

EFFICIENT ELECTRONIC CASH:
NEW NOTIONS AND TECHNIQUES

A Thesis Presented
by

Yiannis S. Tsiounis

to
The Department of Computer Science

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in the field of
Computer Science

Northeastern University
Boston, Massachusetts

June 1997

© Copyright 1997 by Yiannis S. Tsiounis
All Rights Reserved

Acknowledgements

This thesis would not have been possible without the guidance, technical and psychological support of my advisors, fellow researchers, friends and colleagues. I owe the warmest thanks to my academic advisors Agnes Chan, Yair Frankel, Boaz Patt-Shamir and Moti Yung, all of which contributed both technically and on a personal level towards this work, as well as to my external advisor Eytan Modiano. Agnes initiated my involvement with the field of cryptography and gave me a home at the University, providing constant support for my educational and professional requests—however strange they might be.

Moti has been as much a friend and mental guide as has been a teacher, and I cannot decide which of his seemingly endless knowledge and his unique wit has affected me most. Words cannot describe human relationships, so I hope that I can personally, rather than in writing, express my thanks to him. *Ευχαριστώ* Moti!

Boaz never failed to impress me both academically and on a personal level; he deserves special thanks not for having a (good) solution to every problem I have thought of asking, but for tolerating my way of collaboration and my erratic schedule.

I want to extend my thanks to Bulent Yener who gave me the right perspective and guidance during one of the most difficult times of my degree, not to mention seriously boosting my confidence at times where I needed it most. It is also my pleasure to thank George Davida for his comments on my thesis, and Ron Rivest for allowing me to present some of my results at MIT and for numerous helpful discussions.

Part of the work for this thesis was performed under a contract from GTE Laboratories Inc. in Waltham, MA. A summer internship in the Labs, together with the support and understanding of my supervisor Len D’Allotto and my manager Bob McKosky helped me overcome financial concerns and focus on my research work. I have but to appreciate their faith in me even after my graduation.

My parents have set the cornerstone of this work. They have guided me throughout my academic career, motivated and supported my graduate studies in the United States and tolerated my growing temper. Without their initiation I would not have started this degree. *Ευχαριστώ πατέρα και μάνα!*

I cannot thank enough Kiki, not for supporting me through the hurdles of the degree, neither for tolerating my bursts or mental absences during those years; but for giving me the right perspective and helping me become a human rather than only a scientist. I am sure she knows how much I thank her for that—and other things!

Lastly, it is with great joy that I give special thanks to my advisor Yair Frankel for his continuing support and patience, his significant technical contributions towards this thesis, but most of all for his invaluable guidance and advice throughout the duration of my studies. In all respects, this thesis is as much his achievement as it is mine, and it is dedicated to him.

Contents

1	Introduction	3
2	Cryptographic background	9
2.1	Preliminaries	10
2.1.1	Number theory and basic notations	10
2.1.2	One-way functions	13
2.1.3	Trapdoor functions	19
2.1.4	Hard core predicates	21
2.1.5	Indistinguishability	22
2.2	Basic protocols	23
2.2.1	Pseudorandom generators	24
2.2.2	Encryption schemes	26
2.2.3	Digital signatures and general computation protocols	32
2.2.4	Interactive proof systems and zero-knowledge proofs	37
2.2.5	Bit commitment	39
2.2.6	Hash functions and the random oracle model	41
2.2.7	Identification schemes	45
3	Survey of electronic cash	49
3.1	Basic model	51
3.1.1	Security	52
3.1.2	Efficiency	54
3.1.3	Protocols	55
3.2	Allowing exact payments	61

3.2.1	The binary tree approach	64
3.2.2	Protocols	65
3.3	Anonymity concerns (Fair electronic cash)	74
3.4	Further extensions	76
3.4.1	Tamper-resistant hardware (“wallets with observers”)	76
3.4.2	Transferability	78
3.5	Discussion and open problems	78
4	Unlinkable divisible e-cash	81
4.1	Introduction	82
4.2	Notation and simple results	85
4.3	Problem characterization	88
4.3.1	Tightness of lower bound	92
4.3.2	Determining k for a given set	92
4.4	An optimal solution	93
4.4.1	Tightness of upper bound	98
4.5	Generalization to a restricted set of denominations	99
4.6	Achieving unlinkable divisible e-cash	107
4.7	Other applications	108
5	Linkable divisible e-cash	109
5.1	Approaches	110
5.1.1	Using non-divisible coins	111
5.1.2	Achieving unbounded divisibility	116
5.2	Building blocks	117
5.2.1	Range-bounded commitment	117
5.2.2	Attacks and repair on Brands’ scheme	122
5.2.3	Attack and repairs on Okamoto’s scheme	124
5.3	The divisible e-cash scheme	128
5.3.1	The basic idea	129
5.3.2	Protocols	130
5.3.3	Security	135
5.3.4	Efficiency	144
5.4	Optimality based on divisibility precision	146

5.5	Universal framing protection	148
5.6	Open problems	148
6	Fair off-line e-cash	151
6.1	Introduction	152
6.2	Modeling FOLC	154
6.3	General results	156
6.4	The basic off-line electronic cash scheme	160
6.4.1	Security	162
6.5	Owner tracing	165
6.5.1	Trustees trust shop	165
6.5.2	Shops are not trusted	169
6.6	Coin tracing	176
6.7	Fair divisible e-cash	179
6.8	Open problems	180
7	Conclusion	183
	Bibliography	185

List of Figures

3.1	Model of electronic cash.	52
3.2	The withdrawal protocol of Brands' scheme.	60
3.3	The payment protocol of Brands' scheme	62
3.4	Binary tree representation for a \$1,000 coin.	65
3.5	Use of Γ, Ω and Λ values in the binary tree.	71
3.6	Model of "wallets with observers".	77
4.1	Glossary of notation.	86
4.2	The greedy algorithm for coin dispensing.	87
4.3	Algorithm for optimal solution of the k -payment problem.	94
4.4	Algorithm for optimal solution of the k -payment problem with allowed denominations \mathcal{D}	100
4.5	Dynamic programming algorithm for finding least number of coins for the remainder.	101
5.1	Schnorr proof of knowledge of representation of I w.r.t. g_1	124
5.2	The withdrawal protocol for divisible e-cash.	133
6.1	Model of Fair Off-Line e-Cash (FOLC).	155
6.2	Conditional untraceability is necessary in FOLC.	158
6.3	FOLC achieves key exchange.	158
6.4	The payment protocol.	162
6.5	Indirect discourse proof.	166
6.6	Indirect discourse proof for owner tracing when the shops are trusted.	167
6.7	Proof of equality of logarithms.	169
6.8	Owner tracing when shops are not trusted.	171

6.9 The withdrawal protocol supporting coin tracing. 177

List of Tables

3.1	Overview of off-line electronic cash schemes.	56
3.2	Overview of divisible off-line electronic cash schemes.	66
5.1	Efficiency of divisible off-line electronic cash schemes under some sample security parameters.	146
6.1	Efficiency of fair divisible off-line electronic cash under some sample security parameters.	180

Chapter 1

Introduction

This thesis addresses the topic of efficient electronic cash, a way to conduct anonymous electronic payments in an environment of mutual mistrust among the bank and the system's users.

In light of the explosive increase in electronic services, means for electronic payments become an essential asset. Potential applications include cellular phones, Internet services such as directories, digital “cash” cards and Global Positioning systems. As is also the case with physical payments, a multitude of electronic payment methods have been proposed. All payment systems, however, fall into two large categories: account-based (such as credit cards, telephone accounts, or bank accounts from which only checks are drawn) and token-based systems (such as physical cash, pre-paid phone cards, subway tokens or mail stamps). The distinguishing feature between these two approaches is the anonymity that a token-based system can provide to its users: a pre-paid phone card, for example, does not identify the caller, as physical cash do not identify their owner. In contrast, account based systems need, by design, to identify the system's users and their transactions.

Since anonymity of payments is usually connected with anonymity of physical cash, an anonymous token-based electronic payment system is traditionally referred to as an *electronic cash* (also known as *e-cash*, *digital cash*, *digital money*, *electronic money*) system. Electronic cash provide additional functionality from account-based systems, but can emulate the latter via straightforward additions; namely account-based systems can be substituted by electronic cash in which the tokens contain the user's identity. This is the case in both credit-based systems (e.g., credit cards), which

can be substituted by a credit-based token (e-cash) scheme, as well as debit-based systems.

Anonymity is progressively viewed as an attractive feature in economic transactions. Philosophically, the rights of individuals to privacy dictate it. On a more pragmatic side, massive compilation and misuse of personal user data is likely to lead to a general understanding of the danger to privacy originating from the lack of anonymity. We believe that this realization, as well as the increased functionality and security offered by electronic cash, identify e-cash as one of the most important electronic payment proposals.

Security is an indispensable property for any system that proposes to conduct payments, whereas *efficiency* is one of the main reasons for employment of electronic payment systems in general. Efficient and secure electronic cash have recently been achieved, constituting a major step towards truly practical anonymous payment systems. Current schemes, however, operate under a very restricted token-based model which does not allow for payments of exact amounts. Divisible electronic coins which allow for exact payments have been proposed before, but implementations have been inefficient and thus unsuitable for practical purposes.

The goal of this thesis is to propose new notions and techniques leading to electronic cash systems that can be applied in practice; it is our belief that we provide the critical step towards truly efficient systems, thus bridging the gap between research and real-life applications. We proceed in two basic directions towards this ideal, showing how to efficiently extend electronic cash to provide what we judge to be the two major missing links for practical implementations:

- 1. Provide capability for exact payments.**

Efficiency for complete payment systems requires provisions for exact payments; this has traditionally been achieved by providing systems in which coins can be divided. Divisibility, however, has been plagued with implementation difficulties; guaranteeing security while retaining efficiency has been thought of as an elusive goal. This thesis aims to show that providing exact payments is feasible at minimal efficiency costs, while security need not be sacrificed.

We investigate both conceivable approaches towards exact payment systems:

- (a) *keeping a multitude of coins*, as is the current practice in physical cash.

We analyze the exact complexity required for this approach; this is a combinatorial result with application to both physical and electronic cash. Its direct application on the latter emulates coins that can be divided in such a way that their portions are unlinkable by the bank. Namely, the portions of a coin are not only anonymous, but even a collaboration of the bank and shops cannot determine if these portions came from the same coin or the same user. We call such schemes *unlinkable divisible electronic cash*.

- (b) *achieving efficient (linkable) divisible electronic cash*. In such systems coins are unlinkable (i.e., the bank and shops cannot determine whether two anonymous coins belong to the same user) but portions of the same coin can be linked to the “parent” coin. In this case we show that it is feasible to construct an efficient system in which a single coin can be divided to smaller portions. Such systems have been traditionally called *divisible electronic cash*.

The above combinatorial result can also be used to emulate linkable divisible electronic cash. We first present an efficient instantiation using this approach and then describe a very efficient divisible electronic cash system and compare the two systems. We thereby show that both directions are viable, with each being optimal for certain system requirements.

2. Control the anonymity of users.

The complete anonymity offered by electronic cash is a cause of concern for governments and banks; a completely anonymous system could be used for criminal activities such as money laundering, purchase of illegal goods or committing perfect (anonymous) crimes. Thus, it has been proposed to allow a judge/escrow agent(s)/Trustee(s) to remove anonymity of users when necessary. Irrespective of any social or political arguments in favour or against this approach, current practices in all payment systems (such as physical cash, credit cards, checks, cellular phones) indicate that no wide-scale application of e-cash will be allowed to proceed without the ability of a governmental body to revoke user anonymity when necessary. In support of this argument, the need for controlling anonymity is highlighted in a recent NSA report by Law, Sabett and Solinas [88], as well

as in a survey of the legal issues in e-cash by Froomkin [60]. Anonymity controlled systems are traditionally called *fair*, following the terminology of [92], since they balance the anonymity requirement with the need of governments to trace fraudulent users.

Fair electronic cash systems have already been proposed, but require interaction with escrow agent(s); this of course is an undesirable property and prevents the systems from being used in practice. In this work we present an efficient method for tracing malicious users without involving the agent(s) in normal system operation. We furthermore show how to extend divisible electronic cash with this capability thus providing a complete anonymous payment system.

This thesis is organized as follows. Chapter 2 gives *a general cryptographic background*, necessary for the understanding of basic concepts used in later chapters; this can also be seen as an introduction to modern cryptography as it is largely unconnected with electronic cash. Chapter 3 formally *models electronic cash* and presents current schemes; this is intended as an overview of the area, focusing on efficient protocols that are to be built upon in later chapters.

Chapter 4 formalizes the problem of keeping a multitude of coins for exact payments, presents an analysis of the required complexity, and describes a provably optimal solution. This result is then used to emulate *unlinkable divisible electronic cash*. Chapter 5 then employs this result to emulate *linkable divisible electronic cash*, and continues with the description of a very efficient (linkable) divisible electronic cash system. The two systems are compared to show optimality under various system requirements.

Finally, Chapter 6 shows how to achieve efficient fair electronic cash, with the escrow agent(s) being off-line during system operation; we call this system *fair off-line electronic cash*. An integration of this approach to the divisible systems of the previous chapters is also described.

Most of the results in this thesis have appeared in preliminary form as technical reports or will appear as extended abstracts in various conferences and journals. The material of Chapter 4 is covered in “Exact analysis of exact change” [54] and is joint work with Yair Frankel and Boaz Patt-Shamir. Most of the work in Chapter 5 is included in “Mis-representation of identities in e-cash schemes and how to prevent it” [22], joint work with Agnes Chan, Yair Frankel and Phil MacKenzie, and in “How

to break and repair e-cash protocols based on the representation problem” [24] and “Range-bounded commitment and its application to efficient e-cash” [26], both collaborations with Agnes Chan and Yair Frankel. Finally, most of the results in Chapter 6 appear in “Indirect Discourse proofs: achieving Fair Off-Line e-Cash” [55] which is joint work with Yair Frankel and Moti Yung and in “Anonymity Control in E-Cash Systems” [43] which is joint work with George Davida, Yair Frankel and Moti Yung. In addition, the results of Chapter 6 and most of the results of Chapter 5 have been submitted for international patents.

Chapter 2

Cryptographic background

Our goal in this chapter is to encompass all notions and protocols necessary for the understanding of electronic cash; such self-sufficiency necessitates an overview of modern cryptography, due mainly to the complexity of electronic cash systems. Ever since Julius Caesar used a transposition cipher for “secure” message communication, cryptography has focused on encryption systems. The importance of which became apparent during War World II, when the deciphering of the Enigma machine gave the allies a significant tactical advantage. In recent years cryptography was mainly the concern of national governments with scientific research by private institutions being almost non-existent; the situation changed abruptly two decades ago, with the publication of Diffie and Hellman’s paper “New directions in cryptography” [44]. They proposed to base cryptography on computationally difficult problems, so that an adversary may theoretically be able to break a system, but this task is computationally infeasible. Modern cryptography is thus inherently linked to computational complexity and number theory, the latter being the main source of “intractable” problems.

Our presentation is divided into two main parts. The first section discusses number theory and some intractable problems, and how these are formalized for use in cryptography. We assume familiarity with basic algebraic concepts, such as groups and modular arithmetic, and basic complexity theory issues: deterministic, non-deterministic and probabilistic Turing Machines, time complexity, and the respective classes of problems computable in deterministic polynomial time P , non-deterministic polynomial time NP , and bounded probabilistic polynomial time BPP . We stress that in our discussion we refer solely to uniform Turing Machines; generalizations for

the non-uniform case are usually straightforward, but are outside the scope of this thesis.

The second section of the chapter elaborates on the implications of the computational approach to cryptography, by presenting protocols for a variety of problems, which would not have been thought solvable in the classical approach. We restrict ourselves to the most basic of protocols which serve as building blocks for more complex systems, such as electronic cash. Although most basic notions and protocols are formalized to guarantee clarity and unambiguity, not all concepts are presented with strict mathematical rigor. Our major source of formal definitions and notation are Goldreich's notes [63, 62], which provide an excellent framework for cryptographic discussions.

2.1 Preliminaries

In this section we give an overview of the cryptographic primitives used in upcoming protocols. We start with some basic number theory concepts which provide a framework for the description of “hard” computational problems. The notions of one-way and trapdoor functions are the first cryptographic primitives discussed; they are central in the understanding of modern cryptography, which is based on the difficulty of inverting efficiently computable functions. Hard-core predicates, a way to “concentrate” the difficulty of inverting a function, are then presented; these are the tools for applying one-way functions in cryptographic protocols. The section is concluded with the notion of indistinguishability of random variables, a necessary tool for the formalization of security in cryptography.

2.1.1 Number theory and basic notations

Modern cryptography is based on the existence of efficient algorithms available to the legitimate user, versus the intractability of retrieving information for an adversary. Throughout this thesis, a task will be considered *intractable* if it cannot be performed by a probabilistic polynomial-time Turing machine *p.p.t. TM*, otherwise we will say that it can be *efficiently computed*. By allowing the use of probabilistic machines, all properties of our algorithms are required to hold with a high probability. Conversely, we also allow an intractable problem to be solved by a p.p.t. TM, but only with a

very small probability.

To clarify the notion of “small” and “high” probabilities, we define the success probability of an algorithm to be *negligible* if it is smaller than the reciprocal of any polynomial on the input length (naturally, only positive polynomials are considered), and *non-negligible* if there exists a polynomial such that, for all sufficiently large inputs, the success probability is higher than its reciprocal. (Note that this is a strong negation of the notion of negligible probability; namely functions/probabilities may be neither negligible nor non-negligible.) The motivation behind this definition is that if an algorithm succeeds with negligible probability, then by repeating the algorithm polynomially many times the probability of success remains negligible, hence this algorithm cannot be used to efficiently solve the original problem. In generalizing this notion, we call a quantity “negligible” on some parameters, if it is smaller than the reciprocal of any polynomial of these parameters. In regards to probabilities, we also define the probability of an event to be *overwhelming* on some parameters, if its complement is negligible on these parameters. The relevant parameters are not always stated explicitly, as they are usually clear from the context.

We now proceed to describe some elementary number theory concepts. For more detailed discussions of these we refer to [84, Secs. 4.5.2–4.5.4], [77, Chs. 12, 13], or to introductory number theory books, such as [98, Chs. 2, 3], [76, Ch. 3], [73, Chs. 2, 3].

Throughout this thesis, we denote $|\alpha|$ to be the size of an integer in bits, unless otherwise stated; i.e., $|\alpha| = \log_2 A$, where A is the absolute value of α .

Euler Totient function

Let N be a positive integer. We represent *the multiplicative group modulo N* to be the set of positive integers that are relatively prime to N and are between 1 and N :

$$Z_N^* = \{x | 1 \leq x < N, \gcd(x, N) = 1\} .$$

Using the Euclidean algorithm (the oldest non-trivial algorithm—with records dating to 300 B.C.—still in use today) it can be determined in time polynomial in $|N|$ and $|x|$ whether or not $\gcd(x, N) = 1$, i.e., whether $x \in Z_N^*$. The extended Euclidean algorithm allows computation of $\gcd(x, N)$, again in polynomial time in x and N . The cardinality of Z_N^* is denoted by the *Euler Totient function* $\phi(n)$ and can be

directly derived given the prime factorization of N . Let

$$N = \prod_{i=1}^n p_i^{\epsilon_i}, \text{ with } p_i \text{ distinct primes,} \quad (2.1)$$

then

$$\phi(N) = \prod_{i=1}^n (p_i^{\epsilon_i} - p_i^{\epsilon_i-1}) \quad (2.2)$$

In particular, this means that if N is prime, $\phi(N) = N - 1$, as expected.

We say that x is a *quadratic residue modulo N* , if there exists an integer $w \in Z_N^*$ such that $x \equiv w^2 \pmod{N}$, otherwise it is a *quadratic non-residue mod N* . We denote by QR_N and QNR_N the set of quadratic residues and non-residues, respectively, in the multiplicative group Z_N^* . We also define the quadratic residuosity predicate to be

$$Q_N(x) = \begin{cases} 0 & \text{if } x \text{ is a quadratic residue mod } N, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

If N is prime then $Q_N(x)$ can be computed in time polynomial in $|N|$. In addition, a number x is a quadratic residue mod N if and only if it is a quadratic residue modulo each prime factor of N . Therefore, given the prime factorization of N , as in eq. (2.1), $Q_N(x)$ can be computed in time polynomial in $|N|$, for any $x \in Z_N^*$. If $N \neq 2$ is prime then any number $x \in QR_N$ has two distinct square roots, $w^2 \equiv x \pmod{N}$ and $(-w)^2 \equiv x \pmod{N}$. From this it can be seen that in the general case, for all x such that $Q_N(x) = 0$, the number of solutions $w \in Z_N^*$ to $w^2 \equiv x \pmod{N}$ is exactly 2^n , where n is the number of *distinct* prime factors of N . Lastly, it is also clear that if $Q_N(x) = Q_N(y) = 0$ then $Q_N(xy) = 0$, and if $Q_N(x) \neq Q_N(y)$ then $Q_N(xy) = 1$.

For a prime P , the *Legendre symbol* of $x \pmod{P}$ is defined to be

$$(x/P) = \begin{cases} 1 & \text{if } x \text{ is a quadratic residue mod } P, \text{ and} \\ -1 & \text{otherwise.} \end{cases}$$

Given the prime factorization of N from eq. (2.1), the *Jacobi symbol* of $x \pmod{N}$ is defined as

$$(x/N) = \prod_{i=1}^n (x/p_i)^{\epsilon_i},$$

where (x/p_i) is the Legendre symbol of $x \pmod{p_i}$. It can be shown that the Jacobi symbol (x/N) can be computed in time polynomial in $|N|$. From the definition of the Jacobi symbol it can be derived that

$$(xy/N) = (x/N) \cdot (y/N) \quad (2.3)$$

The Jacobi symbol provides some information about the quadratic residuosity of $x \bmod N$. Namely, if $(x/N) = -1$ then x is in QNR_N , i.e., $Q_N(x) = 1$. However, if $(x/N) = 1$, no efficient algorithm is known for computing $Q_N(x)$ non-negligibly better than random guessing, i.e., with probability significantly better than $\frac{1}{2}$.

From the above it immediately follows that in the case where $N = P \cdot Q$, with P, Q primes, Z_N^* consists of 4 classes, $Z_{(i,j)} = \{x \in Z_N^* \mid (x/P) = i, (x/Q) = j\}$, with $i, j \in \{1, -1\}$.

An integer N is a *Blum integer* if it is the product of two primes, each congruent to 3 mod 4. It can be shown that when $N = P \cdot Q$ is a Blum integer, each element in QR_N has a unique square root which is also in QR_N . In particular, every element $x \in QR_N$ has 4 square roots y_1, \dots, y_4 for which $y_1 \in Z_{(1,1)}, y_2 \in Z_{(1,-1)}, y_3 \in Z_{(-1,1)}, y_4 \in Z_{(-1,-1)}$. In addition, $y_1 \equiv -y_4 \pmod N$ and $y_2 \equiv -y_3 \pmod N$. Clearly, these properties hold for any 2^t -th root of x , for any integer t .

A special case of Blum integers are the *Williams integers*, in which $P \equiv 3 \pmod 8$ and $Q \equiv 7 \pmod 8$. In this case, for any $x \in Z_N^*$, *exactly one* of $x, -x, 2x, -2x$ is in QR_N . This can be easily shown from the fact that $(-1/P) = (-1/Q) = (2/P) = -1, (2/Q) = 1$ and from property (2.3).

Quadratic Residuosity Problem (QRP)

The Quadratic Residuosity Problem (QRP), stated as finding $Q_N(x)$ for some $x \in Z_N^*$ where N is a Blum integer, is assumed to be an intractable problem [69]. This is stated as the Quadratic Residuosity Assumption.

Quadratic Residuosity Assumption (QRA)

The Quadratic Residuosity Assumption (QRA) states that no efficient algorithm can solve the QRP, with probability non-negligibly better than random guessing.

2.1.2 One-way functions

Consider an encryption scheme: the sender computes the encryption function on some input and sends it to the recipient, who recovers the encrypted message. A necessary property for the encryption function is that it cannot be inverted by an adversary. We formalize this property by requiring that the function is *one-way*, i.e., it is easy to compute but hard to invert. In the uniform computational model there are two flavours of one-way functions; strong one-way, or simply one-way functions, and weak one-way. Intuitively, strong one-way functions are easy to compute but hard to invert,

except with negligible probability; whereas for weak one-way functions there exists a polynomial inverting algorithm which is only required to fail with non-negligible probability. Interestingly, the existence of weak one-way functions implies that strong one-way functions can be constructed, although there do exist functions that are weak one-way but not strong one-way. Here we define and use the most common flavour of strong one-way functions; the formal definition and the above results about weak one-way functions can be found in Goldreich’s book [62].

Definition 2.1.1 ((strong) one-way functions) *A function $f: \{0, 1\}^* \mapsto \{0, 1\}^*$ is called (strongly) one-way if it satisfies the following two conditions*

- (1) *Easy to compute: There exists a deterministic polynomial-time algorithm A which, on input x , outputs $f(x)$, i.e., $A(x) = f(x)$.*
- (2) *Hard to invert: For every probabilistic polynomial-time algorithm A' , every positive polynomial $p(\cdot)$, and all sufficiently large n ’s*

$$\text{Prob}\left(A'(f(U_n), 1^n) \in f^{-1}(f(U_n))\right) < \frac{1}{p(n)},$$

where U_n denotes a random variable uniformly distributed over $\{0, 1\}^n$.

Informally, the definition allows only a negligible probability of success for the inverting adversarial algorithm A' . Note that the length of the desired output is supplied as input to the inverting algorithm A' in unary form 1^n . This is used to rule out the possibility that a function is considered one-way, simply because the inverting algorithm does not have enough time to print the output: an example would be a function which compresses a number represented in unary into a binary string; its inverse, although trivial to compute, takes exponential time to print.

A stronger notion than one-way functions is that of *one-way permutations*, which are one-way functions that are also bijections. Such functions are useful for constructing more complex notions, such as trapdoor permutations, as will be seen in Section 2.1.3.

Although the above definition is intuitive, it is nevertheless unsuitable for describing candidate one-way functions. In practice we need two properties from a cryptographic function. First, it must be “secure”. But in modern cryptography security is based on the intractability of some problems—except for unconditionally secure cryptosystems, e.g., encryption based on a random pad, which require key

lengths too large for practical implementations. Therefore the *size* of the input is a crucial factor in determining the security of a function or system, since even NP-hard problems are easily solvable for small inputs. Hence it is more convenient from a technical perspective to define an infinite collection of functions, each operating on a finite domain, instead of a single function over an infinite domain. The second property is that we should be able to define several functions for a specific input size: if a company buys an encryption function from a vendor, then a rival company must be given a different “version” of that function, even if they request the same security parameter, i.e., input size. Thus, for every given size requirement, or *security parameter*, we must be able to efficiently choose from a variety of “equally good” functions. This choice is made at random to protect guessing of the selected function.

We now present the formulation for **collections of functions** which captures the aforementioned properties and is used for describing further notions, such as trapdoor functions, trapdoor permutations and hash functions, as well as candidates for one-way functions. As mentioned above, each member of the collection operates on a finite domain, while all functions share a single evaluating algorithm which, given the description of the specified function and an element in its domain, returns the value of the function at the given point. Formally:

Definition 2.1.2 (collection of functions) *A collection of functions consists of an infinite set of indices, denoted \bar{I} , a finite set D_i for each $i \in \bar{I}$, and a function f_i defined over D_i .*

In order for a collection of functions to be applicable, there must exist an efficient function-evaluating algorithm F which, on input $i \in \bar{I}$ and an element $x \in D_i$ returns $f_i(x) \in \{0, 1\}^*$. In addition, however, there must exist efficient ways to randomly select an index, specifying a function over a sufficiently large domain, and to randomly select an element of the domain, given the domain’s index. Informally, the last two requirements guarantee that there is an efficient way to describe a function and its domain, as well as elements within that domain. The sampling property of the index set is captured by an efficient algorithm I that, on input an integer n in unary, randomly selects a $p(n)$ -bit long index, where $p(\cdot)$ is some polynomial, specifying the function and its domain; the sampling property of the domains is captured by an efficient algorithm D which, on input an index i randomly selects an element in D_i . Specifically for one-way functions, we also require that every efficient algorithm, on

input an index i and an element in the range of f_i , fails to invert the function, except with negligible probability. Formally:

Definition 2.1.3 (collection of one-way functions) *A collection of functions $\{f_i: D_i \mapsto \{0, 1\}^*\}_{i \in \bar{I}}$, is called (strongly) one-way if there exist three probabilistic polynomial-time algorithms, I, D and F , such that the following conditions hold*

(1) *easy to sample and compute: The output distribution of algorithm I , on input 1^n , is a random variable over the set $\bar{I} \cap \{0, 1\}^n$. The output distribution of algorithm D , on input $i \in \bar{I}$, is a random variable over D_i . For $i \in \bar{I}$ and $x \in D_i$, $F(i, x) = f_i(x)$.*

(2) *hard to invert: For every probabilistic polynomial-time algorithm A' , every polynomial $p(\cdot)$, and all sufficiently large n*

$$\text{Prob} \left[A'(f_{I_n}(X_n), I_n) \in f_{I_n}^{-1}(f_{I_n}(X_n)) \right] < \frac{1}{p(n)} ,$$

where I_n is a random variable describing the output distribution of algorithm I on input 1^n , X_n is a random variable describing the output of algorithm D on input I_n , and the probability is taken over the distributions induced by the sampling algorithms I and D .

For practical applications we relax the definition, allowing I to output indices of length $p(n)$, for some polynomial $p(\cdot)$, and all algorithms, including I , to fail with negligible probability.

In light of this definition, we say that a *triplet of probabilistic polynomial-time algorithms (I, D, F) constitutes a collection of one-way functions*, if there exists a collection of functions for which these algorithms satisfy the above two conditions. Goldreich [62] has shown that the above two definitions of one-way functions are equivalent.

Candidate one-way functions

The existence of one-way functions implies that $P \neq NP$; however the converse is not true. It is thus not surprising that no function is known to be one-way. Candidate functions that are widely believed to be one-way are the discrete exponentiation and multiplication functions, i.e., the inverses of the discrete logarithm and the factoring function. These functions are used as the basis for a large portion of cryptographic

protocols, including most electronic cash schemes. These functions are described in this subsection, together with the representation problem, the Rabin function, the factoring permutations, and RSA, which are also used extensively in cryptography.

The Discrete Logarithm Assumption (DLA)

The Discrete Logarithm Problem (DLP) is defined by a triplet of algorithms, $(I_{DLP}, D_{DLP}, F_{DLP})$. On input 1^n , the index selecting algorithm I_{DLP} selects uniformly a prime P with $2^{n-1} \leq P < 2^n$, and a primitive element g in the multiplicative group modulo P , i.e., a generator of this cyclic group. The domain selecting algorithm D_{DLP} , on input (P, g) , selects uniformly $x \in Z_{P-1}$, a residue modulo $P - 1$. Algorithm F_{DLP} , on input $((P, g), x)$, halts outputting

$$y \equiv DLP_{P,g}(x) \stackrel{\text{def}}{=} g^x \pmod{P}.$$

Inverting $DLP_{P,g}(x)$ amounts to extracting the discrete logarithm of x , base g , modulo P . *The Discrete Logarithm Assumption (DLA)* states that no p.p.t. TM can, on input (P, g, y) , output x , non-negligibly better than random guessing. Surveys of the best known algorithms for breaking the DLA are given in [36, 99, 37].

Alternatively, for the DLP one can use a prime P , where $P = aQ + 1$, for an integer $a > 1$ and randomly chosen prime Q , and choosing g to be a generator of the unique subgroup G_Q of the multiplicative group Z_P^* . For $a = 2$ the primes generated are believed to make DLA stronger. If a is larger than 2 the computational burden of algorithm F_{DLP} is reduced, while the difficulty of the problem is not [107]. The efficiency gain results from the fact that current implementations of modular exponentiation algorithms run in time $O(n^2 \cdot m)$, where n is the length of the modulus (in this case P) and m the length of the exponent; when g is a generator of G_Q , instead of Z_P^* , the length of the exponent is at most $|Q| \approx |P| - |a|$, instead of being at most $|P|$. Current implementations usually require that the modulo P is at least 500 bits long and Q is at least 160 bits long.

The representation problem in groups of prime order

A problem which is equivalent in difficulty to the DLP is *the representation problem in groups of prime order*, introduced by Brands [15], which can be seen as a generalization of the DLP. Informally, an instance of the problem consists of a prime P , a uniformly chosen tuple of generators (g_1, g_2, \dots, g_m) of Z_P^* and a uniformly chosen $x \in Z_P^*$. The problem is to find a representation of x with respect to (g_1, g_2, \dots, g_m) ,

i.e., find a tuple (a_1, a_2, \dots, a_m) with $a_i \in Z_{P-1}$, such that $x = \prod_{i=1}^m g_i^{a_i}$. It is known that this problem is as difficult as breaking the DLA on group Z_P^* [15].

The factoring problem

Let x, y be integer numbers and $N = x \cdot y$. An algorithm F solves *the factoring problem* if, on input N , outputs (x, y) . The best algorithms known [84, Sec. 4.5.4], [108, 109, 110], [77, Chs. 12, 13] for factoring an integer N run in time $L(P) = 2^{O(\sqrt{\log P \log \log P})}$, where P is the second biggest prime factor of N . Hence, if the factors (x, y) are randomly chosen large primes the factoring problem is assumed to be intractable. In practice it is only necessary to randomly choose x, y as large numbers, since there is a very high probability that these numbers will contain sufficiently large prime factors.

Protocols that are based on the difficulty of factoring usually require that x and y are at least 256 bit primes, and preferably larger than 512 bits [75], since factoring 430 bit numbers is well within the reach of today's computing machinery [3, 47].

The Rabin function

Rabin [111] showed that finding square roots modulo an integer is equivalent in difficulty to factoring the integer. Hence its inverse, squaring modulo an integer, called *the Rabin function* [112, 111], is one-way if and only if factoring is intractable. The Rabin collection of functions is defined by a triplet of algorithms (I_R, D_R, F_R) . On input 1^n , I_R uniformly selects two primes P, Q of equal length n , and outputs $N = P \cdot Q$. Algorithm D_R , on input N , selects uniformly an element $x \in Z_N^*$, and algorithm F_R , on input (N, x) , outputs

$$\text{Rabin}_N(x) \stackrel{\text{def}}{=} x^2 \bmod N$$

The Rabin function is a 4-to-1 mapping on the multiplicative group modulo N . Note that D_R selects an element in Z_N^* , which contains $\phi(N) = (P-1) \cdot (Q-1) \approx N - 2\sqrt{N} \approx N$ elements. This algorithm can thus be substituted by one that uniformly selects an element in $\{1, \dots, N\}$ and discards all elements that are not relatively prime to N . Primality testing is currently possible with efficient probabilistic algorithms.

If N is a Blum integer then, as we have seen in Section 2.1.1, each element in QR_N has a unique square root which is also in QR_N , hence in this case the Rabin function induces a permutation over QR_N , the quadratic residues modulo N . The formal definition of this collection of **factoring permutations** can be inferred from

the definition of the Rabin function, and is thus omitted.

RSA

A generalization of the Rabin function is *RSA* [115]. The main difference is that exponentiation in RSA uses an integer $e \neq 2$ that is relatively prime to $\phi(N)$. Formally, the collection is defined by a triplet of algorithms $(I_{RSA}, D_{RSA}, F_{RSA})$. On input 1^n , I_{RSA} uniformly selects primes P, Q of the same length $|P| = |Q| = n$ and an integer $e \in Z_N^*$ that is relatively prime to $\phi(P \cdot Q) = (P - 1)(Q - 1)$. It outputs (N, e) , where $N \stackrel{\text{def}}{=} P \cdot Q$. On input (N, e) , D_{RSA} uniformly selects an element $x \in Z_N^*$, and algorithm F_{RSA} , on input $((N, e), x)$, outputs

$$\text{RSA}_{N,e}(x) \stackrel{\text{def}}{=} x^e \bmod N .$$

In contrast to the Rabin function, RSA is a permutation on its domain [115]; this can be shown based on the fact that e is relatively prime to $\phi(N)$. Although proving that RSA is as difficult as factoring is a well-known open problem, all current algorithms attacking RSA do so by, explicitly or implicitly, trying to factor N [84, Sec. 4.5.4], [62]. Hence, current implementations use the same parameters as those used for factoring, with suggestions for 256 to 1,024-bit primes being the most common. Note however that, as shown by [71], using low exponents e , such as 2 or 3, is insecure. The “handbook of theoretical Computer Science” [77, Ch. 13] provides a helpful discussion on the security of RSA.

RSA can be slightly modified to have the trapdoor property, as described in the following section, which makes it useful for a variety of cryptographic protocols.

2.1.3 Trapdoor functions

A family of one-way functions is said to be a trapdoor collection of functions, if there exists some secret information, called the “trapdoor information” or “trapdoor key”, for each function, with which the inverse can be computed with high probability in probabilistic polynomial time. The collection is one of trapdoor permutations, if the functions are bijections, i.e., they are based on one-way permutations. A trapdoor permutation can be the basis for public-key encryption and digital signature schemes: the permutation is made public, while the trapdoor key is kept secret. Anyone can encrypt by applying the permutation to a message, but only the holder of the private key can invert the permutation to recover the message. For a signature, the holder

of the private key applies the inverse permutation on a message; anyone can verify the signature by applying the permutation on it, while it is guaranteed that only the holder of the private key could have signed the message.

The formal definition of trapdoor functions/permutations follows; here, $t(i)$ denotes the trapdoor information, which is different for each function f_i . While $t(i)$ cannot be efficiently computed by i , the index generating algorithm, I , can efficiently generate corresponding pairs $(i, t(i))$.

Definition 2.1.4 (collection of (strong) trapdoor functions (permutations))

Let I be a probabilistic algorithm, and let $I_1(1^n)$ (resp. $I_2(1^n)$) denote the first (resp. second) half of the output of $I(1^n)$. A triplet of algorithms, (I, D, F) , is called a collection of (strong) trapdoor functions (resp. permutations) if the following two conditions hold

- (1) The algorithms induce a collection of one way functions (resp. permutations):
The triplet (I_1, D, F) constitutes a collection of one-way functions (resp. permutations).
- (2) Easy to invert with trapdoor: There exists a (deterministic) polynomial-time algorithm, denoted F^{-1} , such that for every $(i, t(i))$ in the range of I and for every $x \in D_i$, it holds that $F^{-1}(t(i), F(i, x)) = x$.

The definition can be relaxed to allow the conditions to be satisfied with overwhelming probability, by allowing I to output, with negligible probability, pairs $(i, t(i))$ for which either f_i is not a permutation or $F^{-1}(t(i), F(i, x)) \neq x$ for some $x \in D_i$.

Candidate trapdoor functions and permutations

RSA

A candidate one-way trapdoor permutation is a slight modification of the *RSA collection* of functions, presented in Section 2.1.2. The index generating algorithm I_{RSA} is modified to produce the additional output (N, d) , where d is the multiplicative inverse of e modulo $\phi(N) = (P-1) \cdot (Q-1)$: this inverse exists since e is relatively prime to $\phi(N)$. The string d serves as the trapdoor information: the inverting algorithm F_{RSA}^{-1} is identical to F_{RSA} , but takes $((N, d), y)$ as input

$$F_{RSA}^{-1} [(N, d), F_{RSA}((N, e), x)] \stackrel{\text{def}}{=} (x^e)^d \bmod N \equiv x \bmod N .$$

The Rabin function

The Rabin function can also be modified to become a trapdoor function. In this case I_R also outputs (P, Q) , the factorization of N , as the trapdoor information. The 4 square roots of a number can then be computed by extracting a square root modulo each of the primes, and combining the result using the Chinese Remainder Theorem [76, 73, 98].

In addition, if N is a Blum integer **the collection of factoring permutations**, similarly modified, forms a collection of trapdoor permutations.

2.1.4 Hard core predicates

If f is a one-way function, then given y it is infeasible to find a preimage $x \in f^{-1}(y)$. However, this does not mean that it is infeasible to extract some partial information about x , such as its parity, a few bits in its binary representation or other potentially important information. For example in RSA, which is considered to be one-way, the Jacobi symbol of y is equal to that of x , hence some partial information about x is certainly leaked. This limits the applicability of one-way functions in tasks where all partial information about a preimage must be hidden, for example in secure encryption (see Section 2.2.2 below). Fortunately, assuming the existence of one-way functions, it is possible to construct one-way functions which hide specific partial information about the preimage, which is easy to compute from the preimage itself [68, 69]. This partial information can be considered as a “hard core” on the difficulty of inverting f . Formally:

Definition 2.1.5 (hard-core predicate) *A polynomial-time computable predicate $b: \{0, 1\}^* \mapsto \{0, 1\}$ is called a hard-core of a function f if for every probabilistic polynomial-time algorithm A' , every polynomial $p(\cdot)$ and all sufficiently large n*

$$\text{Prob}[A'(f(U_n)) = b(U_n)] < \frac{1}{2} + \frac{1}{p(n)} ,$$

where U_n is a random variable uniformly distributed over $\{0, 1\}^n$.

It is clear that a hard-core predicate must be unbiased, or else it would be easy to guess with probability better than half, regardless of the function f [62].

Observe that a predicate can be hard-core for a function f simply because f loses some information about its input; for example, consider a function which simply drops

the first bit of the input, and a predicate which requests this first bit. However, if f does not lose any information, i.e., it is one-to-one, then hard-cores for f exist only if f is one-way [62]. In fact, every (strong) one-way function f has a hard-core predicate [65], also called “hard bit” for f :

Theorem 2.1.1 *Let f be a (strong) one-way function, and let $g(x, r) \stackrel{\text{def}}{=} (f(x), r)$, where $|x| = |r|$. Let $b(x, r)$ denote the inner-product mod 2 of the binary vectors x and r . Then the predicate b is a hard-core of the function g .*

We remark that simple hard-core predicates are known for the RSA and Rabin [1] as well as the DLP [13] collections presented in Section 2.1.2. Specifically, the least significant bit is a hard-core for the RSA collection, while assuming that the DLP collection is one-way, it is infeasible to guess whether $x < \frac{P}{2}$, i.e., the most significant bit of x , when given $DLP_{P,g}(x) = g^x \bmod P$.

2.1.5 Indistinguishability

In both scientific and real-life applications, it is essential to know when two objects are considered to be equivalent. A natural way of defining equivalence, is to look at indistinguishability: objects are considered to be equivalent if they are indistinguishable from each other. Intuitively, indistinguishability can be defined in a number of ways: perfect (objects are indeed the same), statistical (objects differ in very few—sub-polynomially many—points), and computational (objects may be different, but no efficient algorithm can tell them apart). Since we can only employ efficient algorithms in practice, it seems that computational indistinguishability is the most useful of these notions; objects that are computationally indistinguishable can, for all practical purposes, be considered equivalent. For applications in cryptography we are mainly interested in distinguishing between probability distributions, a notion introduced by Yao [131] and Goldwasser and Micali [69]. The formalization is restricted to computational indistinguishability and is achieved by looking at infinite sequences of distributions, rather than working with fixed distributions. Such sequences are called probability ensembles.

Definition 2.1.6 (ensembles) *Let I be a countable index set. An ensemble indexed by I is a sequence $X = \{X_i\}_{i \in I}$, where each X_i is a random variable over $\{0, 1\}^i$.*

We are now ready to define indistinguishability. For simplicity, we use the set N of natural numbers as the index set.

Definition 2.1.7 (polynomial-time indistinguishability) *Two ensembles, $X \stackrel{def}{=} \{X_n\}_{n \in N}$ and $Y \stackrel{def}{=} \{Y_n\}_{n \in N}$, are indistinguishable in polynomial time if for every probabilistic polynomial-time algorithm A , every polynomial $p(\cdot)$, and all sufficiently large n*

$$|\text{Prob}(A(X_n, 1^n) = 1) - \text{Prob}(A(Y_n, 1^n) = 1)| < \frac{1}{p(n)},$$

where the probabilities are taken over the corresponding random variables X_i or Y_i and the internal coin tosses of A .

In *typical cases*, where the length of X_n (resp. Y_n) and n are polynomially related, i.e., $|X_n| < \text{poly}(n)$ and $n < \text{poly}(|X_n|)$, giving 1^n as an input to D is not necessary. We nevertheless include it for clarity and concreteness.

2.2 Basic protocols

In this section we describe some of the most fundamental cryptographic protocols. These are important in isolation, but in addition they are useful as building blocks for more complex systems. We concentrate on protocols applicable in electronic cash, but the complexity of the latter results in the inclusion of most of the basic cryptographic concepts. Implementation examples for most concepts are presented, mostly for the sake of clearer understanding; more focus is given in encryption and signature schemes where numerous systems are described, as they are to be used in subsequent chapters.

We begin with a discussion on pseudorandom generators, which underly all cryptographic systems, since “random bits” are necessary in all cryptographic systems. We continue with encryption schemes, since they are to be used in transferring information among electronic cash participants. Digital signatures and blind digital signatures are presented next; these are necessary for non-repudiation, a requirement for a variety of cryptographic systems, electronic cash in particular. Interactive proofs, in general, and zero-knowledge proofs in particular, are discussed next; the ability to prove a statement, especially without revealing any other information in the process, is a very powerful and necessary tool for a variety of applications. In our context,

zero-knowledge proofs and blind digital signatures are the tools for providing user anonymity in payment transactions.

Zero-knowledge proofs use bit commitment schemes as building blocks. Bit commitment, a way to digitally simulate a sealable opaque envelope, is therefore described next. It is followed by hash functions, which provide a way to compress a message while to all efficient algorithms the compression looks as a one-to-one function. Hash functions are an essential sub-protocol for most cryptographic schemes, in particular digital signatures, pseudorandom generators and message authentication schemes, all of which are needed in e-cash. We continue with a concept bearing some similarity to hash functions, namely random oracles, which can be seen as hash functions whose output looks random. Random oracles allow the construction of very efficient digital signatures, pseudorandom generators and authentication schemes and are thus crucial in developing efficient e-cash. We conclude with a discussion on identification schemes, which can be seen as specialized zero-knowledge proofs and which are directly applicable in e-cash as well as a variety of cryptographic systems. Efficient identification schemes and a general way to convert them into signature schemes conclude the section.

2.2.1 Pseudorandom generators

Consider a cryptographic system that is used by two rival companies. The algorithms used by the system are completely known/available to both companies, hence there clearly must be some information that is unique and secret to each application: there must be a way to “break the symmetry” of information available to the users of the system, or to adversaries who manage to obtain a description of the algorithms. Given that users must also be prevented from *guessing* that secret information, it is essential that the symmetry be broken using random bits. Therefore *the notion of randomness is central in cryptography*; “good quality” random bits, i.e., ones that cannot easily be guessed, are necessary for every protocol.

However, defining randomness, and furthermore classifying a bit sequence as being random, has been an elusive goal. Instead, we will define a string to be *pseudorandom* if no efficient algorithm can distinguish it from a uniformly chosen string of the same length. A pseudorandom string that satisfies this definition can then be used in place of a “truly random” string: since the cryptographic protocols utilize only efficient

algorithms, if the behaviour of an algorithm was detectably altered by the substitution of a pseudorandom string in place of a random one, then the same algorithm could be used to efficiently distinguish this pseudorandom string from a uniformly chosen one; a contradiction to the definition of pseudorandom strings.

To formalize our discussion, we begin with the notion of *pseudorandom ensembles*. These are ensembles which are computationally indistinguishable from the *standard uniform ensemble*, $\{U_n\}_{n \in \mathbb{N}}$, where each U_n is in turn defined to be a random variable uniformly distributed over the set of strings of length n . The concept of a pseudorandom generator (*PRG*) should now be clear: it is an efficient algorithm that generates pseudorandom sequences, or, in terms of ensembles, the ensemble of its output sequences is a pseudorandom ensemble, thus providing “good quality” random-looking bits for cryptographic protocols. We note that this definition poses very strict requirements on PRGs, in the sense that the generated strings must pass *all* efficient tests for randomness, not just some statistical, or of any other form, tests. M. Blum and Micali [13] introduced the first method for designing a provably secure PRG based on hard-core predicates.

We can visualize a pseudorandom generator as either a probabilistic algorithm or, equivalently, as a deterministic algorithm that is given a random string as an input. The latter formalization is clearer, and clarifies another aspect of PRGs: the expansion property. Clearly, if our deterministic algorithm outputs a sequence equal to length as its random input, then it is nothing more than a trivial algorithm. Hence, a PRG is also required to expand its input. This requirement also reflects a desirable aspect in cryptographic protocols: we would like to *generate pseudorandom ensembles using as little randomness as possible*. This requirement roots in the cost, in terms of time and space complexity but also in terms of hardware, of generating uniformly chosen strings. Formally, we define:

Definition 2.2.1 (pseudorandom generator) *A pseudorandom generator is a deterministic polynomial-time algorithm, G , satisfying the following two conditions:*

(1) *Expansion: there exists a function $l: \mathbb{N} \mapsto \mathbb{N}$ for which $l(n) > n$ for all $n \in \mathbb{N}$, such that $|G(s)| \geq l(|s|)$, for every $s \in \{0, 1\}^*$.*

The function l is called the expansion factor of G .

(2) *Pseudorandomness: the ensemble $\{G(U_n)\}_{n \in \mathbb{N}}$ is pseudorandom.*

The input, s , to the generator is called its *seed*, and it is required to be uniformly chosen in $\{0, 1\}^n$ in order for the generator to output a pseudorandom string. This is intuitive, since otherwise we would require a deterministic algorithm to create random-looking strings, starting from non-random input, clearly a contradiction. The function of a PRG can be alternatively seen as that of expanding a random input. It allows us to extract from a short uniformly chosen input several useful random-looking strings that can be used for practical applications, i.e., that can be given as input to efficient algorithms that cannot distinguish pseudorandom from uniformly chosen strings. It should not come as a surprise then, that the output of a PRG cannot be uniformly distributed [62], or even statistically close to uniform.

Notice that in the definition we allow the expansion factor to be very small: even a function l , for which $l(n) = n + 1$ would suffice. Although this does not look like a significant gain, given a PRG G_1 with expansion factor $l_1(n) = n + 1$, there is a universal way to construct a PRG G_2 with expansion factor $l_2(n) = p(n)$, for *any polynomial*, $p(\cdot)$ [62]. Intuitively, G_2 applies G_1 on a seed of size n , to get $n+1$ bits. It outputs the first bit which is “pseudorandom” by assumption, and re-applies G_1 to the last n bits, which constitute a pseudorandom string and hence are indistinguishable, by the efficient algorithm G_1 , from uniformly chosen bits. This continues iteratively until $p(n)$ bits are outputted.

2.2.2 Encryption schemes

Providing *secret communication over insecure media* is the most basic problem of cryptography. The setting of the problem consists of two parties communicating through a channel that is possibly tapped by an adversary. The sender *encrypts* a message before transmission and the receiver *decrypts* the scrambled string to obtain the original message. The legitimate receiver possesses some secret information, called the *decryption key*, allowing him to correctly decrypt transmitted messages.

