

A Fair Payment System with Online Anonymous Transfer

by Binh D. Vo
B.S. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2005

SUBMITTED TO
THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY
FEBRUARY 2007

©2006 Binh D. Vo. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
document in whole and in part in any medium now known or hereafter created.

Signature of Author

Department of EECS
October 26, 2006

Certified by

Ronald Rivest, Viterbi Professor of EECS
Thesis Supervisor

Certified by

Susan Hohenberger, Department of EECS
Thesis Co-Supervisor

Accepted by

Arthur C. Smith, Professor of EECS,
Chairman, Department Committee on Graduate Theses

A Fair Payment System with Online Anonymous Transfer
by
Binh D. Vo
Submitted to the
Department of Electrical Engineering and Computer Science
on October 26, 2006 In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Physical cash can be anonymously transferred. Transferability is a desirable property because it allows for flexible, private commerce where neither the seller nor the buyer must identify themselves to the bank. In some cases, however, anonymity can be abused and lead to problems such as blackmail and money laundering. In 1996, Camenisch, Piveteau, and Stadler introduced the concept of *fairness* for (non-transferable) ECash, where a trusted authority can revoke the anonymity of certain transactions as needed.

To our knowledge, no current ECash system supports both anonymous transfer and fairness. We have designed and implemented such a system. Also, we formally describe a set of desirable properties for ECash systems and prove that our system meets all of these properties under the Strong RSA assumption and the Decisional Diffie-Hellman assumption in the random oracle model. Furthermore, we provide extensions for our system that could allow it to deal with offline payments and micropayments.

Our system has been implemented in java. Tests have shown that it performs and scales well, as expected.

Thesis Supervisor: Ronald Rivest
Title: Viterbi Professor of EECS

Thesis Co-Supervisor: Susan Hohenberger
Department: Electrical Engineering and Computer Science

1 Introduction

Electronic payment systems, also known as ECash systems, were introduced by Chaum [4]. These systems aim to allow users to engage in digital payment, while providing anonymity. Unlike other digital payment systems, such as credit cards, ECash allows a person to keep their personal commerce private – from how much he spends on groceries to where he buys his gas. For more details on ECash systems, refer to the NSA’s survey of payment systems [9]. Unconditional anonymity, however, also has drawbacks. It is sometimes desirable that law enforcement agencies be able to revoke the anonymity of certain transactions when blackmail, money laundering, or other illegal activities are suspected. To handle this, Camenisch et al. [8] proposed *fair* ECash, where a trusted authority can revoke the anonymity of transactions as needed, and introduced a simple payment system with this property. In fair ECash systems, payments are anonymous under normal circumstances; however, when a bank observes suspicious activity, it can ask for the help of a trusted authority, colloquially called the judge, to uncover the cash owner. As a matter of checks and balances, it should be noted that neither the judge nor the bank can trace any transaction on their own.

The notion of *transferability* is fundamental to normal cash. That is, a recipient of a coin can reuse that coin to pay someone else. Transferability in an ECash system also increases flexibility in its use. For example, “coin shops” can be opened to obtain electronic coins and sell them to users without direct interaction with the bank. In addition to making the system easier to use, this limits the amount of information mining the bank can perform about users’ spending (or receiving) habits, improving privacy. Or, electronic coins could be printed onto normal paper and used as cash at properly equipped real stores. This also greatly improves ease of use in the system, for example the aforementioned “coin shops” could simply be local software stores selling ECash cards that can be scanned into users’ computers at home. However, as with normal cash, the increased flexibility afforded by transferability makes illegal activities like money laundering easier. Thus, in a transferable payment system, fairness is even more important.

ECash systems are conventionally divided into those that are *online* and those that are *offline*. In an online system, the bank must be contacted for each transaction, although the anonymity of at least the payer remains protected (our system will also protect the anonymity of the payee). In an offline system, a transaction can occur without contacting the bank. Offline systems offer a greater degree of flexibility, while online systems better protect the bank by instantly detecting double spending of coins. The only online anonymous transfer system known to us was first proposed by Burk and Pfitzmann [3], and later rediscovered by Simon [18]. Unfortunately, neither the Burk-Pfitzmann system nor the Simon systems provide any form of traceability, i.e., they are not fair payment schemes.

We have designed a system, which we will describe below, that provides both fairness and anonymous transfer in the online model by extending the Burk-Pfitzmann system. To achieve this, we make use of the *group signature protocol*, which was introduced by Chaum [5] to allow any one of a group of authorities to sign a value in a way that allows anyone to verify a signature under the group’s public key without revealing the identity of the authority who signed it. Further, the group owner can uncover the identity as needed.

We also show how to extend our system to work in the offline model, and also to efficiently handle large volumes of micropayments. We begin by fully articulating a formal set of desirable properties for any electronic payment system, so as to have clearly defined goals for our extensions. These properties are designed to describe both online and offline payment systems.

1.1 Contributions

In this paper, we describe a number of contributions. First, we introduce and discuss the *SAFT* properties, a set of desirable properties we have constructed to evaluate existing payment systems. These properties

include definitions of different levels of security and anonymity, two properties that describe the degree to which money and identity are protected by the system. They also define the traits of transferability and fairness, which describe abilities that are useful for a system to provide. With this framework, we can more cleanly compare and contrast existing payment systems.

This framework also serves to outline the goals the SAF_cT system, a new payment system we have introduced that aims to completely satisfy all of the $SAFT$ properties. We have designed and implemented this system, as well as proven that it meets each of the $SAFT$ properties. We have also proposed two extensions to this system which are not implemented: a probabilistic payment protocol that allows the system to be used for large volumes of micropayments while alleviating computational and network load for the bank, and an offline payment protocol that allows a number of transactions to occur in sequence before the bank must be contacted.

1.2 Roadmap

In section 2, we will layout and discuss the $SAFT$ properties, which serve as a basis to evaluate other payment systems described in the paper and as a set of goals to achieve in our own payment system.

In section 3, we will discuss and evaluate several existing payment systems, to give an idea of the current states of payment systems, and to show how the $SAFT$ properties work. Our system will attempt to improve upon these systems by satisfying all of the $SAFT$ properties.

