Abstract of "Sharing Secrets for Fun and Profit" by Mira Belenkiy, Ph.D., Brown University, May 2008.

Electronic cash is an important tool for preserving on-line privacy. It allows a user to make purchases without revealing his identity to the merchant and prevents banks from monitoring the transactions of all their users. In this thesis, we use secret sharing techniques to extend electronic cash.

We examine the problem of fair exchange that lets a user atomically exchange an electronic coin for some digital good or service. While all fair exchange protocols require the help of a trusted third party, in an optimistic protocol, the trusted party becomes involved only if something goes wrong. We construct the first optimistic fair exchange protocol for exchanging a single electronic coin for a digital good or service. We use secret sharing to extend the protocol to allow a user to efficiently exchange multiple electronic coins for multiple goods and services in one atomic transaction. We also show how a user can pay a single electronic coin for multiple goods or services, which need not be delivered atomically.

We apply these fair exchange protocols to create incentives in anonymous peer-to-peer filesharing systems. We show how to perform an efficient fair exchange for very large files (such as movies); the trusted third party can resolve problems without downloading the entire file. We also show how to escrow electronic coins to allow peers to barter for files. If one of the peers fails to deliver, he acquires the escrowed coin.

Electronic cash can be used for anonymous authentication. Instead of purchasing a good or service, the user may use electronic coins to access restricted resources. For example, the user may purchase a license to download twenty songs a month from a provider. We show how to use secret sharing to create flexible policies. In the case where the electronic coins are stored on a small hardware device, the extra flexibility may be used to provide glitch protection in case the user accidentally spends a few extra coins.

Finally, we end by developing a new secret sharing protocol for a disjunctive multi-level access structure.

Sharing Secrets for Fun and Profit

by
Mira Belenkiy
B. Sc. (Computer Science) Brandeis University
M. Sc. (Computer Science) Brown University

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2008

This dissertation by Mira Belenkiy is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Prof. Anna Lysyanskaya, Director

Recommended to the Graduate Council

Date _____ _____

Dr. Jan Camenisch, Reader
IBM Research Laboratory, Zurich

Date _____ _____

Prof. Roberto Tamassia, Reader
Brown University

Date _____ _____

Prof. John Jannotti, Reader
Brown University

Approved by the Graduate Council

Date _____ _____

Sheila Bonde
Dean of the Graduate School

Vita

Mira Belenkiy was born in Moscow, Russia in 1981. She received a Bachelor of Science degree with Highest Honors in Computer Science and Mathematics from Brandeis University in 2003, and a Master of Science degree in Computer Science from Brown University in 2005. She completed her Doctorate in Computer Science at Brown University in 2008; her thesis examined applications of secret sharing to electronic cash and fair exchange. Her research focuses on various types of anonymous credentials. She has been awarded a Department of Homeland Security Fellowhip in 2004.

Acknowledgements

My time at Brown would not have been the same without the support of my colleagues, family, and friends.

My biggest thanks go to my advisor Anna Lysyanskaya. She has been a wonderful mentor, who taught me both the technical and the creative skills of successful research. I am grateful for her guidance and encouragement to tackle challenges that I would have never had the courage to take on alone.

I am grateful to John Jannotti for our many conversations, which have given me a new perspective on life and on research. He has taught me the value of considering the simpler approach first. I also want to thank Roberto Tamassia for his help and career advice. Many thanks to Dude Neergaard at Oak Ridge National Laboratory who made me feel welcome in Tennessee and showed me the empirical side of research.

I owe a lot to my wonderful co-authors. Markulf Kohlweiss forced me to examine my unconscious assumptions and defend my deductions. Melissa Chase taught me never to give up on a proof. Jan Camenisch has shown me that nothing is impossible. Working with them has made me a better researcher.

The Brownie group has shown me the value of looking beyond the math and to the practical and social applications of research. I will always remember our weekly discussions on game theory, incentives, and social networks. My sincere thanks to Alptekin Kupcu, Chris Erway, and Eric Rachlin.

These five years would not have been the same without my friends. Jenine Turner, William Headden, Dan Grollman, David McClosky, Nina Levetin, and Ariel Bender who were always ready to share a joke or a bowl of ice-cream.

Finally, I want to thank my family, without whom I could have never gotten this far. My husband Ilya Belenkiy cheered me on when things went well and supported me when they did not. My parents, Alexander and Laura Meyerovich, and my brother Leo Meyerovich have always been there for me and encouraged me every step of the way.

My work at Brown has been supported by a Department of Homeland Security Fellowship and National Science Foundation research grants.

Contents

Chapter 1

Introduction

1.1 Introduction

Privacy is almost non-existent on the Internet today. To complete a simple economic transaction, such as buying a book, a user reveals so much information about himself that the merchant can steal the user's identity! Indeed, identity theft and credit card fraud are a major security threat facing the world today.

Electronic cash (e-cash), invented by Chaum [36, 37], is an important tool that can help restore on-line privacy. E-cash allows a user to withdraw money from a bank and then spend it anonymously. Since merchants never learn any information about their customers, they cannot steal their identity. A user's privacy even extends to the bank. Even if the bank colludes with the merchants, it will be unable to link any e-coins to an honest user (or even check if two e-coins correspond to the same anonymous user). The only exception is if a user double-spends his e-coins (i.e. forges new money); in this case, the bank will be able to identify the user and link him to all the e-coins he has spent. The bank can even revoke users. E-cash has been studied extensively. We give an overview in Section 2.4.

This work studies ways in which e-cash can be used to solve real-world problems. We show that secret sharing techniques often play an important role in extending the functionality of e-cash. Secret sharing is a fundamental cryptographic primitive. A dealer creates shares of a secret and gives each share to a user. A group of *authorized* users should be able to reconstruct the secret. A group of *unauthorized* users, i.e. a group of users that does not contain a subgroup of authorized users, should learn no information about the secret. Section 2.3 gives an overview of some results in secret sharing, as they relate to this work.

We address a wide variety of problems. We develop a new type of e-cash, called endorsed e-cash, that can be fairly exchanged for electronic goods and services. We use endorsed e-cash to simultaneously pay multiple e-coins. We also show how it can be used to provide incentives in peer-to-peer systems, such as anonymous remailers and file sharing networks. A variation of e-cash can be used for anonymous authentication. Finally, we develop new techniques in secret sharing that might be useful for constructing divisible e-cash.

Thesis. *Secret sharing techniques are useful for efficiently extending the privacy preserving properties of electronic cash.*

The work presented in this dissertation comes from "Endorsed E-Cash," IEEE Security and Privacy 2007 [29], "How to win the clone wars: efficient periodic n-times anonymous authentication," ACM CCS 2006 [23], "Making P2P Accountable without Losing Privacy," WPES 2007 [10], and "Disjunctive Multi-Level Secret Sharing," [75].

1.2 Buying Digital Content

E-cash was originally invented to let users securely and privately purchase content on-line.[1] It protects the anonymity of a user who does not trust the merchant and bank with his private data. Careful attention is paid to protecting the merchant and bank from users who try to forge e-coins. Despite this environment of mutual mistrust, traditional e-cash Spend protocols do not ensure a *fair exchange* of e-cash for content.

A fair exchange protocol involves two parties. Each party has some content that the other one wants. At the end of the protocol, either both parties should have acquired the content it wants, or neither of them should have learned anything about the other party's data. All fair exchange protocols must involve a trusted third party (TTP). In an optimistic fair exchange [76, 5], the TTP gets involved only if one of the parties claims something went wrong.

Most fair exchange literature focuses on exchanging digital signatures and not on e-cash, but there are a few exceptions. Asokan, Shoup and Waidener [5] apply their optimistic fair exchange scheme for digital signatures to Brand's e-cash [20]. The result compromises the privacy of the user. If the exchange fails and the user ever tries to reuse the same e-coin, the bank can link it to the failed exchange. The only other fair exchange schemes for e-cash in the literature are due to Jakobsson's [61] and Reiter, Wang and Wright's [86]. These two schemes are even less fair to the user: if the user tries to reuse an e-coin from a failed exchanged, he is guilty of double-spending and his identity is revealed. See Section 2.4.3 for a full survey of the literature.

We solve the fair exchange problem for e-cash. We split every e-coin into two parts, the unendorsed e-coin *coin* and an endorsement x. The user gives the merchant *coin* and then performs a fair exchange of the digital content for x. The unendorsed e-coin is an encrypted version of the e-coin and cannot be spent without x. If the fair exchange fails, the user can generate a new unendorsed e-coin from the same coin with a different endorsement x'. The two unendorsed e-coins cannot be linked to each other, even if the user eventually endorses one of them. The user can make an exponential number of promises of the same e-coin, as long as he only endorses one.

Most goods and services cost more than one coin. The user wants to execute a single fair exchange, at the end of which the merchant either gets *all* of the e-coins or *none*. We solve this problem using a technique from threshold secret sharing (see Section 2.3.3). Suppose an item costs n e-coins. The user creates n unendorsed e-coins. Then the user takes the n endorsements, and creates a polynomial f of degree $n-1$ such that $f(i) = x_i$, where x_i is the ith endorsement. This can be done using straightforward Lagrange interpolation (see Section 2.1.3). Then the user gives the merchant the n unendorsed e-coins and $n-1$ *different* points on the polynomial $f(n+1), \ldots, f(2n-1)$. The merchant takes advantage of the fact that endorsements are really openings of Pedersen commitments to efficiently verify that the points on f are consistent with the n endorsements. Once the merchant is satisfied, the user and merchant perform a fair exchange of *a single* endorsement for the digital content. Once the merchant learns one endorsement, he can interpolate f to learn all of the other

[1] Every on-line good or service can be reduced to some sort of digital content, whether it is an e-book, verifiable computation, or signed receipt for a physical item.

endorsements and deposit all n e-coins. This protocol can be extended to the situation when the merchant also has multiple goods or services he is selling.

We describe endorsed e-cash in Chapter 4 for more details.

1.3 Incentives in Peer-To-Peer Networks

Peer-to-peer networks are designed to provide services to its users anonymously. As long as all users contribute their fair share to the network, a peer-to-peer system can be more efficient than a centralized system. However, many users would rather consume services and provide nothing in return. We can use endorsed e-cash to efficiently provide incentives to selfish users without compromising privacy. Our results can be found in Chapter 5 We present several examples of how we can provide accountability in peer-to-peer networks without compromising privacy:

Onion routing. Anonymous remailers, invented by David Chaum [35], also known as onion routers, let users communicate anonymously, even in the face of traffic analysis. A user chooses a sequence of routers to relay a message. Then, the user encrypts the message in many layers of encryption; each router peels off a single layer to learn the forwarding address of the next router in the chain. However, most peers in the network are selfish and prefer to send only their own messages. Users needs a way to anonymously provide routers with an incentive to forward messages.

The naive solution would be to include an e-coin in each encryption layer of the message. This will not work because a router can deposit the e-coin without forwarding the message. Suppose we include an unendorsed e-coin in each layer and make the router contact the next router in the chain in order to get the endorsement. This also will not work because the next router in the chain has no incentive to talk to the previous router.

We construct threshold endorsed e-cash to solve the onion routing problem. Threshold endorsed e-cash lets the user give away shares of the endorsement; if a merchant collects m shares, where m is set by the user, the merchant can calculate the endorsement and deposit the e-coin. See Section 4.2 for details. Suppose each router, upon peeling off a layer of encryption, sees (1) a threshold unendorsed e-coin, (2) the endorsement to the previous router's e-coin, and (3) the endorsement to the next router's e-coin. In order to collect its own e-coin, the router must contact the previous and next routers in the chain. The router performs a fair exchange with the previous router in the chain. Then the router must forward the onion to the next router in the chain, so that it can perform a fair exchange of endorsements with the next router. As a result, a router only gets paid if it forwards the message.

File sharing. One of the most popular application of peer-to-peer systems is file sharing. Initially, file sharing systems relied on the altruism of users. Each user who joined the system indicated what files he was willing to share. Any interested peer could connect to the user's computer and download the files. These altruistic file sharing networks were very slow in practice because users would hack their clients to avoid sharing files with other peers. Instead, they would use their entire bandwith to download the files *they* wanted.

BitTorrent [40] proposed the first practical solution to the problem of selfish users. It provided incentives for users to share files: if a user wanted to download a file from another user, he would have to offer a file in return. If the user sent the file too slowly, the other user would either decrease *his* file transfer rate, or cut off the slow user completely. Due to this incentive systems, BitTorrent downloads are very fast and BitTorrent is one of the most popular file sharing systems today.

The problem with BitTorrent is that it provides non-fungible incentives. Suppose Alice has a file that Bob wants, Bob has a file that Charlie wants and Charlie has a file that Alice wants. In the BitTorrent model, all three users are stuck. Alice will not talk to Bob because he has nothing for her, Bob won't talk to Charlie, and Charlie won't talk to Alice. BitTorrent resolves this dilemma by encouraging users to altruistically give away bandwith once they have finished their own download.

We can use e-cash to provide fungible incentives for file sharing. There are many problems associated with using e-cash in peer-to-peer networks. Some are pure policy issues: how do users join the system, are variable prices allowed, how does the system handle inflation/deflation, etc. In this work, we are mostly concerned with the cryptographic problems. We have developed a protocol that lets Alice sell a her file to Bob. We use endored e-cash as a building block for the fair exchange protocol. Our new protocol lightens the burden on the TTP; the TTP never has to download the entire file in order to verify that Alice is honest. We also show how Alice and Bob can trade files by placing endorsed e-coins into escrow. Alice and Bob can perform this step once and then choose to engage an arbitrary number of transactions with each other.

Other Distributed Systems. Peer-to-peer systems can provide a variety of different services, including distributed computation, distributed storage, and distributed look-up. Endorsed e-cash makes it possible to pay a service provider conditional on the performance of some well defined service.

1.4 Anonymous Authentication

E-cash can be used for anonymous authentication. Instead of withdrawing a wallet of e-coins and using them to purchase goods and services, a user can withdraw an e-token dispenser, and use the e-tokens to authenticate with verification authorities. In Section 6.1, we show how to slightly modify the dispenser to create new e-tokens every time period. For example, with regular e-cash, the user could withdraw a dispenser that lets him authenticate ten times, while with this modification, the dispenser lets him authenticate ten times per day.

We can apply secret sharing techniques to protect users who accidentally clone their e-tokens. (For example, an honest user's computer may crash and erase the fact that a particular e-token was already spent). In standard e-cash, every time a user spends an e-coin, he generates a polynomial f of degree 1 (i.e. a straight line) such that $f(0) = id$, his identity. The user provides the merchant with a single point randomly chosen on that line. If the user spends the same e-coin more than once, he reveals two points, and the merchant or bank can interpolate the line to learn the user's identity. Since every e-coin has a line with a different slope, an honest user who spends different e-coins each

time does not have to worry about compromising his anonymity.

In Section 6.2, we extend this technique to protect users who occasionally reuse e-tokens. Recall that we allow users to authenticate n times per time period. Each time a user reuses an e-token, we call the extra e-token a clone. We group the time periods into non-overlapping time intervals. We preserve the anonymity of users who create less than m clones per time interval. Each time a user shows an e-token, he gives the verifier a point on two polynomials, K and E. Polynomial E is of degree $m-1$, such that $E(0) = id$; it is unique to each user and time interval. The polynomial K is of degree 1. It is unique to each e-token. If the user reuses an e-token, the verifier can interpolate K and learn a link-id. The link-id lets the verifier group clones made by the same user in the same time interval. If the verifier gets m clones with the same link-id, he can then interpolate the polynomial E and learn the user's identity.

A drawback of the basic glitch protection scheme is that all time intervals must be non-overlapping. Suppose a user is be permitted five glitches per day. If the clock resets at midnight, a malicious user may create five clones just before midnight and five more immediately after. To prevent this behavior, we create window glitch protection. A time interval now consists of any W adjacent time periods. So in the above example, the user would be caught if he made more than five clones in a twenty-four hour period. Section 6.3 describes how we achieve window glitch protection. The general idea is a user reveals a point on W different equations K_i that each correspond to a different link-id (one for each time interval that a time period is part of) and W different polynomials E_i.

In chapter 4, we decribe how to apply e-cash to anonymous authentication and create glitch protection extensions using secret sharing techniques.

1.5 New Results in Secret Sharing

Our work has shown that secret sharing techniques are very useful for secure anonymous transactions using e-cash. In this section, we describe a new secret sharing scheme. We then explain how it fits into the e-cash framework.

Disjunctive Multi-Level Secret Sharing. Introduced by Simmons [93], a disjunctive multi-level access structure assigns all users to some level L. Each level is given a threshold t_L, and these thresholds form an increasing sequence. A group of users is authorized if there exists some level L such that there are at least t_L users in the group at levels $0...L$.

Simons [93] gives a very general exponential time solution to this problem, that is more of a proof that a solution exists. Brickell [21] gives a concrete exponential time solution to the problem; using Shoup's [91] guess-and-check algorithm, it can be sped up to expected polynomial time. Brickell's solution only works on secrets chosen from certain types of fields, which depend on the access structure. Simmons' and Brickell's solutions allow new users to join the system at any time and at any level. Tassa [95] gives a polynomial time solution, but his solution does not allow new users to join the system. See Section 2.3.5 for a comprehensive literature survey of this problem.

We construct a new disjunctive multi-level secret sharing scheme. Our scheme runs in polynomial

time (it is actually faster than Tassa's scheme). It allows new users to join the system at any time and at any level. Our scheme is based on the Birkhoff interpolation problem (see Section 2.1.3). The dealer selects a random polynomial f such that the coefficient of the highest order monomial is equal to the secret. The share of each user is either a point on f or a point on a derivative of f, depending on the user's level. We prove security by reducing our disjunctive secret sharing scheme to Tassa's conjunctive multi-level secret sharing scheme [95] (see Section 2.3.4 for Tassa's construction).

Suppose a dealer wants to share a secret s. The dealer chooses a random polynomial f of degree $t_n - 1$, where n is the highest level, such that the coefficient of x^{t_n-1} is s. The share of user u at level L is $f^{(t_n-t_L)}(u)$. A dealer can also prove that he has distributed consistent shares by publishing commitments to all of the coefficients of f. An authorized set of users can recover the secret by solving a system of linear equations to learn the coefficents of f.

In Chapter 3, we decribe the new disjunctive multi-level secret sharing scheme in more detail and prove it is secure.

Possible applications. A recent trend in e-cash protocols is divisible e-cash, which allows users to pay an arbitrary value, or at least a power of two, in a single transaction. For example, the first provably secure scheme is due to Canard and Gouget [33]. All divisible e-cash schemes to date are based on the same idea: the bank gives the user a signature on a seed to a pseudorandom function (the bank does not learn what the seed is). The user uses the seed to generate a serial number and two more seeds. Each seed is then used to generate a serial number and two more seeds. The result is a binary tree of serial numbers. Leaf nodes in the tree represent e-coins. To spend multiple e-coins, a user gives the merchant the seed of a node higher up in the tree; the merchant can apply the pseudorandom function to the seed to generate all the child nodes and their serial numbers.

Advanced secret sharing techniques may offer a radically different and more efficent solution to the divisible e-cash problem. Suppose we use disjunctive multi-level secret sharing directly. The user would reveal a single serial number for the wallet and a share of his ID. If the e-coin is worth 2^L dollars, he would reveal a share at level $n - L$ (where n is a constant). In this construction, it would be in the user's best interest to spend $t_0 - 1$ e-coins worth 2^n dollars, $t_1 - 1$ e-coins worth 2^{n-1} dollars, etc. Thus the wallet is not arbitrarily divisible.

The ideal secret sharing access structure for divisible e-cash is the weighted threshold secret sharing access structure: each user is assigned a weight, and if the sum of the weights in a group of users is higher than a threshold, then the users are authorized. To apply this to e-cash, a user would assign a weight to each e-coin. If the total weight of the spent e-coins is higher than the amount withdrawn from the bank, then the user has overspent and the bank should be able to use the deposited e-coins to learn the user's identity.

The only universal weighted threshold secret sharing scheme in the literature simply gives each user multiple shares of the secret; a user with weight w would get w shares. This is highly inefficient; to spend a coin worth w dollars would require calculating and revealing w shares. We need a construction where a user reveals at most $\log w$ shares. Beimel et al. [9] show that it is possible to perform weighted threshold secret sharing using only one share, but their construction takes

exponential time and only works for a small subset of weight distributions. Beimel et al. use disjunctive multi-level secret sharing as one of their building blocks; it might be possible to expand on their scheme.

1.6 Conclusion

My work so far has used e-cash, and extensions of e-cash, to solve a wide variety of problems. Endorsed e-cash has been especially useful for solving problems ranging from fair payment of multiple e-coins to onion routing. Secret sharing techniques provide the mathematical building blocks for these solutions.

The rest of this dissertation is organized as follows: Chapter 2 overviews some fundamental cryptography and provides an in-depth introduction to secret sharing and e-cash. Chapter 3 presents a new disjunctive multi-level secret sharing scheme. Chapter 4 presents endorsed e-cash and shows how it can be used to pay for electronic goods and services. Chapter 5 examines how endorsed e-cash can be applied to concrete peer-to-peer applications. Finally, Chapter 6 shows how to use a variant of e-cash for anonymous authentication.

Chapter 2

Background

This chapter covers fundamental cryptography and provides an in depth look at e-cash and secret sharing. Section 2.1 introduces some notation and provides a basic mathematical background. Section 2.2 covers complexity theory and fundamental cryptographic primitives. Section 2.3 introduces secret sharing and surveys past results, while Section 2.4 does the same for electronic cash.

2.1 Basics

We go over some basic notation and mathematical definitions.

Let S be a set. Then $|S|$ is the number of elements in S. $\mathcal{P}(S)$ is the power set of S – it is the set of all subsets of S.

Let $\vec{a} = (a_0, a_1, \ldots, a_n)$ and $\vec{b} = (b_0, b_1, \ldots, b_n)$ be vectors. We can compute the dot product as $\vec{a} \cdot \vec{b} = a_0b_0 + a_1b_1 + \cdots + a_nb_n$. We can also perform pairwise multiplication as $\vec{a} \diamond \vec{b} = (a_0b_0, a_1b_1, \ldots, a_nb_n)$.

Let $f(x) = \sum a_i x^i$ be a polynomial of degree d. The ith formal derivative of f evaluated at x is defined as:

$$f^{(i)}(x) = \sum_{j=0}^{d} \frac{j! a_j x^{j-i}}{(j-i)!}$$

If $\vec{a}(x) = (f_0(x), f_1(x), \ldots, f_n(x))$ is a vector, then $\vec{a}^{(i)}(x) = ({f_0}^{(i)}(x), {f_1}^{(i)}(x), \ldots, {f_n}^{(i)}(x))$.

2.1.1 Groups and Fields

We briefly cover some concepts from group theory. For a more comprehensive treatment, see Dummit and Foote [49].

Let $\mathbb{G}$ be nonempty set, and $*$ be a binary operation on $\mathbb{G} \times \mathbb{G}$. We say that $\mathbb{G}$ forms a group under the operation $*$ if the following conditons hold:

1. **Closure.** The operation $*$ always maps to $\mathbb{G}$: $\forall a, b \in \mathbb{G} : a * b \in \mathit{Group}$.
2. **Associativity.** The operation $*$ is associative: $\forall a, b, c \in \mathbb{G} : a * (b * c) = (a * b) * c$.
3. **Identity.** There exists an element $e \in \mathbb{G}$ such that $\forall a \in \mathbb{G} : e * a = a * e = a$.
4. **Inverses.** Every element in $\mathbb{G}$ has an inverse: $\forall a \in \mathbb{G} : \exists b \in \mathbb{G} : a * b = e$.

We say that a group $\mathbb{G}$ is *commutative* if $\forall a, b \in \mathbb{G} : a * b = b * a$. A group $\mathbb{G}$ is *finite* if $|\mathbb{G}|$ (the number of elements in $\mathbb{G}$) is finite. The number of elements of a finite group is called its order. We write $\mathbb{G} = \langle g \rangle$ to indicate that g is a generator of $\mathbb{G}$; this means that any element of $\mathbb{G}$ can be expressed as g^a, for some integer a.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups under operations $*$ and $\cdot$, respectively. We say that a function $\phi : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a *homomorphic function* if $\forall a, b \in \mathbb{G} : \phi(a * b) = \phi(a) \cdot \phi(b)$.

We will frequently encounter the following four groups:

Group $\mathbb{Z}_q$. The group $\mathbb{Z}_q$ contains all the integers modulo q, where q is prime. The group operation is addition modulo q. It can be thought of as $0, 1, \ldots, q-1$. The order of $\mathbb{Z}_q$ is q. Every element of $\mathbb{Z}_q$ is a generator.

Group $\mathbb{Z}_q^*$. The group $\mathbb{Z}_q^*$ is contains all the integers modulo q, where q is prime, that are relatively prime to q. The group operation is multiplication modulo q. It can be thought of as $1, \ldots, q-1$. The order of $\mathbb{Z}_q$ is q.

Group $\mathbb{Z}_n^*$. The group $\mathbb{Z}_n^*$ contains all the integers that are relatively prime to n and smaller than n, where $n = pq$ and p and q are prime. The group operation is multiplication modulo n. The order of the group is $(p-1)(q-1)$.

Abstract Group. We will frequently talk about some general group $\mathbb{G}$. In our notation, we will consider the group operation to be multiplication; the operator will be denoted either as $*$ or omitted entirely.

A field is an algebraic object similar to a group, except that it is defined with two binary operations $(+, \cdot)$. Let $\mathbb{F}$ be a non-empty set. We say that $(\mathbb{F}, +, \cdot)$ is a field if:

1. **Additive Group.** $(\mathbb{F}, +)$ is a commutative group. Let 0 be the identity in this group.
2. **Multiplicative Group.** $(\mathbb{F}/\{0\}, \cdot)$ is a commutative group. Let 1 be the identity in this group.
3. **Distinct Identities.** $0 \neq 1$ $\mathbb{F}$ has distinct additive and multiplicative identities.
4. **Distributive Property.** The distributive property holds on the operations '+' and '$\cdot$': $\forall a, b, c \in \mathbb{F} : a \cdot (b + c) = (a \cdot b) + (b \cdot c)$.

This work deals primarily with finite fields. All finite fields are of order q^β, where q is a prime number and β is a positive integer. A generic finite field can be denoted as either $GF(q^\beta)$ or $\mathbb{F}_{q^\beta}$.

2.1.2 Bilinear Maps

We briefly describe bilinear maps; for more details on pairing based cryptography, see Boyen [19].

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be groups of prime order q. We say that a function $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear map if it is:

Bilinear. $\forall g \in \mathbb{G}_1, h \in \mathbb{G}_2 : \mathbf{e}(g^a, h^b) = \mathbf{e}(g, h)^{ab}$.

Non-degenerate. For all generators $g \in \mathbb{G}_1$, $h \in \mathbb{G}_2$: $\mathbf{e}(g, h) \neq 1$.

We introduce the function $\mathsf{Bilinear\_Setup}(1^k)$, that outputs $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, g_T, \mathbf{e})$, where $\mathbb{G}_1$,$\mathbb{G}_2$, and $\mathbb{G}_T$ are groups of order q (where q is a k-bit prime), along with an efficiently computable bilinear map $\mathbf{e}$. In some cases, $\mathbb{G}_1 = \mathbb{G}_2$, so the bilinear map is redefined to $\mathbf{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$. Such groups, based on Weil and Tate pairings over eliptic curves (see Silverman [92]) and have been extensively used in cryptographic literature over the past few years.

2.1.3 Polynomial Interpolation

We describe some basic ideas from polynomial interpolation; see [95] for a more detailed overview.

Let $f(x) = \sum a_i x^i$ be a polynomial of degree d. Recall that $f^{(i)}(x)$ is the ith formal derivative of f evaluated at x; $f^{(0)}(x) = f(x)$.

Lagrange interpolation. Given set S of $d+1$ points, we can perform Lagrange interpolation to construct a unique polynomial that intercepts every point in S:

$$f(x) = \sum_{u \in S} f(u) \prod_{\substack{v \in S \\ v \neq u}} \frac{x - u}{v - u}$$

Using the naive approach, it takes $O(d^2)$ field operations to evaluate $f(x)$ for some x. Shamir [90] shows there is also an $O(d \log^2 d)$ algorithm. If we have less than $d+1$ points, then for a fixed x, every point is equally likely to be $f(x)$.

Hermite Interpolation. Suppose we are given $d+1$ points that either lie on f or on one of its derivatives:

$$f_{i,x} = f^{(i)}(x)$$

It is possible to uniquely reconstruct the polynomial $f \in \mathbb{R}[x]$ if, for each x, the points $f_{i,x}$ we get come from an unbroken sequence of derivatives beginning at the 0th derivative: $f_{0,x}, f_{1,x}, \ldots, f_{j,x}$. In this case, we can reconstruct f by performing Gaussian elimination to learn $a_0, a_1, \ldots, a_d$.

Birkhoff Interpolation. This is a generalization of the Hermite interpolation problem. For any x, the sequence of derivatives $f^{(i)}(x)$ may be broken, or not begin at the 0th derivative. In this case, a solution may not exist, or there may be more than one polynomial that fits the requirements. There exist some known necessary and sufficient conditions for the Birkhoff interpolation problem. See Tassa [95] for details. If a solution does exist, we can perform Gaussian elimination to learn $a_0, a_1, \ldots, a_d$.

2.2 Cryptographic fundamentals

We describe some fundamental concepts in public-key cryptography. For a detailed overview, see Goldreich [56].

2.2.1 Complexity Theory

We measure the complexity of an algorithm in terms of how long it takes to run and the amount of memory space it uses. We use standard asymptotic complexity notation. If we say that a function $f(k)$ is $O(g(k))$, this means that there exist positive values a, b, such that $\forall k \geq a : f(k) \leq b \cdot g(k)$. An algorithm is said to be polynomial time if, given input of length k, it runs in $O(k^c)$ time, for some constant c. An algorithm is said to run in exponential time if, given input k, it runs in c^k time, for some constant $c > 1$. Finally, we say that a function f is negligible if $f(k) < k^{-c}$ for all $c > 1$. We typically use $\nu(k)$ to indicate a negligible function.

We typically measure the running time of an algorithm in terms of the number of basic steps. In some cases, this can is the number of field operations: multiplication, division, addition, and subtraction. Sometimes we count the number of exponentiations performed. An exponentiation can be single base (e.g. g^a) or multi-base (e.g. $g^a h^b$). In a good implementation, multi-base exponentiation takes only slightly longer than single-base exponentiation.

We prove the security of a cryptographic construction by showing that breaking the construction is as hard as solving some difficult problem, under plausible harness assumptions. To determine how hard a problem is, we upper bound the probability that a probabilistic polynomial time Turing machine (PPTM) can ever solve it. We model the situation as a game. In the game, a series of events can occur. We use notation developed by Goldwasser, Micali and Rivest [60]. We write $\Pr[a; b; c : d]$ to mean the probability of event d, in the probability space defined by a, b, and c. We write $x \leftarrow S$ to denote choosing a value x from some set S. Unless otherwise stated, x is chosen uniformly at random from the elements of S. If A is an algorithm, with inputs $(a, b, c, \ldots)$, then $x \leftarrow A(a, b, c, \ldots)$ is the output of A, chosen over the coin flips performed by A. For example, $\Pr[a \leftarrow \mathbb{Z}_q : a \bmod 2 = 0]$ is the probability of choosing an even number at random from $\mathbb{Z}_q$.