Traditional encryption schemes are *symmetric*, also called *private key*: both users hold the same secret key, enabling encryption and decryption of messages. Thus, the security of private encryption schemes presupposes the secrecy of the encryption keys, giving rise to the *key distribution* problem. A typical solution is to use a secure but usually more expensive channel, e.g., secure physical transmission.

The notion of computational complexity, introduced in the last two decades, allows the construction of asymmetric encryption schemes, i.e., ones that use different keys for encryption and decryption. These provide a radical solution to the key distribution problem. Namely, it becomes possible to construct *public-key* encryption schemes, as proposed by Diffie and Hellman [44], in which given the encryption key it is computationally infeasible to extract the decryption key. Public-key encryption schemes trivially solve the key distribution problem, since the encryption key can be publicized. In addition, in the common setting of n users wishing to communicate securely via pairwise private channels, only n pairs of private-public keys have to be created, a significant gain over the $\frac{n(n-1)}{2}$ secret keys needed for private encryption schemes.

One of the important results in cryptography in the last years has been the success in formally defining and constructing secure encryption schemes, a work pioneered by Goldwasser and Micali [68, 69]. Informally, an encryption scheme is secure if seeing the encrypted message (*ciphertext*) does not give *any* partial information about the message (*plaintext*) that was not known beforehand. Intuitively, we would like a secure encryption scheme to capture:

- *Computational hardness*: encryption and decryption should be efficient procedures, but given the ciphertext no efficient algorithm should be able to obtain any partial information about the plaintext.
- *Security with respect to any probability distribution of messages*: the scheme should remain secure regardless of the messages that are encrypted; hence security should be defined without any assumptions on the distribution of the plaintext.
- *Hardness of gaining any partial information*: given the ciphertext it should be hard not only to find the plaintext, but also to obtain *any* partial information about it, e.g., size, parity, least significant bit, etc. The motivation here is that there should be no limit on how a system is used, hence there is no way of determining what information about the message is the most “important.” We therefore require that secure schemes prevent extraction of any partial information.
- *Hardness against a-priori information*: security should be guaranteed even if

the adversary has some a-priori information about the message, e.g., language used, relation between messages, etc.

An encryption scheme that satisfies these requirements will be called *semantically secure*. We now proceed to formally define encryption schemes and semantic security.

Definition 2.2.2 (Public-key encryption scheme) A Public-key encryption scheme consists of three probabilistic polynomial-time algorithms (G, E, D) as follows:

- (1) G is a key generating algorithm, with $G(1^n) = (e, d)$, where e is the public key, d is the private key, n is a security parameter, and $|e| = |d| = n$.
- (2) E is an encryption algorithm and D is a decryption algorithm, such that for every message m of size $|m| = n$, every pair of keys (e, d) generated by G on input 1^n , and all the possible coin tosses of E ,

$$D_{G(1^n)}(E_{G(1^n)}(m)) \stackrel{\text{def}}{=} D(E(m, e), d) = m .$$

The definition captures *symmetric* encryption schemes for $d = f(e)$, where f is a publicly computable polynomial-time function.

For all practical purposes we can relax the definition so that the public and private keys are of length polynomial on the security parameter and that the decryption succeeds with overwhelming probability in n , where the probability is taken over the coin tosses of G, E and D .

Definition 2.2.3 (semantically secure encryption) An encryption scheme (G, E, D) is said to be semantically secure if, for every ensemble $X = \{X_n\}_{n \in \mathbb{N}}$ of polynomial random variables, for every polynomial function h , for every function f , and for every probabilistic polynomial time algorithm A , there exists a probabilistic polynomial time algorithm A' such that for every constant $c > 0$ and for every sufficiently large n ,

$$\text{Prob} \left[A(E_{G(1^n)}(X_n), h(X_n), 1^n) = f(X_n) \right] \leq \text{Prob} \left[A'(h(X_n), 1^n) = f(X_n) \right] + \frac{1}{n^c} ,$$

where the probability is taken over the coin tosses of A (resp. A'), E and G , and the distribution of X .

Intuitively, given any a-priori information, $h(X_n)$, no algorithm A can obtain some information, $f(X_n)$, from the ciphertext that could not have been efficiently computed by A' without the ciphertext.

There is an alternative way to define secure encryption, which sometimes proves more useful in practice. The definition is based on indistinguishability; intuitively, if it is infeasible for an adversarial algorithm to distinguish between the encryptions of *any* two messages, even if these messages are given, then the encryption should not reveal any information about the messages. It has been proven that *an encryption scheme secure in the sense of indistinguishability is semantically secure* [93]. The inverse direction also holds under an additional condition. We refrain from presenting these proofs, but the formal definition of security in the sense of indistinguishability is useful for our purposes:

Definition 2.2.4 (encryption secure in the sense of indistinguishability) *An encryption scheme (G, E, D) is said to be secure in the sense of indistinguishability if, for every probabilistic polynomial time algorithm F (for “Find”), for every probabilistic polynomial time algorithm A , for every constant $c > 0$ and for every sufficiently large n ,*

$$\text{Prob} \left[F(1^n) = (a, b) \text{ such that } \left| \text{Prob} \left\{ A((a, b), E_{G(1^n)}(a)) = 1 \right\} - \text{Prob} \left\{ A((a, b), E_{G(1^n)}(b)) = 1 \right\} \right| > \frac{1}{n^c} \right] < \frac{1}{n^c},$$

where the probability is taken over the coin tosses of F, A, E and G .

At this point it should be clear that no deterministic encryption scheme, i.e., one that always produces the same encryption for a message, can be semantically secure; in the simple case when a message is retransmitted an adversary will immediately recognize it. However, such encryption schemes are widely in use, on the assumptions that the message space is sufficiently large and message repetition is therefore rare. These assumptions are justified when messages are chosen uniformly from a large message space. An example of a deterministic public key encryption scheme is the RSA collection of functions (see Section 2.1.2), with d and e serving as the private and public keys, respectively.

Examples of secure encryption schemes

The construction of semantically secure encryption schemes can be based on cryptographic primitives such as pseudorandom generators, one-way trapdoor permutations or one-way permutations. We present two representative examples: private-key

encryption based on pseudorandom generators and public-key encryption based on one-way trapdoor permutations [12, 63]. In addition we describe the El-Gamal [45] public-key encryption, a very efficient probabilistic scheme.

To illustrate **private-key encryption**, consider the following scenario. Let Alice (A) and Bob (B) be two parties that wish to communicate securely using a shared key. If both have access to a Pseudorandom Generator (PRG) then they can choose the private key to be a uniformly chosen seed to the PRG. As in all private-key systems, the key is assumed to be exchanged secretly, by outside secure means. Encryption is done by XORing the messages with different portions of the pseudorandom output, i.e., no portion is used twice, while decryption is done by again XORing the same portion of the PRG output. Essentially, the output of the generator is used as a “one-time pad.” Using a one-time pad in this manner is proven to result in an unconditionally secure encryption scheme [77, Ch. 13] [119]; hence insecurity of the proposed scheme, i.e., upon using a PRG output instead of truly random strings, can be transformed into a test which distinguishes pseudorandom strings from uniformly chosen strings, a contradiction on the definition of the former (see Section 2.2.1 above).

To exemplify **public-key encryption** consider a collection of one-way trapdoor permutations (I, D, F) , as defined in Section 2.1.3, and a hard-core predicate $H(\cdot)$ for this collection (as discussed in Section 2.1.4 such predicates exist for any one-way collection). Then, the following triplet (G, E, D) of probabilistic polynomial-time algorithms defines a public-key encryption scheme [12, 63]:

- *Key generation:*

$$G(1^n) = I(1^n) = (i, t(i)) ,$$

where $e = i$ is the public encryption key and $d = t(i)$ is the trapdoor information which serves as the secret decryption key.

- *Encryption:* For any n -bit message $m = b_1, \dots, b_n$, the encryption algorithm is defined as

$$E(m, e) = ((c_1, y_1), \dots, (c_n, y_n)) ,$$

where c_j, y_j are computed as follows, for each $j = 1, \dots, n$:

$$\begin{aligned} x_j &= D(e), \text{ i.e., } x_j \text{ is randomly sampled in } D_e, \\ y_j &= F(e, x_j), \text{ i.e., } y_j = f_e(x_j), \text{ and} \\ c_j &= m_j \oplus H(x_j). \end{aligned}$$

- *Decryption*: For any encrypted message $C = ((c_1, y_1), \dots, (c_n, y_n))$, the decryption algorithm

$$D(C, d) = m_1 \cdots m_n ,$$

computes, for each $j = 1, \dots, n$:

$$x_j = F^{-1}(d, y_j), \text{ i.e., } x_j = f_e^{-1}(y_j), \text{ and}$$

$$m_j = c_j \oplus H(x_j).$$

Intuitively, the above encryption scheme is semantically secure, as no information about each message bit is revealed; we refer to Goldreich’s notes [63] for a formal proof.

The **El-Gamal public-key encryption** scheme [45] is based on the DLA (see Section 2.1.2 for a description). Similar to the examples above, it is a probabilistic encryption scheme, i.e., a specific message has many—exponential in the security parameter—possible encryptions. Formally, the El-Gamal encryption scheme is defined by a triplet (G, E, D) of probabilistic polynomial-time algorithms, with the following properties:

- The *key generating algorithm*, G , on input 1^n , where n is the security parameter, outputs a public key, $e = (p, g, y)$, and a private key, $d = (p, g, x)$, where
 - (p, g) is an instance of the DLP collection, i.e., p is a uniformly chosen prime of length $|p| = n$, and g is a uniformly chosen generator of Z_p^* ,
 - x is a uniformly chosen element of Z_{p-1} , and
 - $y \equiv g^x \pmod{p}$.
- The *encryption algorithm*, E , on input (p, g, y) and a message $m \in Z_p$, uniformly selects an element k in Z_{p-1} and outputs

$$E((p, g, y), m) = (g^k \pmod{p}, my^k \pmod{p}) .$$

- The *decryption algorithm*, D , on input (p, g, x) and a ciphertext (y_1, y_2) , outputs

$$D((p, g, x), (y_1, y_2)) = y_2(y_1^x)^{-1} \pmod{p} .$$

Intuitively, a message is “masked” by multiplying it with a randomly chosen number. Information that allows the holder of the private key to obtain this random number and decrypt the plaintext is included in every ciphertext.

As mentioned in Section 2.1.2, the efficiency of the algorithms can be improved if the key generating algorithm selects g as a generator of the unique subgroup Z_q^* of Z_p^* , with q a prime and $a > 1$ an integer such that $p = aq + 1$.

2.2.3 Digital signatures and general computation protocols

Being able to “sign” a digital document is as important as signing physical documents. Naturally, the properties of digital signatures must reflect those of their physical counterparts:

- each user must be able to *efficiently generate his own signature* on documents of his choice;
- each user must be able to *efficiently verify* whether a given string is a signature of another specific user on a specific document; but
- *nobody should be able to efficiently produce signatures of other users* to documents they did not sign.

The idea of a digital signature satisfying these requirements first appeared in Diffie and Hellman’s paper “New directions in Cryptography” [44]. They proposed that each user publishes a “public key,” while keeping secret a “private key,” also called the “secret key”. Signatures are produced with the private key and they are verified using the public key. The concept thus mirrors that of public-key encryption and is similarly based on one-way functions, which prevent users from obtaining the private key given the public key. Formally:

Definition 2.2.5 (digital signature scheme) *A signature scheme is a triplet of algorithms (G, S, V) with a security parameter n . The key generating algorithm G is a probabilistic polynomial-time algorithm that generates ordered pairs (s, v) of a private (signing) key, s , and a public (verification) key, v :*

$$G(1^n) = (s, v) .$$

S is a signing probabilistic polynomial-time algorithm which produces signatures σ , with respect to a given key s produced by $G(1^n)$, for every message m of length $|m| = n$, taken from some set M :

$$S(s, m) = \sigma .$$

Lastly, V is a verifying probabilistic polynomial-time algorithm, which checks whether σ is a valid signature for m , with respect to the key v . For every (s, v) in the range of $G(1^n)$, and every $m \in M$ of length $|m| = n$, if σ is in the range of $S(s, m)$, then

$$V(v, m, \sigma) = 1 ,$$

otherwise $V(v, m, \sigma) = 0$.

Here the last condition can be relaxed, to allow the verifier a negligible probability of failure. We note that the signing algorithm can be transformed into a deterministic algorithm by including the coin tosses of S into s .

Note that the above definition does not guarantee unforgeability; unforgeability of digital signatures can be defined in a number of ways, depending both on the attacks an adversary mounts and on the forgery attempted. In regards to attacks, they range from

- a *known plaintext attack*, in which the adversary is given a set of signatures and the respective messages, to
- a *directed chosen plaintext attack*, where the adversary chooses a list of messages and asks the signer for their signatures, to
- the most severe *adaptive chosen plaintext attack*, or simply chosen message attack (*CMA*), in which the adversary uses the signer as an “oracle,” asking for signatures on messages of his choice.

In terms of forgery, there are several levels of success for an attacker:

- in *existential forgery* the adversary succeeds in obtaining a signature on *one* message, which may not be of his choice, or even meaningful;
- in *selective forgery* the adversary obtains a signature on a message of his choice; and
- in *universal forgery* the adversary, although unable to find the secret key of the signer, is able to forge the signature of any message. Finally,
- a *total break* means that the adversary succeeds in obtaining the signer’s private key.

Here we formally define the most general form of forgery, namely *existential forgery under adaptive chosen message attacks*:

Definition 2.2.6 (existentially unforgeable signature schemes) *A signature scheme (G, S, V) will be called existentially unforgeable under adaptive chosen plain-text attacks, or simply existentially unforgeable, if for all probabilistic polynomial-time forging algorithms F , for all polynomials $p(\cdot)$, and for all sufficiently large n ,*

$$\text{Prob} \left[F^{S_s}(v) = (m, \sigma) \text{ s.t. } V(v, m, \sigma) = 1 \text{ and } m \notin Q(S_s) \right] < \frac{1}{p(n)},$$

where F^{S_s} denotes that F can use the signing algorithm S with secret key s as an oracle for polynomially many queries, $Q(S_s)$ is the set of messages that have been signed by S_s on behalf of F , and where the probability is taken over the coin tosses of F, G, S, V and the pairs (s, v) as generated by $G(1^n)$.

Note that while the forging algorithm F has access to S_s it still does not know the secret key s .

Lamport [87] first showed how to satisfy existential unforgeability by introducing *tag-type* signatures: to sign one bit a user publishes a one-way function f and an ordered pair $(f(a), f(b))$, where a, b are random strings. To sign a “0”, the user reveals a , whereas a “1” is signed by revealing b . The triplet $(f, f(a), f(b))$ is thus the public key, whereas (f, a, b) is the secret key.

Bellare and Micali [7] showed how to base *tag-type*, existentially unforgeable signatures on *any* trapdoor permutation, f . Informally, the signer chooses a trapdoor function, f , from a collection of trapdoor functions, F , and publishes the public key $PK = \{f, a_{1,0}, a_{1,1}, \dots, a_{n,0}, a_{n,1}\}$ where $a_{i,j}$ are random strings and $n - 1$ are the bits needed to describe a function chosen from the collection F . The secret key consists of the trapdoor information of f . To sign a bit the signer first picks a new function g from the collection F and concatenates the bit to be signed with the description of g . Then he signs the n -bit long message, m , by presenting a preimage of f on $a_{i,j}$, where j is the value of the i -th bit of m . The preimage can be computed since the user knows the trapdoor information of f . After signing the bit, the public key is changed to $\{g, a_{1,0}, a_{1,1}, \dots, a_{n,0}, a_{n,1}\}$, i.e., the trapdoor function is refreshed. The procedure continues until the message to be signed is exhausted. Clearly, the signature process can be sped up by signing more than one bit at a time, by simply using $n - 1 + m$ pairs of random strings to sign a message of length m .

The resulting signature is existentially unforgeable under adaptive chosen message attacks [7]. Note that the signature has a “memory”: to verify the signature on a

message, the verifier must verify the signatures on all previous messages, down to the first; then he can verify that the public key is correct. Hence, verifiers must maintain a list of past signatures. This requirement can be slightly relaxed if at each iteration the signer signs two trapdoor functions; then a binary tree of signatures is maintained and, although the memory requirement remains large, verification requires traversing a path that is logarithmic, instead of linear, to the messages signed so far.

Although trapdoor permutations seemed to be necessary for digital signatures, Naor and Yung [96] showed how to construct a tag-type existentially unforgeable signature based on *any* one-way permutation.

Although tag-type signatures are provably secure, their implementation is clearly inefficient. Descriptions of more practical signature schemes based on the RSA and DLP collection of functions are given in Sections 2.2.6 and 2.2.7.

Blind signatures

A special type of digital signatures are blind signatures, introduced by Chaum in [27]. In a blind signature scheme the *signer* is willing to sign any message on behalf of a *requestor*, without seeing this message or the resulting signature. The model is thus enhanced by including two additional protocols available to the requestor, one for blinding a message to be signed and one for un-blinding the response of the signer to obtain the signature on the original message. The definition of the underlying signature scheme remains intact. Similarly, security requires that no party other than the signer should be able to sign *any* message, except for the messages legitimately signed on behalf of the requestor, even after querying the signer polynomially many times. In addition, it is required that the signer obtains no information about the message and the resulting signature.

Chaum showed how to construct a blind signature scheme based on RSA. Assume for simplicity that the RSA trapdoor collection is a secure signature scheme (the RSA collection, and a general way to turn a trapdoor permutation into a signature scheme, are described in Section 2.1.3). Section 2.2.6 shows how RSA can be used to construct an efficient existentially unforgeable signature scheme; the construction presented here is directly applicable to this extension. Informally, given an instance $(N, e), (N, d)$ of the RSA collection selected by I_{RSA} and a message m to be signed, the requestor invokes $D_{RSA}(N, e)$ to uniformly select a blinding string b in Z_N^* , and

computes

$$m' \equiv m \cdot F_{RSA}((N, e), b) \equiv mb^e \pmod{N} .$$

The signer responds with a signature on the blinded string:

$$\sigma' \equiv F_{RSA}^{-1}((N, d), m') \equiv m^d \equiv m^d b \pmod{N} ,$$

from which the requestor can then extract the signature on m , by computing $\sigma \equiv \sigma' b^{-1} \equiv m^d \pmod{N}$. It can be shown that, as required for a blind signature, the signer obtains no information about m or σ .

General computation protocols

Digital signatures are a very important notion in cryptography, as they provide a new horizon beyond simply “secure communication.” Users can now make self-binding statements that no-one else can imitate, thus a legitimate system can be thought of as including all users, with adversaries considered as internal yet dishonest users. In this wide sense, *cryptography is concerned with any problem in which one wishes to limit the affect of dishonest users.*

General solutions to such problems are given by general computation protocols introduced by Yao [131], also called *fault-tolerant protocols* or *secure distributed computation protocols* or simply *cryptographic protocols*, which allow any set of parties to compute any function based on their secret inputs, such that the output of the function becomes known to every party but the input of each party remains secret. These protocols provide *privacy* against the players’ inputs and *robustness*, in the sense that no party can “influence” the output of the function—beyond the influence obtained by selecting its own input.

It is outside the scope of this thesis to provide an exhaustive list of the advances in this area; we thus limit the discussion to a few of the most interesting results. For the case of only two interacting parties it has been shown by Goldreich et. al. [66] that there exist *two-party* cryptographic protocols for every interactive computational problem that only require the existence of *any* trapdoor permutation. Even stronger results, allowing up to unconditional security, exist for *multi-party* protocols [66, 61, 29, 10], where it also makes sense to allow adversaries to be *active*: an adversarial user not only tries to obtain illegal information from the protocol (this is called a *passive* adversary) but may also deviate from the description of the protocol. Clearly,

cryptographic protocols against active adversaries require at least a majority of honest users.

As far as this thesis is concerned the most important fact about general computation protocols is that although they can be used to solve electronic cash, as shown by [41, 105], they are extremely slow to implement since their generality forces them to operate on a gate-by-gate basis: the function to be computed is broken into a circuit and every gate of the circuit is evaluated separately. Each of these evaluations is performed interactively, therefore *the communication complexity depends on the computational complexity of the honest participants*. This clearly makes the protocols inappropriate for practical applications. Therefore, general computation protocols are important in showing the existence of a solution, however this solution is usually not applicable in practice.

2.2.4 Interactive proof systems and zero-knowledge proofs

An interactive proof is a protocol in which a “prover” conveys a convincing argument to a probabilistic, polynomially-bounded “verifier.” The notion was introduced by Goldwasser, Micali and Rackoff [70] and Babai and Moran [4, 5] and is formalized by considering proofs of membership into a language. For a concrete definition, let P, V be two interactive probabilistic machines, having access to a common input tape and sending messages through a pair of communication tapes, but which are otherwise inaccessible to one another. We can then define an interactive proof system for a language L :

Definition 2.2.7 (interactive proof system) *A pair of probabilistic interactive machines (P, V) , is called an interactive proof system for a language L if machine V is polynomial-time and the following two conditions hold:*

(1) Completeness: *For every $x \in L$*

$$\text{Prob}(\langle P, V \rangle(x) = 1) \geq \frac{2}{3}$$

(2) Soundness: *For every $x \notin L$ and any machine B*

$$\text{Prob}(\langle B, V \rangle(x) = 1) \leq \frac{1}{3}$$

Here, completeness refers to the success of an honest prover, whereas soundness limits the success probability of *any* dishonest potential prover. Note that we could alternatively define completeness (resp. soundness) to hold with overwhelming (resp. negligible) probability on the length of x . But the above is a more general definition since, as is the case in *BPP*, the error probability of any such system can be made exponentially small by repeating the interaction polynomially many times.

It can be shown that all languages in *NP* have an interactive proof system [62].

An interesting class of interactive proofs are “zero-knowledge” proofs [70]. The idea is to prove a statement without revealing *any* information to the verifier other than the correctness of the statement, even if the latter behaves in an arbitrarily adversarial way. This vague requirement is formalized by showing that the verifier himself could generate a view that is indistinguishable from his view of the protocol. Zero-knowledge can be defined in a perfect, statistical or computational sense, depending on the indistinguishability of the two views, but the latter notion is the most general and interesting in practice and the only one presented here. For the formalization it is required that it is efficient to generate a simulated view that is computationally indistinguishable from the output of a, potentially dishonest, verifier V^* after interacting with P :

Definition 2.2.8 ((computational) zero-knowledge) *Let (P, V) be an interactive proof system for some language L . We say that (P, V) is (computational) zero-knowledge if for every probabilistic polynomial-time interactive machine V^* there exists a probabilistic polynomial-time algorithm M^* so that the following two ensembles are computationally indistinguishable*

- $\{\langle P, V^* \rangle(x)\}_{x \in L}$ (i.e., the output of V^* after interacting with P on common input x);
- $\{M^*(x)\}_{x \in L}$ (i.e., the output of M^* on input x).

Machine M^ is called a simulator for the interaction of V^* with P .*

An alternative intuitive way to view the definition is that the interaction of V^* with P can be efficiently simulated; i.e., P does not convey any knowledge that could not be extracted in his absence.

A common technique for the zero-knowledge proof of a proposition is to first “scramble” the proposition in some way and present it to the verifier. The verifier

challenges the prover to either (a) prove the scrambled proposition, or (b) prove that the scrambled proposition is true if and only if the original proposition is true. Either of the two conditions reveals nothing to the verifier; however the prover has probability of $1/2$ of being caught if the statement is false, as long as the choice between (a) and (b) is uniform. Thus, exponentially high confidence can be achieved by the verifier after only a linear number of iterations of the procedure. Observe that for such a technique to work it is necessary to “bind” the prover in the construction of the scrambled proposition. Otherwise the prover could easily fake the proof by (i) proving a scrambled proposition that is unconnected to the original statement, or (ii) performing a correct scrambling that results in a false proposition, and selecting (i) or (ii) for each challenge (a) or (b) respectively. This binding is achieved with a *bit commitment* scheme, discussed in the next section.

Zero-knowledge proofs are very useful in cryptography and in electronic cash in particular. Proofs of identity (a party proves knowledge of his secret key without revealing it), of knowledge of some information (e.g., the user proves knowledge of the coin’s inner construction upon payment), or of the validity of encrypted messages (e.g., the user proves that he correctly encrypted his identity in an electronic coin) are clarifying examples. A general result in this area is by Goldreich, Micali and Wigderson [67] who showed that for *every* language in *NP* there is a zero-knowledge proof of membership, assuming the availability of secure encryption schemes.

2.2.5 Bit commitment

Commitment schemes are a major building block for many cryptographic protocols; they enable a party to commit to a value without revealing it. At a later time the party can “open” the commitment, while it is guaranteed that the “opening” can yield only a single value, the one committed to in the committing phase. Commitment schemes are the digital analogue of sealable opaque envelopes.

Informally, a commitment scheme is a two-phase, two-party protocol through which one party, called the *sender*, can commit himself to a *value* so that the following two conditions are satisfied:

- (1) *Secrecy*: At the end of the first phase (the *commit phase*), the other party, called the *receiver*, does not gain any knowledge of the sender’s value. This requirement must be satisfied even for faulty receivers.

- (2) *Unambiguity*: Given the transcript of the interaction in the commit phase, there exists exactly one value which the receiver may later, i.e., in the second phase (called the *reveal phase*), accept as a legal “opening” of the commitment. This requirement must be satisfied even for faulty senders.

Formally,

Definition 2.2.9 (bit commitment scheme) *A bit commitment scheme is a pair of probabilistic polynomial-time interactive machines, denoted (S, R) (for sender and receiver), satisfying:*

- *Input specification*: The common input is an integer n presented in unary (the security parameter). The private input to the sender S is a bit u .
- *Secrecy*: The receiver R , even when deviating arbitrarily from the protocol, cannot distinguish a commitment to 0 from a commitment to 1. Namely, for every probabilistic polynomial-time machine R^* , the following ensembles, describing the output of R^* in the two cases, are polynomially indistinguishable
 - $\langle S(0), R^* \rangle(1^n)$ and
 - $\langle S(1), R^* \rangle(1^n)$

- *Unambiguity*:

Preliminaries

- **A receiver’s view of an interaction** with the sender, denoted (r, \bar{m}) , consists of the random coins r used by the receiver and the sequence of messages \bar{m} received from the sender.
- Let $\sigma \in \{0, 1\}$. We say that a receiver’s view of such interaction, (r, \bar{m}) , is a **possible σ -commitment** if there exists a string s such that \bar{m} describes the messages received by R when R uses local coins r and interacts with machine S which uses local coins s and has input $(\sigma, 1^n)$.
- We say that the receiver’s view (r, \bar{m}) is **ambiguous** if it is both a possible 0-commitment and a possible 1-commitment.

The unambiguity requirement states that, for all but a negligible fraction of the coin tosses of the receiver R , there exists no sequence of messages from the sender which, together with these coin tosses, forms an ambiguous receiver view.

Namely, for all but a negligible fraction of the $r \in \{0,1\}^{\text{poly}(n)}$ there is no \bar{m} such that (r, \bar{m}) is ambiguous.

There are two flavours of bit commitment schemes. In their basic form above, secrecy is defined analogously to the indistinguishability of encryptions, i.e., the receiver is polynomially bounded, whereas unambiguity allows an unbounded sender to cheat only with negligible probability, regardless of its computational power. The first scheme of this form was presented by Blum [11] for two-party coin flipping, and is based on quadratic residuosity. For $N = P \cdot Q$ a Blum integer (i.e., P, Q primes with $P \equiv Q \equiv 3 \pmod{4}$), a bit is committed by sending a quadratic residue mod N for a zero, or a quadratic non-residue for a one. The scheme is unambiguous, since no element can be both a residue and a non-residue; and secret, as long as the polynomially bounded receiver cannot break the QRA (see Section 2.1.1 for a definition of quadratic residues, Blum integers and the QRA). In this, as in all commitment schemes, we assume without loss of generality that on the second phase the sender presents his view of the commitment phase; this “opens” the commitment and allows the receiver to verify it. Naor [95] has shown how to design bit-commitment schemes based on any one-way function.

A dual flavour of bit commitment, called “perfect bit commitment” introduced by Brassard et al. in [18], allows the receiver to be computationally unbounded, i.e., secrecy is defined in an information theoretic way. In this case, however, unambiguity is satisfied only computationally, i.e., the sender is no longer allowed to be unbounded. It can be shown that there cannot exist bit commitment schemes where both the secrecy and unambiguity are satisfied in an information-theoretic way [63].

2.2.6 Hash functions and the random oracle model

Consider the problem of message integrity: messages transmitted between two users need to be authenticated, i.e., each user must be guaranteed that each arriving message is unaltered. A trivial solution would be to digitally sign each transmitted message. However, digital signatures are typically costly procedures and their use must be minimized for message transmission to be efficient. A way to solve the problem efficiently is for the sender to compute and digitally sign a hash of each message

and for the receiver to re-compute the hash and verify the signature. The hash function, h , needed for this setting is publicly known and requires no secret keys, while it guarantees that given a value x it is computationally hard to find a $y \neq x$ such that $h(x) = h(y)$, i.e., collisions are hard to find.

Hash functions are a major building block for several cryptographic protocols, including pseudorandom generators [64], digital signatures [96], bit commitment schemes [25] and message authentication. There are two basic flavours of hash functions. The strongest notion, as set forth by Damgård in [40] is that of collection of **Collision Intractable Hash Functions (CIHF)**, also called *strong collision free hash functions*, in which given a randomly chosen member h of the collection it is computationally infeasible to find a pair x, y with $x \neq y$, such that $h(x) = h(y)$. It can be shown that if a function belongs in a CIHF collection then it is one-way. Damgård shows how to construct CIHFs assuming there exists a collection of claw-free permutations: informally, this is a collection of pairs of functions which are easy to evaluate, both have the same range, but it is infeasible to find a range element with preimages of it under both functions.

A weaker notion is that of collection of **Universal One-Way Hash Functions (UOWHF)**, also called *weak collision free hash functions*, in which given a member h of the collection and an element x in the domain of h , it is infeasible to find a $y \neq x$, such that $h(x) = h(y)$. In other words, the adversary is not allowed to choose both x and y simultaneously, but the selection must be independent. The concept was proposed by Naor and Yung [96], and is applicable to the message authentication problem as discussed above, but also on digital signatures, as shown by its constructors.

The role of a hash function is to compress its input, in such a way that none can efficiently find two inputs that compress to the same string. Regardless of the nature of the function, an adversary can always select values at random from the domain of the function in the hope that they hash on the same value. Counter-intuitively, it can be shown that if the range of a hash function is of size n , a guessing algorithm does not need to perform 2^{n-1} iterations (on average) in order to find a collision, but rather only $O(\sqrt{2^n})$. This is called the *birthday paradox*, and puts a lower bound on the range size of hash functions. Currently, a range of 160 bits is considered to be sufficient for most applications.

Hash functions are used in a wide variety of cryptographic protocols that seek to

be very efficient; especially in electronic cash, all schemes that can be efficiently implemented in practice utilize hash functions. There are two explanations for this fact: first, heuristic instantiations of CIHFs, such as MD5 [114], SHA [97], RIPE-MD [113], are very efficient, usually orders of magnitude faster than digital signatures or public encryptions schemes; and second, in practice for applications such as electronic cash where efficiency is very important, a significant relaxation is made regarding these functions. Namely, it is assumed that heuristic hash functions are not only collision intractable, i.e., it is computationally hard to find two inputs that hash on the same value, but also that their output is random. To be more specific, hash functions are assumed to behave like random oracles.

A **random oracle** is a function whose output is random and unpredictable: a random oracle is defined as a table with an input and an output field, such that the output field is random. Therefore the outputs $O(\alpha)$ and $O(\beta)$ produced by any two inputs α and β in the domain of a random oracle O are random and independent. Additionally, a random oracle is as good as any other random oracle, since both map their inputs to random strings. A random oracle can be substituted by a keyed PRG, i.e., a PRG whose output can be determined only by use of a key, but with the key being hidden in a black (tamper-proof) box; the tamper-proof box is used to hide the key even from the holder of the device. With this substitution the security of the original system is maintained, with the additional assumptions that the PRG is secure and the black box is tamper-proof. From the above it is clear that an implementation of a random oracle requires either exponential space, for storage of the input-output table, or tamper-proof hardware, in which the secret key for the PRG can be stored.

When systems assume the existence of random oracles, we will say that they operate under the **random oracle model**. Asserting that heuristically sound hash functions behave like random oracles is a very strong assumption. However, it is commonly used in practice and in electronic cash in particular [100, 15, 26, 55], especially in light of a construction by Bellare and Rogaway [8] showing instantiations of random oracles based on efficient hash function, such as MD5 [114] and SHA [97]. Informally, given a CIHF H , a random oracle h can be constructed as follows:

$$h(x) = H(c, 0, x) || H(c, 1, x) || H(c, 2, x) || \dots ,$$

where a prefix of sufficient length is chosen as the output of $h(x)$, and c is a constant,

unique for each oracle h ; i.e., choosing a constant $c' \neq c$ would result in another random oracle, g .

Random oracles are very powerful tools, allowing the construction of very efficient signatures. In addition, as shown in the next section, any identification scheme can be turned into a secure signature scheme under the random oracle model. These very efficient signatures are used as building blocks for a multitude of cryptographic schemes.

To exemplify such constructions, we describe how the RSA trapdoor collection $(I_{RSA}, D_{RSA}, F_{RSA})$ (we refer to Sections 2.1.2 and 2.1.3 for a detailed description of this collection) can be transformed into a provably secure signature scheme. This is a recent result, due to Bellare and Rogaway [9]. The triplet (G, E, V) of polynomial-time algorithms, with security parameters n, n_0, n_1 such that $n \geq n_0 + n_1 + 1$, and random oracles $h: \{0, 1\}^* \mapsto \{0, 1\}^{n_1}, g: \{0, 1\}^{n_0} \mapsto \{0, 1\}^{n-n_1-1}$ is an existentially unforgeable signature scheme if

- The *key generating* algorithm, G , on input 1^n , runs I_{RSA} to obtain the public key (N, e) and the secret key (N, d) .
- The *signing* algorithm, S , on input the private key (N, d) and a message m , uniformly selects a string r in $\{0, 1\}^{n_0}$ and computes

- (1) $w = h(m, r)$,
- (2) $r^* = g_1(w) \oplus r$,
- (3) $x = 0||w||r^*||g_2(w)$, and
- (4) $y \equiv F_{RSA}^{-1}((N, d), x) \equiv x^d \pmod{N}$,

where g_1 returns the first n_0 bits of g , g_2 the remaining $n - n_0 - n_1 - 1$ bits of g , and \oplus denotes bitwise XOR. S outputs the signature y on m .

- The *verifying* algorithm, V , on input the public key (N, e) , a message m , and a signature y ,
 - (1) computes $x \equiv F_{RSA}((N, e), y) \equiv y^e \pmod{N}$,
 - (2) breaks up x as $b||w||r^*||\gamma$,
 - (3) computes $r = r^* \oplus g_1(w)$,
 - (4) outputs $\begin{cases} 1 & \text{if } h(m, r) = w \text{ and } g_2(w) = \gamma \text{ and } b = 0, \text{ or} \\ 0 & \text{otherwise.} \end{cases}$

2.2.7 Identification schemes

There are many scenarios where zero-knowledge proofs of identity, introduced by Feige, Fiat and Shamir [52, 51, 48, 49], also called *identification schemes*, are necessary: a user wants to convince the bank of his identity when withdrawing funds from his account, but does not want the bank to obtain any of his secret information; to remotely log in to a computer station the user must prove his identity to the server, without revealing any other information to him, or to potential eavesdroppers. Informally, in an identification scheme a *prover* wants to prove to a *verifier*, in a zero-knowledge manner, that he knows some secret information verifiable by the verifier; usually the secret information is the prover's private key on a secure encryption scheme. Identification schemes can thus be based both on the private and public-key paradigm, but clearly the latter is more desirable as no secure key distribution method is needed and one public key suffices for all potential verifiers.

There is a general way to construct an identification scheme. Assume that the prover, P , and the verifier, V , have access to a secure public-key encryption scheme and a secure bit commitment scheme. V accepts P 's identity if P proves knowledge of his secret key, but the proof must be zero-knowledge:

- (1) V uniformly selects a string and computes a commitment on it, which he sends to P ,
- (2) P uniformly selects another string of the same predetermined length and sends it to V ,
- (3) V encrypts the XOR of the two strings under P 's public key, and sends the result to P ,
- (4) P decrypts the message, and sends a commitment on it to V ,
- (5) V reveals his commitment,
- (6) after P verifies it, he reveals the commitment on the decrypted message; finally,
- (7) if the decrypted message is correct, V accepts the proof.

The protocol convinces V that P can decrypt a randomly chosen message, hence that he holds the private key, while it is zero-knowledge for V ; it is easy to verify that V can simulate his view of the protocol.

An ideal identification scheme has minimal computation and communication complexity, enabling its implementation on computationally weak machines, such as smart cards. Such schemes are necessary for electronic cash that seek to be applied in similar environments. We present **the Schnorr identification scheme** [116] which is based on the DLA and is extensively used in later chapters:

- The *setup* consists of an instance of the DLP collection (see Section 2.1.2 for a description) with the operations being performed on a subgroup of prime order; i.e., a uniformly chosen prime q of length $|q| = n$, where n is the security parameter, a prime $p = \alpha q + 1$ for some integer $\alpha > 1$ and a uniformly chosen generator g of G_q . The identifying information of the prover P is $I \equiv g^u \pmod{p}$, where u is P 's secret information, uniformly chosen in Z_{p-1} .
- In the *protocol*, the prover P proves to the verifier V his identity by asserting that he knows u , without revealing it:
 - (1) P computes and sends to V $y \equiv g^m \pmod{p}$, for $m \in_R Z_q$, where \in_R denotes “chosen at random”,
 - (2) V replies with a uniformly chosen string $e \in_R \{0, 1\}^n$,
 - (3) P responds with $r \equiv eu + m \pmod{q}$; and
 - (4) V checks whether

$$g^r \stackrel{?}{\equiv} I^e y \pmod{p} .$$

P can cheat by guessing e and choosing $y \equiv I^{-e} g^m \pmod{p}$, but the probability of this happening is $\frac{1}{2^n}$, i.e., negligible in n . In terms of security for P , the scheme is not zero-knowledge, but instead conjectured to provide no useful information about u .

Note that under the random oracle model the scheme can be made non-interactive; the challenge e can be computed as a hash of the identifying string I and the randomly chosen y :

$$e = H(I, y) .$$

It is important to note that in this case the security parameter n must be enlarged to prevent the prover from guessing the output of the hash function. In particular, when H is a random oracle n must be doubled [8].

This technique can be applied to any identification scheme, and it can also be used to **convert any identification scheme into a signature scheme** [51]; the

only modification is that the message to be signed is included in the input of the hash function. The idea is to (1) substitute the receiver's uniformly chosen challenge, e , with the pseudorandom, and thus indistinguishable by the polynomially-bounded prover, output of the random oracle, and (2) force the prover to run the oracle after selecting the message m . To illustrate the construction, we present **the Schnorr signature scheme** [116]:

- The *key generating algorithm*, on input the security parameter n , generates the *signing key* (p, q, g, u, H) , with q a uniformly chosen prime of length $|q| = n$, $p = \alpha q + 1$ a prime, for some integer $\alpha > 1$, g a uniformly chosen generator of G_q , u a uniformly chosen element of Z_q and H a hash function that is assumed to behave like a random oracle, and the *verifying key* $(p, q, g, I \equiv g^u \pmod{p}, H)$.
- The *signing algorithm*, on input (p, q, g, u, H) and a message m , selects uniformly a string $x \in_R Z_q$ and computes
 - (1) $y \equiv g^x \pmod{p}$,
 - (2) $e = H(m, I, y)$, with $|e| > n$, and
 - (3) $r \equiv eu + m \pmod{q}$.

The algorithm ends outputting (m, y, e, r) .

- The *verifying algorithm*, on input (p, q, g, I, H) and (m, y, e, r) verifies that the following two conditions hold
 - (1) $g^r \equiv I^e y \pmod{p}$, and
 - (2) $e = H(m, I, y)$.

Recently, Pointcheval and Stern [107] have shown that under the random oracle model, i.e., when the hash function H is a random oracle, Schnorr signatures, and in fact all signature schemes converted from of identification schemes, are existentially unforgeable under adaptive chosen plaintext attacks. This illustrates the practicality of hash functions, especially in light of efficient implementations such as MD5 [114] and SHA [97].

Chapter 3

Survey of electronic cash

Imagine the surprise of customers of physical and Internet-based shops to realize that not only all their transactions have been monitored and recorded, but also that this information has been used for direct-marketing techniques, adjustment of insurance premiums, determination of their credit-worthiness, and other legitimate or illegitimate tasks—often without their consent or even knowledge. This unpleasant scenario does not belong to the distant future but is approaching rapidly, aided by electronic payments. The latter make it extremely easy, and sometimes necessary, e.g., in credit card transactions, for banking institutions to record each customer's payments and construct user spending profiles. The problem of decreased user privacy would never reach a dangerous, or even irritating, threshold if only physical cash were used for payments, since the latter preserves the users' anonymity.

Noting this problem, D. Chaum [27, 28] introduced anonymous electronic payments named, appropriately, “electronic cash” due to their similarity in terms of providing user anonymity with physical cash. Similar to physical cash the bank issuing the electronic *coins*, even in collaboration with a shop, cannot trace the coins to their owner. We assume of course that the shop possesses no out-of-band knowledge of the user's identity, such as looking at his identification documents or finding his computer's network address, i.e., we concentrate on the payment protocol. It is then up to the user or to appropriate network protocols or transmission methods to prevent identification in the remaining of the process.

Electronic cash (or “e-cash”), however, provides stronger anonymity than physical cash, since the latter clearly displays serial numbers which can be recorded to allow

further tracing. Contrary to their physical counterparts, e-cash have an inherent limitation; they are easy to copy and reuse. To guard against what is traditionally called “double-spending” of electronic coins, Chaum [27] proposed for the bank to keep a list of spent coins and check every deposited coin against this list. Hence the bank has to be *on-line* at payment, or else the shop will have no guarantee that the coins received are valid. Forcing the bank to be on-line at payment is a very strict requirement; consider that in current payment systems the number of payments is at least an order of magnitude larger than the number of deposits, which are performed in batch mode usually at the end of each day. Checking each payment on-line would force banks to migrate to more expensive computers which allow real-time payment verifications. Furthermore, these computers would present a single point of failure for the system since no payments could be conducted off-line.

To bypass the on-line requirement, Chaum, Fiat and Naor [32] proposed to *detect* instead of preventing double-spending. In this model, called *off-line electronic cash*, anonymity holds only as long as users are legitimate, whereas double-spenders are identified. Off-line electronic cash has since enjoyed considerable attention in the research community, aiming to optimize the originally proposed system for practical applications.

We stress that anonymity is the distinguishing property of e-cash; user privacy is guaranteed even against collaboration of all other involved parties, i.e., bank and shops. There have been several references in the literature for anonymous-like systems, providing partial privacy under the assumption of no collaboration between involved parties [6, 91, 89, 120, 130], or assuming the central bank is honest. These are very strong and potentially unjustified assumptions, especially in light of financial benefits from disclosing personal information and efficient ways of collaboration between interested parties, such as bulk transfers of electronic files. An implication of this thesis is that truly anonymous e-cash can be implemented very efficiently without sacrificing security in comparison to existing account-based or anonymous-like systems, thus significantly limiting any efficiency advantages of systems operating under such relaxed anonymity models.

The off-line property is equally significant in e-cash, even in scenarios where shops and bank happen to be the same party (e.g., postal services): *on-line* systems require

constant and real-time involvement of the bank in every payment transaction, resulting in excessive communication and computation costs. In contrast, off-line systems usually operate in dual mode, verifying high-cost transactions on-line (to prevent high-volume fraud—even though the fraudulent party is identified), while the bulk of payments are processed in batch mode by the bank. We note that several currently employed payment systems—e.g., credit cards, bank and personal checks—operate in this dual-mode operation, showing the viability of the concept.

Therefore, *throughout this thesis*, “*electronic cash*” (or *e-cash*) will refer to “*off-line anonymous electronic cash*,” unless otherwise noted.

In this chapter we first present a formal model of electronic cash in Section 3.1 and protocols that operate under this basic model in Section 3.1.3. We then proceed to motivate and discuss extensions of the basic model, first for providing exact payments in Section 3.2 where some relevant protocols are discussed, then to anonymity revocation mechanisms in Section 3.3, and finally to other relevant extensions in Section 3.4. Section 3.5 concludes with a discussion and open problems. For concreteness and clarity we restrict detailed presentation of relevant work to the protocols that are built upon in later chapters; a high-level description of previous research is given otherwise.

3.1 Basic model

An anonymous off-line electronic cash (e-cash) system consists of three collections of probabilistic, polynomially-bounded parties, a bank \mathcal{B} , users \mathcal{U} , and shops \mathcal{S} , and three main procedures: withdrawal, payment and deposit (see Figure 3.1). Users and shops maintain an account with the bank, while

- \mathcal{U} withdraws *electronic coins* from his account, by performing a *withdrawal protocol* with bank \mathcal{B} over an authenticated channel.
- \mathcal{U} spends a coin by participating in a *payment protocol* with a shop \mathcal{S} over an anonymous channel, and
- \mathcal{S} performs a *deposit protocol* with the bank \mathcal{B} , to deposit the user’s coin into his account.

The system is *off-line* if during payment the shop \mathcal{S} does not communicate with the bank \mathcal{B} . It is *anonymous* if the bank \mathcal{B} , in collaboration with the shop \mathcal{S} , cannot trace

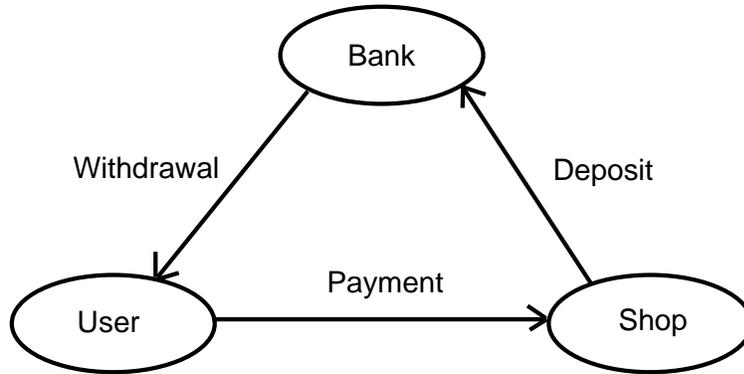


Figure 3.1: Model of electronic cash.

the coin to the user. However, in the absence of tamper-proof hardware, electronic coins can be copied and spent multiple times by the user \mathcal{U} . This has been traditionally referred to as *double-spending*. In anonymous on-line e-cash, double-spending is prevented by having the bank check if the coin has been deposited before. In off-line anonymous e-cash, however, this solution is not possible; instead, as proposed by Chaum et. al. [32], *the system guarantees that if a coin is double-spent the user's identity is revealed with overwhelming probability*.

Throughout this thesis, three additional protocols are sometimes defined, for clarity of design: the system setup, the shop setup, and the user setup (account opening). They describe the system initialization, namely creation and posting of public keys and opening of bank accounts. Although they are certainly part of a complete system, these protocols are frequently omitted as their functionality can be easily inferred from the description of the three main procedures; in this thesis the setup descriptions are provided only when necessary for clarity and unambiguity.

3.1.1 Security

Security of e-cash is defined in terms of four requirements: *Untraceability* (of user's payments), *Unreusability* (i.e., identification of double-spenders), *Unforgeability* (of electronic coins) and *Unexpandability* (N withdrawn coins cannot be expanded into $N + 1$ valid coins). For concreteness, a formal security model is defined here, based on previous work by Franklin and Yung [57]:

Definition 3.1.1 (Secure anonymous (off-line) e-cash) *An off-line electronic cash scheme with security parameter n is secure, if it satisfies the following conditions, the first three of which need only be satisfied with overwhelming probability (in n):*

- (1) **Unreusability:** *If a coin is successfully deposited twice, then the identity of at least one misbehaving user can be efficiently computed and proven from the bank's views of the deposits.*
- (2) **Unforgeability:** *If, from the views of users and shops of arbitrarily many withdrawal, payment and deposit protocols, some probabilistic polynomial-time Turing Machine (p.p.t. TM) can compute a single coin that will lead to two successful purchase (or deposit) protocols, then the fact of reuse and the identity of at least one of these users/shops can be efficiently computed and proven from the bank's views of the respective deposits.*
- (3) **Unexpandability:** *No p.p.t. TM can, from the views of users and shops of N withdrawal and their respective payment protocols, compute $N + 1$ distinct coins that will lead to successful purchases (or deposits).*
- (4) **Untraceability:** *Let p.p.t. TM \mathcal{M} have access to all bank's views of withdrawal, payment and deposit protocols. Then for any two coins C_i, C_j and withdrawals W_0, W_1 , such that C_i, C_j are the coins originating from W_0, W_1 , \mathcal{M} cannot distinguish non-negligibly better (in n) than random guessing whether C_j came from W_0 or W_1 .*

The use of a p.p.t. TM in the definition is to model user collaboration as views to the TM. Thus it is required that no coalition of users/shops can compromise the security of an e-cash system.

Unlinkability: Unlinkability is defined as the property in which given a set of coins one cannot identify two coins that came from the same account. Our untraceability definition guarantees that unlinkability is satisfied in general, as shown in Section 5.3.3. In many previous e-cash systems, untraceability did not exclude coin linkability. Although unlinkability does not add to the theoretical anonymity of a system, it is very important in practice: if a user's coins are *linkable*, identification can be performed by conventional means, such as correlating payments' locality, time, size, type, frequency, or by finding a single payment in which the user identified himself. We refer to Section 5.3.3 for a detailed discussion of unlinkability and how it is

covered by the untraceability theorem.

3.1.2 Efficiency

For implementation of e-cash to be meaningful, efficiency is of utmost importance. However, there may be many ways of determining what renders a cryptographic protocol “efficient,” since there is a great variance of efficiency requirements among different applications. Thus, instead of attempting to formally define efficiency, we treat each system separately, as we would do with any computer program. Nevertheless, there are three basic criteria that can be used to separate efficient protocols from systems of pure theoretical value. These are informally described below:

1. *E-cash based on general computation protocols.* General computation protocols (described in Section 2.2.3) are very powerful in that they allow arbitrary computations, but they are extremely slow to implement. Therefore, following Franklin and Yung [58, 57], we consider no e-cash scheme to be efficient if it utilizes “general computation” protocols. An alternative way of expressing this requirement is the following:

No cryptographic protocol is *efficient*, if its communication complexity is dependent on the computational complexity of the honest participants.

The motivation is that, typically, schemes not based on general computation protocols can be modified to produce practical variants, based on more specialized—but commonly used in practice—assumptions.

