot k}$ and publish $c_{B,1} = (\mathsf{e}(g_1, g_1)^{b \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$. Tag ciphertext as a first-level ciphertext for user $B$.

- **Decryption** ($\mathsf{Dec}_1, \mathsf{Dec}_2$). To decrypt a first-level ciphertext $c_{A,1} = (\alpha, \beta)$ with secret key $sk_A = a$, run $\mathsf{Dec}_1$ to compute $\beta / \alpha^{1/a} = m$. To decrypt a second-level ciphertext $c_{A,2} = (\alpha, \beta)$ with secret key $sk = a$, run $\mathsf{Dec}_2$ to compute $\beta / \mathsf{e}(\alpha, g_1)^{1/a} = m$.

**Discussion of Scheme $\Sigma_1$.** This scheme is very attractive; it is unidirectional, non-interactive, proxy invisible, collusion-safe, key optimal, and non-transitive. In particular, notice that first-level encryptions intended for the delegator are safe even if the delegatee and the proxy collude. Indeed, the weak secret $g_1^{1/a}$ cannot be used to decrypt first-level encryptions (but only second-level ones, which Bob and the proxy can open anyway).

The scheme is also very efficient since both encryption and decryption operations are similar to those of plain El Gamal while the bilinear map computation is only performed by the proxy.

The standard security of this scheme depends upon the Decisional Bilinear Diffie-Hellman Inversion (DBDHI) assumption that the following problem is hard in $(\mathbb{G}_1, \mathbb{G}_T)$:

$$\text{Given } (g_1, g_1^a, g_1^b, Q), \text{ for } g_1 \xleftarrow{R} \mathbb{G}_1,\ a, b \xleftarrow{R} \mathbb{Z}_q^2$$
$$\text{and } Q \in \mathbb{G}_T, \text{ decide if } Q = \mathsf{e}(g_1, g_1)^{a/b}.$$

To see where the above assumption comes into play, think of $g_1^a$ as $g_1^{b \cdot k}$ for some $k \in \mathbb{Z}_q$. Now, consider the second-level ciphertext $c_{A,2} = (g_1^a, m \cdot Q)$ encrypted for public key $g_1^b$ for message $m$. If $Q = \mathsf{e}(g_1, g_1)^{a/b} = \mathsf{e}(g_1, g_1)^{b \cdot k/b} = \mathsf{e}(g_1, g_1)^k$, then $c_{A,2}$ is a proper encryption of

$m$; otherwise, it is an encryption of some other message $m' \neq m$. Thus, an adversary which breaks the above decisional assumption can easily be made into an adversary which breaks the semantic security of this scheme.

Recently, Dodis and Yampolskiy [95] used a stronger version of DBDHI called $y$-Decisional Bilinear Diffie-Hellman Inversion ($q$-DBDHI) where for a random $g_1 \in \mathbb{G}_1$, $x \in \mathbb{Z}_q$, and $Q \in \mathbb{G}_T$, given $(g_1, g_1^x, g_1^{x^2}, \ldots, g_1^{x^y}, Q)$, it is hard to decide if $Q = \mathsf{e}(g_1, g_1)^{1/x}$ or not. They gave evidence for the hardness of this assumption by showing that it holds in the generic group model.

The digital-identity security of the above scheme relies on the 2-DL assumption; that is, given $(g_1, g_1^a, g_1^{a^2})$, compute $a$. This appears necessary to generate the appropriate re-encryption keys. However, 2-DL is implied by 2-DBDHI.

Although the $y$-DBDHI assumption seems plausible, by making a few alterations to this core idea we are able to provide a solution under what appears to be even milder assumptions.

**Using General Bilinear Maps for $\Sigma_1$.** In our presentation of $\Sigma_1$, we assumed a double-isomorphism bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, which, based on current knowledge, would limit us to implementations using supersingular curves. We could allow for more candidate implementations using ordinary curves by assuming a bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with arbitrary isomorphisms; that is, we will not use any of the isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ in our scheme or its proof, but neither will our scheme or proof break if efficient isomorphisms exist. To use a general bilinear map, the following must occur:

1. the public keys and first half of second-level ciphertexts are set in $\mathbb{G}_1$,
2. the re-encryption keys set in $\mathbb{G}_2$,
3. the complexity assumption changes to (at least) a version of 2-DBDHI, for groups where isomorphisms may not exist and thus the input is explicitly provided in both groups: for random $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $a \in \mathbb{Z}_q$, and $Q \in \mathbb{G}_T$, given $(g_1, g_1^a, g_1^{a^2}, g_2, g_2^a, g_2^{a^2})$, it is hard to decide if $Q = \mathsf{e}(g_1, g_2)^{1/a}$. Note that this assumption is no stronger than regular 2-DBDHI set in groups with isomorphisms.

### $\Sigma_2$: a Secure Unidirectional Scheme

We now present our main scheme. The global system parameters *params* remain unchanged.

We assume that all ciphertexts are accompanied by a tag that identifies: (1) the relevant public key and (2) the type of decryption algorithm needed.

- **Key Generation (KeyGen).** A user $A$'s key pair is of the form $pk_A = (\mathsf{e}(g_1, g_1)^{a_1}, g_1^{a_2})$ and $sk_A = (a_1, a_2)$. (A user can encrypt and delegate decryption rights all under $\mathsf{e}(g_1, g_1)^{a_1}$; if the value $g_1^{a_2}$ is present, it signifies that the user is willing to accept delegations.)
- **Re-Encryption Key Generation (ReKey).** A user $A$ delegates to $B$ publishing the re-encryption key $rk_{A \rightarrow B} = g_1^{a_1 \cdot b_2} \in \mathbb{G}_1$, computed from $B$'s public information.

- **First-Level Encryption** ($\mathsf{Enc}_0, \mathsf{Enc}_1$). To encrypt a message $m \in \mathbb{G}_T$ under $pk_A$ in such a way that it can only be decrypted by the holder of $sk_A$, run $\mathsf{Enc}_1$ to output $c_{A,1} = (\mathsf{e}(g_1, g_1)^{a_1 \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$. To achieve proxy invisibility, but necessarily involve the second portion of the recipient's public key, run $\mathsf{Enc}_0$ to output $c_{A,0} = (\mathsf{e}(g_1, g_1)^{a_2 \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$.

- **Second-Level Encryption** ($\mathsf{Enc}_2$). To encrypt a message $m \in \mathbb{G}_T$ under $pk_A$ in such a way that it can be decrypted by $A$ and her delegatees, output $c_{A,2} = (g_1^k, m \cdot \mathsf{e}(g_1, g_1)^{a_1 \cdot k})$.

- **Re-Encryption** ($\mathsf{ReEnc}$). Anyone can change a second-level ciphertext for $A$ into a first-level ciphertext for $B$ with $rk_{A \to B} = g_1^{a_1 \cdot b_2}$. From $c_{A,2} = (g_1^k, m \cdot \mathsf{e}(g_1, g_1)^{a_1 \cdot k})$, compute $\mathsf{e}(g_1^k, g_1^{a_1 \cdot b_2}) = \mathsf{e}(g_1, g_1)^{b_2 \cdot a_1 \cdot k}$ and publish $c_{B,0} = (\mathsf{e}(g_1, g_1)^{b_2 \cdot a_1 \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^{a_1 \cdot k})$ which is equivalent to $(\mathsf{e}(g_1, g_1)^{b_2 \cdot k'}, m \cdot \mathsf{e}(g_1, g_1)^{k'})$ for some $k' \in \mathbb{Z}_q$. Update ciphertext tag appropriately.

- **Decryption** ($\mathsf{Dec}_0, \mathsf{Dec}_1, \mathsf{Dec}_2$). To decrypt a first-level ciphertext $c_{A,0} = (\alpha, \beta)$ with secret key $a_2 \in sk_A$, run $\mathsf{Dec}_0$ to compute $\beta / \alpha^{1/a_2} = m$. To decrypt a first-level ciphertext $c_{A,1} = (\alpha, \beta)$ with secret key $a_1 \in sk_A$, run $\mathsf{Dec}_1$ to compute $\beta / \alpha^{1/a_1} = m$. To decrypt a second-level ciphertext $c_{A,2} = (\alpha, \beta)$ with secret key $a_1 \in sk_A$, run $\mathsf{Dec}_2$ to compute $\beta / \mathsf{e}(\alpha, g_1)^{a_1} = m$.

**Theorem 5.5 (Security of $\Sigma_2$).** *Any adversary that $(1/2+\varepsilon)$-breaks the standard security of the proxy re-encryption scheme $\Sigma_2$ can be used to $(1/2 + \varepsilon/2)$-break the extended Decisional Bilinear Diffie-Hellman (eDBDH) assumption in $(\mathbb{G}_1, \mathbb{G}_T)$; that is, for random $g_1 \in \mathbb{G}_1$, random $a, b, c \in \mathbb{Z}_q^3$, and $Q \in \mathbb{G}_T$, given $(g_1, g_1^a, g_1^b, g_1^c, \mathsf{e}(g_1, g_1)^{b \cdot c^2}, Q)$, it is hard to decide if $Q = \mathsf{e}(g_1, g_1)^{a \cdot b \cdot c}$. Moreover, any adversary that $\varepsilon$-breaks the digital-identity security of $\Sigma_2$ can be used to $\varepsilon$-break the discrete logarithm assumption in $\mathbb{G}_1$.*

*Proof.* Our security definition quantifies over all encryption algorithms $\mathsf{Enc}_i \in \{\mathsf{Enc}\}$; in this case, we have three algorithms $\mathsf{Enc}_0, \mathsf{Enc}_1$, and $\mathsf{Enc}_2$.

First, let's consider an $\mathsf{Enc}_0$ ciphertext of the form $(\mathsf{e}(g_1, g_1)^{a_2 \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$, where $g_1^{a_2}$ is part of the public key. Observe that the re-encryption keys of the form $g_1^{b_1 \cdot a_2}$ reveal no additional information about $a_2$. The security of this cryptosystem follows from the DBDH assumption. Since this is a simple case, let us sketch the solution. On DBDH input $(g_1, g_1^a, g_1^b, g_1^c, Q)$, treat $a$ as $a_2$ and $(b \cdot c)$ as $k$. (All other secret keys may be selected at random.) On the challenge ciphertext, encrypt $m$ as $(Q, m \cdot \mathsf{e}(g_1^b, g_1^c))$. If the adversary correctly distinguishes the ciphertext, then output "$Q = \mathsf{e}(g_1, g_1)^{a \cdot b \cdot c}$"; otherwise output "$Q \neq \mathsf{e}(g_1, g_1)^{a \cdot b \cdot c}$." Now, if $Q = \mathsf{e}(g_1, g_1)^{a \cdot b \cdot c}$, then the challenge ciphertext is well-formed, and DBDH is broken with probability $1/2 + \varepsilon$; otherwise, the message is information-theoretically hidden from the adversary, and thus our DBDH response is correct with probability exactly $1/2$. That covers standard security. It is also easy to see that digital-identity security follows; if the adversary outputs $sk_A = (a_1, a_2)$, then we recover $a_2 = a$ to break the DL assumption in $\mathbb{G}_1$.

Next, let's consider the more complicated cases. Take $\mathsf{Enc}_1$ ciphertext of the form $(\mathsf{e}(g_1, g_1)^{a_1 \cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$. This construction is equivalent to that of the form $(\mathsf{e}(g_1, g_1)^k, m \cdot \mathsf{e}(g_1, g_1)^{a_1 \cdot k})$ [97]. Now, it is clear if the $\mathsf{Enc}_2$ ciphertext of the form $(g^k, m \cdot \mathsf{e}(g_1, g_1)^{a_1 \cdot k})$ is secure, then so are the $\mathsf{Enc}_1$ versions, since $\mathsf{Enc}_2$ ciphertexts reveal strictly more information (i.e., $g_1^k \in \mathbb{G}_1$). Thus, it suffices to argue the security of the $\mathsf{Enc}_2$ ciphertexts only.

**Standard Security.** Suppose $A$ distinguishes encryptions of $\mathsf{Enc}_2$ with probability $\varepsilon$, we simulate an adversary $S$ that decides eDBDH as follows:

1. On eDBDH input $(y, y^a, y^b, y^c, \mathsf{e}(y, y)^{b \cdot c^2}, \mathsf{e}(y, y)^d)$, the simulator sets up a proxy re-encryption world for the adversary $\mathsf{Adv}$ with the goal of using $\mathsf{Adv}$ to decide if $d = a \cdot b \cdot c$ or not. (Note, we slightly change our notation the eDBDH for clarity in this proof; here $Q = \mathsf{e}(y, y)^d$.)
   To begin, the simulator outputs the global parameters for the system $(g, \mathsf{e}(g, g))$. Here, for reasons we will later see, the simulator sets $g = y^c$ and $\mathsf{e}(g, g) = \mathsf{e}(y, y)^{c^2}$. Next, the simulator sends to adversary $\mathsf{Adv}$ the target public key $pk_T = (\mathsf{e}(y, y)^{b \cdot c^2} = \mathsf{e}(g, g)^b, (y^c)^t = g^t)$, where $t$ is randomly selected from $\mathbb{Z}_q$ by the simulator. Thus, we can think of $(b, t)$ as the secret key of the target user.
2. Next, for $i = 1$ up to $\mathrm{poly}(k)$, $\mathsf{Adv}$ can request via oracle $\mathcal{O}_{rekey}$:
   (a) $rk_{\mathsf{Adv},i \to T}$, a delegation to $T$ from a party corrupted by $\mathsf{Adv}$. $\mathsf{Adv}$ can generate these delegations for as many corrupted users as it likes internally by running $(pk_{\mathsf{Adv},i}, sk_{\mathsf{Adv},i}) \leftarrow \mathsf{KeyGen}(1^k)$ and then running $\mathsf{ReKey}(sk_{\mathsf{Adv},i}, pk_T)$ to obtain $rk_{\mathsf{Adv},i \to T} = (g^t)^{sk_{(\mathsf{Adv},i,1)}}$, where $sk_{\mathsf{Adv},i} = (sk_{(\mathsf{Adv},i,1)}, sk_{(\mathsf{Adv},i,2)})$.
   (b) $rk_{T \to h,i}$, a delegation from $T$ to an honest party. The simulator randomly selects two values $r_{(h,i,1)}, r_{(h,i,2)} \in \mathbb{Z}_q^2$, sets $rk_{T \to h,i} = (y^b)^{r_{(h,i,2)}} = g^{b \cdot (r_{(h,i,2)}/c)}$ and $pk_{h,i} = (\mathsf{e}(g, g)^{r_{(h,i,1)}}, y^{r_{(h,i,2)}} = g^{r_{(h,i,2)}/c})$, and sends $(pk_{h,i}, rk_{T \to h,i})$ to $\mathsf{Adv}$. The corresponding secret key is $sk_{h,i} = (r_{(h,i,1)}, (r_{(h,i,2)}/c))$.
   (c) $rk_{h,i \to T}$, a delegation to $T$ from an honest party. The simulator uses either the recorded value $r_{(h,i,1)}$ from the previous step if the honest party already exists, or generates fresh random values for a new party, and computes $rk_{h,i \to T} = (g^t)^{r_{(h,i,1)}}$.
3. Eventually, $\mathsf{Adv}$ must output a challenge $(m_0, m_1, \tau)$, where $m_0 \neq m_1 \in M$ and $\tau$ is its internal state information. The simulator randomly selects $s \in \{0, 1\}$, computes the ciphertext $c_s = (y^a, m_s \cdot \mathsf{e}(y, y)^d) = (g^{a/c}, m_s \cdot \mathsf{e}(g, g)^{d/c^2})$, sends $(c_s, \tau)$ to $\mathsf{Adv}$, and waits for $\mathsf{Adv}$ to output $s' \in \{0, 1\}$.
4. If $s = s'$, then $S$ guesses "$d = a \cdot b \cdot c$"; otherwise $S$ guesses "$d \neq a \cdot b \cdot c$".

First, we observe that if $d = a \cdot b \cdot c$, then the simulation is perfect; that is, the ciphertext output is of the proper form $(g^{a/c}, m_b \cdot \mathsf{e}(g, g)^{(a \cdot b \cdot c)/c^2} = m_b \cdot \mathsf{e}(g, g)^{b \cdot (a/c)})$ for the user with $sk_{(T,1)} = b$. However, if $d \neq a \cdot b \cdot c$, then $m_b$ is information-theoretically hidden from $\mathsf{Adv}$, since $d$ was chosen independently of $a, b, c$. Thus, if $\mathsf{Adv}$ succeeds with probability $1/2 + \varepsilon$, then $S$ succeeds with probability $(1/2 + \varepsilon)$ (when $d = a \cdot b \cdot c$) and probability exactly $1/2$ (when $d \neq a \cdot b \cdot c$), for an overall success probability of $(1/2 + \varepsilon/2)$.

**Digital-Identity Security.** Suppose an adversary $\mathsf{Adv}$ can recover the secret key of a targeted user $T$ (i.e., $sk_T = (sk_{(T,1)}, sk_{(T,2)})$) with probability $\varepsilon$ by interacting with $T$

according to Definition 5.1, then we can build an adversary $S$ that takes discrete logarithms in $\mathbb{G}_1$ with probability $\varepsilon$. Let us focus our attention on recovering only the value $sk_{(T,1)}$ (which is arguably the most valuable of the two). Our simulator $S$ works as follows:

1. On input $(g_1, g_1^a)$ in $\mathbb{G}_1$, output the global parameters $(g_1, \mathsf{e}(g_1, g_1))$ and the target public key $pk_T = (\mathsf{e}(g_1, g_1^a),\ g_1^{sk_{(T,2)}})$, where $sk_{(T,2)}$ is chosen at random from $\mathbb{Z}_q$. We can think of $sk_{(T,1)} = a$.
2. Next, for $i = 1$ up to poly$(k)$, Adv can request via oracle $\mathcal{O}_{rekey}$:
   (a) $rk_{T\to i}$, a delegation from $T$ to a party corrupted by Adv. $S$ randomly selects $r_{(i,1)}, r_{(i,2)} \in \mathbb{Z}_q^2$, sets $rk_{T\to i} \leftarrow g_1^{a\cdot r_{(i,2)}}$, $pk_i = (\mathsf{e}(g_1, g_1)^{r_{(i,1)}},\ g_1^{r_{(i,2)}})$, and $sk_i = (r_{(i,1)}, r_{(i,2)})$, and sends $(pk_i, sk_i, rk_{T\to i})$ to Adv.
   (b) $rk_{i\to T}$, a delegation to $T$ from a party corrupted by Adv. Adv can generate these delegations internally by running $(pk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^k)$ and then running $\mathsf{ReKey}(sk_i, pk_T)$ to compute $rk_{i\to T} = (g_1^{sk_{(T,2)}})^{sk_{(i,1)}}$.
3. Eventually, Adv must output a purported secret key for $T$ of the form $(\alpha, \beta)$. The simulator returns the value $\alpha$.

The simulation is perfect; thus Adv must not be able to recover the full secret key of $T$, despite accepting and providing numerous delegations to $T$, because otherwise, $S$ can efficiently solve the discrete logarithm problem in $\mathbb{G}_1$. □

**Discussion of Scheme $\Sigma_2$.** This scheme is similar to the previous one, except to *accept* delegations, a user must store *one* additional secret key. That is, by storing one additional key, Bob can accept delegations from a polynomial number of friends. If the delegatee and the proxy collude, they will not recover the delegator's strong secret key. Indeed, they can recover only the weak secret $g_1^{a_1}$ that can only be used to decrypt second-level encryptions (which the delegatee and the proxy can already open anyway).

As in our previous scheme, both encryption and decryption operations are similar to those of plain El Gamal, thus very efficient, while the bilinear map computation is performed only by the proxy. We will provide benchmarks of $\Sigma_2$'s performance in Section 5.4.

Its security relies on an extension of the Decisional Bilinear Diffie-Hellman (DBDH) assumption [29, 81]. The proof of Boneh and Franklin [29] that the DBDH problem is hard in generic groups, in the sense of Shoup [173], can be easily extended to this problem, when one recalls that the additional parameter $\mathsf{e}(g_1, g_1)^{b\cdot c^2}$ is represented as a random string in the range of the mapping. When no delegations are made, original first-level ciphertexts of the form $(\mathsf{e}(g_1, g_1)^{a_1\cdot k}, m \cdot \mathsf{e}(g_1, g_1)^k)$ are exactly like El Gamal [97] and thus their external security only depends on DDH in $\mathbb{G}_T$.

**Using General Bilinear Maps for $\Sigma_2$.** In our presentation of $\Sigma_2$, we assumed a double-isomorphism bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, which, based on current knowledge, would limit us to implementations using supersingular curves. Indeed, our implementation in Section 5.4 uses supersingular curves from the MIRACL library. We could allow for more candidate implementations using ordinary curves by assuming a bilinear map of the

form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with arbitrary isomorphisms; that is, we will not use any of the isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ in our scheme or its proof, but neither will our scheme or proof break if efficient isomorphisms exist. To use a general bilinear map, the following must occur:

1. the first part of the second-level ciphertexts are set in $\mathbb{G}_1$,

2. the (second part of the) public keys and the re-encryption keys are set in $\mathbb{G}_2$,

3. the complexity assumption changes a form of eDBDH where the input is explicitly provided in both $(\mathbb{G}_1, \mathbb{G}_2)$, and DL is for $\mathbb{G}_2$, instead of assuming that the eDBDH input is provided in $\mathbb{G}_1$ with an isomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$.

### 5.3.4 $\Sigma_{temp}$: Temporary Unidirectional Proxy Re-Encryption

In this section, we present our temporary unidirectional proxy re-encryption scheme as it appeared in the journal version of this work [9], which improves the conference version [8] by a slight alteration in the first-level encryption which allows us to prove the scheme's security under DBDH instead of the assumption that, for random $g_1 \in \mathbb{G}_1$, random $a, b, c \in \mathbb{Z}_q^3$, and $Q \in \mathbb{G}_T$, given $(g_1, g_1^a, g_1^b, g_1^c, Q)$, it is hard to decide if $Q = \mathsf{e}(g_1, g_1)^{a \cdot b^2/c}$.

In addition to the global parameters *params* used in the previous schemes, suppose there is a trusted server that broadcasts a random value $h_i \in \mathbb{G}_1$ for each time period $i \geq 1$ to all users. The server can publish all $h_i$ values at once as soon as the system parameters are selected. (Recall that our security proofs hold only for the case where users do not pick their own keys.) We enable Alice to delegate to Bob only for time period $i$, say, while she is on vacation, as follows.

We assume that all ciphertexts are accompanied by a tag that identifies: (1) the relevant public key and (2) the type of decryption algorithm needed.

- **Key Generation** (KeyGen). A user $A$'s key pair is of the form $pk_A = (g_1^{a_0}, g_1^{a_p})$, $sk_A = (a_0, a_p)$, (plus a *temporary* secret $a_i$ for time period $i$ which will be generated in ReKey).

- **Re-Encryption Key Generation** (ReKey). A user $A$ *publicly* delegates to $B$ *during time period* $i$ as follows: (1) $B$ chooses and stores a random value $b_i \in \mathbb{Z}_q$, and publishes $h_i^{b_i}$; then, (2) $A$ computes and publishes $rk^i_{A \rightarrow B} = h_i^{a_p \cdot b_i / a_0}$.

- **First-Level Encryption** ($\mathsf{Enc}_1$). To encrypt $m \in \mathbb{G}_T$ under $pk_A$ during time period $i$ in such a way that it can only be decrypted by $A$, compute $\mathsf{e}(g_1, h_i)^{a_p \cdot k} = \mathsf{e}(g_1^{a_p}, h_i)^k$ and output $c_{A,p} = (\mathsf{e}(g_1, h_i)^{a_p \cdot k}, m \cdot \mathsf{e}(g_1, h_i)^k)$.

- **Second-Level Encryption** ($\mathsf{Enc}_2$). To encrypt $m \in \mathbb{G}_T$ under $pk_A$ during time period $i$ in such a way that it can be decrypted by $A$ and her delegatees, compute $\mathsf{e}(g_1, h_i)^{a_p \cdot k} = \mathsf{e}(g_1^{a_p}, h_i)^k$, and output $c_{A,i} = (g_1^{a_0 \cdot k}, m \cdot \mathsf{e}(g_1, h_i)^{a_p \cdot k})$.

- **Re-Encryption** (ReEnc). Anyone can change a second-level ciphertext for $A$ into a first-level ciphertext for $B$ with $rk_{A\to B,i} = h_i^{a_p \cdot b_i/a_0}$. From ciphertext $c_{A,i} = (g_1^{a_0 \cdot k}, m \cdot \mathsf{e}(g_1, h_i)^{a_p \cdot k})$, compute $\mathsf{e}(g_1, h_i)^{b_i \cdot a_p \cdot k} = \mathsf{e}(g_1^{a_0 \cdot k}, rk_{A\to B})$ and publish the ciphertext $c_{B,i} = (\mathsf{e}(g_1, h_i)^{b_i \cdot a_p \cdot k}, m \cdot \mathsf{e}(g_1, h_i)^{a_p \cdot k}) = (\mathsf{e}(g_1, h_i)^{b_i \cdot k'}, m \cdot \mathsf{e}(g_1, h_i)^{k'})$ where $k' = (a_p \cdot k)$. Update ciphertext tag appropriately.

- **Decryption** ($\mathsf{Dec}_1, \mathsf{Dec}_2$). To decrypt a first-level ciphertext $c_{A,j} = (\alpha, \beta)$ with secret key $a_j \in \{a_p, a_1, a_2, \ldots\}$ (corresponding to a re-encryption from the $j$th time period or a first-level original ciphertext with permanent key $a_p$), run $\mathsf{Dec}_1$ to compute $\beta/\alpha^{1/a_j} = m$. To decrypt a second-level ciphertext $c_{A,j} = (\alpha, \beta)$ with secret key $(a_0, a_p)$, run $\mathsf{Dec}_2$ to compute $\beta^{a_0}/\mathsf{e}(\alpha, h_j)^{a_p} = m$.

**Theorem 5.6 (Security of $\Sigma_{temp}$).** *Any adversary that $(1/2+\varepsilon)$-breaks the standard security of the proxy re-encryption scheme $\Sigma_{temp}$ can be used to $(1/2+\varepsilon/2)$-break the Decisional Bilinear Diffie-Hellman (DBDH) assumption in $(\mathbb{G}_1, \mathbb{G}_T)$; that is, for random $g_1 \in \mathbb{G}_1$, random $a, b, c \in \mathbb{Z}_q^3$, and $Q \in \mathbb{G}_T$, given $(g_1, g_1^a, g_1^b, g_1^c, Q)$, it is hard to decide if $Q = \mathsf{e}(g_1, g_1)^{a \cdot b \cdot c}$. Moreover, any adversary that $\varepsilon$-breaks the digital-identity security of $\Sigma_2$ can be used to $\varepsilon$-break the discrete logarithm assumption in $\mathbb{G}_1$.*

*Proof.* Our security definition quantifies over all encryption algorithms $\mathsf{Enc}_i \in \{\mathsf{Enc}\}$; in this case, we have two algorithms $\mathsf{Enc}_1, \mathsf{Enc}_2$ which produce two different types of ciphertexts. Our security proof will address both styles of ciphertexts.

**Standard Security.** Let $Z$ be the maximum number of time periods. Suppose Adv distinguishes $\mathsf{Enc}_1$ ciphertexts with probability $\varepsilon$ (we will address $\mathsf{Enc}_2$ shortly), we simulate an adversary $S$ that decides DBDH as follows:

1. On input $(g_1, g_1^a, g_1^b, g_1^c, \mathsf{e}(g_1, g_1)^d)$, the simulator sends Adv the global parameters $(g_1, \mathsf{e}(g_1, g_1))$ and the target public key $pk_T = (g_1^t, g_1^a)$, where $t$ is randomly selected from $\mathbb{Z}_q$, and the corresponding secret key is $sk_T = (t, a)$. The simulator also honestly generates the public keys of all other parties.
2. For $j = 1$ up to $Z$ time periods, the simulator selects the public delegation parameter for that time period $h_j = g_1^{x_j}$, where $x_j$ is randomly selected from $\mathbb{Z}_q$. The simulator also generates the delegation acceptance value $D_{(U,j)} = h_j^{z_{(U,j)}}$ for all users $U$, including $T$, where $z_{(U,j)}$ is randomly selected from $\mathbb{Z}_q$.
   (a) Next, for $i = 1$ up to poly($k$), Adv can request via oracle $\mathcal{O}_{rekey}$:
      i. $rk_{\mathsf{Adv},i\to T}$, a delegation to $T$ from a party corrupted by Adv. Adv can generate these delegations internally by running $(pk_{\mathsf{Adv},i}, sk_{\mathsf{Adv},i}) \leftarrow \mathsf{KeyGen}(1^k)$, where $sk_{\mathsf{Adv},i} = (sk_{(\mathsf{Adv},i,0)}, sk_{(\mathsf{Adv},i,p)})$, and running $\mathsf{ReKey}(sk_{\mathsf{Adv},i}, pk_T)$ to compute $rk_{\mathsf{Adv},i\to T} = D_{(T,j)}^{sk_{(\mathsf{Adv},i,p)}/sk_{(\mathsf{Adv},i,0)}}$.
      ii. $rk_{T\to h,i}$, a delegation from $T$ to an honest party with delegation acceptance value $D_{(h,i),j}$ for time period $j$. $S$ computes and sends $rk_{T\to h,i} \leftarrow (g_1^a)^{x_j \cdot z_{(h,i,j)}/t} = D_{(h,i),j}^{a/t}$ to Adv.

iii. $rk_{h,i\to T}$, a delegation to $T$ from an honest party with key $sk_{h,i} = (sk_{(h,i,0)}, sk_{(h,i,p)})$. $S$ trivially computes $rk_{h,i\to T} = D_{(T,j)}^{sk_{(h,i,p)}/sk_{(h,i,0)}}$.

3. Eventually, during the last time period, Adv must output a challenge $(m_0, m_1, \tau)$, where $m_0 \neq m_1 \in M$ and $\tau$ is its internal state information. The simulator randomly selects $s \in \{0,1\}$, computes the ciphertext $c_s = (\mathsf{e}(g_1, g_1)^d, \mathsf{e}(g_1^b, g_1^c) \cdot m_s)$, sends $(c_s, \tau)$ to Adv, and waits for Adv to output $s' \in \{0,1\}$.
4. If $s = s'$, then $S$ guesses "$d = a \cdot b \cdot c$"; otherwise $S$ guesses "$d \neq a \cdot b \cdot c$".