In section 4, we will discuss some basic signature protocols we will need to make use of in our system and its extensions. We will describe what types of signatures we need and how they are used, and then we will give an example of each type of signature that we will use in our system.

In section 5, we will describe our SAF_cT payment system, which is the first payment system to satisfy all of the $SAFT$ properties. We'll begin by showing the protocols which comprise the system, then formally prove how our system meets all of the properties.

In section 6, we describe possible extensions to our system. These extensions take form as additional protocols to the system, and aim to allow payments to be made offline, or probabilistically with expected value in increments less than the minimum coin size of the system.

Finally, in section 7, we will describe our implementation of the SAF_cT system in Java and show timing results for all of the protocols and resulting coin sizes.

2 The $SAFT$ properties in payment systems

This section defines a set of desirable properties for any ECash system P . First, we need to establish some notation. Let P have security parameter k . For every discrete time period t , there are a set of users U_t and a set of coins C_t . For every coin c and time t , let $O_{(c,t)}$ be the owner of c at t . For every user u and time t , let $C_{(u,t)}$ be the set of coins owned by u at t . The system P may have the following entities:

- *Bank*: The bank is responsible for keeping track of which coins are owned by which users in the system. Users exchange physical cash for electronic coins from the bank.
- *User*: A user is any person participating in the payment system as either a buyer or a seller. User may obtain electronic coins from the bank, trade the coins amongst themselves, and deposit the coins with the bank.
- *Judge*: The judge is a trusted authority who remains passive until invoked by the bank to help revoke the anonymity of a suspicious transaction.

The system P is also comprised of the following protocols, to be discussed in detail in section 5, *Setup*, *Register*, *Purchase*, *Transfer*, *Deposit*, and *Trace*.

2.1 Properties

Below are the *SAFT* (pronounced ‘safety’) properties that an eCash system may possess:

- *Security*: A requirement of any payment system is to protect coin value. We define two levels of security: a strongly secure system protects coin value absolutely, while a weakly secure system detects entities that adversely affect coin value so they may be appropriately penalized.
 1. *Weak*: Except with probability negligible in k ,
 - (a) No party or parties can generate a new coin or increase a current coin’s value without identifying themselves.
 - (b) No party or parties except the owner of a coin, can transfer, destroy, or decrease a coin’s value without identifying themselves.
 2. *Strong*: Except with probability negligible in k ,
 - (a) No party or parties, except the bank, can generate a new coin or increase a current coin’s value.
 - (b) No party or parties, except the owner of a coin, can transfer, destroy, or decrease a coin’s value.

The first point of either level of security guarantees that no entity can manipulate the system for profit, and the second point guarantees that no entity can manipulate the system to harm another. We shall use s to denote a weakly secure system and S to denote a strongly secure system.

- *Anonymity*: We provide definitions of anonymity in terms of two stage adversarial games, against an adversary A who knows the public parameters of the payment system and all information held by some subset U of corrupted users and the bank. Under these conditions, we describe three types of anonymity, weak anonymity which describes a system that protects privacy but *allows* linking of payments originating from the same user, strong anonymity which describes a system that protects against such linking, but allows linking sequential payments with the same coin, and total anonymity which prevents any kind of linking of payments.
 1. *Weak*: In the first stage, A requests a series of actions to be performed by the other entities *without knowing the results* (e.g., A may request that u_1 transfer a coin to u_2 without seeing any information that would be held by users not in the corrupted set U). The system is now in some arbitrary, but adversarially-chosen state. Now, A must select users u_0 and u_1 not in U such that each own a non-zero number of coins. Let $b \in \{0, 1\}$ be randomly selected by the experiment. In the second stage, u_0 chooses a coin c and transfers it to u_b . Now, u_b and A run the *transfer* protocol, where u_b attempts to transfer c to A . Finally, A must guess the identity of the user paying him, by outputting $b' \in \{1, 2\}$. The system is *weakly anonymous* if $b = b'$ with probability $\leq 1/2 + \text{negl}(k)$.
 2. *Strong*: In the first stage, A requests a series of actions to be performed by the other entities *and is shown the results*. The system is now in some arbitrary, but adversarially-chosen state. Now, A must select a user u_0 not in U that owns a non-zero number of coins, and users u_1 and u_2 not

in U . Let $b \in \{1, 2\}$ be randomly selected by the experiment. In the second stage, u_0 shows a coin c to A and transfers it to u_b . Now, u_b and A run the *transfer* protocol, where u_b attempts to transfer c to A . Finally, A must guess the identity of the user paying him, by outputting $b' \in \{1, 2\}$. The system is *strongly anonymous* if $b = b'$ with probability $\leq 1/2 + \text{negl}(k)$.

3. *Total*: In the first stage, A requests a series of actions to be performed by the other entities and is shown the results. The system is now in some arbitrary, but adversarially-chosen state. Now, A must select users u_1 and u_2 not in U such that each own a non-zero number of coins. Let $b \in \{0, 1\}$ be randomly selected by the experiment. In the second stage, u_b and A run the *transfer* protocol, where u_b attempts to transfer a coin c to A . Finally, A must guess the identity of the user paying him, by outputting $b' \in \{0, 1\}$. The system is *totally anonymous* if $b = b'$ with probability $\leq 1/2 + \text{negl}(k)$.

We shall denote a *weakly anonymous* system with a , a *strongly anonymous* one with A , and a *totally anonymous* one with A^* .

Note that only a transferable system can be strongly anonymous, since the definition of strong anonymity contains a coin that is transferred twice. A non-transferable system is either weakly anonymous or totally anonymous. Strong anonymity is a type of anonymity which provides protection for users against linking of their payments, but still allows coins to have a traceable history of expenditures, similar to the fashion in which real cash is stamped with serial numbers. See below for further discussion.

- *Fairness*: Fully anonymous systems can be abused for blackmail and money laundering purposes. To prevent this, anonymity breaking powers can be granted to a collaboration of the bank and a trusted authority known as the judge. There are two types of anonymity breaking:

F_c : Given any electronic coin c , the bank and judge can cooperate to identify its owner without learning any information about the ownership of other coins.

F_u : Given any user u , the bank and judge can cooperate to identify the set of coins owned by u without learning any information about the ownership of other coins (aside from the knowledge that they are not owned by u).