2. *Cut-and-choose* systems. The first e-cash systems employed a *cut-and-choose* technique: at withdrawal the user presents $2n$ (where n is the security parameter) “terms”; the bank “cuts-and-chooses” n , for which the user reveals the inner structure. The bank verifies their correctness and blindly signs the remaining n . At payment a similar cut-and-choose technique is employed for the shop to verify a “hint” on the user’s identity, such that upon double-spending two hints identify the user. The cut-and-choose technique is a tool for a zero-knowledge proof of correctness of the coin, thus preserving user anonymity (see Section 2.2.4 for a definition of zero-knowledge proofs). Security is guaranteed

with probability overwhelming in n , but a scheme’s communication, computation and storage requirements are multiplied by a factor of n .

3. *Single-term* systems. Non cut-and-choose schemes are generally called “single-term,” denoting that only one “term” is used per coin, instead of the n terms needed with the cut-and-choose technique. In contrast to cut-and-choose based e-cash, recently proposed single-term systems employ efficient zero-knowledge (ZK) protocols providing the same functionality as cut-and-choose systems with a reduction by a factor of n in storage, computation and communication. This thesis focuses on single-term systems whose efficiency allows immediate practical application.

3.1.3 Protocols

We proceed with providing a short historical view of the advances in electronic cash; this overview excludes enhancements to the basic model such as divisibility or anonymity revocation mechanisms, which will be discussed in later sections. A summary of discussed schemes appears in Table 3.1, in increasing order of efficiency. This section ends with a detailed description of Brands’ electronic cash scheme, which will be used as a building block for several of our results in later chapters.

History

Off-line anonymous electronic cash was introduced by Chaum, Fiat and Naor [32]. Security of their scheme relied on some arbitrary assumptions however and no formal proof was attempted. Although hardly practical, their system demonstrated how off-line e-cash can be constructed and laid the foundation for more secure and efficient schemes to follow. Their methodology was conceptually simple: at withdrawal the bank verifies in a zero-knowledge manner that the user’s identity is “embedded” (encrypted) in a randomly created (by the user) coin, and then provides the user with a blind signature (see Section 2.2.3 for a definition) on this coin; at payment users provide a distinct “hint” on their identity, such that one hint provides a computationally secure and unconditionally blinded commitment on the user’s identity, whereas any two hints identify the user. Hints are verifiable by the shop, i.e., a zero-knowledge proof that the hint corresponds to the identity in the coin is given. At deposit

Scheme	Security	Efficiency
Damgård '90 [41]	Trapdoor one-way permutations (broken)	General-computation protocols, on-line payment
Pfitzmann and Waidner '88 [105]	Trapdoor one-way permutations	General-computation protocols
Franklin and Yung '93 [57]	DLA, trusted party	Cut-and-choose
Chan, Frankel and Tsionis '95 [23]	RSA	Cut-and-choose, pre-processing
Chan, Frankel and Tsionis '96 [25]	RSA, hash functions	Cut-and-choose
Chaum, Fiat and Naor '88 [32]	RSA, hash functions, arbitrary assumptions	Cut-and-choose at withdrawal & payment
Okamoto and Ohta '89 [102]	RSA-based assumption, "multiple blind signatures," hash functions	Cut-and-choose at account-establishment & payment, linkable coins
Cramer and Pedersen '93 [39]	Hash functions, arbitrary assumptions	Single-term
Ferguson '93 [50]	RSA, hash functions, no security proof	Single-term
Brands '93 [15]	DLA, hash functions, one arbitrary assumption. Broken & repaired by [24]	Single-term, fastest to date

Table 3.1: Overview of off-line electronic cash schemes, in increasing order of efficiency.

the shop just transfers a payment transcript to the bank. Upon double-spending a coin the bank identifies the user using the two distinct “hints” on his identity. The zero-knowledge proofs during withdrawal and payment were using the *cut-and-choose* technique. To illustrate the technique we show its application at withdrawal. Let k be the security parameter; i.e., the bank is convinced of the coin’s construction with probability overwhelming in k . The user presents $2k$ “terms,” each of which encodes his identity; the bank “cuts-and-chooses” k , for which the user reveals (decrypts) the inner structure. The bank verifies that they encode the correct identity and blindly signs the remaining k . Clearly, this technique results in excessive communication, computation and storage, since k coins have to be transmitted, stored and verified for each purchase.

Okamoto and Ohta [102] were the first to attempt an improvement on this system. They modified the model by moving the most complex part of the functionality of the withdrawal protocol, namely the zero-knowledge proof of the user’s identity, to the user setup (account establishment) protocol which is executed much less frequently. In the latter the user obtains an untraceable “license” which he uses for withdrawals of coins. Thus, anonymity is established only once, in the form of a pseudonym, instead of being “refreshed” with every withdrawn coin. Hence, a user’s coins are linkable and users may be traced with conventional techniques, i.e., using locality, time, type, size, frequency of payments, or by finding a single payment in which the user identified himself. This was recognized by the authors, who suggested that a compromise between the efficiency and the unlinkability of the system can be found by running the account-establishment protocol more than once. The system relies on more reasonable and clarified assumptions than [32] and, although also based on a cut-and-choose technique, is faster than [32]—at least when the account establishment protocol is performed infrequently; otherwise no improvement is claimed.

At about the same time, Damgård [41] presented a way to base *on-line* electronic cash on general two-party computation protocols and zero-knowledge proofs. The purpose was not to create a practical system but to show that provably secure electronic cash do exist. The scheme had some minor flaws, identified and corrected by Pfitzmann and Waidner in [105], who also showed how to construct a provably secure *off-line* scheme. The latter is provably secure under the general assumption of the *existence* of trapdoor one-way permutations.

To combine the security of Pfitzmann and Waidner’s proposal, with the relative efficiency of regular cut-and-choose systems, Franklin and Yung [57, 58] presented a provably secure scheme that is not based on general computation protocols. Security relied on the DLA and on the existence of a mutually trusted party (equivalently, a general computation protocol could substitute the trusted party’s functionality). Although a cut-and-choose technique was used and efficiency was still not a prime consideration, Franklin and Yung were the first to illustrate how off-line e-cash can be based on the DLA, as well as the first to construct a formal security model; variations of this security model appear in subsequent e-cash systems [23, 25, 100], including all systems in this thesis. Our security model as presented in Section 3.1.1 above is similar to that model, with the main difference being the requirement for unlinkability of coins.

Chaum and Pedersen in [34], attempting to improve the functionality of the basic model, introduced the notion of “wallets with observers”; the model was improved by Cramer and Pedersen in [39]. Both schemes are “single-term,” i.e., they do not rely on inefficient “cut-and-choose” techniques, and they are based on the DLA, but their security relies on several broad and arbitrary assumptions.

Ferguson [50] also presented a “single-term” scheme using a clever combination of RSA signatures to blindly authenticate a coin. Unfortunately, the system’s security remains unproven, while efficiency is inferior to Brands’ scheme [14, 15]. The latter is a “single-term” scheme whose security can be reduced to random oracles and one arbitrary assumption. An extension to “wallets with observers” was also shown requiring minimal additional computation.

Lastly, in a recent collaboration with Chan and Frankel [23] we presented a provably secure off-line e-cash scheme which relies only on the security of RSA. This cut-and-choose based scheme extends the work of Franklin and Yung who aimed to achieve provable security without the use of general computation protocols. Our scheme achieves this goal without the requirement of a trusted party, or of a costly general computation-based initialization procedure. To illustrate the practicality of schemes that are not based on general computation protocols, we then show how to derive an efficient variant based on the random-oracle model; this variant thus achieves provable security based on RSA and the existence of random oracle-like hash functions, while its efficiency is on par with heuristic cut-and-choose based systems,

such as [32, 102].

Brands' e-cash scheme

Brands' [14, 16, 15] scheme is a single-term system for which partial security proofs are provided. This scheme is a refinement of earlier proposals by Brands, Chaum, Cramer, Ferguson and Pedersen [17], Chaum and Pedersen [34] and Cramer and Pedersen [39] and uses the homomorphic properties of discrete logarithms, as previously explored for general applications in [116, 31, 30] and for electronic cash in [57, 58]. It builds on two concepts: Schnorr digital signatures and the Representation problem in groups of prime order (see Sections 2.2.7 and 2.1.2, respectively, for detailed descriptions). The security conforms to our formal model: *Untraceability* is unconditionally guaranteed, while *unexpandability*, *unforgeability* and *unreusability* can be proven under the random oracle model, based on the DLA and one reasonable, albeit arbitrary, assumption.

Recently, however, and in collaboration with Chan, Frankel and MacKenzie [24, 22], *we have shown how to break this scheme, at least in the form presented in [15]*, where efficiency was further improved. We nevertheless also present a modular and efficient addition which secures the scheme against our attacks; the latter is essential since this system is utilized as a building block in several of our results. Our attacks and modular fix are described in detail in Section 5.2.2.

We proceed to describe the scheme in detail, since it is used as a building block for our results in Chapters 5 and 6.

Bank \mathcal{B} 's set up protocol (performed once)

The bank chooses an instance of the DLP collection; informally, primes P and Q are chosen such that $Q|(P-1)$, and generators g, g_1, g_2 of the unique subgroup G_Q (of prime order Q) of the multiplicative group Z_P^* are defined. The bank uniformly selects its secret key $x \in_R Z_Q$, and hash functions $\mathcal{H}, \mathcal{H}_0$, from a collection of pseudorandom, collision intractable hash functions (i.e., $\mathcal{H}, \mathcal{H}_0$ are “random oracles”).

Notation: All arithmetic is performed modulo P , except for the operations involving exponents which are performed modulo Q . The notation here is simplified; for example $h = g^x$ substitutes $h \equiv g^x \pmod{P}$.

\mathcal{B} publishes $P, Q, g, g_1, g_2, h = g^x, \mathcal{H}, \mathcal{H}_0$.

User \mathcal{U} 's setup protocol (performed for each user)

\mathcal{B} associates user \mathcal{U} with $I = g_1^{u_1}$, where $u_1 \in_R Z_Q$ is generated (and only known) by \mathcal{U} , and $g_1^{u_1} g_2 \neq 1$. \mathcal{B} transmits $z = (Ig_2)^x$ to \mathcal{U} (alternatively, as suggested by Brands, \mathcal{B} publishes $h_1 = g_1^x, h_2 = g_2^x$ as part of its public key, and \mathcal{U} can calculate z by himself: $z = h_1^{u_1} h_2$).

Withdrawal (performed over an authenticated channel between \mathcal{B} and \mathcal{U})

The withdrawal protocol creates a “restrictively blind” signature of I . \mathcal{U} will end up with a Schnorr-type [116] signature on $(Ig_2)^s$, where s is a random number (chosen by \mathcal{U} and kept secret). The exact form of the signature is $sign(A, B) = (z', a', b', r')$ with the following two equations being satisfied:

$$g^{r'} = h^{\mathcal{H}(A, B, z', a', b')} a' \quad \text{and} \quad A^{r'} = z'^{\mathcal{H}(A, B, z', a', b')} b' \quad (3.1)$$

Where $A = (Ig_2)^s$ and $B = g_1^{x_1} g_2^{x_2}$, for $s, x_1, x_2 \in_R Z_Q$.

The **withdrawal** protocol appears in Figure 3.2.

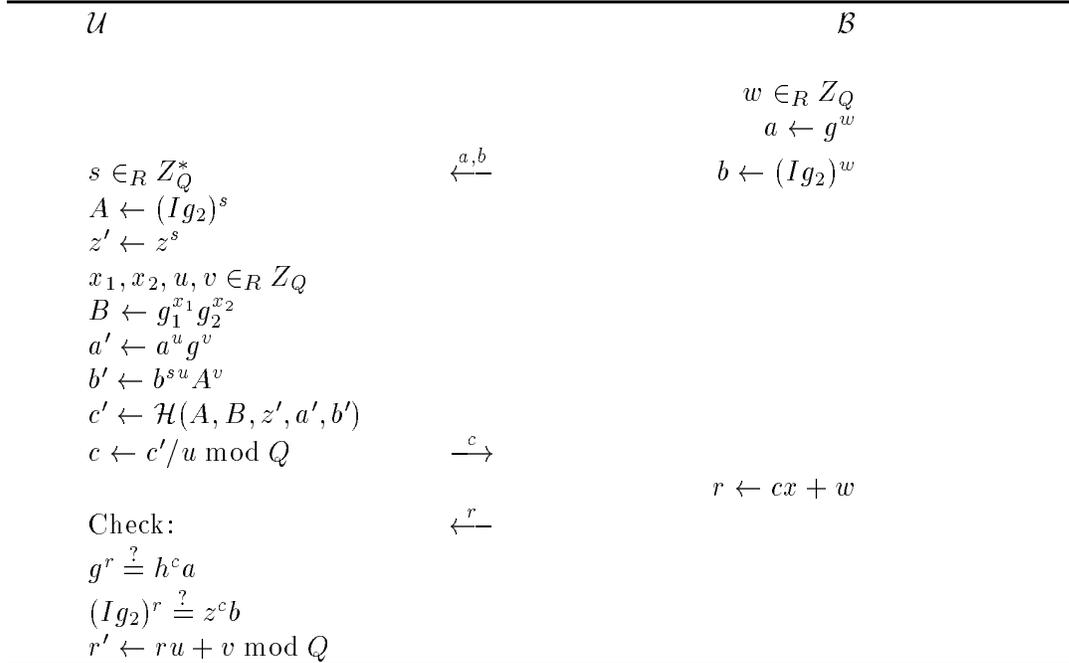


Figure 3.2: The withdrawal protocol of Brands’ scheme, over an authenticated channel between \mathcal{U} and \mathcal{B} .

The idea is for the user to obtain the two Schnorr-type signatures of equation (3.1).

The first is based on the bank’s secret-public key pair (x, h) , ensuring that the bank helped in the signature’s creation—since nobody but the bank knows the secret key x . This “help” of the bank is the solution of the linear equation $r = cx + w$, where g^w is a blinding number that the bank has provided to the user, x is the bank’s secret key, and c is a “question,” which is randomized by the user so that the bank obtains no information about the values used to construct the coin.

The second signature above is based on the coin A . Since the same exponents $(r', \mathcal{H}(A, B, z', a', b'))$ are used in both signatures, this guarantees that as long as the user cannot break the DLA the relative logarithms are equal, i.e., $\log_g h = \log_{z'} A$. Hence the user is indirectly bound in his selection of A, z' , as this pair must be constructed using the same w -based “hint”, $r = cx + w$. Since the only w -based “hint” given to the user is $b = (Ig_2)^w$, the construction seems to guarantee (although this has yet to be proven) that A, z' are represented in terms of (Ig_2) , i.e., $A = (Ig_2)^s, z' = A^x$, for some random blinding factor s chosen by the user.

Payment (performed between \mathcal{U} and \mathcal{S} over an anonymous channel)

At payment time \mathcal{U} supplies to the shop \mathcal{S} a point (r_1, r_2) on a “line”, with (r_1, r_2) being determined by his identity $(I = g_1^{u_1})$ and by a challenge (d) from \mathcal{S} :

$$r_1 = d(u_1s) + x_1, \quad r_2 = ds + x_2$$

Two such points (based on different challenges d, d') will reveal his identity, by computing $g_1^{\frac{du_1s - d'u_1s}{ds - d's}} = g_1^{u_1} = I$. Hence, at deposit time, double-spenders are identified.

The **payment** protocol appears in Figure 3.3.

Deposit: The shop deposits the coin by providing the bank with a transcript of the payment. If the user \mathcal{U} associated with $I = g_1^{u_1}$ double-spends, the bank obtains two transcripts (d, r_1, r_2) and (d', r'_1, r'_2) and the relation

$$g_1^{(r_1 - r'_1)/(r_2 - r'_2)} = I$$

holds, hence the user is identified.

3.2 Allowing exact payments

Efficiency for complete payment systems requires extensions to the basic model due to the need for providing the user with the ability to conduct payments of exact amounts.

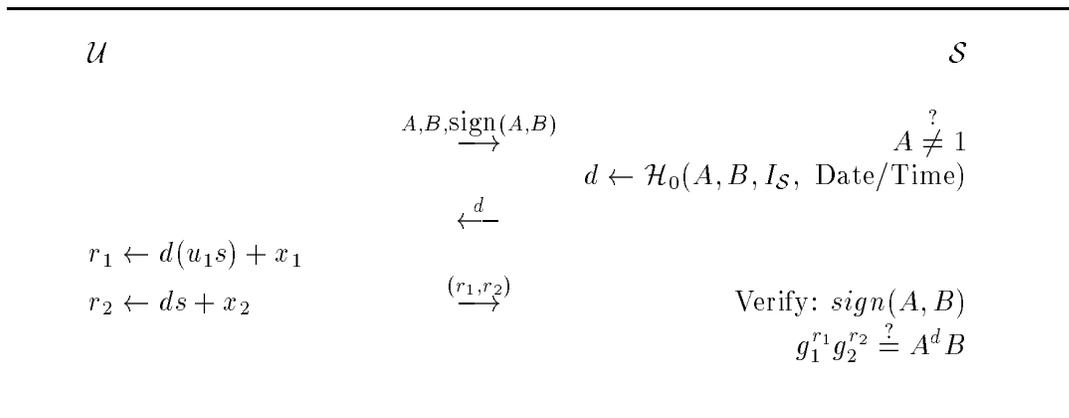


Figure 3.3: The payment protocol of Brands’ scheme, over an anonymous channel between \mathcal{U} and \mathcal{S} .

A basic e-cash system is not practical: computation, communication and storage requirements for exact payments are excessive, since users must keep a multitude of coins—similar to the way multiple coins are needed for exact physical payments; on the other hand change-making simply transfers the problem of keeping exact change from users to shops. In addition, anonymous off-line e-cash is unable to provide “change” in a payment transaction, as the user would have to identify himself to the bank when depositing the change, allowing traceability of his original payment. The only solution would be for the user to contact the bank on-line asking for exact change, as was proposed by Brickell et. al. [19]; however this would defeat the purpose, turning off-line e-cash into on-line.

Motivated by this rationale, Okamoto and Ohta [103] proposed an electronic cash system that is divisible: electronic coins can be divided and paid in any denomination up to their total value. If a user *over-spends*, i.e., if he spends an amount higher than the coin’s total value, he is identified with overwhelming probability—as if he were double-spending an indivisible electronic coin.

Divisible e-cash is modeled exactly as basic e-cash, i.e., a bank, \mathcal{B} , users, \mathcal{U} , and shops, \mathcal{S} , participate in three procedures, *withdrawal*, *payment* and *deposit*. For clarity of description, however, the payment protocol is usually divided into two parts:

- *coin authentication*, where the shop verifies the bank’s signature on a coin, and
- *denomination revelation*, where the user reveals enough information about the spent part of the coin to enable identification upon over-spending.

Both parts exist in the basic model as well but the denomination revelation is far more complex in a divisible system.

Security is defined as in the basic model with provisions taken to account for the fact that there are several payments originating from one withdrawn coin. Here we stress that the portions of a divisible coin are linkable, similar to the way coins in Okamoto and Ohta’s paper [102] are linkable, therefore the untraceability requirement of our definition in Section 3.1.1 is not fully satisfied; this may allow identification of users by means outside of cryptography, e.g., by correlating payments’ time, locality, type, especially when a multitude of payments are made with this coin. Linkability is handled by relaxing the security model for divisible e-cash; to this effect a coin is defined as being the collection of all its “portions,” thus allowing linkability among portions. As this is the only difference from the basic model we refrain from rewriting the security definition; instead we refer the reader to Section 5.3.3 where the four security requirements of the basic model are adapted for divisibility.

The key parameter in divisible e-cash is the *divisibility precision*, \mathcal{N} , defined as $\mathcal{N} = (\text{total coin value})/(\text{smallest divisible unit})$. A divisible coin of value $\$x$ can be spent in several increments of the smallest divisible unit, $\$m$, up to the total amount $\$x$. Clearly, for a basic e-cash scheme to emulate the full functionality of a divisible coin \mathcal{N} parallel protocols would have to be conducted, in each of which a single coin of value equal to m would be withdrawn. In addition, \mathcal{N} distinct payment protocols would be needed for payment of the whole amount. Hence, both the withdrawal and payment would have complexity *linear to the divisibility precision*.

Recently, however, we have proposed an alternative way to provide exact payments [54], by relaxing the system’s requirements: the system now guarantees that k exact payments can be made, where $k \leq \mathcal{N}$. This approach can be realistic if \mathcal{N} and k are chosen appropriately.

Thus we have shown that *there are two possible approaches in providing exact payments in an electronic cash system*:

- (1) either withdraw a divisible coin allowing practically unlimited divisibility precision and number of payments, or
- (2) withdraw a multitude of non-divisible coins that allow k exact payments with divisibility precision \mathcal{N} .

Note that the non-divisible coins withdrawn can be allowed to be linkable,

similar to the way a divisible coin’s portions are linkable.

Although withdrawing several non-divisible coins instead of a single divisible one may sound like an expensive alternative, there is in fact a threshold, depending on k and \mathcal{N} , below which keeping multiple coins is preferable: handling a small collection of non-divisible coins proves to be more efficient than withdrawing and paying one divisible coin, due to the complex structure needed for the latter. An additional advantage to this approach is that it can emulate a divisible coin whose portions are unlinkable, by simply requiring that the withdrawn non-divisible coins are unlinkable. In line with intuition, obtaining unlinkable coins requires much more computation and storage than linkable ones; however this approach represents the first result in *unlinkable divisible e-cash*, i.e., divisible e-cash with unlinkable coin portions, and serves as a lower bound for the efficiency requirements of any unlinkable divisible e-cash scheme.

In this thesis we investigate both ways of providing exact payments; on one hand we construct a very efficient divisible e-cash scheme, and on the other we analyze the exact complexity requirements of a system allowing k exact payments. In this background section, however, we only describe existing divisible schemes; our results in analyzing the complexity of exact payments and in constructing an efficient divisible scheme are presented in Chapters 4 and 5 respectively.

Next we present the binary tree approach, which is fundamental in designing divisible electronic cash schemes.

3.2.1 The binary tree approach

In all divisible e-cash systems coins are represented by a binary tree, as first suggested by Okamoto and Ohta in [103]. The tree is used in an intuitive sense: each node of the tree represents an amount, with the root representing the whole amount, $\$x$, its children half of the amount, its “grand-children” a quarter of the amount and so on. This approach limits the size of the coin by allowing the bank to authenticate only the data necessary to verify the tree’s construction, typically only the root of the tree. Additionally, the coin can be divided in arbitrarily small pieces, depending on the tree’s depth, whereas to pay an amount a only $O(\log(a))$ nodes need to be “spent”; in particular, given the binary representation of a , one node of the coin is spent for each non-zero bit.

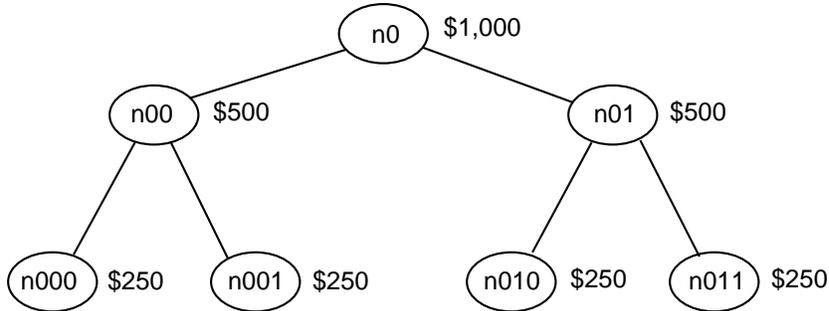


Figure 3.4: Binary tree representation for a \$1,000 coin.

Figure 3.4 shows an example of such a tree for a coin of value \$1,000. The root node is valued to the whole amount, $n_0 = \$1,000$, while $n_{00} = n_{01} = \$500$ and $n_{000} = n_{001} = n_{010} = n_{011} = \250 .

For a tree construction to work properly, the following two rules must be satisfied:

- **Root route rule:** Once spent, a node’s ancestors and descendants cannot be used, i.e., all routes to the root should contain only one spent node.
- **Same node rule:** A node cannot be used more than once.

These two node rules guarantee that a user cannot *over-spend*, i.e., spend more than the total value of the coin. In the context of off-line electronic cash, “cannot spend” means that if these nodes are spent then the user’s identity is revealed.

3.2.2 Protocols

We now overview the results in the area of divisible electronic cash; an outline appears in Table 3.2. Two schemes, due to Eng and Okamoto [46] and Okamoto [100] are described in more detail, as they represent the most complete results in the area. Our work is related to both, with Okamoto’s scheme being used as an underlying block for our result in Chapter 5.

Divisible off-line e-cash was introduced by Okamoto and Ohta in [103] with a scheme based on a cut-and-choose technique, hence of limited efficiency. Security was based on an RSA-related assumption, on hash functions that behave like random oracles and on a “multiple blind signature” assumption.

Scheme	Security	Efficiency
D'Amiungo and Di Crescenzo '94 [42]	One-way functions	General computation protocols
Okamoto and Ohta '91 [103]	RSA-based assumption, "multiple blind signatures," hash functions	Cut-and-choose
Pailles '92 [104]	Schnorr signatures, hash functions, no proof given	Cut-and-choose
Eng and Okamoto '94 [46]	Hash functions, Brands' assumption. Broken and repaired by [24]	Single-term, $O(\mathcal{N})$ at payment
Okamoto '95 [100]	RSA, DLA, hash functions, novel assumption	Single-term, linkable coins
Frankel, Patt-Shamir and Tsiounis '96 [54]*	Hash functions, Brands' assumption	Single term, dependent on number of payments. Unlinkable coin portions
Chan, Frankel and Tsiounis '96 [26]*	Hash functions, Brands' assumption, a variant of [100]'s assumption	Single term

Table 3.2: Overview of divisible off-line electronic cash schemes, in increasing order of efficiency. Results appearing in this thesis (marked with a *) are included for comparison.

Pailles in [104] improved the efficiency of the original scheme, although the cut-and-choose technique was still applied. Security was heuristically based on DLA and hash functions, but this scheme was the first to hint that divisible e-cash can be based on the DLA.

D’Amiano and Di Crescenzo [42] showed how to construct provably secure divisible cash, based on general computation protocols and zero-knowledge proofs. The security of the system is based on the existence of one-way permutations. Unfortunately, the scheme is impractical, since it is based on such general constructions.

Eng and T. Okamoto’s scheme

Eng and T. Okamoto [46] showed how to use Brands’ [15] withdrawal protocol to withdraw divisible coins. The idea rooted on Pailles’ denomination revelation protocol which was based on the DLA—similar to Brands’ scheme. Although efficiency is significantly improved, both withdrawal and payment require time linear to the divisibility precision \mathcal{N} . Security depends on the same assumptions as the Brands’ scheme. Due to the similarity of designs, this scheme is also broken by our recent results [24, 22]; however, our addition to the Brands’ scheme, presented in Section 5.2.2, suffices to repair it.

In the scheme, a coin, represented as a binary tree, is withdrawn using Brands’ withdrawal protocol. At payment the user supplies the shop with information related to the nodes of the tree that are being spent. The idea is for the user to pick some random strings that correspond to the leaves of the tree. From these strings the user constructs (level-by-level) a string that is related to the root of the tree, in such a way that each “parent” string is a secure commitment on its “children’s” strings; hence, once the root’s string is constructed the whole tree cannot be altered by the user. Additionally, revealing a node’s string does not reveal the strings of its children, due to the properties of bit commitment schemes.

The setup and withdrawal protocols of this scheme are similar to Brands’ [15] but the string that is now being signed by the bank is the one related to the root of the tree. At payment, coin authentication, i.e., authentication of the tree, entails verifying the bank’s signature on the root of the tree. During the denomination revelation a verification identical to the one in Brands’ payment is performed for each spent node, with the string of each node substituting for the value B (see Section 3.1.3). In

addition, \mathcal{U} proves to \mathcal{S} that the nodes being spent are correctly related to the signed coin. To this effect the user first proves that a node’s string is committed in its parents’ string by opening the latter’s commitment. This process continues step-by-step until the root’s commitment is revealed; the shop and bank can now check the correctness of the original node, since the bank’s signature on the root’s string has already been verified.

If a user double-spends a node, i.e., if he breaks the same node rule, he is identified by the construction of Brands’ scheme. Whereas violating the root rule is equivalent to spending two nodes on the same path to the root; in this case the commitment for the node higher in the tree is opened, since spending the “lower” node entails opening the commitments of all his ancestors. The opening of the ancestor’s commitment reveals the relation between the strings of the two nodes. Since payment entails presentation of a linear equation based on each spent node’s string, the revelation of the relation between the two strings allows the bank to reduce its number of unknown variables and solve the system of linear equations, thus identifying the user; this idea originated in Pailles scheme [104].

The result is a system with “single-term” coins, i.e., no cut-and-choose technique is used. However, at withdrawal time the user must construct the tree “bottom-up”; hence the divisibility precision is determined and set at withdrawal time and, most importantly, the user’s computation is on the order of the divisibility precision, $O(\mathcal{N})$. Similarly, at payment time the user must either reconstruct the nodes to be spent from scratch or store them in memory; hence either the computation or the storage requirements are in $O(\mathcal{N})$. The authors in fact suggest a compromise between the number of the nodes stored and the computation at payment; however both storage and computation requirements remain a few orders of magnitude higher than those of a non-divisible scheme. Additionally, the user must “link” each node to the root of the tree at payment, hence the communication per spent node is $O(\log(\mathcal{N}))$. The system is nevertheless the first “single-term” proposal for divisible cash and the most efficient to that date.

T. Okamoto’s scheme

Recently, T. Okamoto [100] presented the first divisible e-cash scheme where the protocols were only logarithmically dependent on the divisibility precision. The scheme

uses the model of the non-divisible scheme proposed by Okamoto and Ohta [102], where the functionality of the withdrawal is split into two parts: establishing the unlinkability between coins with a zero-knowledge proof that the coin encrypts the user’s identity, and withdrawing a coin by obtaining a blind signature on the coin concatenated with a random “serial number”. The first and most expensive part, i.e., the zero-knowledge proof, is then moved to the account establishment, meaning that all withdrawn coins are in fact the same basic coin with different serial numbers. Thus all the coins of a user are linkable, unless the account establishment is executed more than once. All protocols of this scheme, *except establishing an account*, are of comparable efficiency with the most efficient *non-divisible* off-line e-cash systems available. Since the account establishment is inefficient, however, there is a tradeoff between unlinkability and efficiency. The scheme’s security is based on RSA, the DLA, random oracle-like hash functions, and two novel number-theoretic assumptions.

Recently, *we have identified a flaw on this scheme* [22] which allows users to overspend undetected. Our attacks, and ways to guard against them, are described in detail in Section 5.2.3.

In Okamoto’s scheme, each user \mathcal{U} has a composite number $N = pq$, such that N is a Williams integer (i.e., p, q are primes with $p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}$) associated with \mathcal{U} .

In **the account establishment protocol** the bank \mathcal{B} publishes its RSA public keys (n_1, K) and (n_2, K) and also (a_1, a_2) as public keys. It also publishes a prime P and a generator $g \in Z_P^*$ of the subgroup G_Q of prime order $Q = (P - 1)/2$. The user \mathcal{U} then selects random primes p, q with $p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}, |p| = |q|, |P| \geq 4|p| + 10$ and shows $g^p, g^q \pmod{P}$ to the bank. The bank then performs a blind signature on $N = pq$ to provide \mathcal{U} with an electronic license, after *the user proves that N is the product of p and q* in a zero-knowledge way (note here that the primality of p, q is not tested). In the process the number N , which uniquely identifies a coin, is not revealed; that is the bank verifies the validity of the coin N of user \mathcal{U} without seeing the coin. This is necessary for user anonymity: since N is shown at payment, all the bank would need to do to trace users is access the account establishment database, where the link $N \rightarrow \mathcal{U}$ would appear. This electronic license (the bank’s signature on N) is then used as a pseudonym by \mathcal{U} at payment; although the bank can trace the coins back to this pseudonym, the user remains anonymous.

In [100] Okamoto shows that this protocol takes approximately 4000 “multi-exponentiations” modulo P , where a multi-exponentiation is an operation of the type $g_1^a g_2^b \pmod{P}$. The estimates assume 256 bit primes p and q , i.e., an RSA modulus of 512 bits, and a security parameter $k = 20$. The security parameter controls the probability that a user can break the system, i.e., in the case of e-cash the probability that he can double-spend undetected. In particular, the system is “tuned” so that the probability of cheating undetected is $1/2^k$, or roughly one in a million for $k = 20$. **Withdrawal** of the coin is nothing more than an RSA blind signature [27] on N concatenated with a random number, b , guaranteeing coin uniqueness. The bank signs a hash function of this string ($\mathcal{H}(N, b)$) to prevent the user from forging the signature; here, as in the rest of this system, the hash function is assumed to behave like a random oracle (see Section 2.2.6 for a definition and possible emulations of random oracles). The bank’s public RSA key is dependent on the value of the coin. The coin, i.e., the composite N , defines a tree such that the *root route rule* and the *same node rule* are satisfied.

The payment protocol, is presented in two parts for clearer understanding: coin authentication and denomination revelation.

At **coin authentication**, \mathcal{U} convinces \mathcal{S} that the coin is legitimate. To this effect the shop verifies

- (1) the correctness of the electronic license,
- (2) the bank’s RSA signature on the coin, and that
- (3) if all the prime factors of N are congruent to $3 \pmod{4}$, then N has an even number of prime factors, an odd number of which are congruent to $3 \pmod{8}$, while the rest are congruent to $7 \pmod{8}$; furthermore, not all of its factors are congruent to either $3 \pmod{8}$ or $7 \pmod{8}$.

This latter property is checked by verifying the Jacobi symbols $(-1/N) = 1$ and $(2/N) = -1$ (we refer the reader to Section 2.1.1 for the definition of Legendre and Jacobi symbols, and a discussion of properties of Williams integers).

The reader should note that the same N and electronic license are revealed for each coin. Hence, coins can be linked. In Chapter 5 we present a system that does not have this property.

At **denomination revelation**, \mathcal{U} presents some data that are specific to the node(s) of the tree that is/are being spent, in such a way that \mathcal{S} is guaranteed that

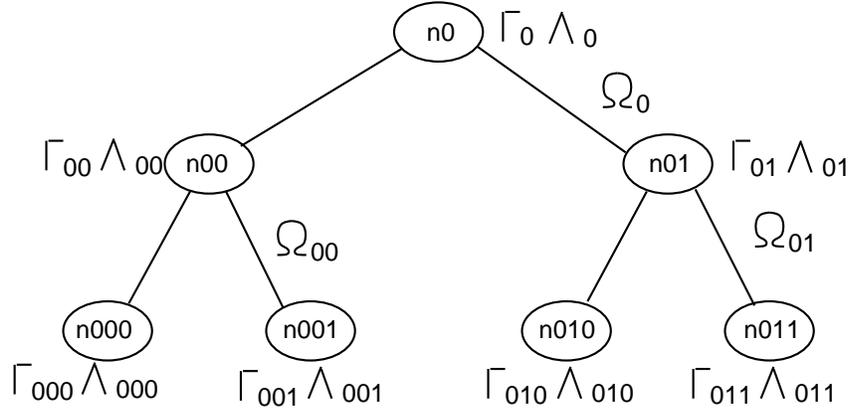


Figure 3.5: Use of Γ, Ω and Λ values in the binary tree.

- (1) N has only two distinct prime factors, and
- (2) if \mathcal{U} violates the root route or the same node rule then N can be factored.

Protocol details:

We now proceed with the details of the denomination revelation protocol, as this, together with step (3) of coin authentication, is used for our result presented in Chapter 5.

Assume that the payment is for an amount X and let $[x_1, x_2, \dots, x_{l+1}]$, $x_i \in \{0, 1\}$ for all $i \in \{1, \dots, n\}$, be the binary representation of X (allowing some most significant bits to be 0), where l is the system's divisibility precision and $2^l V$ is the total value of the coin, i.e., V is the smallest payable fraction. Then if $x_t = 1$ for some $1 \leq t \leq l+1$, \mathcal{U} selects a node $n_{j_1 j_2 \dots j_t}$ among the nodes in the t -th level of the binary tree (see Figure 3.4 in Section 3.2.1) that does not violate the two node rules. The selection is random, so as not to give \mathcal{S} any additional information about \mathcal{U} 's payment history. On the average, $(l+1)/2$ nodes are spent per payment, since, on the average, the binary representation of X contains $(l+1)/2$ bits that are set to 1.

We present the denomination revelation for node $n_{j_1 j_2 \dots j_t}$. To help in understanding the complex protocol a specific example is included at the end of this section, in which node n_{010} is spent. If more than one nodes are spent their payments can be conducted simultaneously. The revelation consists of two parts, one for each of the node rules. Three types of values are introduced, Γ, Ω and Λ ; these are used for the

realization of the root route (Γ, Ω) and same node (Λ) rules (see Figure 3.5).

Notation: For each $a \in Z_N^*$, $\langle a \rangle_{QR}$ denotes the element in $\{a, -a, 2a, -2a\}$ which is in QR_N . $\langle a \rangle_1$ is the element in $\{a, 2a\}$ which has Jacobi symbol of 1. $[a^{1/2}]_{-1}$ is the square root y of a , for which $(y/N) = -1$ and $0 < y < N/2$. $[a^{1/2}]_1$ is similarly defined. $\mathcal{H}_?, \mathcal{H}_\Omega, \mathcal{H}_\Lambda$ are hash functions (assumed to behave like random oracles) mapping their input to Z_N^* . \mathcal{H}_0 is a random-oracle-like hash function mapping to $\{0, 1\}^k$, where $k = O(|p|)$. Note that the security parameter for the payment protocol is $k = 40$.

Now we are ready to describe the denomination revelation protocol. Assume that node $n_{j_1 j_2 \dots j_t}$ of coin $C = \mathcal{H}(N, b)$ is being spent. The following takes place:

1. **(Realizing the root route rule)** User \mathcal{U} computes $\Gamma_{j_1 \dots j_t}$ and sends to shop \mathcal{S} :

$$\Gamma_{j_1 \dots j_t} = [(\langle (\Omega_{j_1 \dots j_{t-1}})^{2^{t-1} j_t} (\Omega_{j_1 \dots j_{t-2}})^{2^{t-2} j_{t-1}} \dots (\Omega_{j_1})^{2 j_2} \mathcal{H}_?(C, 0, N) \rangle_{QR})^{1/2^t} \bmod N]_{-1},$$

where $\Omega_{j_1 \dots j_i} = \langle \mathcal{H}_\Omega(C, j_1, \dots, j_i, N) \rangle_1, i \in \{1, \dots, t-1\}$.

2. \mathcal{S} computes $\Omega_{j_1 \dots j_i}$ when $j_{i+1} = 1$, for $i \in \{1, \dots, t-1\}$. Then \mathcal{S} verifies the validity of $\Gamma_{j_1 \dots j_t}$, by checking that:

$$\begin{aligned} &(\Gamma_{j_1 \dots j_t} / N) = -1, \text{ and} \\ &(\Gamma_{j_1 \dots j_t})^{2^t} \equiv d (\Omega_{j_1 \dots j_{t-1}})^{2^{t-1} j_t} (\Omega_{j_1 \dots j_{t-2}})^{2^{t-2} j_{t-1}} \dots (\Omega_{j_1})^{2 j_2} \mathcal{H}_?(C, 0, N) \\ & \pmod{N}, \end{aligned}$$

for some $d \in \{\pm 1, \pm 2\}$.

3. **(Realizing the same node rule)** \mathcal{U} computes $e = \mathcal{H}_0(C, \text{DATE/TIME}, ID_{\mathcal{S}}, N)$, where $e \in \{0, 1\}^k$. The timing and shop ($ID_{\mathcal{S}}$) information is necessary for non-repudiation: if two instances of the coin are spent with this information being identical then the shop is at fault; else e is different in each instance and the user is identified. For increased security against collision attacks on the hash function, this challenge could also be computed in an interactive way, by setting $e = \mathcal{H}_0(C, \text{DATE/TIME}, ID_{\mathcal{S}}, e', N)$, where $e' \in_R \{0, 1\}^k$ is a random value supplied by \mathcal{S} .

\mathcal{U} then computes $\Lambda_{j_1 \dots j_t}$ such that:

$$(\Lambda_{j_1 \dots j_t})^{2^{k+1}} \equiv 2^{2e} \langle \mathcal{H}_\Lambda(C, j_1, \dots, j_t, N) \rangle_{QR} \pmod{N}$$

4. \mathcal{S} verifies the validity of $\Lambda_{j_1 \dots j_t}$ by checking that:

$$(\Lambda_{j_1 \dots j_t})^{2^{k+1}} \equiv d' 2^{2e} \mathcal{H}_\Lambda(A, B, j_1, \dots, j_t) \pmod{N},$$

for some $d' \in \{\pm 1, \pm 2\}$.

5. **(Verifying that N is a Williams integer)** The steps 1 and 3 above, together with coin authentication, help guarantee that N is a Williams integer.

In particular,

- (1) step 3 guarantees that all of N 's prime factors are congruent to 3 mod 4 with probability $1 - \frac{1}{2^k}$;
- (2) coin authentication then guarantees that N has an even number of prime factors, an odd number of which is congruent to 3 mod 8 while the rest are congruent to 7 mod 8, while not all are congruent to either 3 mod 8 or 7 mod 8; and
- (3) each of step 1 and 3 guarantee that N has only two distinct prime factors with probability 3/4.

Hence, $k/4$ node payments, i.e., executions of steps 1 and 3, are needed in order to assure that N has only two distinct prime factors with an overwhelming probability $1 - 1/2^k$, with each payment allowing a 1/16 probability of success to a cheating user. Hence, if only $m < k/4$ nodes have been spent, \mathcal{S} sends $z_1, \dots, z_{2k'}$ to \mathcal{U} , for $k' = k/4 - m$ and $z_i \in_R Z_N^*$, and \mathcal{U} computes $[(\langle \mathcal{H}_?(z_1) \rangle_{QR})^{1/2} \pmod{N}]_1, \dots, [(\langle \mathcal{H}_?(z_{2k'}) \rangle_{QR})^{1/2} \pmod{N}]_1$. These square root computations are then sent to \mathcal{S} who checks their validity. Each of these operations guarantees again with probability 3/4 that N is a product of two distinct primes.

We encourage the user to read the paragraph which describes our attacks to the above system, included in Section 5.2.3. There the security of the above protocol is analyzed in detail.

Lastly, at **deposit**, \mathcal{S} sends the payment transcript to \mathcal{B} .

Example: For the sake of clarity, we show how the denomination revelation is performed for a specific node, n_{010} . For this node, $t = 3$.

1. **(Realizing the root route rule)** \mathcal{U} computes Γ_{010} :

$$\Gamma_{010} \equiv [(\langle (\Omega_0)^2 \mathcal{H}_? (C, 0, N) \rangle_{QR}^{1/8} \pmod N)_{-1}] ,$$

where $\Omega_0 = \langle \mathcal{H}_\Omega (C, 0, N) \rangle_1$.

2. \mathcal{S} computes Ω_0 and verifies the validity of Γ_{010} , by checking that:

$$\begin{aligned} (\Gamma_{010}/N) &= -1 , \text{ and} \\ (\Gamma_{010})^8 &= d(\Omega_0)^2 \mathcal{H}_? (C, 0, N) \pmod N , \end{aligned}$$

for some $d \in \{\pm 1, \pm 2\}$.

3. **(Realizing the same node rule)** For this part, let us assume that $k = |p| = |q| = 128$. \mathcal{S} sends e to \mathcal{U} .
4. \mathcal{U} verifies e and computes Λ_{010} such that:

$$(\Lambda_{010})^{2^{129}} = 2^{2e} \langle \mathcal{H}_\Lambda (C, 010, N) \rangle_{QR} \pmod N$$

5. \mathcal{S} verifies the validity of Λ_{010} by checking that:

$$(\Lambda_{010})^{2^{129}} = d' 2^{2e} \mathcal{H}_\Lambda (C, 010, N) \pmod N ,$$

for some $d' \in \{\pm 1, \pm 2\}$.

3.3 Anonymity concerns (Fair electronic cash)

We have stressed in the previous sections that the distinguishing feature of electronic cash from other payment systems is the anonymity offered to users. Anonymity, however, can be a cause of major concern to governments. As pointed out by [129, 19, 123, 21], e-cash can be used for criminal activities, such as money laundering or perfect, i.e., anonymous, blackmailing. Similarly, if e-cash is to be used for payment of cellular services user anonymity may provide anonymous communication

for criminal activities. Hence any large scale application of e-cash may be thwarted by a government unless there is a mechanism to prevent such criminal activities—by removing user anonymity. This becomes clearer if we examine current practices towards commerce and telephony: governments require each bank to be able to trace every transaction above a certain amount, e.g., on the order of \$3,000 for U.S.-based banks, while certain laws, such the telephony bill in the U.S., require telephone companies to trace phone calls when requested by the authorities. These issues are also discussed in a recent NSA report by Law Sabett and Solinas [88]. An interesting overview of anonymity issues with emphasis on the legal perspective is given by Froomkin in [60].

Solutions to this problem have been proposed by Brickell, Gemmell and Kravitz [19] as well as Stadler, Piveteau and Camenisch [123, 21], proposing to allow an “escrow agent,” such as a judge, to remove anonymity of users when necessary. Hence the basic model is extended to include a judge, or *Trustee*, who is so called since it is trusted by users to revoke their anonymity only when necessary. It must be emphasized here that the power of the Trustee can be distributed; hence the Trustee can be in practice comprised of several parties, such as a representative of a customer rights organization, a police representative, a judge, or a governmental party. Using standard secret sharing techniques any combination of these parties can be allowed to revoke the anonymity of specific transactions/users. In normal conditions the system operates as anonymous e-cash; however, upon the order of the trustee, the anonymity of a particular user may be revoked. Similar anonymity or privacy-revoking mechanisms have been recently proposed for encryption and signature schemes as well [123, 35, 56]. Such systems are in general called “fair,” following the terminology of [92], since they balance user anonymity and the need of governments to revoke that anonymity.

In all previously proposed escrowed e-cash systems the Trustee must be involved in the withdrawal protocol; in particular, if we view the withdrawal protocol as two distinct procedures, as in [102, 103, 100], one for establishing the anonymity of the user by providing an anonymous “electronic license,” and one for performing a withdrawal based on that anonymous license, the Trustee must be involved in the first part. In addition, the Trustee must store and manage sensitive data for each withdrawal protocol. Hence, only electronic cash protocols in which user coins are *linkable*, i.e., the anonymity of the user is established one time only—at account establishment—can

be efficiently implemented using these systems: otherwise the Trustee would have to be involved in every coin withdrawal. Even in linkable e-cash, however, involvement of the Trustees is required during account establishment. This of course is an undesirable property and prevents the systems from being used in practice.

In this thesis we propose a new approach of *Fair Off-Line electronic Cash (FOLC)*, in which the Trustee is not involved in the basic e-cash system and is only queried when tracing a user's identity is requested. FOLC is described in detail in Chapter 6, where the formal model and our relevant results are presented. We must note that Camenisch, Maurer and Stadler independently proposed a similar system in [20]; we refer the reader to Chapter 6 for a comparison of this with FOLC.

3.4 Further extensions

There have been a number of proposals aiming at extending the functionality and/or security of off-line electronic cash. Divisible electronic cash and fair electronic cash can be seen as such extensions. Here we present two other proposed additions to the basic model, namely tamper-resistant devices which enhance the system's security and transferability which enhances the system's functionality.

3.4.1 Tamper-resistant hardware (“wallets with observers”)

It is easy to prove that in a purely algorithmic e-cash scheme where no tamper-resistant devices are used it is infeasible to prevent a user from double-spending a coin: a user can just backup his computer's full state before a payment, in effect “tricking” the computer to believe that the coin is unspent; then he can reload the backed-up state after the payment and start a new payment with a new shop; in a purely *off-line* system this double-spending can be *detected* but not *prevented*. In order to prevent double-spending tamper-resistant hardware is necessary; this would prevent a user from reading the state of his computer.

The model for electronic cash with tamper-resistant devices is slightly different from that of simple off-line e-cash. An “observer,” in the form of a tamper-resistant device supplied by the bank to each user, is added to the model (see Figure 3.6).

The user acts as an intermediary between the observer and the bank and the protocols are constructed in such a way that

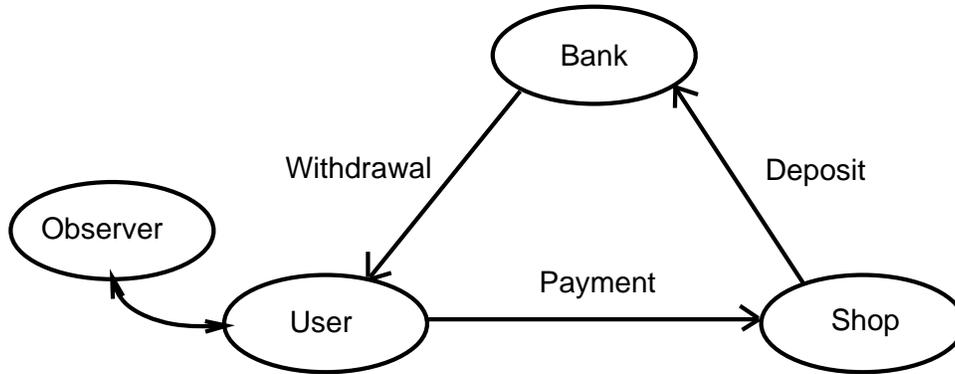


Figure 3.6: Model of “wallets with observers”.

- (1) the user cannot spend a coin without the observer’s consent,
- (2) the observer does not allow the user to over-spend, and
- (3) the user blinds all information going from the observer to the bank and vice-versa.

The third requirement guarantees that even if the bank obtains the observer, for example for maintenance purposes or even by theft, it is still not possible to trace the user’s payments. This requirement is the *only* difference between a regular anonymous off-line e-cash system and a system with observers. The addition of an observer has prompted the name used for this setting: “wallets with observers”.

There have been several efforts in adding provisions for tamper-resistant devices in off-line electronic cash [15, 14, 39] after the introduction of the concept by Chaum and Pedersen [34]. The most recent effort which resulted in the simpler and faster scheme is by Brands [15].

Although adding another layer of protection, utilization of tamper-resistant devices is nevertheless not a panacea for prevention of double-spending, since these devices are not really “tamper-proof”; their contents can always be read by a determined adversary with the right technology. Furthermore, off-line e-cash can be operated in an on/off-line manner, such that large transactions are verified on-line and small ones off-line in such a way as to limit the amount a user can double-spend. Thus the advantage of wallets with observers is questionable, on the grounds of the

increased cost of providing tamper-resistant devices and their limited security advantages over regular e-cash.

3.4.2 Transferability

Transferability is a feature that exists in physical cash but has not been applied in electronic cash. The main reason is the danger involved in transferring electronic coins and the subsequent increase in liability for the banks issuing those coins: an over-spending user is only identified when the coins he spent return to the bank; in addition, a single coin can be over-spent by all users from whose hands it has passed. Although these risks can be minimized via the use of tamper-resistant devices or by applying an on/off-line setting for coin spending, it seems that transferable electronic cash will only proliferate, if at all, after the establishment of non-transferable off-line e-cash.