First, we observe that if $d = a \cdot b \cdot c$, then the simulation is perfect; that is, the ciphertext output is of the proper form $(\mathsf{e}(g_1, g_1)^{a \cdot b \cdot c}, \mathsf{e}(g_1, g_1)^{b \cdot c} \cdot m_s)$ for the user with $sk_{(T,p)} = a$. However, if $d \neq a \cdot b \cdot c$, then $m_s$ is information-theoretically hidden from Adv. Thus, if Adv succeeds with probability $1/2 + \varepsilon$ at distinguishing $\mathsf{Enc}_1$ ciphertexts, then $S$ succeeds with probability $(1/2 + \varepsilon)$ (when $d = a \cdot b \cdot c$) and probability exactly $1/2$ (when $d \neq a \cdot b \cdot c$), for an overall probability of $(1/2 + 2/\varepsilon)$. This contradicts the DBDH assumption when $\varepsilon$ is non-negligible.

Now, suppose that Adv distinguishes $\mathsf{Enc}_2$ ciphertexts with $\varepsilon$ probability, we simulate a different adversary $S$ that decides DBDH as follows:

1. On input $(g_1, g_1^a, g_1^b, g_1^c, \mathsf{e}(g_1, g_1)^d)$, the simulator sends Adv the global parameters $(y = g_1^c, \mathsf{e}(y, y) = \mathsf{e}(g, g)^{c^2})$ and the target public key $pk_T = (y^{1/c} = g_1, y^{a/c} = g_1^a)$ and the corresponding secret key is $sk_T = (1/c, a/c)$. The simulator also honestly generates the public keys of all other parties.
2. For $j = 1$ up to $Z$ time periods, the simulator selects the public delegation parameter for that time period $h_j = y^{x_j} = (g_1^c)^{x_j}$, where $x_j$ is randomly selected from $\mathbb{Z}_q$. The simulator also generates the delegation acceptance value $D_{(U,j)} = h_j^{z_{(U,j)}/c} = g_1^{x_j \cdot z_{(U,j)}}$ for all users $U$, including $T$, where $z_{(U,j)}$ is randomly selected from $\mathbb{Z}_q$. The temporary secret for each honest party is logically set to $(z_{(U,j)}/c)$ by the simulator. These acceptance values are generated by the simulator without the $1/c$ term for all corrupted users.
   (a) Next, for $i = 1$ up to poly$(k)$, Adv can request via oracle $\mathcal{O}_{rekey}$:
      i. $rk_{\mathsf{Adv},i\to T}$, a delegation to $T$ from a party corrupted by Adv. Adv can generate these delegations internally by running $(pk_{\mathsf{Adv},i}, sk_{\mathsf{Adv},i}) \leftarrow \mathsf{KeyGen}(1^k)$, where $sk_{\mathsf{Adv},i} = (sk_{(\mathsf{Adv},i,0)}, sk_{(\mathsf{Adv},i,p)})$, and then running $\mathsf{ReKey}(sk_{\mathsf{Adv},i}, pk_T)$ to compute $rk_{\mathsf{Adv},i\to T} = D_{(T,j)}^{sk_{(\mathsf{Adv},i,p)}/sk_{(\mathsf{Adv},i,0)}}$.
      ii. $rk_{T\to h,i}$, a delegation from $T$ to an honest party with delegation acceptance value $D_{(h,i),j}$ for time period $j$. $S$ computes and sends $rk_{T\to h,i} = D_{(h,i),j}^{sk_{(T,p)}/sk_{(T,0)}} = y^{x_j \cdot z_{(h,i,j)} \cdot (a/c)/(1/c)} = (g_1^a)^{x_j \cdot z_{(h,i,j)}}$.
      iii. $rk_{h,i\to T}$, a delegation to $T$ from an honest party. $S$ computes and sends $rk_{h,i\to T} = D_{(T,j)}^{sk_{(h,i,p)}/sk_{(h,i,0)}}$ to Adv.
3. Eventually, during the last time period, Adv must output a challenge $(m_0, m_1, \tau)$, where $m_0 \neq m_1 \in M$ and $\tau$ is its internal state information. The simulator randomly selects

$s \in \{0,1\}$, computes the ciphertext $c_s = (g_1^b, m_s \cdot (\mathsf{e}(g_1, g_1)^d)^{x_j}) = (y^{b/c}, m_s \cdot \mathsf{e}(y, h_j)^{d/c^2})$, sends $(c_s, \tau)$ to Adv, and waits for Adv to output $s' \in \{0,1\}$.

4. If $s = s'$, then $S$ guesses "$d = a \cdot b \cdot c$"; otherwise $S$ guesses "$d \neq a \cdot b \cdot c$".

Now, we observe that if $d = a \cdot b \cdot c$, then the simulation is perfect; that is, the challenge ciphertext is of the proper form $(y^{sk_{(T,0)} \cdot b}, m_s \cdot \mathsf{e}(y, h_j)^{sk_{(T,p)} \cdot b})$. However, if $d \neq a \cdot b \cdot c$, then $m_s$ is information-theoretically hidden from Adv. Thus, if Adv succeeds with probability $1/2 + \varepsilon$ at distinguishing $\mathsf{Enc}_2$ ciphertexts, then $S$ succeeds with probability $(1/2 + \varepsilon)$ (when $d = a \cdot b \cdot c$) and probability exactly $1/2$ (when $d \neq a \cdot b \cdot c$), for an overall probability of $(1/2 + 2/\varepsilon)$.

**Digital-Identity Security.** Let $Z$ be the maximum number of time periods. Suppose an adversary Adv can recover the secret key of a targeted user $T$ (i.e., $sk_T = (sk_{(T,0)}, sk_{(T,p)})$) with probability $\varepsilon$ by interacting with $T$ according to Definition 5.1, then we can build an adversary $S$ that takes discrete logarithms in $\mathbb{G}_1$. Let us focus our attention on recovering only the value $sk_{(T,p)}$. Our simulator $S$ works as follows:

1. On input $(g_1, g_1^a)$ in $\mathbb{G}_1$, output the global parameters $(g_1, \mathsf{e}(g_1, g_1))$ and the target public key $pk_T = (g_1^{sk_{(T,0)}}, g_1^a)$, where $sk_{(T,0)}$ is chosen randomly from $\mathbb{Z}_q$. We can think of $sk_{(T,p)} = a$.
2. For $j = 1$ up to $Z$ time periods, the simulator publishes the public delegation parameter for that time period $h_j = g_1^{x_j}$, where $x_j$ is randomly selected from $\mathbb{Z}_q$. The simulator also publishes the delegation acceptance value $h_j^{z_{(U,j)}}$ for all users $U$, including $T$, where $z_{(U,j)}$ is randomly selected from $\mathbb{Z}_q$.
3. Next, for $i = 1$ up to poly$(k)$, Adv can request via oracle $\mathcal{O}_{rekey}$:
   (a) $rk_{T \to U}$, a delegation from $T$ to any party. Let $j$ be the current time period. $S$ sets $rk_{T \to U,i} = (g_1^a)^{x_j \cdot z_{(U,j)}/sk_{(T,0)}} = h_j^{a \cdot z_{(U,j)}/sk_{(T,0)}} = h_j^{sk_{(T,p)} \cdot z_{(U,j)}/sk_{(T,0)}}$.
   (b) $rk_{\mathsf{Adv},i \to T}$, a delegation to $T$ from a corrupt party. Adv can generate these delegations internally using the public information of $T$.
4. Eventually, Adv must output a purported secret key for $T$ of the form $(\alpha, \beta)$. The simulator returns the value $\beta$.

The simulation is perfect; thus Adv must not be able to recover the full secret key of $T$, despite accepting and providing numerous delegations to $T$, because otherwise, $S$ can efficiently solve the discrete logarithm problem in $\mathbb{G}_1$. □

**Discussion of Scheme $\Sigma_{temp}$.** A single *global* change can invalidate all previous delegations without *any* user needing to change their public key. The drawback of this scheme when compared to $\Sigma_2$ is that delegatees must store one additional secret for each time period that they accept a delegation. However, somewhat surprisingly, the security of $\Sigma_{temp}$ rests only on the very standard DBDH and DL assumptions.

**Using General Bilinear Maps for $\Sigma_{temp}$.** In our presentation of $\Sigma_{temp}$, we assumed a double-isomorphism bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$, which, based on current

knowledge, would limit us to implementations using supersingular curves. We could allow for more candidate implementations using ordinary curves by assuming a bilinear map of the form $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with arbitrary isomorphisms; that is, we will not use any of the isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ in our scheme or its proof, but neither will our scheme or proof break if efficient isomorphisms exist. To use a general bilinear map, the following must occur:

1. both parts of the public key and the first part of second-level ciphertexts are set in $\mathbb{G}_1$,
2. the re-encryption keys and public time period broadcasts $h_i$ are set in $\mathbb{G}_2$,
3. the complexity assumption changes from DBDH to Co-DBDH, where the input is provided in both $(\mathbb{G}_1, \mathbb{G}_2)$, and DL is for $\mathbb{G}_1$.

## 5.4 Applications

In the beginning of this chapter, we discussed some possible applications of proxy re-encryption, such as forwarding encrypted email and DRM for Apple's iTunes. Previous works [21, 92, 123, 183] have also suggested email forwarding, law enforcement filtering, and performing cryptographic operations on storage-limited devices as some potential applications.

In this section, we explore the benefits that proxy re-encryption can bring to the actively researched, and yet unsolved, area of secure distributed storage. To our knowledge, our implementation represents the first experimental evaluation of a system using proxy re-encryption. Since the focus of this thesis is on the algorithms, this section will only overview of our implementation results. The full system details can be found in the journal version of the work described in this chapter [9].

### 5.4.1 Secure Distributed Storage

**Overview.** In Figure 5-2, we illustrate our file system which uses an untrusted *access control server* to manage access to encrypted files stored on distributed, untrusted devices, called *block stores*. End users on client machines wish to obtain access to integrity-protected, confidential content, such as a DVD. A content owner publishes encrypted content in the form of a many-reader, single-writer file system. The owner encrypts the content (e.g., the DVD) with a unique, symmetric *content key*. This content key is then encrypted with an asymmetric master key to form a *lockbox*. The lockbox resides with the encrypted content it protects.

Untrusted block stores make the encrypted content available to everyone. Users download the encrypted content from a block store, then communicate with an access control server to decrypt the lockboxes protecting the content. The content owner selects which users should have access to the content and gives the appropriate delegation rights to the access control server.

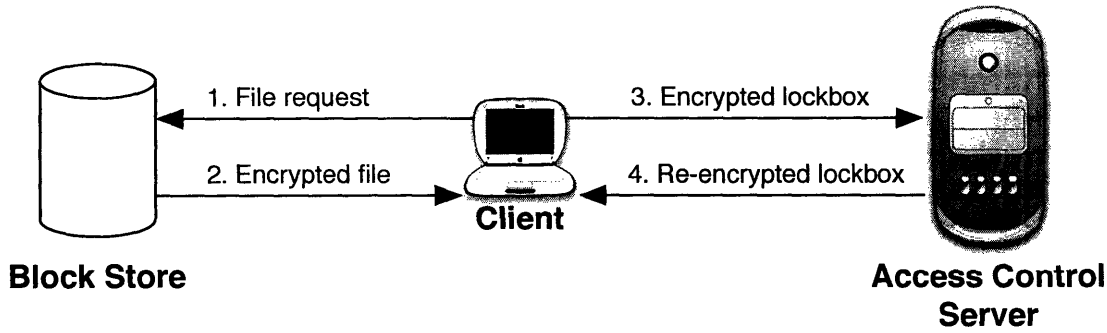


Figure 5-2: Operation of the proxy re-encryption file system. The user's client machine fetches encrypted files from the block store. Each encrypted file has two components: (1) the file content encrypted under a symmetric key, and (2) the symmetric key encrypted under a master public key, called the *lockbox*. The client then transmits the lockbox to the access control server for re-encryption under the user's public key. If the access control server possesses the necessary re-encryption key, it re-encrypts the lockbox and returns the new ciphertext. The client can decrypt the lockbox using the user's secret key, recover the symmetric key, and then decrypt the file content.

**Access Control Using Proxy Re-Encryption.** We propose an improvement on the access control server model that reduces the server's trust requirements by using proxy re-cryptography. In our approach, the content keys used to encrypt files are themselves securely encrypted under a master public key, using a unidirectional proxy re-encryption scheme of the form described in this work. Because the access control server does not possess the corresponding secret key, it cannot be corrupted so as to gain access to the content keys necessary to access encrypted files. The master secret key remains offline, in the care of a content owner who uses it only to generate the re-encryption keys used by the access control server. When an authorized user requests access to a file, the access control server uses proxy re-encryption to directly re-encrypt the appropriate content key(s) from the master public key to the user's public key. (Interestingly, the mis-management of this transfer step caused the security problem [174] in Apple's iTunes DRM mentioned earlier.)

This architecture has significant advantages over systems with trusted access control servers. The key material stored on the access control server cannot be used to access stored files, which reduces the need to absolutely trust the server operator, and diminishes the server's value to attackers. The master secret key itself is only required by a content owner when new users are added to the system, and can therefore be stored safely offline where it is less vulnerable to compromise. Finally, the schemes in Section 5.3.3 are *unidirectional* and *non-interactive*, meaning that users do not need to communicate or reveal their secret keys in order to join the system. This allows content owners to add users to the system without interaction, simply by obtaining their public key. Because this system works with users' long-term keys (rather than generating ephemeral keys for the user), there is an additional

incentive for users not to reveal their secret keys.

The proposed design fundamentally changes the security of an access control server storage system. In this new model, much of the security relies on the strength of a provably-secure cryptosystem, rather than on the trust of a server operator for mediating access control. Because the access control server cannot successfully re-encrypt a file key to a user without possessing a valid re-encryption key, the access control server cannot be made to divulge file keys to a user who has not been specifically authorized by the content owner, unless this attacker has previously stolen a legitimate user's secret key.

**Chefs.** Our file system was implemented on top of Chefs [103], a single-writer, many-reader file system designed by Kevin Fu, and itself built on top of the SFS read-only file system [104]. Chefs tags each encrypted file with a lockbox. In the original Chefs design, the lockbox contains a 128-bit AES key, itself encrypted with a shared group AES key. Chefs assumes an out-of-band mechanism for content owners to distribute group keys to users; this step we now solve with proxy re-encryption.

**Experimental Setup.** In implementing a proxy re-encryption file system, we had two goals in mind. First, we wished to show that proxy re-encryption could be successfully incorporated into a basic cryptographic file system. Second, we sought to prove that the additional security semantics provided by a proxy re-encrypting access control server came at an acceptable cost to end-to-end performance.

For the purposes of our testing, we used two machines to benchmark the proxy-enabled Chefs file system. The client machine consisted of an AMD Athlon 2100+ 1.8 GHz with 1 Gbyte RAM and an IBM 7200 RPM, 40 Gbyte, Ultra ATA/100 hard drive. The server machine was an Intel Pentium 4 2.8 GHz with 1 Gbyte RAM and a Seagate Barracuda 7200 RPM, 160 Gbyte, Ultra ATA/100 hard drive. Both systems were running Debian testing/unstable with the Linux 2.6.8 kernel. The client and the server were situated in different cities, representing a distributed file system scenario. We measured the round-trip latency between the two machines at 13 msec, and the maximum sustained throughput of the network link at 7 Mbit/sec. We implemented the cryptographic primitives for scheme $\Sigma_2$ (Section 5.3.3) using version 4.83 of the MIRACL cryptographic library [170], which contains efficient implementations of the Tate pairing over supersingular curves as well as fast modular exponentiation and point multiplication on elliptic curves.

**Cryptographic Benchmark.** Table 5.2 presents average times over 100 runs of the cryptographic operations in the bilinear proxy re-encryption scheme $\Sigma_2$ (from Section 5.3.3). Our key sizes of 256 and 512 bits, according to the NIST estimates in Chapter 2 (see Table 2.1), offer the equivalent of 3072 and 15360-bit RSA security. Since this level of security is not likely to be necessary until at least 2030, our timing measurements can be considered as very conservative.

These measurements provide a basis for understanding the impact of proxy re-encryption on overall file system performance. These results indicate that re-encryption is the one of the most time consuming operations in our file system, although its overhead is manageable.

We were surprised that our 1.8 GHz AMD Athlon 2100 performed better than our 2.8 GHz Intel Pentium 4 server in the microbenchmarks. We attribute this advantage to modular

| Parameter size | Machine | Encryption | Decryption (by original recipient) | Re-encryption | Decryption (by delegatee) |
|---|---|---|---|---|---|
| 256-bit | client | 3.1 msec | 8.7 msec | 8.6 msec | 1.5 msec |
| | server | 3.3 msec | 8.8 msec | 8.7 msec | 1.5 msec |
| 512-bit | client | 7.8 msec | 22.5 msec | 22.0 msec | 3.4 msec |
| | server | 9.3 msec | 26.5 msec | 26.7 msec | 4.1 msec |

Table 5.2: Average operation times for 100 runs of the $\Sigma_2$ proxy re-encryption scheme on our client and server. All operations refer to re-encryptable second-level ($\mathsf{Enc}_2$) ciphertexts.

arithmetic routines in MIRACL that perform faster on the Athlon. The MIRACL library provides many hints for selecting assembly code optimizations. Because other benchmarks such as the OpenSSL RSA "speed" test run faster on our server, we suspect that the Intel server would perform better with proper selection of optimizations in MIRACL.

**Discussion.** We believe that our experimental results demonstrate the practicality of proxy re-encryption in protecting stored content. Though proxy re-encryption adds a level of overhead to file system, this overhead is not extreme, and can be worth the additional security that comes from using a centralized, semi-trusted access control server.

## 5.5 Contributions

In this chapter, we explored proxy re-encryption from both a theoretical and practical perspective. We formalized the proxy re-encryption primitive of Blaze, Bleumer, and Strauss [21]. We outlined the characteristics and security guarantees of previously known schemes, and compared them to a suite of improved re-encryption schemes we present over bilinear maps. These bilinear-based schemes realize important new features, such as safeguarding the digital identity of the delegator from a colluding proxy and delegatee. Indeed, the realization of unidirectional scheme $\Sigma_2$ answers the open problem proposed by BBS in 1998.

One of the most promising applications for proxy re-encryption is giving proxy capabilities to the key server of a confidential distributed file system; this way the key server need not be fully trusted with all the keys of the system and the secret storage for each user can also be reduced. We implemented this idea in the context of the Chefs file system, and showed experimentally that the additional security benefits of proxy re-encryption can be purchased for a manageable amount of run-time overhead.

## 5.6 Open Problems

There are many open problems in this area. The holy grail for proxy re-encryption schemes– a unidirectional, key optimal, and CCA2 secure scheme –is not yet realized. Another open

problem is to find schemes that are simultaneously unidirectional and multi-use. It is also not known how to prevent the transfer of delegation rights, as discussed in Remark 5.2.

The only unidirectional and key optimal solutions known are those presented in this chapter based on bilinear groups. It would be very interesting to know if this functionality can be efficiently realized without using bilinear maps.

It would also be very interesting if re-encryption schemes existed where the proxy could only translate encryptions of certain messages or encryption containing certain public tags. One promising direction is to extend the temporary scheme $\Sigma_{temp}$ in Section 5.3.4.

We also just brushed the surface of the interesting relationship between proxy re-encryption and program obfuscation. It would be interesting to know if proxy re-encryption could be properly formalized and realized as an obfuscated program.

Finally, we are confident that proxy re-encryption has many important applications beyond secure distributed storage, and we look forward to seeing additional schemes realized under other complexity assumptions and other deployments of the primitive.

# Chapter 6

# Compact E-Cash

This chapter is an extended version of joint work with Jan Camenisch (IBM Zurich Research) and Anna Lysyanskaya (Brown University) [49]. In particular, we acknowledge Anna Lysyanskaya for the formal definitions in Section 6.2.1 (we slightly alter exculpability from [49]) and Jan Camenisch for the optimized protocols in Appendix A.

## 6.1 Introduction

In this chapter and the next one, we turn our attention to electronic cash systems. This is a distinct concept from the Re-Signature (Chapter 4) and Re-Encryption (Chapter 5) schemes discussed earlier. However, we will again use the unique properties of bilinear groups in some parts of our constructions.

Electronic cash was invented by Chaum [73, 74], and has been extensively studied [77, 100, 79, 40, 57, 41, 175, 99, 178, 19]. The main idea is that, even though the same party (a bank $\mathcal{B}$) is responsible for giving out electronic coins, and for later accepting them for deposit, the withdrawal and the spending protocols are designed in such a way that it is impossible for the bank to identify when a particular coin was spent. I.e., the withdrawal protocol does not reveal any information to the bank that would later enable it to trace how a withdrawn coin was spent.

As a coin is represented by data, and it is easy to duplicate data, an electronic cash scheme requires a mechanism that prevents a user from spending the same coin twice (double-spending). There are two scenarios. In the *on-line* scenario [74, 75, 76], the bank is on-line in each transaction to ensure that no coin is spent twice, and each merchant must consult the bank before accepting a payment. In the *off-line* [77] scenario, the merchant accepts a payment autonomously, and later submits the payment to the bank; the merchant is guaranteed that such a payment will be either honored by the bank, or will lead to the identification (and therefore punishment) of the double-spender.

In this chapter, we give an off-line $2^\ell$-spendable unlinkable electronic cash scheme. Namely, our scheme allows a user to withdraw a wallet with $2^\ell$ coins, such that the space required to store these coins, and the complexity of the withdrawal protocol, are proportional to $\ell$.

rather than to $2^\ell$. We achieve this without compromising the anonymity and unlinkability properties usually required of electronic cash schemes. There is a large body of work on divisible or amortized e-cash [159, 156, 136, 137], where each division of a particular coin (e.g., each penny in a dollar) can be linked to the other parts of the same particular coin, but these pieces cannot be linked to the identity of the spender. In this chapter, we obtain full anonymity and unlinkability, while efficiently withdrawing and storing many coins at once.

This problem is well-motivated: (1) communication with the bank is a bottleneck in most electronic cash schemes and needs to be minimized; (2) it is desirable to store many electronic coins compactly, as one can imagine that they may be stored on a dedicated device such as a smartcard that cannot store too much data. This problem has also proved quite elusive: no one has offered a compact e-cash solution (even for a weaker security model) since the introduction of electronic cash in the 1980s.

In addition, a good e-cash scheme should allow one to expose double-spenders to outside third parties in an undeniable fashion. I.e., assuming a PKI, if a user $\mathcal{U}$ with public key $pk_{\mathcal{U}}$ spent a coin more times than he is allowed (in our case, spent $2^\ell + 1$ coins from a wallet containing $2^\ell$ coins), then this fact can be proven to anyone in a sound fashion. This property of an e-cash scheme is satisfied by numerous schemes in the literature [7]. Our solution has this property as well.

Finally, it may often be desirable that an e-cash scheme should allow one to trace *all* coins of a cheating user. It was known that this property can be implemented using a trusted third party (TTP) [175, 46], by requiring that: (1) in each withdrawal protocol a user gives to the bank an encryption under the TTP's public key of a serial number $S$ which will be revealed during the spending protocol; and (2) in each spending protocol, the user submits to the merchant an encryption of the user's public key under the TTP's public key. Then, should a coin with serial number $S$ ever be double-spent, the TTP can get involved and decrypt the serial number of all of this user's coins. But the existence of such a TTP contradicts the very definition of electronic cash: to the TTP, the user is not anonymous! Therefore, another desirable and elusive property of an electronic cash scheme was traceability *without* a TTP. Our scheme achieves this property as well.

Recently, Jarecki and Shmatikov [124] also made a step in this direction. Although their work is not explicitly about electronic cash, it can be thought of in this way. In their scheme, coins are anonymous, but *linkable* (linkability is actually a feature for them). Their scheme allows to withdraw and linkably but anonymously spend a coin $K$ times; but should a user wish to spend the coin $K + 1$ times, his identity gets revealed. As far as electronic cash is concerned, our solution is better for two reasons: (1) their scheme does not achieve unlinkability; and (2) in their protocol, each time a user spends a coin he has to run a protocol whose communication complexity is proportional to $K$, rather than $\log K$, as we achieve. In 1989, Okamoto and Ohta [158] proposed an e-cash scheme with similar functionality, without achieving unlinkability or compact wallets.

Our work can also be viewed as improving on the recent traceable group signatures by Kiayias, Tsiounis, and Yung [133]. In their scheme, once a special piece of tracing information is released, it is possible to trace all group signatures issued by a particular group member;

otherwise this member's signatures are guaranteed to remain anonymous. Normally, in a group signature setting, this piece of information *must* be released by a TTP, as there is no equivalent of a *double-spender* whose misbehavior may automatically lead to the release of the tracing information; however, if a limit is placed on how many signatures a group member may issue, then our e-cash scheme can be viewed as a *bounded* group signature scheme, where a group member can sign a message by incorporating it into the signature proof of a coin's validity. A group manager may allocate signing rights by acting as a bank allocating coins; and if any member exceeds their allocation, the special tracing information is revealed automatically, and all signatures produced by that group member may be traced. Our tracing algorithm is more efficient than that of Kiayias et al. [133]; in our scheme, signatures can be tracked by a serial number (that appears to be random until the user double-spends), while in theirs, *all* existing signatures must be tested, one-by-one, using the special tracing information provided by the TTP, to determine if a certain signer created it or not.

**Our Results.** Let us summarize our results. We give a compact e-cash scheme with all the features described above in the random-oracle model, under the Strong RSA and Decisional Diffie-Hellman Inversion ($y$-DDHI) [25, 95] assumptions in combination with either the External Diffie-Hellman (XDH) [107, 171, 143, 27, 12] or the Sum-Free DDH [89] assumption for groups with bilinear maps. The trade-off between these latter two assumptions is that we achieve a more efficient construction using the XDH assumption, but the Sum-Free DDH assumption is conjectured to hold in a wider class of bilinear groups. (Recall from Chapter 2 that the XDH assumption is known to be false for supersingular curves, however it is believed to hold for ordinary curves. The Sum-Free DDH assumption is believed to hold for either supersingular or ordinary curves.)

Using the Sum-Free DDH assumption, the communication complexity of the spending and of the withdrawal protocol is $O(\ell \cdot k)$ and $O(\ell \cdot k + k^2)$ bits, respectively; it takes $O(\ell \cdot k)$ bits to store all the coins. Alternatively, using the XDH assumption, we give a scheme where the withdrawal and the spending protocols have complexity $O(\ell + k)$, and it takes $O(\ell + k)$ bits to store all the coins. These schemes are presented in Section 6.5.

We also give a scheme where the withdrawal and the spending protocols have complexity only $O(\ell + k)$, and it also takes only $O(\ell + k)$ bits to store all the coins, based on *only* the Strong RSA [105, 14] and the $y$-DDHI [95] assumptions in the random-oracle model. That is, we will *not* require bilinear groups for this scheme, and the cost is that it will not support traceability. If a user over-spends his wallet, only his public key is recovered. This scheme is presented in Section 6.4.

Furthermore, in the model where the bank completely trusts the merchant (this applies to, for example, a subscription service where the entity creating and verifying the coins is one and the same), we have solutions based on the same set of assumptions but *in the standard model*, e.g., no random oracles. In Section 6.6, we discuss the random oracle model in more detail and explain options for removing random oracles from our constructions.

**Using E-Cash as Anonymous Credentials.** Although the results of this chapter, and

Chapter 7, are presented and discussed in terms of electronic cash, they actually apply to the more general notion of anonymous credentials [75, 52, 139, 54]. For example, our e-cash system could be applied directly to implement a system where a user could purchase the ability to later anonymously and unlinkably login to a website at most $N$ times.

**Overview of Our Construction.** Our schemes are based on the signature schemes with protocols due to Camenisch and Lysyanskaya [53, 54], which we will call "CL signatures." These schemes allow a user to efficiently obtain a signature on committed messages from the signer. They further allow the user to convince a verifier that she possesses a signature by the signer on a committed message. Both of these protocols rely on the Pedersen commitment scheme.

To explain our result, let us describe how single-use electronic cash can be obtained with CL signatures, drawing on a variety of previously known techniques [43, 53].

Let $G = \langle g \rangle$ be a group of prime order $q$ where the discrete logarithm problem is hard. Suppose that a user $\mathcal{U}$ has a secret key $sk_{\mathcal{U}} \in \mathbb{Z}_q$ and a public key $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}}$. An electronic coin is a signature under the bank $\mathcal{B}$'s public key $pk_{\mathcal{B}}$ on the set of values $(sk_{\mathcal{U}}, s, t)$, where $s, t \in \mathbb{Z}_q$ are random values. The value $s$ is the *serial number* of the coin, while $t$ is the *value blinding* of this coin. A protocol whereby a user obtains such a signature is called the *withdrawal protocol.*

In the *spending protocol*, the user sends the merchant a Pedersen commitment $C$ to the values $(sk_{\mathcal{U}}, s, t)$, and computes a non-interactive proof $\pi_1$ that they have been signed by the bank. The merchant verifies $\pi_1$ and then picks a random value $R \in \mathbb{Z}_q$. Finally, the user reveals the serial number $s$, and the value $T = sk_{\mathcal{U}} + R \cdot t \bmod q$. Let us refer to $T$ as a *double-spending equation* for the coin. The user must also compute a proof $\pi_2$ that the values $s$ and $T$ correspond to commitment $C$. Finally, the merchant submits $(s, R, T, \pi_1, \pi_2)$ for payment.

Note that one double-spending equation reveals nothing about $sk_{\mathcal{U}}$ because $t$ is random, but using two double-spending equations, we can solve for $sk_{\mathcal{U}}$. So if the same serial number $s$ is submitted for payment twice, the secret key $sk_{\mathcal{U}}$ and therefore the identity of the double-spender $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}}$ can be discovered.

Now, our goal is to adapt single-use electronic cash schemes so that a coin can be used at most $2^\ell$ times. The trivial solution would be to obtain $2^\ell$ coins. For our purposes, however, it is unacceptable, as $2^\ell$ may be quite large (e.g., 1000) and we want each protocol to be efficient.

The idea underlying our system is that the values $s$ and $t$ implicitly define several (pseudorandom) serial numbers $S_i$ and blinding values $B_i$, respectively. In other words, we need a pseudorandom function $F_{(\cdot)}$ such that we can set $S_i = F_s(i)$, and $B_i = F_t(i)$, $0 \leq i < 2^\ell$. Then the user gets $2^\ell$ pseudorandom serial numbers with the corresponding double-spending equations defined by $(s, t)$. Here, the double-spending equation for coin $i$ is $T_i = g^{sk_{\mathcal{U}}}(B_i)^R$, where $R$ is chosen by the merchant. This leaves us with a very specific technical problem. The challenge is to find a pseudorandom function such that, given (1) a commitment to $(sk_{\mathcal{U}}, s, t)$; (2) a commitment to $i$; and (3) the values $S_i$ and $T_i$, the user can efficiently prove that she derived the values $S_i$ and $T_i$ correctly from $sk_{\mathcal{U}}$, $s$, and $t$, i.e., $S_i = F_s(i)$ and

$T_i = g^{sk_{\mathcal{U}}} \cdot (F_t(i))^{R_i}$ for some $0 \leq i < 2^\ell$ and public value $R_i$ provided by the merchant.