We note that no weakly anonymous system can possess F_c because if it is possible to guess that two coins c_1 and c_2 have the same owner with better than random chance, then information about one of them can be inferred from the other.

- *Transferability*: A system is transferable if the recipient of a coin can use the same coin to pay another person without having to deposit and withdraw it at the bank. We use T to denote a system with transferability.

Thus, we can classify any payment system by analyzing to see what subset of the *SAFT* properties that it possesses. We shall see next that currently known payment systems do not always possess all of the *SAFT* properties.

2.2 Strong anonymity vs total anonymity

As was mentioned earlier, strong anonymity is a type of anonymity that applies only to transferable systems. In a transferable system, there are potentially two ways in which payments could be linked. Either an adversary could identify two payments which originated from the same spender, or he could find two payments

that were both sequential transfers of the same coin. Strong anonymity prevents the first type of linking, but not the second, whereas total anonymity prevents any kind of linking of payments.

This type of linking allowed by strong anonymity is similar in fashion to real cash having serial numbers. Because real cash is stamped with these numbers, if a user paid a shop with a bill that had been given to him by that same shop, or by some entity collaborating with that shop, then the shop could identify him. Thus, strong anonymity only protects a user so long as the entity he receives a coin from and the entity he sends it to are not collaborating. However, this can be seen as only a minor weakness, one that the user could easily work around by transferring each coin he receives to himself before spending it elsewhere if he so wished. Note that in this way, strong anonymity does improve slightly upon the serial numbers of real cash, since a real person could not mask the serial numbers of his cash by giving it to himself. This method allows a user to protect himself against two colluding other users, but not against authorities asking the bank to reveal the history of a coin's transfers.

It can be argued that strong anonymity is actually desirable in some ways over total anonymity, especially in fair payment systems aiming to prevent blackmail and money laundering. It allows suspicious funds to be tagged and monitored, and gives a basis for tracing coins in a fair system. For example, if a kidnapper demanded payment in blackmail case, coins given to him could still be linked back to the victim forced to give payment. This would enable the judge to choose appropriate coins to trace to identify the criminal. Thus, this kind of linkability helps authorities decide which coins should be traced. This is functionally similar to the manner in which authorities might record the serial numbers of real cash to eventually track down criminals it was paid to.

3 Related Work

This section presents four well-known ECash schemes and discusses how they satisfy the SAFT properties.

3.1 Chaum's blind signature ECash system

Protocol Payment

1. To obtain a coin, U chooses a random number N and blinding factor M .
 2. U sends $[N]_M$ along with payment to B .
 3. B accepts payment and signs $[N]_M$ with sk_B .
 4. U unblinds this signature with M to obtain $\{N\}_{sk_B}$.
 5. U pays merchant V with $\{N\}_{sk_B}$.
 6. V verifies this signature with pk_B before accepting payment.
 7. V deposits $\{N\}_{sk_B}$ with B , who also verifies with pk_B before returning cash.
-

Figure 1: U paying V anonymously through Chaum's blind signature ECash system

Figure 1 shows Chaum's ECash system implemented based on the RSA assumption. Here, all coins have the same fixed denomination and cannot be destroyed. Since only the bank, as owner of sk_B , can generate coins, and only the possessor of $\{N\}_{sk_B}$ can transfer it, this system is secure. Also, by the nature of blind signatures, no entity can identify the owner of a coin or trace a transaction. Furthermore, no two transactions can be linked in any way. This system is thus totally anonymous. However, the system does

not provide transferability of ECash as each payment must begin with a withdrawal by the spender, and end with a deposit by the receiver. In addition, this scheme does not provide fairness since there is no way to trace the spender. So Chaum's ECash system is an SA^* scheme.

3.2 Burk-Pfitzmann anonymous transfer system

Protocol PurchaseCoin

1. U generates random public/private key pair pk_C and sk_C .
2. U sends pk_C to B with payment.
3. B adds pk_C to a public list of valid coins.
4. U confirms that pk_C is now on the list.

Protocol TransferCoin

1. V generates random public/private key pair pk_{C_V} and sk_{C_V} .
2. V sends pk_{C_V} to U .
3. U selects pk_{C_U} and sk_{C_U} from amongst his coins.
4. U sends $\{pk_{C_U}, pk_{C_V}\}_{sk_{C_U}}$ to B via mixnet.
5. B confirms that pk_{C_U} is on the list of valid coins and uses it to verify the signature.
6. B removes pk_{C_U} from the list and adds pk_{C_V} .
7. V confirms that pk_{C_V} is on the list.

Protocol DepositCoin

1. V selects pk_{C_V} and sk_{C_V} from amongst his coins.
 2. V sends $\{pk_{C_V}\}_{sk_{C_V}}$ to B .
 3. B confirms that pk_{C_V} is on the list of valid coins and uses it to verify the signature.
 4. B removes pk_{C_V} from the list and sends payment to V .
-

Figure 2: Burk-Pfitzmann online anonymous transfer ECash system

Figure 2 shows the Burk-Pfitzmann ECash protocol, which can be built upon any signature system, for example, the RSA signature. In this scheme, only the bank can generate coins in the *PurchaseCoin* protocol. In addition, only a coin's owner can transfer the coin to another via *TransferCoin* or redeemed it to real money via *DepositCoin*. Since coin cannot be destroyed or modified, this system is secure. Also, since no identities are stored by any entity, coins cannot be linked to owners. Separate coins also share no information, so the bank cannot link them. This ensures total anonymity. However, payments cannot be traced, therefore, the system does not support fairness. So the Burk-Pfitzmann system is an SA^*T scheme.

3.3 Camenisch, Piveteau, Stadler anonymous accounts system

Figure 3 shows a payment system [8] proposed by Camenisch et. al based on the discrete logarithm problem and the RSA assumption. In this system, each user has a unique personal account but can also generate multiple *anonymous accounts*. For each of these, the user can present a signature from the judge proving

Protocol OpenPersonalAccount

1. U generates public/private keys pk_{P_U} and sk_{P_U} .
2. U sends pk_{P_U} to B and identifies himself.
3. B stores pk_{P_U} as a personal account of U and returns $\{pk_{P_U}\}_{sk_B}$.