The lack of interest in pursuing research in transferability can also be attributed to the facts that (1) depositing an electronic coin is much more efficient than depositing a physical token, hence transferability becomes less of an issue, and (2) a coin grows in size with every transfer, as shown by Chaum and Pedersen in [33]. This last result is intuitive, since a transferred coin must embed the identity of all users that once owned it so that double-spenders can be identified.

Finally, we must point out that there is a general, although inefficient, algorithm with which transferability can be added to any off-line e-cash system, due to van Antwerpen [126], which has been shown to be asymptotically optimal by Chaum and Pedersen [33].

3.5 Discussion and open problems

The fruit of continuous research in the past few years, current electronic cash systems have evolved to the point where practical applications are now feasible. In basic systems where exact payments are not needed, e.g., some toll-booths or subway systems, an efficient basic off-line scheme can be directly applied.

On the other hand, for more general applications a number of issues remain unsolved, of which the most prominent is providing exact payments. This thesis addresses this topic by examining how any basic e-cash scheme can be used optimally

for this purpose (in Chapter 4), thus achieving *unlinkable divisible e-cash*, as well as providing new results in divisible electronic cash (in Chapter 5). However, using a non-divisible e-cash scheme for exact payments limits the divisibility precision and the number of exact payments, whereas divisible coins allow linkability of payments that originate from the same coin. It remains an open problem whether a divisible e-cash scheme with practically unlimited divisibility and number of exact payments in which portions of the same coin are *unlinkable* exists and can be constructed.

Providing an efficient way of tracing fraudulent transactions, i.e., implementing fair off-line e-cash, has been another open problem for the past years. Recently, however, two independent results, by Camenisch, Maurer, Stadler [20] and the system presented in Chapter 6 in this thesis (parts of which have been published in [55] and [43]) have addressed this issue without significant performance penalties. In addition, fair *divisible* cash is a straightforward combination of our divisible and fair e-cash schemes and is discussed in Section 6.7.

Another open problem is that of providing security proofs for efficient e-cash schemes; these are of major importance for potential implementations, as they significantly limit the financial and legal risks of service providers and banks. Pointcheval and Stern [107] have recently made a great advancement in this direction by providing proofs for several signature schemes, including Schnorr signatures, indicating that proofs of security for Brands' scheme may actually be feasible. Towards that goal the same authors have looked at security proofs for blind signature schemes [106]. Although several schemes, including the ones in this thesis, are analyzed and reduced to simple security assumptions, there remains a multitude of number-theoretic problems to be solved in order for full security proofs to be available for divisible, fair or even basic e-cash schemes. Furthermore, all current practical schemes assume the existence of random oracle-like hash functions. This is a very strong assumption and it would be interesting to see how it can be minimized or completely bypassed.

Chapter 4

Unlinkable divisible e-cash

As we have seen in the previous chapter the basic electronic cash model is insufficient for practical systems as it is token-based: extensions that allow for payments of exact amounts are necessary. As we have highlighted in Section 3.2 there are two possible ways in achieving this functionality: either keep a multitude of coins or devise an electronic cash scheme which allows coins to be divided. Independent of implementation, there are two settings in electronic cash that allow exact payments:

- (1) In the first setting when portions of the same coin, or collections of coins if a multitude of coins are kept, are being paid the bank knows that they originate from the same withdrawal and therefore from the same user; stated in the terminology of our definition in Section 3.1.1 the coin portions are *linkable*. We call this setting *linkable divisible e-cash*.
- (2) In the second setting the payment transcripts originating from the same withdrawal are unlinkable; we call this setting *unlinkable divisible e-cash*.

In this chapter we investigate the complexity of keeping a multitude of coins and devise an optimal algorithm for minimizing it; this result can then be directly applied to achieve unlinkable divisible e-cash—as well as linkable divisible e-cash, as shown in the next chapter. This approach puts a limit on the maximum number of exact payments that can be performed with a single coin; this is not necessarily a problem as the allowed number of payments can be chosen to fit the parameters of a specific application. On the other hand, achieving unlinkable divisible e-cash with practically unlimited number of payments is still an open problem, stated and analyzed in Section 5.6 in the following chapter; our result in this chapter can be

seen both as an approximation and as a lower bound on the efficiency of unlinkable divisible e-cash.

We formalize the complexity needed for keeping a multitude of coins by defining the k -payment problem: given a total budget of \mathcal{N} units, the problem is to represent this budget as a set of coins, so that any k exact payments of total value at most \mathcal{N} can be made using k disjoint subsets of the coins. The actual payments can be made on-line, namely *there is no need to know payment requests in advance*. Thus, given a solution to this problem, a user can withdraw a budget \mathcal{N} and be assured that he can make at least k exact payments of arbitrary amounts up to his/her budget \mathcal{N} . Our aim is to limit the number of coins in a solution, since in the basic non-divisible electronic cash schemes complexity in terms of storage, communication and computation requirements is directly proportional to the number of coins withdrawn. It is however interesting to note that the k -payment problem has additional applications in other resource-sharing scenarios, as discussed in [54].

Our results include a complete characterization of the k -payment problem as follows. First, we prove a necessary and sufficient condition for a given set of coins to solve the problem. Using this characterization, we prove that the number of coins in any solution to the k -payment problem is at least $kH_{\mathcal{N}/k}$, where H_n denotes the n -th element in the harmonic series. This condition can also be used to efficiently determine k , i.e., the maximal number of exact payments, which a given set of coins allows. Secondly, we give an algorithm which produces, for any \mathcal{N} and k , a solution with minimal number of coins. In the case that all denominations are available, the algorithm finds a coin allocation with at most $(k+1)H_{\mathcal{N}/(k+1)}$ coins. We also show that both upper and lower bounds are the best possible. The algorithm generalizes naturally to the case where some of the denominations are not available.

4.1 Introduction

Consider the following everyday scenario. You want to withdraw \mathcal{N} units of money from your bank. The teller asks you “how would you like to have it?” Let us assume that you need to have “exact change,” i.e., given any payment request $P \leq \mathcal{N}$, you should be able to choose a subset of your “coins” whose sum is precisely P . Let us further assume that you would like to withdraw your \mathcal{N} units with the least possible

number of coins. In this case, your answer depends on your estimate of how many payments you are going to make. In the worst case, you may be making \mathcal{N} payments of 1 unit each, forcing you to take \mathcal{N} 1-coins. On the other extreme you may need to make only a single payment P . In this case, even if you don't know P in advance, $\log_2(\mathcal{N} + 1)$ coins are sometimes sufficient, as seen later in this chapter. Here we provide a complete analysis of the general question, which we call the *k-payment problem*: what is the smallest set of coins which enables one to satisfy any k exact payment requests of total value up to \mathcal{N} .

Motivation. In any payment system, be it physical or electronic, some transactions require payments of exact amounts. Forcing shops to provide change to a customer simply shifts the problem from the customers to the shops. The number of coins is particularly important in electronic cash where communication, computation and storage requirements depend on the number of withdrawn coins, while the typical “smart-card” used to store them has small memory space and computational power [94].

The k -payment problem

Definition 4.1.1 *Given a budget, denoted \mathcal{N} , and a number of payments, denoted k , the (\mathcal{N}, k) -payment problem, or simply the k -payment problem, is to find, for each $i \geq 1$, the number of i -coins (also called coins of denomination i), denoted c_i , such that the following two requirements are satisfied.*

1. Budget compliance: $\mathcal{N} = \sum_{i=1}^{\infty} ic_i$, and
2. k -partition: *For any sequence of k payment requests, denoted P_1, P_2, \dots, P_k , with $\sum_{\ell=1}^k P_\ell \leq \mathcal{N}$, there exists a way to exactly satisfy these payments using the coins. That is, there exist non-negative integers $p_{i\ell}$, where $p_{i\ell}$ represents the number of i -coins used in the ℓ -th payment, such that*
 - $\sum_i ip_{i\ell} = P_\ell$ for each $1 \leq \ell \leq k$, and
 - $\sum_\ell p_{i\ell} \leq c_i$ for each $i \geq 1$.

The problem can thus be broken into two parts: (1) given \mathcal{N} and k find the c_i 's, i.e., how to partition \mathcal{N} into coins. This is henceforth referred to as the *coin allocation* problem. And (2) given the c_i 's find $p_{i\ell}$, i.e., how to make a payment. This is called the *coin dispensing* problem.

There are a few possible variants of the k -payment problem. First, in many systems not all denominations are available; for example, we do not know of any physical system with 3-coins. Even in the electronic cash realm, denominations may be an expensive resource, because each denomination requires a distinct pair of secret/public keys of the central authority [15, 100]. Thus an interesting variant of the allocation problem may be the *restricted denominations* version, where the set of possible solutions is restricted to only those in which $c_i = 0$ if $i \notin \mathcal{D}$, for a given *allowed denomination set* \mathcal{D} .

Also, one may consider the *on-line* coin dispensing problem, where the algorithm is required to dispense coins after each payment request, without knowledge of future requests, or the *off-line* version, where the value of all k payments is assumed to be known before the first coin is dispensed. We only consider the on-line problem here, since optimal coin allocation and dispensing in the off-line setting are straightforward.

Overview of results: We show that the coin dispensing problem can be easily solved by a greedy strategy when denominations are not restricted, even in the on-line setting. Most of our results concern the coin allocation problem and we sometimes refer to this part of the problem as the k -payment problem. The first basic result, given in Theorem 4.3.1, is a simple necessary and sufficient condition for a sequence c_1, c_2, \dots of coins to solve the allocation problem. Using this characterization, we establish (in Theorem 4.3.6) a lower bound of $kH_{\mathcal{N}/k} \approx k \ln(\mathcal{N}/k)$ on the number of coins in any solution for all \mathcal{N} and k , where H_n denotes the n -th element of the harmonic series. This lower bound is the best possible, as shown in Theorem 4.3.8. We also use the characterization condition to efficiently determine the maximal number k for which a given collection of coins solves the k payment problem (Corollary 4.3.9). Our next major result is an efficient algorithm which finds a solution for *any* \mathcal{N} and k (Theorem 4.4.1) using the least possible number of coins. In terms of \mathcal{N} and k , the number of coins is never more than $(k+1)H_{\mathcal{N}/(k+1)} \approx (k+1) \ln(\mathcal{N}/(k+1))$ (Theorem 4.4.5). Similarly to our lower bound, we show that the upper bound is the best possible (Theorem 4.4.7). Finally we show that the algorithm extends naturally to the restricted-denominations case.

Related work: To the best of our knowledge, we are the first to formulate the general k -payment problem, and hence the first to analyze it. One related classic combinatorial problem is *k-partition*, where the question is in how many ways can a

natural number \mathcal{N} be represented as a sum of k positive integers. This problem is less structured than the k -payment problem, and can be used to derive lower bounds; however, these bounds are suboptimal: see Section 4.2. Another reference worth mentioning appears in [38], where they briefly study the problem of how to represent a given \mathcal{N} with the least number of coins under various denomination systems. For a treatment of the harmonic series, which turns out to be a central quantity in our work, see [83]. The *postage-stamp problem (PSP)* is also closely related: cast in our terms, PSP is to find a set of denominations which will allow to pay any request of value $1, 2, 3, \dots, N$ using at most h coins, so that N is maximized. PSP can be viewed as an inverse of the 1-payment problem: there is one payment to make, the number of coins is given, and the goal is to find a denomination set of a given size which will maximize the budget. We remark that PSP is considered a difficult problem even for a very small number of denominations. See, e.g., [117, 118, 72] for more details. Another related question is *change making* [90], which is the problem of how to represent a given budget with the least number of coins from a given allowed denomination set. General change-making is (weakly) NP-hard. Kozen and Zaks [86], and Verma and Xu [128] study the question of which denomination sets allow one to use the greedy strategy for optimal change-making.

Organization: In Section 4.2 we give some preliminary observations and briefly discuss a few suboptimal results. In Section 4.3 we prove a characterization of the k -payment problem and a lower bound on the number of coins in any solution. In Section 4.4 we present and analyze an optimal algorithm, while Section 4.5 extends the algorithm to a restricted set of denominations. Section 4.6 shows how our algorithm can be used to construct unlinkable divisible e-cash, while the chapter is concluded with applications in other areas in Section 4.7.

4.2 Notation and simple results

In this section we develop some intuition for the k -payment problem by presenting a few simple upper and lower bounds on the number of coins required. The notation we shall use throughout this chapter is summarized in Figure 4.1. The solution S to which the c_i, T_i and m symbols refer to should be clear from the context. Note that $\mathcal{N} = T_m$ holds for any given set of coins.

Parameters of problem specification:

- \mathcal{N} : the total budget.
- k : the number of payments.

Quantities related to solution specification S :

- c_i ($i \geq 0$): the number of coins of denomination i (a.k.a. i -coins) in S . By convention, $c_0 = 0$.
- T_i ($i \geq 0$): the budget allocated in S using coins of denomination i or less. Formally, $T_i = \sum_{j=1}^i j c_j$.
- m : the largest denomination of a coin in S . Formally, $m = \max\{i \mid c_i > 0\}$.

Quantities related to making a payment:

- c'_i, T'_i, m' : refer to the respective quantities after a payment has been made.

Standard quantities:

- $H_n = \sum_{i=1}^n \frac{1}{i}$. We define $H_0 = 0$ so that $H_{i+1} = H_i + 1/(i+1)$ for all $i \geq 0$.
-

Figure 4.1: Glossary of notation.

For the remainder of this chapter, fix \mathcal{N} and k to be arbitrary given positive integers. Note that we may assume without loss of generality that $\mathcal{N} \geq k$, since payment requests of value 0 can be ignored.

Let us now do some rough analysis of the k -payment problem. As already mentioned above, the case $k = \mathcal{N}$ is trivial to solve: take $c_1 = k$ and $c_i = 0$ for $i \neq 1$. Clearly, no better solution is possible since $c_1 < k$ would not allow to satisfy k payments of value 1 each. The case of $k = 1$ is also quite simple, at least when $\mathcal{N} = 2^g - 1$ for an integer $g \geq 1$: we can solve it with $g = \log_2(\mathcal{N} + 1)$ coins of denominations $1, 2, 4, \dots, 2^{g-1}$. Given any request P , we can satisfy the coin dispensing problem by using the coins which correspond to the ones in the binary representation of P .

However, it is not immediately clear how one can generalize this solution to arbitrary \mathcal{N} and k . Consider an arbitrary \mathcal{N} : the usual technique of “rounding up” to the next power of 2 does not seem appropriate in the k -payment problem: can we ask the teller of the bank to round up the amount we withdraw just because it is more convenient for us? But let us ignore this point for the moment, and consider the problem of general k . If we were allowed to make the dubious assumption that we may enlarge \mathcal{N} , then one simple solution would be to duplicate the solution for 1-payment k times, and let each payment use its dedicated set of coins. Specifically, this means that we allocate k 1-coins, k 2-coins, k 4-coins and so on, up to $k 2^{\lceil \log_2(\mathcal{N}+1) \rceil - 1}$ -coins.

```

GREEDY-DISPENSE ( $P$ )
1  while  $P > 0$ 
2      do  $i \leftarrow \max\{l \mid l \leq P, c_l > 0\}$             $\triangleright$   $i$  is highest possible denomination
3           $j \leftarrow \min(\lfloor P/i \rfloor, c_i)$                   $\triangleright$  use as many  $i$ -coins as possible
4          dispense  $j$   $i$ -coins
5           $c_i \leftarrow c_i - j$ 
6           $P \leftarrow P - ji$ 

```

Figure 4.2: The greedy algorithm for coin dispensing. P is the amount to be paid.

The result is about $k \log_2 \mathcal{N}$ coins, and the guarantee we have based on this simplistic construction is that we can pay k payments, but for all we know *each* of these payments must be of value at most \mathcal{N} . However, the total budget allocated in this solution is in fact $k\mathcal{N}$, and thus it does not seem to solve the k -payment problem as stated, where the only limit on the value of payments is placed on their *sum*, rather than on individual values.

Jumping ahead to later results, it may be interesting to remark here that one corollary of our work is that if coin dispensing is done with a greedy strategy (see Figure 4.2), then the above “binary” construction for coin allocation indeed solves the general k -payment problem. More precisely, assume that $(\mathcal{N}/k) - 1$ is a power of 2. Then the coin allocation algorithm allocates k coins of each denomination $1, 2, 4, \dots, 2^{\log_2((\mathcal{N}/k)+1)-1}$. Clearly, the number of coins is $k \log_2((\mathcal{N}/k) + 1)$ and their sum is \mathcal{N} . Coin dispensing is performed with the *greedy strategy*: at each point, the largest possible coin is used. A formal description of the greedy dispensing algorithm, called GREEDY-DISPENSE, is presented in Figure 4.2. However, one should note that, perhaps surprisingly, our results also indicate that the binary algorithm is *not* the right generalization for $k > 1$: the best algorithm (described in Section 4.4) yields a factor of about $(1 - \ln 2)$ improvement, i.e., roughly 30% less coins.

Let us now re-consider the question of general \mathcal{N} . Once we have an algorithm for infinitely many values of k and \mathcal{N} , generalizing to arbitrary \mathcal{N} and k is easy: find a solution for \mathcal{N}' and k' , where $\mathcal{N}' \geq \mathcal{N}$ and $k' \geq k + 1$, then dispense a payment of value $\mathcal{N}' - \mathcal{N}$. The remaining set of coins is a coin allocation of total budget \mathcal{N} , and it can be used for k additional payments since the original set solved the $(k+1)$ -payment problem.

We close this section with a simple lower bound on the number of coins required in any solution to the k -payment problem. The bound is based on a counting argument and we only sketch it here. For $k = 1$, the number of distinct possible payment requests is $\mathcal{N} + 1$ (0 is allowed). Observe that the algorithm must dispense a different set of coins in response to each request. It follows that the number of coins in any solution must be at least $\log_2(\mathcal{N} + 1)$. This argument can be extended to general k , using the observation that the number of distinct responses of the algorithm (disregarding order) is at least $p_k(\mathcal{N})$, the number of ways to represent \mathcal{N} as a sum of k positive integers. Using standard bounds for partitions [127] and the fact that the number of responses is exponential in the number of coins, one can conclude that the number of coins is $\Omega(k \log(\mathcal{N}/k^2))$.

4.3 Problem characterization

In this section we prove a simple condition to be necessary and sufficient for a set of coins to correctly solve the k -payment problem. Next, using this result, we obtain a tight lower bound on the number of coins in any solution to the k -payment problem. Finally, we outline an efficient algorithm which, given a set of coins S , determines the maximal k for which S is a solution of the k -payment problem.

The necessity proof is not hard: the intuition is that the “hardest” cases are when all payment requests are equal. The more interesting part is the sufficiency proof. The reader is referred to Figure 4.1 for the notation.

Theorem 4.3.1 *S solves the k -payment problem if and only if $T_i \geq ki$ for all $0 \leq i < m$.*

Proof. A direct corollary of Lemmata 4.3.2 and 4.3.3 below. □

We start by upper bounding m .

Lemma 4.3.2 *If S solves the k -payment problem, then $m \leq \lceil \mathcal{N}/k \rceil$.*

Proof. By contradiction. Suppose $m > \lceil \mathcal{N}/k \rceil$, and consider k payments of values $\lfloor \mathcal{N}/k \rfloor$ and $\lceil \mathcal{N}/k \rceil$ such that their total sum is \mathcal{N} . Clearly, none of these payments can use m -coins, and since $c_m \geq 1$ by definition, the total budget available for these payments is at most $N - m$, a contradiction. □

The following lemma is slightly stronger than the condition in Theorem 4.3.1. We use this version in the proof of Theorem 4.3.6.

Lemma 4.3.3 *If S solves the k -payment problem, then $T_i \geq ki$ for all $0 \leq i \leq \lfloor \mathcal{N}/k \rfloor$.*

Proof. Let $i \leq \lfloor \mathcal{N}/k \rfloor$. Then $ki \leq \mathcal{N}$. Consider k payments of value i each: each such payment can be done only with coins of denomination at most i , hence $T_i \geq ki$. \square

We next prove sufficiency. We start by showing that if the condition of Theorem 4.3.1 holds for $k = 1$, then any single payment of value up to \mathcal{N} can be satisfied by S with GREEDY-DISPENSE .

Lemma 4.3.4 *Let P be a payment request, and suppose that in S we have that for some j ,*

- (1) $P \leq T_j$, and
- (2) $T_i \geq i$ for all $0 \leq i < j$.

Then P can be satisfied by GREEDY-DISPENSE using only coins of denomination j or less.

Proof. We prove, by induction on j , that the claim holds for j and any P . The base case is $j = 1$: then by condition (1) we have that $T_1 \geq P$ and hence there are at least P 1-coins, which can be used to pay any amount up to their total sum. For the inductive step, assume that the claim holds for j and all P , and consider $j + 1$. Let a be the number of $(j + 1)$ -coins dispensed by GREEDY-DISPENSE , namely,

$$a = \min \left(\left\lfloor \frac{P}{j+1} \right\rfloor, c_{j+1} \right) .$$

Let R denote the remainder of the payment after the algorithm dispenses the $(j + 1)$ -coins, i.e., $R = P - a(j + 1)$. Note that (2) trivially holds after dispensing the $(j + 1)$ -coins; we need to show that (1) holds as well. We consider two cases. If $a = c_{j+1}$, then using condition (1) of the induction hypothesis we get

$$\begin{aligned} T_j &= T_{j+1} - c_{j+1}(j + 1) \\ &= T_{j+1} - a(j + 1) \\ &\geq P - a(j + 1) \\ &= R , \end{aligned}$$

and we are done for this case.

If $a < c_{j+1}$, then since the algorithm is greedy, it must be the case that $R < j + 1$. On the other hand, by (2) we have that $T_j \geq j$, and hence $T_j \geq R$ and we are done in this case too. \square

The following lemma is the key invariant preserved by GREEDY-DISPENSE . It is interesting to note that while the algorithm proceeds from larger coins to smaller ones the inductive proof goes in the opposite direction. Recall that “primed” quantities refer to the value after a payment is done.

Lemma 4.3.5 *If $T_i \geq ki$ holds for all $0 \leq i < m$, then after GREEDY-DISPENSE dispenses any amount up to T_m , $T'_i \geq (k - 1)i$ holds for all $0 \leq i < m'$.*

Proof. By induction on i . For $i = 0$ the claim is trivial. Assume that the claim holds for all $l < i$, and consider i . Let $S_i = T_i - T'_i$ be the amount dispensed using coins of denomination at most i .

Define $j = \min \{j \mid j > i, c'_j > 0\}$, namely j is the smallest remaining denomination which is larger than i . Note that j is well defined since $i < m'$, namely i is not the largest remaining coin. Next, note that since all the coins of denomination $i + 1, i + 2, \dots, j - 1$ whose sum is $T_{j-1} - T_i$ were used by the algorithm, we have that

$$S_{j-1} = T_{j-1} - T_i + S_i \tag{4.1}$$

Now, observe that since the algorithm is greedy and since at least one j -coin was not used by the algorithm, it must be the case that the total amount dispensed using coins of denomination smaller than j is less than j , i.e., $S_{j-1} \leq j - 1$. Using Eq. (4.1), we get that $T_i \geq T_{j-1} - j + 1 + S_i$. Finally, using the assumption applied to $j - 1$, we obtain

$$\begin{aligned} T'_i &= T_i - S_i \\ &\geq T_{j-1} - j + 1 \\ &\geq (j - 1)k - j + 1 \\ &= (j - 1)(k - 1) \\ &\geq i(k - 1) \end{aligned}$$

\square

We now complete the proof of the characterization.

Proof: (of Theorem 4.3.1) The necessity of the condition follows directly from Lemmas 4.3.2 and 4.3.3. For the sufficiency, assume that $T_i \geq ik$ for all $0 \leq i < m$, and consider a sequence of up to k requests of total value at most \mathcal{N} . After the l -th request is served by GREEDY-DISPENSE, we have, by inductive application of Lemma 4.3.5, that $T'_i \geq (k-l)i$ for all $0 \leq i < m$. Moreover, by Lemma 4.3.4, any amount up to the total remainder can be paid from S , so long as $k-l > 0$, which completes the proof. \square

We can now prove a lower bound on the number of coins in any solution to the k -payment problem.

Theorem 4.3.6 *The number of coins in any solution to the k -payment problem is at least $kH_{\lfloor \mathcal{N}/k \rfloor} \approx k \ln \frac{\mathcal{N}}{k}$.*

For the proof of Theorem 4.3.6, we first prove the following lemma.

Lemma 4.3.7 *For any number j , $\sum_{i=1}^j c_i = \sum_{i=1}^{j-1} \frac{T_i}{i(i+1)} + \frac{T_j}{j}$.*

Proof.

$$\begin{aligned} \sum_{i=1}^j c_i &= \sum_{i=1}^j \frac{T_i - T_{i-1}}{i} \\ &= \sum_{i=1}^{j-1} \left(\frac{T_i}{i} - \frac{T_i}{i+1} \right) + \frac{T_j}{j} \\ &= \sum_{i=1}^{j-1} \frac{T_i}{i(i+1)} + \frac{T_j}{j}. \end{aligned}$$

\square

Proof of Theorem 4.3.6. By Lemma 4.3.7 and Lemma 4.3.3:

$$\begin{aligned} \sum_{i=0}^{\lfloor \mathcal{N}/k \rfloor} c_i &= \sum_{i=1}^{\lfloor \mathcal{N}/k \rfloor - 1} \frac{T_i}{i(i+1)} + \frac{T_{\lfloor \mathcal{N}/k \rfloor}}{\lfloor \mathcal{N}/k \rfloor} \\ &\geq \sum_{i=1}^{\lfloor \mathcal{N}/k \rfloor - 1} \frac{ki}{i(i+1)} + k \end{aligned}$$

$$\begin{aligned}
&= k \sum_{i=1}^{\lfloor \mathcal{N}/k \rfloor - 1} \frac{1}{i+1} + k \\
&= k(H_{\lfloor \mathcal{N}/k \rfloor} - 1) + k \\
&= kH_{\lfloor \mathcal{N}/k \rfloor} . \quad \square
\end{aligned}$$

4.3.1 Tightness of lower bound

The lower bound of Theorem 4.3.6 is the best possible in general. This can be seen by considering a number k which is divisible by all numbers $1, 2, 3, \dots, \mathcal{N}/k$; for example, given a natural number l , let $k = l!$ and $\mathcal{N} = kl$. In this case the solution with $c_i = k/i$ for $1 \leq i \leq \mathcal{N}/k$ solves the k -payment problem by Theorem 4.3.1, and its total number of coins is precisely $kH_{\lfloor \mathcal{N}/k \rfloor}$. Formally:

Theorem 4.3.8 *For any natural numbers n_1, n_2 , there are infinitely many $N > n_1, k > n_2$ for which there exists a solution for the (N, k) -payment problem with exactly $kH_{\lfloor N/k \rfloor}$ coins.*

Proof. This can be seen by simply choosing a natural number l for which $l! > n_2, l \cdot l! > n_1$, and setting $k = l!, \mathcal{N} = kl$ as above. \square

4.3.2 Determining k for a given set

Consider the “inverse problem:” we are given a set of coins, with c_i coins of denomination i for each i , and the question is how many payments can we make using these coins, i.e., find k . Note that k is well defined: we say that k is 0 if there is a payment request which cannot be satisfied by the set, and k is never more than the total budget. Theorem 4.3.1 can be directly applied to answer such a question efficiently, as implied by the following simple corollary.

Corollary 4.3.9 *Let S be a given coin allocation. Then S solves the k -payment problem if and only if $k \leq \min \{ \lfloor T_i/i \rfloor \mid 1 \leq i < m \}$.*

This efficient solution to this problem allows for “recalculation” of the available payments. To elaborate on this point, consider a solution to the (\mathcal{N}, k) -payment problem: the set of coins satisfying the solution has been chosen so that all possible worst cases of payment requests can be satisfied. But it may be the case that by conducting l payments of value \mathcal{N}_l the remaining coins can satisfy more than $k - l$

arbitrary payment requests. That is, they not only solve the $(\mathcal{N} - \mathcal{N}_l, k - l)$ -payment problem but in fact they are sufficient to solve the $(\mathcal{N} - \mathcal{N}_l, k')$ -payment problem, for some $k' > k - l$. A simple example is having the first payment request being of value m , where m is the highest denomination of the solution set. Then GREEDY-DISPENSE will dispense one coin of denomination m and the remaining coins not only satisfy the $(\mathcal{N} - m, k - 1)$ problem, but actually the $(\mathcal{N} - m, k)$ problem.

Thus a sample application of this corollary would be to check, after k payments, if the remaining coin set allows for any additional arbitrary exact payments. This would optimize the use of the available coins.

4.4 An optimal solution

We now present a coin allocation algorithm which finds a minimal solution to the k -payment for arbitrary \mathcal{N} and k . We prove the optimality of the algorithm, and an upper bound on the number of coins it generates. In Section 4.5 we generalize the algorithm to handle a restricted set of denominations.

The allocation algorithm. Given arbitrary integers $\mathcal{N} \geq k > 0$, the algorithm presented in Figure 4.3, called ALLOCATE, finds an optimal coin allocation. Intuitively, the algorithm works by scanning all possible denominations $i = 1, 2, 3, \dots$ and adding, for each i , the least number of i -coins which suffices to make $T_i \geq ki$. The number of coins c_i is thus an approximation of the i -th harmonic element multiplied by k . The last coin is treated differently to avoid overflow from the total budget of \mathcal{N} .

Theorem 4.4.1 *Let S denote the solution produced by ALLOCATE in Figure 4.3. Then S solves the k -payment problem using the least possible number of coins.*

Proof. Follows from Lemma 4.4.3 and Lemma 4.4.4 below. □

The important properties of the algorithm are stated in the following loop invariant.

Lemma 4.4.2 *Whenever line 4 is executed by ALLOCATE, the following assertions hold.*

$$(i) \quad t = \sum_{j=1}^i j c_j \leq \mathcal{N}.$$

$$(ii) \quad \text{if } \mathcal{N} - t > i \text{ then } t \geq ik.$$

```

ALLOCATE( $\mathcal{N}, k$ )
1  $c_i \leftarrow 0$  for all  $i$  ▷ initialize
2  $t \leftarrow 0$ 
3  $i \leftarrow 0$ 
4 while  $\mathcal{N} - t > i$  ▷ consider denominations in order
5     do  $i \leftarrow i + 1$ 
6     if  $t < ki$ 
7         then  $c_i \leftarrow \min\left(\left\lceil \frac{ki-t}{i} \right\rceil, \left\lfloor \frac{\mathcal{N}-t}{i} \right\rfloor\right)$  ▷ add  $i$ -coins but don't overflow
8          $t \leftarrow t + ic_i$ 
9 if  $\mathcal{N} > t$  ▷ add remainder by a single last coin
10 then  $c_{\mathcal{N}-t} \leftarrow c_{\mathcal{N}-t} + 1$ 

```

Figure 4.3: Algorithm for optimal solution of the k -payment problem.

(iii) if $i > 0$ then $t < i(k + 1)$.

Proof. Line 4 can be reached after executing lines 1–3 or after executing the loop of lines 5–8. In the former case, we have $t = i = 0$ and the lemma holds trivially.

Suppose now that line 4 is reached after an iteration of the loop. We denote by t_0 the value of the variable t before the last execution of the loop. If lines 7 and 8 are not executed, then (i) holds trivially. If they are executed, then we have that $t = t_0 + ic_i$, and $t \leq \left\lfloor \frac{\mathcal{N}-t_0}{i} \right\rfloor \cdot i + t_0 \leq \mathcal{N}$, and hence (i) holds after the loop is executed. Also note that $t \leq t_0 + \left\lceil \frac{ki-t_0}{i} \right\rceil \cdot i < t_0 + (ki - t_0 + i) = i(k + 1)$, and therefore (iii) holds true after the execution of the loop.

Finally, we prove that (ii) holds after executing the loop. If lines 7–8 are not executed, (ii) holds trivially. Suppose that lines 7–8 are executed, and that $i < \mathcal{N} - t$. In this case, by line 8, $i < \mathcal{N} - t_0 - ic_i$, which means that

$$c_i < \frac{\mathcal{N} - t_0}{i} - 1 < \left\lfloor \frac{\mathcal{N} - t_0}{i} \right\rfloor.$$

Therefore, by line 7, $c_i = \left\lceil \frac{ki-t}{i} \right\rceil$, and hence $t = t_0 + \left\lceil \frac{ki-t_0}{i} \right\rceil \cdot i \geq t_0 + ki - t_0 = ki$, and we are done. \square

Using Lemma 4.4.2 and Theorem 4.3.1, correctness is easily proven. We introduce two new variables to separate the handling of the remainder:

- c_i^* , for all i , is the value of the c_i variable when line 9 is executed.
- $T_i^* = \sum_{j=1}^i jc_j^*$.

Lemma 4.4.3 *S solves the k-payment problem.*

Proof. By (i) of Lemma 4.4.2, in conjunction with lines 9–10 of the code, we have that upon completion of the algorithm, $\sum_i ic_i = \mathcal{N}$. By (ii) of Lemma 4.4.2, and since (by lines 4 and 10) the largest denomination is $m \leq \mathcal{N} - T_m^*$, we have that $T_i \geq T_i^* \geq ki$ for all denominations $i < m$, and therefore, by Theorem 4.3.1, *S* solves the *k*-payment problem. \square

Proving optimality takes more work. We give the full proof below.

Lemma 4.4.4 *Let \mathcal{T} be any solution for the k-payment problem, with largest denomination n and d_i i -coins for $1 \leq i \leq n$. Then $\sum_{i=1}^m c_i \leq \sum_{i=1}^n d_i$.*

Proof. We use the following additional definitions.

- $U_j = \sum_{i=1}^j id_i$.
- $\Delta_0 = 0$ and $\Delta_i = \Delta_{i-1} + d_i - c_i^*$ for $i > 0$.

Note that by definitions, the following holds for all $i \geq 0$:

$$\Delta_i = \sum_{j=1}^i d_j - \sum_{j=1}^i c_j^* \quad (4.2)$$

$$d_i = c_i^* + \Delta_i - \Delta_{i-1} \quad (4.3)$$

The following claim “nearly” proves the lemma.

Claim A. $\sum_{i=1}^j d_i \geq \sum_{i=1}^j c_i^*$ for all $1 \leq j < n$.

Proof of Claim A: By contradiction. Suppose $\sum_{i=1}^l d_i < \sum_{i=1}^l c_i^*$ for $l < n$, and suppose that l is the smallest such index, i.e., (using Eq. (4.2)), $\Delta_l < 0$ and $\Delta_j \geq 0$ for all $0 \leq j < l$. Hence, by Eq. (4.3), we have that $d_l < c_l^* - \Delta_{l-1}$, which implies, by integrality, that $d_l \leq c_l^* - \Delta_{l-1} - 1$. Therefore, since $T_l^* < (k+1)l$ by (iii) of Lemma 4.4.2, we have

$$\begin{aligned} U_l &= U_{l-1} + ld_l \\ &\leq \sum_{i=1}^{l-1} id_i + lc_l^* - l\Delta_{l-1} - l \\ &= \sum_{i=1}^{l-1} i(c_i^* + \Delta_i - \Delta_{i-1}) + lc_l^* - l\Delta_{l-1} - l \end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{l-1} ic_i^* + \sum_{i=0}^{l-2} (i\Delta_i - (i+1)\Delta_i) + (l-1)\Delta_{l-1} + lc_l^* - l\Delta_{l-1} - l \\
&= T_l^* - \sum_{i=1}^{l-1} \Delta_i - l \\
&\leq T_l^* - l \\
&< (k+1)l - l \\
&= kl ,
\end{aligned}$$

i.e., $U_l < kl$, contradiction to Theorem 4.3.1, since $l < n$. \square

We now complete the proof of the lemma by showing that $\sum_{i=1}^n d_i \geq \sum_{i=1}^m c_i$. We consider two cases. If $n > m$ then using Claim A, and the fact that $d_n \geq 1$ by definition of n , we get

$$\sum_{i=1}^m c_i \leq \sum_{i=1}^m c_i^* + 1 \leq \sum_{i=1}^{n-1} d_i + 1 \leq \sum_{i=1}^n d_i ,$$

and we are done for this case.

So suppose $n \leq m$. Let $b = \sum_{i=n}^m c_i - d_n$. We prove the lemma by showing that $\sum_{i=1}^{n-1} (d_i - c_i) \geq b$. First, note that since by Claim A we have $\Delta_i \geq 0$ for $i \leq n-1$, and using Eq. (4.3), we get

$$\begin{aligned}
U_{n-1} - T_{n-1}^* &= \sum_{i=1}^{n-1} i(d_i - c_i^*) \\
&= \sum_{i=1}^{n-1} i(\Delta_i - \Delta_{i-1}) \\
&= (n-1)\Delta_{n-1} - \sum_{i=0}^{n-2} \Delta_i \\
&\leq (n-1)\Delta_{n-1} .
\end{aligned}$$

On the other hand, since $U_n = \mathcal{N} = T_m$, we get

$$\begin{aligned}
U_{n-1} &= U_n - nd_n \\
&= T_m - n \left(\sum_{i=n}^m c_i - b \right) \\
&\geq T_m - \sum_{i=n}^m ic_i + bn \\
&= T_{n-1} + bn .
\end{aligned}$$

Therefore, it follows from Eq. (4.2) that

$$\sum_{i=1}^{n-1} (d_i - c_i^*) = \Delta_{n-1} \geq \left\lceil \frac{U_{n-1} - T_{n-1}^*}{n-1} \right\rceil \geq \left\lceil \frac{T_{n-1} + bn - T_{n-1}^*}{n-1} \right\rceil \quad (4.4)$$

We now consider two sub-cases. If $T_{n-1} = T_{n-1}^*$ then $c_i = c_i^*$ for $1 \leq i \leq n-1$, and Eq. (4.4) reduces to

$$\sum_{i=1}^{n-1} (d_i - c_i) \geq \left\lceil \frac{nb}{n-1} \right\rceil \geq b,$$

and we are done for this sub-case.

Otherwise, $T_{n-1} - T_{n-1}^* \geq 1$ and therefore

$$\begin{aligned} \sum_{i=1}^{n-1} (d_i - c_i) &\geq \sum_{i=1}^{n-1} (d_i - c_i^*) - 1 \\ &\geq \left\lceil \frac{T_{n-1} + bn - T_{n-1}^*}{n-1} \right\rceil - 1 \\ &\geq \left\lceil \frac{bn+1}{n-1} \right\rceil - 1 \\ &\geq b, \end{aligned}$$

and the proof of Lemma 4.4.4 is complete. \square

Theorem 4.4.1 proved that the number of coins in the solution produced by ALLOCATE is optimal. We now give a bound on that number in terms of \mathcal{N} and k . There is a nice interpretation to the theorem below: it says that the worst penalty for having \mathcal{N} and k which are not “nice” is equivalent to requiring an extra payment (i.e., solving the $(k+1)$ -payment problem) for “nice” \mathcal{N} and k .

Theorem 4.4.5 *The number of coins in the solution produced by ALLOCATE is at most $(k+1)H_{\lceil \mathcal{N}/(k+1) \rceil} \approx (k+1) \ln \frac{\mathcal{N}}{k+1}$.*

First, we prove an upper bound on m , which is slightly sharper than the general bound of Lemma 4.3.2.

Lemma 4.4.6 *The largest denomination of a coin generated by ALLOCATE satisfies $m \leq \left\lceil \frac{\mathcal{N}}{k+1} \right\rceil$.*

Proof. By (iii) of Lemma 4.4.2 we have $T_i^* < (k+1)i$, and in particular

$$m > \frac{T_m^*}{k+1} \quad (4.5)$$

By (ii) of Lemma 4.4.2 we have that $T_{m-1}^* \geq k(m-1)$, or that $m \leq \frac{T_{m-1}^*}{k} + 1$. Since $c_m \geq 1$, we have that $T_{m-1}^* \leq T_m^* - m$. Using also Eq. (4.5), we get that

$$\begin{aligned}
m &\leq \frac{T_{m-1}^*}{k} + 1 \\
&\leq \frac{T_m^* - m}{k} + 1 \\
&< \frac{T_m^* - \frac{T_m^*}{k+1}}{k} + 1 \\
&= \frac{T_m^*}{k+1} + 1 \\
&\leq \frac{\mathcal{N}}{k+1} + 1,
\end{aligned}$$

i.e., $m < \frac{\mathcal{N}}{k+1} + 1$, and by integrality, $m \leq \left\lceil \frac{\mathcal{N}}{k+1} \right\rceil$. \square

Proof: (of Theorem 4.4.5) By (iii) of Lemma 4.4.2, and by integrality, $T_i^* \leq (k+1)i-1$ for all $1 \leq i \leq m$. This, in conjunction with Lemmas 4.3.7 and 4.4.6, implies that

$$\begin{aligned}
\sum_{i=1}^m c_i^* &= \sum_{i=1}^{m-1} \frac{T_i^*}{i(i+1)} + \frac{T_m^*}{m} \\
&\leq \sum_{i=1}^{m-1} \frac{i(k+1) - 1}{i(i+1)} + k + 1 - \frac{1}{m} \\
&= \sum_{i=1}^{m-1} \frac{i(k+1)}{i(i+1)} - \sum_{i=1}^{m-1} \frac{1}{i(i+1)} + k + 1 - \frac{1}{m} \\
&= (k+1)(H_m - 1) + k \\
&\leq (k+1)H_{\lceil \mathcal{N}/(k+1) \rceil} - 1,
\end{aligned}$$

and therefore, $\sum_i c_i \leq \sum_i c_i^* + 1 \leq (k+1)H_{\lceil \mathcal{N}/(k+1) \rceil}$, as required. \square

4.4.1 Tightness of upper bound

The upper bound given by theorem 4.4.5 is the best possible in general. To see this, consider the case where $k+1$ is divisible by $2, 3, 4, \dots, m$, and $\mathcal{N} = 1 + \sum_{i=1}^m \lceil k/i \rceil \cdot i = m(k+1)$. For example, let m be any natural number, and take $k = m! - 1$ and $\mathcal{N} = m(m! - 1)$. The reader may verify that for these \mathcal{N} and k , we have largest denomination $m = \lceil \mathcal{N}/(k+1) \rceil$, $c_i = (k+1)/i$ for $1 \leq i \leq m$, and $c_i = 0$ otherwise.

Therefore the number of coins for such \mathcal{N} 's and k 's is precisely $(k + 1)H_{\lceil \mathcal{N}/(k+1) \rceil}$. Formally:

Theorem 4.4.7 *For any natural numbers n_1, n_2 , there are infinitely many $N > n_1, k > n_2$ such that any solution for the (N, k) -payment problem requires at least $(k + 1)H_{\lceil N/(k+1) \rceil}$ coins.*

Proof. This can be seen by simply choosing a natural number m for which $m! - 1 > n_2$, $l \cdot k > n_1$, and setting $k = l! - 1$, $\mathcal{N} = kl$ as above. \square

4.5 Generalization to a restricted set of denominations

First, observe that the problem is solvable if and only if “1” is not forbidden: if there are no 1-coins then certainly we cannot satisfy any payment request of value 1; and if 1 is allowed, then the trivial solution of \mathcal{N} 1-coins works for all k .

To get an optimal algorithm for an arbitrary set of denominations which allows for a solution, we follow the idea of the ALLOCATE algorithm in Figure 4.3: ensure, with the least number of coins, that $T_i \geq ik$ for all $1 \leq i < m$. The treatment of the remainder (lines 9–10 in Figure 4.3) is more complicated here, but has the same motivation: add the remainder with the least possible number of coins. Remainder allocation is precisely the *change-making* problem. In the general case this is solved with dynamic programming, although for some allowed denomination sets the greedy strategy works too [86, 128]. The main algorithm, called ALLOCATE-GENERALIZED, is given in Figures 4.4–4.5. Note that the dynamic algorithm, called DYNAMIC-DISPENSE, is limited in the way it handles the remainder; namely it is only permitted to utilize coins up to a certain value i . This is necessary to avoid large “gaps” in the allocated coins, which in turn could result in a violation of the necessity condition, i.e., Theorem 4.3.1. In the proof of Theorem 4.5.1 below we show that any competing algorithm is forced to follow the same strategy and thus our ALLOCATE-GENERALIZED is optimal.

Also notice that, in contrast to the unrestricted case, ALLOCATE-GENERALIZED only allocates i -coins if all the necessary coins for covering the “gap” up to the next available denomination are available (line 8); this turns out to be necessary in order to give enough “freedom” to DYNAMIC-DISPENSE, which examines the possibility that

the remainder requires less coins if coins of the higher denomination are *not* allocated.

We point out that if we have a base-2 denomination system, i.e., $\mathcal{D} = \{2^i \mid i \geq 0\}$, then in the case that $\mathcal{N} = k(2^g - 1)$ ALLOCATE-GENERALIZED produces the solution described in Section 4.2: k coins of each denomination $1, 2, 4, \dots, 2^{g-1}$.

In order to handle the “gaps” in allowed denominations, we introduce the following notation.

- \underline{i} , the allowed denomination immediately lower than i .
- \bar{i} , the allowed denomination immediately higher than i .

```

ALLOCATE-GENERALIZED( $\mathcal{N}, k, \mathcal{D}$ )
1   $c_i \leftarrow 0$  for all  $i$ 
2   $t \leftarrow 0$ 
3   $i \leftarrow 0$ 
4  while  $\mathcal{N} - t \geq \bar{i}$ 
5      do  $i \leftarrow \bar{i}$ 
6           $j \leftarrow \bar{i} - 1$            $\triangleright j$  is the largest den. smaller than the next one in  $\mathcal{D}$ 
7          if  $t < kj$                    $\triangleright$  ensure  $T_l \geq kl$  for all  $l$  up to  $j$ 
8              then if  $\lfloor \frac{\mathcal{N}-t}{i} \rfloor \geq \lfloor \frac{kj-t}{i} \rfloor$        $\triangleright$  check if budget is exhausted
9                  then                                      $\triangleright$  if not, add coins
10                      $c_i \leftarrow \lfloor \frac{kj-t}{i} \rfloor$ 
11                      $t \leftarrow t + ic_i$ 
12                 else                                      $\triangleright$  let the dynamic algorithm handle
                                                             $\triangleright$  the remainder
13                                     goto 14
14 if  $\mathcal{N} > t$ 
15     then  $\mathcal{D}' \leftarrow \{d \mid d \in \mathcal{D}, d \leq i\}$        $\triangleright$  put an upper bound for the dynamic
                                                             $\triangleright$  algorithm
16         add coins returned by DYNAMIC-DISPENSE( $\mathcal{N} - t, \mathcal{D}'$ )     $\triangleright$  see
                                                             $\triangleright$  Figure 4.5

```

Figure 4.4: Algorithm for optimal solution of the k -payment problem with allowed denominations \mathcal{D} .

We now prove sufficiency and optimality of ALLOCATE-GENERALIZED; the proofs are similar to those for the non-restricted case, and follow the steps of Theorem 4.4.1 and Lemmas 4.4.2–4.4.4.

Theorem 4.5.1 *Let S denote the solution produced by ALLOCATE-GENERALIZED,*

```

DYNAMIC-DISPENSE( $R, \mathcal{D}$ )
1  for  $i \leftarrow 1$  to  $R$                                 ▷ initialize  $M$  with single coins
2      do if  $i \in \mathcal{D}$ 
3          then  $M[i] \leftarrow \{i\}$ 
4          else  $M[i] \leftarrow \emptyset$ 
5  while  $M[R] = \emptyset$                                 ▷ at most  $R$  iterations here
6      do for  $i \leftarrow R$  downto 1
7          do if  $M[i] = \emptyset$ 
8              then for  $j \in \mathcal{D}$  s.t.  $j < i$             ▷ try to get  $i$  by adding
                                                         ▷ another coin
9                  do if  $M[i - j] \neq \emptyset$ 
10                     then  $M[i] \leftarrow M[i - j] \cup \{j\}$ 
11 return  $M[R]$ 

```

Figure 4.5: Dynamic programming algorithm for finding least number of coins whose sum is R units using denomination set \mathcal{D} . M is an R -length vector of multisets: before the l 'th iteration of the “while” loop, $M[i]$ is either empty or a multiset of at most l coins of total value exactly i .

shown in Figure 4.4. Then S solves the k -payment problem with allowed denominations \mathcal{D} using the least possible number of coins.

Proof. The proof is broken into Lemmas 4.5.3 and 4.5.5 below. □

We again use the following loop invariant to capture the important properties of the algorithm.

Lemma 4.5.2 *Whenever line 4 is executed by ALLOCATE-GENERALIZED, the following assertions hold.*

- (i) $t = \sum_{j=1}^i jc_j \leq \mathcal{N}$.
- (ii) if $\mathcal{N} - t \geq \bar{i}$ then $t \geq jk$, where $j = \bar{i} - 1$.
- (iii) if $i > 0$ then $t < jk + i$.

Proof. Line 4 can be reached after executing lines 1–3 or after executing the loop of lines 5–11. In the former case, we have $t = i = 0$ and the lemma holds trivially.

Suppose now that line 4 is reached after an iteration of the loop. We denote by t_0 the value of the variable t before the last execution of the loop. If lines 10 and 11 are not executed, then (i) holds trivially. If they are executed, then $t = t_0 + ic_i$, and

$t \leq \left\lceil \frac{\mathcal{N}-t_0}{i} \right\rceil \cdot i + t_0 \leq \mathcal{N}$, hence (i) holds after the loop is executed. Also note that whether or not lines 10 and 11 are executed $t \leq t_0 + \left\lceil \frac{kj-t_0}{i} \right\rceil \cdot i < t_0 + (kj-t_0+i) = jk+i$, and thus (iii) is true after the execution of the loop.

Last, we show that (ii) holds after executing the loop. If line 8 is not executed, or if lines 12–13 are executed then (ii) holds trivially. Otherwise lines 10–11 are executed, and $c_i = \left\lceil \frac{kj-t}{i} \right\rceil$, hence $t = t_0 + \left\lceil \frac{kj-t_0}{i} \right\rceil \cdot i \geq t_0 + kj - t_0 = kj$, as required. \square

Using Lemma 4.5.2 and Theorem 4.3.1, we prove correctness below.

Let c_i^* and T_i^* denote the values of c_i, T_i respectively before the handling of the remainder in lines 14–16.

Lemma 4.5.3 *S solves the k -payment problem with allowed denominations \mathcal{D} .*

Proof. By (i) of Lemma 4.5.2, in conjunction with lines 14–16 of the code, we have that upon completion of the algorithm, $\sum_i ic_i = \mathcal{N}$. Let m be the highest denomination allocated by the algorithm. Then, from lines 14–16 and by (ii) of Lemma 4.5.2, we have that $T_l = T_i \geq T_i^* \geq k(\bar{i} - 1) \geq kl$, for all $1 \leq i < m$ and $i \leq l < \bar{i} \leq m$. Therefore, by Theorem 4.3.1, S solves the k -payment problem. \square

For optimality we first need to show that DYNAMIC-DISPENSE allocates the minimum number of coins for the remainder. This is direct from the following lemma. Its proof is omitted since it is immediate from the algorithm’s description in Figure 4.5.