2.2.2 Complexity Assumptions

We describe the complexity assumptions required by our work. The first two assumptions are about composite order groups. We say that a $2k$-bit integer n is a RSA modulus if $n = pq$, where p and q are equal length primes.

Definition 2.2.1 (Strong RSA Assumption [8, 53])**.** *Let $n = pq$ be a $2k$ bit integer, where p and q are equal length primes. primes. The Strong RSA assumption states that no PPTM adversary $\mathcal{A}$ can compute values h, e such that $h^e \equiv g \bmod n$:*

$$\Pr[g \leftarrow \mathbb{Z}_n^*; (h, e) \leftarrow \mathcal{A}(g, n) : (e > 1) \wedge (h^e \equiv g \bmod n)] \leq \nu(k)$$

Definition 2.2.2 (Paillier Assumption [83]). *Let $n = pq$ be a $2k$ bit integer, where p and q are equal length primes. We define the set $P = \{a^n | a \in \mathbb{Z}_{n^2}\}$. The Paillier assumption states that no PPTM can distinguish a random element of P from a random element of $\mathbb{Z}_{n^2}$:*

$$\Pr[g_0 \leftarrow P; g_1 \leftarrow \mathbb{Z}_{n^2}; b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}(n, g_b) : b' = b] \leq \frac{1}{2} + \nu(k)$$

The remaining assumptions deal with groups of prime order q. The first three, the discrete logarithm assumption, the Diffie-Hellman assumption, and the decisional Diffie-Hellman assumption, are well accepted. The remaining ones, especially the LRSW and the Sum-Free Decisional Diffie-Hellman, are more recent and are considered to be very strong assumptions.

Definition 2.2.3 (Discrete Logarithm Assumption [45]). *Let $\mathbb{G}$ be a group of order q, where q is a k-bit prime. Let g be a randomly chosen generator of $\mathbb{G}$. The discrete logarithm assumption states that no PPTM that gets as input $h = g^x$ can compute x with more than negligible probability:*

$$\Pr[x \leftarrow \mathbb{Z}_q; h \leftarrow g^x; x' \leftarrow \mathcal{A}(q, g, h) : g^{x'} = h] \leq \nu(k)$$

Definition 2.2.4 (Diffie-Hellman Assumption [45]). *Let $\mathbb{G}$ be a group of order q, where q is a k-bit prime. Let g be a randomly chosen generator of $\mathbb{G}$. The Diffie-Hellman assumption states that no PPTM that gets as input (g, g^x, g^y) can compute g^{xy} with more than negligible probability:*

$$\Pr[x, y \leftarrow \mathbb{Z}_q; h \leftarrow \mathcal{A}(q, g, g^x, g^y) : h = g^{xy}] \leq \nu(k)$$

Definition 2.2.5 (Decisional Diffie-Hellman Assumption (DDH) [79]). *Let $\mathbb{G}$ be a group of order q, where q is a k-bit prime. Let g be a randomly chosen generator of g. The Decisional Diffie-Hellman assumption states that no PPTM that gets as input (g, g^x, g^y) can distinguish g^{xy} from a random element in $\mathbb{G}$ with more than negligible probability:*

$$\Pr[x, y \leftarrow \mathbb{Z}_q; h_0 \leftarrow g^{xy}; h_1 \leftarrow \mathbb{G}; b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}(q, g, g^x, g^y, h_b) : b' = b] \leq \frac{1}{2} + \nu(k)$$

Definition 2.2.6 (y-Diffie-Hellman Inversion Assumption (y-DHI) [77]). *Let $\mathbb{G}$ be a group of order q, where q is a k-bit prime. Let g be a randomly chosen generator of $\mathbb{G}$. The y-DHI assumption states that no PPTM that gets as input $(g, g^x, \ldots, g^{(x^y)})$ can compute $g^{1/x}$ with more than negligible probability:*

$$\Pr[x \leftarrow \mathbb{Z}_q; h \leftarrow \mathcal{A}(q, g, g^x, \ldots, g^{(x^y)}) : h = g^{1/x}] \leq \nu(k)$$

Definition 2.2.7 (y-Decisional Diffie-Hellman Inversion Assumption (y-DDHI) [15, 48]). *Let $\mathbb{G}$ be a group of order q, where q is a k-bit prime. Let g be a randomly chosen generator of $\mathbb{G}$. The y-DDHI assumption states that no PPTM that gets as input $(g, g^x, \ldots, g^{(x^y)})$ can distinguish $g^{1/x}$ from a random element in $\mathbb{G}$ with more than negligible probability:*

$$\Pr[x \leftarrow \mathbb{Z}_q^*;\ h_0 = g^{1/x};\ h_1 \leftarrow \mathbb{G};\ b \leftarrow \{0,1\};$$
$$b' \leftarrow \mathcal{A}(g, g^x, g^{x^2}, \ldots, g^{x^y}, h_b) : b = b'] < \frac{1}{2} + \nu(k)$$

Definition 2.2.8 (Strong DDH Inversion Assumption (SDDHI)[23])**.** *Suppose that $g \in \mathbb{G}$ is a random generator of order $q \in \Theta(2^k)$. Let $\mathcal{O}_a(\cdot)$ be an oracle that, on input $z \in \mathbb{Z}_q^*$, outputs $g^{1/(a+z)}$. Then, no PPTM adversary $\mathcal{A}^{(\cdot)}$ that does not query the oracle on x can distinguish $g^{1/(a+x)}$ from random:*

$$\Pr[a \leftarrow \mathbb{Z}_q^*;\ (x, \alpha) \leftarrow \mathcal{A}^{\mathcal{O}_a}(g, g^a);\ h_0 = g^{1/(a+x)};\ h_1 \leftarrow \mathbb{G};$$
$$b \leftarrow \{0,1\};\ b' \leftarrow \mathcal{A}^{\mathcal{O}_a}(h_b, \alpha) : b = b'] < \frac{1}{2} + \nu(k).$$

Definition 2.2.9 (External Diffie-Hellman Assumption (XDH) [54, 89, 73, 16])**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be groups of prime order q, where q is a k-bit prime, and let $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a bilinear map. The XDH assumption states that the Decisional Diffie-Hellman (DDH) problem is hard in $\mathbb{G}_1$.*

The XDH assumption is conjectured to hold only in non-supersingular curves.

Definition 2.2.10 (Sum-Free Decisional Diffie-Hellman Assumption (SF-DDH) [47])**.** *Suppose that $g \in \mathbb{G}$ is a random generator of order $q \in \Theta(2^k)$. Let L be any polynomial function of k. Let $\mathcal{O}_{\vec{a}}(\cdot)$ be an oracle that, on input a subset $I \subseteq \{1, \ldots, L\}$, outputs the value $g_1^{\beta_I}$ 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 $\mathcal{A}^{(\cdot)}$,*

$$\Pr[\vec{a} = (a_1, \ldots, a_L) \leftarrow \mathbb{Z}_q^L; (J, \alpha) \leftarrow \mathcal{A}^{\mathcal{O}_{\vec{a}}}(1^k); y_0 = g^{\prod_{i \in J} a_i};$$
$$y_1 \leftarrow \mathbb{G}; b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}^{\mathcal{O}_{\vec{a}}}(y_b, \alpha)\ :\ b = b' \wedge$$
$$R(J, Q) = 1] < 1/2 + 1/\nu(k),$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{\vec{a}}(\cdot)$.

Definition 2.2.11 (Lysyanskaya-Rivest-Sahai-Wolf Assumption (LRSW) [70])**.** *Let $(q, \mathbb{G}, \mathbb{G}_T, g, g_T, \mathbf{e}) \leftarrow \mathsf{Bilinear\_Setup}(1^k)$. Let $\mathcal{O}_{x,y}$ be an oracle that on input $m \in \mathbb{Z}_q$, randomly chooses h, and outputs the triple (h, h^y, h^{x+mxy}). The LRSW assumption states that no PPTM can generate this triple on its own:*

$$\Pr[(q, \mathbb{G}, \mathbb{G}_T, g, g_T, \mathbf{e}) \leftarrow \mathsf{Bilinear\_Setup}(1^k); x, y \leftarrow \mathbb{Z}_q; (Q, m, h, h_1, h_2) \leftarrow \mathcal{A}^{\mathcal{O}_{x,y}}(q, \mathbb{G}, \mathbb{G}_T, g, g_T) :$$
$$(m \notin Q) \wedge (h \in \mathbb{G}) \wedge (h_1 = h^y) \wedge (h_2 = h^{x+mxy})] \leq \nu(k)$$

Lysyanskaya et al. [70] show that the LRSW assumption holds in the generic group model and is independent of the Diffie-Hellman assumption.

Definition 2.2.12 (Decisional Bilinear Diffie-Hellman Assumption (DBDH) [63, 17])**.** *Let $(q, \mathbb{G}, \mathbb{G}_T, g, g_T, \mathbf{e}) \leftarrow \mathsf{Bilinear\_Setup}(1^k)$. The DBDH assumption states that no PPTM can distinguish g^{xyz} from*

a random value in $\mathbb{G}_T$*:*

$$\Pr[(q, \mathbb{G}, \mathbb{G}_T, g, g_T, \mathbf{e}) \leftarrow \mathsf{Bilinear\_Setup}(1^k); x, y, z \leftarrow \mathbb{Z}_q; h_0 = \mathbf{e}(g,g)^{xyz}; h_1 \leftarrow \mathbb{G}_T; \\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}(q, g, g^x, g^y, g^z, h_b) : b = b'] \leq \frac{1}{2} + \nu(k)$$

The DBDH assumption is implied by y-DDHI or Sum-Free DDH.

Definition 2.2.13 (y-Decisional Bilinear Diffie-Hellman Inversion Assumption (y-DBDHI) [14])**.** *Let* $(q, \mathbb{G}, \mathbb{G}_T, g, g_T, \mathbf{e}) \leftarrow \mathsf{Bilinear\_Setup}(1^k)$*. The y-DBDHI assumption states that given the tuple* $(g, g^x, \ldots, g^{(x)^y})$*, no PPTM can distinguish* $\mathbf{e}(g,g)^{1/x}$ *from a random value in* $\mathbb{G}_T$*:*

$$\Pr[(q, \mathbb{G}, \mathbb{G}_T, g, g_T, \mathbf{e}) \leftarrow \mathsf{Bilinear\_Setup}(1^k); x \leftarrow \mathbb{Z}_q; h_0 = \mathbf{e}(g,g)^{1/x}; h_1 \leftarrow \mathbb{G}_T; \\ b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}(q, g, g^x, \ldots, g^{(x)^y}, h_b) : b = b'] \leq \frac{1}{2} + \nu(k)$$

2.2.3 Hash Functions

Definition 2.2.14 (Collision resistant hash function)**.** *Let* $H = h_k$ *be a family of functions,* $h_k : \mathbb{D}_k \rightarrow \mathbb{R}_k$*. We say that H is a family of hash functions if* $|\mathbb{R}_k| < |\mathbb{D}_k|$*. We say that H is a family of collision resistant hash functions if no PPTM can find two values in* $\mathbb{D}_k$ *that map to the same value in* $\mathbb{R}_k$*:*

$$\Pr[h \leftarrow H(1^k); (x, y) \leftarrow \mathcal{A}(h) : (h(x) = h(y)) \wedge (x \neq y)] \leq \nu(k)$$

For this work, we assume the existence of efficient collision resistant hash functions.

2.2.4 Merkle Hash Trees

A Merkle hash tree [74] uses a collision resistant hash function to efficiently commit to a large block of data.

Suppose that we have a block consisting of 2^ℓ chunks $\mathit{chunk}_1, \ldots, \mathit{chunk}_{2^\ell}$. Consider a rooted binary tree with 2^ℓ leaves (*i.e.* it is a binary tree of height ℓ). Associated with every node the tree, there is the address a of the node. Specifically, the label a associated with the root node is the empty string ε. The label of a left (resp. right) child is derived by concatenating 0 (resp., 1) to the label of the parent node. Thus, the label a associated with the i^{th} leaf is the integer i written in binary.

Stored at a node labeled with a, is a value v_a. At the i^{th} leaf, the value stored is $v_i = h(\mathit{chunk}_i)$. For each internal node a, $v_a = h(v_{a||0}||v_{a||1})$. By MHash($\mathit{block}$) we denote the value v_ε obtained by applying this procedure to $\mathit{block} = \mathit{chunk}_1||\ldots||\mathit{chunk}_{2^\ell}$.

To prove that *chunk* is the i^{th} chunk of the block represented by $\mathit{bhash} = \text{MHash}(\mathit{block})$, reveal the values v_{a_j} where for $1 \leq j \leq \ell$, the label a_j is obtained by taking the first $j-1$ bits of the binary representation of i, and concatenating to them the negation of the j^{th} bit. (For example, if $i = 0101$, then $a_1 = 1$, $a_2 = 00$, $a_3 = 011$ and $a_4 = 0100$.) In the Merkle tree, the node a_j will be the sibling of the j^{th} node on the path from the root to the chunk in question.

To verify that $(v_1, ..., v_\ell)$ is a valid proof that *chunk* is the i^{th} chunk of *block* associated with *bhash*, first we compute the labels a_j (as above). We know that $v_j = v_{a_j}$ is the value that should be associated with node labelled a_j. Let $i = i_1 i_2 \dots i_\ell$, *i.e.* i_j is the j^{th} bit of the ℓ-bit binary representation of i. Let $b_j = i_1 \dots i_j$ be the j-bit prefix of i. First, we know that $v_i = h(chunk)$. For each j, $\ell - 1 \geq j \geq 0$, compute $v_{b_j} = h(v_{b_j||0}||v_{b_j||1})$. We can do it because one of $(v_{b_j||0}, v_{b_j||1})$ is $v_{a_{j+1}}$, and the other one is computed in the previous step. Finally, verify that $v_\varepsilon = bhash$.

If two conflicting proofs can be constructed (*i.e.* for $chunk \neq chunk'$, there are proofs that each of them is the i^{th} chunk of *block* associated with *bhash*), then a collision in h is found, contradicting the assumption that h is collision-resistant.

2.2.5 Pseudorandom Functions

Definition 2.2.15 (Pseudorandom Function)**.** *Let $\mathcal{F} = \{f_k\}$ be a family of functions $f_k : \mathbb{D}_k \to \mathbb{R}_k$. Let $\mathcal{O}_k(\cdot)$ be an oracle that on input from $\mathbb{D}_k$ outputs a random value in $\mathbb{R}_k$ ($\mathcal{O}$ always gives consistent responses to the same input). $\mathcal{F}$ is a family of pseudorandom functions if no PPTM can distinguish the output of $f_k \in \mathcal{F}$ from $\mathcal{O}_k(\cdot)$:*

$$\Pr[b \leftarrow \mathcal{A}^{\mathcal{O}_k(\cdot)}(1^k) : b = 0] - \Pr[b \leftarrow \mathcal{A}^{f_k(\cdot)}(1^k) : b = 0] \leq \frac{1}{2} + \nu(k)$$

Dodis and Yampolskiy [48] construct a pseudorandom function that lends itself nicely to efficient zero-knowledge proofs. Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q, where q is a k-bit prime. Let s be a random element of $\mathbb{Z}_q^*$. Then:

$$\mathsf{DY}_{g,s}(x) = g^{1/(s+x)}$$

Dodis and Yampolskiy [48] show that the Dodis-Yampolskiy PRF $\mathsf{DY}_{g,s}(x)$ is secure 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). Camenisch et al. [23] show that the DY PRF is secure for all inputs in $\mathbb{Z}_q^*$ in the generic group model, given the SDDHI assumption.

2.2.6 Commitment Schemes

A commitment scheme consists of three protocols: $\mathsf{ComSetup}$, Commit, and Verify.

1. $\mathsf{ComSetup}(1^k)$. Outputs *params*, the protocol parameters.
2. $\mathsf{Commit}(params, x)$. Outputs a commitment C to the value x.
3. $\mathsf{Verify}(params, C, x, proof)$. Outputs 1 if C is a valid commitment to x.

A commitment scheme should have the following two properties:

Binding. No PPTM can open a commitment to two different values:

$$\Pr[params \leftarrow \mathsf{ComSetup}(1^k); (C, x, proof, x', proof') \leftarrow \mathcal{A}(params) : (x \neq x') \wedge \mathsf{Verify}(params, C, x, proof) \wedge \mathsf{Verify}(params, C, x', proof')] \leq \nu(k)$$

Hiding. Let x and y be two values chosen uniformly at random from the domain of committed values. We define two distribution ensembles:

$$X = \{params \leftarrow \mathsf{ComSetup}(1^k) : (C, proof) \leftarrow \mathsf{Commit}(params, x)\}$$
$$Y = \{params \leftarrow \mathsf{ComSetup}(1^k) : (C, proof) \leftarrow \mathsf{Commit}(params, x)\}$$

We say that the commitment scheme is perfectly, statistically, or computationally hiding if ensembles X and Y are (respectively) information theoretically, statistically, or computationally indistinguishable.

Definition 2.2.16 (Secure commitment scheme)**.** *A perfectly/statistically/computationally secure commitment scheme is binding and perfectly/statistically/computationally hiding.*

Pedersen Commitments [84]. Protocol $\mathsf{ComSetup}(1^k)$ outputs a group $\mathbb{G}$ of order q, where q is a k-bit prime, and generators $(g_0, \ldots, g_m)$. To commit to values $(x_1, \ldots, x_m) \in \mathbb{Z}_q^m$, pick a random value $r \in \mathbb{Z}_q$ and set:

$$\mathsf{PedCom}(x_1, \ldots, x_m; r) = g_0^r \prod_{i=1}^{m} g_i^{x_i}$$

The protocol Verify takes as input $(x_1, \ldots, x_m, r)$, calculates $\mathsf{PedCom}(x_1, \ldots, x_m; r)$, and outputs 1 iff the result equals to the commitment C. Pedersen commitments are perfectly hiding.

Fujisaki-Okamoto Commitments [53]. Fujisaki and Okamoto extend the Pedersen Commitment scheme to $\mathbb{Z}_n^*$, where $n = pq$, and p, q are safe primes. Later, Damgaard and Fujisaki [43] generalize this construction to work for a larger class of composite order groups.

We denote a commitment to values $(x_1, \ldots, x_m)$ using randomness r as $\mathsf{FujOkaCom}(v_1, \ldots, v_m; r)$. Fujisaki-Okamoto commitments are statistically hiding.

2.2.7 Zero-Knowledge Proofs of Knowledge

Goldwasser, Micali and Rackoff [59] introduce the concept of zero-knowledge proofs. The idea is that a prover $\mathcal{P}$ proves to a verifier $\mathcal{V}$ that some statement is true, without revealing any other additional information. For example, $\mathcal{P}$ might prove that he knows a value x such that $g^x = h$. Goldreich, Micali and Wigderson [57] showed how to prove any statement in NP in zero-knowledge, assuming the existence of a commitment scheme.

Let L be a language, and $R : \{0,1\}^* \times \{0,1\}^*$ a relation such that $(x, w) \in R$ if w is a witness proving that $x \in L$. A zero-knowledge proof of knowledge proof system is a two-party protocol. The input of the prover $\mathcal{P}$ is (x, w). The input of the verifier $\mathcal{V}$ is x. At the end of the protocol, $\mathcal{V}$ should output 1 if $(x, w) \in R$. The verifier learns no information about w.

$\mathsf{Setup}(1^k)$ outputs the public parameters *params*.

$\mathsf{Prove}(\mathcal{P}(x, w, params), \mathcal{V}(x, params))$ at the end of execution, the verifier $\mathcal{V}$ outputs 0 or 1 depending on whether it accepts. The prover $\mathcal{P}$ outputs nothing.

A zero-knowledge proof of knowledge proof system needs to have the following three properties.

Completeness. Suppose an honest prover $\mathcal{P}$ gets input (x, w) such that $(x, w) \in R$, and an honest verifier $\mathcal{V}$ gets input x. Then the verifier should accept with probability 1.

Soundness. The probability that an honest verifier $\mathcal{V}$ accepts a proof for $x \notin L$ is negligible:

$$\Pr[params \leftarrow \mathsf{Setup}(1^k); (state, x \notin L) \leftarrow \mathcal{P}^*(params); \mathsf{Prove}(\mathcal{P}^*(state), \mathcal{V}(x)) : 1 \leftarrow \mathcal{V}] \leq \nu(k).$$

Zero-knowledge. For every dishonest verifier $\mathcal{V}^*$, there exists a simulator $\mathcal{S}$ so that $\forall(x, w) \in R$, no PPTM adversary can distinguish between the output of $\mathcal{V}^*$ after running $\mathsf{Prove}(\mathcal{P}(x, w), \mathcal{V}^*)$ and the output of $\mathcal{S}$. In other words, $\forall(x, w) \in R$, the following two distribution ensembles are indistinguishable:

$$X = \{params \leftarrow \mathsf{Setup}(1^k); output \leftarrow \mathsf{Prove}(\mathcal{P}(x, w, params), \mathcal{V}^*(x, params)) : (output, params)\}$$
$$Y = \{(params, output) \leftarrow \mathcal{S}(1^k) : (output, params)\}$$

If the soundness property holds even if the prover is computationally unbounded, then the protocol is called a zero-knowledge *proof*, while if soundness holds only if the prover is a PPTM, then the protocol is a called a zero-knowledge *argument*. We can also distinguish the strength of the zero-knowledge property – whether the output of $\mathcal{S}$ and $\mathcal{P}$ are information theoretically (i.e. perfectly), statistically, or computationally indistinguishable. Zero-knowledge proofs may be interactive or non-interactive.

Bellare and Goldreich [11] define a proof of knowledge, which is an extension of a zero-knowledge proof. Now, insted of proving that there exists a witness w that $x \in L$, we can prove that we *know* a witness w. To do this, we replace the soundness property with a stronger requirement called *extraction:*

Extraction. For every dishonest prover $\mathcal{P}^*$, if an honest verifier would accept its proofs, then there exists a PPTM extractor $\mathcal{E}$ that can calculate the witness w. We define:

$$\forall x : p(x) = \Pr[params \leftarrow \mathsf{Setup}(1^k); \mathsf{Prove}(\mathcal{P}^*(x, w, params), \mathcal{V}(x)) : 1 \leftarrow \mathcal{V}]$$

If $\forall(x, w) \in R$, $p(x)$ is $O(2^{-u(k)})$, for some polynomial $u(k)$, then there exists a PPTM algorithm $\mathcal{E}$, that when given oracle access to $\mathcal{P}^*$ and the ability to choose the setup parameters *params*, calculates a value w' such that $(x, w') \in R$ with probability at least $1 - \nu(k)$.

Besides proofs of knowledge, we can also consider signatures of knowledge. The prover generates a digital signature that states "a person who knows a witness for the statement $x \in L$ signed this document."

We adopt the notation developed by Camenisch and Stadler [32]. We write $\mathsf{PK}\{(variables) : properties\}$ to indicate a zero-knowledge proof of knowledge of *variables* which meet some *properties*. For example, $\mathsf{PK}\{(\alpha, \beta) : A = g^\alpha \wedge B = A^\beta\}$ is a proof of knowledge of two values, α and β such

that $A = g^\alpha \wedge B = A^\beta$. We will always use Greek letters to indicate unknowns and Roman letters to indicate publicly known values. When we perform non-interactive zero-knowledge proofs, or signatures of knowledge, we write SPK instead of PK. So $\mathsf{PK}\{(\alpha, \beta) : A = g^\alpha \wedge B = A^\beta\}(m)$ is a signature on message m.

2.2.8 Digital Signatures

A stateless signature scheme consists of three protocols: SigSetup, Sign, SigVerify.

$\mathsf{SigSetup}(1^k)$ is a probabilistic algorithm that outputs a k-bit signing key sk and k-bit verification key vk.

$\mathsf{Sign}(sk, m)$ is a probabilistic algorithm that outputs a signature σ on message m using signing key sk.

$\mathsf{SigVerify}(vk, m, \sigma)$ outputs 1 if σ is a valid signature on m under the verification key vk.

Definition 2.2.17 (Secure Signature Scheme [60])**.** *A secure signature scheme must satisfy two properties. Firstly, it must be correct:*

$$\forall m : \Pr[(sk, vk) \leftarrow \mathsf{SigSetup}(1^k); \sigma \leftarrow \mathsf{Sign}(sk, m) : \mathsf{SigVerify}(vk, m, \sigma) = 1] = 1$$

It must also be secure against an adaptive chosen message attack. We give the adversary $\mathcal{A}$ access to a signing oracle $\mathsf{Sign}(sk, \cdot)$. $\mathcal{A}$ must honestly record every query to $\mathsf{Sign}(sk, \cdot)$ on a query tape Q. We say that the signature scheme is secure if:

$$\Pr[(sk, vk) \leftarrow \mathsf{SigSetup}(1^k); (Q, m, \sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}(sk,\cdot)}(vk) : (m \notin Q) \wedge \mathsf{SigVerify}(vk, m, \sigma)] \leq \nu(k)]$$

CL-Signatures [26, 27, 16]. Camenisch and Lysyanskaya [26] introduced a secure signature scheme with two protocols: (1) An efficient protocol for a user to obtain a signature on the value in a Pedersen or Fujisaki-Okamoto commitment [84, 53] without the signer learning anything about the value and (2) An efficient zero-knowledge proof of knowledge of a signature on a committed value. There are three known constructions. The original CL-signature [26] is based on the Strong RSA assumption. Camenisch and Lsysyanskaya [27] and Boneh et al. [16] use bilinear maps to construct more efficient CL-signatures.

2.2.9 Encryption and Verifiable Encryption

An encryption scheme consists of three protocols: Setup, Encrypt, Decrypt.

$\mathsf{Setup}(1^k)$ is a probabilistic algorithm that outputs a k-bit public-key pk and a k-bit secret-key sk.

$\mathsf{Encrypt}(pk, m)$ outputs $ctext$, an encryption of m under the public-key pk.

$\mathsf{Decrypt}(sk, ctext)$ outputs m, a decryption of $ctext$ using the secret-key sk.

Definition 2.2.18 (Semantically Secure Encryption [58])**.** *A secure encryption scheme must have two properties. First, it must be correct:*

$$\forall m : \Pr[(sk, pk) \leftarrow \mathsf{Setup}(1^k); ctext \leftarrow \mathsf{Encrypt}(pk, m) : \mathsf{Decrypt}(sk, ctext) = m] = 1$$

Second, a PPTM adversary $\mathcal{A}$ who gets an encryption of one of two messages of its choice, m_0 and m_1, cannot guess which message corresponds to the ciphertext:

$$\begin{aligned}\Pr[&(sk, pk) \leftarrow \mathsf{Setup}(1^k);\\ &(m_0, m_1, state) \leftarrow \mathcal{A}(sk, pk);\\ &b \leftarrow \{0, 1\};\\ &ctext \leftarrow \mathsf{Encrypt}(pk, m_b);\\ &b' \leftarrow \mathcal{A}(ctext, state) : b = b'] < 1/2 + \nu(k)\end{aligned}$$

Bilinear El Gamal Encryption. We require a cryptosystem where g^x is sufficient for decryption and the public key is $\phi(g^x)$ for some function ϕ. One example is the bilinear El Gamal cryptosystem [17, 6], which is semantically secure under the DBDH assumption.

Block Ciphers. Also known as symmetric-key encryption, a block cipher is an encryption scheme where the Encrypt algorithm takes the secret-key sk as input instead of the public-key. Block ciphers are much faster than public-key encryption schemes. However, they are not semantically secure. They also do not provide provable security guarantees. Some common block ciphers are AES [81], DES [80], and Blowfish [88].

Verifiable Encryption [22]. Camenisch and Damgård [22] developed a technique for turning any semantically-secure encryption scheme into a verifiable encryption scheme. Verifiable encryption lets the sender prove to the receiver statements about the content of a ciphertext. This is typically used in escrow schemes; for example, the sender can prove that the ciphertext is an encryption of his identity under the public key of some trusted third party.

Let Setup, Encrypt, Decrypt be a semantically secure encryption scheme, and let $\mathcal{R}$ be a binary relation and define the language $L_{\mathcal{R}} = \{x | \exists w : (x, w) \in \mathcal{R}\}$. A verifiable encryption scheme $\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt}, L_{\mathcal{R}}$ is actually a zero-knowledge proof system where the prover $\mathcal{P}$ proves to the verifier that he knows a witness w to some statement x, where x is a statement about the contents of the ciphertext.

Camenisch and Shoup [31] construct an efficient verfiable encryption scheme for discrete logarithms. The prover can convince the verifier that a *ciphertext* is an encryption of the discrete logarithm of A, under pk: $\mathsf{PK}\{(\alpha) : ciphertext = \mathsf{Encrypt}(pk, \alpha || \beta) \wedge C = g^\alpha h^\beta\}$. This is equivalent to proving that the plaintext in *ciphertext* is the opening of a Pedersen commitment.

2.2.10 Coin Flipping

Coin flipping, introduced by Blum [13], refers to a multi-party protocol in which two (or more) users jointly contribute randomness to generate a sequence of random bits. As a result, neither party can

predict or influence the values of these bits. (The only exception is that one user can terminate the protocol early if he is unhappy with the value of the bit(s), without revealing the chosen bits to the other party; this problem is inherent to any multi-party protocol).

Suppose two users, $\mathcal{U}_1$ and $\mathcal{U}_2$ want to generate a single random value in $\mathbb{Z}_q^*$. A simple three step protocol would be:

1. $\mathcal{U}_1$ randomly chooses $x_1 \in \mathbb{Z}_q^*$, and sends $\mathsf{Commit}(x_1)$ to $\mathcal{U}_2$.
2. $\mathcal{U}_2$ randomly chooses $x_2 \in \mathbb{Z}_q^*$ and sends x_2 to $\mathcal{U}_1$.
3. $\mathcal{U}_1$ sends $\mathcal{U}_2$ the committed value x_1. $\mathcal{U}_2$ verifies that x_1 is correct. Then both $\mathcal{U}_1$ and $\mathcal{U}_2$ compute the final answer $x = x_1 + x_2$.

If $\mathcal{U}_1$ or $\mathcal{U}_2$ honestly follows the protocol, then x is chosen uniformly at random. If $\mathcal{U}_1$ terminates in step 3 and does not send x_1 to $\mathcal{U}_2$, then only $\mathcal{U}_1$ will learn x. If either of them terminate in step 1 or step 2, then neither learn any information about x. To make sure the protocol is well defined, we can require an honest party to output $\perp$ if his partner terminates early.

Now suppose $\mathcal{U}_1$ and $\mathcal{U}_2$ want to choose a sequence of random values jointly. The naive approach is to repeat the coin flipping protocol above for each value. To save effort it is possible to negotiate only one value and generate all other values either using a either a pseudorandom function or a collision resistant hash function. Let F be such a function. The two users generate the seed x of F using the coin flipping protocol. To generate the ith value in the sequence, the two users calculate $F_x(i)$.

2.3 Secret Sharing Background

A secret sharing scheme lets a dealer give shares of a secret to a group of users. An authorized group of users should be able to work together to reconstruct the secret. If a group of users does not contain an authorized group of users, it should learn nothing about the secret. Secret sharing was introduced by Shamir [90] and Blakley [12] and has been studied extensively. In this section, we discuss some classical secret sharing problems, following the notation and ideas due to Simmons [93] (the reader may also refer to Tassa [95], who summarizes these ideas and is more accessible).