Recently, Dodis and Yampolskiy [95] proposed the following discrete-logarithm-based pseudorandom function (PRF): $F_s(x) = g^{1/(s+x+1)}$, where $s, x \in \mathbb{Z}_q$, and $g$ is a generator of a group $G$ of order $q$ in which the decisional Diffie-Hellman inversion problem is hard. (In the sequel, we denote this PRF as $F^{DY}_{(\cdot)}(\cdot)$.) Using standard methods for proving statements about discrete-logarithm representations, we obtain a zero-knowledge argument system for showing that a pair of values $(S_i, T_i)$ is of the form $S_i = F^{DY}_s(i)$ and $T_i = g^{sk_{\mathcal{U}}} \cdot (F^{DY}_t(i))^{R_i}$ corresponding to the seeds $s$ and $t$ signed by bank $\mathcal{B}$ and to some index $i \in [0, 2^\ell - 1]$.

Note that if $S_i$ and $T_i$ are computed this way, then they are elements of $G$ rather than of $\mathbb{Z}_q$. So this leaves us with the following protocol: to withdraw a coin, a user obtains a signature on $(sk_{\mathcal{U}}, s, t)$. During the spending protocol, the user reveals $S_i$ and the double-spending equation $T_i = g^{sk_{\mathcal{U}}} \cdot (B_i)^{R_i}$, where $sk_{\mathcal{U}}$ is the user's secret key and $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}}$ the corresponding public key. Now, the bank may quickly detect double-spending whenever two coins are deposited with the same serial number, and using the two corresponding double-spending equations $T_1 = g^{sk_{\mathcal{U}}} \cdot B_i^{R_1}$ and $T_2 = g^{sk_{\mathcal{U}}} \cdot B_i^{R_2}$ we can infer the value $(T_1^{R_2}/T_2^{R_1})^{(R_2-R_1)^{-1}} = (pk_{\mathcal{U}}^{R_2} \cdot B_i^{R_1R_2}/(pk_{\mathcal{U}}^{R_1} \cdot B_i^{R_1R_2}))^{(R_2-R_1)^{-1}} = (pk_{\mathcal{U}}^{R_2-R_1})^{(R_2-R_1)^{-1}} = pk_{\mathcal{U}}$. This is sufficient to detect and identify double spenders. We describe this construction in more depth in Section 6.4.

However, the above scheme does not allow the bank to identify the other spendings of the coin, i.e., to generate all the serial numbers that the user can derive from $s$. This feature, called tracing, is motivated by Jarecki and Shmatikov [124] to balance the rational need of law enforcement to observe the transactions of *double-spenders*, while maintaining complete privacy for *honest* users. Let us now describe how we achieve this. For the moment, let us assume that the technique described above allows us to infer $sk_{\mathcal{U}}$ rather than $pk_{\mathcal{U}}$. If this were the case, we could require that the user, as part of the withdrawal protocol, should verifiably encrypt [5, 47, 58] the value $s$ *under her own* $pk_{\mathcal{U}}$, to form a ciphertext $c$. The record $(pk_{\mathcal{U}}, c)$ is stored by the bank. Now, suppose that at a future point, the user spends too many coins and thus her $sk_{\mathcal{U}}$ is discovered. From this, her $pk_{\mathcal{U}}$ can be inferred and the record $(pk_{\mathcal{U}}, c)$ can be located. Now that $sk_{\mathcal{U}}$ is known, $c$ can be decrypted, the seed $s$ discovered, the values $S_i$ computed for all $0 \leq i < 2^\ell$, and hence the database of transactions can be searched for records with these serial numbers.

Let us now redefine the way a user's keys are picked such that we can recover $sk_{\mathcal{U}}$ rather than $pk_{\mathcal{U}}$. Suppose that the group previously referred to as $G$ is replaced by $\mathbb{G}_1$, where we have a bilinear map $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $sk_{\mathcal{U}}$ be an element of $\mathbb{Z}_q$. Let $pk_{\mathcal{U}} = \mathsf{e}(g_1, g_2^{sk_{\mathcal{U}}}) \in \mathbb{G}_T$. Using ideas from Chapter 5, we know how to realize a cryptosystem that uses $pk_{\mathcal{U}}$ as a public key, such that in order to decrypt, it is sufficient to know the value $g_1^{sk_{\mathcal{U}}} \in \mathbb{G}_1$. So, in our scheme, the user $\mathcal{U}$ would encrypt $s$ under $pk_{\mathcal{U}}$ using the cryptosystem $\Sigma_2$ from Chapter 5. From the double-spending equations, the same way as before, the bank infers the value $g_1^{sk_{\mathcal{U}}}$. This value now allows the bank to decrypt $s$.

This is almost the solution, except for the following subtlety: if $\mathbb{G}_1$ has a bilinear map, then the decisional Diffie-Hellman problem may be easy, and so the Dodis-Yampolskiy construction is not a PRF in this setting! We have two solutions for this problem.

First, we can hope that there are bilinear maps where DDH remains hard in $\mathbb{G}_1$ (when there is not an efficiently-computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$), apply one of these bilinear groups, and keep using the Dodis-Yampolskiy PRF. In fact, this idea is already formalized as the External Diffie-Hellman (XDH) [107, 171, 143, 27, 12] assumption, and there is growing evidence that it may hold for bilinear groups instantiated with ordinary curves [147, 12][27, Sec. 8.1], even though this is known to be false for supersingular curves [110].

As a second solution, we can instead assume Sum-Free Decisional Diffie-Hellman [89], which is conjectured to hold for all bilinear groups, and slightly change the construction. This is why this variant of our scheme is a factor of $\ell$ more expensive than the others, where wallets contain $2^\ell$ coins. The details of this construction are given in Section 6.5.

In Section 6.8, we discuss some of the big open problems for electronic cash which this chapter does not address, such as efficiently allowing for multiple denominations in a non-trivial way; i.e., without executing the spending protocol a number of times. One of these open problems, supporting bounded-anonymity with a trusted party, is solved in Chapter 7.

## 6.2 Definition of Security

**Notation:** if $P$ is a protocol between $A$ and $B$, then $P(A(x), B(y))$ denotes that $A$'s input is $x$ and $B$'s is $y$.

Our electronic cash scenario consists of the three usual players: the user, the bank, and the merchant; together with the algorithms: BKeygen, UKeygen, Withdraw, Spend, Deposit, DetectViolation, IdentifyViolator, VerifyViolation, Trace, VerifyOwnership. Let us give some input-output specifications for these protocols, as well as some intuition for what they do.

- The $\mathsf{BKeygen}(1^k, \mathit{params})$ algorithm is a key generation algorithm run by the bank $\mathcal{B}$. It takes as input the security parameter $1^k$ and, if the scheme is in the common parameters model, it also takes as input these parameters *params*. This algorithm outputs the key pair $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$. (Assume that $pk_{\mathcal{B}}$ and $sk_{\mathcal{B}}$ contain the *params*, so we do not have to give *params* explicitly to the bank or parties interacting with the bank again.)

- Similarly, $\mathsf{UKeygen}(1^k, \mathit{params})$ is a key generation algorithm run by the user $\mathcal{U}$, which outputs $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$. Since merchants are a subset of users, they may use this algorithm to obtain keys as well. (Assume that $pk_{\mathcal{U}}$ and $sk_{\mathcal{U}}$ contain the *params*, so we do not have to give *params* explicitly to the user or parties interacting with the user again.)

- In the $\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_{\mathcal{U}}, n), \mathcal{B}(pk_{\mathcal{U}}, sk_{\mathcal{B}}, n))$ protocol, the user $\mathcal{U}$ withdraws a *wallet* $W$ of $n$ coins from the bank $\mathcal{B}$. The user's output is the wallet $W$, or an error message. $\mathcal{B}$'s output is some information $T_W$ which will allow the bank to trace the user should this user double-spend some coin, or an error message. The bank maintains a database $D$ for this trace information, to which it enters the record $(pk_{\mathcal{U}}, T_W)$.

- In a $\mathsf{Spend}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ protocol, a user $\mathcal{U}$ gives one of the coins from his wallet $W$ to the merchant $\mathcal{M}$. Here, the merchant obtains a serial number $S$ of the coin, and a proof $\pi$ of validity of the coin. The user's output is an updated wallet $W'$.

- In a $\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, S, \pi, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$ protocol, a merchant $\mathcal{M}$ deposits a coin $(S, \pi)$ into its account held by the bank $\mathcal{B}$. Whenever an honest $\mathcal{M}$ obtained $(S, \pi)$ by running the Spend protocol with any (honest or otherwise) user, there is a guarantee that this coin will be accepted by the bank. $\mathcal{B}$ adds $(S, \pi)$ to to its list $L$ of spent coins. The merchant's output is nothing or an error message.

- The $\mathsf{DetectViolation}(params, L)$ algorithm detects double-spending in the list $L$ of spent coins whenever two coins have the same serial number.

- The $\mathsf{IdentifyViolator}(params, S, \pi_1, \pi_2)$ algorithm allows to identify double-spenders using a serial number $S$ and two proofs of validity of this coin, $\pi_1$ and $\pi_2$, possibly submitted by malicious merchants, i.e., merchants that deviate from the protocols. This algorithm outputs a public key $pk_U$ and a proof $\Pi$. If the merchants who had submitted $\pi_1$ and $\pi_2$ are *not* malicious, then $\Pi$ is evidence that $pk_{\mathcal{U}}$ is the registered public key of a user that double-spent coin $S$.

- The $\mathsf{VerifyViolation}(params, S, pk_{\mathcal{U}}, \Pi)$ algorithm allows anyone to verify proof $\Pi$ that the user with public key $pk_{\mathcal{U}}$ is guilty of double-spending coin $S$.

- The $\mathsf{Trace}(params, S, pk_{\mathcal{U}}, \Pi, D, n)$ algorithm, given a public key $pk_{\mathcal{U}}$ of a double-spender, a proof $\Pi$ of his guilt in double-spending coin $S$, the database $D$, and a wallet size $n$, computes the serial numbers $S_1, \ldots, S_m$ of all of the coins issued to $\mathcal{U}$ along with proofs $\Gamma_1, \ldots, \Gamma_m$ of $pk_{\mathcal{U}}$'s ownership. If $\mathsf{VerifyViolation}(params, S,\ pk_{\mathcal{U}},\ \Pi)$ does not accept (i.e., $pk_{\mathcal{U}}$ is honest), this algorithm does nothing.

- The $\mathsf{VerifyOwnership}(params, S, \Gamma, pk_{\mathcal{U}}, n)$ algorithm allows anyone to verify the proof $\Gamma$ that a coin with serial number $S$ belongs to a double-spender with public key $pk_{\mathcal{U}}$.

We will now informally define the security properties for the casual reader. The more interested reader will want to skip to the more elaborate formal definitions given in Section 6.2.1.

*Correctness.* If an honest user runs Withdraw with an honest bank, then neither will output an error message; if an honest user runs Spend with an honest merchant, then the merchant accepts the coin.

*Balance.* From the bank's point of view, what matters is that no collection of users and merchants can ever spend more coins than they withdrew. We require that there is a knowledge extractor $\mathcal{E}$ that executes $u$ Withdraw protocols with all adversarial users and extracts $(u \cdot n)$ serial numbers $S_1, \ldots, S_{u \cdot n}$. We require that for every adversary, the probability that an honest bank will accept $(S, \pi)$ as the result of the Deposit protocol, where $S \neq S_i\ \forall 1 \leq i \leq u \cdot n$, is negligible. If $S_1, \ldots, S_n$ is a set of serial numbers output by $\mathcal{E}$ when running Withdraw with public key $pk_{\mathcal{U}}$, we say that coins $S_1, \ldots, S_n$ *belong* to the user $\mathcal{U}$ with $pk_{\mathcal{U}}$.

*Identification of double-spenders.* Suppose $\mathcal{B}$ is honest. Suppose $\mathcal{M}_1$ and $\mathcal{M}_2$ are honest merchants who ran the Spend protocol with the adversary, such that $\mathcal{M}_1$'s output is $(S, \pi_1)$

and $\mathcal{M}_2$'s output is $(S, \pi_2)$. This property guarantees that $\mathsf{IdentifyViolator}(params, S, \pi_1, \pi_2)$ outputs a key $pk_{\mathcal{U}}$ and proof $\Pi$ such that $\mathsf{VerifyViolation}(params,\ S,\ pk_{\mathcal{U}}, \Pi)$ accepts with high probability.

*Tracing of double-spenders.* Given that a user $\mathcal{U}$ is shown guilty of double-spending coin $S$ by a proof $\Pi$ such that VerifyViolation accepts, this property guarantees that $\mathsf{Trace}(params, S, pk_{\mathcal{U}}, \Pi, D, n)$ will output the serial numbers $S_1, \ldots, S_m$ of all coins that belong to $\mathcal{U}$ along with proofs of ownership $\Gamma_1, \ldots, \Gamma_m$ such that for all $i$, $\mathsf{VerifyOwnership}(params, S_i, \Gamma_i, pk_{\mathcal{U}}, n)$ also accepts with high probability.

*Anonymity of users.* From the privacy point of view, what matters to users is that the bank, even when cooperating with any collection of malicious users and merchants, cannot learn anything about a user's spendings other than what is available from side information from the environment. In order to capture this property more formally, we introduce a simulator $\mathcal{S}$. $\mathcal{S}$ has some side information not normally available to players. E.g., if in the common parameters model, $\mathcal{S}$ generated these parameters; in the random-oracle model, $\mathcal{S}$ is in control of the random oracle; in the public-key registration model $\mathcal{S}$ may hold additional information about the bank's keys, etc. We require that $\mathcal{S}$ can create simulated coins without access to any wallets, such that a simulated coin is indistinguishable from a valid one. More precisely, $\mathcal{S}$ executes the user's side of the Spend protocol without access to the user's secret or public key, or his wallet $W$.

*Exculpability.* Suppose that we have an adversary that participates any number of times in the Withdraw protocol with the honest user with public key $pk_{\mathcal{U}}$, and subsequently to that, in any number of legal Spend protocols with the same user. I.e., if the user withdrew $u$ wallets of $n$ coins each, then this user can participate in at most $u \cdot n$ Spend protocols. The adversary then outputs a coin serial number $S$ and a purported proof $\Pi$ that either the user with public key $pk_{\mathcal{U}}$ double-spent coin $S$ or the same user simply owns coin $S$. The *weak* exculpability property postulates that, for all adversaries, the probability $\mathsf{VerifyViolation}(params, S, pk_{\mathcal{U}}, \Pi, n)$ or $\mathsf{VerifyOwnership}(params, S, \Pi, pk_{\mathcal{U}}, n)$ accepts is negligible.

Furthermore, the adversary may continue to engage the user $\mathcal{U}$ in Spend protocols even if it means $\mathcal{U}$ must double-spend some coins of her choosing. If forced to double spend, the user resets the state of her wallet to the state it was in immediately after the wallet was withdrawn. This keeps the method by which a user is forced to double spend well defined. The adversary then outputs $(S, \Pi)$. The *strong* exculpability property postulates that, for all adversaries, when $S$ is a coin serial number *not* belonging to $\mathcal{U}$, the weak exculpability property holds, and when $S$ is a coin serial number *not* double-spent by user $\mathcal{U}$ with public key $pk_{\mathcal{U}}$, the probability that $\mathsf{VerifyViolation}(params, S, \Pi, pk_{\mathcal{U}}, n)$ accepts is negligible.

This ends the informal description of our security definition. This informal description should be sufficient for a general understanding of our security guarantees. We now provide more precise definitions.

### 6.2.1 Formal Definitions

Our goal is to give a formal definition for electronic cash. We want to obtain a definition that would make sense independently of whether a given scheme is realized in the plain model, common random string model, common parameters model, random-oracle model, or any other variety of a model. Let us say a few words on these models.

The standard method [112, 114, 113] for proving that a protocol is zero-knowledge with respect to a value $w$ is to describe a simulator $\mathcal{S}$ that, without knowing $w$, cannot be distinguished from a party with knowledge of $w$. Likewise, the method for proving that a party $A$ "knows" a value $w$ is to describe an extractor $\mathcal{E}$ that, by interacting with $A$ according to some set rules, can extract the value $w$. Of course, for these protocols to be non-trivial, $\mathcal{S}$ and $\mathcal{E}$ require some additional power that is not available to parties in the real world. (We will be more exact about this momentarily.) For example, simulators and extractors are (usually) allowed to interact with a party $A$, reset party $A$ to its state before this interaction, and then interact with party $A$ again. The various models, many of which emerged before they were named, are an attempt to describe the additional power $\mathcal{S}$ and $\mathcal{E}$ must be given. In general, the less power given, the better – because it may make the protocol difficult (or impossible!) to implement in the real world.

We briefly describe a few of the most commonly used models. In the common random string model, all parties are given access to a public random string; in the proof of security, this string may be chosen by $\mathcal{S}$ and $\mathcal{E}$. In the common parameters model, all parties are given access to some public parameters (which may not be random); in the proof of security, these parameters may be chosen by $\mathcal{S}$ and $\mathcal{E}$. In the random oracle model, all parties are given access to an oracle that acts as a hash function; in the proof of security, this oracle is controlled by $\mathcal{S}$ and $\mathcal{E}$. In the plain model, all parties start the protocol without any common information or access to an oracle.

From the informal definitions, the reader may recall that our definition of e-cash relies on the existence of a knowledge extractor $\mathcal{E}$ that extracts the serial number of all the coins that an adversarial user withdraws in a given Withdraw transaction; and the existence of the simulator $\mathcal{S}$ which simulates the malicious merchant's view of the Spend protocol without access to any users' keys or wallets.

Usually, the way that knowledge extractors and zero-knowledge simulators are defined depends on the model in which security is proven, which may vary from construction to construction.

In order to obtain a definition that works equally well in *any* model, we require that a particular protocol is a proof of knowledge or a zero-knowledge proof in the model in which security of the construction is proven. For example, we require that part of the Withdraw protocol is a proof of knowledge of the serial numbers of the coins withdrawn by the user, or part of the Spend protocol is a zero-knowledge proof.

In whatever model we are working, however, we will need concurrently composable proofs of knowledge and concurrently composable zero-knowledge proofs [63, 64, 65].

Let us explain how to specify a knowledge extractor $\mathcal{E}_{\mathsf{Prot},l}$ for a proof of knowledge protocol Prot for language $l$. (resp., zero-knowledge simulator $\mathcal{S}_{\mathsf{Prot},l}$ for protocol Prot for

proving membership in language $l$) in a (relatively) model-independent fashion. First, as we mentioned above in some models, such as a common parameters model, an extractor or simulator is allowed access to some auxiliary information not provided to a participant in the protocol. We denote denote these by *auxext* and *auxsim*, respectively.

Another resource that a knowledge extractor or zero-knowledge simulator must receive is access to the adversary. This includes many possibilities; here are a few examples:

- (IO) Input-output interaction with the adversary. This is typical in the UC-framework [63, 64].
- (BB) Black-box interaction with the adversary. This means that the adversary can be reset to a previous state.
- (NBB) Non-black-box access. This means that we are given the description of the adversary.

Each of these access models can be further augmented with the random-oracle model. For example, the IO-RO model includes input-output access to the adversary's hash function module.

Let $X$-$Y(\mathsf{Adv})$ be our access model to the adversary $\mathsf{Adv}$, where, for example, $X \in \{IO, BB, NBB\}$, and $Y \in \{\varepsilon, RO\}$. By $\mathsf{Alg}^{X\text{-}Y(\mathsf{Adv})}$ we denote that the algorithm $\mathsf{Alg}$ has this type of access to the adversary $\mathsf{Adv}$.

So, for example, if $\mathsf{Alg}$ is an algorithm interacting with adversary $\mathsf{Adv}$ in the $X$-$Y$ model, running on input $x$, then we denote it by $\mathsf{Alg}^{X\text{-}Y(\mathsf{Adv})}(x)$.

In the sequel, by "proof protocol," we mean a protocol between a prover and a verifier.

Thus, a knowledge extractor for proof protocol $\mathsf{Prot}$ for language $l$ in the $X$-$Y$ model would be denoted as $\mathcal{E}_{\mathsf{Prot},l}^{X\text{-}Y(\mathsf{Adv})}(params, auxext, x)$. By definition of what it means to be a knowledge extractor, for properly formed $(params, auxext)$, $\mathcal{E}_{\mathsf{Prot},l}^{X\text{-}Y(\mathsf{Adv})}(params, auxext, x)$ will, with high probability, in expected polynomial time, output a $w$ such that $(x, w) \in l$ whenever the probability that the verifier accepts $x$ when interacting with $\mathsf{Adv}$ in the $X$-$Y$ model, is non-negligible.

Similarly, a zero-knowledge simulator for proof protocol $\mathsf{Prot}$ for language $l$ in the $X$-$Y$ model would be denoted as $\mathcal{S}_{\mathsf{Prot},l}^{X\text{-}Y(\mathsf{Adv})}(params, auxsim, x)$. By definition of what it means to be a zero-knowledge simulator, $\mathcal{S}_{\mathsf{Prot},l}^{X\text{-}Y(\mathsf{Adv})}(params, auxsim, x)$ will, when interacting with $\mathsf{Adv}$ in the $X$-$Y$ model, produce a view that is indistinguishable from the view $\mathsf{Adv}$ obtains when interacting with the prover.

**Balance.** Let a $\mathsf{Withdraw}$ protocol be given. Recall that the bank's input to this protocol includes the user's public key $pk_{\mathcal{U}}$. Let us break up the $\mathsf{Withdraw}$ protocol into three parts: $\mathsf{Withdraw}_b$, $\mathsf{Withdraw}_m$, and $\mathsf{Withdraw}_e$ ("b," "m," and "e" stand for "beginning," "middle," and "end," respectively). $\mathsf{Withdraw}_b$ is the part of the protocol that ends with the first message from the user to the bank. $\mathsf{Withdraw}_e$ part of the protocol is the last message from the bank to the user.

The middle of the protocol is denoted $\mathsf{Withdraw}_m$. Let us think of $\mathsf{Withdraw}_e$ as a proof protocol, where the user is the Prover, and the bank is the Verifier. The Verifier's output of $\mathsf{Withdraw}_m$ can be thought of as the bank's decision for whether or not to proceed to $\mathsf{Withdraw}_e$ and send the last message from the bank to the user.

Let $m_1$ denote the first message that the user sends to the bank in the Withdraw protocol. Let $b_1$ denote the state information of the bank at the moment that $m_1$ was received.

The balance property requires that:

1. In whatever model the scheme is given, there exists an efficiently decidable language $l_S$ and an extractor $\mathcal{E}^{X\text{-}Y(\mathsf{Adv})}_{\mathsf{Withdraw}_m, l_S}(params, auxext, pk_{\mathcal{U}}, b_1, m_1)$ such that for all $b_1$ computed by the bank, and for all $m_1$, it extracts $w = (S_1, \ldots, S_n, aux)$ such that $(b_1, m_1, w) \in l_S$ whenever the probability that the bank accepts in the $\mathsf{Withdraw}_m$ part of the protocol is non-negligible. We say that the extractor outputs $(b_1, m_1, w) \in l_S$ in that case.
2. (If we are in the public parameters model, assume that the values *params* and *auxext* are fixed. Assume that $pk_{\mathcal{B}}$ was generated appropriately.)

   On input $(params, pk_{\mathcal{B}})$, the adversary Adv plays the following game: Adv executes the Withdraw and Deposit protocols with the bank as many time as it wishes. (It can simulate running the Spend protocol with itself.) Let $(b_{1,i}, m_{1,i}, w_i) \in l_S$ be the output of $\mathcal{E}^{X\text{-}Y(\mathsf{Adv})}_{\mathsf{Withdraw}_m, l_S}(params, auxext, pk_i, b_{1,i}, m_{1,i})$ if the $i$th withdrawal protocol was successful. Recall that $w_i = (S_{i,1}, \ldots, S_{i,n}, aux)$ is a list of $n$ serial numbers; we say that these *belong* to $pk_i$. Let $A_\ell = \{S_{i,j} \mid 1 \le i \le \ell, 0 \le j \le n-1\}$ be the list of serial numbers after $\ell$ executions of the Withdraw protocol. Adv wins the game if for some $\ell$, in some Deposit protocol, the honest bank accepts a coin with serial number $S \notin A_\ell$. We require that no probabilistic polynomial-time adversary succeeds in this game with non-negligible probability.

**Identification of double-spenders.** This property guarantees that no probabilistic polynomial time (PPT) adversary has a non-negligible probability of winning the following game:

(If we are in the public parameters model, assume that the values *params* and *auxext* are fixed; assume that $pk_{\mathcal{B}}$ was generated appropriately.)

On input $(params, pk_{\mathcal{B}})$, the adversary Adv plays the following game: Adv executes the Withdraw protocol and Spend protocol with the bank as many time as it wishes. Let $(b_{1,i}, m_{1,i}, w_i) \in l_S$ be the output of $\mathcal{E}^{X\text{-}Y(\mathsf{Adv})}_{\mathsf{Withdraw}_m, l_S}(params, auxext, pk_i, b_{1,i}, m_{1,i})$ if the $i$th withdrawal protocol was successful. Recall that $w_i = (S_{i,1}, \ldots, S_{i,n}, aux)$ is a list of $n$ serial numbers: recall that we say that these *belong* to $pk_i$. Let $A_i$ be the list of serial numbers that belong to public key $pk_i$. Adv wins the game if for some $\ell$, in some Spend protocol, the bank, simulating an honest merchant, accepts a coin with serial number $S \in A_\ell$ twice, i.e. outputs $(S, \pi_1)$ and $(S, \pi_2)$ triggering DetectViolation to accept, and

yet $\mathsf{IdentifyViolator}(params, S, \pi_1, \pi_2)$ output a public key $pk$ and a proof $\Pi$ such that $\mathsf{VerifyViolation}(params, S, pk, \Pi)$ does not accept.

**Tracing of double-spenders.** Here, we play the same sort of game with the adversary as above, but we are more generous as to what constitutes the adversary's success. Suppose that for some $\ell$, in some Spend protocol, the honest merchant accepts a coin with serial number $S$ twice, i.e. outputs $(S, \pi_1)$ and $(S, \pi_2)$, and computes $\mathsf{IdentifyViolator}(params, S, \pi_1, \pi_2) \rightarrow (pk_{\mathcal{U}}, \Pi)$. Let $\{S_i\}$ be the set of serial numbers and $\{\Gamma_i\}$ be the set of ownership proofs output by $\mathsf{Trace}(params, S, pk_{\mathcal{U}}, \Pi, D, n)$. Adv wins if (1) there exists some serial number $S' \in A_\ell$ such that $S' \notin \{S_i\}$, or (2) there exists a serial number $S_j \in \{S_i\}$ such that $\mathsf{VerifyOwnership}(params, S_j, \Gamma_j, pk_{\mathcal{U}}, n)$ does not accept.

**Anonymity of users.** (If we are in the public parameters model, assume that the values *params* and *auxsim* are fixed; *auxsim* is a value available to a simulator but not available to regular participants in this system.)

Here, let us consider an adversary Adv that forms the bank's public key $pk_{\mathcal{B}}$ and then issues the following queries, as many as it wants, in any order:

**PK of $i$** In this query, Adv may request and receive the public key $pk_i$ of user $i$, generated honestly as $(pk_i, sk_i) \leftarrow \mathsf{UKeygen}(1^k, params)$.

**Withdraw with $i$** In this query, Adv executes the Withdraw protocol with user $i$:

$$\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_i, n), \mathsf{Adv}(state, n))$$

where *state* is Adv's state; let us denote the user's output after the $j$'th Withdraw query by $W_j$; note that $W_j$ may be an error message, in that case we say that $W_j$ is an *invalid* wallet.

**Spend from wallet $j$** In this query, if wallet $W_j$ is defined, Adv executes the Spend protocol from this wallet: $\mathsf{Spend}(\mathcal{U}(W_j), \mathsf{Adv}(state))$, where *state* is the adversary's state.

We say that Adv is *legal* if it only spends from valid wallets, and *never* asks to spend more than $n$ coins from the same wallet $W_j$.

We require the existence of a simulator $\mathcal{S}^{X\text{-}Y(\cdot)}(params, auxsim, \cdot)$ such that for all $pk_{\mathcal{B}}$, no legal adversary Adv can distinguish whether he is playing Game R(eal) or Game I(deal) below with non-negligible advantage:

**Game R** The queries Adv issues are answered as described above.

**Game I** The PK and Withdraw queries Adv issues are answered as described above, but in the Spend query, instead of interacting with $\mathcal{U}(W_j)$, Adv interacts with the simulator $\mathcal{S}^{X\text{-}Y(\mathsf{Adv})}(params, auxsim, pk_{\mathcal{B}})$.

(The reason this captures anonymity is that the simulator does not know on behalf of which honest user he is spending the coin.)

**Exculpability.** Recall that the exculpability property guarantees that only users who really are guilty of double-spending a coin can ever get convicted of being a double-spender. Exculpability comes in two flavors: *weak* exculpability means that only users who double-spent *some* coin can be convicted; while *strong* exculpability means that even a guilty user who double-spent some coins cannot be convicted of double-spending other coins, i.e., his guilt can be quantified and he can be punished according to guilt.

To define weak exculpability, we have the adversary Adv launch the following attack against a user $\mathcal{U}$ (assume $n$ is fixed):

**Setup** The system parameters *params* are generated and the user's keys $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$ are chosen. The adversary chooses the bank's public key $pk_{\mathcal{B}}$ in an adversarial fashion.

**Queries** The adversary Adv issues queries to interact with the user $\mathcal{U}$, as follows:

**Withdraw wallet** $j$ The adversary plays the bank's side of the Withdraw protocol with the user $\mathcal{U}$. The user outputs the wallet $W_j$.

**Spend from wallet** $j$ The adversary plays the merchant's side of the Spend protocol where $\mathcal{U}$'s input is wallet $W_j$. For each wallet, the adversary is allowed to execute this query up to $n$ times.

**Success criterion** In the end, the adversary Adv outputs the values $(S, \Gamma, \Pi)$ and wins the game if VerifyViolation$(params, S, \Pi, pk_{\mathcal{U}}, n)$ or VerifyOwnership$(params, S, \Gamma, pk_{\mathcal{U}}, n)$ accepts.

We say that an electronic cash scheme guarantees weak exculpability if the probability that the adversary wins this game is negligible.

To define strong exculpability, we first insist that corresponding to any wallet $W$ obtained by an honest user even from an adversarial bank, there exist at most $n$ valid serial numbers. Denote the set of serial numbers corresponding to $W$ by $A_W$.