Lemma 4.5.4 *When executing line 5 of DYNAMIC-DISPENSE for the k -th time, then $M[i] \neq \emptyset$ if and only if i can be paid with k coins.*

Optimality for ALLOCATE-GENERALIZED is now proven in the lemma below. Although we follow the same lines of Lemma 4.4.4 the proof is more complicated here. Specifically, we first need to prove that the basic algorithm, i.e., before the remainder is handled, is optimal; this can be done with very few modifications of the proof for unrestricted denominations and is essentially completed in Claim B below. In addition, we need to show that combining the basic algorithm with the dynamic algorithm still yields an optimal result; in order to achieve this we “break” the analysis of any competing algorithm in two parts: the first consists of an algorithm allocating an amount equal to the one ALLOCATE-GENERALIZED allocates before the remainder is handled, and the second allocates the remainder. Note that this separation is well

defined, in the sense that it does not limit the set of competing algorithms—as explained in the proof. We then show that regardless of how any competing algorithm is “split” into these two parts, ALLOCATE-GENERALIZED uses at worst an equal number of coins.

Lemma 4.5.5 *Let \mathcal{T} be any solution for the k -payment problem with allowed denominations \mathcal{D} , with largest denomination n and d_i i -coins for $1 \leq i \leq n$. Then $\sum_{i=1}^m c_i \leq \sum_{i=1}^n d_i$.*

Proof. Let \mathcal{T}^* be any subset of solution \mathcal{T} , solving the k -payment problem with allowed denominations \mathcal{D} where the budget is T_m^* , i.e., the amount allocated by ALLOCATE-GENERALIZED before the remainder is handled. There must exist at least one such subset, since the (T_m^*, k) -payment problem is a subset of the (\mathcal{N}, k) -problem; i.e., any request for k payments whose sum is $T_m^* \leq \mathcal{N}$ should be satisfiable by \mathcal{T} .

We first prove that \mathcal{T}^* allocates more coins than the part of ALLOCATE-GENERALIZED up to line 13; then the proof follows from the optimality of DYNAMIC-DISPENSE.

We use the following definitions. These allow us to refer to the part of the competing algorithm (\mathcal{T}^*) that allocates the same amount as ALLOCATE-GENERALIZED before the handling of the remainder.

- $U_j = \sum_{i=1}^j id_i$.
- d_i^* , the number of i -coins allocated by \mathcal{T}^* .
- $U_j^* = \sum_{i=1}^j id_i^*$.
- n^* , the largest denomination in \mathcal{T}^* .
- m^* , the largest denomination allocated by ALLOCATE-GENERALIZED. Hence, $T_{m^*}^* = T_m^*$.
- $\Delta_0 = 0$ and $\Delta_i = \Delta_{i-1} + d_i^* - c_i^*$ for $i > 0$.

Again by the definitions, and for all $i \geq 0$:

$$\Delta_i = \sum_{j=1}^i d_j^* - \sum_{j=1}^i c_j^* \tag{4.6}$$

$$d_i^* = c_i^* + \Delta_i - \Delta_{i-1} \tag{4.7}$$

The following lemma is similar to claim A of Section 4.4. The main difference here

is that we consider only the sub-part \mathcal{T}^* of the competing algorithm; this is necessary due to the more complex handling of the remainder, which may require several coins.

Lemma 4.5.6 $\sum_{i=1}^j d_i^* \geq \sum_{i=1}^j c_i^*$ for all $1 \leq j < n^*$.

Proof. By contradiction. Suppose $\sum_{i=1}^l d_i^* < \sum_{i=1}^l c_i^*$ for $l < n^*$, and that l is the smallest such index, i.e., (using Eq. (4.6)), $\Delta_l < 0$ and $\Delta_j \geq 0$ for all $0 \leq j < l$. Hence, by Eq. (4.7), we have that $d_l^* < c_l^* - \Delta_{l-1}$, which implies, by integrality, that $d_l^* \leq c_l^* - \Delta_{l-1} - 1$. Also, $T_l^* < k(\bar{l} - 1) + l$ by (iii) of Lemma 4.5.2. Therefore, since $\bar{l} - 1 < n^*$, we get

$$\begin{aligned}
U_{\bar{l}-1}^* = U_l^* &= U_{l-1}^* + ld_l^* \\
&\leq \sum_{i=1}^{l-1} id_i^* + lc_l^* - l\Delta_{l-1} - l \\
&= \sum_{i=1}^{l-1} i(c_i^* + \Delta_i - \Delta_{i-1}) + lc_l^* - l\Delta_{l-1} - l \\
&= \sum_{i=1}^{l-1} ic_i^* + \sum_{i=0}^{l-2} (i\Delta_i - (i+1)\Delta_i) + (l-1)\Delta_{l-1} + lc_l^* - l\Delta_{l-1} - l \\
&= T_l^* - \sum_{i=1}^{l-1} \Delta_i - l \\
&\leq T_l^* - l \\
&< k(\bar{l} - 1) + l - l \\
&= k(\bar{l} - 1),
\end{aligned}$$

i.e., $U_{\bar{l}-1}^* < k(\bar{l} - 1)$, contradiction to Theorem 4.3.1, since $\bar{l} - 1 < n^*$. \square

A corollary of the above lemma is that if the competing algorithm used a denomination higher than ours, then it would have to fill in the ‘‘gaps,’’ as required by the necessity condition of Theorem 4.3.1, thus over-running the available budget. This is summarized in the following:

Corollary 4.5.7 $n^* \leq m^*$.

Proof. Suppose not. Then, by lemma 4.5.6, $U_{n^*}^* = \sum_{i=1}^{n^*} id_i^* > \sum_{i=1}^{m^*} id_i^* \geq \sum_{i=1}^{m^*} ic_i^* = T_{m^*}^* = T_m^*$; a contradiction, since by definition $U_{n^*}^* = T_{m^*}^* = T_m^*$. \square

Also notice that $m^* \in \{m, \underline{m}\}$; this is immediate from lines 4, 13 and 15 of ALLOCATE-GENERALIZED.

Next we prove that no solution can use a denomination larger than the largest one used by S .

Lemma 4.5.8 $n \leq m$.

Proof. We distinguish two cases: $m^* = m$ and $m^* = \underline{m} < m$.

If $m^* = m$ then the “while” loop of ALLOCATE-GENERALIZED terminated with line 11, hence the remainder allocated by DYNAMIC-DISPENSE is $\mathcal{N} - T_m^* < \bar{m}$. Thus $n < \bar{m} \implies n \leq m$, since there is not enough remainder to allocate a denomination larger than m .

If $m^* = \underline{m} < m$ then the “while” loop of ALLOCATE-GENERALIZED terminated with line 13, thus on the last iteration the statement of line 8 was false. That is,

$$\left\lfloor \frac{\mathcal{N} - T_{m^*}^*}{m} \right\rfloor < \left\lceil \frac{k(\bar{m} - 1) - T_{m^*}^*}{m} \right\rceil ,$$

and since $T_{m^*}^* = U_{n^*}^*$, the above becomes

$$\left\lfloor \frac{\mathcal{N} - U_{n^*}^*}{m} \right\rfloor < \left\lceil \frac{k(\bar{m} - 1) - U_{n^*}^*}{m} \right\rceil ,$$

which if we remove the ceiling and floor becomes

$$\frac{\mathcal{N} - U_{n^*}^* - m}{m} = \frac{\mathcal{N} - U_{n^*}^*}{m} - 1 < \frac{k(\bar{m} - 1) - U_{n^*}^*}{m} ,$$

and after simplification,

$$\mathcal{N} - m < k(\bar{m} - 1) .$$

Now if $n > m$, then at least one coin of denomination $n > m$ is allocated by \mathcal{T} , hence $\mathcal{N} \geq n + U_m > m + U_m$, or $U_m < \mathcal{N} - m$. Then the above becomes $U_{\bar{m}-1} = U_m < k(\bar{m} - 1)$; a contradiction to theorem 4.3.1 since $\bar{m} - 1 < \bar{m} \leq n$.

Therefore in either case $n \leq m$. □

Hence, if we show that \mathcal{T}^* allocates at least as many coins as ALLOCATE-GENERALIZED for $U_{n^*}^*$ then the proof is complete. This is because the remainder $(\mathcal{N} - U_{n^*}^*)$ is optimally allocated, since the competing algorithm \mathcal{T} has an upper bound n on allowed denominations, which is less or equal than the upper bound m of DYNAMIC-DISPENSE.

Now, if $n^* = m^*$, from claim B \mathcal{T}^* allocates at least as many coins as ALLOCATE-GENERALIZED for $U_{n^*}^*$, and we are done.

So, from lemma 4.5.8, the only case left is when $n^* < m^*$. This is handled similarly to Lemma 4.4.4. Let $b = \sum_{i=n^*}^{m^*} c_i^* - d_{n^*}^*$. We again prove by showing that $\sum_{i=1}^{n^*} (d_i^* - c_i^*) \geq b$. By lemma 4.5.6 we have $\Delta_i \geq 0$ for $i \leq \underline{n}^*$, and from Eq. (4.7) we get

$$\begin{aligned} U_{\underline{n}^*}^* - T_{\underline{n}^*}^* &= \sum_{i=1}^{\underline{n}^*} i(d_i^* - c_i^*) \\ &= \sum_{i=1}^{\underline{n}^*} i(\Delta_i - \Delta_{i-1}) \\ &= \underline{n}^* \Delta_{\underline{n}^*} - \sum_{i=0}^{\underline{n}^*-1} \Delta_i \\ &\leq \underline{n}^* \Delta_{\underline{n}^*} . \end{aligned}$$

Then from $U_{n^*}^* = \mathcal{N} = T_{m^*}^*$, we have

$$\begin{aligned} U_{\underline{n}^*}^* &= U_{n^*}^* - n^* d_{n^*}^* \\ &= T_{m^*}^* - n^* \left(\sum_{i=n^*}^{m^*} c_i^* - b \right) \\ &\geq T_{m^*}^* - \sum_{i=n^*}^{m^*} i c_i^* + b n^* \\ &= T_{\underline{n}^*}^* + b n^* . \end{aligned}$$

Hence,

$$\begin{aligned} n^* \Delta_{\underline{n}^*} &\geq \underline{n}^* \Delta_{\underline{n}^*} \\ &\geq U_{\underline{n}^*}^* - T_{\underline{n}^*}^* \\ &\geq T_{\underline{n}^*}^* + b n^* - T_{\underline{n}^*}^* \\ &= b n^* , \end{aligned}$$

i.e., $\Delta_{\underline{n}^*} \geq b$, and from Eq. (4.6)

$$\sum_{i=1}^{\underline{n}^*} (d_i^* - c_i^*) = \Delta_{\underline{n}^*} \geq b ,$$

as required. □

4.6 Achieving unlinkable divisible e-cash

It is now easy to see how unlinkable divisible e-cash can be constructed using our optimal algorithms for coin allocation and dispensing. A user decides the amount he wishes to withdraw, the minimum payment unit, and the number k of exact payments he will need. Then the divisibility precision $\mathcal{N} = (\text{total coin value})/(\text{smallest divisible unit})$ is calculated; this is the user's *budget* \mathcal{N} (i.e., the budget is counted in terms of the smallest units). In any given system the smallest divisible unit (or *reference unit*) is usually predetermined; however the bank may use different public/secret key pairs to denote signatures on coins based on various reference units, just as distinct signatures denote various denominations.

Once the parameters are set, the user withdraws coins of appropriate denominations, based on the coin allocation algorithm, using the withdrawal protocol of a non-divisible e-cash scheme (e.g., Brands' [15]); each denomination uses a different public key of the bank. Coins are then paid—based on the coin dispensing algorithm—again using the regular payment protocol for each coin.

In the resulting system users withdraw an amount \mathcal{N} which can be divided to allow for at least k exact payments, in such a way that all payments originating from the same coin are unlinkable (since the non-divisible coins are unlinkable); hence an *unlinkable divisible e-cash* system is achieved.

We must note that the multitude of coins that must be withdrawn, stored and paid, result in excessive computation, communication and storage requirements: approximately $k \cdot \ln \frac{\mathcal{N}}{k}$ times those needed for a single non-divisible coin. This seems to be necessary since coins must have no common parts—or else linking is trivial. However, requirements can be notably reduced by allowing the withdrawn coins to be linkable since in this case common parts can be used for distinct coins. This approach is discussed in the next chapter, as it results in *linkable divisible e-cash*.

4.7 Other applications

Another interesting application of the k -payment problem arises in the context of resource sharing. For concreteness, consider a communication link whose total bandwidth is \mathcal{N} . When the link is shared by time-multiplexing, there is a fixed schedule which assigns the time-slots, such as cells in ATM lines, to the different connections. Typical schedules have small time slots, since assigning big slots to small requests entails under-utilization. It is important to note, however, that there is an inherent fixed overhead associated with each time slot, e.g., the header of an ATM cell. It would be therefore desirable to have a multiplexing schedule with time slots of various sizes which can accommodate any set of requests with the least number of slots. Similarly to the withdrawal scenario, an improvement upon the trivial \mathcal{N} unit-slot solution can be achieved, if we know how many connections might be running in parallel. When we know a bound k on this number, and if we restrict ourselves to long-lived connections, the problem of designing a schedule naturally reduces to the k -payment problem.

Chapter 5

Linkable divisible e-cash

In the previous chapter we presented an exact analysis of the number and type of coins needed to emulate divisible e-cash with unlinkable coin portions (*unlinkable divisible e-cash*) under reasonable restrictions on the number of payments. However, as pointed out earlier, a collection of unlinkable coins occupies a prohibitive amount of memory and entails high computation and communication costs. For many practical applications it is essential to use a divisible e-cash system with minimal such requirements, even if this is achieved by allowing linkability among coin portions. Linkability diminishes user anonymity but since in this case it is limited *within* a coin it does not represent a significant threat. Users are free to balance unlinkability between coin portions and the number of coins that they carry by withdrawing the appropriate coin denominations. Coins of high denominations usually result in increased linkability—since presumably more payments will be conducted with them—and vice versa. Hence, realistic solutions can be found; e.g., users may withdraw a divisible coin that they expect to spend in the course of one day, thus limiting linkability while always carrying only a single coin in their electronic wallets.

In this chapter we investigate efficient, but nevertheless provably secure under some defined cryptographic assumptions, linkable divisible electronic cash systems. Section 5.1 discusses the two possible methods in achieving such systems, and their respective properties. We investigate both, by proposing one scheme for each method; in conjunction with our analysis of unlinkable divisible e-cash in the previous chapter, we believe that this represents a comprehensive view of all the currently available ways of providing exact payments in a secure and efficient manner.

The first system, presented in Section 5.1.1, is a corollary of our results in the previous chapter, and is thus provably optimal—for this particular method. The majority of the chapter is devoted in analyzing the second scheme, which is the first truly practical system based on this method, constituting a major improvement over previous proposals. Section 5.2 presents the building blocks for this system, which is then described in Section 5.3.

An alternative way to view the work in this chapter is that we present the most efficient—to date—system achieving unbounded divisibility precision. However, to provide a lower bound for the efficiency requirements of such a system, we also emulate this approach with a system providing bounded divisibility and number of possible payments, since realistic bounds may result in an emulation that is equally practical to the unbounded system; we note that we are the first to investigate this alternative. Furthermore, the emulation we achieve is provably optimal, as it is based on the results of the previous chapter and uses as a building block the most efficient e-cash scheme to date; however, note that *any* e-cash scheme can be used as a building block. Under this prism, Section 5.4 compares the two approaches and analyzes the threshold, which depends on the divisibility precision and number of payments, beyond which the unbounded system is preferable in terms of efficiency. It is interesting to note that this threshold is not trivial, thus—depending on application requirements—either of the two systems may be optimal.

Section 5.5 provides a universal extension to electronic cash schemes with which, if a coin is double spent due to hardware failure or unintentional oversight, the coin cannot be spent again by someone who has transcripts of both payment transactions. This “framing protection” capability is not provided in all existing e-cash schemes, but our solution is universal and very efficient and can be embedded in any system.

The chapter is concluded with Section 5.6 which summarizes and discusses open problems.

5.1 Approaches

As analyzed in Section 3.2 there are two ways for constructing linkable divisible e-cash: either withdraw a multitude of *linkable* non-divisible coins using the algorithms in the previous chapter, or withdraw a single divisible coin. The former algorithm

can be combined with any e-cash system and turns out to be the optimal solution if the divisibility precision and the number of required payments are within some limits. A divisible coin, however, can provide practically unlimited divisibility precision and number of payments; hence there is clearly a threshold above which it is the preferable solution.

In this section we initiate the investigation of both approaches. In Section 5.1.1 we apply our analysis of the previous chapter to Brands' scheme—one of the fastest e-cash schemes to date—modified to allow for withdrawals/payments of linkable coins; allowing such linkability significantly reduces computation and communication overhead, while storage requirements are reduced to levels similar to that of a single or divisible coin. Then, in Section 5.1.2, we describe the alternative method of using a fast divisible e-cash system.

5.1.1 Using non-divisible coins

In this section we show how linkable divisible e-cash can be emulated using the result of our previous chapter. In contrast with unlinkable divisible e-cash, the portions of a coin are now allowed to be linkable; hence, here we present a scheme which given a divisibility precision \mathcal{N} and a maximum number k of exact payments, allows the user to withdraw linkable coins of the appropriate denominations and spend them at payment. The goal is to take advantage of the linkability to minimize the computation, communication and storage requirements for given (\mathcal{N}, k) .

Ordered n -spendable coins

Notice that, if denominations are not an issue, a multitude of linkable coins is equivalent to a single coin that can be spent a fixed, multiple number of times, n . This we define as an *n -spendable coin*. But for a given \mathcal{N} and k , the optimal algorithm of Section 4.4 provides a set of denominations in a deterministic way. Hence, if we can determine the ordering of the payments of the n -spendable coin, i.e., if the shop can verify that the l -th occurrence of the coin is spent for every $1 \leq l \leq n$, then this coin can emulate a multitude of linkable coins of varying denominations: to illustrate a sample case, the first k occurrences of the coin will be of denomination 1, the next $k/2$ of denomination 2, etc. We call such n spendable coins *ordered n -spendable coins*, while we refer to the i -th occurrence for all $1 \leq i \leq n$ as the *i -th instance* of this coin.

In the remainder of this section we show how Brands' scheme can be extended to allow for ordered n -spendable coins; using this extended system for linkable divisible e-cash is then straightforward and a description is omitted. However, calculations of the efficiency of such a system are included at the end of this section and are utilized in the comparison with divisible e-cash in Section 5.4.

Extending Brands' scheme to ordered n -spendable coins

Our choice of Brands' scheme is made due to its efficiency and reasonable security guarantees, but any other e-cash scheme can be similarly extended. We refer the reader to Section 3.1.3 for a detailed description of [15] and we remind the reader that this particular system has been recently broken; the necessary security addition, described in detail in Section 5.2.2 below, is directly applicable to our modification and accounted for in our efficiency calculations.

The coin in Brands' scheme consists of two numbers, $A = (Ig_2)^s, B = g_1^{x_1}g_2^{x_2}$, where I is the user's identity, $s \in_R Z_Q^*$ the user's secret blinding number and B is a random number used for the Schnorr proof of knowledge of the representation of A . B is included in the coin's signature so that if \mathcal{U} double-spends he conducts two Schnorr proofs using the same random number and hence reveals s .

A straightforward way to expand this system to withdraw an ordered n -spendable coin is to have n distinct B 's, one for each payment, i.e., the coin is composed of (A, B_1, \dots, B_n) and the signature of the bank on (A, B_1, \dots, B_n) . The user computes these B_i 's at withdrawal and includes them in the coin's signature; their order in the signature determines the instance spent. If one of these coin instances is double-spent the user is identified as in the original scheme.

There are other ways to perform the extension, e.g., by including more generators [14]. However the above solution seems to be the most efficient and we thus analyze it here.

Additions for user \mathcal{U} at withdrawal:

We assume that \mathcal{U} has access to a Pseudorandom Generator (PRG); random oracles are themselves PRGs, hence this assumption does not weaken the security of the scheme as random oracles are necessary for [15]'s security also. The seed of the generator is of size k_1 , where k_1 is a security parameter.

User \mathcal{U} performs the following:

- **Construct B_i , for $1 \leq i \leq n$.** Pick a random seed $t \in_R \{0, 1\}^{k_1}$ and run the PRG to obtain a string X of length $2n|Q|$, where Q is the order of the generators g_1, g_2 in the multiplicative group Z_P^* . Then each substring of X of length $|Q|$ is used as an exponent in the construction of B_i 's. Here, $X_{i,\dots,j}$ with $0 \leq i, j \leq |X|$, denotes bits i through j of X :

$$B_i = g_1^{X_{2(i-1)|Q+1,\dots,(2i-1)|Q|}} g_2^{X_{(2i-1)|Q+1,\dots,2i|Q|}} ,$$

for all $1 \leq i \leq n$.

- **Insert B_i 's into hash function.** The string B in the hash value $c = \mathcal{H}(A, B, z, a, b)$ that is used for the coin's signature now becomes

$$B = (\mathcal{H}'(B_1), \mathcal{H}'(B_2), \dots, \mathcal{H}'(B_n)) ,$$

where $\mathcal{H}' : \{0, 1\}^{|P|} \mapsto \{0, 1\}^{k_2}$ is a random oracle-like hash function and k_2 a security parameter. Here, k_2 is allowed to be smaller than k_1 , since a potential collision will allow the user to double-spend a single instance of the divisible coin rather than the whole coin, which is what happens if he obtains a collision on the hash function \mathcal{H} . Alternatively, we could link k_2 to the size of k_1 ; the current description is however more general.

The payment protocol:

- **\mathcal{U} : Spend the i -th instance.** Send $A = g_1^{u_1 s} g_2^s, B_i = g_1^{x_1^i} g_2^{x_2^i}$ and

$$r_1^i = d_i u_1 s + x_1^i, \quad r_2^i = d_i s + x_2^i ,$$

for challenge $d_i = \mathcal{H}_0(A, B_i, ID_S, \text{Date/Time})$ computed as in the original scheme.

- **\mathcal{U} : Prove construction of B .** Send $\mathcal{H}(B_{i'})$, for all $i' \neq i$ and the coin's signature (z, a, b, r) .
- **\mathcal{S} : Verify:**

- the construction of B , i.e.,

$$B = (\mathcal{H}'(B_1), \mathcal{H}'(B_2), \dots, \mathcal{H}'(B_n)) ,$$

- the signature (z, a, b, r) on the coin (A, B) , i.e.,

$$g^r = h^{\mathcal{H}(A, B, z, a, b)}_a \quad \text{and} \quad A^r = z^{\mathcal{H}(A, B, z, a, b)}_b ,$$

- and that

$$A^{d_i} B_i = g_1^{r_1^i} g_2^{r_2^i} ,$$

for every spent instance i .

Security is identical to [15], with identification of a user double-spending a coin instance performed as in the original scheme.

Efficiency

We now examine the efficiency of the above scheme. We parameterize the system's requirements for \mathcal{N}, k , while we assume that a modulus P of 512 bits is used and that $|Q| = 160$. The seed to the PRG is of length 160 bits, i.e., $k_1 = 160$, whereas the output of the hash function \mathcal{H}' is assumed to be 80 bits long, i.e., $k_2 = 80$ and the user would have to perform $O(2^{40})$ exponentiations to find a collision. These parameters are suitable for a direct comparison with our divisible scheme appearing in Section 5.4.

Based on our analysis in the previous chapter the number of coin instances, n , is

$$kH_{\lfloor \mathcal{N}/k \rfloor} \leq n \leq (k+1)H_{\lfloor \mathcal{N}/(k+1) \rfloor} .$$

Here we assume for simplicity that $n = kH(\mathcal{N}/k) \approx k \ln \frac{\mathcal{N}}{k}$. If we set $\alpha = \ln \frac{\mathcal{N}}{k}$ this becomes

$$n = k\alpha .$$

We are now ready to proceed with efficiency calculations.

Storage:

The user stores A, z, a, b , each of size $|P| = 512$ and r, c , each of size $|Q| = 160$, as in the original Brands' scheme. In addition the user stores the seed for the PRG and the hash values of the strings B_i , for all $1 \leq i \leq n$; this is needed to free the user from having to recalculate all these values at every payment. Hence the total storage requirements are

$$4 \cdot 512 + 2 \cdot 160 + 160 + 80(k\alpha) \text{ bits} = 316 + 10k\alpha \text{ Bytes.}$$

Withdrawal:

Computation for the user entails one exponentiation for proving account ownership, plus calculating A (one exponentiation), z, a, b and the B_i 's. Each of the latter

requires one “multi-exponentiation”, i.e., an exponentiation of the form $g_1^a g_2^b$; this costs approximately 1.2 exponentiations [84, 100]. Hence the user performs

$$5.6 + 1.2n = 5.6 + 1.2k\alpha \text{ exponentiations modulo } P, \text{ with exponents of size } |Q|.$$

Each such exponentiation requires $O(|Q|)$ modular multiplications where the size of each multiplied number is $|P|$.

In this estimate, as well as the ones in the remaining of this thesis, we ignore the computation needed for calculating the hash values, running the PRG, or performing simple modular additions and multiplications, since these tasks are much faster than modular exponentiations.

Communication at withdrawal is the same as in [15], i.e., the user sends to the bank I, y' of size $|P|$ and r', r_e of size $|Q|$, while the bank sends a', b' of size $|P|$ and e, r of size $|Q|$; here, y', r_e, e are used to prove account ownership via a Schnorr proof of knowledge protocol (see Section 5.2.2, “How to repair the scheme” for a description). Hence both the user and the bank send 168 Bytes to each other.

Payment:

The average number of coin instances spent per payment are $n/k = (k\alpha)/k = \alpha$.

The user *computes* the B_i for the instances to be spent; this takes one multi-exponentiation, i.e., approximately 1.2 exponentiations. Here we ignore the computation for running the PRG, computing the hash functions as well as performing simple modular additions and multiplications. Hence the user performs 1.2α exponentiations, on the average. Computation for the shop is approximately the same, entailing one multi-exponentiation per coin instance for coin verification, plus verifying the construction of B by calculating the hash function.

Communication consists of sending the coin A and its signature (z, a, b, r) , the hash values for each B_i for all $1 \leq i \leq n$, as well as a B_i and two responses, r_1^i, r_2^i , to challenges for each spent instance. Hence, the total amount of communication is

$$512 + 3 \cdot 512 + 160 + n80 + \alpha(512 + 2 \cdot 160) \text{ bits} = 276 + \alpha(84 + 10k) \text{ Bytes.}$$

The communication from the shop to the user can be ignored since the protocol is allowed to be non-interactive.

5.1.2 Achieving unbounded divisibility

Although withdrawing a multitude of ordered n -spendable coins allows users to conduct exact payments, the storage, communication and computation requirements for the participants increase with the divisibility precision, \mathcal{N} , and the number of required payments, k . Hence, in settings where either of these two parameters is sufficiently large a divisible e-cash scheme whose requirements would change at most logarithmically with \mathcal{N} and k would be preferable. Such a scheme can provide practically unlimited divisibility precision and number of payments. *Furthermore, the precision need not be fixed at withdrawal; instead a coin can be divided in arbitrarily small portions upon a payment request.*

As a starting point for our divisible e-cash we utilize Okamoto's [100] scheme, as this is the first practical divisible off-line e-cash scheme, requiring only $O(\log \mathcal{N})$ computation and communication for each of the withdrawal, payment and deposit protocols; this scheme is described in detail in Section 3.2.2. However, Okamoto's set-up procedure in which an anonymous electronic "license" is issued to a user to be used at withdrawal time is quite inefficient and highly depends on the size of the RSA modulus. Hence, as we explained in Section 3.2.2, user coins are linkable, since unlinkability would entail one execution of the costly "account-establishment" protocol per coin. Thus, as Okamoto suggests, "the frequency of license changes should be determined after considering trade-offs between the degree of unlinkability and efficiency desired". These trade-offs essentially force the same license to be used multiple times, thus increasing the user's vulnerability to be traced by other means, such as correlating payments' localities, time, type, etc..

Here, we construct the first truly efficient divisible e-cash system by presenting a unique combination of Okamoto's scheme [100], Brands' [15] e-cash scheme and an efficient *range-bounded commitment* protocol. The main idea is to substitute the inefficient account establishment protocol with one that is 3 orders of magnitudes faster, making it almost as efficient as Brands' withdrawal protocol. Hence, the functionality of the account establishment can be transferred to the withdrawal, allowing coins to be *unlinkable*, as required by our security model for off-line e-cash (presented in Section 3.1.1). The resulting scheme has minimal storage, computation and communication requirements, allowing for its implementation in currently available smart-cards. In addition, proofs of security conforming to our formal model are given,

based on clarified security assumptions.

5.2 Building blocks

To improve the efficiency of Okamoto’s account establishment protocol, we need two basic tools: one that proves that a number is within a certain range (this was pointed out to us by Okamoto [101] who identified an error in our original proposal) and one that blindly signs the product of two committed numbers. For the first, we formalize the notion of a *range-bounded commitment*, in Section 5.2.1, and provide an efficient instantiation based on the DLA. For the second, we utilize a unique variant of Brands’ scheme.

As mentioned in Sections 3.1.3 and 3.2.2, however, we have recently showed [24, 22] that both Okamoto’s and Brands’ e-cash systems can be broken. Thus, before originating the description of our scheme, we describe in Sections 5.2.3 and 5.2.2 the necessary corrections to these underlying protocols.

5.2.1 Range-bounded commitment

The idea of checking whether a number committed using discrete log is in a specific range was first proposed by T. Okamoto in [100] for applicability in electronic cash. Protocols for instantiation of this idea were also presented in [100], as well as subsequent work by the same author [101]. We propose to call such protocols *range-bounded commitments*.

Here we formalize the notion of a range-bounded commitment; informally, this is a protocol between a prover, \mathcal{P} , and a verifier, \mathcal{V} , with which \mathcal{P} can commit to a string, x , and prove to \mathcal{V} that x is in a predetermined range, H , with an accuracy δ . Hence the protocol uses a secure bit commitment as a building block (see Section 2.2.5 for a formal definition of bit commitment); then, with overwhelming probability in the security parameter k , \mathcal{V} is convinced that $|x| \leq (1 + \delta)H$, while \mathcal{P} can conduct an efficient proof as long as $0 \leq x < 2^H$. In the process a small amount of information about x is leaked to \mathcal{V} ; this is negligible in k . Formally,

Definition 5.2.1 (range-bounded commitment) *A range-bounded commitment scheme consists of a pair of probabilistic polynomial-time interactive machines, denoted $(\mathcal{P}, \mathcal{V})$ (for prover and verifier), satisfying:*

- **Input specification:** *The private input to the prover is a string x for which $0 \leq x < 2^H$. The common input is a bit commitment $b(x)$ on x (formally defined in Section 2.2.5), an integer n presented in unary (the security parameter), and an integer H (the specified range).*
- **Completeness:** *The prover can prove that x is in the specified range. Namely, for every x such that $0 \leq x < 2^H$ and for every polynomial $p(\cdot)$ and sufficiently large n*

$$\text{Prob}[(\mathcal{P}, \mathcal{V})(b(x)) = 1] > 1 - \frac{1}{p(n)} ,$$

where the probability is taken over the coin tosses of \mathcal{P} and \mathcal{V} .

- **Soundness:** *The verifier is convinced that x is in the range of H . Namely, there exists a number $\delta \geq 0$ (the accuracy), such that for every probabilistic polynomial-time machine \mathcal{P}^* (a cheating prover), for every x such that $|x| > (1 + \delta)H$ and for every polynomial $p(\cdot)$ and sufficiently large n*

$$\text{Prob}[(\mathcal{P}^*, \mathcal{V})(b(x)) = 1] < \frac{1}{p(n)} ,$$

where the probability is taken over the coin tosses of \mathcal{P}^* and \mathcal{V} .

- **Secrecy:** *The verifier does not obtain any information about x , other than $b(x)$: for every probabilistic polynomial-time machine \mathcal{V}^* (a cheating verifier), there exists a probabilistic polynomial-time machine \mathcal{M}^* (a simulator) so that with probability overwhelming in n the following two ensembles are polynomially indistinguishable*

- $\{(\mathcal{P}, \mathcal{V}^*)(b(x))\}_{0 \leq x < 2^H}$ (i.e., the output of \mathcal{V}^* when interacting with \mathcal{P} on any legitimate input $b(x)$), and
- $\{\mathcal{M}^*(b(x))\}_{0 \leq x < 2^H}$ (i.e., the output of machine \mathcal{M}^* on $b(x)$),

where the probability is taken over the coin tosses of $\mathcal{P}, \mathcal{V}^*$ and \mathcal{M}^* .

Efficient instantiation

We now present an efficient range-bounded commitment protocol based on the DLA, which has originally appeared in [26]. The main idea is for the verifier to challenge the prover with a random value; then the prover has to produce a linear combination based on this random value and x , blinded based on a random number chosen and

committed by him earlier, such that the result is within a certain range. The verifier also checks the correctness of the linear equation using the committed values for x and the prover's random number and his own random challenge. This technique resembles a Schnorr proof of knowledge, but new techniques are necessary as highlighted below.

An interesting point is that while computations are normally performed modulo a prime Q —since this is a DLA-based scheme—computations involving exponents are not performed modulo Q but on the integers, so that the range of the numbers can be checked. Based on x , the choice of the prover's blinding number and the verifier's random challenge, there is a small probability of failure for the protocol; i.e., for a valid x the prover's answer may be outside the specified range. Similarly, since exponents are not computed modulo Q , there is a small amount of information leaked to the verifier, relating to the range of x . However, as we analyze below, both the probability of failure and the probability of \mathcal{V} to extract some information about x are negligible in the protocol's security parameter. Hence the protocol is a range-bounded commitment, satisfying our definition above.

Setup:

Define security parameters H, S, n , where H, n conform to the definition of the range-bounded commitment above, while S is the desired length of the output of the random oracle-like hash function if the protocol is made non-interactive. As shown by Bellare and Rogaway in [8] if the hash function is a random oracle then it is sufficient to set $S = n \cdot (1 + \epsilon)$, where ϵ is a positive constant; however, we treat S here as a separate parameter in order to allow for greater flexibility in the choice of the hash function. Thus, if a weaker function is used, the system can be tuned to allow for a greater output length.

Compute the accuracy $\delta > (2n + 2)/H$; for simplicity of notation this is chosen so that δH is an integer. Given primes Q, P with $P = 2Q + 1$, $|Q| = 2(1 + \delta)H + 6$, and the commitment $X \equiv g^x \pmod{P}$, prover \mathcal{P} will prove to verifier \mathcal{V} that $|x| \leq (1 + \delta)H$.

The protocol:

- (1) \mathcal{P} picks $u_i \in_R \{0, \dots, 2^{(1+\delta)H} - 1\}$, and sends $U_i = g^{u_i}$ to \mathcal{V} .
- (2) \mathcal{V} sends $e_i \in_R \{0, \dots, 2^n - 1\}$.
- (3) \mathcal{P} responds with $u'_i = e_i x + u_i$.
- (4) \mathcal{V} verifies $g^{u'_i} = X^{e_i} U_i$ and $0 \leq u'_i < 2^{(1+\delta)H}$.

The protocol can be collapsed to one move if $e = e_1 \dots e_j = \mathcal{H}_0(X, U_1, \dots, U_j)$, where j is the number of iterations and \mathcal{H}_0 is a random oracle. In this case, it must be executed $j = \lceil S/n \rceil$ times, to guarantee that $|e| \geq S$. When \mathcal{H}_0 is a random oracle $S = 2n$ is sufficient, as shown by Bellare and Rogaway in [8].

Theorem 5.2.1 *Assuming $g^x \pmod{P}$ is a secure bit commitment on x , the above protocol is a range-bounded commitment.*

Proof. The assumption of the theorem is stronger than the DLA, but it is used only for technical reasons. Namely, security can be proven based solely on the DLA as long as $X \equiv g^x \pmod{P}$ hides all information about x ; in reality, however, even assuming the DLA, some minimal information about x is revealed given X . For applicability in our e-cash scheme this leak of information has no impact, since X is already known to \mathcal{V} ; hence the range-bounded commitment protocol can be simplified by introducing this stronger assumption.

Input specification:

The private input to \mathcal{P} is x with $0 \leq x < 2^H$. The security parameters are n, H . The additional parameter S can be treated as an auxiliary input or can be a function of n . The bit commitment $b(x)$ on x is $X \equiv g^x \pmod{P}$.

Soundness:

Notation: For the duration of this proof, $|\alpha|$ denotes the absolute value of a α —not the size of α , as in the rest of this thesis.

Note that, under the DLA, in each iteration the prover has a probability of $1/2^n$ of incorrectly convincing the verifier that x is in the specified range, by selecting $u_i = -e_i x + u'_i$ for some $u'_i \in_R \{0, \dots, 2^{(1+\delta)H} - 1\}$. To see this, assume that x is not in the correct range, that is,

$$|x| > 2^{(1+\delta)H} = H', \tag{5.1}$$

where H' denotes $2^{(1+\delta)H}$ for simplicity of notation.

Further, suppose that in iteration i there is a challenge e_i for which the verification succeeds, i.e.,

$$0 \leq u'_i = u_i + e_i x < H'. \tag{5.2}$$

Note that once \mathcal{P} has committed to $U_i = g^{u_i}$ he cannot obtain a $v_i \not\equiv u_i \pmod{Q}$, such that $g^{v_i} \equiv V_i \equiv U_i \pmod{Q}$, or else he could break the Discrete Logarithm

Problem, a contradiction to the theorem's assumption. Hence u_i is fixed. Thus, for any other challenge $e'_i \in \{0, \dots, 2^n - 1\}$ with $e'_i \neq e_i$, the response would be

$$u''_i = u'_i + e''_i x, \text{ with } e''_i = (e'_i - e_i) \neq 0.$$

Intuitively, by adding, or subtracting, depending on the signs of e''_i and x , a number greater than H' from a number smaller than H' , we should get a number greater than H' or smaller than 0, i.e., the verification would fail. Thus, the user can only succeed in proving an incorrect x if he guesses the challenge e_i , with a probability of 2^{-n} .

Formally, equation (5.1) above can be rewritten as

$$H' < x < Q - H',$$

where, since we are working mod Q , $Q - H' \equiv -H'$. By multiplying the above with $e''_i = (e'_i - e_i)$ we get

$$e''_i H' < e''_i x < e''_i (Q - H') \equiv Q - e''_i H' \pmod{Q},$$

if $e''_i > 0$, or, if e''_i is a negative integer,

$$\begin{aligned} e''_i H' > e''_i x > Q - e''_i H' &\iff \\ |e''_i| H' = (-e''_i) H' < (-e''_i) x = |e''_i| x < Q - (-e''_i) H' = Q - |e''_i| H', \end{aligned}$$

where the equation holds for every possible (i.e., non-zero) value of e''_i . Finally, by adding u'_i we get

$$H' \leq u'_i + |e''_i| H' < u'_i + |e''_i| x = u''_i < Q - |e''_i| H' + u'_i \leq Q,$$

from equation (5.2) and since $|e''_i| H' + u'_i \leq 2^{(1+\delta)H+k} + 2^{(1+\delta)H} < Q$, from the selection of Q . This however is equivalent to

$$u''_i < 0 \text{ and } u''_i > H' \pmod{Q} \iff u''_i < 0 \text{ or } u''_i > H',$$

hence the verification for u''_i would fail.

Completeness:

Conversely, the probability that a legitimate \mathcal{P} , i.e., a prover selecting x such that $0 \leq x < 2^H$, fails to convince \mathcal{V} in some iteration i is the probability that $u_i \geq 2^{(1+\delta)H} - e_i x$. Since there are $2^{(1+\delta)H}$ possible choices for u_i , while only $e_i x$

of these will cause the verification to fail, this probability is $\log_2(e_i x)/2^{(1+\delta)H}$; but $\log_2(e_i x) = 2^{n+H}$, hence the probability becomes $\leq 2^{n+H-(1+\delta)H} = 2^{n-\delta H}$. The probability of this happening in one of j rounds is $1 - (1 - 2^{n-\delta H})^j$ i.e., from the selection of δ , $< 1 - (1 - 2^{n-2})^j$. If we assume, for simplicity, that $j = 2$, then the probability of failure is $1 - (1 - 2^{n-2})^2 \approx 2^{-n-1}$.

Secrecy:

Finally, based on the analysis of [53] of a more generalized protocol, the probability that \mathcal{V} can extract some information regarding x , in addition to that revealed by $X = g^x$, is $j \cdot 2^{-\delta H+1+n} < j \cdot 2^{-n-2+1} = j \cdot 2^{-n-1} = 2^{-n}$ for $j = 2$, assuming that $X \equiv g^x \pmod{P}$ is a secure commitment on x . \square

5.2.2 Attacks and repair on Brands' scheme

In this section we describe two attacks on Brands' scheme as presented in [15] (see also Section 3.1.3), which allow users to mis-represent their identity at account establishment and thus double-spend in an undetectable manner, defeating the most essential security aspect of the scheme. The attacks are also applicable to T. Eng and T. Okamoto's [46] divisible e-cash scheme (described in Section 3.2.2) which uses Brands' protocols as a building block. Along with the attacks we also present an efficient modular fix which is applicable to any use of the Brands' idea.

For a concrete understanding of the attacks, we refer the reader to Section 3.1.3 where the full Brands' scheme, as it appears in [15], is described. Here we show how a user may misbehave in the setup (account-opening) protocol in such a way as to be able to spend a coin arbitrarily many times without being identified.

We remind the reader that the signature used by the bank to sign a coin is a variation of the Schnorr signature [116]. The signature $sign(A, B)$ on the pair $(A, B) \in G_Q \times G_Q$, consists of a tuple $(z, a, b, r) \in G_Q \times G_Q \times G_Q \times Z_Q$, such that:

$$g^r = h^{\mathcal{H}(A, B, z, a, b)} a \quad \text{and} \quad A^r = z^{\mathcal{H}(A, B, z, a, b)} b \quad (5.3)$$

Attack 1: Represent I in (g_1, g_2)

At **setup**, \mathcal{U} , instead of using $I = g_1^{u_1}$, chooses $I = g_1^{u_1} g_2^{u_2}$, for random u_1, u_2 . In the case where \mathcal{B} does not compute z , \mathcal{U} can still compute z by himself given $h_1 = g_1^x$ and $h_2 = g_2^x$: $z = (I g_2)^x = (g_1^{u_1} g_2^{u_2} g_2)^x = h_1^{u_1} h_2^{(u_2+1)}$.

\mathcal{U} conducts **the withdrawal protocol** unmodified, to end up with a signature on $A = g_1^{u_1 s} g_2^{(u_2+1)s}$ and $B = g_1^{x_1} g_2^{x_2}$. \mathcal{U} can do this since he knows z , and the representation of I with respect to (*w.r.t.*) (g_1, g_2) . The signature for $(A = (g_1^{u_1} g_2^{u_2+1})^s, B = g_1^{x_1} g_2^{x_2})$ is $(z' = (g_1^{u_1} g_2^{u_2+1})^{xs}, a' = g^{wu+v}, b' = (g_1^{u_1} g_2^{u_2+1})^{wsu+sv}, r' = \mathcal{H}(A, B, z', a', b')x + wu + v)$ for s, u, v randomly chosen by \mathcal{U} . This is clearly a valid signature as it satisfies (5.3) above.

At **payment** time, the two responses supplied by \mathcal{U} to \mathcal{S} (or, in the case of [46], obtained by \mathcal{B} if \mathcal{U} over-spends) will now be:

$$r_1 = de + x_1 \quad \text{and} \quad r_2 = df + x_2$$

Where (e, f) is the representation of A w.r.t. (g_1, g_2) , i.e. $A = g_1^e g_2^f$, $e = u_1 s, f = (u_2 + 1)s$. After two payments, \mathcal{B} can obtain $e = u_1 s, f = (u_2 + 1)s$. But this last system has two equations and three unknown (to \mathcal{B}) variables, u_1, u_2, s . (Note here that extracting s by using the gcd of (e, f) is impossible since we are working modulo Q .) Hence, \mathcal{B} will not be able to compute u_1, u_2 , *even if it knew the nature of the attack*, and \mathcal{U} will remain unidentified after double-spending.

Note that even after an arbitrary number of payments \mathcal{B} cannot obtain any more information than e and f . Therefore *the user can spend the coin arbitrarily many times without being identified*.

Attack 2: Represent I in (g_1, g_2, g)

In a generalization of the previous attack, \mathcal{U} selects $I = g_1^{u_1} g_2^{u_2} g^{u_3}$ for some u_1, u_2, u_3 . \mathcal{U} obtains z from \mathcal{B} or computes $z = h_1^{u_1} h_2^{u_2+1} h^{u_3}$. We now show that \mathcal{U} is still able to obtain a signature for $A = (g_1^{u_1} g^{u_2+1})^s$ and to spend the coin multiple times without being identified.

All we need to note is that upon receipt of (a, b) from \mathcal{B} at withdrawal time, \mathcal{U} uses $b'' = b/a^{u_3} = (g_1^{u_1} g_2^{u_2+1})^w$ instead of b . The flaw explored here is the extra information ($a = g^w$) given to \mathcal{U} from the bank, which nevertheless is necessary for the protocol's completion.

How to repair the scheme

To counteract both attacks \mathcal{U} must provide \mathcal{B} with a minimal-knowledge proof of knowledge of the representation of I w.r.t. g_1 during setup. An efficient proof of

this form is the Schnorr-type identification scheme [116]. The protocol, described in Figure 5.1, has minimal complexity and should be conducted once, at setup time.

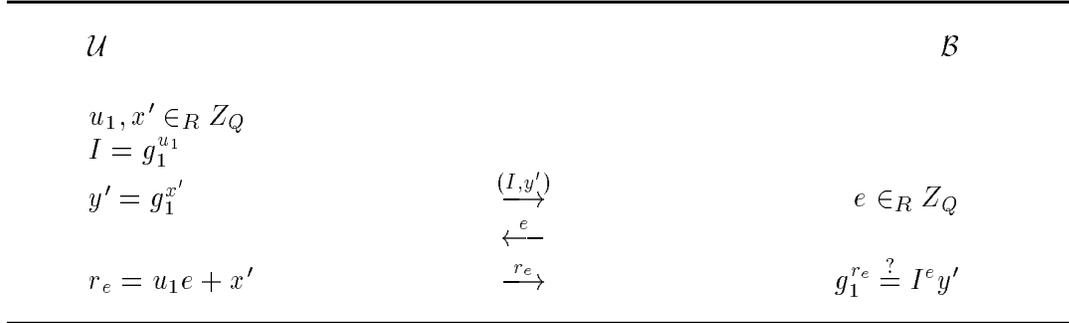


Figure 5.1: Schnorr proof of knowledge of representation of I w.r.t. g_1 .

Note that if \mathcal{U} knows a representation of I under two generator-tuples, e.g., under g_1 and under (g_1, g_2) , then \mathcal{U} could break the representation problem in groups of prime order, which is equivalent to the DLA; this is a contradiction to the scheme’s assumptions.

5.2.3 Attack and repairs on Okamoto’s scheme

In this section we present the attack on Okamoto’s [100] scheme that we identified in [22] together with the actions necessary to counteract it. First, we briefly repeat the main ideas behind this protocol; for a more complete description we refer to Section 3.2.2 or to the original paper [100].

Each user \mathcal{U} has a composite number $N = pq$ such that N is a Williams integer, i.e., with p, q primes such that $p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}$, associated with \mathcal{U} .

In **the account opening protocol** the user \mathcal{U} selects random primes p, q with $|p| = |q|$ and shows $g^p, g^q \pmod{P}$ to the bank \mathcal{B} , where the bank has selected g as a generator of the subgroup of prime order $(P - 1)/2$ of Z_P^* , with P prime such that $|P| \geq 4|p| + 10$. The bank \mathcal{B} provides the user with an “electronic license” on a number $N = pq$ after the user proves in a zero-knowledge way that N is indeed the product of p, q .

Withdrawal of the coin (performed over an authenticated channel between \mathcal{U} and \mathcal{B}) is nothing more than an RSA blind signature [27] on $H(N||b)$, where H is a hash function assumed to act as a random oracle, b is a random value, and the private RSA key used by \mathcal{B} to sign is dependent on the value of the coin.

The payment protocol (performed over an anonymous channel between \mathcal{U} and the shop \mathcal{S}) consists of the coin authentication and denomination revelation stages.

At coin authentication \mathcal{S} verifies the bank’s RSA signature on $H(N||b)$, the correctness of the electronic license on N , and that the Jacobi symbols of -1 and 2 over N are 1 and -1 respectively: $(-1/N) = 1, (2/N) = -1$. This last step guarantees that, if all of N ’s prime factors are congruent to $3 \pmod{4}$, then N has an odd (non-zero) number of prime factors which are congruent to $3 \pmod{8}$, and an odd (non-zero) number of factors congruent to $7 \pmod{8}$.

The number N defines a binary tree which serves as the divisible coin. At denomination revelation a square root mod N with Jacobi symbol of -1 is given for “the number” of each spent node. “The number” of the root is the hash value $\mathcal{H}(A, 0, B)$, while the number of its children is a square root of this, with Jacobi symbol of 1 ; and so on, with a new square root for each level of the tree. If a user spends a node, e.g., the root, and one of its children, thus violating the *root route rule*, the bank obtains two *distinct* square roots of $\mathcal{H}(A, 0, B)$, and can thus factor N . The user also reveals a random k -th square root of another hash value for each spent node so that if he double-spends a node, thus violating the *same node rule*, two distinct square roots are revealed, again allowing the bank to factor N . In addition, with these square roots \mathcal{S} indirectly checks that N has only two prime factors and that N is a Blum integer.

Note that the same N is revealed for each coin with the same license. Hence, coins can be linked. The system that we present in Section 5.3 below does not have this property.

Attacking the scheme

We now show how the user can cheat at account opening so that he can later over-spend by an arbitrary amount without being identified. To this effect we exploit a weakness in the coin authentication and denomination revelation protocols.

The verifications at account establishment, coin authentication and denomination revelation were intended to check that $N = pq$ is a Williams integer and that p, q are committed in g^p, g^q . Upon over-spending N is factored and the user is identified by linking the N at payment with g^p, g^q at account establishment. However, as we noted above, these verifications only guarantee that N

- (1) is a product of p, q , with the latter committed in g^p, g^q ,

- (2) has only two distinct prime factors, and
- (3) one of its distinct prime factors is congruent to 3 mod 8, and the other to 7 mod 8.

However, the above do not rule out the possibility that $N = p^i q^j$, where $p \equiv 3 \pmod{8}$, $q \equiv 7 \pmod{8}$, and i, j are odd integers.

Hence, the user could attack the scheme by choosing, at *account establishment*, primes $p_1 \equiv 3 \pmod{8}$, $q_1 \equiv 7 \pmod{8}$, two odd integers i, j , and two integers $i' < i$, $j' < j$, and using $p = p_1^{i'} q_1^{j'}$, $q = p_1^{i-i'} q_1^{j-j'}$. \mathcal{B} would then give \mathcal{U} an electronic license on $N = pq = p_1^i q_1^j$.

At *withdrawal*, \mathcal{U} uses $N = p_1^i q_1^j$.