2.3.1 Definitions

Definition 2.3.1 (Access structure)**.** *Let $\mathcal{U}$ be a set of users. An access structure $\Gamma \subseteq \mathcal{P}(\mathcal{U})$ must meet the following two conditions: (1)* monotonicity: *if $A \in \Gamma$ and $A \subseteq B$ then $B \in \Gamma$ and (2)* non-triviality: *if $A \in \Gamma$ then $|A| > 0$.*

We say that every set $A \in \Gamma$ is *authorized* and every set $B \notin \Gamma$ is *unauthorized*.

Definition 2.3.2 (Minterm)**.** *Let Γ be an access structure. We say that $A \in \Gamma$ is a minterm if $\forall u \in A : A - \{u\} \notin \Gamma$.*

There are many possible access structures. The most common is the theshold access structure (Section 2.3.3). We will also consider some multi-level access structures: conjunctive (Section 2.3.4) and disjunctive (Section 2.3.5).

In a multi-level access structure, all users are assigned to a level using some function $\mathcal{L} : \mathcal{U} \to \mathbb{Z}$ (each user is assigned to exactly one level, but multiple users can be assigned to the same level). Sometimes, users can be assigned to different levels depending on the context. To handle this, we write $A(\mathcal{L})$ to indicate a set of users A whose levels are calculated using level assignment function $\mathcal{L}$. Thus, for a multi-level access structure, we would write that $A(\mathcal{L}) \in \Gamma$. It is quite possible that for the same A, there exists another level assignment function $\mathcal{L}'$ such that $A(\mathcal{L}') \notin \Gamma$.

To realize an access structure, we need to construct a secret sharing scheme. There are two types of players: users and the dealer. The dealer chooses a secret at random from some domain S and distributes shares of the secret to each user.

Definition 2.3.3 (Secret Sharing Scheme)**.** *Suppose there are n users in the system. Let S be the domain of the secret. A dealer takes a secret $s \in S$, chooses a random string r, and uses the function $(s_1, s_2, \ldots, s_{|A|}) \leftarrow \mathsf{Share}_r(s, A, \mathcal{L}, \Gamma)$ to calculate the shares of users A when they are assigned to levels according to $\mathcal{L}$. We say that a secret sharing scheme is* dynamic *if the dealer can invoke* Share *multiple times with the same randomness r but different sets of users and still get consistent sharings of the same secret.*

Definition 2.3.4 (Secure Secret Sharing Scheme)**.** *A secret sharing scheme is a perfect realization of an access structure Γ if the following two conditions hold:*

1. ***Correctness.*** *Regardless of the secret s and the random choices taken by the dealer, $\forall A \in \Gamma$, the users in A can always reconstruct s.*

2. ***Privacy.*** *$\forall B \notin \Gamma$ the shares of B are information theoretically independent of the secret.*

A good secret sharing scheme should maximize its information rate.

Definition 2.3.5 (Information Rate)**.** *Given the set of all possible secrets S and the set of all possible shares T, the information rate ρ of a secret sharing scheme is $\rho = \log|S|/\log|T|$.*

Definition 2.3.6 (Ideal Secret Sharing Scheme)**.** *An* ideal *secret sharing scheme has information rate* 1*.*

A secret sharing scheme should also optimize the dealer efficiency and user efficiency; these are, respectively, how quickly the dealer computes shares and how quickly an authorized set of users reconstructs the secret. Simmons [93] distinguishes between *extrinsic* schemes where all users get shares from the same domain and *intrinsic* schemes where the domain of shares may differ. Secret sharing schemes are often used as part of other protocols. The more simple and direct a scheme is, the easier it is to use it as a building block.

2.3.2 Verifiable Secret Sharing

Verifiable secret sharing, introduced by Chor et al. [39], and formalized by Feldman and Micali [51], protects users against a malicious dealer. The dealer must publish some verification information that lets users check that their shares are consistent with the shares of other users. A verifiable secret sharing scheme must have the same propeties of correctness and privacy as a regular secret sharing scheme. However, we add the additional properties of completeness and binding.

Definition 2.3.7 (Verifiable Secret Sharing)**.** *A verifiable secret sharing scheme must be a secure secret sharing scheme (see Definition 2.3.4) and must also have the the following two properties:*

1. ***Completeness.*** *For all secrets, if the dealer follows the distribution protocol, and user u follows the verification protocol, then u accepts his share with probability* 1*.*

2. ***Binding.*** *Let k be a security parameter. Suppose a PPTM dealer distributes shares to all the users, using any process he wants. If A_1 and A_2 are two sets of authorized users that accept their shares as valid, then, when they reconstruct secrets s_1 and s_2, respectively, $Pr[s_1 \neq s_2] < 2^{-k}$.*

2.3.3 Threshold Secret Sharing

The most well known access structure is the threshold access structure introduced by Shamir [90] and Blakley [12].

Definition 2.3.8 (Threshold access structure)**.** *We say that Γ is a threshold access structure corresponding to threshold t if $\Gamma = \{A \subseteq \mathcal{U} : |A| \geq t\}$.*

Shamir [90] shows how to share a secret for a threshold access structure. Suppose the threshold is t. To share a secret $s \in \mathbb{F}_q$, the dealer chooses a sequence of values $a_1, a_2, \ldots, a_{t-1}$ at random and sets $a_0 = s$. These values define the polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i$. The share of a user with id u is $f(u)$. Any group of users A, such that $|A| > t$, can reconstruct the secret using Lagrange interpolation:

$$s = f(0) = \sum_{u \in A} f(u) \prod_{\substack{v \in A \\ v \neq u}} \frac{u}{u - v}$$

Let us briefly reinterpret Shamir's construction from the point of view of solving a system of linear equations. The sequence of values $a_0, a_1, \ldots, a_{t-1}$ constitute the t unknowns. Each user u learns a *linear* equation in terms of these variables, where the u^i constitute the known coefficients. If there are t users, then they possess t equations with t variables. The users can solve this system of equations if and only if the equations are linearly independent. Using the coefficients $1, u, u^2, \ldots, u^{t-1}$ ensures that the equations are linearly independent for *any* choice of user ids.

Theorem 2.3.9. *(Security [90]) The above secret scheme is a secure threshold secret sharing scheme.*

Efficiency: It takes $O(t)$ field operations to calculate the share of user u. Calculating s using Lagrange interpolation takes $O(t \log^2 t)$ field operations [90].

2.3.4 Conjunctive Multi-Level Secret Sharing

Tassa [95] introduced the conjunctive multi-level access structure. Each user is assigned to a level between 0 and n. Each level is associated with a threshold t_L; the thresholds form an increasing sequence $t_0 \leq t_1 \leq \cdots \leq t_n$. A group of users is authorized if, $\forall L : 0 \leq L \leq n$, there are at least t_L users at level L.

For example, suppose a university can decides that it takes a group of professors, grad students, and undergraduates to start a meeting. The policy can say that a quorum consists of at least twenty people, three of whom must be professors, and ten of whom must be either professors or grad students. We assign professors to level 0, grad students to level 1, and undergraduates to level 2. The policy can be expressed via the thresholds t_0, t_1, t_2 as follows:

Level	Type	Threshold
0	Professor	3
1	Grad Student	10
2	Undergraduate	20

This means a group of three professors, seven grad students and ten undergraduates are sufficient to start the meeting. A group of four professors, six grad students and ten undergraduates can also start the meeting. Even a group of ten professors and ten undergraduates can start the meeting. In fact, twenty professors are sufficient to start the meeting! The idea is that we need twenty people, and any person from a lower level can replace a person from a higher level.

Definition 2.3.10 (Conjunctive multi-level access structure)**.** *We say that Γ is a conjunctive multi-level access structure corresponding to a sequence of thresholds $t_0 < t_1 < \ldots < t_n$ and level assigning*

function $\mathcal{L}$ *if* $\Gamma = \{A \subseteq \mathcal{U} : \quad \forall L \in [0,n]$ *it holds that* $|\{u \in A : \mathcal{L}(u) \leq L\}| \geq t_L\}$

There are two conjunctive secret sharing schemes in the literature. The earliest, by Tassa [95] is based on the Birkhoff interpolation problem. The second, by Tassa and Dyn [96] interpolates polynomials in two variables. We focus on Tassa's construction [95] because it is more efficient (Tassa and Dyn [96] need to find points in general position) and is a basic building block for the results in Chapter 3.

We now describe Tassa's [95] conjunctive multi-level secret sharing scheme. Suppose the sequence of thresholds is $t_0 < t_1 < \ldots < t_n$, and let $t = t_n$. To share a secret $s \in \mathbb{F}_q$, the dealer chooses a sequence of values $a_1, a_2, \ldots, a_{t-1}$ and sets $a_0 = s$. These values define the polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i$. The share of user u at level $L = \mathcal{L}(u)$ is $f^{(t_{L-1})}(u)$. To reconstruct the secret, the users solve a system of linear equations to learn *all* of the a_i, including $s = a_0$.

Not all ids lead to a valid sharing of the secret. Tassa shows under what conditions derivatives of f are guaranteed to result in linearly independent equations:

Theorem 2.3.11 (Security with monotone ID allocation [95])**.** *Let* Γ *be a conjunctive access structure, with maximum threshold* t*, and let the underlying finite field be* $\mathbb{F}_q$*. Assume that the users in* $\mathcal{U}$ *were assigned ids in an increasing monotone manner, such that* $\forall u, v \in \mathcal{U} : u < v \Leftrightarrow \mathcal{L}(u) < \mathcal{L}(v)$*. Let* $N = \max\{u \in \mathcal{U}\}$*. Then the above secret sharing scheme is a perfect realization of* Γ *as long as* q *is big enough:*

$$q > 2^{-t} \cdot (t+1)^{(t+1)/2} \cdot N^{(t-1)t/2}$$

Theorem 2.3.12 (Security with random ID allocation [95])**.** *Let* Γ *be a conjunctive access structure, with maximum threshold* t*, and let the underlying finite field be* $\mathbb{F}_q$*. Assume a random allocation of user identities. For a randomly choosen* $A \subseteq \mathcal{U}$*, (1) if* $A \in \Gamma$*, then the probability that the shares given to* A *let it reconstruct the secret is at least* $1 - \nu(t,q)$ *and (2) if* $A \notin \Gamma$*, then the probability that the shares reveal no information about the secret in the information theoretic sense is at least* $1 - \nu(t,q)$*, where:*

$$\nu(t,q) = \frac{(t-2)(t-1)}{2(q-t)}$$

Remark. Theorem 2.3.12 states that a random allocation of user ids leads to a correct and private realization of Γ with high probability, if the size of the field $\mathbb{F}_q$ is exponential compared to the highest threshold t.

Efficiency: Calculating the share of a user u takes $O(t)$ field operations (basically, evaluating the polynomial f or its derivative). To interpolate the polynomial f and learn the secret requires solving a system of t linear equations with t variables; using Gaussian elimination, this takes $O(t^3)$ field operations.

We reinterpret Tassa's secret sharing scheme in terms of vector operations. Let $c : \mathbb{F}_q \to \mathbb{F}_q^t$ be defined as $c(x) = (1, x, x^2, \ldots, x^{t-1})$. We write $c^{(i)}(x)$ to denote the ith derivative of that vector. Taking the ith derivative of $c(x)$ zeroes out the first i elements. The share of user u at level L can be written as $c^{(t_{L-1})}(u) \cdot \mathbf{a}$, where $\mathbf{a} = (a_0, a_1, \ldots, a_{t-1})$. Thus, a user at level $L = 0$ gets information

about a_0, but a user at a higher level only learns a linear equation in terms of $(a_{t_{L-1}}, \ldots, a_{t-1})$ because the first t_{L-1} elements of $c(x)$ have been zeroed out.

An authorized set of users can reconstruct the secret by solving a system of linear equations. Suppose we have a set of m users $A(\mathcal{L}) = \{u_0, u_1, \ldots, u_m\}$. We put the users in order by level, so that $\mathcal{L}(u_i) \leq \mathcal{L}(u_j)$, for all $i < j$. We create a coefficient matrix $C_{A(\mathcal{L})}$ corresponding to A, where the ith row of $C_{A(\mathcal{L})}$ is the row vector $c^{(t_{L-1})}(u_i)$. Let σ be the vector of shares known by $A(\mathcal{L})$. To learn the secret, the users need to solve the equation $C_{A(\mathcal{L})} \cdot \mathbf{a} = \sigma$. Suppose $A(\mathcal{L})$ is a minterm of Γ. Tassa [95] shows that in that case, under certain conditions, $C_{A(\mathcal{L})}$ has a non-zero determinant, which means that the users can find a *unique* solution for $\mathbf{a}$ and recover the secret.

Corollary 2.3.13 (Non-zero determinant with monotone ID allocation [95], from the proof of Lemma 2)**.** *Let Γ be a conjunctive access structure, with maximum threshold t, and let the underlying finite field be $\mathbb{F}_q$. Assume that the users in $\mathcal{U}$ were assigned ids in an increasing monotone manner, such that $\forall u, v \in \mathcal{U} : u < v \Leftrightarrow \mathcal{L}(u) < \mathcal{L}(v)$. Let $N = \max\{u \in \mathcal{U}\}$. Furthermore, $2^{-t} \cdot (t+1)^{(t+1)/2} \cdot N^{(t-1)t/2} < q$. Then for every $A(\mathcal{L})$ that is a minterm of Γ, $\det(C_{A(\mathcal{L})}) \neq 0$*

Corollary 2.3.14 (Non-zero determinant with random ID allocation [95], from the proof of Theorem 3)**.** *Let Γ be a conjunctive access structure, with maximum threshold t, and let the underlying finite field be $\mathbb{F}_q$. Assume a random allocation of user identities. Then for every $A(\mathcal{L})$ that is a minterm of Γ, the probability that $\det(C_{A(\mathcal{L})}) \neq 0$ is at least $1 - \nu(t, q)$, where $\nu(t, q) = ((t-2)(t-1))/2(q-t)$.*

2.3.5 Disjunctive Multi-Level Secret Sharing

Simmons [93] introduced the disjunctive multi-level access structure. Each user is assigned to a level between 0 and n. Each level is associated with a threshold t_L; the thresholds form an increasing sequence $t_0 \leq t_1 \leq \cdots \leq t_n$. A group of users is authorized if, $\exists L : 0 \leq L \leq n$ such that there are at least t_L users at level L.

While a *conjuctive* access policy is expressed in terms of *and* clauses, the *disjunctive* policy is stated in terms of *or* clauses. Returning to our university forum example, the policy might state that either three professors, or ten professors and grad students, or twenty undergraduates, grad students and professors are sufficient to start the meeting. In other words, if there are less than three professors in the group, then there should be at least ten people in the group who are either professors or grad students. And if there are less than ten people who fall in that category, then the group must consist of at least twenty people. We assign the thresholds as follows:

Level	Type	Threshold
0	Professor	3
1	Grad Student	10
2	Undergraduate	20

Notice that the thresholds are identical to the ones in the conjunctive example in the previous section. However, the meaning is completely different. In the conjunctive example, *all* constraints had to be satisfied. In the disjunctive example, only one of them needs to be.

Definition 2.3.15 (Disjunctive multi-level access structure)**.** *We say that* Γ *is a disjunctive multi-level access structure corresponding to a sequence of thresholds* $t_0 < t_1 < \ldots < t_n$ *and level assigning function* $\mathcal{L}$ *if* $\Gamma = \{A \subseteq \mathcal{U} : \quad \exists L \in [0, n]$ *such that* $|\{u \in A : \mathcal{L}(u) \leq L\}| \geq t_L\}$

Simmons [93] constructs the earliest disjunctive secret sharing scheme in the literature. To share a secret s, the dealer constructs a sequence of nested hyperplanes $H_0 \subset H_1 \subset \cdots \subset H_n$. Hyperplane H_L is of dimension $t_L - 1$; it takes t_L points to interpolate H_L. The dealer publishes a line. The secret s is the intersection of the line and H_0. The share of user u at level L is a point in H_L. Unfortunately, the dealer cannot use arbitrary points. All of the points must be in general position: no three points may be collinear. Brickell [21] presents an algorithm for finding these points for access structures with a small number of levels; the dealer runs in exponential time in the number of levels and linear time in the size of the field. Since for most applications, the field size is typically exponential, this construction is too inefficient to be used in practice.

Brickell [21] constructs an ideal disjunctive secret sharing scheme, which Shoup [91] subsequently shows runs in *expected* polynomial-time. The dealer chooses a different polynomial for each level, and gives a user u at level L the point $f_L(u)$. By carefully selecting which ids u to use, Brickell ensures that any authorized set of users can reconstruct the secret. Brickell's solution allows the dealer to add new users dynamically. However, there are two major drawbacks to his scheme. First of all, it only works for secrets in $GF(q^\beta)$, where q is prime and β is a function of the number of levels in the access structure and the highest threshold. Thus the domain of the secret depends on the access structure. Secondly, Brickell's claimed result takes exponential time. The bottleneck occurs when the dealer must choose an irreducible polynomial over $GF(q)$. Using Shoup's guess-and-check algorithm [91], it takes an expected $O(\beta^2 \log \beta + \beta \log q)$ field operations to find this polynomial.

Ghodosi, Pieprzyk and Safavi-Naini [55] construct an ideal polynomial-time disjunctive secret sharing scheme that only works for small numbers of users. It is impossible to add new users to the system on the fly. Their algorithm is iterative. Let s be the secret. The dealer chooses a random polynomial f_0 of degree $t_0 - 1$, such that $f_0(0) = s$. The share of user u at level 0 is $f_0(u)$. For each subsequent level L, the dealer constructs a polynomial f_L of degree $t_L - 1$ such that (1) $f_L(0) = s$ and (2) $f_L(v) = f_{L-1}(v) = \ldots = f_0(v)$ for every user v at levels $0...L-1$.

The problem with this approach is that if there are $t_L - 1$ users at levels $0...L-1$, then there is *exactly one* polynomial of degree $t_L - 1$ that can satisfy the two requirements for f_L. That polynomial is f_{L-1}. However, the degree of f_{L-1} is $t_{L-1} - 1 < t_L - 1$. Thus, t_{L-1} users at level L would be able to learn the secret.

Ghodosi et al. overcome this problem by increasing the degree of f_L to some value $T_L - 1$. They call this a $(t_L, N_L)_{T_L}$ extension of the secret sharing scheme (where N_L is the total number of users on level L). As a result, for every level L, at least $T_L \geq t_L$ users of that level or higher are needed to reconstruct the secret. Privacy is preserved. Correctness is not. Ghodosi et al. show how to calculate the T_L as a function of the number of users and thresholds at previous levels.

To compute each f_L, the dealer needs to solve n systems of linear equations, with each system

having up to T_n variables. Using Gaussian elimination, this takes $O(nT_n^3)$ field operations. Reconstructing the secret using Lagrange interpolation takes $O(T_n^2)$ field operations; there is also an $O(T_n \log^2 T_n)$ algorithm [90].

Tassa [95] constructs an ideal disjunctive secret sharing scheme that can be computed in polynomial-time. Tassa shows that the dual of a disjunctive access structure Γ with thresholds t_L is a conjunctive access structure Γ^* with thresholds $t_L^* = |\{u : \mathcal{L}(u) \leq L\}| - t_L + 1$. Tassa [95] (also see Section 2.3.4 and Tassa and Dyn [96]) present two different polynomial-time ideal conjunctive secret sharing schemes. It is possible to use either one to create an ideal monotone span program for Γ^* (see [65] for definition of monotone span programs). Using Fehr's [50] transform on Γ^*, we can compute an ideal monotone span program that realizes Γ in $O(|\mathcal{U}|^3)$ field operations. We can then extract the share of each user from this program. Tassa's scheme is not dynamic, because the thresholds of Γ^* depend on $|\mathcal{U}|$.

2.4 E-Cash Background

E-cash allows users to anonymously pay for goods and services online. An e-coin consists of four parts: (1) a serial number uniquely identifying the e-coin, (2) a double-spending equation that lets the bank identify users who spend a coin with the same serial number more than once, (3) a zero-knowledge proof demonstrating that the bank signed this e-coin, and (4) some additional data about the transaction (e.g. merchant name). The e-coins are completely anonymous. Even though the bank signs each e-coin when a user withdraws it (this is not done anonymously), the bank cannot link any e-coin that a merchant deposits to any user. The bank cannot even link e-coins belonging to the same user to each other. The only exception is when a user spends a coin with the same serial number more than once. In this case, the user is "counterfeiting" e-coins, and the bank can use the double-spending equation to identify the user.

There are three phases to an e-cash transaction. In the first phase, the user withdraws a wallet of e-coins from the bank. The user authenticates himself with the bank. Then the user generates some e-coins, and the bank signs them (without learning what they are). The bank notes the withdrawal on the user's bank account, and logs some trace information for the transaction. In the second phase, the user makes a purchase. First, the user selects the e-coin he will use. Next, the user gives the e-coin to the merchant. The merchant verifies the zero-knowledge proof to ensure the e-coin is signed by the bank. In the third phase, the merchant deposits the e-coins he received from numerous transactions. The bank checks the zero-knowledge proofs (supplied by users) to ensure that the e-coins are valid. The bank also records the serial number and double-spending equation in a log. If a serial number appears in the log more than once, this means a user is spending the same e-coin multiple times.

2.4.1 Definition

We have three types of players: banks, users and merchants. Merchants are a subset of users. We generally use $\mathcal{B}$ to denote a bank, $\mathcal{M}$ to denote a merchant and $\mathcal{U}$ to denote a user. When we write $\mathsf{Protocol}(\mathcal{U}(x), \mathcal{B}(y))$ we mean that there is a protocol called $\mathsf{Protocol}$ between a user $\mathcal{U}$ and a bank $\mathcal{B}$ in which the private input of $\mathcal{U}$ is x and the private input of $\mathcal{B}$ is y.

Our definition of e-cash comes from Camenisch et al. [24]. An e-cash system contains the following set of protocols for transferring wallets and coins between players and for handling cheaters:

$\mathsf{BKeygen}(1^k, params)$ A bank $\mathcal{B}$ invokes $\mathsf{BKeygen}$ to generate $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$, its public/private-key pair.

$\mathsf{UKeygen}(1^k, params)$ A user $\mathcal{U}$ (or a merchant $\mathcal{M}$) invokes $\mathsf{UKeygen}$ to generate $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$, its public/private-key pair.

$\mathsf{Withdraw}(\mathcal{U}(pk_{\mathcal{B}}, sk_{\mathcal{U}}, n), \mathcal{B}(pk_{\mathcal{U}}, sk_{\mathcal{B}}, n))$ This is a protocol between a user $\mathcal{U}$ and a bank $\mathcal{B}$ that lets the user withdraw n coins from his bank account. The user gets either a wallet W of n coins, or an error message. The bank gets either some trace information that it stores in a database, or an error message.

$\mathsf{Spend}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ This is a protocol between a user $\mathcal{U}$ and a merchant $\mathcal{M}$ that transfers one coin from the user's wallet W to the merchant. The merchant gets an e-coin *coin* and the user updates his wallet to contain one less coin.

$\mathsf{Deposit}(\mathcal{M}(sk_{\mathcal{M}}, coin, pk_{\mathcal{B}}), \mathcal{B}(pk_{\mathcal{M}}, sk_{\mathcal{B}}))$ This is a protocol between a merchant $\mathcal{M}$ and a bank $\mathcal{B}$ that lets the merchant deposit a coin he got from a customer into his bank account.

$\mathsf{PublicSecurityProtocols}(protocol, params, arglist)$ This is a set of functions that can be invoked by anybody to identify double spenders and verify their guilt. The bank *finds* double-spenders, but it must be able to *convince* everyone else. The Camenisch et al. protocols [24] include $\mathsf{Identify}(params, coin1, coin2)$ to identify a double spender, $\mathsf{VerifyGuilt}(params, coin, pk_{\mathcal{U}}, proof)$ to publicly verify that user $\mathcal{U}$ had double spent a coin, $\mathsf{Trace}(params, coin, pk_{\mathcal{U}}, proof, database)$ to find all coins spent by a guilty user, $\mathsf{VerifyOwnership}\ (params, coin, proof, pk_{\mathcal{U}})$ to verify that a guilty user spent a particular coin. The exact set of functions depends on the e-cash system and its desired security properties.

A secure e-cash system must maintain the following four properties:

Correctness. If an honest user runs Withdraw with an honest bank, then neither outputs error; if an honest user runs Spend with an honest merchant, then the merchant accepts the e-coin. If an honest merchant runs Deposit with an honest bank on an e-coin given to him by an honest user, then the bank will accept the e-coin.

Anonymity. Even if a malicious bank conspires with one or more malicious merchants, the bank cannot link a user to any coins he spends. We create a simulator $\mathcal{S}$ and give it special powers (e.g. control of random oracle, ability to generate common parameters, control of key generation). The simulator should be able to run the Spend protocol without knowing any information about any user's wallet or public/secret-key pair.

Formally, we create an adversary $\mathcal{A}$ that plays the part of the bank and of all merchants. $\mathcal{A}$ creates the bank's public-key $pk_{\mathcal{B}}$. Then, $\mathcal{A}$ gets access to an interface Game that plays either the real or ideal game; $\mathcal{A}$ must determine which. $\mathcal{A}$ can make four types of queries to Game:

$\mathsf{GameSetup}(1^k)$ generates the public parameters *params* and private parameters *auxsim* for the simulator $\mathcal{S}$.

$\mathsf{GameGetPK}(i)$ returns the public-key of user $\mathcal{U}_i$, generated by $\mathsf{UKeygen}(1^k, params)$.

$\mathsf{GameWithdraw}(i)$ runs Withdraw protocol user $\mathcal{U}_i$: $\mathsf{Withdraw}(\mathcal{U}_i(pk_{\mathcal{B}}, sk_i, n), \mathcal{A}(state, n))$. (We use *state* to denote the state of the adversary; it is updated throughout the course of protocol execution). We call W_j the wallet generated the jth time protocol Withdraw is run.

$\mathsf{GameSpend}(j)$ in the real game, this runs the spend protocol with the user $\mathcal{U}$ that holds the wallet W_j: $\mathsf{Spend}(\mathcal{U}(W_j), \mathcal{A}(state, n))$. In the ideal game, $\mathcal{S}$ pretends to be the user:

$\mathsf{Spend}(\mathcal{S}(params, auxsim, pk_{\mathcal{B}}), \mathcal{A}(state, n))$; $\mathcal{S}$ does not have access to the wallet W_j or know who owns it.

An adversary is *legal* if it never asks a user to double-spend a coin: for all j, the adversary never calls $\mathsf{GameSpend}(j)$ more than n times (where n is the size of the wallet). An e-cash scheme *preserves anonymity* if, for all $pk_{\mathcal{B}}$, no computationally bounded *legal* adversary can distinguish between the real game and the ideal game with more than negligible probability.

Balance. No group of dishonest users and merchants should be able to deposit more coins than they withdraw. We assume that each coin has a serial number (generated during the Withdraw protocol) We create a knowledge extractor $\mathcal{E}$ that executes the Withdraw protocol with u dishonest users and generates un coin serial numbers: $S_1, \ldots, S_{un}$ (we assume each user withdraws n coins). No adversary should be able to successfully deposit a coin with serial number S unless $S \in \{S_1, \ldots, S_{un}\}$. Again, $\mathcal{E}$ must have additional powers, such as control of the random oracle or special knowledge about public parameters.

Culpability and Exculpability. Any user that runs Spend twice on the same coin should be caught by the bank; however, a malicious bank should not be able to conspire with malicious merchants to frame an honest user for double-spending. We omit the specifics of these definitions and refer the reader to Camenisch et al. [24].

Definition 2.4.1 (Secure e-cash system)**.** *We say that an e-cash system is secure if it maintains the correctness, anonymity, balance, and culpability and exculpability properties.*

2.4.2 Compact E-Cash

We overview the CHL [24] compact e-cash scheme. A user has a secret-key $u \in \mathbb{Z}_q$ and public-key g^u. To withdraw n coins, the user randomly chooses $s, t \in \mathbb{Z}_q$ and obtains from the bank a CL-signature σ on (u, s, t) (see Section 2.2.8 for reference). Now the user has a wallet of n coins: $(0, u, s, t, \sigma), \ldots, (n-1, u, s, t, \sigma)$.

To pay a merchant, the user constructs an e-coin (S, T, Φ, R) from the wallet coin (J, u, s, t, σ) (see Algorithm 2.4.1. S is a unique (with high probability) serial number. T and R are needed to trace double-spenders — knowing two different (T, R) values corresponding to the same wallet coin lets the bank learn the user's identity. Φ is a zero-knowledge proof that tells the merchant and bank that the e-coin is valid. Finally, R is as hash of the contract between the user and merchant and should be unique to every transaction (this lets the bank use (T, R) to catch double-spenders).

To deposit and e-coin, the merchant gives (S, T, Φ, R) to the bank, along with his public-key. The bank checks whether it has already seen a coin with serial number S – if yes, then the bank knows that *somebody* is trying to double-spend because S is supposed to be unique. If it has seen (S, R) before, then the merchant is at fault because R is unique to every transaction. If the bank hasn't seen (S, R) before, then the user is at fault and the bank uses the values $(S, T_{old}, \Phi_{old}, R_{old})$ and (S, T, Φ, R) to learn the double-spending user's identity. CHL finds double-spenders in a manner similar to Chaum

et al. [38], but it only learns the user's public-key, and *not* his secret-key (Camenisch et al's extended solution also reveals the secret-key).

Global parameters: Let k be the security parameter. All computation is done in a group $\mathbb{G}$, of prime order $q = \Theta(2^k)$, with generator g. We assume there is a public-key infrastructure.

Spend lets a user $\mathcal{U}$ pay a merchant $\mathcal{M}$ the wallet coin (J, u, s, t, σ): First, the user and merchant agree on a contract *contract* (we assume each contract is unique per merchant). The merchant gives the user his public key $pk_{\mathcal{M}}$. Then, the user runs CalcCoin, as defined in Algorithm 2.4.1, to create the coin (S, T, Φ, R) and sends it to the merchant. Finally, the merchant verifies Φ to check the validity of the coin (S, T, Φ, R).

Algorithm 2.4.1: CalcCoin

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key, $contract \in \{0,1\}^*$
User Data: u private key, g^u public key, (s, t, σ, J) a wallet coin
$R \leftarrow H(pk_{\mathcal{M}}||info)$;
$S \leftarrow \mathsf{DY}_{g,s}(J)$;
$T \leftarrow g^u \cdot \mathsf{DY}_{g,s}(J)^R$;
Calculate ZKPOK Φ of (J, u, s, t, σ) such that:
  $0 \leq J < n$
  $S = \mathsf{DY}_{g,s}(J)$
  $T = g^u \cdot \mathsf{DY}_{g,s}(J)^R$
  $\text{VerifySig}(pk_{\mathcal{B}}, (u, s, t), \sigma) = true$
$\mathsf{DY}_{g,s}(J)$ `Dodis-Yampolskiy pseudorandom function (see Section 2.2.5) and` H `is a collision-resistant hash function.`
return (S, T, Φ, R)

Theorem 2.4.2. *CHL is a secure e-cash scheme, given the security of the following three building blocks:*

1. *CL-signatures, which depend on the Strong RSA Assumption.*
2. *Zero-knowledge proof (or argument) system, which relies on the Strong RSA Assumption and the Random Oracle Model.*
3. *Collision-resistant hash functions.*
4. *Dodis-Yampolskiy pseudorandom function, which for the case of polynomial size input depends on the q-DHI and q-DBDHI assumptions.*

Efficiency. 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.