The strong exculpability game is somewhat similar to the weak exculpability game: the setup and the withdraw query are the same. The only things that are different are as follows:

**Queries** The adversary issues queries to interact with the user $\mathcal{U}$, in addition to Withdraw above, as follows:

**Spend from wallet** $j$ The adversary plays the merchant's side of the Spend protocol where $\mathcal{U}$'s input is wallet $W_j$. For each wallet, the adversary is allowed to execute this query as many times as he wishes. Recall that the wallet keeps state information about how many coins have already been spent, and a wallet where $n$ coins have been spent is not well-defined. To make this

query well-defined in that case, let us say that once $n$ coins are spent from a wallet $W_j$, the state information is reset such that it is the same as it was before any coins were spent.

As a result of this query, the user may produce a serial number that she had already produced before. Let $A_j$ be the set of all serial numbers that the user has produced in a Spend protocol for wallet $j$ so far. If for any $j$, $|A_j| > n$, then the adversary is given the sets of serial numbers $A_{W_{j'}}$ for *all* wallets $W_{j'}$. (This is what the adversary is entitled to because of the traceability requirement.) Let $A'_j$ be the set of serial numbers for wallet $W_j$ that are known to the adversary. Thus $A'_j = A_j$ if no double-spending has ever occurred, and $A'_j = A_{W_j}$ otherwise.

Let $A_{ds}$ be the set of serial numbers that the user has produced more than once (i.e., double-spent) so far.

**Success criterion** In the end, the adversary outputs the values $(S, \Pi, \Gamma)$ and wins the game if *one* of the following conditions is satisfied:

1. The adversary made up a bogus serial number and managed to pin it on the user; i.e., for all $j$, $S \notin A'_j$ and yet (a) VerifyOwnership($params, S, \Gamma, pk_{\mathcal{U}}, n$) or (b) VerifyViolation($params, S, \Pi, pk_{\mathcal{U}}, n$) accepts.
2. The user has spent fewer than $n$ coins from all of his wallets, and yet the adversary managed to produce proof that the user owns or double-spent some coin in wallet $j$; i.e., $|A_{j'}| < n$ for all $j'$, $S \in A_j$ for some $j$, and yet (a) VerifyOwnership($params, S, \Gamma, pk_{\mathcal{U}}, n$) or (b) VerifyViolation($params, S, \Pi, pk_{\mathcal{U}}, n$) accepts.
3. The adversary successfully accuses the user of double-spending a coin which was *not* in fact double-spent: $S \notin A_{ds}$ and VerifyViolation($params, S, \Pi, pk_{\mathcal{U}}, n$) accepts.

In our model, a person *is* their secret key $sk_{\mathcal{U}}$, so any coins obtained by a party in possession of $sk_{\mathcal{U}}$ belong to $pk_{\mathcal{U}}$, and any coins double-spent with knowledge of $sk_{\mathcal{U}}$ were double-spent by $pk_{\mathcal{U}}$. For our particular construction in Section 6.5 (System Two), where one part of $sk_{\mathcal{U}}$ is revealed after double-spending, we define party $\mathcal{U}$ as those persons in possession of *both* parts of $sk_{\mathcal{U}}$.

**Strengthening the definition: the UC framework.** Even though our definition of security is not in the UC framework, note that our definition would imply UC-security [61, 63, 64] whenever the extractor $\mathcal{E}$ and simulator $\mathcal{S}$ are constructed appropriately. In a nutshell, an ideal electronic cash functionality would allow an honest user to withdraw and spend $n$ coins. In this case, if the merchant and bank are controlled by the malicious environment, the simulator $\mathcal{S}$ defined above creates the merchant's and bank's view of the Spend protocol. At the same time, the balance property guarantees that the bank gets the same protection in the real world as it does in the ideal world, and the exculpability property ensures that

an honest user cannot get framed in the real world, just as he cannot get framed in the ideal world.

One promising approach to realizing an ideal functionality for electronic cash is to take a modular approach to its definition. That is, to analyze our scheme (or another one) in a hybrid setting where all parties are given access to ideal functionalities for zero-knowledge and signatures of knowledge [72]. (For more on where signatures of knowledge come into play, see Section 6.6.) This way whenever a UC-secure implementation of these ZK-related components is employed the entire cash system will be UC-secure as well.

## 6.3 Preliminaries

Our e-cash systems use a variety of known protocols as building blocks, which we now briefly review. Many of these protocols can be shown secure under several different complexity assumptions, a flexibility that will extend to our e-cash systems.

### 6.3.1 Known Discrete-Logarithm-Based, Zero-Knowledge Proofs

In the common parameters model, we use several previously known results for proving statements about discrete logarithms, such as (1) proof of knowledge of a discrete logarithm modulo a prime [169] or a composite [105, 86], (2) proof of knowledge of equality of representation modulo two (possibly different) prime [78] or composite [56] moduli, (3) proof that a commitment opens to the product of two other committed values [55, 60, 42], (4) proof that a committed value lies in a given integer interval [70, 55, 55, 37], and also (5) proof of the disjunction or conjunction of any two of the previous [84]. These protocols modulo a composite are secure under the strong RSA assumption and modulo a prime under the discrete logarithm assumption.

When referring to the proofs above, we will follow the notation introduced by Camenisch and Stadler [59] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For instance,

$$PK\{(\alpha, \beta, \delta) : y = g^{\alpha} h^{\beta} \wedge \tilde{y} = \tilde{g}^{\alpha} \tilde{h}^{\delta} \wedge (u \leq \alpha \leq v)\}$$

denotes a "*zero-knowledge Proof of Knowledge of integers $\alpha$, $\beta$, and $\delta$ such that $y = g^{\alpha} h^{\beta}$ and $\tilde{y} = \tilde{g}^{\alpha} \tilde{h}^{\delta}$ holds, where $u \leq \alpha \leq v$,*" where $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\tilde{\mathbb{G}} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. The convention is that Greek letters denote quantities of which knowledge is being proven, while all other values are known to the verifier.

We apply the Fiat-Shamir heuristic [98] to *efficiently* turn such proofs of knowledge into signatures on some message $m$; denoted as, e.g., $SPK\{(\alpha) : y = g^{\alpha}\}(m)$. Alternatively, it is possible to avoid the Fiat-Shamir heuristic using *inefficient* generic tools for robust zero knowledge, as we will discuss in more detail in the next section.

### 6.3.2 Pseudorandom Functions

A useful building block of our e-cash systems is the pseudorandom functions recently proposed by Dodis and Yampolskiy [95], which they expand to verifiable random functions using bilinear groups. Their construction is:

> For every $n$, a function $f \in F_n$ is defined by the tuple $(\mathbb{G}, q, g, s)$, where $\mathbb{G}$ is group of order $q$, $q$ is an $n$-bit prime, $g$ is a generator of $\mathbb{G}$, and $s$ is a seed in $\mathbb{Z}_q$. For any input $x \in \mathbb{Z}_q$ (except for $x = -s - 1 \bmod q$), the function $f_{\mathbb{G},q,g,s}(\cdot)$, which we simply denote as $f^{DY}_{g,s}(\cdot)$ for fixed values of $(\mathbb{G}, q, g)$, is defined as $f^{DY}_{g,s}(x) = g^{1/(s+x+1)}$.

Dodis and Yampolskiy [95] showed that the above construction is a pseudorandom function, under the $y$-DDHI assumption, when either: (1) the inputs are drawn from the restricted domain $\{0,1\}^{O(\log k)}$ only, or (2) the adversary specifies a polynomial-sized set of inputs from $\mathbb{Z}_q^*$ *before* a function is selected from the PRF family (i.e., before the value $s$ is selected).

This is not a problem for our e-cash system, because the DY PRF is evaluated only on the integers one to the wallet size $2^\ell$; thus, since wallets contain at most a polynomial number of coins in the security parameter, these values can always be submitted in advance of selecting the DY PRF from its function family. In the event that one wanted to use our e-cash system as an anonymous credential, and therefore might want to have wallets containing an exponential number of coins, that is also not a problem due to a recent result of Camenisch et al. [51], which proves that the DY construction is secure for inputs drawn arbitrarily and adaptively from $\mathbb{Z}_q$ under an assumption related to $y$-DDHI.

As mentioned in the introduction, we could instead substitute in the Naor-Reingold PRF [149], and replace the $y$-DDHI assumption with the more standard DDH assumption, at the cost of enlarging our wallets from $O(\ell + k)$ bits to $O(\ell \cdot k)$ bits.

### 6.3.3 Pedersen Commitments

Pedersen proposed a perfectly-hiding, computationally-binding commitment scheme [161] based on the discrete logarithm assumption, in which the public parameters are a group $\mathbb{G}$ of prime order $q$, and generators $(g_0, \ldots, g_m)$. In order to commit to the values $(v_1, \ldots, v_m) \in {\mathbb{Z}_q}^m$, pick a random $r \in \mathbb{Z}_q$ and set $C = \mathrm{PedCom}(v_1, \ldots, v_m; r) = g_0^r \prod_{i=1}^m g_i^{v_i}$.

Fujisaki and Okamoto [105] showed how to expand this scheme to composite order groups.

### 6.3.4 CL Signatures

Camenisch and Lysyanskaya [53] came up with a secure signature scheme with two protocols: (1) An efficient protocol between a user and a signer with keys $(pk_S, sk_S)$. The common input consists of $pk_S$ and $C$, a Pedersen commitment. The user's secret input is the set of values $(v_1, \ldots, v_\ell, r)$ such that $C = \mathrm{PedCom}(v_1, \ldots, v_\ell; r)$. As a result of the protocol, the user obtains a signature $\sigma_{pk_S}(v_1, \ldots, v_\ell)$ on his committed values, while the signer does not learn anything about them. The signature has size $O(\ell \cdot \log q)$. (2) An efficient proof of knowledge

of a signature protocol between a user and a verifier. The common inputs are $pk_S$ and a commitment $C$. The user's private inputs are the values $(v_1, \ldots, v_\ell, r)$, and $\sigma_{pk_S}(v_1, \ldots, v_\ell)$ such that $C = \mathrm{PedCom}(v_1, \ldots, v_\ell; r)$. These signatures are secure under the strong RSA assumption. For the purposes of this exposition, it does not matter *how* CL signatures actually work, all that matters are the facts stated above.

Our subsequent e-cash systems will require the strong RSA assumption independently of the CL signatures. By making additional assumptions based on bilinear groups, we can use alternative schemes by Camenisch and Lysyanskaya [54] and Boneh, Boyen and Shacham [27], yielding shorter signatures in practice.

### 6.3.5 Verifiable Encryption

In Section 6.5, we apply a technique by Camenisch and Damgård [47] for turning any semantically-secure encryption scheme into a verifiable encryption scheme. A verifiable encryption scheme is a two-party protocol between a prover/encryptor $\mathcal{P}$ and a verifier/receiver $\mathcal{V}$. Their common inputs are a public encryption key $pk$ and a commitment $A$. As a result of the protocol, $\mathcal{V}$ either rejects or obtains an encryption of the value committed to in $A$ under public key $pk$. The protocol ensures that $\mathcal{V}$ accepts an incorrect encryption only with negligible probability and that $\mathcal{V}$ learns nothing meaningful about the value committed to in $A$. Together with the corresponding secret key $sk$, transcript of this protocol contains enough information to recover the value committed to in $A$ efficiently. We hide some details here and refer to Camenisch and Damgård [47] for the full discussion.

### 6.3.6 Bilinear El Gamal Encryption

We apply the verifiable encryption techniques above to the basic cryptosystem $\Sigma_2$ from Chapter 5. Let us review the pieces we will use. Let $\mathsf{Bilinear_Setup}$ on $1^k$ output $\gamma = (q, g_1, h_1, \mathbb{G}_1,\ g_2,\ h_2,\ \mathbb{G}_2,\ \mathbb{G}_T, \mathsf{e})$, where we have bilinear map $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $(G, E, D)$ denote the standard key generation, encryption, and decryption algorithms of $\Sigma_2$. On input $(1^k, \gamma)$, the key generation algorithm $G$ outputs a key pair $(pk, sk) = (\mathsf{e}(g_1, g_2)^u, g_1^u)$ for a random $u \in \mathbb{Z}_q$. The main idea is that the value $g_1^u$ is enough to decrypt.

To encrypt a message $m \in \mathbb{G}_T$ under $pk$, select a random $k \in \mathbb{Z}_q$ and output the ciphertext $c = (g_2^k, pk^k \cdot m) = (g_2^k, \mathsf{e}(g_1, g_2)^{u \cdot k} \cdot m)$. Then, to decrypt $c = (c_1, c_2)$ with the value $g_1^u$, simply compute $c_2/\mathsf{e}(g_1^u, c_1)$. This encryption scheme is semantically-secure under the Decisional Bilinear Diffie-Hellman (DBDH) assumption [30], i.e., given $(g_2, g_2^a, g_2^b, g_2^c, X)$ for random $a, b, c \in \mathbb{Z}_q$ and $X \in \mathbb{G}_T$, it is hard to decide if $X = \mathsf{e}(g_1, g_2)^{a \cdot b \cdot c}$. (The extended DBDH assumption is only needed for the re-encryption functionality.) In Appendix A of the full version of their paper, Boneh, Goh, and Boyen [26] show that DBDH (and its extended version) are implied by the $y$-DDHI assumption.

Finally, it is worth mentioning that the only property we require here is that the decryption key be of the form $g_1^u$. Boneh and Franklin presented the first such cryptosystem [30].

## 6.4 System One: Compact E-Cash

System One supports the basic algorithms (BKeygen, UKeygen, Withdraw, Spend, Deposit, DetectViolation, IdentifyViolator, VerifyViolation). In this scheme, a wallet of size $O(\ell + k)$ is sufficient to hold $2^\ell$ coins. Double-spending allows the bank to reveal the public key of the user, but *not* to trace her past transactions.

**Global parameters for System One.** Let $1^k$ be the security parameter and let $2^\ell$ be the public wallet size. This scheme works most efficiently by having two different groups:

- $\mathbf{G} = \langle \mathbf{g} \rangle$, where $n$ is a special RSA modulus of $2k$ bits, $\mathbf{g}$ is a quadratic residue modulo $n$, and $\mathbf{h} \in \mathbf{G}$.
- $\mathbb{G} = \langle g \rangle$, where $g$ is an element of prime order $q = \Theta(2^k)$, $h$ is an element in $\mathbb{G}$, and we will assume that DDH is hard in $\mathbb{G}$. (This group does not need to be a bilinear group.)

We define $\text{PedCom}(x_1, \ldots, x_n; r) = h^r \cdot \Pi_{i=1}^n g_i^{x_i}$. Sometimes for simplicity we do not explicitly include the randomness $r$ in the input to the commitment. Elements $h, \{g_i\}$ are assumed to be publicly known elements of the appropriate group. We also recall that the Dodis-Yampolskiy PRF, from Section 6.3.2, is defined as $f_{g,s}^{DY}(x) = g^{1/(s+x+1)}$.

Let the global parameters above be denoted $\zeta$.

We now describe the protocols.

**Bank Key Generation:** $\mathsf{BKeygen}(1^k, \zeta)$**:** The bank $\mathcal{B}$ generates a CL signature key pair $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$ for message space $M$ such that $\mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q \subseteq M$.

**User Key Generation:** $\mathsf{UKeygen}(1^k, \zeta)$**:** Each user $\mathcal{U}$ generates a unique key pair $(pk_{\mathcal{U}}, sk_{\mathcal{U}}) = (g^u, u)$ for a random $u \in \mathbb{Z}_q$. Recall that merchants are a subset of users.

**Withdrawal Protocol:** $\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_{\mathcal{U}}, 2^\ell), \mathcal{B}(pk_{\mathcal{U}}, sk_{\mathcal{B}}, 2^\ell))$**:** A user $\mathcal{U}$ interacts with the bank $\mathcal{B}$ as follows:

1. $\mathcal{U}$ identifies himself to the bank $\mathcal{B}$ by proving knowledge of $sk_{\mathcal{U}}$.
2. In this step, the user and bank contribute randomness to the wallet secret $s$; the user also selects a wallet secret $t$. This is done as follows: $\mathcal{U}$ selects random values $s', t \in \mathbb{Z}_q$ and sends a commitment $\mathbf{A}' = \text{PedCom}(sk_{\mathcal{U}}, s', t; r)$ to $\mathcal{B}$. $\mathcal{B}$ sends a random $r' \in \mathbb{Z}_q$. Then $\mathcal{U}$ sets $s = s' + r'$. $\mathcal{U}$ and $\mathcal{B}$ locally compute $\mathbf{A} = \mathbf{g}^{r'} \cdot \mathbf{A}' = \text{PedCom}(sk_{\mathcal{U}}, s' + r', t; r) = \text{PedCom}(sk_{\mathcal{U}}, s, t; r)$.
3. $\mathcal{U}$ and $\mathcal{B}$ run the CL protocol for obtaining $\mathcal{B}$'s signature on committed values contained in commitment $\mathbf{A}$. As a result, $\mathcal{U}$ obtains $\sigma_{\mathcal{B}}(sk_{\mathcal{U}}, s, t)$.
4. $\mathcal{U}$ saves the wallet $W = (sk_{\mathcal{U}}, s, t, \sigma_B(sk_{\mathcal{U}}, s, t), J)$, where $s, t$ are the wallet secrets, $\sigma_{\mathcal{B}}(sk_{\mathcal{U}}, s, t)$ is the bank's signature, and $J$ is an $\ell$-bit coin counter initialized to zero.
5. $\mathcal{B}$ records a debit of $2^\ell$ coins for account $pk_{\mathcal{U}}$.

**Outline of Spend Protocol:**

At the end of a successful Withdraw protocol, the user obtains a wallet of $2^\ell$ coins as $(sk_{\mathcal{U}}, s, t, \sigma, J)$, where $\sigma$ is the bank's signature on $(sk_{\mathcal{U}}, s, t)$ and $J$ is an $\ell$-bit counter.

1. User computes $R = H(pk_{\mathcal{M}} || info)$, where $info \in \{0,1\}^*$ is provided by the merchant.
2. User sends a coin serial number and double-spending equation:

$$S = F_{g,s}^{DY}(J) \quad , \quad T = pk_{\mathcal{U}} \cdot F_{g,t}^{DY}(J)^R.$$

3. User sends a ZKPOK $\Phi$ of $(J, sk_{\mathcal{U}}, s, t, \sigma)$ proving that:
   - $0 \leq J < 2^\ell$ (standard techniques [70, 55, 55, 37])
   - $S = F_{g,s}^{DY}(J)$ (see this chapter)
   - $T = pk_{\mathcal{U}} \cdot F_{g,t}^{DY}(J)^R$ (see this chapter)
   - $\text{VerifySig}(pk_{\mathcal{B}}, (sk_{\mathcal{U}}, s, t), \sigma) = \text{true}$ (CL signatures [53, 54])
4. If $\Phi$ verifies, $\mathcal{M}$ accepts the coin $(S, (R, T, \Phi))$ and later gives it to the bank during Deposit.
5. $\mathcal{U}$ updates his counter $J = J + 1$. When $J > 2^\ell - 1$, the wallet is empty.

Figure 6-1: A summary of the System One Spend protocol.

**Spend Protocol:** $\text{Spend}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, 2^\ell))$: Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ be a collision-resistant hash function. We highlight the key components of this protocol in Figure 6-1. In detail, a user $\mathcal{U}$ anonymously transfers a coin to merchant $\mathcal{M}$ as follows. (An optimized version appears in Appendix A.)

The protocol follows the outline in Figure 6-1 exactly. The ZKPOK in step 3 can be done as follows:

1. Let $\mathbf{A} = \text{PedCom}(J)$; prove that $\mathbf{A}$ is a commitment to an integer in the range $[0, 2^\ell - 1]$.
2. Let $\mathbf{B} = \text{PedCom}(u)$, $\mathbf{C} = \text{PedCom}(s)$, $\mathbf{D} = \text{PedCom}(t)$; prove knowledge of a CL signature from $\mathcal{B}$ on the openings of $\mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ in that order,
3. Prove $S = F_{g,s}^{DY}(J) = g^{1/(J+s+1)}$ and $T = pk_{\mathcal{U}} \cdot F_{g,t}^{DY}(J)^R = g^{u+R/(J+t+1)}$.
   More formally, this proof is the following proof of knowledge:

$$PK\{(\alpha, \beta, \delta, \gamma_1, \gamma_2, \gamma_3) : \mathbf{g} = (\mathbf{A} \cdot \mathbf{C})^\alpha \cdot \mathbf{h}^{\gamma_1} \wedge S = g^\alpha \wedge$$
$$\mathbf{g} = (\mathbf{A} \cdot \mathbf{D})^\beta \cdot \mathbf{h}^{\gamma_2} \ \wedge \ \mathbf{B} = \mathbf{g}^\delta \cdot \mathbf{h}^{\gamma_3} \ \wedge \ T = g^\delta \cdot (g^R)^\beta\}$$

Use the Fiat-Shamir heuristic to turn all the proofs above into one signature of knowledge, denoted $\Phi$, on the values $(S, T, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \mathbf{g}, \mathbf{h}, n, g, pk_{\mathcal{M}}, R, info)$.

**Deposit Protocol:** $\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, S, \pi, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$: A merchant $\mathcal{M}$ sends to bank $\mathcal{B}$ a coin $(S, \pi = (R, T, \Phi))$. If $\Phi$ verifies and $R$ is fresh (i.e., the pair $(S, R)$ is not already in the list $L$ of spent coins), then $\mathcal{B}$ accepts the coin for deposit, adds $(S, \pi)$ to the list $L$ of spent coins, and credits $pk_{\mathcal{M}}$'s account; otherwise, $\mathcal{B}$ sends $\mathcal{M}$ an error message.

Note that in this deposit protocol, $\mathcal{M}$ must convince $\mathcal{B}$ that it behaved honestly in accepting some coin $(S, \pi)$. As a result, our construction requires the Fiat-Shamir heuristic for turning a proof of knowledge into a signature. If $\mathcal{M}$ and $\mathcal{B}$ were the same entity, and the Withdraw and Spend protocols were interactive, then the bank $\mathcal{B}$ would not need to verify the validity of the coin that the merchant wishes to deposit, and as a result, we could dispense with the Fiat-Shamir heuristic and thus achieve balance and anonymity in the plain model (i.e., not just in the random oracle model).

We note that if $\mathcal{B}$ was satisfied with detecting/identifying double-spenders, without worrying about proving anything to a third party, it need only store $(S, R, T)$ in $L$ for each coin at a considerable storage savings.

**Detection of Double-Spending:** $\mathsf{DetectViolation}(\zeta, L)$: If two deposited coins in database $L$ have the same serial number $S_1 = S_2$, then double-spending is detected.

**Identification of a Double-Spender:** $\mathsf{IdentifyViolator}(\zeta, S, \pi_1, \pi_2)$: Suppose $(R_1, T_1) \in \pi_1$ and $(R_2, T_2) \in \pi_2$ are two entries in the bank's database $L$ of spent coins for the same serial number $S$. Then output the proof of guilt $\Pi = (\pi_1, \pi_2)$ and the identity of the user $pk = \left({T_2}^{R_1}/{T_1}^{R_2}\right)^{(R_1-R_2)^{-1}}$.

Let us explain why this produces the *public* key $pk_{\mathcal{U}}$ of the double-spender. Suppose coin $S$ belonged to some user with $pk_{\mathcal{U}} = g^u$, then each $T_i$ is of the form $g^{u+R_i\alpha}$ for the *same* values $u$ and $\alpha$. (Either this is true or an adversary has been successful in forging a coin, which we subsequently show happens with only negligible probability.) As the bank only accepts coins with fresh values of $R$ (i.e., $R_1 \neq R_2$), it allows the bank to compute:

$$\left(\frac{{T_2}^{R_1}}{{T_1}^{R_2}}\right)^{(R_1-R_2)^{-1}} = \left(\frac{g^{uR_1+R_1R_2\alpha}}{g^{uR_2+R_1R_2\alpha}}\right)^{(R_1-R_2)^{-1}} = g^{\frac{u(R_1-R_2)}{(R_1-R_2)}} = g^u = pk_{\mathcal{U}}.$$

**Verify Guilt of Double-Spending:** $\mathsf{VerifyViolation}(params, S, pk_{\mathcal{U}}, \Pi)$ : Parse $\Pi$ as $(\pi_1, \pi_2)$ and each $\pi_i$ as $(R_i, T_i, \Phi_i)$. Run $\mathsf{IdentifyViolator}(params,\ S,\ \pi_1, \pi_2)$ and compare the first part of its output to the public key $pk_{\mathcal{U}}$ given as input. Check that the values match. Next, verify each $\Phi_i$ with respect to $(S, R_i, T_i)$. If all checks pass, accept; otherwise, reject.

**Efficiency Discussion of System One.** The dominant computational cost in these protocols are the single and multi-base exponentiations. In a good implementation, a multi-base exponentiation is essentially as fast as an ordinary exponentiation. While we do not provide

the full details of the Withdraw protocol, it can easily be derived from the known protocols to obtain a CL signature on a committed signature [53, 48]. Depending on how the proof of knowledge protocol is implemented, Withdraw requires only three moves of communication.

The details of (an optimized version of) the Spend protocol are given in Appendix A. One can verify that a user must compute seven multi-base exponentiations to build the commitments and eleven more for the proof. The merchant and bank need to do eleven multi-base exponentiations to check that the coin is valid. The protocols require two rounds of communication between the user and the merchant and one round between the bank and the merchant.

**Theorem 6.1 (Security of System One).** *System One guarantees balance, identification of double-spenders, anonymity of users, and strong exculpability under the Strong RSA and y-DDHI assumptions in the random oracle model.*

*Proof.* **Balance.** Recall that the Withdraw and Spend protocols are executed sequentially, or with some bounded amount of concurrency that allows for rewinding.

*Part 1.* Let Adv be an adversary who executes $\ell$ Withdraw protocols with an extractor $\mathcal{E} = \mathcal{E}^{BB\text{-}\varepsilon(\mathsf{Adv})}_{\mathsf{Withdraw}_m, l_S}$ acting as the bank (with knowledge of $sk_B$). $\mathcal{E}$ behaves exactly as an honest bank would in all but step 3 of Withdraw. In this step, $\mathcal{E}$ must run its share of the CL protocol for obtaining a signature on the values in commitment $\mathbf{A} = g_0^r \cdot g_1^{sk_\mathcal{U}} \cdot g_2^s \cdot g_3^t = \mathrm{PedCom}(sk_\mathcal{U}, s, t; r)$. As part of this protocol, Adv is required to prove knowledge of the values $(sk_\mathcal{U}, s, t, r)$ in the standard way (see Section 6.3.1). Here, $\mathcal{E}$ will either rewind Adv (interactive proof) or use its control over the random oracle (non-interactive proof using Fiat-Shamir heuristic) to extract the values $(sk_\mathcal{U}, s, t, r)$.

From the value $s$, $\mathcal{E}$ can compute $w = (S_1, \ldots, S_n, sk_\mathcal{U}, s, t, r)$ such that $S_i = F^{DY}_{g,s}(i)$, for $i = 0$ to $n-1$. At the end of each Withdraw execution, $\mathcal{E}$ outputs $(state_\mathcal{E}, \mathbf{A}, w) \in l_S$, where the language $l_S$ is the set of triples $(\cdot, \mathbf{A}, w)$, where the first element may be anything, but the second two must conform to the specification for $w$ immediately above. Let $A_\ell = \{S_{i,j} | 1 \le i \le \ell, 0 \le j \le n-1\}$ be the list of serial numbers after $\ell$ executions of the Withdraw protocol.

*Part 2.* Due to the soundness of the underlying proof of knowledge protocols, we know that $A_\ell$ contains all *valid* serial numbers that Adv can produce, except with negligible probability. Thus, if Adv is to succeed at its game, it must convince an honest $\mathcal{B}$ to accept a serial number for which it cannot generate an honest proof of validity with some non-negligible probability.

Now suppose Adv convinces an honest $\mathcal{B}$ to accept the invalid coin $(S, \pi)$ during Deposit, where $S \notin A_\ell$ and $\pi = (R, T, \Phi)$. Then Adv must have concocted a *false* proof as part of the signature proof $\Phi$ that: (1) $\mathbf{A}$ opens to an integer in $[1, \ldots, 2^\ell]$ or (2) Adv knows a signature from $\mathcal{B}$ on the opening of $\mathbf{B}, \mathbf{C}, \mathbf{D}$ or (3) that $S$ (and $T$) are well formed as in $\Gamma$. Case (1) happens with negligible probability $\nu_1(k)$ under the Strong RSA assumption [70, 55, 55, 37]. Case (2) happens with negligible probability $\nu_2(k)$ under the Strong RSA assumption [53]. Case (3) happens with negligible probability $\nu_3(k)$ under the discrete logarithm assumption [169]. Thus, the Adv succeeds in this case with negligible probability $\nu_1(k) + \nu_2(k) + \nu_3(k)$. Thereby, Adv's total success probability in both parts is also negligible.

**Identification of double-spenders:** Let us first point out a case in which this property does not apply. Observe that whenever the bank issues a CL signature on wallet secrets $s, s'$ such that $|s - s'| < 2^\ell$, the recipients of these signatures can both generate valid spending proofs for some coin $S = f^{DY}_{g,s}(1) = f^{DY}_{g,s'}(s - s' + 1) = g^{1/(s+1)}$. Thus, there will be two valid entries in the bank's database as $(S, \pi_1)$ and $(S, \pi_2)$. This may look suspicious, but since double-spending has *not* occurred, this property does not apply. That said, we point out that the domain from which $s$ is sampled is large enough that this is not a problem. During step 2 of the Withdraw protocol, both the bank and user contribute to the randomness used to select $s \in \mathbb{Z}_q$. Thus, so long as one of them is honest, $s$ will be $2^\ell$-far from any other $s'$ with probability $2^{\ell+1}/q$.

We now return to our main proof. As defined in balance, let $\mathcal{E} = \mathcal{E}^{BB\text{-}\varepsilon(\mathsf{Adv})}_{\mathsf{Withdraw}_m, l_S}$ be the extractor that interacts with the adversary Adv during the Spend protocols to extract a set of valid serial numbers $A_\ell = \{S_{i,j} | 1 \le i \le \ell, 0 \le j \le n-1\}$. Now, suppose the honest merchant $\mathcal{M}$ (simulated by the bank) accepts two coins $(S, \pi_1)$ and $(S, \pi_2)$ for some $S \in A_\ell$. (We already saw in balance that the adversary cannot get an honest merchant to accept $S \notin A_\ell$ with non-negligible probability.)

Parse each $\pi_i$ as $(R_i, T_i, \Phi_i)$. Since an honest $\mathcal{M}$ chooses a fresh value *info*, where $R = H(id_{\mathcal{M}} || info)$, during each new Spend protocol, we know that $R_1 \neq R_2$ with high probability. We also know, due to soundness of ZK proof of validity, that each $T_i = pk_{\mathcal{U}} \cdot F^{DY}_{g,t}(J+1)^R$ for the same values of $pk_{\mathcal{U}}$, $J$ and $t$. Thus, the success of IdentifyViolator$(\zeta, S, \pi_1, \pi_2)$ in recovering $pk_{\mathcal{U}} = g^u$ follows directly from the correctness of the algorithm.