At *payment*, all the verifications would still succeed, since $(-1/N) = 1$, $(2/N) = -1$ and N contains only odd powers of two prime factors. Upon factoring N , however, the bank would only obtain $p_2 = p_1^i \neq p$ and $q_2 = q_1^j \neq q$.

Counteracting the attack

There are two ways of counteracting the attack. One entails some additional actions by the bank after deposit and the shop at payment, while the second is an addition to the account establishment protocol. Since in our scheme, presented in Section 5.3, we substitute the account establishment protocol of [100] by a more efficient withdrawal protocol, only the first countermeasure is applicable to it.

Countermeasure after deposit: First, it is easy to see that *if \mathcal{B} knows the nature of the attack*, it can still trace the user, albeit at a higher cost.

After deposit, the bank must first find i, j and p_1, q_1 from p_2, q_2 and then try all possible combinations of i, j until it finds p, q .

To reduce the required computation, some of the values for i, j can be excluded by having \mathcal{S} check, **at payment time**, whether N is divisible by some small primes, i.e., by putting a lower bound on the size of p_1, q_1 , and hence an upper bound on i, j . We propose for the shop to verify relative primality of N with the first $O(|N|)$ primes, as this balances the need for minimal computation at payment versus reduced computation required for the tracing protocol.

Efficiency considerations: Using the parameters of [100] ($|p| = |q| = 256$ bits), and if the shop, at payment, checks that N is not divisible by the primes less than 2^{13} , then $2^{13} < p_1 < p_2 = p_1^i < N < 2^{512}$ and so $i, j, i+j < 40$. This checking requires

only the same amount of computation as one exponentiation \pmod{N} : there are approximately $2^{13}/13 \approx 630$ primes less than 2^{13} ; 630 divisions are comparable to one exponentiation \pmod{N} used as a measure in the payment [100], where one exponentiation needs 512 multiplications. Since i, j are odd integers, at most 20 integer root operations \pmod{N} , which are as efficient as exponentiations \pmod{N} , are needed for finding i, j, p_1, q_1 . To find p , we need to try all combinations of (i', j') , for $i' < i, j' < j$. Since the worst case occurs when $i = 21, j = 19$, there are at most $21 \cdot 19 = 399$ such combinations. Each combination gives rise to one exponentiation $(g^{p'} \pmod{P})$ which is equivalent to $1/4$ exponentiation \pmod{P} , or $2 \pmod{N}$, due to the reduced size of the exponent p' . (Computing each new candidate value of p requires only 1 multiplication and therefore can be ignored.) Thus, in the worst case, at most 820 exponentiations \pmod{N} are needed to trace \mathcal{U} . In [100], the bank would also have to perform around 400 searches on its account database, one for each candidate value for (p, q) ; in our scheme presented in Section 5.1.2 below no search is necessary, since g^q is revealed at payment and can serve for verification of the correct p', q' . The required computation is well within the limits of the system, especially considering that it is performed on a very rare basis by the bank, which is usually the most computationally powerful of all involved parties.

It is very important to note that *this computation is only performed after it is known that a coin has been double-spent* and the malicious user needs to be identified. This computation will thus be performed very rarely and even then the associated costs can be charged to the malicious user.

Okamoto's fix during withdrawal: The second method, proposed by T. Okamoto [101], actually prevents the attack with reasonable efficiency costs: to this effect the user proves, at account establishment, not only that $N = pq$, for p, q committed by g^p, g^q , but also that the numbers committed by g^p, g^q are relatively prime. This guarantees, using the notation above, that if the user selects primes p, q and odd integers i, j and computes $p = p_1^{i'} q_1^{j'}$, $q = p_1^{i-i'} q_1^{j-j'}$, he has no choice but choosing $i' = j' = 0$; otherwise p, q will not be relatively prime. Hence, upon identification of over-spenders, $p_2 = p$ and $q_2 = q$. The full protocol for this proof appears in [22], supplied by T. Okamoto, and requires the addition of approximately 150 exponentiations \pmod{P} , i.e., 1,200 exponentiations \pmod{N} , at withdrawal.

5.3 The divisible e-cash scheme

As mentioned earlier, the starting point for our scheme is Okamoto’s [100] protocol, corrected as described in the previous section.

Okamoto’s scheme is quite efficient. His withdrawal protocol requires 100 Bytes of data exchanged and one (1) multiplication by the bank \mathcal{B} and the user \mathcal{U} . Moreover, the coin occupies about 200 Bytes of storage whereas all previous schemes required several hundreds of Kilobytes. During payment protocol, for example with a \$1,000 coin with minimum payment of 1 cent, i.e., divisibility precision $\mathcal{N} = 2^{17}$, [100] requires about 23 modular exponentiations for both user \mathcal{U} and shop \mathcal{S} . Previous schemes required several Kilobytes of exchanged data and over 200 modular exponentiations. Hence, [100] remains a significant contribution in electronic cash research.

The major advantage of our system, however, is that the construction of the electronic license, which is the bulk of the computation in [100]’s “user account establishment” protocol, requires the equivalent of *less than two* (2) modular exponentiations of [100], while [100] requires more than 4000. Furthermore, in contrast to our scheme, the number of exponentiations in [100] depends on the length of the RSA modulus, impairing the scalability of the system. In our scheme, account establishment is used to exchange authentication keys to be used later in withdrawal. Similar to [15, 46, 103, 104, 32, 50, 57, 23, 34] we put the functionality of licensing into the withdrawal protocol. This is important, as doing otherwise impairs users’ anonymity by allowing coins to be *linkable*: as highlighted in Section 3.1.1 at the description of our security model, and also mentioned in [105], the more the user uses the same license the more likely he can be traced by other means, i.e., by correlating various payments’ locality, date, type, frequency, or finding one payment where the user identified him/herself.

Our withdrawal protocol requires the user and the bank to perform the equivalent of less than two (2) exponentiations respectively. During payment, the user and shop perform around 25 exponentiations. The size of our coin remains around 300 bytes. Hence, our scheme can be implemented in current smart cards, while allowing for coins to be divisible.

5.3.1 The basic idea

To emulate the functionality of the account establishment protocol all that is needed is a method to provide a shop \mathcal{S} with a number N , such that

- (1) N is a composite of two numbers,
- (2) N is signed by the bank, and
- (3) \mathcal{S} , and subsequently the bank at deposit time, is guaranteed that if N is factored the owner of the coin will be identified.

Condition (1) is satisfied by the payment protocol of Okamoto [100], modified as shown in Section 5.2.3 above, which determines that N is a product of odd powers of two primes congruent to 3 and 7 mod 8 respectively and generates the tree as discussed in Sections 3.2.2, 5.2.3. What we suggest is a new approach for withdrawal, i.e., signing N , and coin authentication, i.e., proving the correctness of N to \mathcal{S} .

Our idea is to use a modified Brands [15] protocol for withdrawal and coin authentication. At account establishment \mathcal{U} associates his identity with $I' = g_3^u$. At withdrawal, he randomly generates $N = pq$ and identifies the particular withdrawal (hence himself) with $I_W = g_1^p$. During withdrawal, \mathcal{U} will end up with a message ($A = g_1^{pq} g_2^q g_3^{uq}, B = [N]$) and a signature $sign(A, B)$ on A, B . Hence (2) above will be guaranteed. The correctness of A and the unforgeability of the signature are guaranteed by the protocol in [15]. (See Section 5.3.3 for a detailed analysis).

To guarantee condition (3) we observe that during payment N is revealed and, if the coin is over-spent, N can be factored in the denomination revelation phase based on the results in [100], and as corrected in Section 5.2.3 above. At coin authentication \mathcal{U} proves that $A = g_1^N X$ for some $X = g_2^\beta g_3^\gamma$. Since $A = g_1^{pq} g_2^q g_3^{uq}$, due to the Brands withdrawal protocol, this indirectly guarantees that $N = pq$, i.e., the factorization of N reveals I' . Notice that, as pointed out to us by Okamoto [101], this only holds if we guarantee that p, q are small enough so there is no wrap-around in the modulus used, i.e., in $g_1^{pq} \pmod{P}$, $pq < Q$, where Q is the order of g_1 in the multiplicative group Z_P^* ; our range-bounded commitment (described in Section 5.2.1 above) is used for this purpose.

5.3.2 Protocols

There are actually two viable variants of our scheme. In one variant, $u = 0$, i.e., only three generators (g, g_1, g_2) are used, and the identifying information of the user \mathcal{U} is g_1^p , created by the user at each withdrawal. This identifying information must be stored by the bank. This variant, apart from requiring less communication/computation from both \mathcal{U} and the bank \mathcal{B} , also allows us to prove the untraceability of our scheme with respect to an assumption similar to one appearing in [100].

If $u \neq 0$, on the other hand, the bank, upon tracing a double-spender, only has to perform a search in its account database in order to locate $I' = g_3^u$. In contrast, in the first variant, it would have to store all identification values $I_W = g_1^p$ appearing in withdrawal protocols and then perform a search among them. Note that this does not increase the order of computation or storage needed by \mathcal{B} , since \mathcal{B} has to store deposit transcripts and perform searches at *each* deposit in a database containing all deposit protocols. However, the proof of this variant depends on a novel assumption, which we nevertheless believe is an interesting number theoretic problem to be analyzed.

We present the scheme when $u \neq 0$. It is then easy to derive the first variant, i.e., when $u = 0$. When we discuss the scheme's security in Section 5.3.3 we present the assumptions needed for both variants.

Initialization

Bank Initialization (setup) procedure:

The bank \mathcal{B} chooses the security parameters n , S and $H = |p| = |q| = |N|/2$, such that

- $2H$ is the size of the RSA modulus used; this parameter controls security for the user, since anonymity depends on the difficulty of factoring this modulus.
- n is the probability of failure of the protocol, i.e., the user is allowed to overspend a *single* coin with probability at most 2^{-n} ; thus n controls security for the bank.
- S is the desired length of the outputs of the random oracle-like hash functions used in the protocol. (In Section 2.2.6 we discuss necessary lower bounds on the output size of hash functions originating from the birthday paradox). S can be parameterized based solely on n ; for example if hash functions behave

like random oracles, $S = 2n$ is sufficient, as shown by Bellare and Rogaway in [8]. Here we present the scheme in its most general form however, allowing fine-tuning of S depending on the specific hash function to be used.

The bank then computes δ , such that $\delta > (2n + 2)/H$, and selects primes Q, P , with $P = 2Q + 1$, $|Q| = 2(1 + \delta)H + 6$. All arithmetic is performed in G_Q , except for the operations involving exponents, which are performed in Z_Q . \mathcal{B} chooses

- Four generators g, g_1, g_2, g_3 of G_Q ,
- $\mathcal{H}, \mathcal{H}_0, \mathcal{H}_1, \dots$, from a family of collision intractable hash functions, behaving like random oracles, and
- A private key $x \in_R Z_Q$ (a different key is used for every denomination).

\mathcal{B} publicizes the description of G_Q , i.e., P, Q , the generator-tuple (g, g_1, g_2, g_3) , the description of $\mathcal{H}, \mathcal{H}_0, \mathcal{H}_1, \dots$, and its public keys $h = g^x, h_i = g_i^x$, for $i = (1, 2, 3)$.

User Initialization (account establishment) procedure:

The user \mathcal{U} shows by physical or other means his identity to the bank \mathcal{B} and then associates himself with $I' = g_3^u$ to \mathcal{B} . The user also proves knowledge of u , by running a Schnorr proof of knowledge (as shown in Section 5.2.2 “How to repair the scheme” above). The bank verifies that $I' \neq \{1, g_3\}$. In the first variant \mathcal{U} sends to \mathcal{B} his public key using any public key cryptosystem so that an authenticated channel can be created at withdrawal (\mathcal{U} already has \mathcal{B} ’s keys from the bank initialization protocol).

Withdrawal

The signature used by the bank to sign the coin is the same as that of the Brands’ scheme [15]; it is summarized in equation (5.3) in Section 5.2.2 above.

At the beginning of the withdrawal protocol, the user creates an authenticated channel with the bank. This is needed in all e-cash (and physical cash!) protocols to guarantee that only the owner of an account withdraws money from it and that the user is communicating with the real bank. If $u \neq 0$, i.e., the second variant is used, this functionality is included in our withdrawal protocol: the user \mathcal{U} proves to the bank \mathcal{B} that he knows the representation of I' w.r.t. g_3 , and \mathcal{U} is convinced of \mathcal{B} ’s identity by checking the correctness of $sign(A, B)$.

The withdrawal protocol (over an authenticated channel between \mathcal{U} and \mathcal{B}), also appearing in Figure 5.2:

- \mathcal{U} : (This step can be pre-computed.)
 Select primes $p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}$, $|p| = |q| \leq H = (|Q| - 6)/[2(1 + \delta)]$ at random, and calculate $N = pq$.
 Send $I_W = g_1^p$ and $I' = g_3^u$ to \mathcal{B} .
- \mathcal{U}, \mathcal{B} : Perform a Schnorr proof of knowledge [116] that \mathcal{U} knows the representation of I' w.r.t. g_3 :
 \mathcal{U} chooses random $u_{I'} \in_R Z_Q$ and sends $U_{I'} = g_3^{u_{I'}}$; \mathcal{B} sends with a random challenge $c_{I'} \in_R Z_Q$; \mathcal{U} responds with $r_{I'} = c_{I'} u + u_{I'}$; and \mathcal{B} verifies that $g_3^{r_{I'}} = I'^{c_{I'}} U_{I'}$.
- \mathcal{U}, \mathcal{B} : Use *the range-bounded commitment* base g_1 with input $I_W = g_1^p$, in an interactive way and with just one iteration, to prove that $|p| \leq (1 + \delta)H$. This also proves, indirectly, that \mathcal{U} knows a representation of I_W w.r.t. g_1 .
- \mathcal{B} : Set $I = I_W I' (= g_1^p g_3^u)$, and check that $I_W \neq \{1, g_1\}$, $I g_2 \neq 1$.
 Pick $w \in_R Z_Q$, and send $a' = g^w$, $b' = (I g_2)^w$ to \mathcal{U} .
- \mathcal{U} : Compute $z' = (h_1)^p h_2 (h_3)^u$ [= $(I g_2)^x$ since $h_1 = g_1^x$, $h_2 = g_2^x$, $h_3 = g_3^x$]. This step can be pre-computed, or z' can be supplied by \mathcal{B} .
 Let $A = (I g_2)^q = g_1^N g_2^q g_3^{uq}$, $B = [N, Y = g_2^q]$, and $z = z'^q$.
 Pick $v_1, v_2 \in_R Z_Q$ and compute $a = a'^{v_1} g^{v_2}$ and $b = b'^{v_1} A^{v_2}$.
 Compute the challenge $c = \mathcal{H}(A, B, z, a, b)$, and send the blinded challenge $c' \equiv c/v_1 \pmod{Q}$ to \mathcal{B} .
- \mathcal{B} : Send the response $r' \equiv c'x + w \pmod{Q}$ to \mathcal{U} , and debit \mathcal{U} 's account.
- \mathcal{U} : Accept iff $g^{r'} = h^{c'} a'$ and $(I g_2)^{r'} = z'^{c'} b'$. Compute $r \equiv r'v_1 + v_2 \pmod{Q}$, to get the signature (z, a, b, r) on (A, B) .

Payment and deposit

Coin authentication:

The user \mathcal{U}		The bank \mathcal{B}
$p \equiv 3 \pmod{8}, q \equiv 7 \pmod{8}$ $B = [N = pq, Y = g_2^q]$ $I_W = g_1^p$	$\xrightarrow{I', I_W}$	$I = I_W I'$ $I_W \stackrel{?}{\neq} \{1, g_1\}, I g_2 \stackrel{?}{\neq} 1$
Proves knowledge of representation of I' w.r.t. g_3	————	Verifies knowledge
Proves I_W is a bounded commitment	————	Verifies proof
$A = (I g_2)^q$ $z = z'^q = h_1^{pq} h_2^q h_3^{uq} [= (I g_2)^{xq}]$ $v_1, v_2 \in_R Z_Q$ $a = a'^{v_1} g^{v_2}$ $b = b'^{q v_1} A^{v_2}$ $c = \mathcal{H}(A, B, z, a, b)$ $c' = c/v_1$	$\xleftarrow{a', b'}$	$w \in_R Z_Q$ $a' = g^w, b' = (I g_2)^w$
$CHECK\Gamma$ $g^{r'} = h^{c'} a'$ $(I g_2)^{r'} = z'^{c'} b'$ Compute: $r = r' v_1 + v_2$	$\xrightarrow{c'}$	$r' = c' x + w$
	$\xleftarrow{r'}$	

Figure 5.2: The withdrawal protocol for divisible e-cash, over an authenticated channel between \mathcal{U} and \mathcal{B} .

- \mathcal{U} : Pick $x_3 \in_R Z_Q$. Compute

$$Y = g_2^q, Y_3 = g_3^{uq}, y_3 = g_3^{x_3}, d = \mathcal{H}_1(A, B, Y_3, y_3, \text{date/time}, ID_S) .$$

This is a non-interactive approach but one could add a random challenge from \mathcal{S} into the hash (\mathcal{H}_1) if desired. In either case, $[A, B, Y_3, y_3, \text{date/time}, ID_S]$ must be included in the hash (as in the self-challenging Schnorr proof of knowledge described in Section 2.2.6), so that even if \mathcal{S} and \mathcal{U} collaborate the subsequent proofs of knowledge are still valid. The non-interactive case allows for the payment protocol to be conducted in one move, from \mathcal{U} to \mathcal{B} .

This step, except for the challenge, if an interactive approach is used, can be pre-processed.

- **\mathcal{U} sends the coin to \mathcal{S} :** Send $A, B = [N, Y], \text{sign}(A, B), Y_3, y_3$, and respond to challenge d with $r_3 = duq + x_3$.
- **\mathcal{S} verifies that the coin is legitimate:**

1. **Verify the signature $\text{sign}(A, B)$,** and that

$$Y \neq g_2, Y \neq g_2^N, A \neq 1, Y_3 \neq g_3, (-1/N) = 1, (2/N) = -1 .$$

2. **Verify that \mathcal{U} knows a representation of Y_3 with respect to g_3 ,** using the Schnorr proof of knowledge [116]:

$$g_3^{r_3} \stackrel{?}{=} y_3 Y_3^d .$$

3. **Prove that q is chosen correctly:** Use *the range-bounded commitment* base g_2 with $Y = g_2^q$, to prove that $|q| \leq (1 + \delta)H$. As discussed in Section 5.2.1 for the interactive case the challenge e is computed based on a hash function—as d above—so that even a collaboration of \mathcal{U} and \mathcal{S} cannot forge the proof. Hence, as discussed in Section 5.2.1 above, $\lceil S/n \rceil$ iterations of the protocol are performed so that the output of the hash function is of length at least S .
4. **Verify that A is correctly constructed:**

$$g_1^N Y Y_3 \stackrel{?}{=} A .$$

5. **Limit the way \mathcal{U} can misbehave:** Check whether N is divided by the first $|N|$ primes that are congruent to 3 mod 4. This helps in identification of over-spenders, as described in Section 5.2.3 in “countermeasure after deposit” above.

Denomination revelation: We use [100]’s protocol, as described in Section 3.2.2, with the only modification being the substitution of the coin (C, N) in the hash functions of Okamoto with our coin, (A, B) . This protocol guarantees that if one of the node rules is violated then \mathcal{U} has released enough information to allow \mathcal{B} to factor N .

Deposit: \mathcal{S} sends the payment transcript to \mathcal{B} . If one of the node rules are violated the bank traces the user with the protocol described in Section 5.2.3 in “countermeasure after deposit” above.

5.3.3 Security

We now present the security model and prove the security of our scheme.

As with Okamoto [100], our security model has been based on Franklin and Yung’s work [57]; however our model is extended to work for divisible, unlinkable coins. We model the security of the scheme by requiring that it satisfies four requirements which are slightly stronger than the respective properties in [100], which are included in brackets for comparison:

- (1) *unreusability* [No overspending],
- (2) *untraceability* [No tracing],
- (3) *unexpandability* [No forging and No swindling], and
- (4) *unforgeability*.

Formal definitions of these requirements are given in theorems 5.3.2, 5.3.3 and 5.3.4 below. Note that “No swindling” is guaranteed by combining unreusability and unexpandability: even a collaboration of users/shops cannot over-deposit the withdrawn/paid coins. Unforgeability, which guarantees that no collaboration of users can withdraw/construct a single coin not embedding the identity of one of these users, is not covered in the model of Okamoto [100].

As in the model for basic e-cash, the use of a *non-uniform*, probabilistic polynomial time Turing machine (*p.p.t. TM*) in our model simulates user collaboration as views to the TM. Thus, in establishing the security we prove that even a collaboration of users and/or shops cannot break the scheme.

Our proofs of security are based on the following **assumptions**:

- (1) (when $u = 0$) (**Factoring and Diffie-Hellman II**)

Let $P = 2Q + 1$, Q, p_0, q_0, p_1, q_1 be primes, $N_0 = p_0q_0$, $N_1 = p_1q_1$, and $H = |p_0| = |q_0| = |p_1| = |q_1| \leq (|Q| - 6)/(2(1 + \delta))$ for some sufficiently large $\delta > 0$. Let the order of g in the multiplicative group Z_P^* be Q . Then,

no p.p.t. TM \mathcal{M} can, given $P, Q, \delta, g, [Y_0(\equiv g^{q_0} \pmod{P}), N_0(= p_0q_0)], [Y_1(\equiv g^{q_1} \pmod{P}), N_1(= p_1q_1)]$ and $[I_r(\equiv g^{p_r} \pmod{P}), I_{1-r}(\equiv g^{p_{1-r}} \pmod{P})]$ ($r \in_R \{0, 1\}$), compute r with probability better than $1/2 + 1/H^c$, for all constants c and sufficiently large H ; i.e., \mathcal{M} cannot compute r non-negligibly better in H than random guessing.

- (2) (**Withdrawal protocol**) If random oracle-like hash functions exist then our withdrawal protocol is a *restrictive blind signature protocol*: the message $m = Ig_2 = g_1^{u_1} g_3^{u_3} g_2$ is signed by the string $A = g_1^\alpha g_3^\gamma g_2^\beta$, in such a way that $\alpha/\beta = u_1, \gamma/\beta = u_3$.
- (3) (**Hash functions**) Hash functions $(\mathcal{H}, \mathcal{H}_0, \mathcal{H}_1, \dots)$ behave like truly random functions, i.e., random oracles.

Remarks:

- (1) An assumption similar to *Factoring and Diffie-Hellman II* appears in [100]. It implies that the Discrete Logarithm Assumption (DLA) holds and that factoring is difficult, since if either can be solved the assumption doesn't hold. If $u \neq 0$, i.e., the second variant of our scheme is used, then this assumption needs to be modified as follows:

(Multiple Factoring and Diffie-Hellman)

Let $P, Q, H, \delta, p_0, q_0, p_1, q_1, N_0, N_1, Y_0, Y_1, I_r, I_{1-r}$ be defined as in *Factoring and Diffie-Hellman II*, and $u_0, u_1 \in_R Z_Q$. Then,

no p.p.t. TM can, given $P, Q, \delta, g, [Y_0, N_0, Y_0'(\equiv g^{u_0 q_0} \pmod{P})], [Y_1, N_1, Y_1'(\equiv g^{u_1 q_1} \pmod{P})]$ and $[I_r, I_r'(\equiv g^{u_r} \pmod{P})], [I_{1-r}, I_{1-r}'(\equiv g^{u_{1-r}} \pmod{P})]$ ($r \in_R \{0, 1\}$), compute r with probability better than $1/2 + 1/H^c$, for all constants c and sufficiently large H .

We believe that both Factoring and Diffie-Hellman assumptions represent interesting number theoretic problems to be studied.

- (2) An assumption equivalent to *Withdrawal protocol* appears in Brands' e-cash scheme [15] (assumption 1). Our assumption can be reduced to assumption 1 in [15] as follows: assume it does not hold. Then let $g'_2 = g_1^p g_2$, $g''_2 = g_3^u g_2$. Then, Brands' assumption, with either $(g_1, g_2 = g'_2)$, or $(g_1, g_2 = g''_2)$, does not hold. The other direction is trivial.

Although this assumption is stronger than the DLA there are convincing arguments [15] that suggest that breaking it requires breaking either the Schnorr signature scheme or the DLA.

- (3) The *Hash functions* assumption is difficult to guarantee. As suggested by Okamoto [100] it requires tamper-free devices or an exponential amount of

memory. Bellare and Rogaway [8] suggest an implementation using MD5 in a special manner, as is described in Section 2.2.6. We use it because it clarifies our scheme, and, for all practical purposes, commonly available one-way hash functions can be used.

For our proofs we use the following lemma:

Lemma 5.3.1 (Schnorr signatures) *Schnorr signatures [116] are existentially unforgeable, even when the prover in the Schnorr identification protocol is queried polynomially many times.*

As also mentioned in Section 2.2.7, this lemma has been recently proven by Pointcheval and Stern in [107] under the DLA, which is included in our first two assumptions above, and under the random oracle model, i.e., based on our *Hash functions* assumption.

Theorem 5.3.2 (Unreusability) *Let n be the security parameter. If the successfully deposited nodes of a coin violate the root route rule or the same node rule, then the identity of the coin's owner can be efficiently, i.e., by a p.p.t. TM, computed and subsequently proven from the transcripts of the withdrawal and the deposit protocols with overwhelming probability in n .*

Proof. The *Withdrawal protocol* assumption and lemma 5.3.1, which themselves require assumption *Hash functions* above, together with the verification done at the coin authentication stage, guarantee that the element A (of the coin (A, B)) is of the form $g_1^{pq} g_2^q g_3^{uq}$ for some p, q, u (p, q not necessarily prime) and where $I_W \equiv g_1^p$, $I' \equiv g_3^u \pmod{P}$, and $|p| \leq (1 + \delta)H < |Q|/2$, with probability overwhelming in n . We will also show that p is relatively non-prime to the N included in B , and how this leads to identification of \mathcal{U} , i.e., to I_W and I' .

Steps 2 and 3 in payment guarantee that \mathcal{U} knows a representation of Y, Y_3 w.r.t. g_2, g_3 , respectively: $Y = g_2^t, Y_3 = g_3^{t_3}$, for $t, t_3 \in Z_Q$, where, with probability overwhelming in n , $|t| \leq (1 + \delta)H < |Q|/2$. Together with step 4, this proves that \mathcal{U} knows $(N, t, t_3), (u, p, q)$ such that $A \equiv g_1^N g_2^t g_3^{t_3} \equiv g_1^{pq} g_2^q g_3^{uq} \pmod{P}$.

If $N \not\equiv pq \pmod{Q}$, $t \not\equiv q \pmod{Q}$ or $t_3 \not\equiv uq \pmod{Q}$, then \mathcal{U} would know more than one representation of A with respect to (g_1, g_2, g_3) , which contradicts the representation problem in groups of prime order and hence our *Withdrawal protocol*

assumption. Therefore, $t \equiv q \pmod{Q}$, $t_3 \equiv uq \pmod{Q}$, $N \equiv pq \equiv pt \pmod{Q}$, and $Y \equiv g_2^q, Y_3 \equiv g_3^{uq} \pmod{P}$.

Now, Okamoto in [100] proves, under the assumption that factoring is difficult, which in turn is included in assumption *Factoring and Diffie-Hellman II* above, and assumption *Hash functions* above, that with probability $1 - 1/2^n$, N is guaranteed to be of the form $p^i q^j$ with $p' \equiv 3 \pmod{8}$, $q' \equiv 7 \pmod{8}$ primes, i, j odd integers, and if the root route rule or the same node rule are violated, then \mathcal{B} obtains p^i, q^j . Also, N is blindly signed by \mathcal{B} at withdrawal since it is included in $\mathcal{H}(A, B, z, a, b)$, hence it is unique, and therefore p', q', i, j are also unique.

Hence, $p^i q^j = N \equiv pt \pmod{Q}$. But $N \geq 0$ and $|N| = |2H| \leq |Q| - 6$, thus $0 \leq N < Q/2^5 < Q$. Also, $|pt| < |Q| - 5$ (since $|p|, |t| \leq (1+\delta)H, |Q| = 2(1+\delta)H + 6$), hence $-Q/2^5 \leq pt \leq Q/2^5$. If $pt < 0$ then $pt \equiv Q + pt \pmod{Q}$ with $Q + pt \geq Q - Q/2^5 > Q/2^5 > N$, while $Q + pt \equiv pt \equiv N \pmod{Q} \implies Q + pt = N$, a contradiction. Hence, $0 \leq pt, N \leq Q/2^5$ and $N \equiv pt \pmod{Q} \implies N = pt$; thus $N = pt = p^{i'} q^{j'}$. Since $t|N$, t is either $N, 1$ or $p^{i'} q^{j'}$, for some $i' \leq i, j' \leq j$. But at coin authentication \mathcal{S} has verified that $g_2^t \equiv Y \notin \{g_2, g_2^N\} \pmod{P}$, i.e., $t \notin \{1, N\}$. Thus, $t = p^{i'} q^{j'}$ and, consequently, $p = N/t = p^{i-i'} q^{j-j'}$.

Step 5 of coin authentication, together with our amendment to [100]'s user identification protocol, described in Section 5.2.3 above, guarantee that \mathcal{B} can, with an acceptable overhead, find i, j and $i' \leq i, j' \leq j$ such that $p^{i'} q^{j'} = t = q$, hence

- $g_2^t \equiv g_2^{p^{i'} q^{j'}} \pmod{P}$,
- $p = p^{i-i'} q^{j-j'}$,
- $I_W \equiv g_1^p \pmod{P}$, and
- $I' \equiv A^{(q^{-1})}/(g_1^p g_2) \equiv A^{(t^{-1})}/(g_1^p g_2) \pmod{P}$.

The fact that \mathcal{B} now knows a representation of I_W w.r.t. g_1 constitutes proof of double spending, since finding a representation is equivalent to breaking the DLA (see Section 2.1.2), which is included in our *Withdrawal protocol* assumption above. Therefore, if \mathcal{U} over-spends he is identified with probability overwhelming in n . \square

Theorem 5.3.3 *Let n be the security parameter. We treat the collection of the portions of a coin as being a single, indivisible, coin.*

(Unforgeability) *No p.p.t. TM can, from the views of users of arbitrarily many withdrawal and payment protocols, compute a single coin that does not embed the*

identity of at least one of these users and that will lead to two successful purchase or deposit protocols, except with negligible probability in n .

(Unexpandability) *The probability that, from the views of users and shops of N withdrawal and of N payment protocols, a p.p.t. TM can compute an additional coin that will lead to a successful purchase or deposit is negligible in n .*

Proof. Brands proves in [15], under the DLA, assumption *Hash functions* and lemma 5.3.1 above, that it is infeasible to existentially forge a coin (*unexpandability*) with probability better than $1/2^{|Q|} < 1/2^n$, i.e., random guessing, even when performing the withdrawal protocol polynomially many times and with respect to different account numbers. It is also proven, under assumption *Withdrawal protocol*, that every coin embeds the identity of its owner (*unforgeability*), with probability $1 - 1/2^{|Q|}$. In [14] Brands shows that these proofs are valid even if \mathcal{U} 's identity is represented in more than one generator, as in our case, (g_1, g_3) . Our scheme restricts \mathcal{U} 's power, in comparison to [15]: \mathcal{U} 's secret numbers (p, q) have to be factors of N , where $|N| < |Q|$. Our scheme also proves at payment time that \mathcal{U} knows the representation of A w.r.t. (g_1, g_2, g_3) as in [15, 14]. Hence, the proofs of [15, 14] can be carried over and guarantee the unforgeability and unexpandability of our scheme. \square

The last theorem proves anonymity for the user based on our *Factoring and Diffie-Hellman II*, if $u = 0$, and *Multiple Factoring and Diffie-Hellman*, if $u \neq 0$, assumptions respectively. In order to formalize our treatment of anonymity we create an analogy to encryption. In particular, we view the union of the bank's collection of account establishment and withdrawal protocol together with the shop's payment transcript and the bank's deposit transcript for a coin as *the encryption of the link between this coin and its withdrawal*. This encryption is performed by the user upon withdrawal. If this encryption is *semantically secure* (see Section 2.2.2 for a definition) it follows that a collaboration of the bank and shop cannot trace the user.

To see this, assume otherwise, i.e., that even partial tracing is possible. Then given a bank's view of the deposit and payment protocols for a coin the bank could obtain some information about the identity of the coin's owner. Thus, with probability non-negligibly better than random guessing, it could exclude some withdrawal transcripts from the candidate transcripts for this coin, by simply disregarding the withdrawal transcripts belonging to different owners. Note that there are scenarios in which such

tracing does not help the bank distinguish between the withdrawal transcripts; e.g., when only one user has withdrawn coins, or when the bank already has some a-priori information that limits the possible withdrawals for the candidate payments (coins) to only one user. However, in all such trivial cases there is only one candidate user; in all other scenarios untraceability is handled by our definition below.

We now proceed to formalize anonymity based on this encryption being secure in the sense of indistinguishability, since showing this is easier, while as we have seen in Section 2.2.2 an encryption secure in the sense of indistinguishability is also semantically secure. For simplicity *we model the bank's and shops' collaboration by treating the shops' view of payments as being bank's views, and we include the bank's view of an account establishment in the withdrawal.* Informally, the theorem below states that there do not exist two “plaintexts” (links between C_i, C_j and W_0, W_1) whose encryptions (coins C_i, C_j) can be distinguished non-negligibly better than random guessing.

Theorem 5.3.4 (Untraceability) *Let n be the security parameter. We treat the collection of the portions of a coin as being a single, indivisible, coin.*

Let a p.p.t. \mathcal{M} have access to all \mathcal{B} 's views of withdrawal, payment and deposit protocols. Then for any two coins C_i, C_j and withdrawals W_0, W_1 , such that C_i, C_j are the coins originating from W_0, W_1 , \mathcal{M} cannot distinguish non-negligibly better in n than random guessing whether C_j came from W_0 or W_1 .

This theorem guarantees more than security in the sense of indistinguishability. For the latter it is sufficient to show that it is probabilistically infeasible to find two “plaintexts” (i.e., the two links between W_0, W_1 and C_i, C_j) such that their “encryptions” (i.e., C_i, C_j) are distinguishable non-negligibly better than random guessing in n . The theorem instead shows that *there do not exist* two plaintexts $[(W_0, W_1), (C_i, C_j)]$ such that their encryptions C_i, C_j are non-negligibly distinguishable.

Unlinkability: Recall that in addition to anonymity we want to guarantee unlinkability among the coins of the same user. Here we show that the above theorem also guarantees that linkability is not possible in general:

Let an honest user be a user that follows the protocols. Let U be the number of *honest* users that have engaged in at least one withdrawal protocol, and $w_i (p_i)$

be the number of withdrawal (payment) protocols an honest user i has engaged into. Assume that there exists a p.p.t TM \mathcal{M} which, given some a-priori information, can obtain linking information for coins belonging to the same user with probability non-negligibly better than what could be guessed from the a-priori information alone, for all but a negligible fraction of the possible combinations of U, w_i, p_i .

Then there exists a $U > 1$ and two users $\mathcal{U}_1, \mathcal{U}_2$ for which $w_1 \neq w_2, p_1 = w_1, p_2 = w_2$ whose coins can be linked by \mathcal{M} . Otherwise, for all $U > 1$, the TM \mathcal{M} would not be able to link coins when $w_m \neq w_j, p_m = w_m, p_j = w_j$, for all $m, j \in \{1, \dots, U\}, m \neq j$, i.e., for a non-negligible fraction of the possible combinations of U, w_i, p_i ; a contradiction to the power of \mathcal{M} .

Hence \mathcal{M} can find two coins C_m, C_j such that C_m is linked to $p_1 - 1$ other coins with probability non-negligibly better than random guessing, and C_j to $p_2 - 1$ other coins, again with probability non-negligibly better than random guessing. Assume, WLOG, that $w_1 = p_1 < w_2 = p_2$. Then for any withdrawal views W_1 of user \mathcal{U}_1 and W_2 of user \mathcal{U}_2 , \mathcal{M} can decide, non-negligibly better than random guessing, that C_m originated from W_1 and C_j from W_2 , simply by observing that \mathcal{U}_1 has not engaged in enough withdrawal protocols to produce p_2 coins; thus C_j could not be coming from W_1 .

Therefore if the bank could link users' coins in a general setting, i.e., regardless of the distribution of withdrawn and deposited coins, the system would also violate our definition of untraceability.

We are now ready to proceed with the proof.

Proof: We prove the theorem for $u = 1$ (second variant) under the *multiple Factoring and Diffie-Hellman* assumption. It is then straightforward to obtain the proof for the first variant and verify that the *Factoring and Diffie-Hellman II* assumption is sufficient in this case.

We concentrate on the information revealed in the coin authentication protocol. This is because the information revealed during our denomination revelation protocol, i.e., N and some square roots modulo N of some randomly chosen numbers, is only related to N . Thus if the bank could trace using the denomination revelation, it would do so by obtaining some information about N that did not already possess,

i.e., some information about N 's prime factors p and q . But the information revealed about N is the same as the information revealed by the denomination revelation of Okamoto's [100] scheme. Okamoto, as well as Okamoto and Ohta in [103] which uses a similar protocol, has shown that this information does not suffice to reveal the factorization of N —unless the user over-spends.

Assume that \mathcal{B} has access to an oracle that allows it to distinguish which withdrawal protocol the coin C_j came from. We show that \mathcal{B} can use this oracle to break the *Multiple Factoring and Diffie-Hellman* assumption.

Hence we show how, given an instance of the Multiple Factoring and Diffie-Hellman problem, \mathcal{B} can turn it into a valid instance of the tracing problem and solve it using the tracing oracle.

Given

$$[Y'_0(\equiv g_1^{u_0q_0} \bmod P), Y_0(\equiv g_1^{q_0} \bmod P), N_0(= p_0q_0)], [Y'_1(\equiv g_1^{u_1q_1} \bmod P), Y_1(\equiv g_1^{q_1} \bmod P), N_1(= p_1q_1)]$$

and

$$[I_r(\equiv g_1^{p_r} \bmod P), I'_r(\equiv g_1^{u_r} \bmod P)], [I_{1-r}(\equiv g_1^{p_{1-r}} \bmod P), I'_{1-r}(\equiv g_1^{u_{1-r}} \bmod P)],$$

$$(r \in_R \{0, 1\}),$$

the bank calculates the views:

- W_i (withdrawal view corresponding to $I_i I'_i$, $i \in \{r, 1-r\}$).

Compute $(I_W)_i = I_i$ and $I'_i = I_i^{x'_3} (\equiv g_3^{u_0q_0} \bmod P)$, where $g_3 = g_1^{x'_3}$.

Pick $w_i, c'_i \in_R Z_Q$.

Compute $a'_i = g^{w_i}$, $b'_i = ((I_W)_i I'_i g_2)^{w_i}$, $r'_i = c'_i x + w_i$.

The Schnorr proof of knowledge of the representation of I'_i w.r.t. g_3 is simulated by setting $U_{I'} \equiv g_3^{u_{I'} I'^{-c_{I'}}}$, for random $u_{I'} \in_R Z_Q$ and challenge $c_{I'} \in_R Z_Q$.

A valid view of the range-bounded commitment is simulated by setting $U_l = g_1^{u_l} (I_W)_i^{-e_l}$, for random $u_l \in_R \{0, \dots, 2^{(1+\delta)H} - 1\}$ and challenge $e_l \in_R \{0, \dots, 2^n - 1\}$.

- P_j (payment view corresponding to Y_j, Y'_j, N_j , $j \in \{0, 1\}$):

Compute $Y_j = (Y_j)^{x'_2} (\equiv g_2^{q_j} \bmod P)$, where $g_2 = g_1^{x'_2}$, and $Y'_j = (Y_j)^{x'_3}$.

A valid view of the range-bounded commitment is also simulated in the same way as above. Since in this case a hash function is used the simulator also

modifies the output of the hash function \mathcal{H}_0 to output the pre-selected e_l , for each iteration l ; hence, after selecting e_l and u_l for each of the m iterations, the simulator outputs $e_l = \mathcal{H}_0(X, U_1, \dots, U_m)$, by modifying the output of \mathcal{H}_0 on this particular input. But \mathcal{H}_0 is a random oracle, hence the simulation is computationally indistinguishable from the real protocol since altering polynomially many points of a random oracle still results in a random oracle, and any random oracle is as good as, i.e., polynomially indistinguishable from, any other random oracle. A similar simulation technique is used in, e.g., [55] and is described in Section 6.4.1.

The knowledge of representation of Y_j' w.r.t. g_3 is simulated as in the withdrawal protocol. Changing the output of the random oracle \mathcal{H}_1 is necessary, but the simulation remains (computationally) indistinguishable from the bank's view of the protocol as discussed in the previous paragraph.

Compute $A_j = g_1^{N_j} Y_j Y_j', z_j = A_j^x$.

Pick $f_j \in_R Z_Q$, and compute $a_j = g^{f_j}, b_j = A_j^{f_j}$.

Compute $c_j = \mathcal{H}(A_j, N_j, z_j, a_j, b_j)$ and $r_j = c_j x + f_j$.

The construction above guarantees that \mathcal{B} 's views of the withdrawal protocols and payment protocols are valid, in isolation. To complete the proof we also need to show that the pairs of withdrawal/payment views are valid, i.e., if C_j is the coin originating from W_i then there exists a set of choices that user \mathcal{U}_i could have made in order to obtain coin C_j (of payment P_j) after engaging in W_i . In this case the oracle does provide a valid response. Then \mathcal{B} can use the oracle to see which of W_i corresponds to, e.g., C_0 , and thus it can break the *Multiple Factoring and Diffie-Hellman* assumption.

To see the validity of the views, consider any one of the two valid pairs (W_i, P_j) ; for this, \mathcal{U} could choose v_1, v_2 such that $v_1 = c_j/c_i'$ and $f_j = w_i v_1 + v_2$. Then it is easy to verify that \mathcal{U} would end up with the same coin \mathcal{B} simulated in P_j , after engaging in W_i with identity $I = I_i'(I_W)_i$ and primes p_i, q_i . For completeness, we verify this claim. After engaging in W_i with the above parameters, user \mathcal{U} would end up with a set values for A and the signature (z, a, b, r) . At payment P_j the bank would see a set of the same values. All we need to guarantee is that these values would be equal, i.e., that the above choices of \mathcal{U}_i for v_1, v_2 , result in the coin presented in P_j . Here, i marks the values obtained at withdrawal and j the values expected at payment; we verify their equality:

- $A_i = (Ig_2)^{q_i} = g_1^{p_i q_i} g_2^{q_i} g_3^{u_i q_i}$. Since W_i corresponds to P_j , we have that $u_i = u_j, p_i = p_j, q_i = q_j$, thus $A_i = g_1^{p_j q_j} g_2^{q_j} g_3^{u_j q_j} = g_1^{N_j} Y_j Y_j' = A_j$.
- $r_i = r_i' v_1 + v_2 = c_i' x v_1 + w_i v_1 + v_2$, and from the choice of v_1, v_2 : $r_i = c_i' x v_1 + f_j = c_j x + f_j = r_j$.
- $z_i = A_i^{r_i} = A_j^{r_j} = z_j$.
- $a_i = a_i'^{v_1} g^{v_2} = g^{w_i v_1 + v_2} = g^{f_j} = a_j$.
- $b_i = b_i'^{q_i v_1} A_i^{v_2} = (Ig_2)^{q_i (w_i v_1)} A_i^{v_2} = A_i^{w_i v_1} A_i^{v_2} = A_i^{w_i v_1 + v_2} = A_i^{f_j} = A_j^{f_j} = b_j$.

□

5.3.4 Efficiency

As also mentioned in Sections 5.1.2 and 5.3, this scheme is a major improvement over the fastest divisible e-cash scheme to date [100], with the inefficient withdrawal protocol of the latter being improved by more than three orders of magnitude. The resulting scheme, with both withdrawal and payment protocols being comparable in efficiency to the withdrawal protocol of one of the most efficient *non-divisible* e-cash systems to date [15], represents the first realistic solution for providing divisible anonymous off-line e-cash with unbounded divisibility.

With the purpose of giving a general idea on the efficiency of the system, we examine the efficiency under some reasonable security parameters. We adopt the parameters of [100], which in turn provide security comparable to that of [15], hence this section may also serve as a comparison for the three schemes.

The sample security parameters used for this evaluation are $n = 40$, $H = |p| = |q| = 256$ and $S = 2n = 80$. Thus, $|N| = 512$, $|Q| = 688$ (i.e., $\delta = 0.33$) and $|P| = 689$. Although we believe that 512 bits may not be sufficient for an RSA modulus we use this value for comparison with [100]. However our coin remains small if, e.g., $|N| = 1024$ (+300 Bytes).

We assume that the binary tree has 18 levels, i.e., the divisibility precision is 2^{17} , hence sufficient to divide a \$1,000 coin down to 1 cent. Note however that there is no bound on the divisibility precision and this sample value is only used to estimate the average number of nodes that are spent per payment. We assume the existence of fast, random oracle-like hash functions. No pre-processing is assumed unless explicitly

stated; in practice several of the steps can be pre-computed. We calculate for $u = 0$.

We remark here that the user is allowed to balance the computations performed at payment with the memory required to store values already computed at withdrawal, or otherwise pre-computed. Here we assume that *the system is optimized for minimal storage requirements*, but estimates for other variations are straightforward.

Storage requirements: The information \mathcal{U} needs to store for one coin $(p, q, (a, b, r))$ is 323* Bytes, or up to 495 Bytes if \mathcal{U} stores, rather than recalculating before each payment, A and/or z), and plus 86 Bytes if $u \neq 0$. In comparison, the coins in [100] are 264 Bytes and in [15] 384 Bytes, when the same parameters are used.

Computation and communication: *Our exponentiations are more than 3 times less costly than [100]*, since we use a modulus of 689 bits instead of 1030 and exponentiations require $O(\log Q \log^2 P)$ bit operations [85]; here we assume that a modular multiplication is performed in $O(\log^2 P)$ steps, i.e., in time quadratic to the modulus size [85]. Algorithms performing multiplication in time $O(\log^{1.585} P)$ do exist [84] but are applicable for numbers much larger than the ones we use here. Due to the reduced exponent size some of our exponentiations, including the ones in our range-bounded commitment, are more than 6 times less expensive than [100]. A multi-exponentiation of the form $g_1^{x_1} g_2^{x_2}$ costs the equivalent of 1.2 exponentiations [100, 84, 101].

At withdrawal \mathcal{U} performs the equivalent of 1.4 exponentiations of [100] and \mathcal{B} 1.36. When $u \neq 0$, \mathcal{U} sends 645 Bytes and \mathcal{B} 349 Bytes. \mathcal{U} also needs to calculate one Williams integer, but he can pre-compute one any time before withdrawal. In contrast, in [100] \mathcal{U} needs to perform more than 4,000 *multi*-exponentiations for the same functionality.

In the coin authentication phase, \mathcal{U} transmits 774 Bytes. Exponentiations at payment time in [100] are mod N , and hence are 8 times less expensive than those at withdrawal of [100]. Here we compare to exponentiations mod N . \mathcal{U} needs to perform the equivalent of around 5 exponentiations (if he re-computes A, Y and z) and \mathcal{R} around 7.

In the denomination revelation phase 9 nodes (on average) are paid. For each node, two 512 bit values are sent to \mathcal{R} , for a total of 1,152 Bytes. In addition about 320 Bytes (on average) are sent for verifying that N is a Williams integer. Both \mathcal{U} and \mathcal{R} compute approximately 20 roots \pmod{N} (we include the Williams integer verification) where each root computation is similar to an exponentiation \pmod{N} .

Scheme	Exact payments?	Space (Bytes)	Time (exponentiations)	
			Withdrawal	Payment
Brands [15]	NO	400	10	1
Okamoto, Eng [46]	YES	$400 + 10 S$	10	$\frac{N}{S}$
Okamoto [100]	YES	400	150,000	64
The divisible scheme	YES	926	63	64
Multiple Coins	YES	$316 + 10k \ln \frac{N}{k}$	$6 + 1.2 \cdot k \ln \frac{N}{k}$	$1.2 \cdot \ln \frac{N}{k}$

Table 5.1: Efficiency of divisible off-line electronic cash schemes under some sample security parameters; numbers are rounded for easy comparison. The fastest non-divisible scheme is included for comparison. Exponentiations are shown for the user and assume a modulus of 512 bits with exponents of 160 bits. S is the amount of storage available for the user.

5.4 Optimality based on divisibility precision

As mentioned in earlier sections, there is a threshold, depending on the divisibility precision and number of guaranteed payments, k , below which keeping a multitude of coins can be preferable to utilizing our divisible electronic cash scheme. Table 5.1 gives an evaluation of the efficiency of both schemes under some sample parameters; previous results are included for comparison. We note here that the number of exponentiations as shown in this Table differs from that in the previous section; this is because exponentiations are assumed to be over a modulus of 512 bits with 160 bit exponents, as in [14], instead of over a 1030 bit modulus or a 512 bit modulus, as in the withdrawal and payment protocols respectively of [100].

We examine both schemes under the security parameters described in the previous section, i.e., $n = 40, H = 256, S = 80, |Q| = 688, |P| = 689$ for the divisible scheme and $|Q| = 160, |P| = 512, k_1 = 160, k_2 = 80$ for the Brands-based scheme using multiple coins; these provide comparable security for the bank. Note that the divisible coin scheme has an additional security parameter, n , controlling the probability that a user can forge a single coin; this parameter is implicit in Brands' scheme [15] and thus in our solution using non-divisible coins and it is equal to $|Q|$, i.e., the size of the multiplicative group induced by the generators g, g_1, g_2 , since the user can forge a coin by guessing a valid signature on a randomly created coin. The impact of the size of n on the efficiency of the divisible scheme is higher; fortunately, in practice n need

not be as large as $|Q|$. We instead assume that a value of $n = 40$ is sufficient. Lastly, user anonymity is guaranteed unconditionally in Brands’ [15] scheme, and thus in our multiple coin scheme, whereas in our divisible scheme it depends on the “Factoring and Diffie-Hellman” assumptions.

For the two schemes to be comparable, we calculate for $u \neq 0$ in the divisible scheme, so that the same model for identifying double-spenders, i.e., linking to the account database rather than the withdrawal database, is implied.

As we have seen in Section 5.1.1 above, keeping a multitude of coins results in higher requirements for storage and computation at withdrawal. Thus we assume that this scheme is optimized for minimal storage, whereas our divisible scheme is optimized for minimum computation during payment, so that a realistic threshold between the two can be achieved.

Storage:

When keeping many non-divisible coins, \mathcal{U} needs to store $316 + 10k\alpha$ Bytes, where $\alpha = \ln \frac{N}{k}$ as in Section 5.1.1. For a divisible coin, 926 Bytes need to be stored, if all the random numbers that \mathcal{U} needs to compute for coin authentication are pre-computed and stored at memory.