2.4.3 Fair Exchange

Suppose Alice wishes to purchase some on-line goods from Bob. Alice wants to make sure that she doesn't give away her money unless she actually gets the goods. Bob wants to make sure that

he doesn't give away the goods without getting paid. This is a well-known problem called *fair exchange* [42, 76, 4, 5]. In optimistic fair exchange [76, 5], fairness is ensured by the existence of a trusted third party (TTP) who intervenes only if one of the players claims that something went wrong.

Most of the fair exchange literature focuses on exchanging digital signatures, or on exchanging a digital signature for digital goods or services. There have been several attempts to realize fair exchange of e-cash. Jakobsson's [61] and Reiter, Wang and Wright's [86] schemes' are not fair to the user: the user is not allowed to reuse a coin from a failed exchange. Asokan, Shoup and Waidner [5] show how to exchange Brands' e-cash [20], but their protocol is not quite fair to the user either: if an exchange fails, a user may reuse the coin, but he cannot do so unlinkably. This weakness in all three schemes cannot be solved by a trusted third party. At early stages of the fair exchange protocol, the merchant can ask the trusted third party to terminate the exchange; however, the user would have already revealed too much information about his coin. This is a serious problem because on-line multi-party protocols often fail due to network delay, computers crashing, and etc.

2.4.4 Anonymous Authentication

E-cash can be used for anonymous authentication. Instead of buying goods and services, an e-coin can be used as proof that user is authorized to perform some action.

The most obvious application is for authentication policies that allow a user to perform some action at most n times. In this case, the user would withdraw a wallet of n e-coins and use an e-coin to authenticate. If the user authenticates more than n times, then by the pidgeon-hole principle, he must have double-spent one of the e-coins and the bank can discover his identity. This approach is used by [97, 98, 82], with some variant of e-cash. A recent variation on the theme is Jarecki and Shmatikov's [62] work on anonymous, but *linkable*, authentication where one's identity can be discovered after one carries out too many transactions.

Finally, Damgård et al. [44] relax the double-spending condition slightly. They allow a user to authenticate once per time interval. However, instead of using e-cash, they use general zero-knowledge techniques to achieve their results.

Chapter 3

Disjunctive Multi-Level Secret Sharing

In this chapter, we present a new disjunctive multi-level secret sharing scheme. Our scheme is the most dealer efficient scheme in the literature; it takes $O(|\mathcal{U}|t)$ field operations to calculate the shares of $|\mathcal{U}|$ users, where t is the highest threshold. The only other polynomial time construction is Tassa [95], which takes $O(|\mathcal{U}|^3 t)$ field operationsto generate shares for all users. Ours is also the first polynomial time disjunctive secret sharing scheme that allows the dealer to add users dynamically. See Section 2.3.5 for a survey of the literature on this problem. Our constructions is very simple: the share of a user is simply a point on a randomly chosen polynomial or its derivative. In Section 3.1 we present our construction and prove it is secure. In Section 3.2 we show how to transform it into a verifiable secret sharing scheme.

The results in this chapter are based on "Disjunctive Multi-Level Secret Sharing," [75].

3.1 Fast Disjunctive Multi-Level Secret Sharing Scheme

We present our new disjunctive multi-level secret sharing scheme. Our scheme is very similar to Tassa's [95] conjunctive secret sharing scheme (see Section 2.3.4 for overview). In both schemes, users get a point on either a polynomial or its derivative. In Tassa's conjunctive scheme, the secret is stored in the lowest order coefficient, while in our scheme, it is stored in the highest order coefficient. We prove security by reducing our disjunctive scheme to Tassa's conjunctive secret sharing scheme.

3.1.1 Construction

Let $t_0 < t_1 < \cdots < t_n$ be a sequence of increasing thresholds. The share of user u at level L is a linear equation of t_L variables, one of which is the secret. We choose the coefficients for these equations using Tassa's technique of taking derivatives: user u at level L will receive $f^{(t-t_L)}(u)$, where t is the highest threshold. Using a reduction to conjunctive multi-level secret sharing, we will show that any authorized set of users learns a sufficient number of linearly independent equations to reconstruct the secret.

Construction 3.1.1. *Suppose the sequence of thresholds is $t_0 < t_1 < \ldots < t_n$, and let $t = t_n$. To share a secret $s \in \mathbb{F}_q$, the dealer chooses a random sequence of values $a_0, a_1, \ldots, a_{t-2}$ and sets $a_{t-1} = s$. The sequence of values defines the polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i$. The share of user u at level $L = \mathcal{L}(u)$ is $f^{(t-t_L)}(u)$. The ids u can be chosen either at random or in monotonically decreasing order. Any authorized set of users can solve the system of linear equations to learn $s = a_{t-1}$.*

We reinterpret our disjunctive scheme in terms of vector operations. Let $d : \mathbb{F}_q \to \mathbb{F}_q^t$ be defined as $d(x) = (1, x, x^2, \ldots, x^{t-1})$. We write $d^{(i)}(x)$ to denote the ith derivative of that vector. The share of user u at level L can be written as $d^{(t-t_L)}(u) \cdot \mathbf{a}$, where $\mathbf{a} = (a_0, a_1, \ldots, a_{t-1})$. Suppose we have a set of m users $A(\mathcal{L})$ and the highest level of user is M. The users might not be able to recover the entire vector $\mathbf{a}$ because they would have information only about $(a_{t-t_M}, \ldots, a_{t-1})$ (because taking the derivative zeroes-out the other coefficients of d). Let $\mathbf{a}_M$ and $d_M^{(i)}$ be the $t - t_M$

leftmost coefficients of those respective vectors. In this case, each user $u \in A(\mathcal{L})$ at level L can write his share as $d_M^{(t-t_L)}(u) \cdot \mathbf{a}_M$. We order the users in $A(\mathcal{L}) = \{u_0, u_1, \ldots, u_m\}$ by level, so that $\mathcal{L}(u_i) \geq \mathcal{L}(u_j)$ for all $i < j$. We can then create the coefficient matrix $D_{A(\mathcal{L})}$, where the ith row of $D_{A(\mathcal{L})}$ is $d_M^{(t-t_L)}(u_i)$. Users in $A(\mathcal{L})$ can try to recover $\mathbf{a}_M$ by solving $D_{A(\mathcal{L})} \cdot \mathbf{a}_M = \sigma$, where σ is a vector of their of shares.

3.1.2 Efficiency

To calculate a single share, the dealer must compute a point on the derivative of f. This takes $O(t)$ field operations per user. Reconstructing the secret requires solving a system of t linear equations; using Gaussian elimination, this would take $O(t^3)$ field operations.

3.1.3 Security

Before proving the security of the construction in Section 3.1.1, we first need to prove few interesting claims that show the link between disjunctive and conjunctive access structures and secret sharing schemes.

Let Γ be a disjunctive multi-level access structure with thresholds $t_0 < t_1 < \ldots < t_n$; we set $t = t_n$. Let $\mathcal{L}$ be some level assignment function. We take an arbitrary set of users $A \subseteq \mathcal{U}$; let M be the highest level of user in $A(\mathcal{L})$. We define $\mathcal{L}'$ as: $\mathcal{L}'(u) = M - \mathcal{L}(u)$. We also define the conjunctive access structure Γ' with thresholds $t'_0 < t'_1 < \ldots < t'_M$, where $t'_M = t_M$ and $\forall 0 \leq L < M : t'_L = t_M - t_{M-L-1}$.

We give some intuition about the above transformation. Suppose $A(\mathcal{L})$ is a minterm of Γ. For every user at level M in $A(\mathcal{L})$, there is a user at level 0 in $A(\mathcal{L}')$. We can calculate a lower bound on the number of users in $A(\mathcal{L})$ at level M; if there is not enough, there will be t_{M-1} users at levels $0...M-1$, thus contradicting the fact that $A(\mathcal{L})$ is a minterm. As a result, we can calculate a lower bound on the number of users at level 0 in $A(\mathcal{L}')$. We can do the same for every level. Due to this property, we were able to chose the thresholds for Γ' in such a way as to ensure that $A(\mathcal{L}')$ is a minterm of Γ'. More importantly, the share that a user u receives for Γ is algebraically related to the share that the user receives for Γ'. We now prove these claims formally:

Claim 3.1.2. *If $A(\mathcal{L})$ is a minterm of Γ then $A(\mathcal{L}')$ is a minterm of Γ'.*

Proof. Let $A(\mathcal{L})$ be a minterm of Γ. We need to show that there are at least $t'_M = t_M$ users in $A(\mathcal{L}')$ at levels $0...M$ and $t'_L = t_M - t_{M-L-1}$ users at levels $0...L$ for all $L < M$. Let us begin with level M. We know that $|A| = t_M$ because $A(\mathcal{L})$ is a minterm of Γ. This means that there are exactly $t'_M = t_M$ users in $A(\mathcal{L}')$ at levels $0...M$. The case for levels $L < M$ is also straightforward. If there is a set $B \subset A$ such that $B(\mathcal{L})$ had t_{M-L-1} users at levels $0...M-L-1$ then $B(\mathcal{L}) \in \Gamma$, thus contradicting the fact that $A(\mathcal{L})$ is a minterm of Γ. Therefore, there are at least $t_M - t_{M-L-1} + 1$ users at levels $M-L...M$ in $A(\mathcal{L})$. Applying $\mathcal{L}'$, we see that there are at least $t_M - t_{M-L-1} + 1$ users at levels $0...L$ in $A(\mathcal{L}')$. Ergo, there are at least $t_M - t_{M-L-1}$ users at levels $0...L$ in $A(\mathcal{L}')$. Thus, we have

shown that $A(\mathcal{L}') \in \Gamma'$. $A(\mathcal{L}')$ must be a minterm of Γ because any subset of $A(\mathcal{L}')$ would have less than t'_M users at levels $0...M$, so it would be unauthorized. □

Claim 3.1.3. *If $A(\mathcal{L}) \notin \Gamma$ then $A(\mathcal{L}') \notin \Gamma'$.*

Proof. Suppose $A(\mathcal{L}) \notin \Gamma$. If M is the highest level of user in $A(\mathcal{L})$, then $|A| < t_M = t'_M$. Since Γ' is a conjunctive access structure, $A(\mathcal{L}') \notin \Gamma'$. □

Recall the pairwise multiplication operator $\vec{a} \diamond \vec{b} = (a_0 b_0, \ldots, a_n b_n)$.

Claim 3.1.4. *Let $A(\mathcal{L})$ be a minterm of Γ, and let Γ' be the corresponding conjunctive access structure. If the dealer for Γ and the dealer for Γ' pick the same secret values, then there exists a positive vector $\mathbf{b}$ such that: $D_{A(\mathcal{L})} \cdot \mathbf{a}_M = C_{A(\mathcal{L}')} \cdot (\mathbf{b} \diamond \mathbf{a}_M)$.*

Proof. Let $A(\mathcal{L})$ be a minterm of Γ. Let M be the highest level of user in A. The share of user u at level L according to Γ' is $d_M^{(t-t_L)}(u) \cdot \mathbf{a}_M = d_M^{((t-t_M)+(t_M-t_L))}(u) \cdot \mathbf{a}_M$. We can relate $d_M^{(t-t_M)}$ to c (the vector used to calculate shares in Γ') as follows:

$$\begin{aligned} d_M^{(t-t_M)} &= (x^{t-t_M}, \ldots, x^{t-1})^{(t-t_M)} \\ &= (1, \ldots, x^{t_M-1}) \diamond \mathbf{b} \\ &= c \diamond \mathbf{b} \end{aligned}$$

In the above equations, $\mathbf{b}$ is the vector of coefficients that result from calculating a derivative; all of its entries are positive. Using this result, we get that the share of u in Γ is equal to:

$$\begin{aligned} d_M^{(t-t_L)}(u) \cdot \mathbf{a}_M &= d_M^{((t-t_M)+(t_M-t_L))}(u) \cdot \mathbf{a}_M \\ &= (c^{t_M-t_L}(u) \diamond \mathbf{b}) \cdot \mathbf{a}_M \\ &= c^{t_{L'}-1}(u) \cdot (\mathbf{b} \diamond \mathbf{a}_M) \end{aligned}$$

Therefore, if the dealer for Γ' chooses the same vector of secret values as the dealer for Γ, we would have the relation: $D_{A(\mathcal{L})} \cdot \mathbf{a}_M = C_{A(\mathcal{L})} \cdot (\mathbf{b} \diamond \mathbf{a}_M)$. □

Now we will show that in order to prove privacy and correctness, it is sufficient to prove that for any minterm $A(\mathcal{L}) \in \Gamma$, $\det(D_{A(\mathcal{L})}) \neq 0$.

Claim 3.1.5. *If for every minterm $A(\mathcal{L}) \in \Gamma$ it holds that $\det(D_{A(\mathcal{L})}) \neq 0$, then the secret sharing scheme in Construction 3.1.1 is correct.*

Proof. Assume every minterm of Γ has an associated coefficient matrix with non-zero determinant. Let $A(\mathcal{L})$ be a minterm of Γ. Let M be the highest level of user in $A(\mathcal{L})$ and let σ be a vector of shares owned by $A(\mathcal{L})$. We know that $D_{A(\mathcal{L})} \cdot \mathbf{a}_M = \sigma$. Since $A(\mathcal{L})$ is a minterm, $\det(D_{A(\mathcal{L})}) \neq 0$. This means that we can calculate the inverse of $D_{A(\mathcal{L})}$ and learn $\mathbf{a}_M$, which includes the secret $s = a_{t-1}$. Every authorized set of users contains at least one minterm as a subset. The authorized users can recover the secret using the minterm. □

Claim 3.1.6. *If for every minterm $A(\mathcal{L}) \in \Gamma$ it holds that $\det(D_{A(\mathcal{L})}) \neq 0$, then the secret sharing scheme in Construction 3.1.1 preserves privacy.*

Proof. Assume every minterm in Γ has a corresponding coefficient matrix with a non-zero determinant. We will use the phantom user technique introduced by Tassa [95] to prove Claim 3.1.6. We introduce a phantom user $u_0 \in \mathcal{U}$ and set $\mathcal{L}(u_0) = 0$. No real user will ever get the share assigned to user u_0.

Fix some $A(\mathcal{L}) \notin \Gamma$, and let M be the highest level of user in $A(\mathcal{L})$. For now, assume that $|A| = t_M - 1$. If we let $A_0 = A + \{u_0\}$, then A_0 has t_M users at levels $0...M$, so $A_0(\mathcal{L}) \in \Gamma$. We will show that users A_0 can recover the entire vector $\mathbf{a}_M$. The users can recover $\mathbf{a}_M$ only if the equation $D_{A_0} \cdot \mathbf{a}_M = \sigma$ has a unique solution, which is the case if and only if $\det(D_{A_0}) \neq 0$. This means that the row in D_{A_0} corresponding to u_0 is independent of the rows fcorresponding to users in A. Since u_0 is at the lowest level, it is on the bottom row of D_{A_0}. Therefore, $(0, \ldots, 0, 1) \notin$ row-space(D_A). Thus, the secret $s = a_{t-1}$ is information theoretically independent of the view of A.

Suppose $A_0(\mathcal{L})$ is a minterm. Then, the corresponding matrix D_{A_0} has a non-zero determinant. Let σ be the shares of A_0. We can find a unique solution to the equation $D_{A_0} \cdot \mathbf{a}_M = \sigma$ and learn the entire vector $\mathbf{a}_M$. This means that the secret is independent of the view of A.

However, A_0 might *not* be a minterm of Γ. This is because the addition of u_0 might create a set of t_L users at levels $0...L$, where $L < M$. Therefore, we divide A_0 into two sets of users: A_{low} and A_{high}. A_{low} contains all users at levels $0...L$, while A_{high} contains all users at levels $L+1...M$. A_{low} is a minterm of Γ. (If A_{low} was not a minterm, then we could remove a user from A_{low} and still have t_L users at levels $0...L$. This means we can remove u_0 from A_{low} and still have t_L users at levels $0...L$. In this case, A would be authorized, which is a contradiction.) We divide the vector of shares σ into σ_{low} and σ_{high} in a similar fashion. Finally, we take the vector of unknown secret values $\mathbf{a}_M$ and divide it into $\mathbf{a}_{low}$ and $\mathbf{a}_{high}$.

We now show that the users in A_0 can solve $D_{A_0} \cdot \mathbf{a} = \sigma$. D_{A_0} is a $t_M \times t_M$ matrix. The bottom t_L rows consist of $t_M - t_L$ columns of zeroes on the left, followed by $D_{A_{low}}$. The top $t_M - t_L$ rows consist of $D_{A_{high}}$. (We draw a diagram of D_{A_0} on the next page). To solve for $\mathbf{a}_M$, we create the augmented matrix $D_{A_0}|\sigma$. Then we perform the following two operations:

Step 1: We perform Gaussian elimination on the bottom rows corresponding to A_{low} to learn $\mathbf{a}_{low}$. We can do this because A_{low} is a minterm of Γ, and therefore, the determinant of $D_{A_{low}}$ is non-zero. Gaussian elimination will result in the identity submatrix in the rightmost t_L columns of those rows.

Step 2: Next, we use the bottom t_L rows of D_{A_0} to completely zero out the rightmost t_L columns of D_{A_0}. This leaves the leftmost $t_M - t_L$ columns untouched, but changes σ_{high} to some σ_1.

Graphically, these two steps result in the following transformation:

$$D_{A_0}|\sigma = \left(\begin{array}{cc|c} D_{A_{high}} & & \sigma_{high} \\ 0 & D_{A_{low}} & \sigma_{low} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} D_{A_{high}} & & \sigma_{high} \\ 0 & I & \mathbf{a}_{low} \end{array}\right) \rightarrow \left(\begin{array}{cc|c} D_{A_1} & 0 & \sigma_1 \\ 0 & I & \mathbf{a}_{low} \end{array}\right)$$

Consider the $(t_M - t_L)$ x $(t_M - t_L)$ matrix D_{A_1}. We get the equation $D_{A_1} \cdot \mathbf{a}_{high} = \sigma_1$. The vector σ_1 is whatever results when we zero out the rightmost columns. The matrix D_{A_1} is an abridgement of the rows corresponding to $D_{A_{high}}$. Essentially, we have transformed the shares of A_{high} from access structure Γ to the shares of some set of users A_1 from some other disjunctive access structure Γ_1. Due to the equation $D_{A_1} \cdot \mathbf{a}_{high} = \sigma_1$, we know that the secret values chosen by the dealer for Γ_1 are $\mathbf{a}_{high}$.

It is easy to see that A_1 is in Γ_1. The users in A_1 are assigned to levels $J...K$ via some (unknown) labeling function $\mathcal{L}_1$. The lowest row of D_{A_1} represents the share of the user at level K. Since D_{A_1} is a $(t_M - t_L) \times (t_M - t_L)$ square matrix, we know the threshold for level K is $t_M - t_L$. Since $|A_1| = t_M - t_L$, $A_1(\mathcal{L}_1) \in \Gamma_1$.

If $A_1(\mathcal{L}_1)$ is a minterm of Γ_1, then $\det(D_{A_1}) \neq 0$ and we can solve for $\mathbf{a}_{high}$. If $A_1(\mathcal{L}_1)$ is not a minterm of Γ_1, then we can keep repeating the reduction we performed on A_0 until we have solved for the entire secret vector $\mathbf{a}_{high}$. Since we can solve for $\mathbf{a}_{high}$ and $\mathbf{a}_{low}$, this means we have recovered the entire vector $\mathbf{a}_M$. As stated earlier, this means the secret is information theoretically independent of view of the users in A.

Finally, we have to consider the possibility that $|A| \neq t_M - 1$. If $|A| > t_M - 1$, then $A(\mathcal{L}) \in \Gamma$. If $|A| < t_M - 1$, we can always augment it until it does have $t_M - 1$ users at levels $0...M$. However, the view of this augmented set of users will still be information theoretically independent of the secret. Therefore, the view of A is also independent of the secret. □

We are now ready to prove that our secret sharing scheme is secure.

Theorem 3.1.7. *(Security with monotone id allocation) Let Γ be a disjunctive access structure, with maximum threshold t, and let the underlying finite field be $\mathbb{F}_q$. Assume that the users are assigned ids in a decreasing monotone manner, such that $\forall u, v \in \mathcal{U} : u > v \Leftrightarrow \mathcal{L}(u) < \mathcal{L}(v)$. Let $N = \max\{u \in \mathcal{U}\}$. Then the secret sharing scheme in Construction 3.1.1 is a perfect realization of Γ, as long as:*

$$2^{-t} \cdot (t+1)^{(t+1)/2} \cdot N^{(t-1)t/2} < q$$

Proof of Theorem 3.1.7. Let Γ be a disjunctive secret sharing scheme. By Claims 3.1.5 and 3.1.6, to prove privacy and correctness, all we need to show is that for all minterms $A(\mathcal{L}) \in \Gamma$, $\det(D_{A(\mathcal{L})}) \neq 0$.

Let $A(\mathcal{L})$ be a minterm of Γ, and let Γ' be the corresponding disjunctive access structure. By Claim 3.1.4, if the dealer for Γ and the dealer for Γ' pick the same secret values, then there exists a positive vector $\mathbf{b}$ such that: $D_{A(\mathcal{L})} \cdot \mathbf{a}_M = C_{A(\mathcal{L}')} \cdot (\mathbf{b} \diamond \mathbf{a}_M)$. If $C_{A(\mathcal{L}')}$ has a non-zero determinant, then we can solve for $\mathbf{b} \diamond \mathbf{a}_M$. Since $\mathbf{b}$ is a constant positive vector whose values depend solely on the access structure Γ (recall that $\mathbf{b}$ is the coefficients of a derivative), we can recover $\mathbf{a}_M$. Since this is the case, $D_{a(\mathcal{L})}$ must also have a non-zero determinant.

Therefore, $D_{A(\mathcal{L})}$ has a non-zero determinant if $C_{A(\mathcal{L}')}$ has a non-zero determinant, which occurs if all the conditions in Corollary 2.3.13 hold. We have to ensure that $A(\mathcal{L}')$ is a minterm of Γ', the ids of the users increase monotonically in $\mathcal{L}'$ and that the field is large enough. Since $A(\mathcal{L})$ is a minterm of Γ, by Claim 3.1.2, $A(\mathcal{L}')$ is a minterm of Γ'. Monotonicity is easy; if we assign

users ids in monotonically decreasing order in terms of $\mathcal{L}$, they will be in monotonically increasing order in terms of every possible $\mathcal{L}'$. As far as field size, Tassa's scheme expresses it in terms of N, the highest possible id of user in the system, and t', the highest threshold associated with Γ'. The highest threshold in any Γ' is $t'_M = t_M$ where M is the highest level of authorized user. Therefore, the highest t' is simply t, the highest threshold for Γ. Thus, the lower bound on the size of the field is the same as in Corollary 2.3.13. □

Theorem 3.1.8. *(Security with random id allocation) Let Γ be a disjunctive access structure, with maximum threshold t, and let the underlying finite field be $\mathbb{F}_q$. Assume a random allocation of user identities. For a randomly choosen $A \subseteq \mathcal{U}$, (1) if $A \in \Gamma$, then the probability that the shares given to A let it reconstruct the secret is at least $1 - \nu(t, q)$ and (2) if $A \notin \Gamma$, then the probability that the shares reveal no information about the secret in the information theoretic sense is at leat $1 - \nu(t, q)$, where:*

$$\nu(t, q) = \frac{(t-2)(t-1)}{2(q-t)}$$

Proof of Theorem 3.1.8. The privacy and correctness proof is essentially the same. Once again, we need to prove that $C_{A(\mathcal{L}')}$ has a non-zero determinant. This is true if the the conditions of Corollary 2.3.14 hold. We've already shown that $A(\mathcal{L}')$ is a minterm of Γ'. Id selection is easy: if we assign ids to all users at random in the disjunctive scheme, then they are equally random as far as Γ' is concerned. We know that the determinant is non-zero with probability at least $1 - \nu(t', q')$. If we take the lower bound for $1 - \nu(t', q')$, for every possible Γ' that arises from an authorized set $A(\mathcal{L}) \in \Gamma$, we will have a lower bound on the probability that users in A can reconstruct the secret. We know that the underlying field size q is the same in all cases. By Corollary 2.3.14, we see that $1 - \nu(t', q')$ is lower when t' is higher. The highest value for t' is simply t. Therefore, with random id allocation, our disjunctive construction is correct with probability $1 - \nu(t, q)$. □

3.2 Verifiable Secret Sharing

3.2.1 Construction

We use standard techniques developed by Pedersen [84] to transform our disjunctive secret sharing scheme into a verifiable secret sharing scheme. The dealer will give users a point on the polynomial $f \in \mathbb{F}_q[x]$ as before. The dealer will also publish a Pedersen commitment [84] to each of the coefficients of f. The randomness for committing coefficient a_i of polynomial f will come from the coefficient b_i of a randomly chosen polynomial $g \in \mathbb{F}_q[x]$. If the user's share of the secret is $f^{(d)}(u)$, then the user will get auxiliary verification value $g^{(d)}(u)$. Due to the algebraic properties of Pedersen commitments, $\mathsf{PedCom}(f^{(d)}(u); g^{(d)}(u))$ is a function of the commitments $\mathsf{PedCom}(a_i; b_i)$. Each user will verify that his share is consistent with the values the dealer published.

We now give the scheme in detail. In the setup phase, some trusted third party chooses generators h_1, h_2 of some finite field $\mathbb{F}_q$ of prime order q. The Pedersen commitment [84] of $x, y \in \mathbb{Z}_q$ is

$\mathsf{PedCom}(x;y) = h_1^x h_2^y$. Suppose the sequence of thresholds is $t_0 < t_1 < \ldots < t_n$, and let $t = t_n$. To share a secret $s \in \mathbb{Z}_q$, the dealer performs the following four steps:

1. The dealer chooses a random sequence of values $a_0, a_1, \ldots, a_{t-2}$ and sets $a_{t-1} = s$. The sequence of values defines the polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i$.
2. The dealer chooses a random sequence of values $b_0, b_1, \ldots, b_{t-1}$. The sequence of values defines the polynomial $g(x) = \sum_{i=0}^{t-1} b_i x^i$.
3. The dealer sends each user u at level $L = \mathcal{L}(u)$ his share $(f^{(t-t_L)}(u), g^{(t-t_L)}(u))$.
4. The dealer calculates $C_i = \mathsf{PedCom}(a_i; b_i)$ and publishes $C_0, \ldots, C_{t-1}$.

A user u at level $L = \mathcal{L}(u)$ can verify the validity of his share (x, y) by checking that:

$$\mathsf{PedCom}(x;y) = \prod_{i=t-t_L}^{t-1} C_i^{\frac{i!}{(i-t+t_L)!}u^{i-t+t_L}}$$

3.2.2 Efficiency

It takes the dealer $O(2t)$ field operations to compute the share of each user. On top of this, the dealer publishes $O(t)$ field elements as verification purposes. It takes a user $O(t)$ field operations to verify a share. A group of users can reconstruct the secret in $O(t^3)$ time by performing Gaussian elimination.

We note that Ballico et al. [7] propose a verifiable variant of Tassa's [95] conjunctive secret sharing scheme; their scheme requires the dealer to perform $O(t)$ field operations to compute a user's share but preserves privacy only under the discrete logarithm assumption. Applying their result to our disjunctive scheme is straightforward.

3.2.3 Security

Theorem 3.2.1. *The construction in Section 3.2.1 results in a private, correct, complete, and binding verifiable secret sharing scheme as long as the dealer cannot compute* $\log_{h_1} h_2$.

Proof. Correctness. This follows from Theorem 3.1.7 because each users's share contains the same $f^{(t-t_L)}(u)$ as in Construction 3.1.1.

Privacy. This is also straightforward. The only extra information received by a user u at level L are Pedersen [84] commitments to the coefficients of the polynomials f and g, as well as $g^{(t-t_L)}(u)$. Pedersen commitments do not provide any extra information, while the $g^{(t-t_L)}(u)$ does not provide any more information about g than $f^{(t-t_L)}(u)$ does about f. Thus, an unauthorized set of users gains no advantage when trying to learn the secret.

Completeness. This follows from the fact that:

$$\begin{aligned}\mathsf{PedCom}(x;y) &= \prod_{i=t-t_L}^{t-1} C_i^{\frac{i!}{(i-t+t_L)!}u^{i-t+t_L}} \\ &= \prod_{i=t-t_L}^{t-1} (h_1^{a_i}h_2^{b_i})^{\frac{i!}{(i-t+t_L)!}u^{i-t+t_L}} \\ &= h_1^{f^{(t-t_L)}(u)}h_2^{f^{(t-t_L)}(u)}\end{aligned}$$

Binding. We essentially follow Pedersen's proof [84]. If an authorized set of users accepts all of its shares, then, due to correctness, the users can reconstruct some pair of polynomials f and g that are consistent with their shares. This implies that $C_0 = \mathsf{PedCom}(f(0); g(0))$. Now suppose that there are two (possibly overlapping) sets of users A and A' that reconstruct different secrets from their shares. This means that there exist values s, s', t, t', where $s \neq s'$ and $t \neq t'$, such that $C_0 = \mathsf{PedCom}(s;t) = \mathsf{PedCom}(s';t')$. In this case, we can use the shares of A and A' to calculate $\log_{h_1} h_2$ using standard techniques [84]. All the dealer has to do is find two sets of users with inconsistent shares and use them to calculate the discrete logarithm.

To do this, the dealer starts with an arbitrary minterm A, such that $|A| = t$, the highest threshold. The dealer uses the shares assigned to those users to calculate f and g (a minterm of size t ensures the dealer can reconstruct f and g completely, rather than just their derivatives). Next, the dealer goes through every other user $u \in \mathcal{U}$ and checks if that user's share $(x, y) = (f^{(t-t_L)}(u), g^{(t-t_L)}(u))$, where $L = \mathcal{L}(u)$. If the user's share is inconsistent, the dealer constructs a new set of users $A' = A - V + \{u\}$, where V is the set of users that need to be removed to ensure A' is a minterm. Since u has a share that is not on f, g, the dealer can use the shares of A' to reconstruct a different pair of polynomials with a different secret than that of A. □

Chapter 4

Fair Exchange of E-Cash

In this chapter, we present new protocols for exchanging e-cash for digital goods and services. We use CHL compact e-cash as a starting point. We show how to modify the Spend protocol to let the user engage in a fair exchange with the merchant. In Section 4.1, we introduce endorsed e-cash, which reduces exchanging e-coins to exchanging lightweight endorsements. Then, in section 4.3, we show how to extend endorsed e-cash to allow paying multiple e-coins for a single good or service. While this multiple coin exchange protocol still takes linear time in the number of e-coins, the actual fair exchange (which is the bottleneck) needs to be performed only on one endorsement!.

The results in this chapter appear in "Endorsed E-Cash" [29].

4.1 Endorsed E-Cash

Endorsed e-cash solves the fair exchange problem for e-cash (see Section 2.4.3 for background). Endorsed e-cash is similar to e-cash. The only difference is that spending a coin is split into two stages. In the first stage, a user gives a merchant a blinded version of the coin, a.k.a. an *unendorsed coin*. An unendorsed coin is not a real coin and cannot be deposited with the bank. A user is allowed to issue unendorsed coins as often as he wants — it should be impossible to link two unendorsed versions of the same e-coin. (This is the chief difference between our solution and that of Jakobsson [61] and Asokan et al. [5]). A user can endorse a coin by giving a particular merchant the information he needs to transform the unendorsed coin into a real coin (i.e. an *endorsed coin*) that can be deposited with the bank. As long as a user endorses at most one version of the same wallet coin, he is not a double-spender and cannot be identified.