Protocol OpenAnonymousAccount

1. U generates public/private keys pk_{A_U} and sk_{A_U} .
2. U sends $\{pk_{P_U}\}_{sk_B}$ and pk_{A_U} to J and identifies himself.
3. J verifies $\{pk_{P_U}\}_{sk_B}$, stores pk_{A_U} and U , and returns $\{pk_{A_U}\}_{sk_J}$.
4. U sends $\{pk_{A_U}\}_{sk_J}$ anonymously to the bank.

Protocol DepositAnonymousAccount

1. U chooses a random number N and blinding factor M .
2. U sends $\{[N]_M\}_{sk_{P_U}}$ and pk_{P_U} to B .
3. B verifies, debits U , and signs $[N]_M$ with sk_B .
4. U unblinds this signature with M to obtain $\{N\}_{sk_B}$.
5. U sends $\{N\}_{sk_B}$ and pk_{A_U} anonymously to B .
6. B verifies this signature with pk_B and credits pk_{A_U} .

Protocol Pay

1. Vendor V identifies himself to user U and requests payment.
2. U chooses a random account A_U to pay vendor V .
3. U sends $\{pk_{A_U}, V\}_{sk_{A_U}}$ to B .
4. B debits A_U and credits V , recording this transaction.

Protocol Trace

1. B chooses a transaction it needs to trace, having originated from A_U .
2. B sends A_U to J .
3. J looks up U who owns A_U .

Figure 3: CPS anonymous accounts system

that the judge knows the identity of the account owner. Money is moved from the personal account to the anonymous account via blind signature payment, and spent from there. Basic signatures are used to ensure security of coins for the owner, and blind signatures are used to ensure security of coins for the bank. This system possesses property F_u because the bank and the judge can identify users of single anonymous account but no further. On the other hand, the bank is capable of linking all payments made from any single anonymous account allowing tracing that account to reveal all expenditures made from it. This means that the system is normally not strongly anonymous unless users create an anonymous account for each coin, an impractical solution. Thus, the CPS anonymous accounts system is a SaF_u system.

3.4 Camenisch, Piveteau, Stadler fair blind signature system

Camenisch et al. also proposes a fair payment system based on a concept of fair blind signatures they describe in [10], based on the strong RSA assumption. With fair blind signatures, a trusted third party can reverse the blinding of a number by cooperating with the signer. This has straightforward applications in fair payment systems. By using fair blind signatures in place of blind signatures in any payment system, the bank can cooperate with the judge to identify the sender of any payment so long as they store the numbers that are deposited and the blinded numbers that are withdrawn. This system is more convenient to use than Camenisch anonymous accounts system. However, it cannot list all of the coins owned by a particular user. This system is thus SA^*F_c .

4 Basic signature protocols

This section briefly recalls a set of protocols used as basic blocks in constructing various payment systems to be discussed later. As a notation for interactive protocols between entities U and V , we write $Protocol(U(x), V(y))$ where x is the input by U and y is the input by V . If the protocol returns some value z to U , we write that the protocol returns $U(z)$. We need a basic signature scheme that is unforgeable against adaptive chosen-message attacks [17]. This scheme will allow a user U to have a number N signed by some authority A .

Protocol BasicSignature

1. U sends N to A .
2. A signs N with sk_A to generate $\{N\}_{sk_A}$ and sends this value back to U .
3. Anyone with pk_A can verify that $\{N\}_{sk_A}$ was signed by an entity with access to sk_A .

Protocol $RSASetup(U(1^k))$

1. U generates two random primes p and q , $p \neq q$ and computes $n = pq$
2. U picks random e where $1 < e < (p-1)(q-1)$ and $\gcd(e, (p-1), (q-1)) = 1$
3. U computes d where $1 < d < (p-1)(q-1)$ and $ed = 1 \pmod{(p-1)(q-1)}$
4. $U((n, e), d)$

Protocol $RSASign(U(m, d))$

1. U computes $s = \text{hash}(m)^d \pmod n$
2. Return the signature $U(s)$.

Protocol $RSAVerify(V(s, (n, e)))$

1. Return $V(TRUE)$ if $\text{hash}(m) = s^e \pmod n$, otherwise return $V(FALSE)$.

Figure 4: Basic and RSA signature schemes

Figure 4 describes the structure of a basic signature scheme, and two implementations. The basic signature scheme allows a user U to prove to anyone that a number N was signed by some authority A with

knowledge of sk_A without revealing sk_A . The most widely used implementation of this scheme was proposed by Rivest, Shamir, and Adleman [15] which is GMR secure under the strong RSA assumption in the random oracle model. We use a hash function on the message before signing it to protect against malleability attacks [1].

We will also need a blind signature scheme. These types of schemes allow an authority to sign a value without knowing what it is. In this scheme, a user U can again have a number N signed by an authority A , without A ever seeing or able to compute N .

Protocol BlindSignature

1. U generates a *blinding factor* M .
 2. U blinds N with M to obtain $[N]_M$.
 3. U sends $[N]_M$ to A .
 4. A computes $\{[N]_M\}_{sk_A}$ and sends it to U .
 5. U unblinds $\{[N]_M\}_{sk_A}$ with M to obtain $\{N\}_{sk_A}$.
-

Protocol ChaumBlindSetup($A(1^k)$)

1. A chooses $n = pq$ where p and q are RSA primes.
2. A releases (n, e) as a public key, where e is a prime less than $(p - 1)(q - 1)$.
3. A keeps (p, q, d) as a secret key, where $d = e^{-1} \pmod{(p - 1)(q - 1)}$.
4. Return $A((n, e), (p, q, d))$

Protocol ChaumBlindSign($U(N), A(sk = (p, q, d))$)

1. U picks a blinding factor $b \pmod n$ and uses it to blind N as $m_B = b^e \text{hash}(N) \pmod n$ and sends it to A .
2. A computes $v_B = m_B^d \pmod n$ and returns it to U .
3. Return the signature $U(v = v_b b^{-1} \pmod n)$.

Protocol ChaumBlindVerify($V(N, v, pk = (n, e))$)

1. Return $V(TRUE)$ if $v^e = \text{hash}(N) \pmod n$, otherwise return $V(FALSE)$.
-

Figure 5: Basic and Chaum’s blind signature

Figure 5 shows the structure of a blind signature protocol, and one implementation by Chaum [4]. In addition to signing, these protocols enable the authority A to sign a number N without ever knowing its value. The top part of the figure shows the basic blind signature scheme. In our system, we will use the scheme proposed by Chaum, which is secure under the strong RSA assumption.