Computation:

As a measurement we use exponentiations performed on a 512 bit modulus with 160 bit exponents; these are used in [15], and our multiple coin scheme. Exponentiations of our divisible scheme are performed modulo a 689 bit modulus with 688 bit exponents, and are thus 7.8 times more costly: as mentioned in Section 5.3.4 above exponentiations take $O(|Q||P|^2)$ steps, where $|P|$ is the size of the modulus and $|Q|$ the size of the exponents used.

At *withdrawal* \mathcal{U} performs $6 + 1.2\alpha$ exponentiations for a multitude of coins and 41.9 for a divisible coin. However, if we assume that \mathcal{U} pre-computes all the values that are to be used at payment, he must also calculate y_3, g_1^N , and the U_i ’s used in the range-bounded commitment: these take an additional 21.3 exponentiations, for a total of 63.

At *payment* α exponentiations need to be performed if multiple coins are kept. For the divisible coin, authentication takes 24 exponentiations for the user—but these have already been performed ahead of time and counted at withdrawal—and 38.8 for the shop. Denomination revelation requires 6.4 exponentiations, of the size in [15],

per node for both user and shop; with divisibility precision \mathcal{N} and k payments, the average number of nodes spent will be

$$\frac{1}{2} \log \mathcal{N} \approx \frac{7}{10} \ln \mathcal{N} ,$$

i.e., half the depth of the tree representing the divisible coin, by assuming that half of the bits of the binary representation of \mathcal{N} will be set. But the equivalent of at least 10 node operations need to be performed so that the shop can verify that \mathcal{U} selected a Williams integer (see Section 3.2.2). Hence, assuming that $\frac{1}{2} \log \mathcal{N} \leq 10$, i.e., $\mathcal{N} \leq 2^{20}$, 64 exponentiations need to be performed at payment for the user and 103 for the shop.

5.5 Universal framing protection

Our notion of framing protection is stronger than usual. In most previous e-cash schemes ([46, 100, 104, 15, 32], etc.), if \mathcal{U} double-spends then he reveals his secret information, which can now be used to spend the, already double-spent, coin multiple times, thus framing the user who now would appear to have spent a coin many times, where in fact he spent it only twice. This is a heavy cost, especially if double spending was due to equipment failure or unintentional oversight.

We suggest the following: At withdrawal \mathcal{S} picks $j \in_R Z_Q$, computes $F = g_2^j$ and uses $B = [N, Y, F]$. At each payment, the user proves knowledge of j using a self-challenging Schnorr proof of knowledge. In this case, j is some secret information that is never revealed—even if \mathcal{U} double-spends—hence F serves to prevent framing of \mathcal{U} . In this sense, F , and all operations involving it, is an optional value.

We note that this addition is minimal, requiring one exponentiation at withdrawal and payment and 64 Bytes of storage, and *it is applicable to all e-cash schemes without any modifications*, since the only requirement is a blind signature on F during withdrawal.

5.6 Open problems

As is the case with all efficient e-cash schemes, it remains an open problem to completely prove the security of our scheme under natural cryptographic assumptions,

such as the DLA. Pointcheval and Stern [107] have made the first step towards such security proofs by proving the unforgeability of Schnorr signature schemes under the random oracle model; the same authors extend their results to blind signatures in [106]. These attempts hint that Brands'-type e-cash may be provable in the random oracle model; this would be a very interesting result.

For our scheme to be provably secure, however, it would also be nice to analyze the security of our Factoring and Diffie-Hellman assumptions. This is an interesting number theoretic problem and it may also be applicable in proving the security of Okamoto's divisible e-cash scheme [100].

Another interesting open problem is to find a way to break the linkability between portions of the same coin, therefore constructing an unlinkable divisible coin scheme with practically unlimited divisibility precision. Such a scheme would need to be compared with our result in Chapter 4 which emulates unlinkable divisible e-cash, in the same way our linkable divisible e-cash scheme is compared with the efficiency of a scheme utilizing a multitude of linkable coins in Section 5.4 above. Constructing unlinkable divisible e-cash has turned out to be a difficult problem; in fact, an alternative direction would be to try to prove the infeasibility of constructing unlinkable divisible e-cash that are more efficient than a collection of unlinkable non-divisible coins. This would be an interesting result, even under some assumptions about coin structure, e.g., assuming divisible coins are represented as a binary tree and/or a coin is a random encryption of the user's identity signed by the bank's signature.

Lastly, we would like to investigate other applications of our efficient range-bounded commitment—not necessarily within electronic cash. In addition, it would be interesting to see whether the range-bounded commitment can be used as a building block for more complex protocols, in a similar way that bit commitment is used for oblivious transfer, encryption and digital signatures.

Chapter 6

Fair off-line e-cash

Cryptography has been instrumental in reducing the involvement of over-head third parties in protocols. For example, a *digital signature scheme* assures a recipient that a judge who is not present at message transmission will nevertheless approve the validity of the signature; similarly, in *off-line electronic cash* the bank, although being off-line during purchases, is assured that if a user double spends he is identified.

Here we suggest the notion of *Indirect Discourse Proofs* with which one can prove indirectly yet efficiently that a third party has a certain future capability, i.e., assure Trustees can trace. The efficient proofs presented here employ the algebraic properties of exponentiation or functions of similar homomorphic nature.

Employing this idea we present the concept of “Fair Off-Line e-Cash” (*FOLC*) system which enables tracing protocols for anonymous off-line electronic cash for identifying either the **coin** or its **owner**. As highlighted in Section 3.3 the need to trace and identify coins with owners/withdrawals was recently identified, in order to avoid blackmailing, money laundering or other attacks, as well as comply with regulatory issues. Previous solutions that assured this traceability, called fair e-cash as they balance the need for anonymity and the prevention of criminal activities, involved third parties at money withdrawals. In contrast, FOLC keeps any third party uninvolved, thus it is “fully off-line e-cash” even when law enforcement is added, i.e., it is off-line w.r.t. law enforcement at withdrawals and off-line w.r.t. the bank at payments.

6.1 Introduction

Direct involvement of parties performing only administrative tasks, e.g., assuring that potential future actions are enabled, is undesirable in electronic transactions. Such excessive involvement increases the over-head and the required synchronization among protocol members. Examples of areas where involvement of a judge in the actual protocol was eliminated are digital signatures, contract signing and simultaneous exchange protocols. In this chapter we suggest a basic idea that avoids direct involvement of administrative third parties as a general tool and use it for constructing Fair Off-Line e-Cash. Third party involvement is replaced by a tool in which an active participant proves that in case of future events like mishaps, attacks and disputes, the administrative party (authority, judge, agents) will be able to take proper actions. We call such proofs “indirect discourse proofs”—where the prover refers to the cryptographic capabilities of a third party rather than invoking the third party itself at the time of proof.

A classical area where involvement of third parties in a transaction has been recognized as an undesirable event is the area of electronic-cash, where off-line systems (as introduced in [32] and described in Chapter 3) eliminate the need of the bank’s, \mathcal{B} , involvement in the payment transaction between a user, \mathcal{U} , and a shop, \mathcal{S} . Users withdraw electronic “coins” from the bank and use them to pay a shop. The shop subsequently deposits the coins back to the bank. In the process users remain anonymous, unless they spend a single coin more than once (double-spend). This last property allows the bank to be off-line during payment.

Unfortunately, it was pointed out that anonymous electronic cash can become a tool for criminal activities [129, 19, 123, 60]. The anonymity provided to the owner of a coin allows for money laundering, illegal purchases, perfect blackmailing and other attacks. Hence any large scale application of e-cash may be thwarted by a government unless there is a mechanism to prevent such criminal activities. This is highlighted in a recent NSA report by Law, Sabet and Solinas [88]. Two previous methods were suggested to solve the criminal activity in off-line e-cash systems: escrowed cash by Brickell, Gemmell and Kravitz [19] and signature-message linking via fair blind signatures by Stadler, Piveteau and Camenisch [123, 21]. However, in contrast with our solution, they both employ on-line fair withdrawals, i.e., the Trustee(s) are

involved in every withdrawal.

In this chapter we suggest employing indirect discourse proofs to implement an electronic cash scheme which is fair, i.e., identifiable by proper Trustees but anonymous otherwise. The user proves that the judge (Trustee) will be able to identify the owner of the coin or the coin that originated from a withdrawal when legally required. Yet the Trustee itself does not have to be present in the withdrawal nor in payment or deposit, similar to a judge in a signature scheme. This is an analogous requirement to the bank not being present in the payment transaction. We thus call these “Fair Off-Line e-Cash” systems (FOLC). These systems prevent money laundering, black-mailing, illegal purchases and similar attacks, and are efficient.

Structure of the chapter: In Section 6.2 we model FOLC and present the various notions of traceability. Then, in Section 6.3 we show the necessity of conditional anonymity and certain cryptographic tools in FOLC. We also present a generic modular extension which can turn existing e-cash schemes into FOLC systems. In Section 6.4, we present a modification of the e-cash protocol in [15]. Then we add indirect discourse proofs to this scheme to provide the owner tracing protocol assuming a trusted shop, in Section 6.5, and then without a trusted shop in Section 6.5.2. The coin tracing protocol is realized in Section 6.6, while Section 6.7 discusses how our protocols can be added to the divisible e-cash schemes presented in the previous chapters. Section 6.8 concludes with open problems.

Related work: The results in this chapter are based on collaborative work with Davida, Frankel and Yung and appear in [55, 43]. Very recently and independently, Camenisch, Maurer and Stadler [20] have suggested an off-line solution to e-cash tracing. Our second, more efficient, indirect discourse proof schemes achieve similar results to [20]. The result in [20] is slightly more efficient, requiring about half the communication and computation overhead at payment, but the respective model is weaker: the bank needs to perform a search in the withdrawal database in order to trace a user from his coins (owner tracing), whereas our indirect discourse proofs allow the bank to perform a search on the, significantly smaller, account database. In fact our system can be used in the model proposed by [20], in which case both systems are equally efficient.

Another instantiation similar to [20] can be performed using a recent result by Stadler [122]; the resulting scheme in this case however would be more than one order

of magnitude slower than our efficient protocols—as also verified by Stadler [121].

We also note that in [79], Jakobsson and Yung introduced the bank robbery attack, in which an attacker obtains coins using a non-standard protocol which makes tracing impossible, or in which an attacker obtains the secret key of the coin issuer(s) and mints money on his own. This type of attack was prevented by the introduction of *dual verification signatures*, i.e., signatures with two modes of verification; typically one off-line for the normal case, and one that requires interaction with a bank or bank proxy which is used after a bank robbery attack. The attack model was further strengthened in [81, 80] by the distribution of functionality, allowing the actions of signing and tracing to be performed by quorums of replaceable participants. In [78] the setting, the protocols and the proofs of security are expanded on, and efficiency improvements are suggested. We do not deal with such attacks in our model but mechanisms against such strong and similar attacks can be added in a modular fashion as suggested in [79].

6.2 Modeling FOLC

Fair off-line electronic cash (FOLC) extends anonymous off-line electronic cash and involves a bank, \mathcal{B} , users, \mathcal{U} , shops, \mathcal{S} , and a collection of Trustees, \mathcal{T} , (judges/escrow agents) which act like one party. It is outside the scope of this work to show how the power of the Trustees can be equally distributed among several parties. This is a problem which can be solved by basic secret-sharing or secure distributed computation techniques. \mathcal{T} should be envisioned as being a single trusted entity. FOLC includes five basic protocols, three of which are the same as in off-line electronic cash: a *withdrawal protocol* with which \mathcal{U} withdraws electronic coins from \mathcal{B} while his account is debited, a *payment protocol* with which \mathcal{U} pays the coin to \mathcal{S} , and a *deposit protocol* with which \mathcal{S} deposits the coin to \mathcal{B} and has his account credited. The two additional protocols, also pictured in Figure 6.1, are conducted between \mathcal{B} and \mathcal{T} :

- The *owner tracing* protocol traces the identity of the owner of a specific coin. In this protocol \mathcal{B} gives to \mathcal{T} the view of a deposit protocol. \mathcal{T} returns a string that contains identifying information which \mathcal{B} can use to identify the owner via the account database.

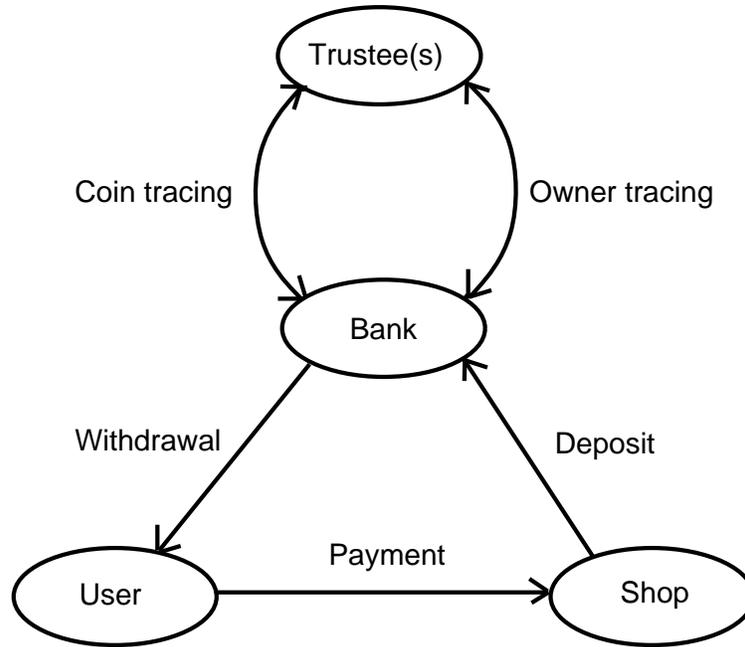


Figure 6.1: Model of Fair Off-Line e-Cash (FOLC).

- The *coin tracing* protocol traces the coin(s) that originated from a withdrawal. In this protocol \mathcal{T} , given the view of a withdrawal protocol from \mathcal{B} , returns some information that originated from this withdrawal. \mathcal{B} can use the returned value to find the coin(s) by accessing its views of the deposit protocols.

Hence, the *owner tracing* protocol allows the authorities to prevent money laundering since they can find the origin of dubious coins. It also allows the authorities to find the identity of a customer that uses a service which is typically anonymous but sometimes requires identification, e.g., when a crime is suspected or as required by the U.S. telephony bill or similar laws worldwide, while still providing anonymity for legitimate users.

The *coin tracing* protocol allows the authorities to find the destination of suspicious withdrawals. This can solve, for example, the blackmailing problem [129]: a customer is blackmailed and forced to anonymously withdraw electronic coins, so that the blackmailer can use these coins without ever being identified, in effect committing a “perfect crime”—of course the victim has to complain and point at withdrawals for

tracing to proceed. The mechanism also enables tracing of activities of a suspect user that is on a criminal list at the time of his withdrawals.

Indirect-discourse proof techniques assure that \mathcal{T} can perform its functions. Note that two facts are important, for efficiency reasons:

- (1) \mathcal{B} supplies only a single view of a withdrawal or deposit protocol to \mathcal{T} during the tracing protocols. Therefore \mathcal{T} does not have to perform any searching but can immediately “open” the requested identifying information.
- (2) The identifying information returned from \mathcal{T} is already included in the corresponding protocols (account opening/deposit). Hence \mathcal{B} only has to perform a search on a presumably indexed database, rather than having to make a computation for each candidate view. Also, in the case of the coin tracing protocol, the identifying information can be broadcasted to all potential shops, in effect “blacklisting” all suspicious coins.

Lastly, we should note that the Trustee has no more power than is required for tracing; in particular the Trustee cannot forge electronic coins nor double-spend undetected.

6.3 General results

In this section we show fundamental requirements for FOLC schemes. We start by showing that perfect anonymity is impossible when tracing is required and that FOLC may require more than the existence of any one-way permutation when implemented based on black-box reductions, otherwise its implementation can serve as a proof separating P from NP. Next we show a basic scheme which achieves tracing, ignoring efficiency, based on any public key encryption, which is believed to require more than the usage of one-way permutation as a black box. The more theoretical investigation of this section gives us better understanding of what is required; in later sections we consider more efficient schemes.

Theorem 6.3.1 *(1) Unconditional untraceability is impossible in FOLC even if only owner tracing or coin tracing is supported. (2) Further, any implementation of FOLC—even if only owner tracing or coin tracing is supported—based on black box reduction from an arbitrary one-way permutation will separate P and NP.*

Proof. (1) Assume the opposite. The information revealed at payment time for a coin which is only spent once must, for owner tracing, allow for \mathcal{T} , which is holding a finite length key, to trace \mathcal{U} . Namely, it can be viewed as an encryption of a user's identity. Unconditional untraceability (which includes unconditional unlinkability of coins) implies that this encryption is unconditional, e.g., a one-time pad. Now by viewing $(\mathcal{U}, \mathcal{B}, \mathcal{S})$ as a single entity, this implies that $(\mathcal{U}, \mathcal{B}, \mathcal{S})$ can encrypt an arbitrary number of messages, one for each coin, to \mathcal{T} ; this fact is illustrated in Figure 6.2. As is well known from Shannon [119], since the encryption is unconditional \mathcal{T} and $(\mathcal{U}, \mathcal{B}, \mathcal{S})$ must share information proportional (linear to) the size of all the plaintexts. This contradicts the fact that \mathcal{T} 's key is of fixed finite length, hence \mathcal{T} can not be off-line during withdrawals. In effect, \mathcal{T} can pre-compute his "engagement" in each withdrawal as is done by Brickell et. al. in [19], but this does not alter the required amount of communication; it only allows a shift to a more convenient time.

Similarly, in case only coin tracing is supported $(\mathcal{U}, \mathcal{B}, \mathcal{S})$ can also be viewed as one party, now producing an encryption of a coin or of some information relevant to the coin based on the Trustee's key; if this encryption is conditionally secure then so is the user's anonymity, since the bank can link a withdrawal to the respective payment. Hence unconditional unlinkability requires this encryption to be unconditional, therefore [119] \mathcal{T} 's key needs to be of length linear to the number of coins; again a contradiction.

(2) Viewing now the two parties $(\mathcal{U}, \mathcal{S})$ and $(\mathcal{B}, \mathcal{T})$, we can achieve "key exchange" between them, by letting \mathcal{U} dynamically choose two names from the exponentially many user names. The first of these signifies a "0" and the second a "1". Then, run a withdrawal for each name (between the two parties) and the respective payments (internal to the first party) and the deposit for one of the two names (between the parties), and finally the owner tracing protocol (internal to the second party). At this point the name chosen by the first party is recognized by the second and the exchange of one bit is performed. Generalizing to n bits is then straightforward. Note that without the last internal interaction between \mathcal{B} and \mathcal{T} , the identity (name) is strongly concealed from the participants due to the anonymity requirement, thus from outsiders, who are based on information available in non-internal interactions, as well. This proves that the protocol is indeed a "key exchange". Figure 6.3 summarizes this fact.

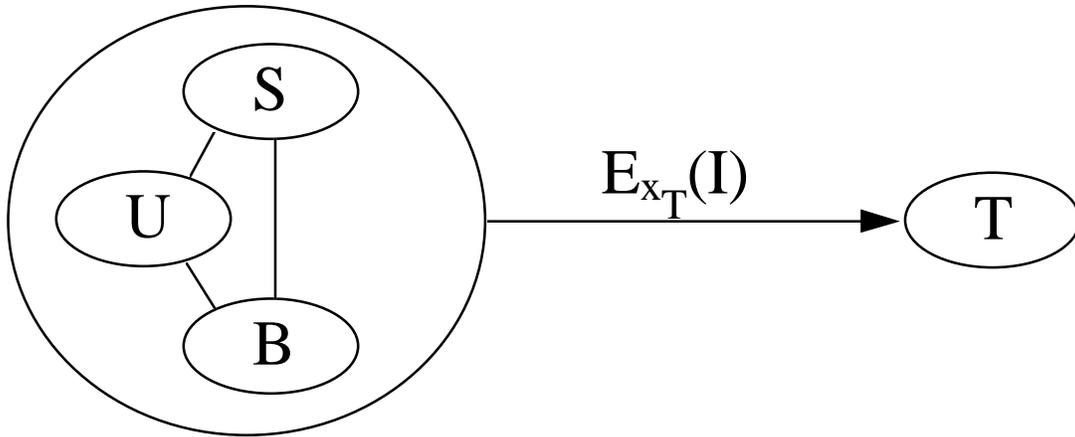


Figure 6.2: Conditional untraceability is necessary in FOLC.

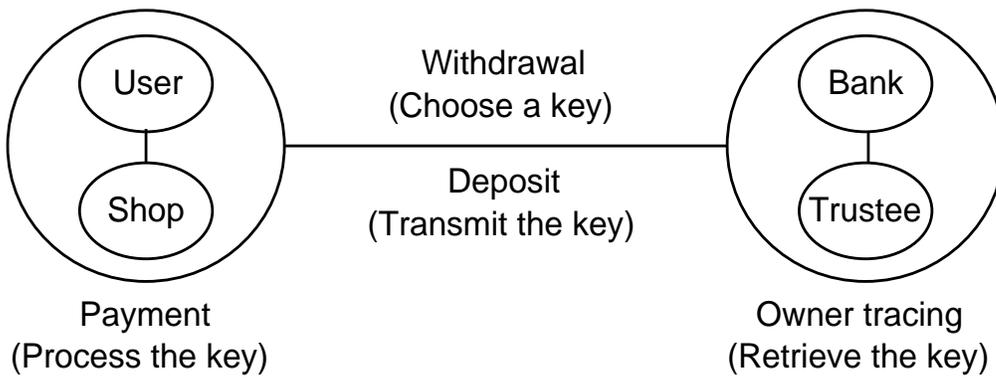


Figure 6.3: FOLC achieves key exchange.

Hence, based on the work of Impagliazzo and Rudich in [74], it is most plausible that we need more than a one-way permutation used as a black-box for constructing FOLC, unless we believe that such a construction is the way to solve the longstanding most important open problem in complexity of P vs. NP, since it will separate the two complexity classes.

In case only coin tracing is supported key exchange can be similarly achieved by grouping the parties as above and letting \mathcal{U} dynamically choose a coin, now serving as a key, uniformly at random from the exponentially many possible coins. Essentially \mathcal{U} chooses his blinding information uniformly at random; this results in a random coin. Then, run the withdrawal (between the two parties), the payment (internal to the first party), and finally the coin tracing protocol (internal to the second party). At this point the coin chosen by the first party is recognized by the second, while it is hidden from outside parties due to the anonymity requirement. The key exchanged in this case is the coin—minus any information that is common between coins, such as a common prefix or suffix. \square

We now demonstrate an optimal polynomially secure scheme; all we assume about the payment is that the information used by the user in the withdrawal and payment protocols can be extended with random strings. This is the case in all existing methods. Typically, this extendible information is the one that is blinded and given to the bank to sign. We reduce such systems in the presence of public-key cryptosystems to FOLC with owner and/or coin tracing.

Theorem 6.3.2 *Given off-line e-cash and public-key encryption, there exists a FOLC system in which anonymity is computationally guaranteed.*

Proof. We assume the existence of an off-line e-cash system in which (1) at withdrawal the user sends to the bank some (extendible) information to process and (2) this information is later presented as a string of bits.

To achieve tracing we perform the following. \mathcal{T} has a public key published. First, the bank “flips coins into the user’s well”. Namely, the user selects a random string and commits to it by encrypting it using \mathcal{T} ’s public key in a probabilistic, i.e., cryptographically secure, encryption scheme [69]; the bank then sends back a random string; the resulted coins are the XOR of the bank’s string and the user’s string. Thus these $poly(k)$ coins are known only to the user and are random as long as either the user

or the bank use random strings. We call this field the tag. Then, the bank and the user engage in the withdrawal of a coin. The user first encrypts his identity with a probabilistic encryption, based on \mathcal{T} 's public key.

The user extends the information he sends to the bank with a probabilistic encryption of his already encrypted identity based on \mathcal{T} 's public key and a probabilistic encryption of the tag. Namely, the user concatenates to the coin information two randomized encryptions. The user proves in zero-knowledge the NP statement that the two encryptions sent represent his encrypted known name and the tag. This is achievable since we assume encryption functions, as shown by Goldreich et. al. in [67].

After the verification the bank is assured that the coin contains the user's name under the Trustee's encryption and a tag. If we require *coin tracing* the coins into the well are opened by \mathcal{T} , by having him decrypt the user's original commitment, and the tag value gets known.

Payment is done by opening the coin and the plaintext of the two encryptions, i.e., the tag and the user's probabilistically encrypted identity. Deposit means forwarding the coin information as in the original deposit protocol along with the plaintexts to the bank. For *owner tracing* the bank can forward the encryption of the user's name to \mathcal{T} for decryption and identification. This implements FOLC. \square

6.4 The basic off-line electronic cash scheme

We now concentrate on efficient schemes which we build based on a modification of [15]. This modification conforms to our security model as presented in Section 5.3.3 under the "withdrawal protocol" assumption appearing in [15]. The scheme is therefore as secure as [15] except that its untraceability is computationally rather than unconditionally secure; as shown in theorem 6.3.1 above, reducing to conditional anonymity is unavoidable for FOLC.

The setup, withdrawal and payment protocols are described below. In the deposit protocol the shop just transfers to the bank a transcript of a payment protocol.

CA establishment: A Certificate Authority for the users', bank's and trustees' public keys is established which is independent of the bank or trustees. (See [124, 2, 59] for legal aspects of establishing a CA).

Trustee’s setup protocol: (performed once by \mathcal{T})

Trustee(s) choose a private key $X_{\mathcal{T}} \in_R Z_Q$ and publish the public key $f_2 = g_2^{X_{\mathcal{T}}}$.

Bank’s setup protocol: (performed once by \mathcal{B})

Primes P and Q are chosen such that $|P - 1| = \delta + k$ for a specified constant δ , and $P = \gamma Q + 1$, for a specified integer γ . Then a unique subgroup G_Q of prime order Q of the multiplicative group Z_P^* and generators g, g_1, g_2 of G_Q are defined.

Secret key $X_{\mathcal{B}} \in_R Z_Q$ is created. We assume, for simplicity, that only one denomination is used. A different key for each denomination is necessary otherwise, as highlighted in Section 5.3.2.

Hash functions $\mathcal{H}, \mathcal{H}_0, \mathcal{H}_1, \dots$, from a family of collision intractable hash functions are also defined.

\mathcal{B} publishes $P, Q, g, g_1, g_2, g_3, (\mathcal{H}, \mathcal{H}_0, \mathcal{H}_1, \dots)$ and its public keys $h = g^{X_{\mathcal{B}}}$, $h_1 = g_1^{X_{\mathcal{B}}}$, $h_2 = g_2^{X_{\mathcal{B}}}$, $h_3 = g_3^{X_{\mathcal{B}}}$.

User’s setup (account opening) protocol: (performed for each user \mathcal{U})

The bank \mathcal{B} associates user \mathcal{U} with $I = g_1^{u_1}$ where $u_1 \in_R Z_Q$ is generated by \mathcal{U} and $g_1^{u_1} g_2 \neq 1$.

\mathcal{U} also proves to \mathcal{B} , using e.g., the Schnorr identification scheme [116], that he knows how to represent I w.r.t. g_1 . The user’s communication is signed by the user and is verifiable by the bank with the user’s public key certificate generated by the CA.

\mathcal{U} computes $z' = h_1^{u_1} h_2 = (I g_2)^{X_{\mathcal{B}}}$.

Withdrawal: (over an authenticated channel between \mathcal{B} and \mathcal{U})

We refer the reader to Section 3.1.3 for a description of Brands’ withdrawal protocol. Our basic protocol presented in this section is similar to this. The only modification is that B is “split” into two halves,

$$B = [B_1, B_2] ,$$

with $B_1 = g_1^{x_1}, B_2 = g_2^{x_2}$ and $x_1, x_2 \in_R Z_Q$ as previously. This is necessary in order to allow A to also be split into two parts at payment; this in turn is needed for our indirect discourse proofs as will be seen in Section 6.5.1 below. The withdrawal protocol appears in Figure 6.9 of Section 6.6, with some additions for coin tracing.

anonymity, which states that it is difficult to find r , given $[g^{a_0}, g^{b_0}], [g^{a_1}, g^{b_1}]$ and $(a_r b_r), (a_{1-r} b_{1-r}), r \in_R \{0, 1\}$, where a_i, b_i ($i = 0, 1$) are primes, in effect also assuming the difficulty of factoring.

Theorem 6.4.1 *Let \mathcal{U} and \mathcal{S} agree on date/time, i.e., their clocks are synchronized. Under the matching Diffie-Hellman assumption and assuming the security of Brands' withdrawal protocol and that a hash-function behaving like a random oracle exists, the protocols presented above form a computationally untraceable off-line coin scheme satisfying unreuseability, unforgeability, unexpandability and untraceability.*

Proof. Unreuseability, unforgeability, unexpandability and untraceability have been formally defined in Section 5.3.3 in the previous chapter, where it was also shown that Brands' scheme [15] conforms to these requirements under the “withdrawal protocol” and “random oracle” assumptions. We first prove that the bank's and shop's security is not decreased. The only modification from [15] for our protocols is that the responses regarding A are verified independently, using A_1, B_1 and A_2, B_2 respectively. The information conveyed to \mathcal{S} and \mathcal{B} is a superset of that shown in the original Brands' [15] protocol. Moreover, \mathcal{S} and \mathcal{B} do not transmit different information than in the original protocol and the verifications performed test for everything the original verifications do. Hence unreuseability is guaranteed. Also, the additional information shown to the shop does not allow it to forge coins or spend the same coin as a user different than the one participating in the payment protocol, or else the user would be able to do the same since the same information is available to the user. Thus unforgeability and unexpandability are still satisfied and the bank's and shops' security is maintained.

Next we prove that anonymity of the user is maintained under the Matching Diffie-Hellman assumption, i.e., untraceability holds. Assume that the bank can trace the user, that is, given the views of two withdrawal protocols and their corresponding payment protocols it can distinguish with non-negligible probability which withdrawal matches which payment. All we need to show is that there exists a polynomial converting algorithm available to the bank which given an instance of the matching Diffie-Hellman problem translates it into an instance of the tracing problem; hence if the bank can trace the user it will break the matching Diffie-Hellman assumption.

The converting algorithm takes as input $[g^{a_0}, g^{a_0 b_0}]$, $[g^{a_1}, g^{a_1 b_1}]$ and $g^{b_r}, g^{b_{1-r}}$, $r \in_R \{0, 1\}$ and produces two withdrawal views W_r, W_{1-r} in which $I_j = g_1^{b_j}$, for $j = \{r, 1-r\}$, and two payment views P_0, P_1 in which $A_{1,i} = g_1^{a_i b_i}$, $A_{2,i} = g_2^{a_i}$, for $i = \{0, 1\}$. The conversion from base g to g_1, g_2 is possible since the bank knows their correspondence (recall that g, g_1, g_2 are chosen by the bank). To create a withdrawal view the algorithm uses the value w chosen by the bank (after the bank sees $g_1^{b_j}$) and a random c' to calculate a', b', r' .

For each payment view, r_1, r_2 and d are chosen at random. B_1, B_2 are then computed so that the verification holds: $B_1 = A_1^{-d} g_1^{r_1}$, $B_2 = A_2^{-d} g_2^{r_2}$. Hence, the output of \mathcal{H}_1 on $(A, B_1, B_2, I_S, \text{Date/Time})$ is chosen to be the value d ; thus in the simulation the function used is \mathcal{H}'_1 , which is still a random oracle (since d is random) and is similar to \mathcal{H}_1 except for its value on some polynomially many points. Nevertheless, this part of the simulated view is indistinguishable from the real protocol since any two random oracles are, by definition, indistinguishable. We note that a typical argument involving a random oracle—see, e.g., [8]—is similar to ours: If the function is assumed to be a random oracle then to derive a reduction we may afford in the construction to modify the oracle on a polynomial size domain. Then, assuming the original function is indeed a random oracle, the security of the design is validated via such reductions. For the remaining of the view, z is computed as A^{X_B} ; a, b are created randomly (for a random exponent e , $a = g^e, b = A^e$); and r is computed as $cX_B + e$, where c is the appropriate hash value.

It is easy to verify the correctness of these views and that a legitimate user could create the same payment view for each withdrawal by selecting u, v such that $u = c/c'$ and $e = wu + v$. Finally, since as shown by Brands in [14] every coin, independent of w , occurs with the same probability for each user, the distribution of the simulated views is indistinguishable from that of valid views, hence the tracing algorithm has a non-negligible probability of success. This can then be used to break the matching Diffie-Hellman problem; a contradiction on the matching Diffie-Hellman assumption. \square

The next sections are devoted to the additional protocols of the FOLC scheme and their claims of correctness and security.

6.5 Owner tracing

We now demonstrate the concept of indirect discourse proofs by directly applying it to owner tracing, first when the shop is considered trusted by the Trustees in Section 6.5.1, and then without this trust in Section 6.5.2. We remark that trusting the shop is actually not so terrible. If a user \mathcal{U}' is framed by \mathcal{U} and \mathcal{S} , he proves to the bank where he spent all his coins or he reveals his secret information for all his withdrawals. Such an audit will convince \mathcal{T} that \mathcal{U} and \mathcal{S} colluded. However, anonymity is lost for \mathcal{U}' , and \mathcal{U} escapes identification.

6.5.1 Trustees trust shop

The basic idea here is simple. An off-line coin by its nature has its owner's identity embedded in it; in our system we also encrypt the user's identity using the El Gamal [45] probabilistic public key encryption system in such a way that the encryption is linked to the coin. Hence, Trustees can open the ciphertext to obtain the identity. An indirect discourse proof during payment assures the shop that the encrypted identity is the same as the one embedded in the coin; this is illustrated in Figure 6.6. Additions to the basic protocol are limited to the payment protocol and are presented below. Note that the payment transcript that is shown at deposit need not include these additions, since the indirect discourse proof is not transferable; hence the deposit protocol remains unchanged.

Recall that the Trustees' secret and public keys are $X_{\mathcal{T}}$ and $f_2 = g_2^{X_{\mathcal{T}}}$ respectively. **Additions to the payment protocol** are limited to the indirect discourse proof, presented in Figure 6.5 (i.e., $[i] + [i']$ is new flow i in payment.)

Owner tracing is now straightforward: the bank sends the El Gamal ciphertext (D_1, D_2) to the Trustees who decrypt it to obtain the identity I of the coin's owner. The shop has to be trusted to verify that (D_1, D_2) indeed encrypt the owner's identity.

Security and efficiency

We now show that the indirect discourse proof is probabilistically secure under no cryptographic assumptions and it does not impair user untraceability of the basic system. Thus, Theorem 6.4.1 holds for this protocol as well; while owner tracing is possible under the security of the withdrawal protocol, the existence of random

\mathcal{U}		\mathcal{S}
$m \in_R Z_Q$		
$D_1 = I g_2^{X_T m}$		
$D_2 = g_2^m$	[1'] $\xrightarrow{D_1, D_2}$	$D_2 \stackrel{?}{\neq} 1$
		$s_0, s_1, s_2 \in_R Z_Q$
		$D' = D_1^{s_0} g_2^{s_1} D_2^{s_2}$
	[2'] $\xleftarrow{D', f_2'}$	$f_2' = f_2^{s_0} g_2^{s_2}$
$V = \mathcal{H}_1((D')^s / (f_2')^{ms})$		
	[3'] \xrightarrow{V}	$V \stackrel{?}{=} \mathcal{H}_1(A_1^{s_0} A_2^{s_1})$

Figure 6.5: Indirect discourse proof: this protocol proves to \mathcal{S} that (D_1, D_2) is an El Gamal encryption of I based on f_2 , where I is the same identity as the one embedded in A_1 .

oracle-like hash functions and the assumption that the shops are trusted.

Theorem 6.5.1 *The indirect discourse proof is correct, i.e., if both \mathcal{U} and \mathcal{S} are legitimate the verification in step 3' succeeds; and it is computationally secure for \mathcal{S} , i.e., \mathcal{S} is convinced of the construction of (D_1, D_2) in relation to (A_1, A_2) . Furthermore, assuming hash functions that behave like random oracles, it is secure for \mathcal{U} , i.e., untraceability holds under the matching Diffie-Hellman assumption.*

Proof. First we prove correctness. Let $F_y(x) = x^y \bmod P$. Observe that $p_1 = F_s(D_1)/F_{ms}(f_2) = I^s g_2^{X_T ms} / g_2^{X_T ms} = A_1$, $p_2 = F_s(g_2) = g_2^s = A_2$, and $p_3 = F_s(D_2)/F_{ms}(g_2) = g_2^{ms - ms} = 1$. If \mathcal{U} and \mathcal{S} behave correctly, i.e., as specified in the protocol, then \mathcal{S} 's verification in flow [3'] passes with probability 1. Just note that $V = \mathcal{H}_1(p_1^{s_0} p_2^{s_1})$.

Next, we demonstrate that the system is secure in two parts: secure for \mathcal{U} (the indirect discourse proof is minimal-knowledge) and secure for \mathcal{S} (\mathcal{S} is convinced of the construction of (D_1, D_2)). For any $s'_0 \in Z_Q$ there exist unique $s'_1, s'_2 \in Z_Q$ such that $f'_2 = f_2^{s'_0} g_2^{s'_2}$ and $D' = D_1^{s'_0} g_2^{s'_1} D_2^{s'_2}$. W.l.o.g., we assume that $D', f'_2 \in G_Q$. (If D' or f'_2 is of the form $v_1 v_2$ where $v_1 \notin G_Q, v_2 \in G_Q$ then we treat it as v_2 .) Hence, since \mathcal{S} is trusted not to reveal (s_0, s_1, s_2) , even if \mathcal{U} has unlimited power he can still not guess s_0 with probability greater than $\frac{1}{Q}$.

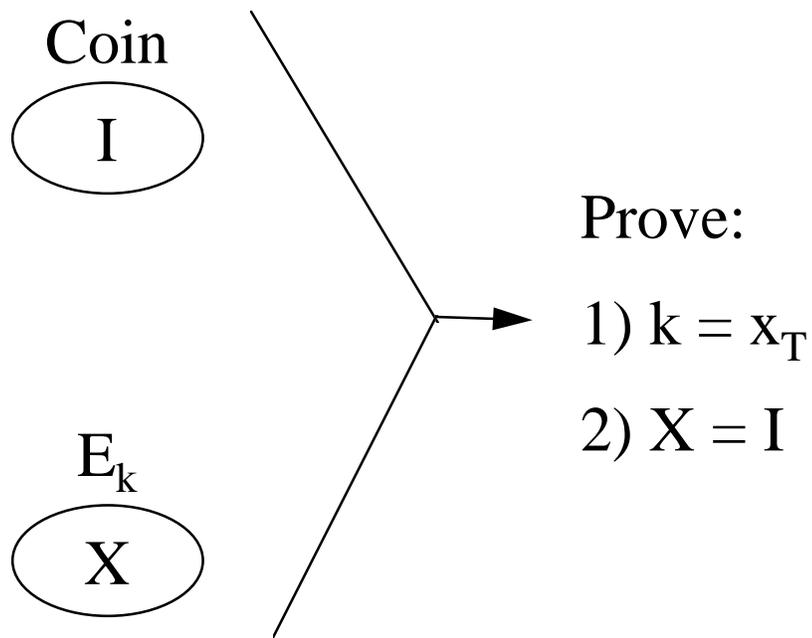


Figure 6.6: Indirect discourse proof for owner tracing when the shops are trusted.

We first prove security of \mathcal{S} by contradiction on finding s_0 . For simplicity, we assume that \mathcal{U} has access to an oracle that solves discrete logarithms.

Let $D_2 = g_2^m$ and let $D_1 \neq g_1^{u_1} g_2^{mX\tau}$, i.e., let D_1, D_2 be constructed incorrectly. In particular, let $D_1 = g_1^{u'_1} g_2^{mX\tau}$, with $u'_1 \neq u_1$ since Q is a prime. \mathcal{U} can clearly compute $Z = (D')^s / (f'_2)^{ms} = (g_1^{u'_1 s})^{s_0} A_2^{s_1}$ since he knows m and s (using discrete log oracle and since Q is a prime). If \mathcal{U} could compute V , w.l.o.g. we assume he could also compute any \tilde{A} where $V = \mathcal{H}_1(\tilde{A})$. Hence he could also compute $Z' = \tilde{A}/Z = g_1^{(u'-u'_1)ss_0}$ for some $\tilde{A} = g_1^{u'ss_0} A_2^{s_1}$ (for some u') where $V = \mathcal{H}_1(\tilde{A})$. Then for this \tilde{A} , \mathcal{U} could compute the log of Z' base $Y = g_1^{(u'-u'_1)s}$, and hence some unique s'_0 for that Z' (using the oracle, and since Q is a prime). Since with extremely small probability $V = \mathcal{H}_1(G_Q)$ and Q is a prime, \mathcal{U} can guess s_0 with probability better than $\frac{1}{Q}$; a contradiction. That is, one of the \tilde{A} will be correct ($u' = u_1$) and that one will return the correct s_0 ; the rest will be wrong but \mathcal{U} has strictly less than Q choices to choose from.

We next show that the only information revealed to \mathcal{S} is (D_1, D_2) , which is an El Gamal encryption of the user's identity I . But as shown in [125] under the matching Diffie-Hellman the El Gamal encryption is semantically secure; hence the El Gamal encryption preserves user anonymity, under the matching Diffie-Hellman assumption. To complete the anonymity proof we show that \mathcal{S} can simulate the history if he is allowed to choose (D_1, D_2) , i.e., that the proof leaks nothing other than the aforementioned El Gamal encryption. The simulator encrypts a D_1, D_2 encoding some user \mathcal{U}' for any coin C consisting of, say, A'_1, A'_2 . It chooses s_0, s_1, s_2 with a random distribution. It then simulates the three flows by responding in [3'] with $\mathcal{H}_1((A'_1)^{s_0} (A'_2)^{s_1})$. Since \mathcal{H}_1 is a random oracle, this distribution is indistinguishable from the real one. Hence, the indirect discourse proof does not leak anything more than the El-Gamal encryption (D_1, D_2) of the user's identity, I , which in turn is semantically secure, i.e., it leaks no information, under the matching Diffie-Hellman assumption. \square

In terms of **efficiency**, this protocol poses indeed minimal additional requirements to \mathcal{U} and \mathcal{S} (3 multi-exponentiations each). Most importantly, \mathcal{T} is off-line during withdrawal, payment and deposit.

6.5.2 Shops are not trusted

The idea here is to prove via a minimal-knowledge proof that the encryption (D_1, D_2) , from Section 6.5 above, is correctly related to an encryption that the bank can decrypt and check at deposit time. To this end we use Schnorr knowledge proofs [116] to show the equality of logarithms of the two encryptions, where the bases of the logarithms are appropriately verified; this is illustrated in Figure 6.8. These kinds of proofs are particularly efficient.

We first describe the proof of equality of logarithms.

Proving equality of logarithms

Setup: A probabilistic polynomial-time (*p.p.t.*) prover \mathcal{P} and a p.p.t. verifier \mathcal{V} .

Common input is A, B, a, b , with a, b generators of G_Q .

Secret input to \mathcal{P} is x , such that $A = a^x, B = b^x$.

The **proof** appears in Figure 6.7.

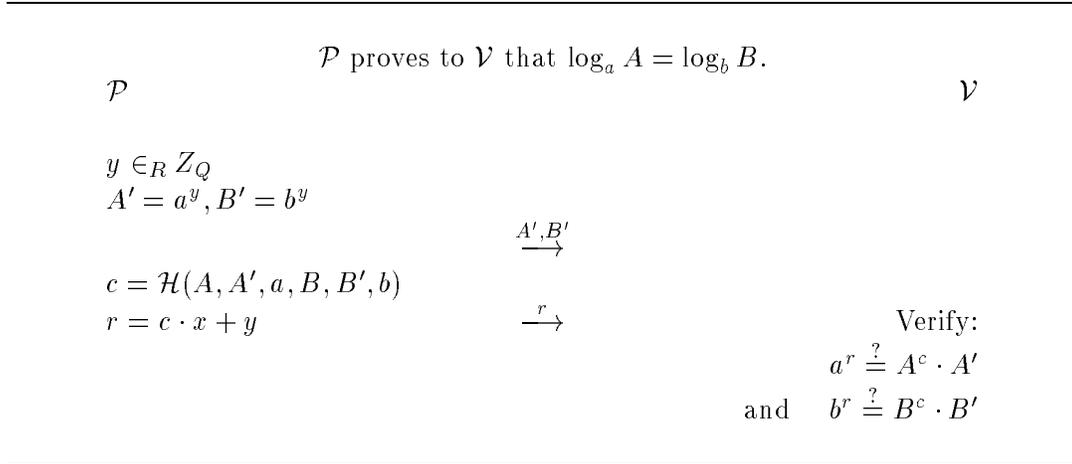


Figure 6.7: Proof of equality of logarithms.

The proof is essentially a set of parallel Schnorr knowledge proofs and *it can be used to prove equality of more than two logarithms*.

Correctness is easily proven. Assume $x_1 = \log_a A \neq \log_b B = x_2$, $A' = a^{y_1}$, $B' = b^{y_2}$, and that for some c, r : $a^r = A^c A'$, $b^r = B^c B'$. Then $r = cx_1 + y_1 = cx_2 + y_2$, i.e., $y_1 = c(x_2 - x_1) + y_2 \neq y_2$ since $x_1 \neq x_2$. Now, for a different challenge $c' \neq c$ and

response r' , it must be that

$$r' = c'x_1 + y_1 = c'x_2 + y_2 ,$$

otherwise the verifier would not accept. Substituting y_1 for its value based on y_2 the above becomes

$$c'x_1 + c(x_2 - x_1) + y_2 = c'x_2 + y_2 ,$$

or, after simplification, $c' = c$, a contradiction to the choice of c' . Hence, \mathcal{P} can succeed in at most one challenge c if the logarithms are not equal; that c is determined by the choice of A', B' , and \mathcal{P} has a probability of cheating $1/Q$, i.e., equal to the probability of guessing the verifier's challenge.

We can now proceed to describe the payment protocol.

We remind the reader that $h_2 = g_2^{X_B}$ is the bank's and $f_2 = g_2^{X_T}$ the Trustees' public keys.

Payment: (Setup and withdrawal are as in the basic system.)

- **\mathcal{U} computes the encryptions:** \mathcal{U} picks $m \in_R Z_Q$, and computes $D_1 = If_2^m, D_2 = g_2^m$, as before, and also $D_T = D_1^s, E_1 = I^s h_2^{ms} = A_1 h_2^{ms}$ and $E_2 = g_2^{ms}$.
- **Conduct indirect discourse proof of (D_1, D_2)** from Section 6.5.
- **\mathcal{U} proves that the encryptions are correctly related:** For this, \mathcal{U} proves the equality of the following logarithms

$$\log_{g_2}(A_2) = \log_{D_2}(E_2) = \log_{D_1}(D_T) \text{ and } \log_{h_2/f_2}(E_1/D_T) = \log_{g_2}(E_2) ,$$

using the parallel Schnorr proof in Figure 6.7. For completeness we include the details of the proof here.

1. \mathcal{U} picks $x_3 \in_R Z_Q$, and computes: (x_2 is from $B_2 = g_2^{x_2}$ in the basic protocol)

$$y_{hf} = (h_2/f_2)^{x_3}, \quad y_2 = g_2^{x_3}, \quad y_{D_1} = D_1^{x_2}, \quad y_{D_2} = D_2^{x_2}$$

2. \mathcal{U} sends all the above plus $r_2 = ds + x_2$ and $r_3 = em + x_3$ to \mathcal{S} , with $d = \mathcal{H}_2(A_1, B_1, A_2, B_2, D_2, E_2, y_{D_2}, D_T, D_1, y_{D_1}, \text{date/time}, ID_S)$ (instead of d as in the basic system), $e = \mathcal{H}_3(E_2, y_2, E_1, D_T, y_{hf}, \text{date/time}, ID_S)$.

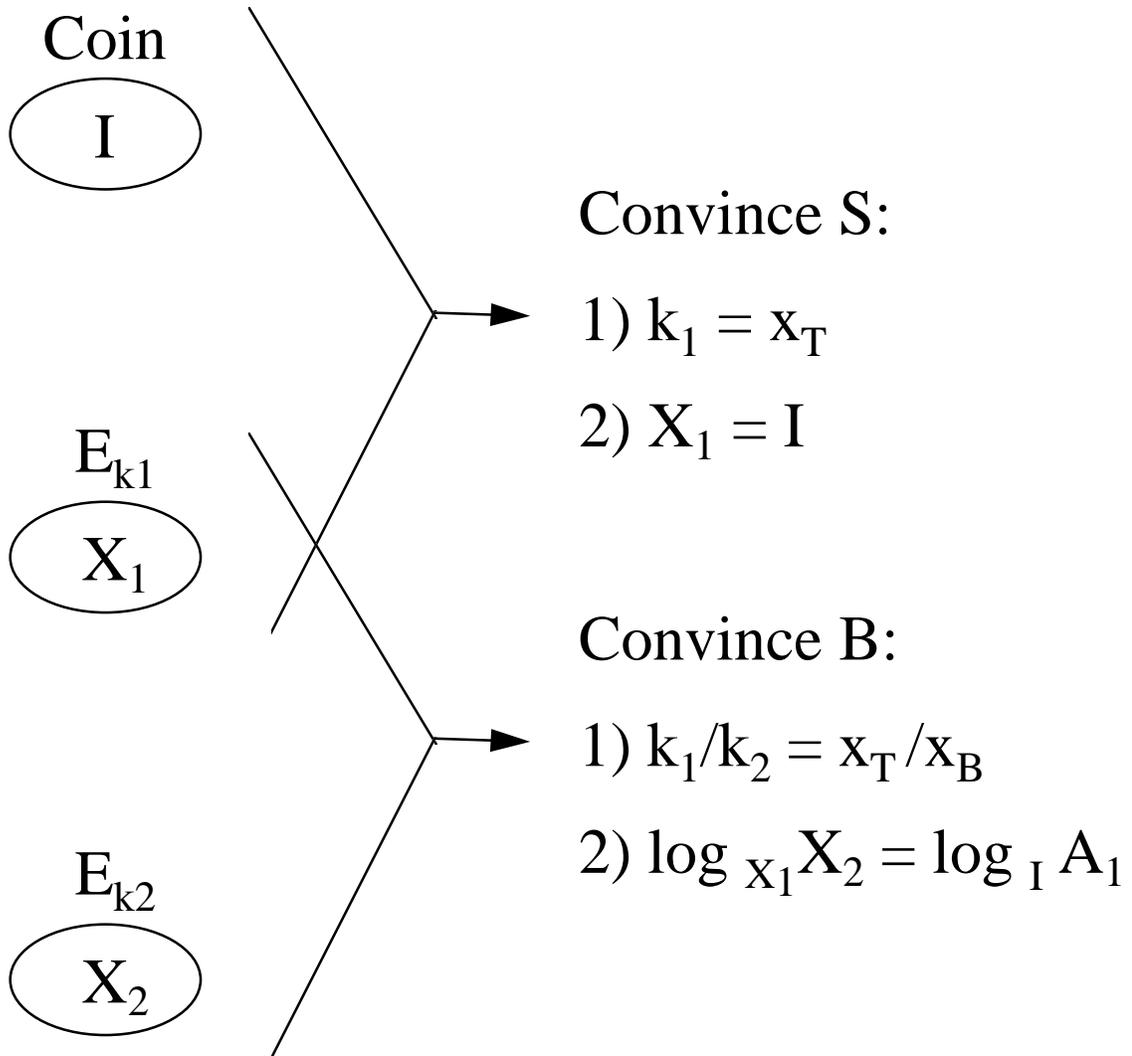


Figure 6.8: Owner tracing when shops are not trusted.