As a result, exchanging e-cash for electronic goods and services is reduced to exchanging lightweight endorsements for electronic goods and services. In our construction, the endorsement is simply the opening of a Pedersen commitment. This allows us to use Asokan, Shoup, and Waidener's [5] fair exchange algorithm (for exhcanging the pre-image of a homomorphic function for digital goods and services).

In this section, we formally define endorsed e-cash, give a construction, and prove its security. We also introduce threshold endorsed e-cash; now the merchant needs to get m out of n possible endorsements in order to recover the e-coin.

4.1.1 Definition

An endorsed e-cash system is almost identical to a regular e-cash system, except Spend is replaced by SplitCoin, ESpend, and Reconstruct. We define the three new protocols:

SplitCoin$(params, W_j, pk_{\mathcal{B}})$ A user $\mathcal{U}$ can take a coin from his wallet and generate $(\phi, x, y, coin')$. The value $coin'$ is a blinded version of the e-coin. The function ϕ is a one-way homomorphic function, such that $\phi(x) = y$. The tuple $(\phi, x, y, coin')$ should have enough information to reconstruct the e-coin.

$\mathsf{ESpend}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ This is the endorsed spend protocol. The user $\mathcal{U}$ privately runs SplitCoin to generate $(\phi, x, y, coin')$. The user gives the merchant $(\phi, y, coin')$, but keeps x for himself. The merchant uses $coin'$ to verify the validity of the unendorsed coin.

$\mathsf{Reconstruct}(\phi, x, y, coin')$ This function (typically used by a merchant) reconstructs a coin that can be deposited with the bank *if and only if* $\phi(x) = y$.

An endorsed e-cash scheme should have the same properties of correctness, anonymity, balance, culpability and exculpability as an e-cash scheme. However, the definitions must be slightly modified to fit the new set of protocols:

Correctness. If an honest user runs Withdraw with an honest bank, then neither will output an error message; if an honest user runs SplitCoin and gives the resulting $(\phi, y, coin')$ to an honest merchant via the ESpend protocol, the merchant will accept; if an honest merchant gets $(\phi, y, coin')$ from an honest user and learns the value $x = \phi^{-1}(y)$, then he'll be able to use Reconstruct to generate a valid coin that an honest bank will accept during the Deposit protocol.

Anonymity. Splitting a coin into two pieces: $(\phi, y, coin')$ and x should not increase the ability of a consortium of a malicious bank and merchants to link a coin to a user. Nor should an adversary be able to link two unendorsed versions of the same coin to each other. Once again, we create a simulator $\mathcal{S}$ and give it special powers. The simulator should be able to run the ESpend protocol without knowing any information about any user's wallet or public/secret-key pair.

Formally, we create an adversary $\mathcal{A}$ that plays the part of the bank and of all merchants. $\mathcal{A}$ creates the bank's public-key $pk_{\mathcal{B}}$. Then, $\mathcal{A}$ gets access to an interface Game that plays either a real game or an ideal game; $\mathcal{A}$ must determine which. $\mathcal{A}$ can make five types of queries to Game:

$\mathsf{GameSetup}(1^k)$ generates the public parameters *params* and private parameters *auxsim* for $\mathcal{S}$.

$\mathsf{GameGetPK}(i)$ returns the public-key of user $\mathcal{U}_i$, generated by $\mathsf{UKeygen}(1^k, params)$.

$\mathsf{GameWithdraw}(i)$ runs Withdraw with user $\mathcal{U}_i$: $\mathsf{Withdraw}(\mathcal{U}_i(pk_{\mathcal{B}}, sk_i, n), \mathcal{A}(state, n))$. We call W_j the wallet generated the jth time the protocol Withdraw is run.

$\mathsf{GameESpend}(j, J)$ gives the adversary an unendorsed coin number J from wallet W_j. In the real game, GameESpend runs the ESpend protocol with the user $\mathcal{U}$ that holds the wallet W_j: $\mathsf{ESpend}(\mathcal{U}(W_j, J, pk_{\mathcal{B}}), \mathcal{A}(state, n))$. In the ideal game, $\mathcal{S}$ plays the part of the user and runs the protocol: $\mathsf{ESpend}(\mathcal{S}(params, auxsim, pk_{\mathcal{B}}), \mathcal{A}(state, n))$. $\mathcal{S}$ knows nothing about the wallet W_j, the particular coin J requested, or the user who owns it. In the end, the adversary gets the unendorsed coin $(\phi, y, coin')$.

$\mathsf{GameEndorse}(\phi, y, coin')$ returns either the endorsement $x = \phi^{-1}(y)$ or an error message if the protocol GameESpend has not previously issued $(\phi, y, coin')$.

An adversary is called *legal* if it never asks a user to double-spend. Suppose two separate calls to $\mathsf{GameESpend}(j, J)$ result in the responses $(\phi, y_1, coin'_1)$ and $(\phi, y_2, coin'_2)$. A legal adversary never calls both $\mathsf{GameEndorse}(\phi, y_1, coin'_1)$ and $\mathsf{GameEndorse}(\phi, y_2, coin'_2)$. An endorsed e-cash scheme preserves anonymity if no computationally bounded *legal* adversary can distinguish between the real and ideal game with more than negligible probability.

Balance. The balance property is the same as in regular e-cash. See Section 2.4.1 for reference.

Culpability and Exculpability. We combine SplitCoin, ESpend, and Reconstruct to create a protocol SPEND that corresponds to the Spend protocol of a standard e-cash scheme. We need to show that the e-cash system $\mathcal{EC} = (\mathsf{BKeygen}, \mathsf{UKeygen}, \mathsf{Withdraw}, \mathsf{SPEND}, \mathsf{Deposit}, \mathsf{PublicSecurityProtocols})$ meets the culpability and exculpability guarantees of a standard e-cash system. We define $\mathsf{SPEND}(\mathcal{U}(W, pk_{\mathcal{M}}), \mathcal{M}(sk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ as follows: First, $\mathcal{U}$ calls the function $\mathsf{SplitCoin}(params, W_j, pk_{\mathcal{B}})$ to generate the tuple $(\phi, x, y, coin')$ and sends it to $\mathcal{M}$. When $\mathcal{M}$ receives $(\phi, x, y, coin')$, he verifies that $(\phi, y, coin')$ is valid (as in ESpend), and checks if $\phi(x) \neq y$ coin. If either test fails, $\mathcal{M}$ rejects. Otherwise, $\mathcal{M}$ creates the corresponding endorsed coin $coin = \mathsf{Reconstruct}(\phi, x, y, coin')$. $\mathcal{M}$ stores *coin* until he is ready to deposit it.

The culpability and exculpability properties provide protection if the user issues only one unendorsed coin per wallet coin – in this case, endorsed e-cash reduces to standard e-cash. So what prevents dishonest merchants from using an endorsement from one coin to generate endorsements for other coins? If a merchant successfully deposits a falsely endorsed coin with the bank, then he violates the balance property. If the merchant uses the fake endorsement to frame a user for double-spending, then he violates anonymity.

4.1.2 Construction

Our endorsed e-cash construction is based on CHL. The wallet coin (J, u, s, t, σ) is the same as before, but the unendorsed coin is a blinded version of the CHL e-coin. Instead of giving the merchant (S, T, Φ, R), the user chooses a random endorsement (x_1, x_2, x_3) and calculates (S', T', Φ', R, y), where $S' = Sg^{x_1}$, $T' = Tg^{x_2}$ and $y = \mathsf{PedCom}(x_1, x_2; x_3)$. The value Φ' is a zero-knowledge proof that the unendorsed coin is valid. Once the merchant learns the endorsement, he can easily reconstruct (S, T, Φ', R), which along with y and (x_1, x_2, x_3) constitutes an endorsed coin that can be deposited with the bank. The user can generate as many unendorsed versions of the same wallet coin as he wants by choosing different endorsements. However, if he endorses two versions of the same wallet coin, the bank will identify him using the same method as in CHL.

Global parameters: Same as in CHL. Additionally, let g, h_1, h_2 be elements in $\mathbb{G}$ whose discrete logarithms with respect to each other are unknown. We define the homomorphic one-way function $\phi : \mathbb{Z}_q^3 \to \mathbb{G}$, where $\phi(a, b, c) = h_1^a h_2^b g^c$. We split the public parameters $params = (params_{CHL}, params_{ZK})$, where $params_{ZK}$ is used for the zero-knowledge proof in the SplitCoin protocol and $params_{CHL}$ is used for everything else (and is, in fact, the same as in the CHL system).

SplitCoin, defined in Algorithm 4.1.1, creates an endorsable coin $(S', T', \Phi', R, (x_1, x_2, x_3), y)$, where (S', T', Φ', R, y) is the unendorsed coin and (x_1, x_2, x_3) is the endorsement (with $\phi(x_1, x_2, x_3) = y$). The values S' and T' are blinded versions of S and T and Φ' is the zero-knowledge proof that S' and T' are formed correctly. The merchant verifies Φ' during the ESpend protocol.

When the merchant receives the endorsement (x_1, x_2, x_3) for his unendorsed coin (S', T', Φ', R, y) he calls Reconstruct to create an endorsed coin $(S = S'/g^{x_1}, T = T'/g^{x_2}, \Phi', R, (x_1, x_2, x_3), y)$. The endorsed coin is almost identical to the original coin (S, T, Φ, R), except that Φ' is a zero-knowledge proof of slightly different information. Possession of that information is sufficient to create a valid CHL coin and the bank can safely accept it. The bank can also identify double-spenders because S, T, R are constructed the same way as in the CHL Spend protocol.

Algorithm 4.1.1: SplitCoin

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key, $contract \in \{0,1\}^*$
User Data: u private key, g^u public key, (s, t, σ, J) a wallet coin
$R \leftarrow H(pk_{\mathcal{M}} || contract)$;
$x_1, x_2, x_3 \leftarrow \mathbb{Z}_q$;
$y \leftarrow \phi(x_1, x_2, x_3)$;
$S' \leftarrow \mathsf{DY}_{g,s}(J) g^{x_1}$;
$T' \leftarrow g^u \mathsf{DY}_{g,t}(J)^R g^{x_2}$;
Calculate zero-knowledge proof of knowledge Φ' of $(J, u, s, t, \sigma, x_1, x_2, x_3)$ such that:
  $y = h_1^{x_1} h_2^{x_2} g^{x_3}$
  $0 \leq J < n$
  $S' = \mathsf{DY}_{g,s}(J) g^{x_1}$
  $T' = g^u \mathsf{DY}_{g,t}(J)^R g^{x_2}$
  $\text{VerifySig}(pk_{\mathcal{B}}, (u, s, t), \sigma) = true$
return $(S', T', \Phi', R, (x_1, x_2, x_3), y)$

4.1.3 Efficiency

SplitCoin is very similar to CalcCoin; it requires two more multi-base exponentiation from the user, one to compute y and one due to its inclusion in the proof, and one more multi-base exponentiation from the merchant and bank to verify the proof. (Note: we compute T' slightly different from CHL, but this has a negligible effect on the computation.) Thus, it takes the user a total of eighteen multi-base exponentiations to compute an e-coin, and it takes the merchant and bank a total of twelve multi-base exponentiations to verify the e-coins.

4.1.4 Security

Theorem 4.1.1. *The endorsed e-cash system described in Section 4.1.2 meets the definition of a secure endorsed e-cash system.*

Proof. Correctness. It is easy to see the system is correct because the key values S, T, R are identical to the CHL e-cash system.

Anonymity. We construct an algorithm $\mathcal{S}$ that impersonates all honest users of the endorsed e-cash system without access to their data during the ESpend protocol. (Recall, in our definition, the adversary accesses an interfaces Game, which either invokes real users or $\mathcal{S}$). $\mathcal{S}$ will use $\mathcal{S}_{CHL}$, the simulator for the CHL Spend protocol, and $\mathcal{S}_{ZK}$, the simulator for the zero-knowledge system, as building blocks. We will show that any adversary $\mathcal{A}$ that can distinguish when the interface Game plays the real game with real users or the ideal game using $\mathcal{S}$ can either (1) break the anonymity of CHL or (2) violate the zero-knowledge property of the zero-knowledge proof system.

$\mathcal{S}$ gets as input $(params, auxsim, pk_{\mathcal{B}})$. The endorsed e-cash system generated $(params, auxsim)$ during GameSetup; some of those parameters are intended for $\mathcal{S}_{CHL}$ and $\mathcal{S}_{ZK}$: $(params_{CHL}, auxsim_{CHL})$ is intended for $\mathcal{S}_{CHL}$ and $(params_{ZK}, auxsim_{ZK})$ is for $\mathcal{S}_{ZK}$.

$\mathcal{S}$ has to simulate ESpend. It gets $(contract, pk_{\mathcal{M}})$ from the adversary. $\mathcal{S}$ executes $\mathsf{Spend}(\mathcal{S}_{CHL}(params_{CHL}, auxsim_{CHL}), \mathcal{S}(contract, pk_{\mathcal{M}}, pk_{\mathcal{B}}, n))$ (n is the size of the wallets), pretending to be a merchant. $\mathcal{S}$ does not need the merchant's secret-key for the Spend protocol. $\mathcal{S}_{CHL}$ gives $\mathcal{S}$ some coin (S, T, Φ, R). $\mathcal{S}$ pretends to run SplitCoin. First it randomly generate (x_1, x_2, x_3). Then it uses the the "endorsement" to calculate: $y = \phi(x_1, x_2, x_3)$, $S' = Sg^{x_1}$, and $T' = Tg^{x_2}$. Then it calls $\mathcal{S}_{ZK}(params_{ZK}, auxsim_{ZK})$ to generate a fake proof Φ'. $\mathcal{S}$ sets $coin' = (S', T', \Phi', R, y)$. It stores $(\phi, (x_1, x_2, x_3), y, coin')$ in a database for later use and returns $(\phi, y, coin')$ to the adversary.

We prove $\mathcal{S}$ is indistinguishable from real users via a hybrid argument. Consider an algorithm $\mathcal{S}_1$ that acts just like a real user, but after constructing a legitimate unendorsed coin, invokes $\mathcal{S}_{ZK}$ to create a fake proof Φ'. If $\mathcal{A}$ can distinguish $\mathcal{S}_1$ from a real user, $\mathcal{A}$ violates the zero-knowledge property of the zero-knowledge proof system. Now consider algorithm $\mathcal{S}_2$ that generates unendorsed coins using $\mathcal{S}_{CHL}$ and $\mathcal{S}_{ZK}$, but makes sure that all unendorsed versions of the same coin have the same serial number. In this case, if $\mathcal{A}$ can distinguish $\mathcal{S}_1$ from $\mathcal{S}_2$, $\mathcal{A}$ violates the anonymity of CHL. Finally, by the definition of SplitCoin, the S' and T' are information theoretically independent of the real serial number. Therefore, $\mathcal{S}_2$ is indistinguishable from $\mathcal{S}$. By the hybrid argument, no adversary can tell when Game is playing the ideal game or the real game.

Balance. We need to show that no consortium of users and merchants can cheat an honest bank. Suppose we have an adversary $\mathcal{A}$ that can break the balance property of our endorsed e-cash system. $\mathcal{A}$ executes the Withdraw protocol u times to withdraw un coins (assuming n coins per wallet). We take the knowledge extractor $\mathcal{E}$ from the CHL system and use it to generate serial numbers $S_1, \ldots, S_{un}$ from all the invocations of Withdraw (recall that our endorsed e-cash uses the same Withdraw protocol as CHL). Eventually, $\mathcal{A}$ produces an endorsed coin $(S, T, \Phi', R, (x_1, x_2, x_3), y)$ that the bank accepts, but $S \notin S_1, \ldots, S_{un}$. Since the bank accepted the endorsed coin, this implies that $\phi(x_1, x_2, x_3) = y$ and Φ' is valid. Since Φ' is formed by a sound zero-knowledge proof system, $\mathcal{A}$ knows values J, u, s, t, σ such that: (1) $S' = Sg^{x_1} = g^{/(s+J)}g^{x_1}$, (2) $T' = Tg^{x_2} = g^{R/(t+J)}g^{x_2}$, and (3) $\text{VerifySig}(pk_{\mathcal{B}}, (u, s, t), \sigma) = true$.

Therefore, we can use $\mathcal{A}$ to create a proof Φ such that the CHL bank accepts the coin (S, T, Φ, R). We construct a reduction that breaks the security of the CHL scheme by playing middleman in the Withdraw and Deposit invocations that $\mathcal{A}$ makes. The reduction can set up the public parameters

for the endorsed e-cash zero-knowledge proof system, and exploit them to extract the values u, s, t, σ from $\mathcal{A}$. As a result, it can construct a valid CHL zero-knowledge proof for coins that $\mathcal{A}$ tries to deposit.

Culpability and Exculpability. Since Reconstruct creates a coin $(S, T, \Phi', R, (x_1, x_2, x_3), y)$ where (S, T, R) are the same as in the CHL system, the CHL PublicSecurityProtocols can remain unchanged. Therefore, culpability and exculpability are preserved. □

4.2 Threshold Endorsed E-cash

Sometimes, we want to require the merchant to acquire several endorsements before reconstructing an e-coin. In this section, we construct a threshold endorsed e-cash system where the merchant needs to get m out of n possible endorsements.

Threshold endorsed e-cash uses ideas from secret-sharing. When exchanging a regular endorsed e-coin, the user gives the merchant a Pedersen commitment to the endorsement, then performs a fair exchange for the endorsement. In threshold endorsed e-cash, the user generates a random polynomial f such that $f(0)$ is equal to the endorsement. The user gives the merchant a Pedersen commitment to a point on the polynomial, then performs a fair exchange for the point. Once the merchant collects enough points, he is able to interpolate f and learn the endorsement.

The merchant wants to make sure that he always gets a point on the same polynomial f. Therefore, the user will give the merchant a Pedersen commitment to the coefficients of f in advance. The merchant will take advantage of the algebraic properties of Pedersen commitments to verify that the Pedersen commitment to the point lies on the polynomial described by the committed coefficients.

Another minor technical detail is that, in our our construction, an endorsement consists of three points (x_1, x_2, x_3). As a result, the user will actually generate three random polynomials (f_1, f_2, f_3). The merchant will verify the values in one efficient step.

We now describe our scheme in detail.

4.2.1 Construction

An unendorsed coin consists of (S', T', Φ', R, y), where $y = \mathsf{PedCom}(x_1, x_2; x_3)$. We can use Pedersen Verifiable Secret Sharing [84] to create shares of the endorsement. For notational convenience, we use (g_1, g_2, g_3) instead of original parameters (h_1, h_2, g) in Section 4.1.2.

To share (x_1, x_2, x_3), the user generates three random polynomials f_1, f_2, f_3 of degree $m-1$ such that $f_j(0) = x_j$. The user stores a secret vector of n points on the polynomial; these points are the endorsements. The user gives the merchant commitments to the coefficients that define the polynomials. Once the merchant learns m points on the polynomials, he can recover (x_1, x_2, x_3) and endorse the coin. Algorithm 4.2.3 describes how the user creates a threshold endorsable coin.

Algorithm 4.2.1: SplitCoinMN

Input: $pk_{\mathcal{M}} \in \{0,1\}^*$ merchant's public key, $contract \in \{0,1\}^*$
User Data: u private key, g^u public key, (s, t, σ, J) a wallet coin
$(S', T', \Phi', R, (x_1, x_2, x_3), y) \leftarrow \mathsf{SplitCoin}(pk_{\mathcal{M}}, contract)$;
$a_{j,0} \leftarrow x_j, \forall j \in \{1,2,3\}$;
$a_{j,k} \leftarrow \mathbb{Z}_q, \forall j \in \{1,2,3\}, \forall k \in [1, m-1]$;
$Z \leftarrow \{Z_k = \prod_{j=1}^{3} g_j^{a_{j,k}} : k \in [0, m-1]\}$;
$X \leftarrow \{X_j^{(i)} = \sum_{k=0}^{m-1} a_{j,k} i^k : j \in \{1,2,3\}, i \in [0, n]\}$;
return (S', T', Φ', R, X, Z)

In $\mathsf{MNSpend}$, the user gives the merchant the threshold unendorsed coin (S', T', Φ', R, Z) and

stores the endorsement X. The merchant needs to verify the unendorsed coin: he uses uses Z, a commitment to the polynomials' coefficients, to calculate Y, a commitment to points on the polynomials: $Y = \{Y^{(i)} = \prod_{k=0}^{m-1} (Z_k)^{i^k} : i \in [0, n]\}$. The merchant sets $y = Y^{(0)}$ and verifies Φ' in the usual way.

Now the merchant needs to get m endorsements. The user has n endorsements: $\{(X_1^{(i)}, X_2^{(i)}, X_3^{(i)}) : i \in [1, n]\}$. They can use the homomorphic one-way function $\phi(a, b, c) = g_1^a g_2^b g_3^c$ to do an optimistic fair exchange because $\phi(X_1^{(i)}, X_2^{(i)}, X_3^{(i)}) = Y^{(i)}$ (remember, Φ' proves the $Y^{(i)}$ are correct). In MNReconstruct, the merchant uses the m points to interpolate the polynomials and learn (x_1, x_2, x_3).

4.2.2 Efficiency

It takes the user m extra multi-base exponentiations to commit to the coefficients of the polynomials f_1, f_2, f_3. It takes the user $3m$ single-base exponentiations to calculate a single endorsement. The zero-knowledge proof remains unchanged. Thus, the user makes a total of $18 + m + 3mn$ multi-base exponentiations to construct a threshold e-coin with n different endorsements. To verify an unendorsed threshold e-coin, the merchant needs to calculate $y = Y^{(0)}$; this takes m single-base exponentiations. Then the merchant verifies the zero-knowledge proof in the usual way. Thus, the merchant performs a total of $12 + m$ multi-base exponentiations to verify an unendorsed threshold e-coin. To check the validity of a single endorsement, the merchant performs $m + 3$ multi-base exponentiations. Thus, the final work done by the merchant is $12 + m + m(m + 3)$. The bank also performs $12 + m + m(m + 3)$ multi-base exponentiations to verify a threshold e-coin for deposit.

4.2.3 Security

This is a straightforward application of Pedersen VSS. The user creates n verifiable shares of the secret (x_1, x_2, x_3) and gives the merchant the standard verification vector. Each endorsement is a share of the secret.

4.3 Paying Multiple E-Coins

Suppose a merchant is selling a car for 19,995 e-coins (an e-coin can be worth a dollar, or some other amount if the system supports different denominations). If a user wants to do a fair exchange, she must verifiably encrypt 19,995 endorsements. Creating and verifying the ciphertexts is computationally expensive. Worse, if the trusted third party becomes involved, it must store *all* of the verifiable encryptions and their tags.

We can significantly reduce the cost of the fair exchange. Examine the unendorsed coin (S', T', Φ', R, y) from Section 3.2. The value $y = \phi(x_1, x_2, x_3)$, where (x_1, x_2, x_3) is the endorsement. A fair exchange of one coin for the car trades the opening of y for the opening of some value K. A fair exchange of n unendorsed coins trades the opening of $(y^{(0)}, \ldots, y^{(n-1)})$ for the opening of K. Because ϕ is really a Pedersen commitment, we can use a Pedersen VSS [84] style algorithm to reduce opening all the $y^{(i)}$ to just opening $y^{(0)}$.

4.3.1 Definition

Definition 4.3.1. *(Secure multiple e-coin fair exchange) A multiple e-coin fair exchange scheme is secure if a merchant who, after enaging in an arbitrary number of fair exchange protocols with a set of users, and receiving N endorsed e-coins, cannot deposit more than N e-coins with the bank.*

We assume that the bank will accept all withdraw requests from the users. Additionally, if the merchant withdraws M e-coins from the bank, then the merchant should not be able to deposit more than $M + N$ e-coins.

4.3.2 Construction

Setup: We will use the same public parameters as the endorsed e-cash system in Section 4.1.2. For notational convenience, we will use (g_1, g_2, g_3) instead of (h_1, h_2, g) (recall that these are three generators of $\mathbb{G}$ whose discrete logarithm representation relative to each other is unknown; we assume the discrete logarithm problem is hard in $\mathbb{G}$). We set $\phi(a, b, c) = g_1^a g_2^b g_3^c$.

User Promise: The user makes n new endorsable coins $(S'^{(i)}, T'^{(i)}, \Phi'^{(i)}, R^{(i)}, (x_1^{(i)}, x_2^{(i)}, x_3^{(i)}), y^{(i)})$, for $i \in [0, n-1]$. The user defines three polynomials f_1, f_2, f_3 of degree $n-1$, such that $\forall i \in [0, n-1], \forall j \in \{1, 2, 3\} : f_j(i) = x_j^{(i)}$ (the user doesn't have to do anything for this). Let set $A = \{0, \ldots, n-1\}$. The user calculates $n-1$ new points $p_j^{(i)}$ on f_1, f_2 and f_3, as follows:

$$\forall i \in [n, 2n-2], \forall j \in \{1, 2, 3\} : p_j^{(i)} = f_j(i) = \sum_{a \in A} f_j(a) \prod_{\substack{b \in A \\ b \neq a}} \frac{i-a}{b-a}$$

The user gives the merchant the n unendorsed coins and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1, 2, 3\}\}$.

Merchant Verifies: The merchant gets n unendorsed coins $(S'^{(i)}, T'^{(i)}, \Phi'^{(i)}, R^{(i)}, y^{(i)})$, for $i \in [0, n-1]$, and uses the $\Phi'^{(i)}$ to verify their validity. Then the merchant checks that the openings of

the $y^{(i)}$ are on the same polynomials as the $p_j^{(i)}$. He does not need to know the openings for this! Let set $B = \{n, \ldots, 2n-2\}$. The merchant accepts only if:

$$\forall i \in [1, n-1] : y^{(0)} = (y^{(i)})^{\prod_{b \in B} \frac{i}{i-b}} \prod_{j=1}^{3} (g_j)^{\sum_{a \in B} \frac{a}{a-i} p_j^a \prod_{\substack{b \in B \\ b \neq a}} \frac{a}{a-b}} .$$

Fair Exchange: The merchant and the user conduct an optimistic fair exchange of the opening of $y^{(0)}$ for the opening of K. The merchant learns $\phi^{-1}(y^{(0)}) = (x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$. If the exchange fails, the user must throw out the unendorsed coins.

Reconstruct: The merchant uses $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$ and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$ to learn the openings of $y^{(1)}, \ldots, y^{(n-1)}$. He sets $C = \{0, n, \ldots, 2n-2\}$, $p_j^{(0)} = x_j^{(0)}$, and calculates:

$$\forall i \in [1, n-1], \forall j \in \{1,2,3\} : x_j^{(i)} = f_j(i) = \sum_{a \in C} f_j(a) \prod_{\substack{b \in C \\ b \neq a}} \frac{x-a}{b-a}$$

4.3.3 Efficiency

The multiple e-coin fair exchange protocol runs in linear time in the number of e-coins. It takes the user eighteen multi-base exponentiations to compute each e-coin and $O(n^2 \log^2 n)$ multi-base exponentiations to compute all the points $p_j^{(i)}$. The merchant performs twelve multi-base exponentiations to verify the unendorsed e-coins and

4.3.4 Security

We want to show that the multi-coin fair exchange presented in Section 4.3.2 is secure. Specifically, we want to show that no malicious merchant that asks users to participate in the multiple-coin fair exchange can falsely endorse coins (so if a user endorses N coins then the merchant can deposit at most N coins). First we prove in Lemma 4.3.2 that if a merchant endorses a coin during a single run of a *failed* multi-coin exchange, then he can calculate discrete logarithms. Then we use the Lemma to show that if the merchant manages to deposit more coins than the users intended to give him, the merchant violates either the security of the endorsed e-cash scheme or the discerete logarithm assumption.

Lemma 4.3.2. *Let $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}), \ldots, (x_1^{(n-1)}, x_2^{(n-1)}, x_3^{(n-1)})$ be numbers in $\mathbb{Z}_q$ selected at random, and let g_1, g_2, g_3 be generators of a group $\mathbb{G}$. We define the function $\phi(a,b,c) = g_1^a g_2^b g_3^c$ and calculate $y^{(0)}, \ldots, y^{(n-1)}$, such that $y_i = \phi(x_1^{(i)}, x_2^{(i)}, x_3^{(i)})$. In addition, the $x_j^{(i)}$ define three polynomials f_1, f_2, f_3 such that $f_j(i) = x_j^{(i)}$ for $0 \leq i \leq n-1$. We calculate $n-1$ points on each of these three polynomials: $\{p_j^{(i)} = f_j(i) : i \in [n, 2n-2], j \in \{1,2,3\}\}$. Suppose there exists an adversary that on input $\mathbb{G}, g_1, g_2, g_3$, $(y^{(0)}, \ldots, y^{(n-1)})$, and $\{p_j^{(i)} = f_j(i) : i \in [n, 2n-2], j \in \{1,2,3\}\}$ outputs (a,b,c) such that $y^{(i)} = \phi(a,b,c)$, for some $i \in [0, n-1]$. Then we can use this adversary to calculate discrete logarithms in $\mathbb{G}$.*

Proof. We construct a reduction that uses the adversary from Lemma 4.3.2 as a black-box to calculate discrete logarithms. The reduction gets y as input. Suppose (x_1, x_2, x_3) is the opening of y; the

reduction does not know these values, but it constructs three polynomials f_1, f_2, f_3 so that $f_j(0) = x_j$. First the reduction randomly chooses $3(n-1)$ numbers in $\mathbb{Z}_q$: $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$; these will be random points that, along with the (unknown) opening of y, define the polynomials f_1, f_2, f_3. Then the reduction calculates $y^{(1)}, \ldots, y^{(n-1)}$. Let $S = [n, 2n-1]$, then:

$$\forall i \in [1, n-1] : y^{(i)} = y^{\prod_{b \in S} \frac{i}{b}} \prod_{j=1}^{3} (g_j)^{\sum_{a \in S} p_j^a \prod_{\substack{b \in S \cup \{0\} \\ b \neq a}} \frac{i-a}{b-a}}$$

The reduction passes $(y, y^{(1)}, \ldots, y^{(n-1)})$ and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$ to the black-box. The black-box responds with an opening to one of the $y^{(i)}$. From this the reduction can interpolate the polynomials and open y. □

We now show that no merchant can take advantage of the multi-coin fair exchange protocol to deposit more coins than the honest users intended to give him. Suppose a dishonest merchant, after running a number of multi-coin fair exchanges in which only N coins should be endorsed, manages to deposit more than N coins. Then we can construct a reduction that uses the merchant as a black-box to either break the balance of anonymity of the endorsed e-cash scheme or to calculate discrete logarithms.

The reduction gets y as input and needs to output (x_1, x_2, x_3) such that $y = h_1^{x_1} h_2^{x_2} g^{x_3}$. The reduction sets up an endorsed e-cash system, using (h_1, h_2, g) as the public parameters. It also uses $\mathcal{S}_{ZK}$, the simulator for the zero-knowledge system Φ' to create $(params_{ZK}, auxsim_{ZK})$ and $\mathcal{S}_{CHL}$, the simulator for the CHL e-cash system to create $(params_{CHL}, auxsim_{CHL})$.