However, thus far, this scheme is only proven to be secure against forgeries up until an adversary has obtained some number l of signatures, where l is poly-logarithmically bounded on the security parameter. Pointcheval provides a method to extend existing blind signatures to provably protect against forgery attacks where l is bounded polynomially. However, it has the further requirement that there exist a neutral third party to moderate the transaction between the authority and the user [13]. For payment systems, this can be awkward. Thus, for our purposes, we assume that the bank will use Chaum’s system and simply change

its public and private keys after having given out a set number of signatures so as to protect itself against forgeries.

Finally, we will need a group signature scheme that allows any member of a group of users to generate a signature for a value, and have their identity protected until opened by a group owner. This scheme includes a group owner G and n group members $\{G_1 \dots G_n\}$.

Figure 6 shows the group signature protocol, first proposed by Chaum et al. [5], to allow a group of authorities $G = \{G_1 \dots G_n\}$, of size n , to produce signatures for the same public key. A single public key pk_G can be used to verify that some secret key sk_{G_i} was used to sign the number N , but cannot be used to identify which. A secret key sk_G can be used to identify the key that was used. In our system, we use the group signature protocol by Ateniese, Camenisch, Joye, and Tsudik [6]. This protocol is shown in the lower part of the figure. It is secure under the Strong RSA assumption and the Decisional Diffie-Hellman assumption in the random oracle model. The assumed security parameters for this protocol are: $\epsilon > 1$, the tightness of the group signature, k , the size of the output of a hash function $hash(x)$, and l_p , the size of the group signature modulus.

5 The SAF_cT payment system

In this section, we show how to use group signatures to extend the Burk-Pfitzmann system to provide both online anonymous transfer and fairness. We will make use of RSA signatures and the group signature scheme outlined in section 4, so our system is based on the strong RSA assumption. Users are required to identify themselves to the judge, before sending and receiving anonymous cash. The judge uses a group signature scheme and grants a group secret key to each identified user, keeping the master key for himself. The judge is responsible for keeping the identity of each user in relation to his secret key. Users use their group secret keys to sign transactions with the bank. When a single transaction needs to be traced, the bank can show the record of this transaction to the judge, who then uses his master key to identify both the sender and receiver. Note that no communication is necessary with the judge except when registering a new account and tracing transactions. In addition, single transactions can be traced independently from others.

5.1 The basic SAF_cT protocols

Figure 7 shows the protocols in the SAF_cT system. We assume that coins are of fixed value although an extension for multiple coin values is straightforward. Henceforth, we will refer to two users U and V , the bank B , and the judge J . The judge J keeps a group signature secret key sk_J , and a publically available public key pk_J . The individual group signatures will be given out to users to keep, so U holds sk_U , which is a group secret signature in J 's group. The bank B maintains a list of valid coins, and a history of all coin transfers.

Note that such a history can become cumbersome over time, but is necessary so the bank can prove that users no longer own signed coins. Alternatively, coins could be made to include a signed expiration date, so that the bank would not be forced to keep a history of all coins ever made. This would not overly restrict users, since they could easily transfer coins to themselves to renew the expiration dates.

Below are the protocols:

- *Setup*: The bank must setup a pair of keys for both RSA signatures used to verify coin transfers, and Chaum blind signatures used to verify coin withdrawals. The judge sets up a group for ACJT group signatures. Recall from section 4 that the Chaum blind signatures scheme only guarantees security if it is used to generate a number of signatures bounded polylogarithmically in the security parameter.

Protocol GroupSignature

1. G_i signs N by computing $\{N\}_{sk_{G_i}}$ and sends it to U .
 2. U can now have $\{N\}_{sk_{G_i}}$ validated by pk_G .
 3. sk_G and $\{N\}_{sk_{G_i}}$ are sufficient to compute G_i .
-

Protocol ACJTGroupSetup($G(1^k, \epsilon, 1^{l_p})$)

1. G chooses lengths $\lambda_1 > \epsilon(\lambda_2 + k) + 2$, $\lambda_2 > 4l_p$, $\gamma_1 > \epsilon(\gamma_2 + k)$, and $\gamma_2 > \lambda_1 + 2$
2. Let $V = (2^{\lambda_1} - 2^{\lambda_2}, 2^{\lambda_1} + 2^{\lambda_2})$ and $T = (2^{\gamma_1} - 2^{\gamma_2}, 2^{\gamma_1} + 2^{\gamma_2})$.
3. G picks l_p -bit primes p', q' such that $p = 2p' + 1$ and $q = 2q' + 1$ are prime. Let $n = pq$.
4. G chooses random elements a, a_0, g, h from the group of quadratic residues modulo n .
5. G chooses x from the group of quadratic residues modulo $p'q'$.
6. Return $G(pk, sk)$ such that $pk = (n, a, a_0, g, h, y = g^x \pmod n)$ and $sk = (p', q', x)$.

Protocol ACJTGroupJoin($U(), G(pk = (n, a, a_0, g, h, y = g^x \pmod n))$)

1. U chooses random x_U from V , and computes $y_U = a^{x_U} \pmod n$, where y_U is a quadratic residue of n .
2. G computes (A_U, e_U) where e_U is a random prime from T and $A_U = (y_U a_0)^{1/e_U} \pmod n$.
3. Return $U(x_U, A_U), G(e_U)$

Protocol ACJTGroupSign($U(N, (x_U, A_U))$)