3. \mathcal{S} verifies (together with $(g_1)^{r_1} \stackrel{?}{=} A_1^d B_1$):

$$(g_2)^{r_2} \stackrel{?}{=} (A_2)^d B_2, \quad (D_2)^{r_2} \stackrel{?}{=} (E_2)^d y_{D_2}, \quad (D_{\mathcal{T}})^{r_2} \stackrel{?}{=} (D_1)^d y_{D_1} \quad (6.1)$$

$$\text{and } (g_2)^{r_3} \stackrel{?}{=} (E_2)^e y_2, \quad (h_2/f_2)^{r_3} \stackrel{?}{=} (E_1/D_{\mathcal{T}})^e y_{hf} \quad (6.2)$$

Deposit: \mathcal{S} sends a payment transcript to \mathcal{B} (the indirect discourse proof can be omitted), and \mathcal{B} verifies the proofs and signatures.

Owner tracing is unchanged, that is the Trustees need only decrypt (D_1, D_2) . However the shop need not be trusted since the bank can also verify the validity of the encryption.

Security and efficiency

We now overview the security of the protocol, in the sense that the protocol is secure for \mathcal{S} (\mathcal{S} can accept the coin, knowing that it is valid for deposit), for \mathcal{B} (\mathcal{B} is convinced that the Trustees can obtain \mathcal{U} 's identity from the coin) and for \mathcal{U} (\mathcal{S} and \mathcal{B} cannot trace \mathcal{U} , nor do they end up with a coin that they can spend).

(6.1), (6.2) are the self-challenging Schnorr identification schemes which prove that \mathcal{U} knows how to compute the corresponding logarithms, and that the following relationships hold (similar to [31] for Simultaneous Discrete Log):

$$(6.1) \iff \log_{g_2}(A_2) = \log_{D_2}(E_2) = \log_{D_1}(D_{\mathcal{T}}) \iff$$

$$\exists s \in Z_Q: A_2 = g_2^s \wedge E_2 = D_2^s \wedge D_{\mathcal{T}} = D_1^s \quad (6.3)$$

$$(6.2) \iff \log_{h_2/f_2}(E_1/D_{\mathcal{T}}) = \log_{g_2}(E_2) \iff$$

$$\exists M \in Z_Q: E_2 = g_2^M \wedge E_1/D_{\mathcal{T}} = (h_2/f_2)^M \quad (6.4)$$

First we show that \mathcal{S} is convinced that (E_1, E_2) is an El-Gamal signature of \mathcal{U} 's identity: \mathcal{S} verified that (D_1, D_2) correctly “encode” \mathcal{U} 's identity (via the indirect discourse proof), i.e., $\exists m \in Z_Q: g_2^m = D_2 \stackrel{(6.3)}{=} (E_2)^{1/s} \stackrel{(6.4)}{=} g_2^{M/s}$, i.e., $M = ms$. Then \mathcal{S} can infer that $D_{\mathcal{T}} \stackrel{(6.3)}{=} D_1^s = I^s f_2^{ms}$ and that $E_1 \stackrel{(6.4)}{=} (h_2/f_2)^M D_{\mathcal{T}} = (h_2/f_2)^{ms} D_{\mathcal{T}} = (h_2/f_2)^{ms} I^s f_2^{ms} = I^s h_2^{ms}$, i.e. that $(E_1 = I^s h_2^{ms}, E_2 = g_2^{ms})$ is the correct encryption of $A_1 = I^s$. Thus, \mathcal{S} can accept the coin, knowing that \mathcal{B} will accept it for deposit.

Next we show security for \mathcal{B} . \mathcal{B} , at deposit time, verifies (by decrypting (E_1, E_2)) that (E_1, E_2) is a correct encryption of A_1 , i.e., $E_1 = A_1 h_2^M = I^s h_2^M$ for $E_2 = g_2^M$. Then, it can infer that $(h_2/f_2)^M D_{\mathcal{T}} \stackrel{(6.4)}{=} E_1 = I^s h_2^M \iff D_{\mathcal{T}} = I^s f_2^M \stackrel{(6.3)}{=} D_1^s \iff D_1 = I f_2^{M/s} = I f_2^m$, for $m = M/s$. Then \mathcal{B} can verify that $D_2 \stackrel{(6.3)}{=} E_2^{1/s} \stackrel{(6.4)}{=} g_2^{M/s} = g_2^m$, i.e., that $(D_1 = I f_2^m, D_2 = g_2^m)$ is an El Gamal encryption with f_2 on I , hence that the Trustees can obtain \mathcal{U} 's identity.

In terms of untraceability for \mathcal{U} , the protocol reveals the following values: $D_1, D_2, E_1, E_2, D_{\mathcal{T}}, A_1, A_2$ in addition to the parallel Schnorr proofs. These values can simplified to:

$$I f_2^m, g_2^m, I^s, g_2^{ms}, f_2^{ms}, g_2^s, f_2,$$

that is E_1 does not reveal anything more to the bank and shop, since it could be reconstructed by the bank from A_1, g_2^{ms} , while $D_{\mathcal{T}}$ reveals f_2^{ms} , since A_1 can be divided out. If we assume that the bank knows the representation of g_1 w.r.t. g_2 then security is summarized in Theorem 6.5.2 below, which includes the notion of *shop protection*: this is a security requirement applicable in the context of FOLC that enhances the security model of basic and divisible e-cash.

Definition 6.5.1 (Shop protection against owner/coin tracing) *A fair off-line electronic cash scheme is said to afford shop protection if, with probability overwhelming in the security parameter n , no p.p.t. TM can construct a coin that can be successfully paid to a legitimate shop and which does not allow for owner and/or coin tracing.*

Theorem 6.5.2 *Unforgeability, unreusability and unexpandability are satisfied, owner tracing is possible, and shop protection w.r.t. owner tracing is guaranteed under the security of the withdrawal protocol and the existence of hash functions behaving like random oracles. Untraceability holds under the following assumption:*

For security parameter k , P a prime with $|P - 1| = \delta + k$ for a specified constant δ , for $g \in Z_P^$ a generator of prime order $Q = (P - 1)/\gamma$ for a specified integer γ and for $a_i, b_i, c_i, d_i \in_R Z_Q$ random for $i = 0, 1$ and with $b_0 \not\equiv b_1 \pmod{Q}$, given $g^d, [g^{c_0}, g^{a_0}, g^{b_0 a_0}, g^{c_0 a_0}, g^{d c_0 a_0}, g^{b_0} g^{d c_0}], [g^{c_1}, g^{a_1}, g^{b_1 a_1}, g^{c_1 a_1}, g^{d c_1 a_1}, g^{b_1} g^{d c_1}]$ and $g^{b_r}, g^{b_1 - r}, r \in_R \{0, 1\}$, find r with probability better than $1/2 + 1/k^c$ for any constant c for large enough k .*

Proof. Unforgeability, unreuseability and unexpandability are not affected by the additional steps, since the same checks performed by the shop and bank in the basic scheme are duplicated here.

As we have seen earlier, owner tracing succeeds as long as the user cannot forge the parallel Schnorr proofs. But the security of Schnorr proofs of knowledge is a requirement for the security of the withdrawal protocol, which we have already assumed to hold.

Shop protection holds, as seen earlier in this section, if the parallel Schnorr proofs and the indirect discourse proof cannot be forged. But the indirect discourse proof succeeds as long as the shop is legitimate, as we have seen in Theorem 6.5.1, and this part is completed.

In order to prove untraceability under the above assumption we use a standard simulation argument, similar to the proof in Theorem 6.4.1. Assume that the bank can trace the user, namely that given the views of two withdrawal protocols and their corresponding payment protocols it can distinguish with non-negligible probability which withdrawal matches which payment. We then need to show that there exists a polynomial converting algorithm (translator) available to the bank which given an instance of the above problem translates it into an instance of the tracing problem; This way we prove that if the bank can trace the user it can use the tracing algorithm as an oracle to break the above assumption.

The translator takes as input $g^d, [g^{c_0}, g^{a_0}, g^{b_0 a_0}, g^{c_0 a_0}, g^{d c_0 a_0}, g^{b_0} g^{d c_0}], [g^{c_1}, g^{a_1}, g^{b_1 a_1}, g^{c_1 a_1}, g^{d c_1 a_1}, g^{b_1} g^{d c_1}]$ and $g^{b_r}, g^{b_{1-r}}, r \in_R \{0, 1\}$ and proceeds as in the proof of Theorem 6.4.1 to produce two withdrawal views W_r, W_{1-r} in which $I_j = g_1^{b_j}$, for $j = \{r, 1-r\}$, and two payment views P_0, P_1 in which

$$\begin{aligned}
A_{1,i} &= g_1^{a_i b_i} , \\
A_{2,i} &= g_2^{a_i} , \\
D_{2,i} &= g_2^{c_i} , \\
E_{1,i} &= A_{2,i} h_2^{c_i a_i} , \\
E_{2,i} &= g_2^{c_i a_i} , \\
f_2 &= g_2^d , \\
D_{1,i} &= g_1^{b_i} f_2^{c_i} , \\
D_{\mathcal{T},i} &= A_{2,i} f_2^{c_i a_i} ,
\end{aligned}$$

for $i = \{0, 1\}$. Note that the bulk of the translation can be performed by the following substitutions:

$$\begin{aligned} d &= X_{\mathcal{T}} , \\ c &= m , \\ b &= s . \end{aligned}$$

Again, the conversion from base g to g_1, g_2 is possible since the bank knows their correspondence (recall that g, g_1, g_2 are chosen by the bank). To complete each withdrawal view the algorithm uses the value w chosen by the bank (after the bank sees $g_1^{b_j}$) and a random c' to calculate a', b', r' .

For the completion of each payment view, r_1, r_2, r_3 and d, e are chosen at random. Then the auxiliary values for the parallel Schnorr proofs are computed, so that the proofs succeed. In particular the translator computes (for simplicity we omit the index i) $B_1 = A_1^{-d} g_1^{r_1}, B_1 = A_2^{-d} g_2^{r_2}$ as in the simulation of the basic protocol in Theorem 6.4.1 and in addition:

$$\begin{aligned} y_{D_2} &= E_2^{-d} D_2^{r_2} , \\ y_{D_1} &= D_1^{-d} D_{\mathcal{T}}^{r_2} , \\ &\text{and} \\ y_2 &= E_2^{-e} g_2^{r_3} , \\ y_{hf} &= (E_1/D_{\mathcal{T}})^{-e} (h_2/f_2)^{r_3} . \end{aligned}$$

Then the outputs of the random oracles $\mathcal{H}_2, \mathcal{H}_3$ on inputs $(A_1, B_1, A_2, B_2, D_2, E_2, y_{D_2}, D_{\mathcal{T}}, D_1, y_{D_1}, \text{date/time}, ID_{\mathcal{S}})$ and $(E_2, y_2, E_1, D_{\mathcal{T}}, y_{hf}, \text{date/time}, ID_{\mathcal{S}})$ are chosen to be d and e respectively, following the same “trick” used in Section 6.4.1 in the proof of Theorem 6.4.1.

The withdrawal views are then completed as in the proof of Theorem 6.4.1.

The proof is completed by verifying that the set of withdrawal views are compatible with the payment views, i.e., that there exists a set of choices that any user could have performed such that the withdrawal views W_r, W_{1-r} would result in the payment views P_0, P_1 ; this guarantees that the tracing oracle would succeed in finding r , which could then be used to break the above assumption. This verification is straightforward and identical for the case examined in the basic protocol; thus omitted here.

□

Efficiency: The protocol poses minimal additional communication and computation requirements, while still keeping \mathcal{T} off-line in all cases. In particular \mathcal{U} needs to perform an additional 7 exponentiations and \mathcal{S} 8. (Note that the computation for $g_2^{r_2} \stackrel{?}{=} A_2^d B_2$ has already been counted in the basic protocol.) Thus a total of 10 multi-exponentiations are needed for owner tracing for the user and 11 for the shop. In addition the bank at deposit only needs to perform the equality of logarithms verifications and the decryption of (E_1, E_2) , since the indirect discourse proof is not transferable. Hence a total of only 9 exponentiations by the bank are needed for owner tracing.

6.6 Coin tracing

Coin tracing is performed with a simple modification of the basic scheme of Section 6.4. Here the user constructs the coin A based on three generators g_1, g_2, g_3 : the first two are used as in the previous sections, for coin verification and owner tracing, while the third is used for coin tracing. We describe the withdrawal, payment and deposit protocols here, as the setup protocols do not need to be changed for coin tracing.

Withdrawal: (over an authenticated channel between \mathcal{B} and \mathcal{U} where \mathcal{U} signs its transmissions)

The protocol appears in Figure 6.9.

In this protocol \mathcal{U} ends up with a Schnorr-type [116] signature on $A = (I g_2)^s g_3$, where s is a random number, chosen by \mathcal{U} and kept secret. Thus the modification here is the construction of A , which previously was $A = (I g_2)^s$.

A closer look at the protocol shows that the new form of A allows the shop at payment, and consequently the bank at deposit, to verify that the value A'_1 used to construct the coin is of the form $g_1^u g_2 g_3^{(s^{-1})}$. This holds under the assumption that the withdrawal protocol is a restricted blind signature. Thus, $A'_1 / (I g_2) = g_3^{(s^{-1})}$ and $\log_{A'_1 / (I g_2)} g_3 = s$. But as can be seen in the withdrawal protocol the bank verifies that $\log_{f_2} A'_2 = \log_{A'_1 / (I g_2)} g_3 = s$, therefore that $A'_2 = f_2^s$. This property can be used for coin tracing.

Payment: (performed between \mathcal{U} and \mathcal{S} over an anonymous channel)

The only addition to the payment protocols of the previous sections is that now \mathcal{S}

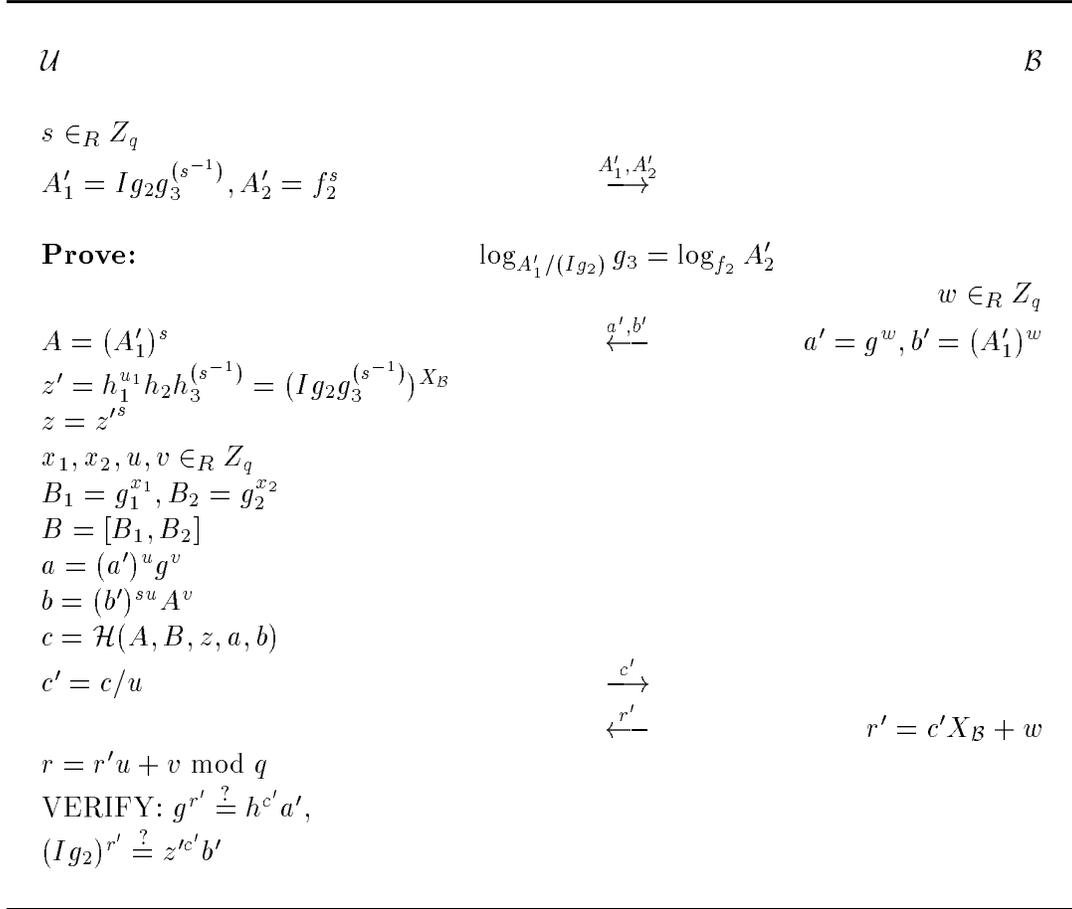


Figure 6.9: The withdrawal protocol supporting coin tracing.

also verifies that A is of the form $A = (Ig_2)^s g_3 = A_1 A_2 g_3$.

Deposit again consists of transferring a transcript of the payment to the bank.

Coin tracing: the bank sends to the Trustees A'_2 ; the Trustees then compute $A_2'^{(x_2^{-1})} = g_2^s = A_2$, which allows coin tracing since A_2 is a value that appears on the spent coin.

Efficiency and security

As it can be seen the Trustees are kept off-line at all times, while the necessary additions are limited to 7 exponentiations for \mathcal{U} at withdrawal and 2 for \mathcal{B} . Also, coin tracing involves only one exponentiation for the Trustees.

Security for the shop and bank remain intact; namely, unreuseability, unforgeability

and unexpandability and the ability to perform coin tracing rely on the withdrawal protocol assumption and the existence of random oracles. Untraceability is more complex to prove and along the lines of the previous section we limit ourselves to stating the following Theorem:

Theorem 6.6.1 *Unforgeability, unreusability and unexpandability are satisfied, coin tracing is possible and shop protection w.r.t. coin tracing is guaranteed under the security of the withdrawal protocol and the existence of hash functions behaving like random oracles. Untraceability holds under the following assumption:*

For security parameter k , P a prime with $|P - 1| = \delta + k$ for a specified constant δ , for $g \in Z_P^$ a generator of prime order $Q = (P - 1)/\gamma$ for a specified integer γ and for $a_i, b_i, c_i, d_i \in_R Z_Q$ random for $i = 0, 1$ and with $b_0 \not\equiv b_1 \pmod{Q}$, given $[g^{a_0}, g^{b_0 a_0}], [g^{a_1}, g^{b_1 a_1}]$ and $[g^{b_r}, g^{1/a_r}], [g^{b_{1-r}}, g^{1/a_{1-r}}]$, $r \in_R \{0, 1\}$, find r with probability better than $1/2 + 1/k^c$ for any constant c for large enough k .*

Similarly, security for the complete system, including owner tracing when the shops are not trusted and coin tracing is secure, as stated in the following:

Theorem 6.6.2 *Unforgeability, unreusability and unexpandability are satisfied, owner and coin tracing are possible and shop protection w.r.t. owner and coin tracing are guaranteed under the security of the withdrawal protocol and the existence of hash functions behaving like random oracles. Untraceability holds under the following assumption:*

For security parameter k , P a prime with $|P - 1| = \delta + k$ for a specified constant δ , for $g \in Z_P^$ a generator of prime order $Q = (P - 1)/\gamma$ for a specified integer γ and for $a_i, b_i, c_i, d_i \in_R Z_Q$ random for $i = 0, 1$ and with $b_0 \not\equiv b_1 \pmod{Q}$, given $g^d, [g^{c_0}, g^{a_0}, g^{b_0 a_0}, g^{c_0 a_0}, g^{d c_0 a_0}, g^{b_0 g^{d c_0}}], [g^{c_1}, g^{a_1}, g^{b_1 a_1}, g^{c_1 a_1}, g^{d c_1 a_1}, g^{b_1 g^{d c_1}}]$ and $[g^{b_r}, g^{1/a_r}], [g^{b_{1-r}}, g^{1/a_{1-r}}]$, $r \in_R \{0, 1\}$, find r with probability better than $1/2 + 1/k^c$ for any constant c for large enough k .*

We omit the proofs of the above Theorems as they are straightforward to derive from the proofs of Theorems 6.4.1 and 6.5.2 in the previous sections. Shop protection

for coin tracing requires only that A'_1, A'_2 are of the correct form; it is straightforward to see that this is the case under the assumption that the withdrawal protocol is secure, and given the verification by the shop at payment.

6.7 Fair divisible e-cash

As explained in Section 5.1, there are two ways of providing divisible electronic cash. The first scheme, utilizing a multitude of non-divisible coins as described in Section 5.1.1, consists of a modular addition to Brands' scheme [15]; hence our FOLC scheme can be directly applied to it by using our basic protocol, as described in Section 6.4, in place of the Brands' scheme. Naturally, the additions needed at payment for both owner and coin tracing need only be performed once per payment, for only one of the coin instances spent (i.e., for one of the B 's). Security is exactly as in our FOLC scheme.

When adding a Trustee to our divisible e-cash with practically unlimited payments, i.e., our scheme described in Section 5.1.2, it is advisable to use the second variant of that scheme, as described in Section 5.3.2, with $u \neq 0$. This

- (1) allows owner tracing to point to the account database—instead of the withdrawal database—thus maintaining efficient tracing, as well as
- (2) disassociates security of our divisible and fair schemes; that is, security of the basic divisible scheme remains intact, as does security of our tracing protocols.

When using the second variant, the withdrawal and payment protocols of our divisible scheme remain intact, as they cover the functionality of the basic protocol of Section 6.4. The coin and owner tracing additions at withdrawal and payment also remain intact; in this case the identity used is $I' = g_3^u$, with g_3^{uq}, g_2^q substituting for the values A_1, A_2 respectively. Security of the two schemes is decoupled: users can forge coins, or their anonymity can be broken, if our divisible scheme is broken; regardless of this, however, the bank can still trace users as long as our FOLC scheme remains secure.

If the first variant is used, with $u = 0$, then security of our tracing protocols is slightly decreased to the levels offered by our divisible scheme; i.e., the user is allowed to forge his identity with probability $\frac{1}{2^n}$ instead of $\frac{1}{2^{|Q-1|}}$ as in our FOLC scheme, where n is a sufficiently large number (40 for our efficiency comparison in

Scheme	Exact payments?	Space (Bytes)	Time (exponentiations)	
			Withdrawal	Payment
The divisible scheme	YES	926	63	64
Multiple Coins	YES	$316 + 10k \ln \frac{N}{k}$	$6 + 1.2 \cdot k \ln \frac{N}{k}$	$1.2 \cdot \ln \frac{N}{k}$
FOLC	YES	—	+9.6	+10.2

Table 6.1: Efficiency of our Fair Divisible off-line electronic cash under some sample security parameters; numbers are rounded for easy comparison. Exponentiations are shown for the user and assume a modulus of 512 bits with exponents of 160 bits.

Section 5.4) but nevertheless smaller than $|Q - 1|$. Again $I_W = g_1^p$ is used as the user’s identity, with g_1^N, g_2^q substituting for A_1, A_2 respectively. Owner tracing in this case does not lead to the account database but instead provides a link between the withdrawal and deposit protocols of a specific coin; thus identification of over-spenders requires searching the—much larger—withdrawal database for the withdrawal view of the specific coin. In this case not all the tracing additions we have devised make sense; in particular, the tracing additions of Sections 6.5 are redundant since a link between the withdrawal and payment is provided directly by the coin tracing protocol. Namely, the Trustees only need to compute $A_2^{X_{\mathcal{T}}} = A'_2$ in order to establish this link. (We remind the reader that A_2 is a value appearing at payment, while A'_2 appears at withdrawal and $X_{\mathcal{T}}$ is the Trustees’ secret key.)

For clarity, we include the efficiency calculations of our divisible e-cash and multiple coin schemes in Table 6.1. A detailed description of the assumptions and parameters is given in Section 5.4; we note that the second variant of our divisible scheme (i.e., $u \neq 0$) is used for the comparison.

6.8 Open problems

The most important open problem is to construct proofs of security of FOLC based on as few assumptions as possible. As highlighted in Sections 3.5 and 5.6 it seems that random oracles is a difficult assumption to remove, but our basic protocol may be closer to provability under the random oracle model, after some recent results by Pointcheval and Stern [107, 106]. In particular, our security proof is based on the Matching Diffie-Hellman and the Brands’ withdrawal assumptions; [107, 106] have

provided significant insight on the possibility of provability of Brands' withdrawal protocol under the random oracle model.

The Matching Diffie-Hellman assumption is an interesting number theoretic problem; we would like to see an analysis of its security. As it is a subset of our "Factoring and Diffie-Hellman" assumptions presented in Section 5.3.3 its analysis can be seen as a first step towards the characterization of these more complex problems.

Last but not least, the notion of indirect discourse proofs deserves closer attention. Such proofs allow a prover to verify that a presented string is correctly related to a signed string; thus enabling for example the shop to check that a probabilistic encryption contains the same identity as the one included in a signed, but anonymous, coin. We would like to see applications of such proofs outside electronic cash, to settings where direct involvement of parties is undesirable. For example, an application to fair blind signatures (see [123] for a definition) is immediate, since these are a subset of fair off-line e-cash. Other applications may be in the field of escrowed identity, as defined by Kilian and Petrank in [82].

Chapter 7

Conclusion

The main goal of this thesis is to show the feasibility of electronic cash as a means for providing anonymous electronic payments. We provide what appear to be the two major missing links towards practical implementation of anonymous off-line electronic cash schemes, namely capability for exact payments and control of user anonymity.

We investigate both conceivable approaches towards exact payments: (a) withdrawing multiple coins of various denominations and (b) providing for coin divisibility. We present a provably optimal algorithm for maintaining multiple coins, while we improve existing results in divisible electronic cash by three orders of magnitude. We furthermore analyze the applicability of each method depending on system parameters, to conclude that our divisible approach is more efficient when a large budget and/or great precision of payments is required; the opposite is true for our multiple coin scheme.

To control user anonymity we present two modular additions that allow for tracing of malicious users by an assigned set of trusted parties (trustees), while limiting the trustees' involvement to one decryption operation per tracing request. We show how our additions can be applied to a simple token-based scheme as well as our divisible schemes, resulting into efficient and secure *fair divisible electronic cash*.

Throughout this thesis we focus on both efficiency and provable security of the proposed systems. We aim to provide schemes that can be applied in current smart-cards without sacrificing security. We thus provide proofs for our protocols, based on a formal security model. In the course we make some technical and theoretical contributions which enhance our understanding of electronic cash protocols.

In particular, we identify security problems in existing electronic cash systems and show how to correct them. We formalize the notion of range-bounded commitments—a method for checking that a committed value is within a certain range without revealing information about it. We provide a formal security model for electronic cash which, unlike previous proposals, also covers unlinkability of electronic coins; our schemes' security is proved based on this model. Lastly, we present general cryptographic requirements for anonymity controlled systems.

Finally we should note that two novel protocols appear in this thesis. An efficient range-bounded commitment protocol and an indirect discourse proof—a way to prove that a third party will be able to perform some future action, without involving that party in the proof. These protocols are useful primitives for more complex systems, therefore we believe that their applicability extends beyond electronic cash.

Bibliography

- [1] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *Siam J. Comput.*, 17(2):194–209, April 1988.
- [2] Information Security Committee of the Section on Science American Bar Association and Technology. Draft digital signature guidelines, January 1996. Available online at <http://www.state.ut.us/ccjj/digsig/dsut-gl.htm> The guidelines are currently being revised.
- [3] D. Atkins, M. Graaf, A. K. Lenstra, and P. C. Leyland. The magic words are squeamish ossifrage. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology. Proc. of Asiacrypt '94 (Lecture Notes in Computer Science 917)*, pages 263–277. Springer-Verlag, 1995. Wollongong, Australia, Nov. 28–Dec. 1.
- [4] L. Babai. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM Symp. Theory of Computing, STOC*, pages 421–429, May 6-8, 1985.
- [5] L. Babai and S. Moran. Arthur-merlin games: A randomized proof system and a hierarchy of complexity classes. *Journal of Computer Science*, 36(2):254–276, 1988.
- [6] M. Bellare, J. A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. iKP – a family of secure electronic payment protocols, 1995. The most recent version is available at <http://www.zurich.ibm.com/Technology/Security/extern/ecommerce/>.

- [7] M. Bellare and S. Micali. How to sign given any trapdoor function. In *Proceedings of the twentieth annual ACM Symp. Theory of Computing, STOC*, pages 32–52, May 2–4, 1988.
- [8] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *First ACM journal on Computer and Communications security*, 1993. Available at <http://www-cse.ucsd.edu/users/mihir/crypto-papers.html>.
- [9] M. Bellare and P. Rogaway. The exact security of digital signatures—how to sign with RSA and Rabin. In U. Maurer, editor, *Advances in Cryptology, Proc. of Eurocrypt '96*, pages 399–416. Springer-Verlag, 1996. Zaragoza, Spain, May 11–16.
- [10] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM Symp. Theory of Computing, STOC*, pages 1–10, May 2–4, 1988.
- [11] M. Blum. Coin flipping by telephone — a protocol for solving impossible problems. In *digest of papers COMPCON82*, pages 133–137. IEEE Computer Society, February 1982.
- [12] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Advances in Cryptology. Proc. of Crypto 84 (Lecture Notes in Computer Science 196)*, pages 289–299. Springer-Verlag, New York, 1985. Santa Barbara, August 1984.
- [13] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *Siam J. Comput.*, 13(4):850–864, November 1984.
- [14] S. Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, CWI (Centre for Mathematics and Computer Science), Amsterdam, 1993. anonymous ftp: <ftp.cwi.nl/pub/CWIreports/AA/CS-R9323.ps.zip>.

- [15] S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology — Crypto '93, Proceedings (Lecture Notes in Computer Science 773)*, pages 302–318. Springer-Verlag, 1993. Available at <http://www.cwi.nl/ftp/brands/crypto93.ps.Z>.
- [16] S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology — Pre-Proceedings of Crypto '93*, 1993.
- [17] S. Brands, D. Chaum, R. Cramer, N. Ferguson, and T. Pedersen. Transaction systems with observers, August 13 1992. Unpublished manuscript.
- [18] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988.
- [19] E. F. Brickell, P. Gemmell, and D. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *Symposium on Distributed Algorithms (SODA)*, Albuquerque, NM, 1995. Available at <http://www.cs.sandia.gov/~psgemme/>.
- [20] J. Camenisch, U. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. In *Esorics '96*, Italy, 1996. To appear. Available at <http://www.inf.ethz.ch/personal/camenisc/publications.html>.
- [21] J. Camenisch, J. M. Piveteau, and M. Stadler. An efficient fair payment system. *ACM-CCS*, March 1996.
- [22] A. Chan, Y. Frankel, P. MacKenzie, and Y. Tsiounis. Mis-representation of identities in e-cash schemes and how to prevent it. In *Advances in Cryptology — Proceedings of Asiacrypt '96 (Lecture Notes in Computer Science 1163)*, pages 276–285, Kyongju, South Korea, November 3–7 1996. Springer-Verlag. Available at <http://www.ccs.neu.edu/home/yiannis/pubs.html>.
- [23] A. Chan, Y. Frankel, and Y. Tsiounis. An efficient off-line electronic cash scheme as secure as RSA. Research Report NU-CCS-96-03, Northeastern University, Boston, Massachusetts, 1995.

- [24] A. Chan, Y. Frankel, and Y. Tsiounis. How to break and repair e-cash protocols based on the representation problem. Research Report NU-CCS-96-05, Northeastern University, Boston, Massachusetts, 1996.
- [25] A. Chan, Y. Frankel, and Y. Tsiounis. Off-line electronic cash as secure as RSA, 1996. Unpublished manuscript.
- [26] A. Chan, Y. Frankel, and Y. Tsiounis. Range-bounded commitment and its application to efficient e-cash, 1996. Submitted for publication. International patent pending.
- [27] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A. T. Sherman, editors, *Advances in Cryptology. Proc. Crypto'82*, pages 199–203, Santa Barbara, 1983. Plenum Press N. Y.
- [28] D. Chaum. Security without identification: transaction systems to make Big Brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.
- [29] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM Symp. Theory of Computing, STOC*, pages 11–19, May 2–4, 1988.
- [30] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In D. Chaum and W. L. Price, editors, *Advances in Cryptology — Eurocrypt '87 (Lecture Notes in Computer Science 304)*, pages 127–141. Springer-Verlag, Berlin, 1988. Amsterdam, The Netherlands, April 13–15, 1987.
- [31] D. Chaum, J.-H. Evertse, J. van de Graaf, and R. Peralta. Demonstrating possession of a discrete logarithm without revealing it. In A. Odlyzko, editor, *Advances in Cryptology. Proc. of Crypto '86 (Lecture Notes in Computer Science 263)*, pages 200–212. Springer-Verlag, 1987. Santa Barbara, California, U.S.A., August 11–15.
- [32] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology — Crypto '88 (Lecture Notes in Computer Science)*, pages 319–327. Springer-Verlag, 1990.

- [33] D. Chaum and T.P. Pedersen. Transferred cash grows in size. In *Advances in Cryptology — Eurocrypt '92, Proceedings (Lecture Notes in Computer Science 658)*, pages 390–407. Springer-Verlag, 1993.
- [34] D. Chaum and T.P. Pedersen. Wallet databases with observers. In E. Brickell, editor, *Advances in Cryptology — Crypto '92, Proceedings (Lecture Notes in Computer Science)*, pages 90–106. Springer-Verlag, New York, 1993. Santa Barbara, California.
- [35] A proposed federal information processing standard for an escrowed encryption standard (EES). Federal Register, July 30, 1993.
- [36] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transaction on Information Theory*, IT-30(4):587–594, July 1984.
- [37] D. Coppersmith, A. Odlyzko, and R. Schroepfel. Discrete logarithms in $GF(p)$. *Algorithmica*, pages 1–15, 1986.
- [38] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, page 353. MIT Press/McGraw-Hill, 1990.
- [39] R. Cramer and T. Pedersen. Improved privacy in wallets with observers. In *Advances in Cryptology: Eurocrypt '93, Proceedings (Lecture Notes in Computer Science 765)*, pages 329–343. Springer-Verlag, 1993.
- [40] I. B. Damgård. Collision free hash functions and public key signature schemes. In D. Chaum and W. L. Price, editors, *Advances in Cryptology — Eurocrypt '87 (Lecture Notes in Computer Science 304)*. Springer-Verlag, Berlin, 1988. Amsterdam, The Netherlands, April 13–15, 1987.
- [41] I. B. Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In *Advances in Cryptology, Proc. of Crypto '88 (Lecture Notes in Computer Science)*, pages 328–335. Springer-Verlag, 1988. Berlin, 1990.
- [42] S. D'Amigo and G. Di Crescenzo. Methodology for digital money based on general cryptographic tools. In *Advances in Cryptology, Proc. of Eurocrypt '94*, pages 157–170. Springer-Verlag, 1994. Italy, 1994.

- [43] G. Davida, Y. Frankel, Y. Tsiounis, and M. Yung. Anonymity control in e-cash. In *1st Financial Cryptography conference*, Anguilla, BWI, February 24-28 1997. Available at <http://www.ccs.neu.edu/home/yiannis/pubs.html>.
- [44] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(6):644–654, November 1976.
- [45] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.
- [46] T. Eng and T. Okamoto. Single-term divisible electronic coins. In *Advances in Cryptology — Eurocrypt '94, Proceedings*, pages 306 – 319, New York, 1994. Springer-Verlag.
- [47] P. Fahn and M. J. B. Robshaw. Results from the RSA factoring challenge. Technical Report TR-501, RSA Laboratories, January 1995.
- [48] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. In *Proceedings of the Nineteenth annual ACM Symp. Theory of Computing, STOC*, pages 210–217, May 25–27, 1987.
- [49] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.
- [50] N. Ferguson. Single term off-line coins. In *Advances in Cryptology — EUROCRYPT '93, (Lecture Notes in Computer Science 765)*, pages 318–328. Springer-Verlag, 1993.
- [51] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. Odlyzko, editor, *Advances in Cryptology, Proc. of Crypto '86 (Lecture Notes in Computer Science 263)*, pages 186–194. Springer-Verlag, 1987. Santa Barbara, California, U. S. A., August 11–15.
- [52] A. Fiat and A. Shamir. Unforgeable proofs of identity. In *Securicom 87*, pages 147–153, March 4–6, 1987. Paris, France.
- [53] Y. Frankel, P. Gemmell, and M. Yung. Witness-based cryptographic program checking and robust function sharing. In *Proceedings of the twenty eighth annual*

- ACM Symp. in Theory of Computing, STOC*, 1996. To appear. Available at <http://www.cs.sandia.gov/~psgemme/>.
- [54] Y. Frankel, B. Patt-Shamir, and Y. Tsiounis. Exact analysis of exact change. In *Proceedings of the 5th Israeli Symposium on the Theory of Computing Systems (ISTCS)*, Ran-Gatan, Israel, June 17–19 1997. Available at <http://www.ccs.neu.edu/home/yiannis/pubs.html>.
- [55] Y. Frankel, Y. Tsiounis, and M. Yung. Indirect discourse proofs: achieving fair off-line e-cash. In *Advances in Cryptology, Proc. of Asiacrypt '96 (Lecture Notes in Computer Science 1163)*, pages 286–300, Kyongju, South Korea, November 3–7 1996. Springer-Verlag. International patent pending. Available at <http://www.ccs.neu.edu/home/yiannis/pubs.html>.
- [56] Y. Frankel and M. Yung. Escrow encryption systems revisited: Attacks, analysis and designs. In *Advances in Cryptology, Proc. of Crypto '95 (Lecture Notes in Computer Science 963)*, pages 222–235, Santa Barbara, California, August 1995. Springer-Verlag.
- [57] M. Franklin and M. Yung. Secure and efficient off-line digital money. In *Proceedings of the twentieth International Colloquium on Automata, Languages and Programming (ICALP 1993), (Lecture Notes in Computer Science 700)*, pages 265–276. Springer-Verlag, 1993. Lund, Sweden, July 1993.
- [58] M. K. Franklin. *Complexity and security of distributed protocols*. PhD thesis, Columbia University, New York, 1993.
- [59] A. M. Froomkin. The essential role of trusted third parties in electronic commerce, October 14 1996. Available at <http://www.law.cornell.edu/jol/froomkin.html>.
- [60] A. M. Froomkin. Flood control on the information ocean: living with anonymity, digital cash, and distributed databases, 1996. Available on-line at <http://www.law.cornell.edu/jol/froomkin.html>.
- [61] Z. Galil, S. Haber, and M. Yung. Cryptographic computations: secure fault-tolerant protocols and the public-key model. In C. Pomerance, editor, *Advances*

- in Cryptology, Proc. of Crypto '87 (Lecture Notes in Computer Science 293)*, pages 135–155. Springer-Verlag, 1988. Santa Barbara, California, U.S.A., August 16–20.
- [62] O. Goldreich. Foundations of cryptography (fragments of a book). Available at <http://www.wisdom.weizmann.ac.il/people/homepages/oded/frag.html>.
- [63] O. Goldreich. Foundations of cryptography, 1989. Class notes.
- [64] O. Goldreich, H. Krawczyk, and M. Luby. On the existence of pseudorandom generators. In *Proceedings of the 29th Annual Symp. on Foundations of Computer Science (FOCS)*, pages 12–24. IEEE Computer Society Press, 1988.
- [65] O. Goldreich and L. A. Levin. A hard-core predicate to any one-way function. *21st Proceedings of the Nineteenth annual ACM Symp. Theory of Computing, STOC*, pages 25–32, 1989.
- [66] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth annual ACM Symp. Theory of Computing, STOC*, pages 218–229, May 25–27, 1987.
- [67] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, July 1991.
- [68] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proc. 14th. Annual Symp. on the Theory of Computing*, pages 365–377, San Francisco, May 1982.
- [69] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.
- [70] S. Goldwasser, S. Micali, and C. Rackoff. Knowledge complexity of interactive proofs. In *Proc. 17th STOC*, pages 291–304, 1985.
- [71] J. Hastad. On using RSA with low exponent in a public key network. In Hugh C. Williams, editor, *Advances in Cryptology: Crypto '85, Proceedings (Lecture Notes in Computer Science 218)*, pages 403–408. Springer-Verlag, 1986. Santa Barbara, California, U.S.A., August 18–20.

- [72] D. Frank Hsu and Xing-De Jia. Construction of distributed loop networks. *SIAM Journal on Discrete Mathematics*, 7(1):57–71, 1994.
- [73] Hua. *Introduction to Number Theory*. Springer, New York, 1982.
- [74] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the twenty first annual ACM Symp. Theory of Computing, STOC*, pages 44–61, May 15–17 1989.
- [75] RSA Security Inc. Official web site: <http://www.rsa.com/>. Frequently asked questions (FAQ).
- [76] K. Ireland and M. Rosen. *A classical introduction to modern number theory*. Springer-Verlag, New York, 1982.
- [77] Edited by J. van Leeuwen. *Handbook of theoretical computer science*. MIT Press, Cambridge, MA, 1990.
- [78] M. Jakobsson. *Privacy vs. Authenticity*. PhD thesis, University of California, San Diego, 1997.
- [79] M. Jakobsson and M. Yung. Revokable and versatile e-money. In *Proceedings of the third annual ACM Symp. on Computer and Communication Security*, March 1996.
- [80] M. Jakobsson and M. Yung. Applying anti-trust policies to increase trust in a versatile e-money system. In *Advances in Cryptology—Proceedings of Financial Cryptography '97*, British West Indies, February 1997.
- [81] M. Jakobsson and M. Yung. Distributed ‘magic ink’ signatures. In *Advances in Cryptology—Proceedings of Eurocrypt '97*, pages 450–464, Konstanz, Germany, May 1997. Springer-Verlag, New York.
- [82] J. Kilian and E. Petrank. Identity escrow, 1997. Personal communication.
- [83] D. E. Knuth. *Fundamental algorithms*, volume 1 of *The art of computer programming*. Addison-Wesley, 1968. Second edition, 1973.
- [84] D. E. Knuth. *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, 1981.

- [85] N. Koblitz. *A course in number theory and cryptography*, volume 114 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1987.
- [86] D. Kozen and S. Zaks. Optimal bounds for the change-making problem. *Theoretical Computer Science*, 123:377–388, 1994.
- [87] L. Lamport. Constructing digital signatures from one-way functions. *SRI intl. CSL-98*, October 1979.
- [88] L. Law, S. Sabett, and J. Solinas. How to make a mint: the cryptography of anonymous electronic cash. No. 96-10-17, National Security Agency, Office of Information Security Research and Technology, Cryptology Division, June 18 1996. For a copy e-mail to 21stCen@ffhsj.com or call at (202) 639-7200. See also the 21st Century Banking Alert page at URL: <http://www.ffhsj.com/bancmail/bancpage.html>.
- [89] S. Low, N. Maxemchuk, and S. Paul. Anonymous credit cards. In *Proceedings of the 2nd. ACM conference on Computer and Communication Security*, Fairfax, Virginia, USA, November 2-4 1994.
- [90] S. Martello and P. Toth. *Algorithms and computer implementations*. J. Wiley and Sons, 1990. Wiley-Interscience Series in Discrete Mathematics.
- [91] G. Medvinsky and B. C. Neuman. Netcash: A design for practical electronic currency on the internet. In *Proceedings of the First annual ACM conference on Computer and Communications Security*, November 1993.
- [92] S. Micali. Fair public-key cryptosystems. In E. F. Brickell, editor, *Advances in Cryptology — Crypto '92, Proceedings (Lecture Notes in Computer Science 740)*, pages 113–138. Springer-Verlag, 1993. Santa Barbara, California, U.S.A., August 16–20.
- [93] S. Micali, C. Rackoff, and B. Sloan. The notion of security for probabilistic cryptosystems. *SIAM Journal of Computing*, 17:412–426, 1988.
- [94] D. Naccache and D. M'Raihi. Cryptographic smart cards. *IEEE Micro*, 16(3):14–23, June 1996.

- [95] M. Naor. Bit commitment using pseudo-randomness. In G. Brassard, editor, *Advances in Cryptology — Crypto '89, Proceedings (Lecture Notes in Computer Science 435)*, pages 481–496. Springer-Verlag, 1990. Santa Barbara, California, U.S.A., August 20–24.
- [96] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the twenty first annual ACM Symp. Theory of Computing, STOC*, pages 33–43, May 15–17, 1989.
- [97] NSA NIST. Proposed federal information processing standard for secure hash standard. *Federal Register*, 57(21):3747–3749, Jan 31 1992.
- [98] I. Niven, H. S. Zuckerman, and H. L. Montgomery. *The theory of numbers*. John Wiley & Sons, New York, fifth edition, 1991.
- [99] A. M. Odlyzko. Discrete logs in a finite field and their cryptographic significance. In N. Cot T. Beth and I. Ingemarsson, editors, *Advances in Cryptology, Proc. of Eurocrypt 84 (Lecture Notes in Computer Science 209)*, pages 224–314. Springer-Verlag, 1984. Paris, France April 1984.
- [100] T. Okamoto. An efficient divisible electronic cash scheme. In Don Coppersmith, editor, *Advances in Cryptology, Proc. of Crypto '95 (Lecture Notes in Computer Science 963)*, pages 438–451. Springer-Verlag, 1995. Santa Barbara, California, U.S.A., August 27–31.
- [101] T. Okamoto, 1996. Personal communication.
- [102] T. Okamoto and K. Ohta. Disposable zero-knowledge authentication and their applications to untraceable electronic cash. In *Advances in Cryptology, Proceedings of Crypto '89 (Lecture Notes in Computer Science 435)*, pages 481–496. Springer-Verlag, 1990. Santa Barbara, California, U.S.A., August 20–24.
- [103] T. Okamoto and K. Ohta. Universal electronic cash. In *Advances in Cryptology — Crypto '91 (Lecture Notes in Computer Science)*, pages 324–337. Springer-Verlag, 1992.
- [104] J. C. Pailles. New protocols for electronic money. In *Proceedings of Ausicrypt '92*, pages 263–274, 1993.

- [105] B. Pfitzmann and M. Waidner. How to break and repair a ‘provably secure’ untraceable payment system. In J. Feigenbaum, editor, *Advances in Cryptology, Proc. of Crypto '91 (Lecture Notes in Computer Science 576)*, pages 338–350. Springer-Verlag, 1992.
- [106] D. Pointcheval and J. Stern. Provably secure blind signature schemes. In *Advances in Cryptology, Proc. of Asiacrypt '96 (Lecture Notes in Computer Science)*, Kyongju, South Korea, November 3–7 1996. Springer-Verlag. To appear. Available at http://www.ens.fr/dmi/equipes_dmi/grecc/pointche/pub.html.
- [107] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. Maurer, editor, *Advances in Cryptology, Proc. of Eurocrypt '96*, pages 387–398, Zaragoza, Spain, May 11–16, 1996. Springer-Verlag. Available at http://www.ens.fr/dmi/equipes_dmi/grecc/pointche/pub.html.
- [108] C. Pomerance. *Analysis and comparison of some integer factorization algorithms*, pages 89–139. Math. Center Amsterdam, 1982.
- [109] C. Pomerance. The quadratic sieve factoring algorithm. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology. Proc. of Eurocrypt 84 (Lecture Notes in Computer Science 209)*, pages 169–182. Springer-Verlag, Berlin, 1985. Paris, France, April 9–11, 1984.
- [110] C. Pomerance, J. W. Smith, and R. Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM J. Comput.*, 17(2):387–403, 1988.
- [111] M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology Technical Report MIT/LCS/TR–212, Cambridge, Massachusetts, January 1977.
- [112] M. O. Rabin. Digitized signatures. In R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, editors, *Foundations of Secure Computation*, pages 155–168, New York, 1978. Academic Press.
- [113] Research and Development in Advanced Communication Technologies in Europe. RIPE integrity primitives: final report of RACE integrity primitives evaluation. Technical Report R1040, RACE, 1992.

- [114] R. L. Rivest. The MD5 digest algorithm, April 1992. In Internet RFC 1321. Also available at <http://theory.lcs.mit.edu/~rivest/publications.html>.
- [115] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Commun. ACM*, 21:294–299, April 1978.
- [116] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.
- [117] E. S. Selmer. On the postage stamp problem with 3 denominations. *Mathematica Scandinavica*, 56(2):105–116, 1985.
- [118] E. S. Selmer. Associate bases in the postage stamp problem. *Journal of Number Theory*, 42(3):320–336, 1992.
- [119] C. E. Shannon. Communication theory of secrecy systems. *Bell System Techn. Jour.*, 28:656–715, October 1949.
- [120] D. Simon. Anonymous communication and anonymous cash. In Neal Koblitz, editor, *Advances in Cryptology, Proc. of Crypto '96 (Lecture Notes in Computer Science 1109)*, pages 61–73, Santa Barbara, California, August 1996. Springer-Verlag.
- [121] M. Stadler, September 1996. Personal communication.
- [122] M. Stadler. Publicly verifiable secret sharing. In *Advances in Cryptology, Proc. of Eurocrypt '96*, pages 190–199. Springer-Verlag, 1996.
- [123] M. Stadler, J. M. Piveteau, and J. Camenisch. Fair blind signatures. In *Advances in Cryptology, Proc. of Eurocrypt '95*, pages 209–219. Springer-Verlag, 1995.
- [124] Utah State. Digital signature act. Utah code ann. tit. 46, ch. 3, 1995. Amended in 1996. Digital Signature Act Amendments, 52nd Leg., Gen. Sess., 1996 Utah Laws 188 (to be codified at Utah Code Ann. tit. 46, ch. 3). History and Current Status are available online at <http://www.state.ut.us/ccjj/digsig/dsut-int.htm>.
- [125] Y. Tsiounis and M. Yung. The semantic security of El Gamal encryption is equivalent to the decision Diffie-Hellman. Technical report, GTE Laboratories Inc., May 1997.

- [126] H. van Antwerpen. Electronic cash. Master's thesis, CWI, 1990.
- [127] J.H. van Lint and R.M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.
- [128] R. Verma and J. Xu. On optimal greedy change making. Technical Report UH-CS-94-03, Department of Computer Science, University of Houston, April 1994.
- [129] B. von Solms and D. Naccache. On blind signatures and perfect crimes. *Computers and Security*, 11(6):581–583, October 1992.
- [130] Y. Yacobi. Efficient electronic money. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology, Proc. of Asiacrypt '94 (Lecture Notes in Computer Science 917)*, pages 153–163. Springer-Verlag, 1995. Wollongong, Australia, Nov. 28–Dec. 1.
- [131] A. C. Yao. Protocols for secure computations. In *23rd Annual Symp. on Foundations of Computer Science (FOCS)*, pages 160–164. IEEE Computer Society Press, 1982.