The reduction runs multi-coin fair exchanges with the merchant. In one of those exchanges (the reduction chooses which one at random), the reduction inserts y into an unendorsed coin. Suppose the merchant wants n coins. Then the reduction prepares the input to the merchant as follows: It asks $\mathcal{S}_{CHL}$ to create n e-coins $(S^{(0)}, T^{(0)}, \Phi^{(0)}, R^{(0)}), \ldots, (S^{(n-1)}, T^{(n-1)}, \Phi^{(n-1)}, R^{(n-1)})$ (the reduction runs Withdraw and Spend the appropriate amount of times). Then the reduction uses y to create an unendorsed coin. It randomly chooses r_1 and r_2 and calculates $S' = Sg^{r_1}$ and $T' = Tg^{r_2}$ (we need to blind S and T; we don't know any valid openings of y, but for any r_1 and r_2 we choose, there exists *some* r_3 such that $\phi(r_1, r_2, r_3) = y$). Then it uses $\mathcal{S}_{ZK}$ to generate a fake proof Φ' such that an honest merchant would accept the unendorsed coin $coin'^{(0)} = (S'^{(0)}, T'^{(0)}, \Phi'^{(0)}, R'^{(0)}, y)$. Next the reduction chooses the random points on three polynomials: $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$. Finally, the reduction chooses the appropriate $y^{(1)}, \ldots, y^{(n-1)}$ (using the same method as in the proof of Lemma 4.3.2) and uses $\mathcal{S}_{CHL}$ and $\mathcal{S}_{ZK}$ to create the unendorsed coins $coin'^{(1)}, \ldots, coin'^{(n-1)}$. The reduction gives $coin'^{(0)}, \ldots, coin'^{(n-1)}$ and $\{p_j^{(i)} : i \in [n, 2n-2], j \in \{1,2,3\}\}$ to the merchant.

Eventually, the merchant outputs a list of more than N coins for deposit. At least one of these coins must be fake. If it is an entirely new coin then the merchant violated the balance property of the endorsed e-cash scheme. The only other possibility is that the coin was from a terminated multi-coin fair exchange. With non-negligible probability, the reduction would have inserted y into that fair exchange. In this case, by Lemma 4.3.2, the merchant violated the discrete logarithm assumption.

If the merchant *fails* to output more than N coins, then the merchant violated anonymity because it distinguished the simulator from real users (this can be shown with a straightforward reduction).

Chapter 5

Endorsed E-Cash Applications

In this chapter we show how endorsed e-cash can be used to add incentives to peer-to-peer systems without sacrificing user privacy.

Peer-to-peer systems leverage the cooperative behavior of a large group of relative resource poor nodes to provide each user with useful services. The decentralized nature of peer-to-peer networks provides many advantages, including fault-tolerance and scalability, yet seems to preclude simple enforcement mechanisms for ensuring that each peer does its part. For example, quotas and strong reputations are difficult to maintain in a peer-to-peer network, especially when user privacy is a concern.

We would like to design systems that provide fungible benefits to cooperative nodes, rewarding those that continue to provide service with credit towards future service. The goal is to make each user to contribute as much as he receives from the system. Equitable exchange would make the system significantly more robust, as it would no longer be dependent on the contributions of altruistic users. These benefits should be secure, anonymous, and fungible. Further, the complete system should continue to self-scale.

In Section 5.1 we show how to use threshold endorsed e-cash to create incentives for onion remailers. Then, in Section 5.2 we examine the problem of on-line file sharing. Finally, in Section 5.3, we end by showing how endorsed e-cash can be used to augment other types of peer-to-peer systems. The work in this chapter originally appeared in "Endorsed E-Cash," IEEE Security and Privacy 2007 [29] and "Making P2P Accountable without Losing Privacy," Workshop on Privacy in the Electronic Society 2007 [10].

5.1 Onion Remailing

Onion routing, invented by David Chaum [35], allows a user to send a message without revealing to outside observers that he is communicating with the receiver. The sender chooses a sequence of routers to pass the message. The sender first encrypts the message under the recipient's public-key, then under the public-key of each router in the chain. Onion routing derives its name from this layered encryption.

The sender then gives the ciphertext to the first router. The router decrypts the ciphertext to learn the name of the next router in the chain and a new ciphertext. The router passes the new ciphertext to the next router. The process repeats until the message reaches the intended the recipient. Under certain forms of encryption, the only information that each router learns is the location of the next router in the chain; a router does not even learn its distance to the sender or recipient [28]. As a result, an adversary who corrupts some of the routers in an onion remailing network cannot determine which sender is communicating with which recipient. In a sufficiently busy network, an adversary may not even be able to determine which nodes are originating or receiving messages, rather than merely passing them.

However, in an onion remailing network, routers have no incentive to consume bandwidth by passing messages. It is desirable to create incentives without jeopardizing the anonymous nature of

onion remailing. The naive solution would be to include an e-coin in each encryption layer of the message. This will not work because a router can deposit the e-coin without forwarding the message.

Suppose we include an unendorsed e-coin in each layer and make the router contact the next router in the chain in order to get the endorsement. This also will not work because the next router in the chain has no incentive to talk to the previous router. Reiter, Wang and Wright [86] tried this approach (using Jakobsson's [61] ripped e-cash rather than endorsed e-cash).

We propose using threshold endorsed e-cash. The sender would place a threshold unendorsed e-coin in each encryption layer. The e-coin would require two endorsements in order to be deposited. When a router receives a message, it will decrypt it to learn:

1. The name of the next router in the chain
2. The name of the previous router in the chain
3. The ciphertext to pass to the next router in the chain
4. A threshold unendorsed e-coin that requires two endorsements
5. The endorsement needed by the next router in the chain
6. The endorsement needed by the previous router in the chain

The router would be able to get one of the endorsements it needs from the previous router in the chain. The two routers would perform a fair exchange because each one has an e-coin endorsement that the other one needs. If the previous router in the chain is the original sender, it would still go through the fair exchange protocol to hide its identity as the originator of the message.

Next the router would pass the new ciphertext to the next router in the chain. It has an incentive to do this because the second endorsement it needs can only be decrypted by the next onion router. The next onion router would decrypt the ciphertext to get its own threshold unendorsed e-coin and two endorsements. The current and next router would perform a fair exchange of endorsements. Note that if the next router in the chain is the recipient, it must get an unendorsed e-coin that requires only one endorsement. If the recipient has no e-coin that needs endorsing, then it has no incentive to give the endorsement to the last router in the chain. As a result, each router has less incentive to pass the message because it knows that there is a good chance that it might be the last router and therefore it will not get paid.

5.2 File Sharing

File sharing is one the most popular applications of peer-to-peer networks. It is a fast way to obtain music, movie, and other files because there is no central distribution point that can form a bottleneck. Measurement studies and theoretical analyses have shown selfish behavior common among users of peer-to-peer file-sharing systems, and effective for obtaining an unfair share of resources [1, 64, 69, 68]. The most popular file-sharing system, BitTorrent [41], employs a "tit-for-tat" mechanism aimed at encouraging fairness, but a recent study has suggested that BitTorrent's effectiveness is due largely to the altruistic behavior of a small number of high-bandwidth nodes [85] rather than fair contributions from all participants. Most nodes leave the system as soon as they finish downloading; only the most altruistic continue to donate upload capacity.

There have been many prior attempts to solve the free rider problem in file sharing systems. They typically fall into three classes.

Barter Mechanisms. In a bartering system, users trade files (or blocks of files). The best example of this is the popular BitTorrent system. Users query randomly chosen peers to try set up a file exchange. If each peer has a piece of a file that the other one wants, then the two peers begin to transmit data to each other. Since each user has a limited upload bandwidth, the goal is to find a set of trading partners that would give back as much useful data as possible. As a result, peers constantly test new trading partners, keeping the best ones. Users have an incentive to share their data because otherwise their partners would move on to find better peers. However, as previously mentioned, BitTorrent is easily exploited [85].

An inherent problem with any barter mechanism is that incentives fail when no dual exchange is possible. There have been attempts to extend the barter economy by identifying chains of indebted nodes that can be used for transitive exchanges [78, 2]. However, this leads to unnecessary complexity in the system, creating an expensive trade-off between efficiency and fairness.

Reputation Systems. Local and centralized reputation systems have been used to provide incentives in peer-to-peer systems [101, 72, 78, 2]. Reputation schemes are ineffective when users can easily shed a discredited pseudonym. This limits their applicability in privacy-preserving peer-to-peer systems [46, 71]. Friedman and Resnick [52] show that in this environment, participants must distrust new users until they have earned enough reputation, leading to inefficiency.

Currency Based. Fungibility is the biggest benefit of a currency based system. There have been several prior attempts use currency to incentivize file-sharing. KARMA [99] and Scrivener [78] aim to provide incentives for resource sharing, but are built atop unincentivized DHT systems. BAR Gossip [67] also incentivizes file-sharing, but strictly controls which peers may exchange data, potentially underutilizing bandwidth. Mojo Nation [100] used a currency for incentives that was not provably secure. None of these systems ensured a fair-exchange of money for data.

In Section 5.2.1, we create a new optimistic fair exchange protocol that lets Alice buy a block of a file from Bob. In Section 5.2.2 we describe an improved dispute resolution protocol that lowers the the load on the TTP to be logarithmic in the size of the exchanged file, rather than linear as in Section 4.1 and Asokan et al. [5].

In Section 5.2.3, we create a new fair exchange protocol that lets Alice and Bob trade two blocks; we call this process bartering. Our barter protocol let Alice and Bob engage in many efficient barter transactions after they each place a single e-coin into escrow. This offers two advantages over prior work. The Asokan *et al.* [5] digital content exchange protocol requires a TTP to verify that each block of the exchanged files is correct and to sign the encryption of each block together with the commitment to the encryption key. Our protocol only assumes that both parties know the desired hash of each block; thus no trusted signer is required. Also, since we let Alice and Bob maintain a continuous relationship; they only have to create one e-coin each to initiate a relationship, and can reuse the e-coins indefinitely.

Our protocols require a symmetric block cipher (see Section 2.2.9): we write $\mathsf{Encrypt}(K, block)$ to denote encrypting *block* with key K; $\mathsf{Decrypt}(K, ctext)$ means decrypting *ctext* using key K. We assume that a block is large enough to be divided into chunks and $\mathsf{Encrypt}(K, block)$ encrypts each chunk separately. We also need to perform escrow [31] (also known as verifiable encryption, see Section 2.2.9): $\mathsf{Escrow}_{TTP}(data, contract)$ encrypts *data* under the public-key of the TTP. The decryption key to the escrow is a combination of the TTP's secret-key and the *contract*; this lets the TTP ensure he decrypts the escrow only when the terms of the *contract* have been fulfilled. Anybody who knows the TTP's public-key and the *contract* can verify that the escrow is valid. Finally, write $\mathsf{Com}(data)$ to denote a commitment that can only be opened to *data* (this should be a Pedersen commitment for efficiency [84], see Section 2.2.6).

We use Merkle hash trees [74] to create short descriptions of a block (or ciphertext). We write $\text{MHash}(block)$ to denote a Merkle hash of *block*. A person who knows the entire file can publish $(chunk, proof, \text{MHash}(block))$ to prove that *chunk* is in (or is not in) *block*; *proof* is short, efficiently calculated, and includes the position of the chunk in the block. See Section 2.2.4 for details on how Merkle trees work. Finally, we require a collision-resistant hash function h (see Section 2.2.3).

5.2.1 Buying Files

We present our protocol that lets Alice buy a file block from Bob. Before the start of the protocol, Alice acquires $bhash = \text{MHash}(block)$ from a trusted authority. (It is impossible to do a secure exchange of digital content without some authority certifying that the content Alice gets is the content Alice wants). Alice and Bob will agree on a *timeout* by when Bob must provide Alice with the block. The protocol works as follows:

1. Bob chooses a random key K and sends $ctext = \mathsf{Encrypt}(K, block)$ to Alice.

2. Alice constructs an endorsed e-coin $(coin', endorsement)$. Alice calculates $chash = \text{MHash}(ctext)$, chooses a random value r and calculates the exchange ID $v = h(r)$. Alice sets *contract* =

Algorithm 5.2.1: AliceResolve, run by the TTP

Input: Exchange ID r (ensures only Alice can resolve)
$v \leftarrow h(r)$;
if $< v, K > \in DB$ **then**
  send K to Alice.
end

Algorithm 5.2.2: BobResolve, run by the TTP

Input: key K, *escrow*, and $contract = \{bhash, chash, timeout, coin', v\}$
if $currentTime < timeout$ **then**
  $endorsement \leftarrow$ Decrypt($escrow$);
  if *endorsement not valid for coin'* **then**
    return error.
  end
  Run VerifyKey with Bob for K, using $(bhash, chash)$ from the contract.
  if K *verifies* **then**
    add $< v, K >$ to DB.
    send *endorsement* to Bob.
  else
    return error.
  end
end

$(bhash, chash, timeout, coin', v)$. She escrows the *endorsement* under the TTP's public-key: $escrow = \mathsf{Escrow}_{TTP}(endorsement, contract)$. Alice sends Bob $(coin', contract, escrow)$.

3. Bob verifies that $(coin', contract, escrow)$ is formed correctly. If he is satisfied, he establishes a secure connection to Alice using standard techniques and sends the key K. Otherwise, Bob terminates.

4. If Alice receives a K that lets her decrypt *ctext* correctly before *timeout*, she responds with *endorsement*. Otherwise, Alice waits until *timeout* and then calls AliceResolve(r) on the TTP, as in Algorithm 5.2.1.

5. If Bob does not receive a correct *endorsement* before *timeout*, he calls BobResolve(K, *escrow*, *contract*) on the TTP, as in Algorithm 5.2.2. During BobResolve, Bob will run the interactive VerifyKey algorithm, to prove to the TTP that K is the correct decryption key. We describe this process in Section 5.2.2.

Security. Suppose Alice wants to avoid paying. If an Bob calls BobResolve before *timeout* and provides the TTP with a valid decryption key (see Section 5.2.2), he is guaranteed to be paid, as long as the unendorsed e-coin is valid. If the unendorsed e-coin is invalid (either badly formed $coin'$ or incorrect *contract*), Bob would not accept it and terminate in step 4 without giving Alice the key K.

Suppose Bob wants to avoid giving Alice a correct key. If he calls BobResolve after *timeout* he will not get paid. If he calls BobResolve before *timeout*, due to the *contract* associated with the

Algorithm 5.2.3: VerifyKey

TTP's Input: Two Merkle hashes *bhash* and *chash*, key K
Bob's Input: Ciphertext $c_0||\ldots||c_n$, key K
`Step 1: TTP's challenge`
The TTP sends Bob a set of indices I.
`Step 2: Bob's response`
Bob replies with $(c_i, cproof_i, bproof_i)$ for every $i \in I$, where $cproof_i$ proves that c_i is the Merkle tree corresponding to *chash* and $bproof_i$ proves that $b_i = \mathsf{Decrypt}(K, c_i)$ is in the Merkle tree corresponding to *bhash*.
`Step 3: Verification`
The TTP *accepts* the key if Bob responds with valid $(c_i, cproof_i, bproof_i)$ for every $i \in I$, and *rejects* otherwise.

escrow, he can only get paid if he deposits the correct key K. Alice can then retrieve K at her convenience.

5.2.2 Efficient Dispute Resolution

We give a protocol that lets a seller prove to the TTP that he provided the buyer with the correct ciphertext and decryption key.

Recall that when Bob calls BobResolve in Algorithm 5.2.2, he has to prove to the TTP that he has provided it with the correct key. Specifically, he has to show that K decrypts $ctext = c_0||c_1||\ldots||c_n$ to $block = b_0||b_1||\ldots||b_n$, where *ctext* is in *chash* and *block* is in *bhash*. Bob and the TTP execute VerifyKey, as shown in Algorithm 5.2.3.

Algorithm VerifyKey will detect if chunk c_i does not decrypt correctly. We call such a chunk *corrupted*. If Bob corrupts an nth fraction of the chunks, and the TTP verifies k chunks, then the TTP will catch Bob with probability $1-(1-n)^k$. Suppose Bob corrupts 10% of the chunks. To catch Bob with 90% probability, the TTP needs to check 22 chunks; to catch Bob with 80% probability, the TTP needs to check 16 chunks. This approach might not deter a *malicious* Bob who just wants to perform a denial of service attack. However, a *selfish* Bob who wants to try to get paid for a bad file block would be deterred. This is good enough for our purposes.

5.2.3 Bartering for Files

We present a new protocol that lets Alice and Bob perform a fair exchange of two files. The exchange proceeds in two phases. First, Alice and Bob give each other an unendorsed e-coin and an escrow of the endorsement. This establishes a collateral that an aggrieved party can collect if something goes wrong. In the second phase, Alice and Bob perform a fair exchange of the file. If the exchange fails, the wronged party can ask the TTP to endorse the e-coin. As long as Alice and Bob are honest, they can continue in a bartering relationship indefinitely using the same e-coin as collateral.

Suppose Alice has $block_A$ and she wants $block_B$ which is owned by Bob. They both get $hash_A = \mathrm{MHash}(block_A)$, $hash_B = \mathrm{MHash}(block_B)$ from a trusted authority (*i.e.* the tracker). They perform

the exchange as follows:

1. Alice chooses a new signing key $(sk_{\mathcal{A}}, pk_{\mathcal{A}})$ and gives $pk_{\mathcal{A}}$ to Bob. Bob does the same, responding with $pk_{\mathcal{B}}$.

2. Alice creates an endorsed e-coin $(coin'_A, endorsement)$ and calculates $escrow_A = \mathsf{Escrow}_{TTP}(endorsement, contract)$, where the *contract* states that the TTP can endorse $coin'$ for anyone who presents some $contract'$ that is (1) signed by $pk_{\mathcal{A}}$ and (2) whose terms are fulfilled. Alice gives $(coin'_A, escrow_A)$ to Bob, who performs the corresponding operation and gives Alice $(coin'_B, escrow_B)$.

3. Alice calculates a ciphertext $ctext_A$ and a commitment to the decryption key $K'_A = \mathsf{Com}(K_A)$. Alice gives $(ctext_A, K'_A)$ to Bob. Bob similarly computes $(ctext_B, K'_B)$ and gives it to Alice.

4. Alice and Bob both compute $contract' = (pk_{TTP}, pk_{\mathcal{A}}, K'_A, \text{MHash}(ctext_A), hash_A, pk_{\mathcal{B}}, K'_B, \text{MHash}(ctext_B), hash_B)$. This contract states that to collect collateral, one of two conditions must be met: (1) the owner of $pk_{\mathcal{B}}$ can prove that the opening of K'_A does not decrypt a ciphertext corresponding to $\text{MHash}(ctext_A)$ to a plaintext corresponding to $hash_A$, or (2) the owner of $pk_{\mathcal{A}}$ can prove that the opening of K'_B does not decrypt a ciphertext corresponding to $\text{MHash}(ctext_B)$ to a plaintext corresponding to $hash_B$. This can be proved using standard techniques from Merkle hashes.

5. Alice gives Bob her signature on $contract'$ and Bob gives Alice his signature on $contract'$.

6. Alice and Bob execute a fair exchange protocol where Alice gets K_B (the opening of K'_B) and Bob gets K_A (the opening of K'_A). This can be done using Asokan *et al.* fair exchange [5].

7. If K_B does not decrypt $ctext_B$ correctly, Alice goes to the TTP with the signed $contract'$, $escrow_B$, K_B, a proof showing that $ctext_B$ did not decrypt correctly, and a proof that she knows the secret key corresponding to $pk_{\mathcal{A}}$. The TTP would give Alice the endorsement to Bob's e-coin and his signature on the e-coin. Alice can bring the endorsed e-coin to the bank and deposit it in her account. Bob would do the same if K_A is incorrect.

Note: Showing that $ctext_B$ does not decrypt correctly can be done efficiently. Alice gives the TTP the signed $contract'$, K_B, and a *chunk* that does not decrypt correctly. The TTP can check that K_B is the promised opening of K'_B. Then the TTP can test if (1) *chunk* is in $ctext_B$, (2) $\text{MHash}(ctext_B)$ is in $chash_B$ and (3) $\mathsf{Decrypt}(K_B, chunk)$ is *not* in $hash_B$.

Steps 1 and 2 of the protocol only have to be done once to establish a bartering relationship between Alice and Bob. Subsequently, Alice and Bob can perform steps 3–7 to exchange a block. The bartering protocol has more efficient conflict resolution. If Bob cheats Alice, Alice can show the TTP which chunk decrypted incorrectly. As a result, (1) conflict resolution is more efficient and (2) a cheating Bob is caught with *overwhelming probability*. Finally, we note that, bartering two files is more efficient for the users than executing two purchase protocols; this is because the Asokan *et*

al [5] fair exchange only has to be performed once instead of twice (per block). This is true even if Alice and Bob decide to preserve anonymity by using new signing keys and e-coins each time they exchange files.

Security. The main security challenge is to ensure that Alice cannot deposit the e-coin she put up for collateral (and that Bob cannot deposit his collateral e-coin). We have to make an assumption about the endorsed e-cash deposit protocol: the bank can verify the contract associated with an endorsed e-coin. Specifically, the bank will have to verify that the TTP signed the e-coin (the TTP's signing key is included in the contract, so the bank does not have to know the TTP's identity in advance). As a result, to deposit the e-coin under $contract'$, Alice has to get the TTP's signature. Alice cannot enforce clause (1) of the contract because she does not know Bob's secret key. Alice can enforce clause (2) only if Bob cheats, in which case she is entitled to get her e-coin back. The only other option Alice has is to deposit her e-coin under a new contract. If Alice does this, and Bob later deposits the endorsed e-coin under the old contract, the bank will see that Alice double-spent an e-coin. Due to the construction of endorsed e-cash, if the same e-coin is deposited under two different contracts, the bank can trace the owner of the e-coin. Thus Alice cannot deposit her own collateral e-coin unless Bob cheats. The same argument shows that Bob also cannot deposit his own collateral e-coin unless Alice cheats.

5.3 Other Applications

Endorsed e-cash can be used to add incentives to many distributed peer-to-peer systems. This section gives some insights on how to endorsed e-cash be used in several different applications.

Distributed Lookup. In BitTorrent, a user contacts a centralized tracker for a list of neighbors who are likely to have the file the user wants. It would be nice to eliminate this bottleneck. There are many other situations when a user would like to search a peer-to-peer network for a content provider. Thus, we want to introduce incentives to *encourage* peers to help other users find files and to *discourage* unnecessary lookup queries.

One approach is to pay each node along the query path; the interaction between search depth and query price has been studied in random-tree networks [66]. To avoid flooding and scale efficiently, we would rather lookup using a structured Distributed Hash Tree (DHT)such as Chord [94]. Incentivizing the distributed lookup remains an open problem even though some other incentive systems [99, 78] assumed the presence of cooperative DHT layer.

A DHT distributes data amongst a set of peers. A user searching for a particular file might not know exactly who has the file. However, the user know how far he is from the file and can tell from any other user's id, how far that user is from the file. Furthermore, each user knows enough different peers that given any filename, the user can locate a peer who is half-way closer to the file. If Alice wants a file, she knows somebody who is half-way closer to the file. And that person knows somebody who is half-way closer. In a perfect world, Alice can hop from peer to peer until she zeroes in on the file. In the real world, a selfish peer might not want to waste bandwidth responding to Alice's queries.

We can use endorsed e-cash to create incentives for users to participate in a DHT look-up. Suppose Alice wants to find some file. She asks Bob, the closest peer she knows to the file, for the next hop. Alice pays for this information using endorsed e-cash. The contract specifies that the e-coin is released only if the next hop node is at least half-way closer to the file than Bob (which can be checked using hashes). Then Alice would contact the next node and repeat the process. Alice pays all peers that help her find the file, encouraging peers to cooperate, and discouraging fake lookups.

Distributed Storage. A distributed storage system allows a user to backup her data on another peer's machine. Suppose Alice wants to backup her data, and Bob offers her the use of his machine. If Alice pays Bob upfront, then Bob has no incentive to store the data, he already got the money. However, if Alice pays him upon retrieval, then if she decides she no longer needs the data, she would never pay and Bob would have wasted his resources storing her data.

We propose that Alice pays Bob upfront, but Bob give her a *warranty* check in return (via a fair exchange protocol) that contains a Merkle hash of the data (for the TTP to easily verify without downloading the whole data), an unendorsed e-coin, and an escrow of the endorsement. If Bob ever loses or corrupts the data, the arbiter would decrypt the escrow and pay Alice for her damage. The bank can ensure that Bob always maintains a balance large enough to cover his liability. When Alice

gets her data, the check is invalidated.

Distributed computation. In current distributed computing projects, like Rosetta@Home [87], people voluntarily donate their excess CPU cycles to perform computation-heavy tasks. We can transform this one-way system into a mutually beneficial computing cluster. Users can accept outside jobs when their CPU is not fully loaded, and pay other users to perform some computations when they need more resources.

Suppose Alice wants some computation-heavy task to be done (although I/O-heavy tasks may also be considered). Endorsed e-cash can provide an incentive for users to not only contribute in this computation but also to perform it correctly. For some problems, such as graph coloring (in NP), verifying the correctness of the answer is easy, and so can be a part of the contract. In optimization problems, the contract may specify that the best answer within a deadline will be paid (the most), and again the verification is easy. Unfortunately, this is not true for some other types of problems. Hence, Alice should off-load the job to multiple independent users [3], and pay the majority in case of a dispute.

Chapter 6

Clone Wars

We can use e-cash to perform anonymous authentication. A user withdraws a wallet of e-coins. To authenticate, the user produces an e-coin. Thus, the user can perform some action once for every e-coin in his wallet.

Suppose we want to allow the user to perform some action n times every time period (e.g. download 30 songs a month). In Section 6.1 we show how to let the user refresh his wallet of e-coins every time period without interacting with the bank.

Since we introduce the concept of time, users must worry about accidentally double-spending their e-coins if their clocks are reset. For this reason, in Section 6.2 and Section 6.3, we introduce glitch protection. We allow users to double-spend a limitted number of e-coins without compromising their anonymity. The bank can detect (and reject) the double-spent e-coins, but cannot link them to the honest-but-faulty user unless the user double-spends too often.

The results in this chapter originally appeared in "How to win the clone wars: efficient periodic n-times anonymous authentication," [23].

6.1 Periodic Authentication

We use a very simple trick to transform CHL [24] compact e-coins into authentication e-tokens that can be refreshed every time period. Recall the construction of compact e-cash outlined in Section 2.4.2. A wallet contains upto n e-coins, numbered $0....n-1$. The user maintains a counter J and constructs an e-coin by evaluating a pseudo-random function on J. Since the value of J remains hidden, the user must perform an SRSA range proof to demonstrate that $0 \leq J < n-1$.

To allow periodic authentication, we need to expand the range of J. Suppose the first few bits of J correspond to the time period, while the remaining $\log n$ bits of J are a counter 0 to n (where n is the maximum number of e-tokens per time period). A user could prove that $2^T \leq J < 2^T + n$.

There is only one slight difficulty with this approach. CHL compact e-cash is efficient because it uses the Dodis-Yampolskiy pseudo-random function to transform the counter J into the e-coin serial number and double-spending equation. Dodis and Yampolskiy [48] proved the their pseudo-random function is secure only on imputs from a restricted domain of $\log k$ bits (where k is the security paramter). For periodic anonymous authentication, we want to expand the range of J as much as possible. Camenisch et al. [23] show that the Dodis-Yampolskiy pseudo-random function is secure for inputs from a domain of k-bits, under the SDDHI assumption, and that the SDDHI assumption holds in the generic group model.

6.1.1 Definition

Our definitions for periodic n-times anonymous authentication are based on the e-cash definitions of [24] and [25]. We define a scheme where users $\mathcal{U}$ obtain e-token dispensers from the issuer $\mathcal{I}$, and each dispenser can dispense up to n anonymous and unlinkable e-tokens per time period, but no more; these e-tokens are then given to verifiers $\mathcal{V}$ that guard access to a resource that requires authentication (e.g., an on-line game). $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{I}$ interact using the following algorithms:

IKeygen$(1^k, params)$ is the key generation algorithm of the e-token issuer $\mathcal{I}$. It takes as input 1^k and, if the scheme is in the common parameters model, these parameters *params*. It outputs a key pair $(pk_{\mathcal{I}}, sk_{\mathcal{I}})$. Assume that *params* are appended as part of $pk_{\mathcal{I}}$ and $sk_{\mathcal{I}}$.

UKeygen$(1^k, pk_{\mathcal{I}})$ creates the user's key pair $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$ analogously.

Obtain$(\mathcal{U}(pk_{\mathcal{I}}, sk_{\mathcal{U}}, n), \mathcal{I}(pk_{\mathcal{U}}, sk_{\mathcal{I}}, n))$ At the end of this protocol, the user obtains an e-token dispenser D, usable n times per time period and (optionally) the issuer obtains tracing information t_D and revocation information r_D. $\mathcal{I}$ adds t_D and r_D to a record $R_{\mathcal{U}}$ which is stored together with $pk_{\mathcal{U}}$.

Show$(\mathcal{U}(D, pk_{\mathcal{I}}, t, n), \mathcal{V}(pk_{\mathcal{I}}, t, n))$. Shows an e-token from dispenser D in time period t. The verifier outputs a token serial number (TSN) S and a transcript τ. The user's output is an updated e-token dispenser D'.

Identify$(pk_{\mathcal{I}}, S, \tau, \tau')$. Given two records (S, τ) and (S, τ') output by honest verifiers in the Show protocol, where $\tau \neq \tau'$, computes a value $s_{\mathcal{U}}$ that can identify the owner of the dispenser D that generated TSN S.

The value $s_{\mathcal{U}}$ may also contain additional information specific to the owner of D that (a) will convince third parties that $\mathcal{U}$ is a violator (weak exculpability), that (b) will convince third parties that $\mathcal{U}$ double-showed this e-token (strong exculpability), or that (c) can be used to extract all token serial numbers of $\mathcal{U}$ (traceability).

A periodic n-times anonymous authentication scheme needs to fulfill the following three properties:

Soundness. Given an honest issuer, a set of honest verifiers are guaranteed that, collectively, they will not have to accept more than n e-tokens from a single e-token dispenser in a single time period. There is a knowledge extractor $\mathcal{E}$ that executes u Obtain protocols with all adversarial users and produces functions, $f_1, \ldots, f_u$, with $f_i : \mathbb{T} \times \mathbb{I} \rightarrow \mathbb{S}$. $\mathbb{I}$ is the index set $[0..n-1]$, $\mathbb{T}$ is the domain of the time period identifiers, and $\mathbb{S}$ is the domain of TSN's. Running though all $j \in \mathbb{I}$, $f_i(t, j)$ produces all n TSNs for dispenser i at time $t \in \mathbb{T}$. We require that for every adversary, the probability that an honest verifier will accept S as a TSN of a Show protocol executed in time period t, where $S \neq f_i(j, t)$, $\forall 1 \leq i \leq u$ and $\forall 0 \leq j < n$ is negligible.