1. U picks random w of $2l_p$ bits and computes:
 $T_1 = A_U y^w \pmod n$, $T_2 = g^w \pmod n$, and $T_3 = g^{e_U} h^w \pmod n$.
2. U picks random numbers:
 r_1 with $\epsilon(\gamma_2 + k)$ bits, r_2 with $\epsilon(\lambda_2 + k)$ bits,
 r_3 with $\epsilon(\gamma_1 + 2l_p + k + 1)$ bits, and r_4 with $\epsilon(2l_p + k)$ bits
3. U computes:
 $d_1 = T_1^{r_1} / (a^{r_2} y^{r_3}) \pmod n$, $d_2 = T_2^{r_1} / g^{r_3} \pmod n$,
 $d_3 = g^{r_4} \pmod n$, $d_4 = g^{r_1} h^{r_4} \pmod n$, and
 $c = \text{hash}(g||h||y||a_0||a||T_1||T_2||T_3||d_1||d_2||d_3||d_4||N)$
4. U computes $s_1 = r_1 - c(e_U = 2^{\gamma_1})$, $s_2 = r_2 - c(x_U = 2^{\lambda_1})$, $s_3 = r_3 - ce_U w$, and $s_4 = r_4 - cw$
5. Return $U(c, s_1, s_2, s_3, s_4, T_1, T_2, T_3)$

Protocol ACJTGroupVerify($V((c, s_1, s_2, s_3, s_4, T_1, T_2, T_3), pk = (n, a, a_0, g, h, y = g^x \pmod n))$)

1. V computes $v_1 = a_0^c T_1^{s_1 - c2^{\lambda_1}} / (a^{s_2 - c2^{\lambda_1}} y^{s_3}) \pmod n$, $v_2 = T_2^{s_1 - c2^{\lambda_1}} / g^{s_3} \pmod n$, $v_3 = T_2^c g^{s_4} \pmod n$, $v_4 = T_3^c g^{s_1 - c2^{\lambda_1}} h^{s_4} \pmod n$, and $c' = \text{hash}(g||h||y||a_0||a||T_1||T_2||T_3||v_1||v_2||v_3||v_4||N)$
2. Return $V(TRUE)$ if $c = c'$, otherwise return $V(FALSE)$.

Protocol ACJTGroupOpen($G((c, s_1, s_2, s_3, s_4, T_1, T_2, T_3), sk = (p', q', x))$)

1. Return $G(A_U = T_1 / T_2^x \pmod n)$.
-

Figure 6: ACJT group signature

Protocol Setup:

1. The judge J computes $(pk_J, sk_J) = ACJTGroupSetup(1^k, \epsilon, 1^l)$
2. The bank B computes $(pk_B, sk_B) = RSASetup(1^k)$
3. B computes $(pk_B^b, sk_B^b) = ChaumBlindSetup(1^k)$

Protocol Register:

1. User U and J compute $sk_U = ACJTGroupJoin(U(), J(sk_J))$

Protocol Purchase:

1. User U computes $(pk_C, sk_C) = RSASetup(1^k)$
2. U pays B and they compute $\sigma_{sk_B^b}(pk_C) = ChaumBlindSign(U(pk_C), B(sk_B^b))$
3. U computes $\sigma_{sk_U}(pk_C) = ACJTGroupSign(pk_C, sk_U)$
4. U sends $\sigma_{sk_B^b}(pk_C)$ and $\sigma_{sk_U}(pk_C)$ to B anonymously
5. B checks that $ChaumBlindVerify(\sigma_{sk_B^b}(pk_C), pk_B^b) = true$
6. B checks that $ACJTGroupVerify(\sigma_{sk_U}(pk_C), pk_J) = true$
7. B now computes $\sigma_{sk_B}(pk_C) = RSASign(pk_C, sk_B)$ and returns it to U
8. U keeps $(\sigma_{sk_U}(pk_C), \sigma_{sk_B}(pk_C))$ as a coin.

Protocol Transfer:

1. User V computes $(pk_{C_V}, sk_{C_V}) = RSASetup(1^k)$
2. V computes $\sigma_{sk_V}(pk_{C_V}) = ACJTGroupSign(pk_{C_V}, sk_V)$ and sends it to U
3. U computes $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}) = RSASign((pk_{C_U}, pk_{C_V}), sk_{C_U})$
4. V sends $\sigma_{sk_{C_U}}((pk_{C_U}, pk_{C_V}), \sigma_{sk_V}(pk_{C_V}))$ to B
5. B checks that $RSASignVerify(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}), pk_{C_U}) = true$
6. B checks that $ACJTGroupVerify(\sigma_{sk_V}(pk_{C_V}), pk_J) = true$
7. B keeps $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V})$ as proof this coin has been relinquished
8. The bank now computes $\sigma_{sk_B}(pk_{C_V}) = RSASign(pk_{C_V}, sk_B)$ and returns it to V
9. V keeps $(\sigma_{sk_V}(pk_{C_V}), \sigma_{sk_B}(pk_{C_V}))$ as a coin.

Protocol Deposit:

1. User U chooses a coin pk_C and sends $\sigma_{sk_C}(pk_C) = RSASign(pk_C, sk_C)$ to the bank.
2. The bank checks that $RSASignVerify(\sigma_{sk_C}(pk_C), sk_C) = true$
3. B keeps $\sigma_{sk_C}(pk_C)$ as proof this coin has been relinquished and pays U

Protocol Trace:

1. B wishes to have coin $(\sigma_{sk_U}(pk_{C_U}), \sigma_{sk_B}(pk_{C_U}))$ traced to identify U , the owner of the coin
2. B sends $\sigma_{sk_U}(pk_C)$ to J .
3. J computes $U = ACJTGroupOpen(\sigma_{sk_U}(pk_C))$

Figure 7: The $SAF_C T$ payment system

Thus, for safe usage, the bank must choose a new blind signature secret key after using it to create this many signatures. For the sake of simplicity, we write the protocols assuming the bank is operating within this number of signatures.