**Anonymity of users:** We capture anonymity by describing the simulator $S$ for our construction. Let the adversary Adv, representing a colluding bank and merchant(s), create and publish a public key $pk_{\mathcal{B}}$. Next, Adv may request the public key $pk_i$ of any user $i$. The adversary can engage in the Withdraw protocol with any user $i$ as many times as he likes. Finally, Adv will be asked to engage in a *legal* number of Spend protocols with some real user $j$ (with a real wallet $W_j$) or a simulator $\mathcal{S}$ (without $W_j$ or even knowledge of $j$).

The simulator $\mathcal{S} = \mathcal{S}^{IO\text{-}RO(\mathsf{Adv})}(params, auxsim, n)$ is given as input the global parameters *params*, some additional information *auxsim* possibly required by other simulators called as subroutines, and the total number of coins in a valid wallet $n$. This simulator $\mathcal{S}$ is given control of the random oracle as well as input-output access to the adversary Adv.

Our simulator $\mathcal{S}$ executes each new Spend request by Adv as follows:

1. $\mathcal{S}$ receives an optional transaction string $info \in \{0,1\}^*$.
2. $\mathcal{S}$ gathers the appropriate transaction information, such as *info*, $pk_{\mathcal{B}}$, current time, etc., and computes $R = H(id_{\mathcal{M}} || info)$.
3. Next $\mathcal{S}$ chooses random values $u, s, t \in \mathbb{Z}_q$ and a random $J \in [0, \ldots, n-1]$, and computes $S = f^{DY}_{g,s}(J+1)$ and $T = g^u \cdot F^{DY}_{g,t}(J+1)^R$.
4. $\mathcal{S}$ sends Adv the coin $(S, \pi)$, where $\pi = (R, T, \Phi)$, and $\Phi$ is the following simulated signature proof $\Phi$:
   (a) $\mathbf{A} = \mathrm{PedCom}(J+1)$ and a (real) proof that $\mathbf{A}$ is a commitment to an integer in the range $[1, \ldots, n]$,

(b) $\mathbf{B} = \text{PedCom}(u)$, $\mathbf{C} = \text{PedCom}(s)$, $\mathbf{D} = \text{PedCom}(t)$ and a *simulated* proof of knowledge of a CL signature from $\mathcal{B}$ on the openings of $\mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ in that order. ($\mathcal{S}$ invokes the appropriate CL simulator [53, 54] for this step, which requires control of the random oracle.)

(c) and a (real) proof $\Gamma$ that $S = g^{1/(J+s+1)}$ and $T = g^{u+R/(J+t+1)}$.

Observe above that all the difficulty of $\mathcal{S}$'s job is handled by the simulator for a proof of knowledge of a CL signature, which uses standard techniques.

We now explain why the output of $\mathcal{S}$ is computationally indistinguishable from the output of a real user. First, during the Withdraw protocol, Adv does not learn anything about the set of secrets $(u', s', t')$ that it signs due to the security of the CL signatures. In fact, these values can be information-theoretically hidden from Adv by requesting a signature on $(u', s', t', r')$ for a random $r'$ that is otherwise discarded [53, 54].

Thus, the values $s$ and $t$ chosen by $\mathcal{S}$ are indistinguishable from those chosen by real users. Recall that a coin consists of the tuple $(S, (R, T, \Phi))$, where $R$ is (indirectly) chosen by Adv. Due to the security of the DY PRF [95], the coin parts $S = F^{DY}_{g,s}(J+1)$ and $T = g^u \cdot F^{DY}_{g,t}(J+1)^R$ are computationally indistinguishable from random elements in $\mathbb{G}_2$, and therefore, equally likely to have been generated by any user with $pk_i = g^{u_i}$ and with any coin counter value $J \in [0, \ldots, n-1]$. Observe that $T$ looks random due to the fact that $F^{DY}_{g,t}(J+1)$ is computed in a cyclic group of prime order, and is thus a random generator in $\mathbb{G}$ raised to an arbitrary, non-zero power. Finally, recall that our simulated $\Phi$ consisted of a group of (perfectly hiding) Pedersen commitments, two true proofs, and one simulated proof. The simulated proof is the only difference between $\mathcal{S}$'s $\Phi$ and that of a real user. However, we ran the CL signature proof simulator to generate the simulated proof, and thus, by the security of the protocols associated with CL signatures, Adv distinguishes between Game R (with a real user) and Game I (with $\mathcal{S}$) with only negligible probability based on Strong RSA [53].

**Strong Exculpability:** In our construction, it is easy to see that corresponding to a wallet $W$, even one obtained as a result of interacting with a malicious bank $\mathcal{B}$, there are exactly $n$ serial numbers that an honest user $\mathcal{U}$ can produce. Each serial number is a deterministic function $F^{DY}_{(g,s)}(J)$ of the seed $s$ and the counter $J \in [0, n-1]$. Since a user must provide a ZK proof of validity for each coin, to spend $n+1$ or more times from the same wallet requires that two coins have the same serial number by the pigeonhole principle.

In our construction, we did not define a VerifyOwnership algorithm, because it is an algorithm associated with tracing coins and System One does not support tracing. Let us now define it as the algorithm that rejects on all inputs. Thus, all parts of the exculpability definition relating to VerifyOwnership are trivially satisfied.

It remains to argue that Adv cannot, after playing the strong exculpability game, output values $(S, \Pi)$ such that VerifyViolation$(params, S, \Pi, pk_{\mathcal{U}}, n)$ accepts and yet this user did *not* double-spend this particular coin $S$. This is also fairly trivial. Recall that in System One the proof of guilt is $\Pi = (\pi_1, \pi_2)$ such that the coins $(S, \pi_1)$ and $(S, \pi_2)$ were accepted by honest merchants. Part of the proof of validity $\pi$ involves proving knowledge of the user's

secret key $sk_{\mathcal{U}}$ (see step 3(c) of Spend). Recall that even by double-spending, a user's public key is the only information leaked. Due to the soundness of our ZK proofs, it must have been the user, with knowledge of $sk_{\mathcal{U}}$, that spent coin $S$ twice. □

### 6.4.1 System Zero: A Simpler E-Cash System

Let us step back for a moment. Not all e-cash applications require that double-spenders be identified. For example, when the bank and the merchant are the same entity (e.g., an online service), it might be enough to detect and *block* double-spending.

Consider the following reduced system: a wallet simply consist of a PRF seed $s$, the bank's signature on $s$, and a counter $J$: $(s, \sigma_{\mathcal{B}}(s), J)$. To spend a coin, the user produces the serial number $f^{DY}_{g,s}(J)$ along with a proof of its validity.

The benefit for removing the identification requirement is two-fold: our protocols become more efficient (roughly 25% computation savings) and users are not required to generate and register a public key. The compactness of this new wallet remains superior to all existing online e-cash systems, e.g., [74, 75, 76].

## 6.5 System Two: Compact E-Cash with Tracing

We now extend System One to allow coin tracing. Let us begin by sketching how this can be accomplished. Suppose for the moment that the IdentifyViolator algorithm recovered $sk_{\mathcal{U}}$ rather than $pk_{\mathcal{U}}$ for a double-spender. We change the withdrawal protocol so that the user $\mathcal{U}$ must also provide the bank $\mathcal{B}$ with a verifiable encryption of her wallet secret $s$ (used to generate the coin serial numbers) under *her own public key* $pk_{\mathcal{U}}$. This way, if $\mathcal{U}$ double-spends, $\mathcal{B}$ can recover $sk_{\mathcal{U}}$, decrypt to obtain her secret $s$, and compute the serial numbers $S_i = f^{DY}_{g,s}(i)$, for $i = 0$ to $2^\ell - 1$, of all coins belonging to $\mathcal{U}$.

(Notice that recovering $sk_{\mathcal{U}}$ would actually allow the bank to decrypt *all* ciphertexts corresponding to different wallets for user $\mathcal{U}$, and thus, trace *all* coins withdrawn from account $pk_{\mathcal{U}}$. We will return to this point later.)

Now that we understand the high-level construction, we will go into more details as to how we realize it, because some technical issues arise. Recall that a user's keys in the previous system were of the form $pk_{\mathcal{U}} = g^u$ and $sk_{\mathcal{U}} = u$, and that the IdentifyViolator algorithm recovered $g^u$ when a user double-spent. Suppose there was a semantically-secure cryptosystem where the value $g^u$ was sufficient to decrypt and the new public key was a one-way function of $g^u$. Given such a cryptosystem, we can use the verifiable encryption techniques of Camenisch and Damgård, as described in Section 6.3.5, which can be applied to any semantically-secure encryption scheme. This will allow user $\mathcal{U}$ to leave a verifiable encryption of her wallet secret $s$ under her own $pk_{\mathcal{U}}$ with the bank $\mathcal{B}$ during the withdrawal protocol. No one but $\mathcal{U}$ knows the corresponding $sk_{\mathcal{U}}$. However, by using the same coin structure as System One, double-spending allows $\mathcal{B}$ to recover the value $g^u$. Thus, $\mathcal{B}$ can decrypt the ciphertexts and trace the cheating user's coins. All we need is a cryptosystem with keypairs of this new form.

Luckily, the bilinear El Gamal scheme (see Section 6.3.6) does exactly this. It supports public keys of the form $pk_{\mathcal{U}} = \mathsf{e}(g_1, g_2)^u$, for $u \in \mathbb{Z}_q$, where knowing $g_1^u$ is sufficient for decryption. This would be the entire solution, except one technical difficulty arises.

**The Problem with Plugging in a Bilinear Group.** The Dodis-Yampolskiy PRF [95], used in the double-spending equation, is only known to be a PRF in groups where DDH is hard. However, to recover the decryption key $g_1^u$ for the bilinear El Gamal cryptosystem, we need to set the double-spending equation in the group $\mathbb{G}_1$, where there exists a mapping $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, and thus DDH may be easy. Obviously, this is a problem, because the anonymity of the coins relies on the pseudorandomness of the PRF.

We propose two different solutions under two different assumptions to overcome this technical hurdle. Let us now describe these two solutions, and then provide a detailed construction using the second one.

### 6.5.1 Solution #1: Use DY PRF, make XDH Assumption.

As described in Chapters 2 and 3, the XDH Assumption posits that DDH is hard in $\mathbb{G}_1$ given the existence of $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. If XDH holds, then our construction above just works. We may continue to use the DY PRF [95] to create our coins, as we did in System One. Where we used $\mathbb{G}$, a generic group of prime order where DDH was hard, in System One, we can now replace it with $\mathbb{G}_T$, the range of the bilinear mapping, for all items *except* the double-spending equation $T_i = g_1^u \cdot f_{g_1,t}^{DY}(i)^R$ which is now set in $\mathbb{G}_1$. The user's key pairs are now of the form $pk_{\mathcal{U}} = \mathsf{e}(g_1, g_2)^u$ and $sk_{\mathcal{U}} = u$ for the bilinear El Gamal cryptosystem (Chapter 5), where $g_1^u$ is sufficient for decryption.

Due to the obvious similarities with System One, the above comments are sufficient to describe the entire system.

### 6.5.2 A New PRF for DDH-Easy Groups

Suppose that DDH is not hard in $\mathbb{G}_1$. Is there another PRF that we can employ instead? We are aware of only one candidate PRF for a DDH-easy group due to Dodis [89]. Unfortunately, we could not adapt his construction for use in System Two without performing expensive proofs of knowledge. For efficiency reasons, we instead propose a new PRF construction [49] specifically to solve our problem. In Section 6.5.3, we give a second solution using our PRF.

Let us begin by recalling the definition of a *sum-free* encoding as given by Dodis [89].

**Definition 6.2 (Sum-Free).** *We say that an encoding $V : \{0,1\}^n \rightarrow \{0,1\}^m$ is* sum-free *if for all distinct elements $a, b, c, d$ in the set of encodings $\{V(s)\}_{s \in \{0,1\}^n}$, it holds that $a + b \neq c + d$, where $a, b, c, d$ are viewed as $m$-bit 0/1-vectors and $+$ is bitwise addition over the integers.*

Recall that bitwise addition and subtraction does not allow for carry bits; e.g., the bitwise addition of binary strings 011 and 001 over the integers is 012.

Dodis [89] proved that if $V$ is any sum-free encoding, and $\langle g \rangle$ is a group of order $q$, then $f_{g,(\cdot)}^V$, defined as follows, is a PRF: the seed for this PRF consists of values $t_i \in \mathbb{Z}_q$, for

$0 \leq i \leq 3\ell$; let $\vec{t} = (t_0, \ldots, t_{3\ell})$; the function $f^V_{g,\vec{t}}$ is defined as

$$f^V_{g,\vec{t}}(x) = g^{t_0 \prod_{V(x)_i=1} t_i}.$$

Dodis' proof holds under the Sum-Free DDH assumption (also introduced by Dodis [89]). As he observes, it seems reasonable to make such an assumption even of groups where DDH is easy, because the sum-free property precludes trivial use of the bilinear map to solve DDH.

For our purposes, we need the encoding $V$ to have nice algebraic properties. We define an encoding $V : \{0,1\}^\ell \mapsto \{0,1\}^{3\ell}$ as $V(x) = x \circ x^2$, where $\circ$ denotes concatenation, and multiplication is over the integers.

**Lemma 6.3.** *The encoding $V(x) = x \circ x^2$ described above is sum-free.*

*Proof.* Observe that $V : \{0,1\}^\ell \rightarrow \{0,1\}^{3\ell}$. Suppose that $A, B, C, D$ are elements of $\{V(s)\}_{s \in \{0,1\}^\ell}$. We can parse each element as $A = a \circ a^2$, $B = b \circ b^2$, etc.

We show that if $A + B = C + D$, then without loss of generality $a = c$ and $b = d$, implying that $A, B, C, D$ are not distinct. If $A + B = C + D$, it follows that

$$a + b = c + d \tag{6.1}$$

$$a^2 + b^2 = c^2 + d^2 \tag{6.2}$$

For the moment, let $+$ and $-$ be addition and subtraction, respectively, over the integers. We can rearrange equation 6.2 as $a^2 - d^2 = c^2 - b^2$. By factoring this new equation, we have $(a + d)(a - d) = (c + b)(c - b)$. We can rearrange equation 6.1 as $a - d = c - b$. We can then substitute this into the previous equation to obtain $(a + d)(a - d) = (c + b)(a - d)$. By canceling $(a - d)$ from both sides, we arrive at $a + d = c + b$. Subtracting equation 6.1, we have $d - b = b - d$, which implies $2d = 2b$, and $d = b$ directly follows. The same logic shows that $c = a$.

Lastly, it suffices to observe that this analysis is the same when $+$ and $-$ are bitwise addition and subtraction, respectively. □

### 6.5.3 Solution #2: Use CHL PRF, make Sum-Free DDH Assumption.

Solution #1 has the benefit of being very simple and efficient, however, it cannot be implemented using supersingular curves. Using the PRF just presented, however, we devise a second solution that is conjectured to work in either ordinary or supersingular curves. Instead of using the DY PRF everywhere, we instead use the new PRF in only the double-spending equation $T$.

We now present the details of System Two implemented according to this strategy.

**Global parameters for System Two.** Let $1^k$ be the security parameter and let $2^\ell$ be the public wallet size. This scheme works most efficiently by having four different groups:

- $\mathbf{G} = \langle \mathbf{g} \rangle$, where $n$ is a special RSA modulus of $2k$ bits, $\mathbf{g}$ is a quadratic residue modulo $n$, and $\mathbf{h} \in \mathbf{G}$.

- $\mathbb{G}_1 = \langle g_1 \rangle = \langle h_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle = \langle h_2 \rangle$, and $\mathbb{G}_T$ of prime order $q$ such that exists a bilinear mapping $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

Let the global parameters above be denoted $\zeta$.

Our second system supports all algorithms mentioned in Section 6.2, specifically including the tracing algorithms: (BKeygen, UKeygen, Withdraw, Spend, Deposit, DetectViolation, IdentifyViolator, VerifyViolation, Trace, VerifyOwnership). We assume a standard signature scheme (SKeygen, Sign, SVerify).

**Bank Key Generation:** $\mathsf{BKeygen}(1^k, \zeta)$**:** The bank $\mathcal{B}$ generates a CL signature keypair $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$, as before, in the group $\mathbf{G}$ with composite modulus $n$, although they could moved to the bilinear groups [54].

**User Key Generation:** $\mathsf{UKeygen}(1^k, \zeta)$**:** Each user $\mathcal{U}$ runs the signature key generation algorithm $\mathsf{SKeygen}(1^k, \zeta) \rightarrow (svk_{\mathcal{U}}, ssk_{\mathcal{U}})$ and the bilinear El Gamal key generation algorithm to obtain $(ek_{\mathcal{U}}, dk_{\mathcal{U}}) = (\mathsf{e}(g_1^u, g_2), g_1^u)$, and outputs $pk_{\mathcal{U}} = (ek_{\mathcal{U}}, svk_{\mathcal{U}})$ and $sk_{\mathcal{U}} = (dk_{\mathcal{U}}, ssk_{\mathcal{U}})$.

**Withdrawal Protocol:** $\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_{\mathcal{U}}, 2^\ell), \mathcal{B}(pk_{\mathcal{U}}, sk_{\mathcal{B}}, 2^\ell))$**:** A user $\mathcal{U}$ interacts with the bank $\mathcal{B}$ as follows:

1. $\mathcal{U}$ identifies himself to the bank $\mathcal{B}$ by proving knowledge of $sk_{\mathcal{U}} = (dk_{\mathcal{U}}, ssk_{\mathcal{U}})$.
2. As in System One, in this step, $\mathcal{U}$ and $\mathcal{B}$ contribute randomness to the wallet secret $s$, and the user selects wallet secrets $\vec{t} = (t_0, \ldots, t_{3\ell})$, where $t_i \in \mathbb{Z}_q$ for all $i$. As before, this is done as follows: $\mathcal{U}$ chooses random values $s'$ and $\vec{t}$, and sends the commitment $\mathbf{A}' = \mathrm{PedCom}(u, s', \vec{t})$ to $\mathcal{B}$, obtains a random $r'$, sets $s = s' + r'$, and then both $\mathcal{U}$ and $\mathcal{B}$ locally set $\mathbf{A} = \mathbf{g}^{r'} \cdot \mathbf{A}' = \mathrm{PedCom}(u, s, \vec{t})$.
3. $\mathcal{U}$ forms a verifiable encryption $Q$ of the value $s$ under his own key $ek_{\mathcal{U}} = \mathsf{e}(g_1^u, g_2)$. (This encryption can be proved correct relative to commitment $\mathbf{A}$.) $Q$ is signed by $\mathcal{U}$. $\mathcal{B}$ verifies the correctness of $Q$ and the signature $\sigma$ on $Q$. $\mathcal{U}$ obtains a CL signature from $\mathcal{B}$ on the values committed in $\mathbf{A}$ via the protocol for getting a signature on a set of committed values.
4. $\mathcal{B}$ debits $2^\ell$ from account $pk_{\mathcal{U}}$, records the entry $(pk_{\mathcal{U}}, Q, \sigma)$ in his database $D$, and issues $\mathcal{U}$ a CL signature on $Y$.
5. $\mathcal{U}$ saves the wallet $W = (sk_{\mathcal{U}}, s, \vec{t}, \sigma_B(u, s, \vec{t}), J)$, where $J$ is an $\ell$-bit counter set to zero.

**Spend Protocol:** $\mathsf{Spend}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, 2^\ell))$**:** The only change from System One is in the calculation of the security tag $T$ and the subsequent proof $\Phi$. Assume $info \in \{0,1\}^*$ and $R \in \mathbb{Z}_q^*$ are obtained as before.

Recall the details of the serial number of the coin $S = f_{g_T,s}^{DY}(J) = g_T^{1/(J+s+1)}$, the security tag $T = dk_{\mathcal{U}} \cdot f_{g_1,\vec{t}}^{V}(J)^R = g_1^{u + Rt_0 \prod_{\{i:V(J)_i=1\}} t_i}$. The signature proof $\Phi$ of their validity consists of:

**Outline of Spend Protocol:**

At the end of a successful Withdraw protocol, the user obtains a wallet of $2^\ell$ coins as $(sk_{\mathcal{U}}, s, t_0, \ldots, t_{3\ell}, \sigma, J)$, where $\sigma$ is the bank's signature on $(sk_{\mathcal{U}}, s, t_0, \ldots, t_{3\ell})$ and $J$ is an $\ell$-bit counter.

1. User computes $R = H(pk_{\mathcal{M}} || info)$, where $info \in \{0,1\}^*$ is provided by the merchant.
2. User sends a coin serial number and double-spending equation:
$$S = F^{DY}_{g_T,s}(J) \quad , \quad T = pk_{\mathcal{U}} \cdot F^{DY}_{g_1,\vec{t}}(J)^R.$$
3. User sends a ZKPOK $\Phi$ of $(J, sk_{\mathcal{U}}, s, t_0, \ldots, t_{3\ell}, \sigma)$ proving that:
   - $0 \leq J < 2^\ell$ (standard techniques [70, 55, 55, 37])
   - $S = F^{DY}_{g_T,s}(J)$ (see this chapter)
   - $T = pk_{\mathcal{U}} \cdot F^{V}_{g_1,\vec{t}}(J)^R$ (see this chapter)
   - VerifySig$(pk_{\mathcal{B}}, (sk_{\mathcal{U}}, s, t_0, \ldots, t_{3\ell}), \sigma)$ =true (CL signatures [53, 54])
4. If $\Phi$ verifies, $\mathcal{M}$ accepts the coin $(S, (R, T, \Phi))$ and later gives it to the bank during Deposit.
5. $\mathcal{U}$ updates his counter $J = J + 1$. When $J > 2^\ell - 1$, the wallet is empty.

Figure 6-2: A summary of the System Two (Solution #2) Spend protocol.

(a) $\mathbf{A}_0 = \text{PedCom}(J)$ and a proof that $\mathbf{A}$ is a commitment to an integer in the range $[0, 2^\ell - 1]$; a commitment $\mathbf{A}_1 = \text{PedCom}(J^2)$ and a proof that it is a commitment to the square of the opening of $\mathbf{A}_0$; and finally, a commitment $\mathbf{A}_2 = \text{PedCom}(V(J)) = \text{PedCom}(J \circ J^2)$ and a proof that it was formed correctly.

(b) $\mathbf{B}_i = \text{PedCom}(V(J)_i)$ for $i = 1$ to $3\ell$ (commitments to the bits of $V(J)$) and proof that each $\mathbf{B}_i$ opens to either 0 or 1; that is, $PK\{(\gamma_1, \gamma_2) : \mathbf{B}_i/\mathbf{g} = \mathbf{h}^{\gamma_1} \vee \mathbf{B}_i = \mathbf{h}^{\gamma_2}\}$,

(c) proof that $\mathbf{A}_2$ and $\{\mathbf{B_i}\}$ are consistent; $PK\{(\gamma) : \mathbf{A}_2/(\mathbf{g} \cdot \prod_{i=1}^{3\ell} \mathbf{B}_i^{2^{i-1}}) = \mathbf{h}^\gamma\}$,

(d) commitments to $\mathcal{U}$'s secret key $u$, and wallet secrets $s$ and $\vec{t}$: $\mathbf{C} = \text{PedCom}(u)$, $\mathbf{D} = \text{PedCom}(s)$, $\mathbf{E}_i = \text{PedCom}(t_i)$ for $i = 0$ to $3\ell$, and proof of knowledge of a CL signature from $\mathcal{B}$ on the openings of $\mathbf{C}$, $\mathbf{D}$, and all $\mathbf{E}_i$'s in that order,

(e) the following commitments that will help in proving that $T$ was formed correctly: $\mathbf{F}_0 = \mathbf{E}_0$, $\mathbf{F}_i = \text{PedCom}(\prod_{\{j \leq i : V(J)_j = 1\}} t_j)$ for $i = 1$ to $3\ell$,

(f) a proof that $S = f^{DY}_{g_T,s}(J)$ and $T = g_1^u \cdot (f^V_{g_1,\vec{t}}(J))^R$. (We provide more details of this step in Appendix A.2.) Proving the statement about $S$ is done as in System One. Proving the statement about $T$ can be done as follows: Prove, for every $1 \leq i \leq 3\ell$ that $\mathbf{F}_i$ was formed correctly, corresponding to the committed value $t_i$

and the value (bit) contained in the commitment $\mathbf{B}_i$. That is to say:

$$PK\{(\alpha,\beta,\delta) \ : \ \mathbf{F}_i = \text{PedCom}(\alpha) \ \wedge \ \mathbf{F}_{i-1} = \text{PedCom}(\beta) \ \wedge \\ \mathbf{E}_i = \text{PedCom}(\delta) \ \wedge \ \Big(\big(\mathbf{B}_i = \text{PedCom}(0) \ \wedge \ \alpha = \beta\big) \ \vee \\ \big(\mathbf{B}_i = \text{PedCom}(1) \ \wedge \ \alpha = \beta \cdot \delta\big)\Big)\}$$

Note that, if all $\mathbf{F}_i$'s are formed correctly, then $\mathbf{F}_{3\ell}$ is a commitment to the discrete logarithm of the value $f^V_{g_1,\vec{t}}(J)$ which is $t_0 \cdot \prod_{i=1}^{3\ell} t_i^{(V(J))_i}$. So we can prove the validity of tag $T$ as follows:

$$PK\{(\alpha,\beta) \ : \ T = g_1^{\alpha+\beta R} \wedge \mathbf{C} = \text{PedCom}(\alpha) \wedge \mathbf{F}_{3\ell} = \text{PedCom}(\beta)\}$$

As in System One, using the Fiat-Shamir heuristic we turn these multiple proofs of knowledge into one signature, secure in the random-oracle model.

**Deposit Protocol:** $\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, S, \pi, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$**:** Same as System One: $\mathcal{M}$ sends $\mathcal{B}$ the coin $(S, \pi = (R, T, \Phi))$. During deposit, the bank may store only $(S, R, T)$ in database $L$ to obtain all desired functionality – except the ability to convince a third party of anything, such as a double-spender's identity or which coins belong to him.

**Detection of Double-Spending:** $\mathsf{DetectViolation}(\zeta, L)$**:** Same as System One: if two deposited coins in database $L$ have the same serial number $S_1 = S_2$, then double-spending is detected.

**Identification of a Double-Spender:** $\mathsf{IdentifyViolator}(\zeta, S, \pi_1, \pi_2)$**:** Similar to System One: compute the identity of the user $g_1^u = \left(T_2{}^{R_1}/T_1{}^{R_2}\right)^{(R_1-R_2)^{-1}}$, then output the proof of guilt $(g_1^u, \pi_1, \pi_2)$. Recall that $g_1^u$ is secret information about the user with public key $\mathsf{e}(g_1, g_2)^u$.

Alternatively, $\Pi$ could just be $g_1^u$ since this secret information would not be leaked unless the user deviated from the described protocols. However, since a user could have her secret key stolen, for policy reasons we keep the proof of guilt as the transaction logs in $L$ allowing to compute $g_1^u$.

**Verify Guilt of Double-Spending:** $\mathsf{VerifyViolation}(\zeta, S, pk_{\mathcal{U}}, \Pi)$**:** Same as System One: run the $\mathsf{IdentifyViolator}$ algorithm on the two coins contained in $\Pi$, accept if the output corresponds to the user with key $pk_{\mathcal{U}}$.

**Trace Transactions:** $\mathsf{Trace}(\zeta, S, pk_{\mathcal{U}}, \Pi, D, 2^\ell)$**:** Parse $\Pi$ as $(dk, \pi_1, \pi_2)$ and $pk_{\mathcal{U}}$ as $(ek_{\mathcal{U}}, svk_{\mathcal{U}})$. The bank checks that $\mathsf{e}(dk, g_2) = ek_{\mathcal{U}}$; if not, it aborts. Otherwise the bank searches its database $D$, generated during the withdrawal protocol, for verifiable encryptions tagged with the public key $pk_{\mathcal{U}}$. For each matching entry $(pk_{\mathcal{U}}, Q, \sigma)$, $\mathcal{B}$ does the following: (1) runs the Camenisch-Damgård decryption algorithm on $Q$ with $dk$ to recover the value $s$; and (2) then for $i = 0$ to $2^\ell - 1$, outputs a serial number $S_i = f^{DY}_{g_T,s}(i)$ and a proof of ownership $\Gamma_i = (Q, \sigma, dk, i)$.

**Verify Ownership of Coin:** $\mathsf{VerifyOwnership}(\zeta, S, \Gamma, pk_{\mathcal{U}}, 2^{\ell})$: Parse the proof $\Gamma$ as $(Q, \sigma, dk, i)$. Check that $\sigma$ is $pk_{\mathcal{U}}$'s signature on $Q$ and that $i$ is in the range $[0, 2^{\ell}-1]$. Next, verify that $dk$ is $pk_{\mathcal{U}}$'s decryption key by checking that $\mathsf{e}(dk, g_2) = ek_{\mathcal{U}}$. Finally, run the verifiable decryption algorithm on $Q$ with $dk$ to recover $s'$ and verify that $S = f^{DY}_{g_T,s'}(i)$. If all checks pass, the algorithm accepts, otherwise, it rejects.

**Efficiency Discussion of System Two.** In Withdraw, the number of communication rounds does not change from System One, but one of the multi-base exponentiations will involve $3\ell$ bases and hence its computation will take longer. Let us discuss the computational load of the verifiable encryption. For a cheating probability of at most $2^{-k}$, the user must additionally compute $k$ exponentiations and $2k$ encryptions with the bilinear El Gamal scheme. To verify, the bank also must perform $k$ exponentiations but only $k$ encryptions. Upon recovery of the double-spender's secret key, the bank needs to perform at most $k$ decryptions and $k$ exponentiations. Furthermore, the bank needs to compute all the $2^{\ell}$ serial numbers each of which takes one exponentiation.

Details of the proof in step (f) of the Spend protocol are in Appendix A.2. In Spend, the user must compute a total of $7 + 9\ell$ and $17 + 21\ell$ multi-base exponentiations for the commitments and the signature proof, respectively. The merchant and the bank also need to perform $17 + 21\ell$ multi-base exponentiations. For each of these, there is one multi-base exponentiation with $3\ell$ exponents while all the others involve two to four bases.