Identification. There exists an efficient function ϕ with the following property. Suppose the issuer and verifiers $\mathcal{V}_1, \mathcal{V}_2$ are honest. If $\mathcal{V}_1$ outputs (S, τ) and $\mathcal{V}_2$ outputs (S, τ') as the result of Show protocols, then Identify$(pk_{\mathcal{I}}, S, \tau, \tau')$ outputs a value $s_{\mathcal{U}}$, such that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$, the violator's public key. In the sequel, when we say that a user has *reused* an e-token, we mean that there exist (S, τ) (S, τ') that are both output by honest verifiers.

Anonymity. An issuer, even when cooperating with verifiers and other dishonest users, cannot learn anything about an honest user's e-token usage behavior except what is available from

side information from the environment. This property is captured by a simulator $\mathcal{S}$ which can interact with the adversary as if it were the user. $\mathcal{S}$ does not have access to the user's secret or public key, or her e-token dispenser D.

Formally, we create an adversary A that will play the part of the issuer and of all verifiers. A will create the public and private-keys of the issuer and verifiers. Then, A will be given access to an environment Env that is either using real users or a simulator; A must determine which. A can make four types of queries to Env:

EnvSetup(1^k) generates the public parameters $params$ (if any) and the private parameters $auxsim$ for the simulator (if there is one).

EnvGetPK(i) returns the public-key of user $\mathcal{U}_i$, generated by UKeygen($1^k, pk_{\mathcal{I}}$).

EnvObtain(i) runs the Obtain protocol with user $\mathcal{U}_i$: Obtain($\mathcal{U}(pk_{\mathcal{I}}, sk_{\mathcal{U}}, n), A(state)$). (We use state to denote whatever state the adversary maintains). We call D_j the dispenser generated the jth time protocol $Obtain$ is run.

EnvShow($j, pk_{\mathcal{I}}, t$) behaves differently depending on whether the environment is using a simulator. If the environment is using real users, it will simply run the Show protocol with the user $\mathcal{U}$ that holds the dispenser D_j: Show($\mathcal{U}(D_j, pk_{\mathcal{I}}, t, n), A(state)$). If the environment is using a simulator $\mathcal{S}$, then it will run the Show protocol with it: Show($S(params, auxsim, pk_{\mathcal{I}})$, $A(state)$); $\mathcal{S}$ will not have access to the dispenser D_j or know who owns it.

An adversary is *legal* if it never asks a user to use the same dispenser to show more than n e-tokens in the same time-period. We say that an e-token scheme preserves anonymity if no computationally bounded legal adversary can distinguish when the environment is playing with users and when it is using a simulator.

6.1.2 Agreeing on the Time

Something as natural as time becomes a complex issue when it is part of a security system. First, it is necessary that the value of time period identifier t be the same for all users that show e-tokens in that period. Secondly, it should be used only for a single period, i.e., it must be unique. Our construction in Section 6.1.3 allows for the use of arbitrary time period identifiers, such as those negotiated using the hash tree protocol in [44]. The same is true for the basic glitch protection scheme in Section 6.2. For window glitch protection, in Section 6.3, we assume a totally ordered set of time period identifiers.

If all parties have perfect clocks, then the current time (truncated in order to get the desired period size) fulfills all required properties. Since perfect clocks may be an unrealistic assumption, one of the parties must be motivated to enforce correctness. It is in the interest of verifiers to ensure that all users that show an e-token during a particular time period use the same time period identifier; otherwise, a dishonest user could create extra e-tokens within one time period by using different time period identifiers. Users are also interested in ensuring they all use the same time period identifier;

otherwise, a verifier could link e-tokens that use similarly biased time period identifiers. In addition, users have a strong interest in ensuring that time period identifiers are unique, i.e. that they are never reused. Otherwise, e-tokens from different time periods would look like clones (even if they are not) and a verifier will be able to learn the user's identity.

Damgård et al. [44] describe a protocol that allows multiple users and a single verifier to agree on the same unique value for a time period identifier by having every user contribute sufficient randomness (i.e. uniqueness) to it. Their solution allows users to agree on a unique value without keeping additional state. Their approach can be used in our basic system Section 6.1.3, and for glitch protection (Section 6.2), but not in window glitch protection (Section 6.3), which requires that time period identifiers conform to a metric, i.e. that it is possible to compute the value for future time period identifiers from the current one.

PRACTICE: In the real world, uniqueness can be enforced by checking that the system clock has not been turned back since the last invocation. Sameness for a given global period is more difficult to ensure. It is impossible to have a global notion of time in a distributed systems, so the only thing we can hope for, is to get the processes to agree on the time within a specific bound. Thus, this remains a possible line of attack for cheating verifiers. The situation can be improved by avoiding running the protocol at period boundaries.

Another practical decision is whether we want to have users announce the time, and verifiers check it, or whether we want to have verifiers announce the time and users check it. Each approach allows different attacks, e.g., by manipulating the users' time servers.

6.1.3 Construction

We use CHL e-cash as a starting point. We perform a zero-knowledge proof to show that an e-token serial number $S = \mathsf{DY}_{g,s}(J)$, where s is signed by the bank and J is a counter between 1 and n. (To prove that $1 \leq J \leq n$, we need to perform an SRSA range proof.) By assuming the full Dodis-Yampolskiy PRF, we can extend the range of legal values for J to $O(2^k)$, where k is our security parameter. As a result, we can use the first bits of J can represent the time period, while the last few bits can represent the token counter. Moreover, we can even add additional parameters. We modify the range proof to show that $2^T \leq J \leq 2^T + n$, where T is the time period and n is the maximum number of tokens per time period.

Let k be a security parameter. We choose a group $\langle g \rangle = \mathbb{G}$ of order q, where q is a k-bit prime. Let $l_q \in O(k)$, l_x, l_{time}, and l_{cnt} be system parameters such that $l_q \geq l_x \geq l_{\text{time}} + l_{\text{cnt}} + 2$ and $2^{l_{\text{cnt}}} - 1 > n$, where n is the number of tokens we allow per time period. In the following, we assume implicit conversion between binary strings and integers, e.g., between $\{0,1\}^l$ and $[0, 2^l - 1]$. For suitably defined l_{time}, l_{cnt}, and l_x define the function $c : \{0,1\}^{l_x - l_{\text{time}} - l_{\text{cnt}}} \times \{0,1\}^{l_{\text{time}}} \times \{0,1\}^{l_{\text{cnt}}} \to \{0,1\}^{l_x}$ as:

$$c(u, v, z) := \left(u 2^{l_{\text{time}}} + v\right) 2^{l_{\text{cnt}}} + z \ .$$

Let $F_{(g,s)}(u, v, z) := \mathsf{DY}_{g,s}(c(u, v, z))$.

Issuer Key Generation: In $\mathsf{IKeygen}(1^k, params)$, the issuer $\mathcal{I}$ generates two cyclic groups:

1. A group $\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle = \mathbf{G}$ of composite order $\mathbf{p'q'}$ that can be realized by the multiplicative group of quadratic residue modulo a special RSA modulus $N = (2\mathbf{p'}+1)(2\mathbf{q'}+1)$. In addition to CL signatures, this group will be needed for zero-knowledge proofs of knowledge used in the sequel. Note that soundness of these proof systems is computational only and assumes that the prover does not know the order of the group.

2. A group $\langle g \rangle = \langle \tilde{g} \rangle = \langle h \rangle = \mathbb{G}$ of prime order q with $2^{l_q - 1} < q < 2^{l_q}$.

The issuer must also prove in zero-knowledge that N is a special RSA modulus, and $\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle$ are quadratic residues modulo N. In the random oracle model, one non-interactive proof may be provided. In the plain model, the issuer must agree to interactively prove this to anyone upon request.

Furthermore, the issuer generates a CL signature key pair (pk, sk) set in group $\mathbf{G}$. The issuer's public-key will contain $(\mathbf{g}, \mathbf{h}, \mathbf{G}, g, \tilde{g}, h, \mathbb{G}, pk)$, while the secret-key will contain all of the information.

User Key Generation: In $\mathsf{UKeygen}(1^k, pk_{\mathcal{I}})$, the user chooses a random $sk_{\mathcal{U}} \in \mathbb{Z}_q$ and sets $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}} \in \mathbb{G}$.

Get e-Token Dispenser: $\mathsf{Obtain}(\mathcal{U}(pk_{\mathcal{I}}, sk_{\mathcal{U}}, n),\ \mathcal{I}(pk_{\mathcal{U}},\ sk_{\mathcal{I}}, n))$. Assume that $\mathcal{U}$ and $\mathcal{I}$ have mutually authenticated. A user $\mathcal{U}$ obtains an e-token dispenser from an issuer $\mathcal{I}$ as follows:

1. $\mathcal{U}$ and $\mathcal{I}$ agree on a commitment C to a random value $s \in \mathbb{Z}_q$ as follows:
   (a) $\mathcal{U}$ selects s' at random from $\mathbb{Z}_q$ and computes $C' = \mathsf{PedCom}(sk_{\mathcal{U}}, s'; r) = g^{sk_{\mathcal{U}}} \tilde{g}^{s'} h^r$.
   (b) $\mathcal{U}$ sends C' to $\mathcal{I}$ and proves that it is constructed correctly.
   (c) $\mathcal{I}$ sends a random r' from $\mathbb{Z}_q$ back to $\mathcal{U}$.
   (d) Both $\mathcal{U}$ and $\mathcal{I}$ compute $C = C'\tilde{g}^{r'} = \mathsf{PedCom}(sk_{\mathcal{U}}, s'+r'; r)$. $\mathcal{U}$ computes $s = s'+r' \bmod q$.

2. $\mathcal{I}$ and $\mathcal{U}$ execute the CL signing protocol on commitment C. Upon success, $\mathcal{U}$ obtains σ, the issuer's signature on $(sk_{\mathcal{U}}, s)$. This step can be efficiently realized using the CL protocols [26, 27] in such a way that $\mathcal{I}$ learns nothing about $sk_{\mathcal{U}}$ or s.

3. $\mathcal{U}$ initializes counters $T := 1$ (to track the current period) and $J := 0$ (to count the e-tokens shown in the current time period). $\mathcal{U}$ stores the e-token dispenser $D = (sk_{\mathcal{U}}, s, \sigma, T, J)$.

Use an e-Token: $\mathsf{Show}(\mathcal{U}(E, pk_{\mathcal{I}}, t, n), \mathcal{V}(pk_{\mathcal{I}}, t, n))$. Let t be the current time period identifier with $0 < t < 2^{l_{\text{time}}}$. (We discuss how two parties might agree on t in Section 6.1.2.) A user $\mathcal{U}$ reveals a single e-token from a dispenser $D = (sk_{\mathcal{U}}, s, \sigma, T, J)$ to a verifier $\mathcal{V}$ as follows:

1. $\mathcal{U}$ compares t with T. If $t \neq T$, then $\mathcal{U}$ sets $T := t$ and $J := 0$. If $J \geq n$, abort!

2. $\mathcal{V}$ sends to $\mathcal{U}$ a random $R \in \mathbb{Z}_q^*$.

3. $\mathcal{U}$ sends to $\mathcal{V}$ a token serial number S and a double spending tag E computed as follows:

$$S = F_{(g,s)}(0, T, J), \qquad E = pk_{\mathcal{U}} \cdot F_{(g,s)}(1, T, J)^R$$

4. $\mathcal{U}$ and $\mathcal{V}$ engage in a zero-knowledge proof of knowledge of values $sk_{\mathcal{U}}$, s, σ, and J such that:
   (a) $0 \leq J < n$,
   (b) $S = F_{(g,s)}(0, T, J)$,
   (c) $E = g^{sk_{\mathcal{U}}} \cdot F_{(g,s)}(1, T, J)^R$,
   (d) VerifySig$(pk_{\mathcal{I}}, (sk_{\mathcal{U}}, s), \sigma)$=true.

5. If the proof verifies, $\mathcal{V}$ stores (S, τ), with $\tau = (E, R)$, in his database. If he is not the only verifier, he also submits this tuple to the database of previously shown e-tokens.

6. $\mathcal{U}$ increases counter J by one. If $J \geq n$, the dispenser is empty. It will be refilled in the next time period.

Technical Details. The proof in Step 4 is done as follows:

1. $\mathcal{U}$ generates the commitments $C_J = g^J h^{r_1}$, $C_u = g^{sk_{\mathcal{U}}} h^{r_2}$, $C_s = g^s h^{r_3}$, and sends them to $\mathcal{V}$.

2. $\mathcal{U}$ proves that C_J is a commitment to a value in the interval $[0, n-1]$ using standard techniques [34, 30, 18].

3. $\mathcal{U}$ proves knowledge of a CL signature from $\mathcal{I}$ for the values committed to by C_u and C_s in that order. This step can be efficiently realized using the CL protocols [26, 27].

4. $\mathcal{U}$ as prover and $\mathcal{V}$ as verifier engage in the following proof of knowledge, using the notation by Camenisch and Stadler [32]:

$$\begin{aligned} PK\{(\alpha, \beta, \delta, \gamma_1, \gamma_2, \gamma_3) : g &= (C_s g^{c(0,t,0)} C_J)^{\alpha} h^{\gamma_1} \wedge \\ S = g^{\alpha} \ \wedge \ g &= (C_s g^{c(1,t,0)} C_J)^{\beta} h^{\gamma_2} \wedge \\ C_u &= g^{\delta} h^{\gamma_3} \wedge E = g^{\delta} (g^R)^{\beta}\} \ . \end{aligned}$$

   $\mathcal{U}$ proves she knows the values of the Greek letters; all other values are known to both parties.

Let us explain the last proof protocol. From the first step we know that C_J encodes some value $\hat{J}$ with $0 \leq \hat{J} < n$, i.e., $C_J = g^{\hat{J}} h^{\hat{r}_J}$ for some $\hat{r}_J$. From the second step we know that C_s and C_u encoded some value $\hat{u}$ and $\hat{s}$ on which the prover $\mathcal{U}$ knows a CL signature by the issuer. Therefore, $C_s = g^{\hat{s}} h^{\hat{r}_s}$ and $C_u = g^{\hat{u}} h^{\hat{r}_u}$ for some $\hat{r}_s$ and $\hat{r}_u$. Next, recall that by definition of $c(\cdot, \cdot, \cdot)$ the term $g^{c(0,t,0)}$ corresponds to $g^{t2^{l_{\text{cnt}}}}$. Now consider the first term $g = (C_s g^{c(0,t,0)} C_J)^{\alpha} h^{\gamma_1}$ in the proof protocol. We can now conclude the prover $\mathcal{U}$ knows values $\hat{a}$ and $\hat{r}$ such that $g = g^{(\hat{s}+t2^{l_{\text{cnt}}}+\hat{J})\hat{a}} h^{\hat{r}}$ and $S = g^{\hat{a}}$. From the first equation is follows that $\hat{a} = (\hat{s} + (t2^{l_{\text{cnt}}} + \hat{J}))^{-1} \pmod{q}$ must hold provided that $\mathcal{U}$ is not privy to $\log_g h$ (as we show via a reduction in the proof of security) and

thus we have established that $S = F_{(g,\hat{s})}(c(0,t,\hat{J}))$ is a valid serial number for the time period t. Similarly one can derive that $E = g^{\hat{u}} \cdot F_{(g,\hat{s})}(c(1,t,\hat{J}))^R$, i.e., that E is a valid double-spending tag for time period t.

Identify Cheaters: Identify$(pk_{\mathcal{I}}, S, (E,R), (E',R'))$. If the verifiers who accepted these tokens were honest, then $R \neq R'$ with high probability, and proof of validity ensures that $E = pk_{\mathcal{U}} \cdot f_s(1,T,J)^R$ and $E' = pk_{\mathcal{U}} \cdot f_s(1,T,J)^{R'}$. The violator's public key can now be computed by first solving for $f_s(1,T,J) = (E/E')^{(R-R')^{-1}}$ and then computing $pk_{\mathcal{U}} = E/f_s(1,T,J)^R$.

6.1.4 Efficiency

To analyze the efficiency of our scheme, it is sufficient to consider the number of (multi-base) exponentiations the parties have to do in $\mathbb{G}$ and $\mathbf{G}$. In a decent implementation, a multi-base exponentiation takes about the same time as a single-base exponentiation, provided that the number of bases is small. For the analysis we assume that the Strong RSA based CL-signature scheme is used.

Obtain: both the user and issuer perform 3 exponentiations in $\mathbf{G}$. Show: the user performs 12 multi-base exponentiation in $\mathbb{G}$ and 23 multi-base exponentiations in $\mathbf{G}$, while the verifier performs 7 multi-base exponentiation in $\mathbb{G}$ and 13 multi-base exponentiations in $\mathbf{G}$. If n is odd, the user only needs to do 12 exponentiations in $\mathbf{G}$, while the verifier needs to do 7. To compare ourselves to the Damgård et al. [44] scheme, we set $n = 1$. In this case, Show requires that the user perform 12 multi-base exponentiation in $\mathbb{G}$ and 1 multi-base exponentiations in $\mathbf{G}$ and the verifier perform 7 multi-base exponentiation in $\mathbb{G}$ and 1 multi-base exponentiations in $\mathbf{G}$. Damgård et al. requires $57{+}68r$ exponentiations in $\mathbb{G}$, where r is the security parameter (i.e., 2^{-r} is the probability that the user can cheat). Depending on the application, r should be at least 20 or even 60. Thus, our scheme is an order of magnitude more efficient than Damgård et al.

6.1.5 Security

Theorem 6.1.1. *Protocols* IKeygen, UKeygen, Obtain, Show, *and* Identify *described above achieve* soundness, identification, *and* anonymity *properties in the plain model assuming* Strong RSA, *and* y-DDHI *if* $l_x \in O(\log k)$ *or* SDDHI *otherwise.*

Proof. Soundness. Informally, in our system, tokens are unforgeable, because each token serial number (TSN) is a deterministic function $F_{(g,s)}(0,t,J)$ of the seed s, the time period t, and $J \in [0, n-1]$. Thus, there are only n valid TSNs per time period, and since a user must provide a ZK proof of validity for the token, to show $n+1$ or more times requires that two shows use the same TSN by the pigeonhole principle.

More formally, we will describe a knowledge extractor $\mathcal{E}$ that, after executing u Obtain protocols with an adversary $\mathcal{A}$ acting on behalf of all malicious users, can output functions $f_1, \ldots, f_u$ that allow to compute all possible token serial numbers that $\mathcal{A}$ could output in any given time period t. Let n be the number of shows allowed per time period. Our extractor $\mathcal{E}$ operates as follows:

1. In step one of Obtain, $\mathcal{E}$ behaves as an honest issuer and agrees on a Pedersen commitment $C = g^{sk_i}\tilde{g}^s h^r = \mathsf{PedCom}(sk_i, s; r)$ with $\mathcal{A}$, where sk_i is whatever secret key $\mathcal{A}$ choses to use and s is the PRF seed.

2. In step two, $\mathcal{E}$ must run the CL signing protocol with $\mathcal{A}$ to provide $\mathcal{A}$ with a signature on (sk_i, s). As part of the CL protocol, $\mathcal{A}$ is required to prove knowledge of (α, β, γ) such that $C = g^\alpha \tilde{g}^\beta h^\gamma$. There are a number of ways to guarantee that this proof of knowledge is *extractable*; in this step, $\mathcal{E}$ employs one of the methods of CL to extract the secrets $(sk_\mathcal{U}, s)$ from $\mathcal{A}$. (Here we will enforce that Obtain protocols must be run sequentially, so that rewinding does not become a problem.)

3. $\mathcal{E}$ outputs the function f_i as the description of the Dodis-Yampolskiy pseudorandom function, together with the seed s.

Since $\mathcal{E}$ knows the value s used for every dispenser, it can calculate the token serial number $S := F_{(g,s)}(0, t, J)$. The CL signatures and its protocols are secure under the Strong RSA assumption.

Identification of Violators. Suppose (S, E, R) and (S, E', R') are the result of two Show protocols with an honest verifier(s). Since the verifier(s) was honest, it is the case that $R \neq R'$ with high probability since an honest verifier chooses $R \in \mathbb{Z}_q^*$ at random. Due to the soundness of the ZK proof of validity, it must be the case that $E = pk_\mathcal{U} \cdot F_{(g,s)}(1, t, J)^R$ and $E' = pk_\mathcal{U} \cdot F_{(g,s)}(1, t, J)^{R'}$ for the same values of s, t, J and $pk_\mathcal{U}$. Thus, the violator's public key can be computed as follows:

$$\left(\frac{E^{1/R}}{E'^{1/R'}}\right)^{\frac{RR'}{R'-R}} = \left(\frac{g^{sk_\mathcal{U}/R} \cdot F_{(g,s)}(1,t,J)}{g^{sk_\mathcal{U}/R'} \cdot F_{(g,s)}(1,t,J)}\right)^{\frac{RR'}{R'-R}} = \left(g^{sk_\mathcal{U}(1/R - 1/R')}\right)^{\frac{RR'}{R'-R}} = g^{sk_\mathcal{U}} = pk_\mathcal{U}.$$

To be explicit with respect to our definition of this property, the value $s_\mathcal{U} := pk_\mathcal{U}$ and the function ϕ is the identity.

Anonymity. Informally, the intuition for anonymity is that the issuer does not learn the PRF seed during Obtain. Then showing a token in Show consists of releasing (S, E), where S and E are functions of the PRF (indistinguishable from random) and a zero-knowledge (ZK) proof that the token is valid (reveals one bit). Thus, if a user is honest, nothing about her identity is revealed by two random-looking numbers and a ZK proof.

More formally, will describe a simulator $\mathcal{S}$ which an adversary $\mathcal{A}$ cannot distinguish from an honest user during the Show protocol. Recall that $\mathcal{A}$, playing the role of a coalition of adversarial issuer and verifiers, first runs the Obtain protocol u times with honest users, who then obtain a dispenser D. Let the number of allowed shows per time period be n.

Now, at some point, $\mathcal{A}$ outputs a value $j \in [1, u]$ and a time period t. $\mathcal{A}$ will now execute the Show protocol with either the honest user $\mathcal{U}$ that holds the token dispenser D_j at time t *or* with simulator $\mathcal{S}$, whose input is only the global parameters *params*, t, n, and the issuer's public key $pk_\mathcal{I}$. To impersonate an unknown user, $\mathcal{S}$ behaves as follows:

1. In steps one and two of the Show protocol, $\mathcal{S}$ does nothing.

2. In step three, $\mathcal{S}$ sends to $\mathcal{A}$ random values $(S, E) \in \mathbb{G}^2$.

3. In step four, $\mathcal{S}$ *simulates* a proof of knowledge of (z, s, J, σ) for the statements:

   (a) $0 \leq J < n$,

   (b) $S = F_{(g,s)}(0, t, J)$,

   (c) $E = g^z \cdot F_{(g,s)}(1, t, J)^R$.

   (d) VerifySig$(pk_{\mathcal{I}}, (z, s), \sigma)$=true.

Proving this statement, in the honest setting, follows the standard discrete-logarithm-based Σ-protocol, as we detailed in Section 6.1.3. Thus, for this step, $\mathcal{S}$ can simulate this Σ-protocol in the two standard ways: (1) rewind the adversary (interactive proof) or (2) use its control over the random oracle (non-interactive proof). To prevent any rewinding difficulties, Show protocols should be executed sequentially.

This simulator's behavior is indistinguishable from a user with dispenser D_j. The zero knowledge proof is standard. The random values (S, E) are indistinguishable from the user's real (S', E') due to the security of the Dodis-Yampolskiy PRF, which relies on y-DDHI (for small system parameters) and otherwise SDDHI. Specifically, SDDHI is required for whenever parameter l_x becomes superlogarithmic due to a technicality in the original proof of security for the Dodis-Yampolskiy PRF. □

6.2 Basic Glitch Protection

In our periodic n-times anonymous authentication scheme, a user who shows two tokens with the same TSN becomes identifiable. (Recall that only n unique TSN values are available to a user per time period.) A user might accidentally use the same TSN twice because of hardware breakdowns, clock desychronization, etc. We want to protect the anonymity of users who occasionally cause a *glitch* (repeat a TSN in two different tokens), while still identifying users who cause an excessive amount of glitches. A user might be permitted up to m glitches per *monitoring interval* (e.g., year). Any TSN repetition will be detected, but the user's anonymity will not be compromised until the $(m+1)$st glitch. A token that causes a glitch is called a *clone*.

We divide time periods into monitoring intervals. So if a single time period last a day, then our monitoring interval could be a week, a month, a year, etc. For the basic glitch protection scheme, we will have non-overlaping monitoring intervals; each time period will belong to exactly one monitoring interval.

Suppose a user has u glitches in one monitoring interval. Our goal is to design a scheme where:

- if $u = 0$, all shows are anonymous and unlinkable;
- if $1 \leq u \leq m$, all shows remain anonymous, but a link-id L is revealed, making all clones linkable;
- if $u > m$, the user's public key is revealed.

One can think of link-id L as a pseudonym (per monitoring interval) that is hidden in each token released by the same user (much in the same way that the user's public key was hidden in each token released by a user in the basic scheme). If tokens (S, τ) and (S, τ') caused a glitch, then we call (S, τ, τ') a *glitch tuple*, where by definition $\tau \neq \tau'$. We introduce a new function GetLinkId that takes as input a glitch tuple and returns the link-id L. Once $m+1$ clones are linked to the same pseudoym L, there is enough information from these collective original and cloned transcripts to compute the public key of the user.

We continue to use identifier $t \in \mathbb{T}$ for (indivisible) time periods. Identifier $v \in \mathbb{V}$ refers to a monitoring interval. We give two glitch protection schemes: Section 6.2 considers disjoint monitoring intervals, while Section 6.3 works on overlapping monitoring intervals. For the first scheme, we assume the existence of an efficient function $M_{\mathbb{V}}$ that maps every time period t to its unique monitoring interval $v \in \mathbb{V}$.

Our basic glitch protection scheme tolerates up to m clones per monitoring interval v; monitoring intervals are disjoint.

6.2.1 Definition

A periodic authentication scheme with basic glitch protectionhas almost the same protocols as a regular periodic authentication scheme. We replace Showwith ShowGPand Identifywith IdentifyGP. We also add a new protocol called GetLinkId.

$\mathsf{ShowGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m), \mathcal{V}(pk_{\mathcal{I}}, t, n, m))$**.** Shows an e-token from dispenser D in time period t and monitoring interval $v = M_{\mathbb{V}}(t)$. The verifier obtains a token serial number S and a transcript τ.

$\mathsf{GetLinkId}(pk_{\mathcal{I}}, S, \tau, \tau')$**.** Given e-tokens (S, τ, τ'), where $\tau \neq \tau'$ by definition, computes a link-id value L.

$\mathsf{IdentifyGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+\ell+1}'))$**.** Given $m+1$ glitch tuples where for each i, $\mathsf{GetLinkId}(S_i, \tau_i, \tau_i')$ produces the same link-id L, computes a value $s_{\mathcal{U}}$ that can be used to compute the public key of the owner of the dispenser D from which the TSNs came.

We need to extend the anonymity and identification properties to handle the fact that honest users might occasionally make clones. Users that never clone should have the same security guarantees as they would in a basic anonymous authentication scheme without glitch protection: the result of every show must be *anonymous* and *unlinkable*. For a glitchy user, the verifier should be able to link clones together, but still be unable to identifier a user that has less than m glitches per monitoring interval.

Defining the extended anonymity property is tricky because we have to let the adversary request clones. We achieve this result by adding a fifth command to our environment: $\mathsf{EnvClone}(token)$. The adversary controls which user shows it which e-token, what dispenser the user is supposed to use, the value of the counter J in the dispenser, and the time-interval for which this e-token was created. Therefore, this command gives it complete control over how and when users make clones. A scheme offers glitch protection if the adversary cannot distinguish between real users and a simulator, as long as it does not ask for too many clones (where too many is determined by the type of glitch protection).

There is a slight problem with this definition. The simulator is supposed to return a clone that is *linked* to some set of other clones generated by the same dispenser. The simulator can easily link clones of the same e-token because they have the same serial number. However, by the definition of anonymity, the simulator does not know what dispenser generated the e-token, nor what were that dispenser's other e-tokens.

Fortunately, the environment *can* keep track of this information. Therefore, the environment will pass an extra argument *linkid* to the simulator whenever the adversary invokes EnvClone (we make the environment do this rather than the adversary to ensure consistency). The *linkid* will be a unique integer independent of the user's id, dispenser, counter, etc. This *linkid* will let the simulator link only those e-tokens that ought to be linked. If the adversary never asks for clones, the anonymity property is the same as for n-times anonymous authentication without glitch protection.

Note: It's easy to see that whenever the adversary asks for an e-token, via either the EnvShow or EnvClone command, the environment can always consult its transcript of past communication to calculate $(\mathcal{U}, j, J, t)$, where $\mathcal{U}$ is the identity of the user, D_j is the dispenser that generated the e-token, J is the counter value D_j used, and t is the time-interval. Thus, we will assume that this information is available to the environment without explicitly calculating it. The environment will

store a table *clones* that will count the number of clones of a particular type; it will use this table to ensure the adversary does not try to create more clones than is allowed by the glitch protection scheme. The environment will also have a table $LinkIDs$ that will store the *linkid*s it will pass to the simulator.

Defining identification is also a little tricky because we need a way to *link* clones that were generated by different values (j, J, t) and to identify users from these clones. We create a new helper function $\mathsf{GetLinkId}(S, \tau_1, \tau_2)$ that outputs a value L that is the same for all clones made by the a dispenser in the same time interval. In otherwords, if a single dispenser made e-tokens (S, τ_1), (S, τ_2), (S', τ_1') and (S', τ_2'), then $\mathsf{GetLinkId}(S, \tau_1, \tau_2) = \mathsf{GetLinkId}(S', \tau_1', \tau_2')$. Using $\mathsf{GetLinkId}$, we can require that the function $\mathsf{IdentifyGP}$ return the public-key of the violator when it gets as input a sufficiently long sequence of clones that are all linked (as defined by $\mathsf{GetLinkId}$). It is upto the designers of a glitch-protection scheme to instantiate $\mathsf{GetLinkId}$ and ensure that it does not clash with the anonymity and glitch protection property.

We formally define the GP Anonymity and GP Identification properties of an n-times anonymous authentication scheme with basic glitch protection.

GP Anonymity. The adversary will interact with the environment. The environment will generate a table *clones* that will count how many clones a dispenser j made in time interval $v \in \mathbb{V}$ and a corresponding list $LinkIDs$, such that $LinkIDs(j, v)$ is unique for every pair (j, v). The first time the adversary invokes $\mathsf{EnvShow}(j, *, t)$ (it does not matter which verifier the adversary uses), the environment will set $clones(j, M_{\mathbb{V}}(t)) = 0$. Whenever the adversary invokes $\mathsf{EnvClone}(etoken)$, before fulfilling the request, the environment will check if $clones(j, v) \geq m$. If yes, the environment will output $error$. Otherwise the environment will fulfill the request and increment $clones(j, v)$; if it is using a simulator, the environment will give the simulator $LinkIDs(j, v)$ as input.

We say that a scheme offers GP Anonymity if, in this game, no computationally bounded adversary can tell if the environment is using real users or a simulator.

GP Identification. Suppose the issuer and verifiers are honest and they receive $m+1$ glitch tuples $\mathsf{Input} = (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}')$ with the same $L = \mathsf{GetLinkId}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$ for all $1 \leq i \leq m+1$. Then with high probability algorithm $\mathsf{IdentifyGP}(pk_{\mathcal{I}}, \mathsf{Input})$ outputs a value $s_{\mathcal{U}}$ for which there exists an efficient function ϕ such that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$, identifying the violator.