- *Register:* To register with the system, a user U must join the group managed by J .
- *Purchase:* When user U wishes to purchase a coin from the bank B , U randomly generates a public/private key pair pk_C and sk_C . He holds sk_C as the coin representation while blinding pk_C and sending it to the bank with payment to get a blind signature. U anonymously sends this signature and his pk_C signed by his group private key back to B . B uses the group public key to verify that U is a registered user, verifies his own blind signature, then adds pk_C to the valid coin list. B also signs this coin and returns the signature to U . In cases of disputes, the signature verifies that U still owns the coin unless the bank can produce another signature by pk_C showing that this coin has been transferred or deposited,
- *Transfer:* When U pays V a coin, V first randomly generates a public/private key pair pk_{C_V} and sk_{C_V} and holds sk_{C_V} . He signs pk_{C_V} with his group secret key and transmits it to U . U chooses a coin C_U to pay V with. He signs and sends a transfer request identifying pk_{C_U} and pk_{C_V} , the sending and receiving coins. This, along with the signature by V 's secret group key are signed with sk_{C_U} and sent to B . B confirms that this request was authorized by a valid coin's owner by verifying this with pk_{C_U} and checking that pk_{C_U} is on the coin list. B also verifies that the recipient is a traceable user by verifying the signature of pk_{C_V} with the group public key. B then removes pk_{C_U} from the list and adds pk_{C_V} . B also stores a record of this transaction and returns to V a signed proof of coin creation for dispute resolution.
- *Deposit:* When V wishes to deposit a coin for cash, he sends both the coin's secret and public keys to the bank. B checks that this is a valid coin, then removes the coin from the list and credits V 's bank account.
- *Trace:* If B and J wish to trace a suspicious transfer from pk_{C_U} to pk_{C_V} , B reveals to J the record of this transfer, along with the stored signatures which were generated with the secret group signature keys held by U and V . Since J holds the group master key, he can identify the user whose key was used to generate the coin.

5.2 Analysis

We will now show that the SAF_cT satisfies the SAF_cT properties.

5.2.1 Assumptions

There are a number of well-known assumptions we will need to make in order to prove our system meets all of the SAF_cT properties. This need arises from the signature systems we used as building blocks.

- *Strong RSA Assumption:* Given integers N , e , and C s.t. N is the product of two primes, $2 < e < N$ is coprime to $\phi(N)$, and $0 < C < N$, it is intractible to find integer P s.t. $P^e = C \pmod N$. At the time of this writing, there is no known way to solve this problem more efficiently than to factor N , which is hard for large N .

- *Decisional Diffie-Hellman Assumption:* Within a cyclic group G of order q , for a randomly chosen generator g and random (a, b, c) such $0 \leq a < q$, $0 \leq b < q$, and $0 \leq c < q$, (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are computationally indistinguishable. This assumption is believed to be true as long as taking discrete logs in G is hard.

These assumptions are further made within the *Random Oracle Model*. This is a model where some random oracle (typically a hash function) behaves in the following way:

- If given a query x that it has not seen before, it returns a random response with a uniform probability being anywhere within the output space.
- If given a query x s.t. it has been given the query x before and returned y , it returns y .

5.2.2 Security

The SAF_cT payment system provides security by protecting the bank and user's value with RSA signatures in the same manner as in the Burk-Pfitzmann system.

Theorem 1 *Under the strong RSA assumption in the random oracle model, no group of entities without the bank can create a valid coin even if they have compromised all users and all keys except for that of the bank and may adaptively choose coins to observe.*

Proof: Assume for contradiction that an adversary A , given the judge's private key sk_J , the bank's public keys pk_B and pk_B^b , a set of all users' private keys $(sk_{U_1}, \dots, sk_{U_n})$, and the ability to act within the system as a complete set of corrupted users, can create a valid coin with the bank. We show that this enables construction of an adversary A' , that can break the security of RSA signatures with hashes, which are also GMR secure [17] under the strong RSA assumption in the random oracle model.

A' is constructed as follows:

First, A' will perfectly create a simulated world of the SAF_cT payment system. It runs the setup protocol of the SAF_cT system choosing keys for the bank pk_B, sk_B, pk_B^b , and sk_B^b . It will also run the setup protocol for some set of corrupted users $U_1 \dots U_n$.

A' will now engage in the first stage of a challenge session with A , in which A will request that a series of purchases, transfers, and deposits can occur between corrupted users.

- *Purchase:* If U_x is to purchase a coin pk_{C_x} , A' must produce $(\sigma_{sk_{U_x}}(pk_{C_x}), \sigma_{sk_B}(pk_{C_x}))$. Since A' knows sk_{U_x} , and can ask for any signatures by sk_B , it can do this.
- *Transfer:* If U_x is to transfer a coin pk_{C_x} into a coin pk_{C_y} owned by U_y , A' must produce $(\sigma_{sk_{U_y}}(pk_{C_y}), \sigma_{sk_B}(pk_{C_y}), \sigma_{sk_{C_x}}(pk_{C_x}, pk_{C_y}))$. Since A' knows (sk_{U_y}, sk_{C_x}) , and can ask for any signatures by sk_B , it can do this.
- *Deposit:* Trivial, since A' needs not produce any new signatures.

A must now produce a valid coin X , consisting of $(\sigma_{sk_U}(pk_X), \sigma_{sk_B}(pk_X))$. A' will then return $\sigma_{sk_B}(pk_X)$ as a valid signature by sk_B , thus breaking the security of RSA.

Theorem 2 *Under the strong RSA assumption in the random oracle model, no group of entities without the owner of a coin can cause that coin to be invalidated even if they have compromised all users and all keys except for that of the coin (even including that of the owner), and may adaptively choose other coins to observe.*

Proof: Ownership of a coin pk_C in the the SAF_cT system is defined to be given to the person who knows sk_C . Assume for contradiction that an adversary A , given the judge's private key sk_J , the bank's public and private keys (pk_B, sk_B) and (pk_B^b, sk_B^b) , a set of all users' private keys $(sk_{U_1}, \dots, sk_{U_n})$, and the ability to act within the system as a set of corrupted users, can invalidate a coin pk_C owned by U_h after adaptively viewing any coins except for this coin. We show that this enables construction of an adversary A' , that can break the security of RSA signatures with hashes, which are also GMR secure [17] under the strong RSA assumption in the random oracle model.

A' is constructed as follows:

First, A' will perfectly create a simulated world of the SAF_cT payment system. It runs the setup protocol of the SAF_cT system choosing keys for the bank pk_B, sk_B, pk_B^b , and sk_B^b . It will also run the setup protocol for some set of corrupted users $U_1 \dots U_n$, and one honest user U_h .

A' will now engage in the first stage of a challenge session with A , in which A will request that a series of purchases, transfers, and deposits can occur between corrupted users.