**Theorem 6.4 (Security of System Two).** *System Two guarantees balance, identification of double-spenders, tracing of double-spenders, anonymity of users, and weak and strong exculpability under the Strong RSA, y-DDHI, and either the XDH (using Solution #1) or Sum-Free DDH (using Solution #2) assumptions in the random oracle model.*

*Proof sketch.* **Balance:** *Part 1.* The extractor $\mathcal{E} = \mathcal{E}^{BB\text{-}\varepsilon(\mathsf{Adv})}_{\mathsf{Withdraw}_m, l_S}$ proceeds exactly as in System One, the only difference being that this time he will recover more messages, e.g., $(u, s, t_0, \ldots, t_{3\ell})$.

*Part 2.* As before, we know that Adv cannot produce a truly valid serial number $S$ (i.e., one for which it need not fake a proof of validity) which is not in the set $A_f$ output by the extractor except with negligible probability. So it remains to analyze Adv success in faking proofs of validity for $S \notin A_{\ell}$. The analysis is similar to before, except that the signature proof $\pi = (R, T, \Phi)$ is more complicated. If Adv succeeds in convincing $\mathcal{B}$ to accept $(S, \pi)$, then he must have concocted a *false* proof as part of $\Phi$ that: (1) $\mathbf{A}$ opens to an integer in $[1, \ldots, 2^{\ell}]$ or (2) some $\mathbf{B}_i$ opens to either 0 or 1 or (3) $\mathbf{A}$ and $\{\mathbf{B}_i\}$ are consistent or (4) Adv knows a signature from $\mathcal{B}$ on the opening of $\mathbf{C}, \mathbf{D}, \{\mathbf{E}_i\}$ or (5) that $S$ (and $T$) are well formed as in $\Gamma$. Case (1) happens with negligible probability $\nu_1(k)$ under the Strong RSA assumption [70, 55, 55, 37]. Cases (2), (3), and (5) happen with negligible probability $\nu_2(k)$ under the discrete logarithm assumption [169]. Case (4) happens with negligible probability $\nu_3(k)$ under the Strong RSA assumption [53]. Thus, Adv's total success probability is also negligible.

**Identification of double-spenders:** The identification of double-spenders property of System Two follows directly from the proof of this property for System One.

**Tracing of double-spenders:** First, the adversary has only negligible chance of winning by the existence of a serial number $S' \in A_\ell$ not output by Trace. This is because, in each Withdraw protocol, the secret $s$ that $\mathcal{E}$ extracts (to generate $S_i = f^{DY}_{g_T,s}(i)$, for $0 \leq i < n$, for $A_\ell$) is also verifiably encrypted under the user's public key $ek_{\mathcal{U}} = \mathsf{e}(g_1, g_2)^u$ and given to the bank. Due to the completeness of the bilinear El Gamal scheme [8] and the soundness of the verifiable encryption protocol of Camenisch and Damgård [47], we know that the bank can decrypt to obtain $s$ once part of the user's secret key $g_1^u$ is known. For System Two, it is easy to see that $g_1^u$ can be recovered from double-spent coins given the identification of double-spenders property above. (Recall that in System Two, one computes the user's secret information $g_1^u$ on the way to finding part of the user's public key $ek_{\mathcal{U}}$.) With high probability, the Trace algorithm will obtain $s$, for each wallet, and output all serial numbers in $A_\ell$.

Secondly, the adversary has only a negligible chance of winning by the existence of a serial number $S_j$ and corresponding proof $\Gamma_j$ output by Trace($params, S, pk_{\mathcal{U}}, \Pi, D, n$) such that the algorithm VerifyOwnership($params, S_j, \Gamma_j, pk_{\mathcal{U}}, n$) does not accept. This is because verifying the ownership of a coin $S_j$ in our construction is tantamount to re-running the (deterministic) Trace algorithm and checking its output.

**Anonymity of users:** The proof of anonymity for System Two follows that of System One, with the three exceptions being that:

1. One must argue that Adv does not learn any information about the users during the verifiable encryption protocol added to Withdraw that will later help it distinguish. This follows from the semantic-security of the bilinear El Gamal scheme [8] and the verifiable encryption technique of Camenisch and Damgård [47] which rely on the DBDH (which is later subsumed by the $y$-DDHI assumption in the theorem statement, as we discussed in Section 5.3.3) and the Strong RSA assumptions, respectively.
2. The assumption under which we claim that double-spending equation $T$ is indistinguishable from a random element in $\mathbb{G}_1$ due to the pseudorandomness of our new PRF $f^V_{(g_1,\cdot)}(\cdot)$ changes from $y$-DDHI to Sum-Free DDH.
3. And finally, although the simulated signature proof $\Phi$ is more complicated, it remains that the only simulated part of $\Phi$ is performed for $\mathcal{S}$ (with IO, random oracle access) by the CL protocol simulator under the Strong RSA [53].

**Strong Exculpability:** In this system, the proof of guilt corresponding to VerifyViolation is identical to System One: two valid coins $(S, \pi_1)$ and $(S, \pi_2)$ such that IdentifyViolator($params$, $S, \pi_1, \pi_2$) outputs $g_1^{sk_{\mathcal{U}}}$ associated with the user. The only difference here is that $g_1^{sk_{\mathcal{U}}}$ is part of the user's secret information as opposed to being her public key as in System One. Recall that knowledge of the value $sk_{\mathcal{U}}$ is required to create a coin proof of validity $\pi$; it is not enough to know $g_1^{sk_{\mathcal{U}}}$. This comes from step (d) of the Spend protocol, where the user is

required to prove knowledge of the opening of commitment $\mathbf{C} = \mathrm{PedCom}(sk_{\mathcal{U}})$ as part of the CL protocol. Thus, Adv cannot output two such proofs for coin $S$ unless he asks the user to make them.

Unlike System One, this system has a non-trivial VerifyOwnership algorithm defined. First, we claim that no adversary can produce a serial number $S$ that does not belong to the set of serial numbers of $\mathcal{U}$ known to Adv, and a proof $\Gamma$ such that VerifyOwnership($params$, $S$, $\Gamma$, $pk_{\mathcal{U}}$, $n$) accepts with non-negligible probability. Recall that a proof for VerifyOwnership in System Two consists of $(Q, \sigma, dk, i)$, where $Q$ is a verifiable encryption of a wallet secret $s$, $\sigma$ is the user's signature on $Q$, $dk$ is the user's decryption key, and $i$ is an integer in $[0, \ldots, 2^{\ell} - 1]$. There is only one $dk$ corresponding to $ek_{\mathcal{U}} \in pk_{\mathcal{U}}$ (which is easy to test as $\mathsf{e}(dk, g_2) = ek_{\mathcal{U}}$). Decryption of $Q$, to say $s$, given $dk$ is deterministic, as is checking that $S = f^{DY}_{\mathsf{e}(g_1,g_2),s}(i)$ for $i$ in the right range. Thus, to forge a proof of ownership against $pk_{\mathcal{U}}$ for a serial number $S$, the adversary needs to produce a seed $s$ and number $J$ such that $S = F^{DY}_{\mathsf{e}(g_1,g_2),s}(J+1)$, and a ciphertext $Q$ that is an encryption of $s$ signed by $\mathcal{U}$'s signing key. It is easy to see that the ability to do so violates either the security of the signature scheme used for signing $Q$ (using CL signatures [53] this is under Strong RSA) or the completeness of the encryption scheme used to generate the ciphertext $Q$. $\square$

## 6.6 On Removing Random Oracles

Our proofs of security for both System One and System Two are in the random oracle model (in addition to the common parameters model). In the random oracle model, when proving security, control over the output of the hash function is given to the simulator or extractor. This model is very attractive because it often enables simple, efficient proofs. Canetti, Goldreich, and Halevi first exposed flaws in this model by presenting a (contrived) cryptographic protocol that is provably secure in the random oracle model, but for which there provably does *not* exist a real hash function that can securely implement it [66].

In our systems, we use random oracles in only one way: the Fiat-Shamir heuristic [98], which is a method of turning interactive proofs of knowledge into non-interactive ones via a hash function. Unfortunately, Goldwasser and Kalai [115] demonstrated that this is not always theoretically sound. There has not, however, been a practical attack on Fiat-Shamir based protocols, and the results of Goldwasser and Kalai leave open the possibility that some uses of this heuristic may be sound.

So, why not perform interactive proofs and dispense with these random oracles? We can, in all protocols except Spend. The reason is that a spent coin consists of $(S, R, T, \Phi)$, where $\Phi$ is a non-interactive, publicly-verifiable proof of validity. The user must give this to the merchant during Spend; then the merchant must be able to pass on this proof to the bank during Deposit; finally, the bank must have the proof available for VerifyViolation and VerifyOwnership.

While the Fiat-Shamir heuristic is not the only method for removing interaction from proofs, it does more. It allows for *signatures of knowledge* (or signature proofs of knowledge) [59], where we can think of the witness to the proof as a secret key that can be used

to sign messages. That is, a verifier can check that someone who knew a witness to a public statement also certified a given message. We use this property to efficiently bind all pieces of the coin together and to optionally include additional information specific to a transaction, such as the merchant's identity, so that the merchant is the only person who can deposit a particular spent coin. (And thus a hacker cannot steal and then deposit a merchant's coins.)

The concept (and name) of a signature of knowledge was first introduced by Camenisch and Stadler [59] in 1997. Recently, Chase and Lysyanskaya provide a formal treatment of this primitive [72]. As one of their results, they provide an (inefficient) construction in the common random string model that assumes a simulation-sound non-interactive zero knowledge proof system for efficient provers and a special type of encryption scheme, where the public keys are uniformly distributed bitstrings. Their construction does *not* require random oracles. We could use their tool to remove random oracles from our compact e-cash construction, and our protocols would remain polynomial-time, but the efficiency of our Spend protocol would become too expensive to be used in practice today.

## 6.7 Contributions

This chapter presents efficient off-line anonymous e-cash schemes where a user can withdraw a wallet containing $2^\ell$ coins each of which she can spend unlinkably. Our first result is a scheme, secure under the strong RSA and the $y$-DDHI assumptions, where the complexity of the withdrawal and spend operations is $O(\ell+k)$ and the user's wallet can be stored using $O(\ell+k)$ bits, where $k$ is a security parameter. The best previously known schemes require at least one of these complexities to be $O(2^\ell \cdot k)$. In fact, compared to previous e-cash schemes, our whole wallet of $2^\ell$ coins has about the same size as *one* coin in these schemes. Our scheme also offers exculpability of users, that is, the bank can prove to third parties that a user has double-spent. This scheme does *not* require bilinear groups.

We then extend our scheme to our second result, the first e-cash scheme that provides traceable coins without a trusted third party. That is, once a user has double spent one of the $2^\ell$ coins in her wallet, *all* her spendings of these coins can be traced. We present two alternate constructions, both requiring bilinear groups. One construction shares the same complexities with our first result but requires the XDH assumption none to be false for supersingular curves. The second construction works on more general types of elliptic curves, but the price for this is that the complexity of the spending and of the withdrawal protocols becomes $O(\ell \cdot k)$ and $O(\ell \cdot k + k^2)$ bits, respectively, and wallets take $O(\ell \cdot k)$ bits of storage. All our schemes are secure in the random oracle model.

## 6.8 Open Problems

Although this chapter improves the general efficiency of e-cash, several elusive efficiency improvements evaded us. First, we were not able to asymptotically optimize the spending or depositing multiple coins at once. One approach to this problem is to create coin denomi-

nations. While our e-cash system can be replicated to support various denominations, there remains the difficulty of making change.

At a technical level, it would be interesting to know if a $O(\ell+k)$ complexity Spend protocol is possible without random oracles, when the merchant and the bank are not the same entity. (Note that generic NIZK tools do not approach this level of efficiency.) Similarly, it remains open how to efficiency realize tracing without using either a trusted party or bilinear groups.

At a more conceptual level, the anonymity guarantee of our schemes (where no TTP exists!) may be too permissive for actual deployment. That is, the guaranteed anonymity of the system may make it ripe for abuses such as tax evasion and money laundering. In the next Chapter, we directly address this problem. We provide technical solutions that balance a user's desire for anonymity with the societal need to prevent monetary frauds: our solutions will *not* use a TTP.

# Chapter 7

# Compact E-Cash in the Bounded-Anonymity Model

This chapter is an extended version of joint work with Jan Camenisch (IBM Zurich Research) and Anna Lysyanskaya (Brown University) [50].

## 7.1 Introduction

In the previous chapter, we presented an off-line electronic cash system. In such a system, spending a coin once does not leak any information about a user's identity. If a user attempts to spend the same coin twice, however, information is leaked that leads to the identification of this user. An e-cash scheme with such a mechanism is a good illustration of how one can balance anonymity with accountability: a user can remain anonymous unless she performs a forbidden action. The system is designed in a way that detects and allows to punish this type of anonymity abuse.

In this chapter, we consider what other actions may be forbidden, and how to realize e-cash schemes that would hold users accountable should they perform such actions. At the same time, we protect the anonymity of those users who obey the rules.

We introduce the *bounded-anonymity business model.* In this model, associated with each merchant there is a publicly-known limit to the number of e-coins that a user may anonymously give to this merchant. This limit cannot be exceeded even if the user and the merchant collude. Should any user attempt to exceed the limit for any merchant, and should this merchant attempt to submit the resulting e-coins for deposit to the bank, the user's identity will be discovered, and further penalties may be imposed.

In the real world, this corresponds to restrictions that governments set on unreported transactions. For example, in the U.S., banks are required by law to log and report all transactions over $10,000. These restrictions are set up to ensure proper taxation and to prevent money laundering and other monetary frauds. Another example application is an anonymous pass with usage limitations. For example, consider the following amusement park pass: "This pass is good to enter any Disney park up to four times, with the restriction

that the Magic Kingdom can be entered at most twice." Until now, it was not known how to realize such passes anonymously.

Interestingly, in the real world it is impossible to set such restrictions on cash transactions. A merchant may be required by law to report that he received a lot of money in cash, but he may choose not to obey the law! In contrast, we show that with e-cash, it is possible to enforce the bounded-anonymity business model. The cost for achieving this is roughly double the cost of achieving regular anonymous e-cash.

There have been several previous attempts to solve this problem, but until now it remained an elusive open problem in electronic cash, as well as one of the arguments why the financial community resisted any serious deployment of e-cash, due to money laundering regulations.

Some of the past efforts suggested using a trusted third party to mitigate this problem [175, 46, 133, 125]. This TTP could trace transactions to particular users. The TTP approach is undesirable. First of all, the whole idea of electronic cash is to ensure that no one can trace e-cash transactions. Secondly, in these past solutions, the only way that a TTP can discover money laundering or other violations of the bounded-anonymity model is by tracing each transaction, which is very expensive. In a variant that reduces the trust assumption about the TTP, Kügler and Vogt [132] propose an e-cash scheme where the bank has the ability to trace coins by specially marking them during the withdrawal protocol. This tracing is auditable, i.e., a user can later find out whether or not his coins were traced (this involves an additional trusted judge). Still, this system does not allow one to discover money laundering, unless it involves the marked coins, and the user must still trust the judge and bank for her anonymity. Another variant [158, 124] deters money laundering by offering users only a limited form of anonymity. Users' coins are anonymous, but linkable, i.e., coins from the same user can be identified as such. Here it is easy to detect if a user exceeds the spending limit with some merchant. However, this weak form of anonymity is not suitable for all applications, and goes against the principle of e-cash.

Another set of papers [176, 153] addressed a related problem of allowing a user to show a credential anonymously and unlinkably to any given verifier up to $k$ times. They give a nice solution, but it is not clear how it can be applied to *off-line* electronic cash as opposed to on-line anonymous authentication. I.e., showing an anonymous credential in their scheme more than $k$ times allows a verifier to link the $(k+1)$-st show to a previous transaction, but does not lead to the identification of the misbehaving user. In contrast, in our scheme, any such violation leads to identification of the user even if the verifier (merchant) colludes with the user, and optionally to trace all the user's past transactions. Recently, Teranishi and Sako [177] provided a solution in the *on-line* setting that allows authorities to identify a cheater, however, their solution is exponentially less efficient than ours.

Finally, Sander and Ta-Shma [168] propose to limit money laundering by dividing time into short time periods and issuing at most $k$ coins to a user per time period (a user can deposit his unspent coins back into his account). This way, a user cannot spend more than $k$ coins in a single transactions because he has at most $k$ coins at any given time.

**Outline of this Chapter.** We present the first e-cash scheme in the bounded-anonymity business model. A user may withdraw, and anonymously and unlinkably spend an unlimited number of coins, so long as she does not: (1) double-spend a coin, or (2) exceed the spending limit for any merchant. Different merchants may have different spending limits, but a merchant's limit applies uniformly to all of its customers. As we will see, this limit does not need to be known (or even fixed) at the time that the user withdraws coins from the bank, but the limit will need to be known at the time that the user spends a coin with the merchant and also when the bank later accepts a spent coin for deposit.

Our scheme allows to efficiently detect either of two system violations. We also show how to augment it so as to allow to reveal the identity of the misbehaving user. Finally, in addition to discovering the identity of a misbehaving user, one is also able to trace all of the user's previous e-coins. Simultaneously achieving all functionality described above will require bilinear groups, while achieving bounded-anonymity without tracing is possible using other algebraic groups.

Our construction takes as a starting point the e-cash system of Camenisch, Hohenberger, and Lysyanskaya (CHL) [49], as covered in Chapter 6, which is the most efficient known. The cost of our resulting withdrawal and spend protocols is roughly double that of the previous chapter. The size of the coin storage remains the same, but we also require the user to store a counter for each merchant with whom the user does business, which appears to be optimal. Thus we maintain our previous asymptotic complexity: $2^\ell$ coins can be stored in $O(\ell + k)$ bits and the complexity of the withdrawal and spend protocols is $O(\ell + k)$, where $k$ is the security parameter.

## 7.2 Definition of Security

We now generalize the definition of CHL [49], as in Section 6.2, to handle violations beyond double-spending. Our off-line e-cash scenario consists of the three usual players: the user, the bank, and the merchant; together with the algorithms BKeygen, UKeygen, Withdraw, Spend, Deposit, {DetectViolation$^{(i)}$, IdentifyViolator$^{(i)}$, VerifyViolation$^{(i)}$}, Trace, and VerifyOwnership. Informally, BKeygen and UKeygen are the key generation algorithms for the bank and the user, respectively. A user interacts with the bank during Withdraw to obtain a wallet of $2^\ell$ coins; the bank stores optional tracing information in database $D$. In Spend, a user spends one coin from his wallet with a merchant; as a result the merchant obtains the serial number $S$ of the coin, the merchant record locator $V$ of the coin, and a proof of validity $\pi$. Recall that our basic e-cash scheme in Chapter 6, the serial number was used to detect double-spending. Similarly, the merchant record locator will be used to detect exceeding the spending limit for a merchant.

In Deposit, whenever an honest merchant accepts a coin $C = (S, V, \pi)$ from a user, there is a guarantee that the bank will accept this coin for deposit. The bank stores $C = (S, V, \pi)$ in database $L$. At this point, however, the bank needs to determine if $C$ violates any of the system conditions.

For each violation $i$, we define a tuple of algorithms {DetectViolation$^{(i)}$, IdentifyViolator$^{(i)}$,

$\mathsf{VerifyViolation}^{(i)}\}$. Here, we have two violations.

- *Violation 1: Double-spending.* In $\mathsf{DetectViolation}^{(1)}$, the bank tests if two coins, $C_1 = (S_1, V_1, \pi_1)$ and $C_2 = (S_2, V_2, \pi_2)$, in $L$ have the same *serial number* $S_1 = S_2$. If so, the bank runs the $\mathsf{IdentifyViolator}^{(1)}$ algorithm on $(C_1, C_2)$ and obtains the public key $pk$ of the violator and a proof of guilt $\Pi$. Anyone can run $\mathsf{VerifyViolation}^{(1)}$ on $(pk, S_1, V_1, \Pi)$ to be convinced that the user with public key $pk$ double-spent the coin with serial number $S_1$.

- *Violation 2: Money-laundering.* In $\mathsf{DetectViolation}^{(2)}$, the bank tests if two coins, $C_1 = (S_1, V_1, \pi_1)$ and $C_2 = (S_2, V_2, \pi_2)$, in $L$ have the same *merchant record locator* $V_1 = V_2$. If so, the bank runs the $\mathsf{IdentifyViolator}^{(2)}$ algorithm on $(C_1, C_2)$ and obtains the public key $pk$ of the violator and a proof of guilt $\Pi$. Anyone can run $\mathsf{VerifyViolation}^{(2)}$ on $(pk, S_1, V_1, \Pi)$ to be convinced that the user with public key $pk$ exceeded the bounded-anonymity business limit by spending the coin with merchant record locator $V_1$.

Optionally, after any violation, the bank may also run the $\mathsf{Trace}$ algorithm on a valid proof of guilt $\Pi$ to obtain a list of all serial numbers $S_i$ ever spent by the cheating user, with public key $pk$, along with a proof of ownership $\Gamma$. Anyone can run $\mathsf{VerifyOwnership}$ on $(pk, S_i, \Gamma)$ to be convinced that the user with public key $pk$ was the owner of the coin with serial number $S_i$.

**Security.** We generalize the security definition of CHL for e-cash [49], as covered in Section 6.2. The formalizations of correctness, balance, and anonymity of users remain unchanged. Roughly, *balance* guarantees that an honest bank will never have to accept for deposit more coins than users withdrew, while *anonymity of users* assures users that they remain completely anonymous unless they violate one of the known system conditions. We now informally describe three additional properties. These properties are generalizations of CHL's identification and tracing of double-spenders, and their exculpability, to apply to any specified violation, in particular those above. Let *params* be the global parameters, including the number of coins per wallet, $2^\ell$, and a (possibly unique) spending limit for each merchant. (Recall that each merchant may have a different spending limit, but that a merchant's limit will apply uniformly for all of its customers.)

*Identification of violators.* Suppose coins $C_1 = (S_1, V_1, \pi_1)$ and $C_2 = (S_2, V_2, \pi_2)$ are the output of an honest merchant (or possibly merchants) running two $\mathsf{Spend}$ protocols with the adversary or they are two coins that an honest bank accepted for deposit. This property guarantees that, with high probability, if, for some $i$, the algorithm $\mathsf{DetectViolation}^{(i)}(params, C_1, C_2)$ accepts, then $\mathsf{IdentifyViolator}^{(i)}(params, C_1, C_2)$ outputs a key $pk$ and a proof $\Pi$ such that $\mathsf{VerifyViolation}^{(i)}(params, pk, S_1, V_1, \Pi)$ accepts.

*Tracing of violators.* Suppose $\mathsf{VerifyViolation}^{(i)}(params, pk, S, V, \Pi)$ accepts for some violation $i$ derived from coins $C_1, C_2$. This property guarantees that, with high probability, $\mathsf{Trace}(params, pk, C_1, C_2, \Pi, D)$ outputs the serial numbers $S_1, \ldots, S_m$ of all coins belonging to the user of $pk$ along with proofs of ownership $\Gamma_1, \ldots, \Gamma_m$ such that for all $j$, $\mathsf{VerifyOwnership}(params, pk, S_j, \Gamma_j)$ accepts.

*Exculpability.* Suppose an adversary participates any number of times in the Withdraw protocol with an honest user with key $pk$, and subsequently to that, in any number of *non-violation* Spend protocols with the same user. The adversary then outputs an integer $i$, a coin serial number $S$, and a purported proof $\Gamma$ that the user with key $pk$ committed violation $i$ and owns coin $S$. The *weak* exculpability property states that, for all adversaries, the probability that VerifyOwnership$(params, pk, S, \Gamma)$ accepts is negligible.

Furthermore, the adversary may continue to engage the user in Spend protocols, forcing her to violate the system conditions. The adversary then outputs $(i, S, V, \Pi)$. The *strong* exculpability property states that, for all adversaries: (1) when $S$ is a coin serial number *not* belonging to the user of $pk$, weak exculpability holds, and (2) when the user of $pk$ did *not* commit violation $i$, the probability that VerifyViolation$^{(i)}(params, pk, S, V, \Pi)$ accepts is negligible.

As mentioned in Chapter 6.2, a worthwhile project is to write and realize an ideal functionality for electronic cash in the UC framework. We mention that the notion of multiple violations introduced in this chapter, in particular the notion of bounded anonymity, could be analyzed using a hybrid approach as before, where access is given to ideal functionalities for zero-knowledge and signatures of knowledge.

## 7.3 Compact E-Cash in the Bounded-Anonymity Model

### 7.3.1 Roadmap to Understanding the Construction

As in the CHL compact e-cash scheme, a user withdraws a wallet of $2^\ell$ coins from the bank and spends them one by one. Also, as in the CHL scheme, we use a pseudorandom function $F_{(\cdot)}(\cdot)$ whose range is some group $G$ of large prime order $q$.

At a high level, a user forms a wallet of $2^\ell$ coins by picking five values, $(x, s, t, v, w)$ from an appropriate domain to be explained later, and running an appropriate secure protocol with the Bank to obtain the Bank's signature $\sigma$ on these values.

Suppose that the user wants to spend coin number $i$ by buying goods from merchant $M$. Suppose that only up to $K$ transactions with this merchant may be anonymous. Let's say that this is the user's $j$-th transaction with $M$, $j \leq K$. Associated with the $i$-th coin in the wallet is its serial number $S = F_s(i)$. Associated with the $j$-th transaction with the merchant $M$ is the merchant's record locator $V = F_v(M, j)$.

The first idea is that in the Spend protocol, the user should give to the merchant the values $(S, V)$, together with a (non-interactive zero-knowledge) proof that these values are computed as a function of $(s, i, v, M, j)$, where $0 \leq i < 2^\ell$, $0 \leq j < K$, and $(s, v)$ correspond to a wallet signed by the Bank. Note that $S$ and $V$ are pseudorandom, and therefore computationally leak no information; and the proof leaks no information because it is zero-knowledge.

Suppose that a user spends more than $2^\ell$ coins. Then he must have used some serial number more than once, since there are only $2^\ell$ possible values $S$ of the form $F_s(i)$ where

$0 \leq i < 2^\ell$. (This is the CHL observation.) Similarly, suppose that a user made more than $K$ transactions with $M$. Then he must have used some merchant record locator more than once, since for a fixed $M$, there are only $K$ different values $V = F_v(M, j)$, $0 \leq j < K$. Therefore it is easy to see that double-spending and violations of the bounded-anonymity business model can be detected.

Now we need to explain how to make sure that using any $S$ or $V$ more than once leads to identification. Remember that besides $s$ and $v$, the wallet also contains $x$, $t$ and $w$. The value $x \in \mathbb{Z}_q$ is such that $g^x$ is a value that can be publicly linked to the user's identity. (Here $g$ is a generator of the group $\mathbb{G}$.) For example, for some computable function $f$, $f(g^x)$ can be the user's public key. Suppose that as part of the transaction the merchant contributes a random value $R \neq 0$, and the user reveals $T = g^x \cdot F_t(i)^R$ and $W = g^x \cdot F_w(M, j)^R$, together with a proof that $T$ and $W$ are computed appropriately as a function of $(R, x, t, i, w, M, j)$ corresponding to the very same wallet and the same $i$ and $j$. Again, $T$ and $W$ are pseudorandom and therefore do not leak any information.

If a user uses the same serial number $S = F_s(i)$ twice, and $q$ is appropriately large, then with high probability in two different transactions she will receive different $R$'s, call them $R_1$ and $R_2$, and so will have to respond with $T_1 = g^x \cdot F_t(i)^{R_1}$, $T_2 = g^x \cdot F_t(i)^{R_2}$. It is easy to see that the value $g^x$ can then be computed as follows: $g^x = T_1/(T_1/T_2)^{R_1/(R_1-R_2)}$. This was discovered by CHL building on the original ideas of offline e-cash [77].

We show that it is also the case that if the user uses the same merchant's record locator number $V$ twice, then $g^x$ can be found in exactly the same fashion. Suppose that in the two transactions the merchant used the same $R$. In that case, the Bank can simply refuse to deposit this e-coin (since it is the same merchant, he is responsible for his own lack of appropriate randomization). So suppose that the merchant used two different $R$'s, $R_1$ and $R_2$, giving rise to $W_1$ and $W_2$. It is easy to see that $g^x = W_1/(W_1/W_2)^{R_1/(R_1-R_2)}$.

Thus, a double-spending or a violation of the bounded-anonymity model leads to identification. The only remaining question is how this can be adapted to trace other transactions of the same user. Note that $g^x$ is not necessarily a public value, it may also be the case that only $f(g^x)$ is public, while knowledge of $g^x$ gives one the ability to decrypt a ciphertext which was formed by verifiably encrypting $s$ (for example, Boneh and Franklin's cryptosystem [30] has the property that $g^x$ is sufficient for decryption). When withdrawing a wallet, the user must give such a ciphertext to the bank. In turn, knowledge of $s$ allows the Bank to discover serial numbers of all coins from this wallet and see how they were spent.

Finally, note that the values $(x, v, w)$ should be tied to a user's identity and not to a particular wallet. This way, even if a user tries to spend too much money with a particular merchant from different wallets, it will still lead to detection and identification.

### 7.3.2 Our Construction

We use the same building blocks as in Chapter 6: the Dodis-Yampolskiy pseudo-random function [95], i.e., $f^{DY}_{(g,s)}(x) = g^{1/(s+x+1)}$, where $g$ is the generator of a suitable group; CL-signatures [53] and the related protocols to issue signatures and prove knowledge of

signatures; and the Bilinear El Gamal cryptosystem [8, 9] (see Chapter 5) used with the Camenisch-Damgård [47] verifiable encryption techniques.

**Global parameters.** Let $1^k$ be the security parameter and let $2^\ell$ be the number of coins per wallet. This scheme work most efficiently by having four different groups:

- $\mathbf{G} = \langle \mathbf{g} \rangle$, where $n$ is a special RSA modulus of $2k$ bits, $\mathbf{g}$ is a quadratic residue modulo $n$, and $\mathbf{h} \in \mathbf{G}$.
- $\mathbb{G}_1 = \langle g_1 \rangle = \langle h_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle = \langle h_2 \rangle$, and $\mathbb{G}_T$ of prime order $q$ such that exists a bilinear mapping $\mathsf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

Furthermore, we have two hash functions $H_T : \{0,1\}^* \to \mathbb{G}_T$ and $H_1 : \{0,1\}^* \to \mathbb{G}_1$. Let $F_{(g,s)}(x) = f^{DY}_{(g,s)}(x)$, and when $H$ is a hash function whose range is an appropriate group, let $G^H_s(M,x) = f^{DY}_{(H(M),s)}(x)$. Let the global parameters above be denoted $\zeta$.

We now describe the protocols of our system: BKeygen, UKeygen, Withdraw, Spend, Deposit, $\{\mathsf{DetectViolation}^{(i)}, \mathsf{IdentifyViolator}^{(i)}, \mathsf{VerifyViolation}^{(i)}\}_{i\in[1,2]}$ (where the two violations are double-spending and exceeding the spending-limit with a merchant), Trace, and VerifyOwnership.