6.2.2 Construction

Recall that in our basic scheme, an e-token has three logical parts: a serial number $S = F_{(g,s)}(0, T, J)$, a tag $E = pk_{\mathcal{U}} \cdot F_{(g,s)}(1, T, J)^R$, and a proof of validity. If the user shows a token with TSN S again, then he must reveal $E' = pk_{\mathcal{U}} \cdot F_{(g,s)}(1, T, J)^{R'}$, where $R \neq R'$, and the verifier can solve for $pk_{\mathcal{U}}$ from (E, E', R, R').

Now, in our glitch protection scheme, an e-token has four logical parts: a serial number $S = F_{(g,s)}(0, T, J)$, a tag K that exposes the link-id L if a glitch occurs, a tag E that exposes $pk_{\mathcal{U}}$ if

more than m glitches occur, and a proof of validity.

We instantiate $K = L \cdot F_{(g,s)}(2, T, J)^R$. Now a double-show reveals L just as it revealed $pk_{\mathcal{U}}$ in the original scheme. The link-id for monitoring interval v is $L = F_{(g,s)}(1, v, 0)$.

Once the verifiers get $m + 1$ clones with the same link-id L, they need to recover $pk_{\mathcal{U}}$. To allow this, the user includes tag $E = pk_U \cdot \prod_{i=1}^{m} F_{(g,s)}(3, v, i)^{\rho_i} \cdot F_{(g,s)}(4, T, J)^R$. (Here, it will be critical for anonymity that the user and the verifier *jointly* choose the random values $R, \rho_1, \ldots, \rho_m$.)

Now, suppose a user causes $m + 1$ glitches involving ℓ distinct TSNs. Given $(E, R, \rho_1, \ldots, \rho_m)$ from each of these $(m+\ell+1)$ tokens, the public key of the user can be computed by repeatedly using the elimination technique that allowed the discovery of L from (K, K', R, R'). We have $(m + \ell + 1)$ equations E and $(m+\ell+1)$ unknown *bases* including $pk_{\mathcal{U}}$ and the $F_{(g,s)}(\cdot, \cdot, \cdot)$ values. Thus, solving for $pk_{\mathcal{U}}$ simply requires solving a system of linear equations.

$\mathsf{ShowGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m), \mathcal{V}(pk_{\mathcal{I}}, t, n, m))$. Let $v = M_{\mathbb{V}}(t)$. A user $\mathcal{U}$ shows a single e-token from a dispenser $D = (sk_{\mathcal{U}}, s, \sigma, T, J)$ to a verifier $\mathcal{V}$ as follows:

1. $\mathcal{U}$ compares t with T. If $t > T$, then $\mathcal{U}$ sets $T := t$ and $J := 0$. If $J \geq n$, abort!
2. $\mathcal{V}$ and $\mathcal{U}$ jointly choose $R, \rho_1, \ldots, \rho_m$ uniformly at random from $\mathbb{Z}_q^*$ (see Section 2.2.10 for details).
3. $\mathcal{U}$ sends $\mathcal{V}$ an interval serial number S, a double spending tag K encoding the link-id L, and a special $(m + 1)$-cloning tag E:

$$\begin{aligned} S &= F_{(g,s)}(0, T, J), \\ K &= F_{(g,s)}(1, v, 0) \cdot F_{(g,s)}(2, T, J)^R, \\ E &= pk_{\mathcal{U}} \cdot F_{(g,s)}(3, v, 1)^{\rho_1} \cdots \\ &\qquad \cdot F_{(g,s)}(3, v, m)^{\rho_m} \cdot F_{(g,s)}(4, T, J)^R \end{aligned}$$

4. $\mathcal{U}$ performs a zero-knowledge proof that the values above were correctly computed.
5. If the proof verifies, $\mathcal{V}$ stores (S, τ), where $\tau = (K, E, R, \rho_1, \ldots, \rho_m)$, in his database.
6. $\mathcal{U}$ increments counter J by one. If $J \geq n$ the dispenser is empty. It will be refilled in the next time period.

$\mathsf{GetLinkId}(pk_{\mathcal{I}}, S, (K, E, R, \vec{\rho}), (K', E', R', \vec{\rho'}))$. Returns

$$L = \frac{K}{(K/K')^{(R-R')^{-1}R}}.$$

$\mathsf{IdentifyGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}'))$. Let the $m + 1$ glitch tuples include ℓ distinct TSN values. We extract the values $(E_i, R, \rho_1, \ldots, \rho_m)$ (or $(E_i', R', \rho_1', \ldots, \rho_m')$) from all $m + \ell + 1$ unique transcripts. Now, we use the intuition provided above to solve for $pk_{\mathcal{U}}$.

6.2.3 Security

Theorem 6.2.1. *The construction in Section 6.2.2 is a secure periodic n-times anonymous authentication scheme with basic glitch protection. It fulfills the soundness, GP anonymity and GP identification properties.*

Proof. Soundness: The soundness proof is identical to the soundness proof for the basic n-times anonymous authentication scheme.

GP Anonymity: All we need to do to prove GP anonymity is to explain how the simulator will respond to EnvSetup, EnvShow, and EnvClone.

To ensure consistency, the simulator will store a table $Token$ of information about e-tokens it has previously shown. It will also have a table $Link$ indexed by link-ids provided by the environment. $Link(linkid)$ will be a random number; the simulator will ensure that when the adversary runs GetLinkId on a double-show with link-id $linkid$, the output will always be $Link(linkid)$.

During EnvShow, the simulator will run the Show protocol with the adversary. The simulator will perform steps 1, 2 and 3 as normal; at the end of which it will have random $(R, \rho_1, \ldots, \rho_m)$. In step 4 The simulator will randomly choose $(S, \hat{L}, E) \in \mathbb{G}^3$. In step five, $\mathcal{S}$ *simulates* a proof of knowledge of (z, s, J, σ) for the statements:

(a) $0 \leq J < n$,

(b) $S = F_{(g,s)}(0, T, J)$

(c) $\hat{L} = F_{(g,s)}(1, v, 0) \cdot F_{(g,s)}(2, T, J)^R$

(d) $E = {pk_{\mathcal{U}}}^{\rho_0} \cdot F_{(g,s)}(3, v, 1)^{\rho_1} \cdots F_{(g,s)}(3, v, m)^{\rho_m} \cdot F_{(g,s)}(4, T, J)^R$

(e) VerifySig$(pk_{\mathcal{I}}, (z, s), \sigma)$=true.

Simulating such a proof is a standard operation; see the anonymity proof in Section 6.1.3 for details. Call the e-token resulting from this execution *etoken*. The simulator will store $Token(etoken) = (S, R, \hat{L}, F = \phi)$. If the token is ever cloned with some link-id $linkid$, the simulator will retroactively set F so that $\hat{L} = Link(linkid) \cdot F^R$.

When the adversary invokes EnvClone$(etoken)$, the environment will give the simulator a $linkid$. Then the adversary and the simulator will run through the Show protocol. The simulator will have to produce an e-token that is a clone of *etoken* and that is linked to all other e-tokens with link-id *linkid*.

Once again, the simulator will perform steps 1, 2 and 3 as normal; at the end of which it will have random $(R', R'_1, \ldots, R'_m)$. In step 4, the simulator will perform the following operations: first, it will retrieve $Token(etoken) = (S, R, \hat{L}, F)$. If $F = \phi$, then the simulator will calculate $F = (\hat{L}/Link(linkid))^{1/R}$ and update the entry $Token(etoken)$ accordingly. Then, the simulator will create a new e-token: $S' = S$, $\hat{L}' = Link(linkid)F^{R'}$, and E' will be a random number. In step 5, the simulator will simulate a proof of knowledge of (z, s, J, σ) for the statements:

(a) $0 \leq J < n$,

(b) $S' = F_{(g,s)}(0, T, J)$

(c) $\hat{L}' = F_{(g,s)}(1, v, 0) \cdot F_{(g,s)}(2, T, J)^{R'}$

(d) $E' = pk_{\mathcal{U}}{}^{R'_0} \cdot F_{(g,s)}(3, v, 1)^{R'_1} \cdots F_{(g,s)}(3, v, m)^{R'_m} \cdot F_{(g,s)}(4, T, J)^{R'}$

(e) VerifySig$(pk_{\mathcal{I}}, (z, s), \sigma)$=true.

It will do this in the same manner as for Show. After terminating, the simulator will store $Token(etoken') = (S, R', \hat{L}', F)$.

We sketch out why the output of the simulator is indistinguishable from that of real users: Until the adversary asks for a clone, an e-token is a completely random number. It is generated in the same way as in the original proof of anonymity; therefore the output of the simulator is indistinguishable from that of real users. When the adversary asks for a clone, the simulator retroactively sets the link-id to be $Link(linkid)$, where $linkid$ is provided by the environment. This ensures that the link-ids are consistent. The $(m+1)$-cloning tags E are all random numbers; this is fine because the adversary never gets $m+1$ clones and therefore they should provide no information.

GP Identification: By the soundness property, we know that the glitch-tuples given to the function IdentifyGP$(pk_{\mathcal{I}}, (S_1, \tau_1, \tau'_1), \ldots, (S_{m+1}, \tau_{m+1}, \tau'_{m+1}))$ are correctly formed. The only remaining question is whether the $(m+1)$-clone tags E_i^b contain enough information to solve for $pk_{\mathcal{U}}$.

Since the issuer never learns the seed of the PRF used to calculate the $m+1$-clone tags E_i^b, as far as it is concerned, each tag E_i^b is the product of $m+2$ randomly chosen unknowns. One of the unknowns is $pk_{\mathcal{U}}$, m of the unknowns are the same for every tag from that monitoring interval, and one of the unknowns is unique to the cloned e-token. Therefore, if a user clones ℓ different e-tokens ($|\{S_1, \ldots, S_{m+1}\}| = \ell$), then there are $m+1+\ell$ different unknowns among all the E_i^b. How many distinct tags E_i^b are there? If there are ℓ different multi-shown e-tokens, then there are exactly $m+1+\ell$ different tags E_i^b: there are ℓ distinct E_i and $m+1$ distinct E'_i. With high probability, a randomly generated system of $m+1+\ell$ equations with $m+1+\ell$ unknowns will have a unique solution, in which case GP Identify will find it using Gaussian Elimination.

More specifically if we represent the $y = m+\ell+1$ equations as a $y \times y$ matrix, the success probability P_S corresponds to the probability that a random matrix of this size is invertible. The first vector in your matrix is arbitrary, except that it should not be 0. So there are $(q^y - 1)$ choices. The second vector should be linearly independent of the first; i.e., it should not lie in a 1-dimensional subspace. So there are $(q^y - q)$ choices. The third vector should not lie in a 2-dimensional subspace, so there are $(q^y - q^2)$ choices. Etc. So in general, there are $(q^y-1)\cdot(q^y-q)\cdots(q^y-q^{(y-1)})$ invertible $y \times y$ matrices. To get the probability one divides this number by the total number of matrices, that

is $q^{(y^2)}$. It is easy to see that

$$\begin{aligned} P_S &= \frac{(q^y-1)\cdot(q^y-q)\cdots(q^y-q^{(y-1)})}{q^{(y^2)}} = \left(1-\frac{1}{q^y}\right)\cdot\left(1-\frac{1}{q^{y-1}}\right)\cdots\left(1-\frac{1}{q}\right) \\ &\geq \left(1-\frac{1}{q}\right)\cdot\left(1-\frac{1}{q}\right)\cdots\left(1-\frac{1}{q}\right) \\ &= \left(1-\frac{1}{q}\right)^y = \sum_{i=0}^{y}(-1)^{(i \bmod 2)}\cdot\binom{y}{i}\cdot(1/q)^{y-i} \\ &= 1+\sum_{i=1}^{y}(-1)^{(i \bmod 2)}\cdot\binom{y}{i}\cdot(1/q)^{y-i}. \end{aligned}$$

As q is exponential in the security parameter k, P_S is bounded below by $1-\nu(k)$, with $\nu(k)$ a negligible function in k. □

6.3 Window Glitch Protection

The basic glitch protection scheme prevents users from creating more than m clones in a single monitoring interval. If two neighboring time periods fall in different monitoring intervals, then a malicious user can create m clones in each of them. We want to catch users who make more than m clones within any W consecutive time periods.

6.3.1 Definition

We define an interval of consecutive time-periods to be a window. For convenience, we will consider each time period identifier t to be an integer, and time periods t and $t+1$ to be neighbors. Each time period is in W different windows of size W. If we let a time period define the *end* of a window, then time period t would be in windows $t, t+1, \ldots, t+W-1$.

(m, W)-Window glitch protection allows a user to clone at most m e-tokens during any window of W consecutive time periods. We describe the new protocols associated with a window glitch protection scheme:

$\mathsf{ShowWGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m, W), \mathcal{V}(pk_{\mathcal{I}}, t, n, m, W))$.
Shows an e-token from dispenser D for time period t. The verifier obtains a serial number S and a transcript τ.

$\mathsf{GetLinkIds}(pk_{\mathcal{I}}, S, \tau, \tau')$.
Given two e-tokens (S, τ) and (S, τ'), outputs a *list* of W link-ids $L_1, \ldots, L_W$.

$\mathsf{IdentifyWGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}'))$. Given $m+1$ glitch tuples where for each i, the same link-id L is in the list of link-ids produced by $\mathsf{GetLinkId}(S_i, \tau_i, \tau_i')$, computes a value $s_{\mathcal{U}}$ that can be used to compute the public key of the owner of the dispenser D from which the TSNs came.

We modify the *GP Anonymity* and *GP Identification* properties to apply to window glitch protection.

WGP Anonymity. The environment will keep a table *clones* that will count how many clones every user made during every window of length W, and a table $LinkIDs$ with a random unique entry for each time interval $i \in \mathbb{V}$. Each time the adversary invokes EnvClone, before fullfilling the request, the environment will increment the values at $clones(\mathcal{U}, t), \ldots, clones(\mathcal{U}, t+w-1)$. If any of those result in a value greater than m, the environment will output $error$. Otherwise, the environment will run the Show protocol; if it is using the simulator, the environment will give it $LinkIDs(t), \ldots, LinkIds(t+w-1)$ as input.

WGP Identification. Suppose the issuer and verifiers are honest. Should they receive a list of $m+1$ glitch tuples $\mathsf{Input} = (S_1, \tau_1, \tau_2'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}')$, such that $\exists L : \forall i : L \in \mathsf{GetLinkIds}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$, then with high probability $\mathsf{IdentifyWGP}(pk_{\mathcal{I}}, \mathsf{Input})$ outputs a value $s_{\mathcal{U}}$ for which there exists an efficient function ϕ such that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$, identifying the violator.

6.3.2 Construction

Intuitively, we replicate our basic glitch solution W times to create overlapping windows of W time periods.

$\mathsf{ShowWGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m, W)$. We modify the ShowGP protocol as follows. In step 3, the user and verifier jointly choose random numbers $R_1, \ldots, R_W$ and $\rho_{1,1}, \ldots, \rho_{W,m}$. In step 4, the user calculates essentially the same values S, K, E, except that now she calculates separate K_i and E_i tags for every window in which time period T falls:

$$\begin{aligned} S &= F_{(g,s)}(0, T, J) \\ K_i &= F_{(g,s)}(1, T+i, 0) \cdot F_{(g,s)}(2, T, J)^{R_i} \\ E_i &= pk_{\mathcal{U}} \cdot F_{(g,s)}(3, T+i, 1)^{\rho_{i,1}} \cdots \\ &\qquad \cdot F_{(g,s)}(3, T+i, m)^{\rho_{i,m}} \cdot F_{(g,s)}(4, T, J)^{R_i} \end{aligned}$$

Finally, in step 5, the user proves to the verifier that the values $S, K_1, \ldots, K_W, E_1, \ldots, E_W$ are formed correctly. That, along with the random numbers generated in step 3, forms the transcript stored in steps 6. Step 7 is unchanged.

$\mathsf{GetLinkIds}(pk_{\mathcal{I}}, S, \tau, \tau')$. Returns the link-ids:

$$L_i = \frac{K_i}{(K_i/K_i')^{(R_i - R_i')^{-1} R_i}}, \quad 1 \le i \le m+1.$$

$\mathsf{IdentifyWGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_2'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}'))$. For all i, let $L \in \mathsf{GetLinkIds}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$, that is, let L be the link-id each glitch tuple has in common. Let these $m+1$ glitch tuples include ℓ distinct TSN values. We extract the values $(E_{i,j}, R_i, \rho_{i,1}, \ldots, \rho_{i,m})$ (or $(E_{i,j}', R_i', \rho_{i,1}', \ldots, \rho_{i,m}')$) from all $m+\ell+1$ unique transcripts, where j depends on where L falls in the list $\mathsf{GetLinkIds}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$. Now, we use the same techniques as before to solve for $pk_{\mathcal{U}}$.

6.3.3 Security

Theorem 6.3.1. *The scheme in Section 6.3.2 is a secure periodic n-times anonymous authentication scheme with window glitch protection. It fulfills the soundness, WGP anonymity and WGP identification properties.*

Proof. Soundness: The soundness proof is identical to the soundness proof for the basic n-times anonymous authentication scheme.

WGP Anonymity: The WGP anonymity proof follows closely after the GP anonymity proof in Section 6.2. Essentially, the only difference between basic glitch protection e-tokens and window glitch protection e-tokens is that we now have $\hat{L}_1, \ldots, \hat{L}_W$ instead of just one $\hat{L}$ and $E_1, \ldots, E_W$ instead of just E. Therefore, step 4 in both showing and cloning will construct each $\hat{L}_i$ and E_i using the same techniques as for $\hat{L}$ and E, and the simulated proof in step 5 will reflected the more complicated construction. We give a sketch:

Once again, the simulator will store tables $Token$ and $Link$ (though entries in $Token$ will be longer because e-tokens contain more data). During EnvShow, the simulator will run the Show protocol in essentially the same manner. The simulator will perform steps 1, 2 and 3 as normal; at the end of which it will have random $R_1, \ldots, R_W, \rho_{1,1}, \ldots, \rho_{W,m}$. In step 4 the simulator will randomly choose $(S, \hat{L}_1, \ldots, \hat{L}_W, E_1, \ldots, E_W) \in \mathbb{G}^{1+2W}$. In step five, the simulator will fake the proof of knowledge of (z, s, J, σ) showing the e-token is formed correctly, using the technique as Section 6.1.3. Let $\vec{R} = (R_1, \ldots, R_W)$, $\vec{\rho} = (\rho_{1,1}, \ldots, \rho W, m)$, $\vec{\hat{L}} = (\hat{L}_1, \ldots, \hat{L}_W)$, and $\vec{F} = (F_1, \ldots, F_W)$. The simulator will store $Token(etoken) = (S, \vec{R}, \vec{\rho}, \vec{\hat{L}}, \vec{F})$. If the token is ever cloned with some link-ids $linkid_1, \ldots, linkid_W$, the simulator will retroactively set all the F_i so that $\hat{L}_i = Link(linkid_i) \cdot F_i^{R_i}$.

The simulator also clones e-tokens in the same way as the simulator for GP anonymity. It looks up the original e-token in $Token(etoken)$. In step 4, the simulator uses $linkid_i$ to calculate the value for F_i, and from that calculates the $\hat{L}_i$. Then it randomly chooses the E_i. In step 5 it once again simulates a zero-knowledge proof of knowledge showing the e-token is correctly formed. Finally, it stores the new e-token in $Token(etoken')$.

WGP Identification: Observe that IdentifyWGP is identical to IdentifyGP, once the appropriate link-id L is chosen. If there are $\ell = |\{S_1, \ldots, S_{m+1}\}|$ different cloned e-tokens, then there are $m + 1 + \ell$ distinct tags E^b_{i,j_i} and $m+1+\ell$ unknowns. Therefore, with high probability, the system of equations can be solved via Gaussian Elimination to reveal $pk_{\mathcal{U}}$. □

Bibliography

[1] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] Kostas G. Anagnostakis and Michael B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, 2004.

[3] N. Andrade, F. Brasileiro, W. Cirne, and M. Mowbray. Discouraging free riding in a peer-to-peer cpu-sharing grid. In *HPDC 2004*, 2004.

[4] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 6–17. ACM press, 1997.

[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, April 2000.

[6] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*, 2005.

[7] Edoardo Ballico, Giulia Boato, Claudio Fontanari, and Fabrizio Granelli. Hierarchical secret sharing in ad hoc networks through birkhoff interpolation. In *International Conference on Telecommunications and Networking (TeNE 2005)*, 2005.

[8] 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 *Lecture Notes in Computer Science*, pages 480–494. Springer Verlag, 1997.

[9] Amos Beimel, Tamir Tassa, and Enav Weinreb. Characterizing ideal weighted threshold secret sharing. In *Theory of Cryptography Conference – TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 600–619. Springer-Verlag, 2005.

[10] Mira Belenkiy, Melissa Chase, C. Chris Erway, John Jannotti, Alptekin Küpçü, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *Workshop on Privacy in the Electronic Society*, pages 31–40, 2007.

[11] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer-Verlag, 1992.

[12] George Robert Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference 1979*, volume 48 of *American Federation of Information Processing Societies Proceedings*, pages 313–317, 1979.

[13] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, 15(1):23–27, 1983.

[14] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity based encryption without random oracles. In *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 2004.

[15] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer-Verlag, 2004.

[16] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures using strong diffie hellman. In *CRYPTO 2005*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer-Verlag, 2005.

[17] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Verlag, 2001.

[18] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 431–444. Springer Verlag, 2000.

[19] Xavier Boyen. A promenade through the new cryptography of bilinear pairings. In *Proceedings of the IEEE Information Theory Workshop (ITW 2006)*, pages 19–23. IEEE Press, 2006.

[20] Stefan Brands. Untraceable off-line cash in wallets with observers. manuscript, CWI, 1993.

[21] Ernest F. Brickell. Some ideal secret sharing schemes. In *Journal of Combinatorial Mathematics and Combinatorial Computing*, volume 9, pages 105–113, 1989.

[22] 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 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 331–345. Springer Verlag, 2000.

[23] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysysanskaya, and Mira Meyerovich. How to win the clone wars: Efficient periodic n-times anonymous authentication. In *ACM CCS*, pages 201–210, 2006.

[24] Jan Camenisch, Susan Hohenberger, and Anna Lysysanskaya. Compact e-cash. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT*. lncs, 2005.

[25] Jan Camenisch, Susan Hohenberger, and Anna Lysysanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks – SCN*. lncs, 2006.

[26] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Giuseppe Persiano, editor, *Security in communication networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer Verlag, 2002.

[27] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology — CRYPTO 2004*, Lecture Notes in Computer Science. Springer Verlag, 2004.

[28] Jan Camenisch and Anna Lysyanskaya. A formal treatment of onion routing. In Victor Shoup, editor, *Advances in Cryptology — CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 169–187. Springer-Verlag, August 2005.

[29] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy*, May 2007.

[30] 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 *Lecture Notes in Computer Science*, pages 107–122. Springer Verlag, 1999.

[31] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology — CRYPTO 2003*, Lecture Notes in Computer Science. Springer Verlag, 2003. To appear.

[32] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer Verlag, 1997.

[33] Sebastien Canard and Aline Gouget. Divisible e-cash systems can be truly anonymous. In *Advances in Cryptology — EUROCRYPT 2007*, pages 482–497, 2007.

[34] 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 *Lecture Notes in Computer Science*, pages 561–575. Springer Verlag, 1998.

[35] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[36] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology — Proceedings of CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[37] David Chaum. Blind signature systems. In David Chaum, editor, *Advances in Cryptology — CRYPTO '83*, page 153. Plenum Press, 1984.

[38] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer Verlag, 1990.

[39] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 383–395, 1985.

[40] Bram Cohen. Incentives build robustness in bittorrent. In *Procceedings of the 1st Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA*, 2003.

[41] Bram Cohen. Incentives build robustness in bittorrent. In *Proc. 2th International Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, February 2003.

[42] B. Cox, J. D. Tygar, and M. Sirbu. Netbill security and transaction protocol. In *Proceedings of the First Usenix Workshop on Electronic Commerce*, pages 77–88, 1995.

[43] Ivan Damgård and Eiichiro Fujisaki. An integer commitment scheme based on groups with hidden order. In *ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 77–85. Springer Verlag, 2002.

[44] Ivan Bjerre Damgård, Kasper Dupont, and Michael Ostergaard Pedersen. Uncloneable group identification. In Serge Vaudenay, editor, *Advances in Cryptology — EUROCRYPT '06*, volume 4004 of *Lecture Notes in Computer Science*, pages 555–572. Springer-Verlag, 2006.

[45] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *IEEE Trans. on Information Theory*, IT-22(6):644–654, Nov. 1976.

[46] Roger Dingledine, Nick Mathewson, and Paul Syverson. Reputation in p2p anonymity systems. In *P2PECON '03: Proceedings of Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[47] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 1–17. Springer Verlag, 2002.

[48] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *Proceedings of the Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 416–431, 2005.

[49] David S. Dummit and Richard M. Foote. *Abstract Algebra*. John Wiley & Sons, Inc., third edition, 2004.

[50] Serge Fehr. Efficient construction of the dual span program. Available at `http://citeseer.ist.psu.edu/fehr99efficient.html`, 1999.

[51] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–893, 1997.

[52] Eric Friedman and Paul Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001.

[53] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In Burt Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30. Springer Verlag, 1997.

[54] Steven D. Galbraith. Supersingular curves in cryptography. In *Advances in Cryptology — ASIACRYPTO '01*, volume 2248 of *Lecture Notes in Computer Science*, pages 495–513. Springer Verlag, 2001.

[55] Hossein Ghodosi, Josef Pieprzyk, and Rei Safavi-Naini. Secret sharing in multilevel and compartmented groups. In C. Boyd and E. Dawson, editors, *ACISP '98: Proceedings of the Third Australasian Conference on Information Security and Privacy*, volume 1438 of *Lecture Notes in Computer Science*, pages 367–378. Springer Verlag, 1998.

[56] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[57] 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.

[58] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proc. 14th Annual ACM Symposium on Theory of Computing (STOC)*, pages 365–377, 1982.

[59] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 27th Annual Symposium on Foundations of Computer Science*, pages 291–304, 1985.

[60] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[61] Markus Jakobsson. Ripping coins for a fair exchange. In *Advances in Cryptology — EUROCRYPT '95*, volume 921, pages 220–230. Springer Verlag, 1995.

[62] Stanisław Jarecki and Vitaly Shmatikov. Handcuffing big brother: an abuse-resilient transaction escrow scheme. In *Advances in Cryptology — EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 590–608. Springer Verlag, 2004.

[63] Antoine Joux. A one-round protocol for tripartite Diffie-Hellman. In *Proceedings of the ANTS-IV conference*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer-Verlag, 2000.

[64] Seung Jun and Mustaque Ahamad. Incentives in bittorrent induce free riding. In *P2PECON '05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 116–121, 2005.

[65] Mauricio Karchmer and Avi Wigderson. On span programs. In *8th Annual Conference on Structures in Complexity Theory (SCTC'93)*, pages 102–111. IEEE Computer Society Press, 1993.

[66] Jon Kleinberg and Prabhakar Raghavan. Query incentive networks. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, 2005.

[67] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proceedings of the 2006 USENIX Operating Systems Design and Implementation (OSDI)*, November 2006.

[68] Nikitas Liogkas, Robert Nelson, Eddie Kohler, and Lixia Zhang. Exploiting bittorrent for fun (but not profit). In *Proc. 5th International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, February 2006.

[69] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free Riding in BitTorrent is Cheap. In *Proc. 5th Workshop on Hot Topics in Networking (HotNets-V)*, November 2006.

[70] Anna Lysyanskaya, Ron Rivest, Amit Sahai, and Stefan Wolf. Pseudonym systems. In Howard Heys and Carlisle Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[71] Sergio Marti and Hector Garcia-Molina. Identity crisis: Anonymity vs. reputation in p2p systems. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, 2003.

[72] Sergio Marti and Hector Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484, 2006.

[73] Noel McCullagh and Paulo S.L.M. Barreto. A new two-party identity-based authenticated key agreement. In *Topics in Cryptology – CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 262–274, 2005.

[74] Ralph Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer Verlag Berlin, 1988.

[75] Mira Meyerovich. Disjunctive multi-level secret sharing. manuscript, 2006.

[76] Silvio Micali. Certified e-mail with invisible post offices. Presentation at the 1997 RSA Security Conference, 1997.

[77] Shigeo Mitsunari, Ryuichi Sakai, and Masao Kasahara. A new traitor tracing. In *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, volume E85-A, No. 2, pages 481–484, 2002.

[78] Animesh Nandi, Tsuen-Wan "Johnny" Ngan, Atul Singh, Peter Druschel, and Dan S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference (Middleware 2005)*, Grenoble, France, November 2005.

[79] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.

[80] Standards (U.S.) National Bureau of. Data encryption standard (des). Fed. Inform. Proc. Standards Publication 46, Nat. Techn. Inform. Service, Springfield, VA, April 1977.

[81] Standards (U.S.) National Bureau of. Advanced encryption standard (aes). Federal Information Processing Standards Publication 197, 2001.

[82] Lan Nguyen and Rei Safavi-Naini. Dynamic k-times anonymous authentication. In *Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 318–333, New York, 2005.

[83] Pascal Paillier. Public-key cryptosystems based on composite residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–239. Springer Verlag, 1999.

[84] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.

[85] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do incentives build robustness in bittorrent? In *Proc. 4th USENIX/ACM Symposium on Networked Systems Design and Implementation*, April 2007.

[86] Michael Reiter, XiaoFeng Want, and Matthew Wright. Building reliable mix networks with fair exchange. In *Applied Cryptography and Network Security: Third International Conference*, pages 378–392. Lecture Notes in Computer Science, June 2005.

[87] Rosetta@Home. `http://boinc.bakerlab.org/rosetta/`.

[88] Brude Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204. Springer Verlag, 1993.

[89] Mike Scott. Authenticated id-based key exchange and remote log-in with simple token and pin number. `http://eprint.iacr.org/2002/164`, 2002.

[90] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[91] Victor Shoup. Fast construction of irreducible polynomials over finite fields. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1993.

[92] Joseph Silverman. *The arithmetic of elliptic curve*. Springer-Verlag, 1986.

[93] Gustavus J. Simmons. How to (really) share a secret. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 390–448. Springer Verlag, 1988.

[94] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM Conference*, August 2001.

[95] Tamir Tassa. Hierarchical threshold secret sharing. In Moni Naor, editor, *Theory of Cryptography: First Theory of Cryptography Conference, TCC 2004*, Lecture Notes in Computer Science, pages 473–490, Cambridge, MA, 2004.

[96] Tamir Tassa and Nira Dyn. Multipartite secret sharing by bivariate interpolation. In *ICAALP 2006, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 288–299, Venice, Italy, 2006.

[97] Isamu Teranishi, Jun Furukawa, and Kazue Sako. k-times anonymous authentication. In *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 308–322, Japan, 2004.

[98] Isamu Teranishi and Kazue Sako. k-times anonymous authentication with a constant proving cost. In *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 525–542, New York, NY, 2004.

[99] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. Karma: A secure economic framework for p2p resource sharing. In *Proc. of Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[100] Brian Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proc. 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, March 2002.

[101] Bo Zhu and Sushil Jajodia. Building trust in peer-to-peer systems: a review. *International Journal of Security and Networks*, 1(1/2):103–112, 2006.