**Bank Key Generation:** $\mathsf{BKeygen}(1^k, \zeta)$**:** The bank $\mathcal{B}$ generates CL signing keys $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$ set in group $\mathbf{G}$ as before.

**User Key Generation:** $\mathsf{UKeygen}(1^k, \zeta)$**:** Each user generates a key pair of the form $sk_{\mathcal{U}} = (x, v, w)$ and $pk_{\mathcal{U}} = (\mathsf{e}(g_1, h_2)^x, \mathsf{e}(g_1, h_2)^v, \mathsf{e}(g_1, h_2)^w)$, where $x$, $v$, and $w$ are chosen randomly from $\mathbb{Z}_q$. Each user also generates a signing keypair for any secure signature scheme. Each merchant also publishes a unique identity string $id_{\mathcal{M}}$.

Also, the bank and each merchant individually agree on a public upper-bound $N_{\mathcal{M}}$ for the number of coins any single customer can spend with that merchant.

**Withdrawal Protocol:** $\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_{\mathcal{U}}, 2^\ell), \mathcal{B}(pk_{\mathcal{U}}, sk_{\mathcal{B}}, 2^\ell)$**:** A user $\mathcal{U}$ withdraws $2^\ell$ coins from the bank $\mathcal{B}$ as follows. The user and the bank engage in an interactive protocol, and if neither report an error, then at the end:

1. $\mathcal{U}$ obtains $(s, t, \sigma)$, where $s, t$ are random values in $\mathbb{Z}_q$, and $\sigma$ is the bank's signature on $(sk_{\mathcal{U}}, s, t)$, i.e., $(x, v, w, s, t)$.
2. $\mathcal{B}$ obtains a verifiable encryption of $s$ under $\mathsf{e}(g_1, h_2)^x$, i.e., the first element from the user's public key $pk_{\mathcal{U}}$, together with the user's signature on this encryption. $\mathcal{B}$ does not learn anything about $sk_{\mathcal{U}}$, $s$, or $t$.

Step one can be efficiently realized using the Camenisch-Lysyanskaya signatures and the related protocols [54]. Step two can be realized by applying the Camenisch-Damgård [47] verifiable encryption techniques to the Bilinear El Gamal cryptosystem [8, 9] as described in Chapter 5. All these steps are essentially the same as in the CHL e-cash scheme, the exception being the secret key signed which now also includes $v$ and $w$ besides $x$.

**Spend Protocol:** $\mathsf{Spend}(\mathcal{U}(wallet, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, 2^{\ell}))$: A user $\mathcal{U}$ spends one coin with a merchant $\mathcal{M}$ with a spending limit of $N_{\mathcal{M}}$ coins as follows. As in CHL, the user keeps a private counter $i$ from 0 to $2^{\ell}-1$ for the number of coins spent in her wallet. Additionally, the user now also keeps a counter $j_{\mathcal{M}}$ for each merchant $\mathcal{M}$ representing the number of coins she has spent with that merchant.

1. $\mathcal{U}$ checks that she is under her spending limit with merchant $\mathcal{M}$; that is, that $j_{\mathcal{M}} < N_{\mathcal{M}}$. If not, she aborts.
2. $\mathcal{M}$ sends fresh values $R_1, R_2 \in \mathbb{Z}_q^2$ to $\mathcal{U}$. $\mathcal{M}$ may either keep a counter or select random values; we just require that $\mathcal{M}$ not have used either of these values previously with any user.
3. $\mathcal{U}$ sends $\mathcal{M}$ the $i$-th coin in her wallet on her $j_{\mathcal{M}}$-th transaction with $\mathcal{M}$. Recall that $sk_{\mathcal{U}} = (x, v, w)$. This coin consists of a serial number $S$ and a wallet check $T$, where

   $$S = F_{(e(g_1,h_2),s)}(i) = e(g_1, h_2)^{1/(s+i+1)} \quad , \quad T = g_1^x (F_{(g_1,t)}(i))^{R_1} = g_1^{x+R_1/(t+i+1)}$$

   and two money laundering check values $V$ and $W$, where

   $$V = G_v^{H_T}(id_{\mathcal{M}}, j_{\mathcal{M}}) = H_T(id_{\mathcal{M}})^{1/(v+j_{\mathcal{M}}+1)} \quad ,$$
   $$W = g_1^x (G_w^{H_1}(id_{\mathcal{M}}, j_{\mathcal{M}}))^{R_2} = g_1^x H_1(id_{\mathcal{M}})^{R_2/(w+j_{\mathcal{M}}+1)}$$

   and a zero-knowledge proof of knowledge (ZKPOK) $\pi$ of $(i, j_{\mathcal{M}}, sk_{\mathcal{U}} = (x, v, w), s, t, \sigma)$ proving that

   (a) $0 \le i < 2^{\ell}$;
   (b) $0 \le j_{\mathcal{M}} < N_{\mathcal{M}}$;
   (c) $S = F_{(e(g_1,h_2),s)}(i)$, i.e., $S = e(g_1, h_2)^{1/(s+i+1)}$;
   (d) $T = g_1^x \cdot (F_{(g_1,t)}(i))^{R_1}$, i.e., $T = g_1^{x+R_1/(t+i+1)}$;
   (e) $V = G_v^{H_T}(id_{\mathcal{M}}, j_{\mathcal{M}})$, i.e., $V = H_T(id_{\mathcal{M}})^{1/(v+j_{\mathcal{M}}+1)}$;
   (f) $W = g_1^x \cdot (G_w^{H_1}(id_{\mathcal{M}}, j_{\mathcal{M}}))^{R_2}$, i.e., $W = g_1^x \cdot H_1(id_{\mathcal{M}})^{R_2/(w+j_{\mathcal{M}}+1)}$ ; and
   (g) $\mathrm{VerifySig}(pk_{\mathcal{B}}, (sk_{\mathcal{U}} = (x, v, w), s, t), \sigma)$=true.

   The proof $\pi$ can be made non-interactive using the Fiat-Shamir heuristic [98].
4. If $\pi$ verifies and the value $V_j$ was never seen by $\mathcal{M}$ before, then $\mathcal{M}$ accepts and saves the coin $(R_1, R_2, S, T, V, W, \pi)$. If the value $V_j$ was previously seen before in a coin $(R_1', R_2', S', T', V, W', \pi')$, then $\mathcal{M}$ contacts the bank which runs $Open(W', W, R_2', R_2)$.

   Let us define the $Open(\cdot,\cdot,\cdot,\cdot)$ algorithm as:

   $$Open(A, B, D, D) := \frac{A}{(A/B)^{C/(C-D)}} \quad .$$

   If $\mathcal{M}$ executed the Spend protocols honestly (i.e., chose fresh random values at the start of each protocol), then with high probability $R_2 \neq R_2'$, and we have

$Open(W', W, R_2', R_2) = g_1^x$. Thus, the merchant can identify the user by computing $e(g_1^x, h_2)$, which is part of $\mathcal{U}$'s public key. This allows an honest merchant to protect itself from customers who would try to overspend with it. (If the merchant is dishonest or lazy, the bank will catch the overspending at deposit time.)

Steps 3(a,c,d) are the same as in the CHL scheme whereas Steps 3(b,e,f) are new; note, however, that they are rather similar to Steps 3(a,c,d). Finally, Step 3(g) needs to be adapted properly. Consequently, Steps 3(a) and 3(b) can be done efficiently using standard techniques [70, 55, 37]. Steps 3(c) to 3(f) can be done efficiently using techniques of Camenisch, Hohenberger, and Lysyanskaya [49]. Step 3(g) can be done efficiently using the Camenisch and Lysyanskaya signatures [54].

**Deposit Protocol:** $\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, S, \pi, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$**:** A merchant $\mathcal{M}$ deposits a coin with bank $\mathcal{B}$ by submitting the coin $(R_1, R_2, S, T, V, W, \pi)$. The bank checks the proof $\pi$; it if does not verify, the bank rejects immediately. If there exists a coin in $L$ with the same values for $(R_1, S)$ or the same values for $(R_2, V)$, then the bank rejects immediately and punishes the merchant. (This situation won't occur if the merchant is honest.) Otherwise, the bank adds the coin to its database $L$.

**Detection of Double-Spending:** $\mathsf{DetectViolation}^{(1)}(\zeta, L)$**:** $\mathcal{B}$ checks that the spender of the coin has not overspent her wallet; that is, the bank searches for two deposited coins in $L$ with the same serial number $S$.

**Identification of a Double-Spender:** $\mathsf{IdentifyViolator}^{(1)}(\zeta, S, C_1, C_2)$**:** Suppose that two coins $C_1 = (R_1, R_2, S, T, V, W, \pi)$ and $C_2 = (R_1', R_2', S, T', V', W', \pi')$ with the same serial number are found. We know that $R_1 \neq R_1'$, otherwise $\mathcal{B}$ would have refused to accept the coin. Now, the identity of the user is discovered by running $Open(T', T, R_1', R_1) = g_1^x$. $\mathcal{B}$ identifies the user as person with public key containing $e(g_1^x, h_2)$. $\mathcal{B}$ outputs the proof of guilt $\Pi = (g_1^x, C_1, C_2)$.

**Detection of Money Laundering:** $\mathsf{DetectViolation}^{(2)}(\zeta, L)$**:** Second, $\mathcal{B}$ checks that the spender of the coin has not exceeded her spending limit with merchant $\mathcal{M}$. That is, the bank searches for two coins in database $L$ with the same money-laundering check value $V$.

**Identification of Money Launderer:** $\mathsf{IdentifyViolator}^{(2)}(\zeta, V, C_1, C_2)$ : Suppose that two coins $C_1 = (R_1, R_2, S, T, V, W, \pi)$ and $C_2 = (R_1', R_2', S', T', V, W', \pi')$ with the same merchant check value are found. When know that $R_2 \neq R_2'$, otherwise $\mathcal{B}$ would have refused to accept the coin. Now, the identity of the user is discovered by executing $Open(W', W, R_2', R_2) = g_1^x$. $\mathcal{B}$ identifies user as person with public key containing $e(g_1^x, h_2)$. $\mathcal{B}$ outputs the proof of guilt $\Pi = g_1^x, C_1, C_2$.

**Trace Transactions:** $\mathsf{Trace}(\zeta, \Pi)$ : $\mathcal{B}$ computes the serial number of all coins owned by the violator as follows:

1. Parse $\Pi$ as $(g_1^x, C_1, C_2)$.
2. $\mathcal{B}$ uses $g_1^x$ to decrypt the encryption of $s$ left with the bank during the withdraw protocol.
3. Next, $\mathcal{B}$ uses $s$ to compute the serial numbers $S_j = F_{(e(g_1,h_2),s)}(j)$ for each coin $j = 0$ to $2^\ell - 1$ of all coins in the user's wallet.
4. (In fact, the bank can use $g_1^x$ to decrypt the secret of *all* the user's wallets and trace those transactions in the same way.)

**Verifying Violations:** For completeness, we point out explicitly how the violation-related protocols work. Let $C_1 = (R_1, R_2, S, T, V, W, \pi)$ and $C_2 = (R_1', R_2', S', T', V', W', \pi')$ be one existing and one newly deposited coin. Detecting double-spending or money-laundering involves checking $S_1 = S_2$ or $V_1 = V_2$, respectively. The identification algorithm runs *Open* on the appropriate inputs, and the resulting proof of guilt is $\Pi = (C_1, C_2)$. Verifying the violation entails successfully checking the validity of the coins, detecting the claimed violation, running *Open* to obtain $g_1^x$, and checking its relation to $pk$. (Recall that knowledge of $x$, not just $g_1^x$, is required to create a valid coin. Thus the leakage of one violation cannot be used to spend the user's coins or fake another violation.) The trace algorithm involves recovering $s$, from the encryption $E$ signed by the user during Withdraw, and computing all serial numbers. The proof of ownership $\Gamma = (E, \sigma, g_1^x)$, where $\sigma$ is the user's signature on $E$. Verifying ownership for some serial number $S$ involves verifying the signature $\sigma$, checking that $e(g_1^x, h_2) = pk$, decrypting $E$ to recover $s$, computing all serial numbers $S_i$, and testing if, for any $i$, $S = S_i$.

**Efficiency Discussion.** The detailed protocols are rather similar to the detailed ones of the CHL scheme [49] and require slightly less than double the work for the participants). As indication of the protocols efficiency let us state some numbers here. For instance, one can construct Spend such that a user must compute fourteen multi-base exponentiations to build the commitments and twenty more for the proof. The merchant and bank need to do twenty multi-base exponentiations to check that the coin is valid. The protocols require two rounds of communication between the user and the merchant and one round between the bank and the merchant. If one takes Option (2) above, then it is thirteen multi-base exponentiations to build the commitments and eighteen more for the proof. Verification by bank and merchant takes eighteen multi-base exponentiations.

**Theorem 7.1 (Security of System).** *In the bounded-anonymity business model, our scheme guarantees balance, anonymity of users, identification of violators, tracing of violators, and strong exculpability under the Strong RSA, y-DDHI, and either the XDH or Sum-Free DDH assumptions in the random oracle model.*

Let us being with some informal intuition for the security of this scheme, followed by an actual proof sketch. Our contribution is the observation that the core observations of CHL in Chapter 6 can be extended to prevent violations beyond double-spending, particularly through careful use of the PRF.

*Balance.* For each wallet, $s$ deterministically defines exactly $N$ values that can be valid serial numbers for coins. To overspend a wallet, a user must either use one serial number twice, in which case she is identifiable, or she must forge a CL signature or fake a ZK proof.

*Anonymity of users.* A coin is comprised of four values $(S, T, V, W)$, which are pseudorandom and thus leak no information about the user, together with a (non-interactive, zero-knowledge) proof of validity, which since it is zero-knowledge also leaks nothing.

The only abnormality here is that, when computing $V$ and $W$, the base used for the PRF is the hash of the merchant's identity (as opposed to the fixed bases used to compute $S$ and $T$). Treating hash $H$ as a random oracle, we see that given $G_v^H(id_{\mathcal{M}}, j)$, the output of $G_v^H(\cdot,\cdot)$ on any other input, in particular $G_v^H(id'_{\mathcal{M}}, j)$ for $id_{\mathcal{M}} \neq id'_{\mathcal{M}}$, is indistinguishable from random. Specifically, if an adversary given $G_v^H(id_{\mathcal{M}}, j) = f^{DY}_{(H(id_{\mathcal{M}}),v)}(j) = H(id_{\mathcal{M}})^{1/(v+j+1)}$ can distinguish $H(id'_{\mathcal{M}})^{1/(v+j+1)}$ from random for some random, fixed $H(id_{\mathcal{M}})$ and $H(id'_{\mathcal{M}})$, then it is solving DDH.

*Exculpability.* First, a user cannot be proven guilty of a violation if he follows all the protocols honestly, because the proof of guilt includes the user's *secret* value $g_1^x$. This value is not leaked when the user follows the protocols. Second, even a cheating user cannot be proven guilty of a crime he didn't commit— e.g., double-spending one coin does not enable a false proof of money-laundering twenty coins —because: (1) guilt is publicly verifiable from the coins themselves, and (2) knowledge of $x$ is required to create coins. The value $g_1^x$, which is leaked by a violation, is not enough to spend a coin from that user's wallet.

This ends our intuition on the proof, now let us provide additional details.

*Proof sketch.* We sketch the proof based on the definition outlined in Section 7.2.

*Balance:* Let Adv be an adversary who executes $n$ Withdraw protocols with an extractor $\mathcal{E}$ acting as the bank. During each Withdraw, $\mathcal{E}$ acts exactly as an honest bank would, except that when Adv gives a proof of knowledge of $(x, s, t, v, w)$, $\mathcal{E}$ extracts these values. To extract using the CL techniques requires rewinding, thus, Withdraw should be executed sequentially, or alternatively, only a bounded number of concurrent executions should be allowed. From $s$, $\mathcal{E}$ can compute, with high probability, all legal serial numbers that Adv can compute for that wallet. Let $L$ denote the set of all serial numbers from all wallets. Now suppose that Adv can cause an honest bank to accept a coin with serial number $S \notin L$. Then it follows that Adv must have concocted a false proof of validity, which happens with negligible probability under the Strong RSA and discrete logarithm assumptions.

*Anonymity of users:* Recall that the Chapter 6 definition of anonymity requires a simulator $\mathcal{S}$ that can successfully execute the user's side of the Spend protocol without access to a user's wallet, secret key, or even the user's public key. Suppose $\mathcal{S}$ wishes to spend a coin with merchant $\mathcal{M}$, where the wallet size is $N$ and $\mathcal{M}$'s anonymous-spending limit is $M$. $\mathcal{S}$ chooses random values $(x, s, t, v, w)$ and any $0 \leq i < 2^\ell$ and $0 \leq j_{\mathcal{M}} < M$. Next, $\mathcal{S}$ creates the coin serial number $S = F_{(e(g_1,h_2),s)}(i)$, merchant record locator $V = G_v^{H_T}(id_{\mathcal{M}}, j_{\mathcal{M}})$, and check values $T = g_1^x \cdot F_{(g_1,t)}(i)^{R_1}$ and $W = g_1^x \cdot G_w^{H_1}(id_{\mathcal{M}}, j_{\mathcal{M}})^{R_2}$ *exactly* as an honest user would. Because these values are pseudorandom, they are indistinguishable from a real coin. $\mathcal{S}$ can honestly prove that $(S, T, V, W)$ are well-formed and that $i, j_{\mathcal{M}}$ are in the appropriate ranges. The only part of the proof that $\mathcal{S}$ must fake is that $\mathcal{S}$ has a signature from the bank

on $(x, s, t, v, w)$. Here, $\mathcal{S}$ invokes the appropriate CL simulator for this step, which requires control of the random oracle for the Fiat-Shamir heuristic. This also requires that Spend be executed with bounded concurrency.

*Identification of violators:* As described in balance, an extractor $\mathcal{E}$ executes multiple Spend protocols with an adversary Adv, with high probability, extracting $(x, s, t, v, w)$. For each wallet, $\mathcal{E}$ can compute all valid coin serial numbers $S_i = F_{e(g_1,h_2),s)}(i)$ for $0 \leq i < 2^\ell$ and all valid merchant record locators $V_j = G_v^{H_T}(id_\mathcal{M}, j_\mathcal{M})$ for $0 \leq j_\mathcal{M} < M$ and some merchant $\mathcal{M}$. Now suppose that Adv can cause an honest bank to accept two coins $C_1 = (S_1, T_1, V_1, W_1, \pi_1, R_1, R_1')$ and $C_2 = (S_2, T_2, V_2, W_2, \pi_2, R_2, R_2')$, with the same serial number $S_1 = S_2$ or two coins with the same merchant record locator $V_1 = V_2$. For these collisions to happen, the coins must come from the same wallet $(x, s, t, v, w)$, with either the same $i$ or the same $(id_\mathcal{M}, j_\mathcal{M})$, and be well formed. (Otherwise, the user must have faked the proof of validity.) With high probability then, either $g_1^x = T_1/(T_1/T_2)^{R_1/(R_1-R_2)}$ or $g_1^x = W_1/(W_1/W_2)^{R_1'/(R_1'-R_2')}$, which reveals the user's identity.

The proof of guilt $\Pi$ for violation $i = 1$ (double-spending) contains valid coins $(C_1, C_2)$. Anyone may check that $S_1 = S_2$, $g_1^x = T_1/(T_1/T_2)^{R_1/(R_1-R_2)}$, and $g_1^x$ corresponds to $pk$ (e.g., $f(g_1^x) = pk$). Likewise, the proof of guilt for violation $i = 2$ (exceeding the anonymous-business bound with a particular merchant) is valid coins $(C_1, C_2)$ such that $V_1 = V_2$, $g_1^x = W_1/(W_1/W_2)^{R_1'/(R_1'-R_2')}$, and $g_1^x$ corresponds to $pk$.

One technicality is what happens if two wallet seeds $s, s'$ are such that $|s - s'| < 2^\ell$; thus these wallets overlap in at least one serial number. If the same serial number is spent from both wallets, the bank will flag them as being double-spent, and yet the identification algorithm will not work, because no double-spending has actually occurred. Since this could be confusing for the bank, we avoid this problem, as we did in Chapter 6, by having the bank contribute randomness to the choice of $s$ for each wallet. When $s, s'$ are drawn at random from $\mathbb{Z}_q$, the chance that they overlap is $2^{\ell+1}/q$.

*Tracing of violators:* Recall that during Withdraw, each user is required to give the bank a *verifiable* encryption $E$ of her wallet secret $s$ such that her own key $g_1^x$ suffices to decrypt. The user is also required to sign this encryption. By the identification of violators property, we know that a system violation allows the bank to recover $g_1^x$ with high probability. By the integrity of the verifiable encryption, each wallet belonging to the cheating user can be opened, with high probability, using $g_1^x$ to recover the wallet seed $s$ and compute all serial numbers $S_i = F_{(e(g_1,h_2),s)}(i)$ for $0 \leq i < 2^\ell$.

The proof of ownership $\Gamma$ for some coin with serial number $S$ is comprised of the encryption $E$, the decryption key $g_1^x$, and the signature on $E$. A verifier should first check that $g_1^x$ corresponds to $pk$, and that the signature on $E$ is valid. He may then decrypt $E$ to recover $s$, compute all serial numbers $S_i = F_{(e(g_1,h_2),s)}(i)$ for $0 \leq i < 2^\ell$, and test that, for some $i$, $S = S_i$.

*Strong exculpability:* Strong exculpability has two parts. First, that no adversary interacting with an honest user can produce values $(i, S, V, \Pi)$ such that $\mathsf{VerifyViolation}^{(i)}(params, S, V, \Pi)$ accepts, unless the user with $pk$ is guilty of violation $i$. Recall that the proof $\Pi$ is comprised of two *valid* coins $(C_1, C_2)$ such that they reveal $g_1^x$. Since $g_1^x$ is secret information, its

release signals that the user must have committed *some* violation. The reason the user must be at fault for the *specific* violation at hand (and not some previous one) is that knowledge of $x$ (not simply $g_1^x$) is required to create valid coins.

The second part is that no adversary, even after forcing a user to violate a system condition, can produce $(i, S, \Gamma)$ such that $\mathsf{VerifyOwnership}(params, pk, S, \Gamma)$ unless the coin with serial number $S$ actually belongs to the user with $pk$. Recall that the proof $\Gamma$ is comprised of a verifiable encryption $E$, the user's signature $\sigma$, and a decryption key $g_1^x$. The soundness of the verifiable encryption and the signature scheme (for example, CL signatures [53]) is based on Strong RSA and the discrete logarithm assumption. Thus, with high probability, using $g_1^x$ to decrypt $E$ reveals some plaintext $s$ that in turn allows to compute all serial numbers belonging to that user. □

## 7.4 Scaling Back the Punishment for System Violators

In oral presentations of our previous results, some audience members felt that tracing *all* previous (and future) transactions of a user who might have accidentally violated one of the system conditions is too harsh. To make the punishment for system violators more lenient or simply to make the system more efficient, two other options are available:

*Option (1): violation is detected and user's identity is revealed.* This system operates as the above except that during the Withdraw protocol the user does not give the bank verifiable encryptions of her wallet secret $s$. Then later during the Deposit protocol, the bank may still detect the violation and identify the user, but will not be able to compute the serial numbers of other transactions involving this user.

*Option (2): violation is detected.* This system operates as the Option (1) system, except that during the Spend protocol, the user does not provide the merchant with the money-laundering check value $Y_j$. Then later during the Deposit protocol, the bank may still detect the violation, but will not be able to identify the user.

## 7.5 On Removing Random Oracles

Our discussion from Section 6.6 on removing random oracles differs slightly for this system. In addition to Fiat-Shamir, our proof of security also relies on modeling the output of the hash functions $H_T$ and $H_1$ used in Spend as random. Here, it is not necessary that a simulator or extractor have control over the outputs of these functions; rather we require that the discrete logarithm relationships between the outputs are unknown.

## 7.6 Contributions

In an electronic cash (e-cash) system, a user can withdraw coins from the bank, and then spend each coin anonymously and unlinkably. For some applications, it is desirable to set a

limit on the dollar amounts of anonymous transactions. For example, governments require that large transactions be reported for tax purposes. In this chapter, we present the first e-cash system that makes this possible without a trusted party. In our system, a user's anonymity is guaranteed so long as she does not: (1) double-spend a coin, or (2) exceed the publicly-known spending limit with any merchant. The spending limit may vary with the merchant, but a merchant's limit applies to all of its customers uniformly. Violation of either condition can be detected, and can (optionally) lead to identification of the user and discovery of her other activities. While it is possible to balance accountability and privacy this way using e-cash, this is impossible to do using regular cash.

Our scheme is based on the recent compact e-cash system of Camenisch et al., as described in Chapter 6. It is secure under the same complexity assumptions in the random-oracle model. We inherit its efficiency: $2^\ell$ coins can be stored in $O(\ell + k)$ bits and the complexity of the withdrawal and spend protocols is $O(\ell + k)$, where $k$ is the security parameter.

## 7.7 Open Problems

The results of this chapter open the door for the question: what other forbidden actions should result in a loss of anonymity? The goal would be *not* to introduce a TTP to achieve these features, and thus maintain the idea that an honest user's transactions are unlinkable and anonymous *to everyone*, while at the same time some necessary restrictions can be enforced.

The system in this chapter makes use of random oracles in its proof of security in more places than the previous chapter. Thus, it would be even more interesting to see if random oracles could be removed from an efficient scheme with bounded-anonymity.

# Appendix A

# Full Protocols to Spend Coins

To provide the full protocols to spend a coin, we need to provide some details about the signature scheme the bank uses.

Let $QR_n$ denote the set of quadratic residues modulo $n$. Let $\mathbf{Z}, \mathbf{U}, \mathbf{V}$ be elements of $QR_n$ that are part of the public key of the bank.

Let $\ell_n$ denote the number of bits of the bank's RSA modulus $n$ and let $\ell_\varnothing$, e.g., $\ell_\varnothing$ be a security parameter controlling the statistically zero knowledge property of the proof protocol as well as the statistically hiding property of the commitment schemes we use.

A signature of the bank on the seed (message) $s$ consists of the values $(\mathbf{Q}, e, v)$, where $e \in \{2^{\ell_e} - 2^{\ell'_e}, 2^{\ell_e} + 2^{\ell'_e}\}$ is a random prime, $v \in \{0,1\}^{\ell_n + \ell_\varnothing}$ is a random integer, and $\mathbf{Q} \in \langle \mathbf{U} \rangle \subset QR_n$, such that the following holds

$$\mathbf{Z} \equiv \mathbf{Q}^e \mathbf{V}^v \mathbf{U}^s \pmod{n} \ .$$

We note that the bank does not learn $s$ when issuing this signature. Instead, the bank and the user run an efficient two-party protocol where the output of the user will be $(\mathbf{Q}, e, v)$ [53].

In case the bank signs a block of messages, say $(u, s, t)$, at once, we replace $\mathbf{V}$ by public values $(\mathbf{V_1}, \mathbf{V_2}, \mathbf{V_3})$. The signature still consists of values $(\mathbf{Q}, e, v)$ such that

$$\mathbf{Z} \equiv \mathbf{Q}^e \mathbf{V_1}^u \mathbf{V_2}^s \mathbf{V_3}^t \mathbf{U}^s \pmod{n} \ .$$

To make our proof more efficient by roughly a factor of two, we will change the range of $J$, the coin counter. Our goal is to have a range that contains an odd number of elements. This is not strictly necessary but gives us a more efficient proof when $J$ lies in an odd interval. Thus, we need that $J_0 - J_1 \leq J \leq J_0 + J_1$ holds for some $J_0$ and $J_1$. For instance, setting $J_1 = 2^{\ell-1}$ and $J_0 = 2^\ell$ we have $2^{\ell-1} \leq J \leq 3 \cdot 2^{\ell-1}$ and thus can spend $2^\ell + 1$ coins. Note that now $J$ can no longer be 0 and a therefore the serial number $S$ can be computed as $S = g^{1/(J+s)}$.

## A.1 System One: Full Protocol

We now describe how the user spends a coin in System One in more detail. Recall that the user has obtained from the bank a signature $(\mathbf{Q}, e, v)$ on the "messages" $u$, $s$, and $t$.

Let $J$ be the number of the coin. By construction we have $J_0 - J_1 \leq J \leq J_0 + J_1$ and hence $0 \leq J_1^2 - (J - J_0)^2 = J_1^2 - J_0^2 + 2J_0J - J^2$ for all $J$. Thus, to show that a coin number $J$ lies in the required interval, we only need to show that for $J_1^2 - (J - J_0)^2$ is a positive number. Now, due to Lagrange, each positive number can be represented as the sum of four squares.

We are now ready to describe how a user spends one of her coins. we merged all the single proofs of $\Phi$ into one and get some efficiency improvements, e.g., the commitments $\mathbf{C}$ and $\mathbf{D}$ are not needed. Also, we choose $J$ differently and thus compute $S$ and $T$ in a slightly different manner.

1. $\mathcal{M}$ (optionally) sends a string *info* $\in \{0,1\}^*$ containing transaction information to $\mathcal{U}$ and/or identifies himself by proving knowledge of $sk_{\mathcal{M}}$ (We leave this step out.).
2. $\mathcal{M}$ chooses a random $R \in \mathbb{Z}_q$ such that $R \neq 0$ and sends $R$ to $\mathcal{U}$ (or it is a result of a hash function). The only property needed here is that $R$ has not previously been used by the merchant.
3. If $J > J_0 + J_1$, the user aborts as she has already sent all coins.
4. The user computes the serial number $S = g^{1/(J+s)}$ and a (now fixed) security tag $T = pk_{\mathcal{U}} \cdot g^{R/(J+t)}$
5. The user computes values $w_1$, $w_2$, $w_3$, and $w_4$ such that $\sum_i w_i = J_1^2 - (J - J_0)^2$ (e.g., using an algorithm by Rabin and Shallit [163]).
6. The user chooses random values $r_A$,$r_B$, $r_1$, $r_2$, $r_3$, and $r_4$ where each $r \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$, and computes the commitments $\mathbf{A} = \mathbf{g}^J\mathbf{h}^{r_A}$, $\mathbf{B} = \mathbf{g}^u\mathbf{h}^{r_B}$ and $\mathbf{W_i} = \mathbf{g}^{w_i}\mathbf{h}^{r_i}$ for $i = 1,\ldots,4$, The user chooses $r \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$ and computes $\mathbf{Q}' := \mathbf{Q}\mathbf{U}^r$. Note that $(\mathbf{Q}', e, v+r)$ is also a valid signature on the message $s$ but that $\mathbf{Q}$ and $\mathbf{Q}'$ are statistically independent.
7. The user computes the following signature proof of knowledge:

$$
\begin{aligned}
\Psi = SPK\{&(\alpha,\beta,\gamma,\delta,\varepsilon,\rho_1,\ldots,\rho_7):\\
&\mathbf{Z} = \pm\mathbf{Q}'^{\varepsilon}\mathbf{V_1}^{\mu}\mathbf{V_2}^{\sigma}\mathbf{V_3}^{\tau}\mathbf{U}^{\zeta} \;\wedge\; \mathbf{A} = \pm\mathbf{g}^{\gamma}\mathbf{h}^{\rho_A} \;\wedge\; g = S^{\sigma}S^{\gamma} \;\wedge\\
&\mathbf{W_i} = \pm\mathbf{g}^{\nu_i}\mathbf{h}^{\rho_i} \;\wedge\; \mathbf{g}^{J_1^2-J_0^2} = \pm(\mathbf{A}\mathbf{g}^{-2J_0})^{\gamma}\mathbf{W_1}^{\nu_1}\mathbf{W_2}^{\nu_2}\mathbf{W_3}^{\nu_3}\mathbf{W_4}^{\nu_4}\mathbf{h}^{\rho_5} \;\wedge\\
&\mathbf{B} = \mathbf{g}^{\mu}\mathbf{h}^{\rho_6} \;\wedge\; 1 = \mathbf{B}^{\sigma}\mathbf{B}^{\tau}(1/\mathbf{g})^{\alpha}\mathbf{h}^{\rho_7} \;\wedge\; g^R = T^{\sigma}T^{\tau}(g^R)^{\alpha} \;\wedge\\
&\sigma \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_{\mathcal{H}}+2} \;\wedge (\varepsilon - 2^{\ell_e}) \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_{\mathcal{H}}+1}\}\\
&(\mathbf{Z},\mathbf{V_1},\mathbf{V_2},\mathbf{V_3},\mathbf{U},\mathbf{g},\mathbf{h},n,g,S,T,\mathbf{A},\mathbf{B},\mathbf{W_1},\mathbf{W_2},\mathbf{W_3},\mathbf{W_4},\mathit{info})
\end{aligned}
$$

This is simply an optimized version of our previous proof of knowledge in Chapter 6.4. The first line of the proof (starting with $\mathbf{Z}$) is to verify the CL signature, the user's key, and the coin serial number $S$. The second line (starting with $\mathbf{W_i}$) is to verify that the coin wallet number is from the appropriate range of size $2^\ell$, i.e., when the $j$th coin is being spent from a wallet, $j$ is the coin wallet number. The third line (starting with $\mathbf{B}$ is to verify the double-spending equation $T$. The fourth line (starting with $\sigma$) is to verify the parameters are in the appropriate ranges. The final line (starting with $(\mathbf{Z}\ldots)$ contains the "message" being signed by this signature proof of knowledge.

8. If $\Phi$ verifies, $\mathcal{M}$ accepts the coin $(R, S, T, \Phi)$ and subsequently submits $(R, S, T, \Phi)$ to the bank $\mathcal{B}$. If $R$ is not already in $L$, then $\mathcal{B}$ accepts the coin and stores the values $(R, S, T)$ in a database of spent coins $L$. If $S$ was already in $L$, then $\mathcal{B}$ initiates the IdentifyViolator protocol.
9. $\mathcal{U}$ updates his counter $J = J + 1$ as before. When $J > 2^\ell - 1$, the wallet is empty.

We see that, to spend a coin, a user needs to compute 7 multi-base exponentiations to build the commitments and 11 multi-base exponentiations to carry out the proof. The merchant and bank need to do 11 multi-base exponentiations to spend a coin.

## A.2 System Two: Partial Protocol

The Spend protocol in System Two is more involved than that of System One. From the description in Chapter 6.5, it may not be clear how to implement step (f) of the Spend protocol. We now describe how to construct these signature proofs in more detail (although further optimizations are also possible).

Recall that in step (f) a user $\mathcal{U}$ must prove to a merchant $\mathcal{M}$ that the serial number $S$ and the security tag $T$ are *valid* elements of a coin. In particular, $\mathcal{U}$ is tasked with showing that $S = f^{DY}_{g_T,s}(J)$ and $T = g_1^u(f^V_{g_1,\vec{t}}(J))^R$. The essence of this proof is to show that $S$ and $T$ are using the same $J$ value.

More formally, we denote $\Gamma = \Gamma_0 \wedge \Gamma_1 \wedge \ldots \wedge \Gamma_{3\ell}$ as the signature proof of knowledge where $\Gamma_0$ is the following signature proof of knowledge:

$$\Gamma_0 = SPK\{(\alpha, \beta, \gamma, \delta_1, \delta_2, \delta_3) : \mathbf{g} = (\mathbf{AD})^\alpha \mathbf{h}^{\delta_1} \wedge S = g_T^\alpha \wedge \mathbf{C} = \mathbf{g}^\beta \mathbf{h}^{\delta_2} \wedge \mathbf{F_{3\ell}} = \mathbf{g}^\gamma \mathbf{h}^{\delta_3} \wedge T = g_1^\beta (g_1^R)^\gamma\}(S, T, \mathbf{A}, \{\mathbf{B_i}\}, \mathbf{C}, \mathbf{D}, \{\mathbf{E_i}\}, \{\mathbf{F_i}\}, \mathbf{g}, \mathbf{h}, n, g_1, g_2, g_T, pk_{\mathcal{M}}, R, info)$$

After this point in the proof we can consider the values $(u, s, \vec{t}, J)$ as fixed (although unknown to $\mathcal{M}$) and value $R$ as public, what remains to be shown is that $\mathbf{F_{3\ell}} = \mathbf{g}^\gamma \mathbf{h}^{\delta_3}$ where $f^V_{g_1,\vec{t}} = g_1^\gamma$ and the $\mathcal{U}$ knows $\gamma, \delta_3$. To show this, we successively build up to $\mathbf{F_{3\ell}}$ by showing that for each intermediate commitment $\mathbf{F_i}$, for $i = 1$ to $3\ell$, either: (1) the $i$th bit of $V(J)$ is zero and thus $\mathbf{F_i}$ and $\mathbf{F_{i-1}}$ open to the same value, or (2) the $i$th bit of $V(J)$ is one and thus $\mathbf{F_i}$ opens to the opening of $\mathbf{F_i}$ times $t_i$ (where $t_i$ is the corresponding secret in $\vec{t}$). The user already proved that each $\mathbf{B_i}$ commits to the $i$th bit of $V(J)$ in steps (1b) and (1c).

Then for $i = 1$ to $3\ell$, we have the signature proof of knowledge $\Gamma_i$:

$$\Gamma_i = SPK\{(\alpha_i, \beta_i, \phi_i, \{\gamma_i^{(n)}\}) : \left(\mathbf{F_i} = \mathbf{F_{i-1}}\mathbf{h}^{\gamma^{(2)}} \wedge \mathbf{B_i} = \mathbf{h}^{\gamma_i^{(3)}}\right) \vee \left(\mathbf{F_i} = \mathbf{F_{i-1}}^{\alpha_i}\mathbf{h}^{\gamma_i^{(4)}} \wedge \mathbf{E_i} = \mathbf{g}^{\alpha_i}\mathbf{h}^{\gamma_i^{(5)}} \wedge \mathbf{B_i} = \mathbf{g}\mathbf{h}^{\gamma_i^{(6)}}\right)\} (S, T, \mathbf{A}, \{\mathbf{B_i}\}, \mathbf{C}, \mathbf{D}, \{\mathbf{E_i}\}, \{\mathbf{F_i}\}, \mathbf{g}, \mathbf{h}, n, g_1, g_2, g_T, pk_{\mathcal{M}}, R, info).$$

# Bibliography


[1] Ben Adida and Douglas Wikström. Obfuscated ciphertext mixing, 2005. Cryptology ePrint Archive: Report 2005/394. `http://eprint.iacr.org/2005/394`.

[2] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, 2002.

[3] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT '02*, volume 2332 of LNCS, pages 83–107, 2002.

[4] ANSI X9.62 and FIPS 186-2. Elliptic Curve Digital Signature Algorithm (ECDSA), 1998.

[5] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):591–610, 2000.

[6] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles, 2005. Cryptology ePrint Archive: Report 2005/385. `http://eprint.iacr.org/2005/385`.

[7] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO '00*, volume 1880 of LNCS, pages 255–270, 2000.

[8] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In Dan Boneh and Dan Simon, editors, *Network and Distributed System Security Symposium (NDSS)*, pages 29–43, 2005.

[9] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. *ACM Transactions in Information and System Security (TISSEC)*, 9, No. 1:1–30, February, 2006.

[10] Giuseppe Ateniese and Susan Hohenberger. Proxy Re-Signatures: New Definitions, Algorithms, and Applications. In Catherine Meadows and Paul Syverson, editors, *12th ACM Conference on Computer and Communications Security (CCS)*, pages 310–319, 2005.

[11] Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. In David Naccache and Pascal Paillier, editors, *Public Key Cryptography (PKC)*, volume 2274 of LNCS, pages 80–98, 2002.

[12] Lucas Ballard, Matthew Green, Breno de Medeiros, and Fabian Monrose. Correlation-resistant storage. Technical Report TR-SP-BGMM-050705, Johns Hopkins University, Computer Science Department, 2005. `http://spar.isi.jhu.edu/~mgreen/correlation.pdf`.

[13] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO '01*, volume 2139 of LNCS, pages 1–18, 2001.

[14] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of LNCS, pages 480–494, 1997.

[15] Paulo S.L.M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In Moti Yung, editor, *Advances in Cryptology – CRYPTO '02*, volume 2442 of LNCS, pages 354–368, 2002.

[16] Paulo S.L.M. Barreto and Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography (SAC)*, volume 3897 of LNCS, pages 319–331, 2005.

[17] Mihir Bellare and Gregory Neven. Transitive Signatures Based on Factoring and RSA. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT '02*, volume 2501 of LNCS, pages 397–414, 2002.

[18] Mihir Bellare and Gregory Neven. Transitive signatures: New schemes and proofs. *IEEE Transactions on Information Theory*, 51(6):2133–2151, 2005.

[19] Mihir Bellare and Adriana Palacio. GQ and Schnorr Identification Schemes: Proofs of Security against Impersonation under Active and Concurrent Attacks. In Moti Yung, editor, *Advances in Cryptology – CRYPTO '02*, volume 2442 of LNCS, pages 162–177, 2002.

[20] Matt Blaze. A cryptographic file system for UNIX. In *ACM Conference on Computer and Communications Security*, pages 9–16, 1993.

[21] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of LNCS, pages 127–144, 1998.

[22] Matt Blaze and Martin Strauss. Atomic proxy cryptography. Technical report, AT&T Research, 1997.

[23] Alexandra Boldyreva. Efficient Threshold Signature, Multisignature, and Blind Signature Schemes based on the Gap-Diffie-Hellman-group signature Scheme. In Yvo Desmedt, editor, *Public Key Cryptography (PKC)*, volume 2567 of LNCS, pages 31–46, 2003.

[24] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of LNCS, pages 223–238, 2004.

[25] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of LNCS, pages 54–73, 2004.

[26] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of LNCS, pages 440–456, 2005.

[27] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures using strong Diffie-Hellman. In Matthew K. Franklin, editor, *Advances in Cryptology – CRYPTO '04*, volume 3152 of LNCS, pages 41–55, 2004.

[28] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of LNCS, pages 506–522, 2004.

[29] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil Pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[30] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO '01*, volume 2139 of LNCS, pages 213–229, 2001.

[31] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of LNCS, pages 416–432, 2003.

[32] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO '05*, volume 3621 of LNCS, pages 258–275, 2005.

[33] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In Joe Kilian, editor, *Theory of Cryptography (TCC)*, volume 3378 of LNCS, pages 325–341, 2005.

[34] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[35] Dan Boneh, Hovav Shacham, and Ben Lynn. Short signatures from the Weil pairing. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT '01*, volume 2248 of LNCS, pages 514–532, 2001.

[36] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. In *Topics in Algebraic and Noncommutative Geometry, Contemporary Mathematics*, volume 324, pages 71–90. American Mathematical Society, 2003.

[37] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT '00*, volume 1807 of LNCS, pages 431–444, 2000.

[38] Joan Boyar, David Chaum, Ivan Damgård, and Torben P. Pedersen. Convertible undeniable signatures. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 537 of LNCS, pages 189–205, 1990.

[39] Xavier Boyen and Brent Waters. Compact group signatures without random oracles. In *Advances in Cryptology – EUROCRYPT '06*, (to appear), 2006.

[40] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI, April 1993.

[41] Stefan Brands. Untraceable off-line cash in wallets with observers. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773, pages 302–318, 1993 of LNCS.

[42] Stefan Brands. Rapid demonstration of linear relations connected by boolean operators. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of LNCS, pages 318–333, 1997.

[43] Stefan Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy.* PhD thesis, Eindhoven Inst. of Tech. The Netherlands, 1999.

[44] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A simple public-key cryptosystem. In Chi-Sung Laih, editor, *Advances in Cryptology – ASIACRYPT '03*, volume 2894 of LNCS, pages 37–54, 2003.

[45] Friederike Brezing and Annegret Weng. Elliptic Curves Suitable for Pairing Based Cryptography. *Designs, Codes and Cryptography*, 37(1):133–141, 2005.

[46] Ernie Brickell, Peter Gemmel, and David Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Symposium on Discrete Algorithms (SODA)*, pages 457–466, 1995.

[47] Jan Camenisch and Ivan Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 of LNCS, pages 331–345, 2000.

[48] Jan Camenisch and Jens Groth. Group signatures: Better efficiency and new theoretical aspects. In Carlo Blundo and Stelvio Cimato, editors, *Security in Communication Networks (SCN)*, volume 3352 of LNCS, pages 120–133, 2004.

[49] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact E-Cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of LNCS, pages 302–321, 2005.

[50] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing Accountability and Privacy Using E-Cash, 2006. Manuscript.

[51] Jan Camenisch, Susan Hohenberger, Anna Lysyanskaya, Markulf Kohlweiss, and Mira Meyerovich. How to Win the Clone Wars: Efficient Periodic $n$-Times Anonymous Authentication, 2006. Manuscript.

[52] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *Advances in Cryptology – CRYPTO '02*, volume 2442 of LNCS, pages 61–76, 2002.

[53] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks (SCN)*, volume 2576 of LNCS, pages 268–289, 2002.

[54] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew K. Franklin, editor, *Advances in Cryptology – CRYPTO '04*, volume 3152 of LNCS, pages 56–72, 2004.

[55] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number $n$ is the product of two safe primes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of LNCS, pages 107–122, 1999.

[56] Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of LNCS, pages 413–430, 1999.

[57] Jan Camenisch, Jean-Marc Piveteau, and Markus Stadler. Blind signatures based on the discrete logarithm problem. In Alfredo de Santis, editor, *Advances in Cryptology – EUROCRYPT '94*, volume 950 of LNCS, pages 428–432, 1994.

[58] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO '03*, volume 2729 of LNCS, pages 126–144, 2003.

[59] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1296 of LNCS, pages 410–424, 1997.

[60] Jan Leonhard Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem.* PhD thesis, ETH Zürich, 1998.

[61] Ran Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute of Science, Rehovot 76100, Israel, June 1995.

[62] Ran Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of LNCS, pages 455–469, 1997.

[63] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[64] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[65] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO '01*, volume 2139 of LNCS, pages 19–40, 2001.

[66] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 209–218, 1998.

[67] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT '03*, volume 2656 of LNCS, pages 255–271, 2003.

[68] Certicom Inc. Ecc cryptography tutorial, 2006. Available at `http://www.certicom.com/index.php?action=ecc_tutorial,ecc_tut_3_2`.

[69] Herve Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of LNCS, pages 542–558, 2005.

[70] Agnes Chan, Yair Frankel, and Yiannis Tsiounis. Easy come – easy go divisible cash. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of LNCS, pages 561–575, 1998.

[71] Denis Charles. On The Existence of Distortion Maps on Ordinary Elliptic Curves, 2006. Cryptology ePrint Archive: Report 2006/128. `http://eprint.iacr.org/2006/128`.

[72] Melissa Chase and Anna Lysyanskaya. On Signatures of Knowledge. In *Advances in Cryptology - CRYPTO '06*, (to appear),2006.

[73] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology - CRYPTO '82*, pages 199–203. Plenum Press, 1982.

[74] David Chaum. Blind signature systems. In David Chaum, editor, *Advances in Cryptology - CRYPTO '83*, pages 153–156. Plenum Press, 1983.

[75] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[76] David Chaum. Online cash checks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89*, volume 434 of LNCS, pages 289–293, 1989.

[77] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology - CRYPTO '90*, volume 403 of LNCS, pages 319–327, 1988.

[78] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology - CRYPTO '92*, volume 740 of LNCS, pages 89–105, 1992.

[79] David Chaum and Torben Pryds Pedersen. Transferred cash grows in size. In Rainer A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92*, volume 658 of LNCS, pages 390–407, 1993.

[80] David Chaum and Hans Van Antwerpen. Undeniable signatures. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, volume 435 of LNCS, pages 212–216, 1989.

[81] Jung Hee Cheon and Dong Hoon Lee. Diffie-Hellman problems and bilinear maps, 2002. Cryptology ePrint Archive: Report 2002/117. `http://eprint.iacr.org/2001/117`.

[82] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *8th IMA International Conference on Cryptography and Coding*, volume 2260 of LNCS, pages 360–363, 2001.

[83] Clifford Cocks and Richard Pinch. Identity-based cryptosystems based on the Weil Pairing. In I.F. Blake, G. Seroussi, and N.P. Smart, editors, *Advances in Elliptic Curve Cryptography*, chapter 9. Cambridge University Press, 2005.

[84] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, volume 839 of LNCS, pages 174–187, 1994.

[85] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT '02*, volume 2332 of LNCS, pages 45–64, 2002. Full and revised version available at `http://www.shoup.net`.

[86] Ivan Damgård and Eiichiro Fujisaki. An integer commitment scheme based on groups with hidden order. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT '02*, volume 2501 of LNCS, 2002.

[87] Alexander W. Dent. Adapting the weaknesses of the random oracle model to the generic group model. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT '02*, volume 2501 of LNCS, pages 100–109, 2002.

[88] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, November 1976.

[89] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *Public Key Cryptography (PKC)*, volume 2567 of LNCS, pages 1–17, 2003.

[90] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. Intrusion-resilient public-key encryption. In Marc Joye, editor, *Cryptographers' Track at the RSA Conference (CT-RSA)*, volume 2612 of LNCS, pages 19–32, 2003.

[91] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. A generic construction for intrusion-resilient public-key encryption. In Tatsuaki Okamoto, editor, *Cryptographers' Track at the RSA Conference (CT-RSA)*, volume 2964 of LNCS, pages 81–98, 2004.

[92] Yevgeniy Dodis and Anca Ivan. Proxy cryptography revisited. In *Network and Distributed System Security Symposium*, February 2003.

[93] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT '02*, volume 2332 of LNCS, pages 65–82, 2002.

[94] Yevgeniy Dodis and Leonid Reyzin. Breaking and repairing optimistic fair exchange from PODC 2003. In Moti Yung, editor, *ACM Workshop on Digital Rights Management (DRM)*, pages 47–54, 2003.

[95] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In Serge Vaudenay, editor, *Public Key Cryptography (PKC)*, volume 3386 of LNCS, pages 416–431, 2005.

[96] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *Proc. 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 542–552, 1991.

[97] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In George R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of LNCS, pages 10–18, 1984.

[98] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of LNCS, pages 186–194, 1986.

[99] Yair Frankel, Yiannis Tsiounis, and Moti Yung. Indirect discourse proofs: Achieving efficient fair off-line e-cash. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT '96*, volume 1163 of LNCS, pages 286–300, 1996.

[100] Matthew Franklin and Moti Yung. Towards provably secure efficient electronic cash. In Andrzej Lingas, Rolf G. Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming, International Colloquium (ICALP)*, volume 700 of LNCS, pages 265–276, 1993.

[101] David Freeman. Constructing Pairing-Friendly Elliptic Curves with Embedding Degree 10. In *Algorithmic Number Theory Symposium (ANTS) VII (to appear)*, 2006.

[102] Kevin Fu. Group sharing and random access in cryptographic storage file systems. Master's thesis, Massachusetts Institute of Technology, May 1999.

[103] Kevin Fu. *Integrity and access control in untrusted content distribution networks.* PhD thesis, Massachusetts Institute of Technology, September 2005.

[104] Kevin Fu, M. Frans Kaashoek, and David Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Systems*, 20(1):1–24, 2002.

[105] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of LNCS, pages 16–30, 1997.

[106] Steven Galbraith. Elliptic curve cryptography, 2006. `http://www.isg.rhul.ac.uk/~sdg/ecc.html`.

[107] Steven D. Galbraith. Supersingular curves in cryptography. In Colin Boyd, editor, *Advances in Cryptology – ASIACRYPT '01*, volume 2248 of LNCS, pages 495–513, 2001.

[108] Steven D. Galbraith, Keith Harrison, and David Soldera. Implementing the Tate Pairing. In Claus Fieker and David R. Kohel, editors, *Algorithmic Number Theory (ANTS) '02*, volume 2369 of LNCS, pages 324–337, 2002.

[109] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers, 2006. `http://eprint.iacr.org/2006/165.pdf`.

[110] Steven D. Galbraith and Victor Rotger. Easy Decision-Diffie-Hellman Groups. *LMS Journal of Computation and Mathematics*, 7:201–218, 2004.

[111] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: Securing remote untrusted storage. In *Network and Distributed System Security Symposium*, pages 131–145, February 2003.

[112] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a method of cryptographic protocol design. In *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 174–187. IEEE Computer Society Press, 1986.

[113] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[114] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of LNCS, pages 171–185, 1987.

[115] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 102–115. IEEE Computer Society Press, 2003.

[116] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *Foundations of Computer Science (FOCS)*, pages 553–562, 2005.

[117] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[118] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, 17(2):281–308, 1988.

[119] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. Universal Reencryption for Mixnets. In Tatsuaki Okamoto, editor, *Cryptographers' Track at the RSA Conference (CT-RSA)*, volume 2964 of LNCS, pages 163–178, 2004.

[120] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *Advances in Cryptology – EUROCRYPT '06*, (to appear), 2006.

[121] Anthony Harrington and Christian Jensen. Cryptographic access control in a distributed file system. In *8th ACM Symposium on Access Control Models and Technologies (SACMAT 2003)*, pages 158–165, 2003.

[122] Susan Hohenberger. The Cryptographic Impact of Groups with Infeasible Inversion, S.M. Thesis, Massachusetts Institute of Technology, May 2003.

[123] Markus Jakobsson. On quorum controlled asymmetric proxy re-encryption. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography (PKC)*, volume 1560 of LNCS, pages 112–121, 1999.

[124] Stanislaw Jarecki and Vitaly Shmatikov. Handcuffing big brother: an abuse-resilient transaction escrow scheme. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of LNCS, pages 590–608, 2004.

[125] Stanislaw Jarecki and Vitaly Shmatikov. Probabilistic escrow of financial transactions with cumulative threshold disclosure. In Andrew S. Patrick and Moti Yung, editors, *Financial Cryptography (FC)*, volume 3570 of LNCS, pages 172–187, 2005.

[126] Antoine Joux. A one-round protocol for tripartite Diffie-Hellman. In Wieb Bosma, editor, *Algorithmic Number Theory Symposium (ANTS) IV*, volume 1838 of LNCS, pages 385–394, 2000.

[127] Antoine Joux and Kim Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups. *Journal of Cryptology*, 16(4):239–247, 2003. First appeared as Cryptology ePrint Archive: Report 2001/003.

[128] Ari Juels, David Molnar, and David Wagner. Security and privacy issues in e-passports, 2005. Cryptology ePrint Archive: Report 2005/095. `http://eprint.iacr.org/2005/095`.

[129] David Kahn. *The Codebreakers: The Story of Secret Writing*. The Macmillan Company, New York, 1967.

[130] Burt Kaliski. TWIRL and RSA Key Size, May 6, 2003. Available at `http://www.rsasecurity.com/rsalabs/node.asp?id=2004#table1`.

[131] Mahesh Kallahalla, Erik Riedel, Ram Swaminathan, Qian Wang, and Kevin Fu. Plutus – scalable secure file sharing on untrusted storage. In *FAST Conference on File and Storage Technologies*, 2003.

[132] Dennis Kügler and Holger Vogt. Fair tracing without trustees. In Paul F. Syverson, editor, *Financial Cryptography '01*, volume 2339 of LNCS, pages 136–148, 2001.

[133] Aggelos Kiayias, Yiannis Tsiounis, and Moti Yung. Traceable signatures. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of LNCS, pages 571–589, 2004.

[134] Lea Kissner and David Molnar. Provably Secure Substitution of Cryptographic Tools, 2006. Cryptology ePrint Archive: Report 2006/004. `http://eprint.iacr.org/2006/004`.

[135] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[136] Moses Liskov. Amortized e-cash. Master's thesis, Massachusetts Institute of Technology, February 2001.

[137] Moses Liskov and Silvio Micali. Amortized e-cash. In Paul F. Syverson, editor, *Financial Cryptography (FC) '01*, volume 2339 of LNCS, pages 1–20, 2001.

[138] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *Advances in Cryptology – EUROCRYPT '06*, (to appear), 2006.

[139] Anna Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[140] Anna Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In Moti Yung, editor, *CRYPTO 2002*, LNCS, pages 597–612. Springer Verlag, 2002.

[141] Masahiro Mambo and Eiji Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. *IEICE Trans. Fund. Electronics Communications and Computer Science*, E80-A/1:54–63, 1997.

[142] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In *ACM Conference on Computer and Communications Security (CCS)*, pages 48–57, 1996.

[143] Noel McCullagh and Paulo S. L. M. Barreto. A new two-party identity-based authenticated key agreement. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA*, volume 3376 of LNCS, pages 262–274, 2004.

[144] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *ACM Communication and Computer Security 2001*, pages 245–54. ACM press, 2001.

[145] Silvio Micali and Ronald L. Rivest. Micropayments revisited. In Bart Preneel, editor, *Cryptographer's Track at the RSA Conference (CT-RSA)*, volume 2271 of LNCS, pages 149–163, 2002.

[146] Victor S. Miller. Use of Elliptic Curves in Cryptography. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218 of LNCS, pages 417–426, 1985.

[147] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Trans. Fundamentals*, E84-A(5):1234–1243, 2001.

[148] David Molnar. Homomorphic Signature Schemes, Honors Thesis, Harvard College, April 2003.

[149] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM*, 51, Number 2:231–262, 2004.

[150] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 427–437, Baltimore, Maryland, 1990. ACM.

[151] National Security Agency. The Case for Elliptic Curve Cryptography, Accessed on April 11, 2006. `http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm`.

[152] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55:165–172, 1994.

[153] Lan Nguyen and Rei Safavi-Naini. Dynamic $k$-times anonymous authentication. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security (ACNS)*, volume 3531 of LNCS, pages 318–333, 2005.

[154] NIST. Special Publication: 800-57: Recommendation for Key Management, January 2003. Available at `http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html`.

[155] Kazuo Ohta and Tatsuaki Okamoto. Multisignature schemes secure against active insider attacks. *IEICE Trans. Fundamentals*, E82-A/1:21–31, 1999.

[156] Tatsuaki Okamoto. An efficient divisible electronic cash scheme. In Don Coppersmith, editor, *Advances in Cryptology – CRYPTO '95*, volume 963 of LNCS, pages 438–451, 1995.

[157] Tatsuaki Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Trans. Computer Systems*, 6(4):432–441, 1998.

[158] Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 435 of LNCS, pages 481–496, 1990.

[159] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of LNCS, pages 324–337, 1991.

[160] Pascal Paillier. Public key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, volume 1592 of LNCS, pages 223–238, 1999.

[161] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO '92*, volume 576 of LNCS, pages 129–140, 1992.

[162] Birgit Pfitzmann and Michael Waidner. Formal Aspects of Fail-Stop Signatures. Interner Bericht 22/90, Fakultät für Informatik, Universität Karlsruhe, 1990.

[163] Michael O. Rabin and Jeffery O. Shallit. Randomized algorithms in number theory. *Communications on Pure and Applied Mathematics*, 39:239–256, 1986.

[164] David Reed and Liba Svobodova. Swallow: A distributed data storage system for a local network. In A. West and P. Janson, editors, *Local Networks for Computer Communications*, pages 355–373. North-Holland Publishing Company, 1981.

[165] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[166] Ahmad-Reza Sadeghi and Michael Steiner. Assumptions related to discrete logarithms: Why subtleties make a real difference. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT '01*, volume 2045 of LNCS, pages 244–262, 2001.

[167] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of LNCS, pages 457–473, 2005.

[168] Tomas Sander and Ammon Ta-Shma. Flow control: a new approach for anonymity control in electronic cash systems. In Matt Franklin, editor, *Proc. Third International Conference on Financial Cryptography '99*, volume 1648 of LNCS, pages 46–61, 1999.

[169] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[170] Michael Scott. MIRACL library. Indigo Software. `http://indigo.ie/~mscott/#download`.

[171] Mike Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number, 2002. Cryptology ePrint Archive: Report 2002/164. `http://eprint.iacr.org/2002/164`.

[172] Adi Shamir. Identity-based cryptosystems and signature schemes. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of LNCS, pages 47–53, 1985.

[173] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of LNCS, pages 256–266, 1997.

[174] Tony Smith. DVD Jon: buy DRM-less Tracks from Apple iTunes, March 18, 2005. Available at `http://www.theregister.co.uk/2005/03/18/itunes_pymusique`.

[175] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology - EUROCRYPT '95*, volume 921 of LNCS, pages 209–219, 1995.

[176] Isamu Teranishi, Jun Furukawa, and Kazue Sako. $k$-times anonymous authentication (extended abstract). In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT '04*, volume 3329 of LNCS, pages 308–322, 2004.

[177] Isamu Teranishi and Kazue Sako. $k$-times anonymous authentication with a constant proving cost. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography (PKC)*, volume 3958 of LNCS, pages 525–542, 2006.

[178] Yiannis S. Tsiounis. *Efficient Electonic Cash: New Notions and Techniques.* PhD thesis, Northeastern University, Boston, Massachusetts, 1997.

[179] Eric R. Verheul. Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT '01*, volume 2045 of LNCS, pages 195–201, 2001.

[180] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT '05*, volume 3494 of LNCS, pages 114–127, 2005.

[181] Hoeteck Wee. On obfuscating point functions. In *Symposium on Theory of Computing (STOC)*, pages 523–532, 2005.

[182] Yuliang Zheng. Signcryption and its applications in efficient public key solutions. In Eiji Okamoto, George Davida, and Masahiro Mambo, editors, *Information Security Workshop (ISW)*, volume 1396 of LNCS, pages 291–312, 1997.

[183] Lidong Zhou, Michael A. Marsh, Fred B. Schneider, and Anna Redz. Distributed blinding for El Gamal re-encryption. Technical Report 2004–1924, Cornell Computer Science Department, 2004.