- *Purchase:* If U_x is to purchase a coin pk_{C_x} , A' must produce $(\sigma_{sk_{U_x}}(pk_{C_x}), \sigma_{sk_B}(pk_{C_x}))$. Since A' knows sk_{U_x} , and can ask for any signatures by sk_B , it can do this.
- *Transfer:* If U_x is to transfer a coin pk_{C_x} into a coin pk_{C_y} owned by U_y , A' must produce $(\sigma_{sk_{U_y}}(pk_{C_y}), \sigma_{sk_B}(pk_{C_y}), \sigma_{sk_{C_x}}(pk_{C_x}, pk_{C_y}))$. Since A' knows (sk_{U_y}, sk_{C_x}) , and can ask for any signatures by sk_B , it can do this.
- *Deposit:* Trivial, since A' needs not produce any new signatures.

A new coin will now be placed into the system in the ownership of U_h by one of the following methods:

- *Purchase:* U_h will purchase a coin pk_C , where sk_C is the secret key given as input to A' . A' must produce $(\sigma_{sk_{U_x}}(pk_{C_x}), \sigma_{sk_B}(pk_{C_x}))$. Since A' knows sk_{U_x} and sk_B , it can do this.
- *Transfer:* A corrupted user U_x will transfer a coin pk_{C_x} into a coin pk_C owned by U_h , where sk_C is the secret key given as input to A' . A' must produce $(\sigma_{sk_{U_y}}(pk_{C_y}), \sigma_{sk_B}(pk_{C_y}), \sigma_{sk_{C_x}}(pk_{C_x}, pk_{C_y}))$. Since A' knows $(sk_{U_y}, sk_{C_x}, sk_B)$, it can do this.

Finally, A will now invalidate a coin in one of two ways:

- *Deposit:* A will compute $\sigma_{sk_C}(pk_C)$ to give to the bank to invalidate the coin pk_C .
- *Transfer:* A will produce $\sigma_{sk_C}(pk_C, pk_{C'})$ for some $pk_{C'}$.

In either case, this message will be output by A' , breaking RSA security.

5.2.3 Anonymity

The SAF_cT payment system maintains the anonymous attribute of the Burk-Pfitzmann system due to the nature of group signatures. To show this, we will use the following result by Ateniese et al. [6]

Theorem 3 *Under the strong RSA assumption and the DDH assumption in the random oracle model, the group signature given by Ateniese, Camenisch, Joye, and Tsudik is resistant to the following two-stage game: In the first stage, an adversary is given access to all members' secret keys and an oracle that will*

reveal the authorship of any group signatures. It will choose two identities i_0 and i_1 , a message m and some state information S . In the second stage, the adversary is given a group signature sig of m by i_0 or i_1 with equal probability. The adversary may continue to query the authors of any signatures besides sig . It cannot guess the true identity with probability greater than $1/2 + \text{negl}(k)$.

Proof: Proven by Ateniese et. al. [6].

Theorem 4 *Under the strong RSA assumption, the SAF_cT payment system is weakly anonymous.*

Proof: Assume for contradiction that an adversary A , given the judge's public key pk_J , the bank's public and secret keys (pk_B, sk_B) and (pk_B^b, sk_B^b) , a set of all users' public keys $(pk_{U_1}, \dots, pk_{U_n})$, and a set of some users' secret keys $(sk_{U_1}, \dots, sk_{U_a})$, can beat the game outlined for weak anonymity in section 2 with non-negligible advantage ϵ . We show that this enables construction of an adversary A' , which given the group public key pk , and a set of all the users' secret keys $(sk_{U_1}, \dots, sk_{U_n})$ of an ACJT group signature system can break the game in theorem 5.2.3 with advantage ϵ . A' is constructed as follows:

First, A' will perfectly create a simulated world of the SAF_cT payment system. It runs the setup protocol of the SAF_cT system choosing keys for the bank pk_B, sk_B, pk_B^b , and sk_B^b . The judge's portion of the setup protocol is left out, but it is assigned pk_J equal to the challenge key pk . Furthermore, for each sk_{U_i} A' receives, it mimics the register protocol, assigning each user a secret key to match the challenge sk_{U_i} .

A' will now engage in the first stage of a challenge session with A , in which A will request that a series of purchases, transfers, and deposits can occur.

- *Purchase:* If U_x is to purchase a coin pk_{C_x} , A' must produce $(\sigma_{sk_{U_x}}(pk_{C_x}), \sigma_{sk_B}(pk_{C_x}))$. Since A' knows (sk_{U_x}, sk_B) , it can do this.
- *Transfer:* If U_x is to transfer a coin pk_{C_x} into a coin pk_{C_y} owned by U_y , A' must produce $(\sigma_{sk_{U_y}}(pk_{C_y}), \sigma_{sk_B}(pk_{C_y}), \sigma_{sk_{C_x}}(pk_{C_x}, pk_{C_y}))$. Since A' knows $(sk_{U_y}, sk_B, sk_{C_x})$, it can do this.
- *Deposit:* Trivial, since A' needs not produce any new signatures.

A' will then engage in the second stage of a challenge session with A , in which A will choose users (u_0, u_1, u_2) where u_0 possesses some coin c . A' must randomly select $b \in \{1, 2\}$ where c will be transferred to u_b to become c' . It will then be transferred back to A , becoming c'' . A is then required to output a guess $b' \in \{1, 2\}$ where $b' = b$ with probability greater than $1/2 + \epsilon$.

For the first transfer, A' must produce $(\sigma_{sk_B}(pk_{c'}), \sigma_{sk_c}(pk_c, pk_{c'}))$. It can do this, because it knows (sk_B, sk_c) . It must also produce $\sigma_{sk_{u_b}}(pk_{c'})$. To do this, it engages in its own challenge session where it returns the message $pk_{c'}$ and requests that it be signed by either u_1 or u_2 . For the second transfer, it passes $\sigma_{sk_{u_b}}(pk_{c'})$ back to A , along with $\sigma_{sk_B}(pk_{c''})$, produced with sk_B .

A is now required to output b' , and A' will output the same. Since A' perfectly simulates a world for A , A' has the same advantage as A . Therefore, since A has non-negligible advantage, so too does A' , which contradicts theorem 5.2.3.

5.2.4 SAF_cT

Theorem 5 *Under the strong RSA assumption and the DDH assumption in the random oracle model, the SAF_cT system satisfies the SAF_cT properties.*

Proof: We know that the system has security and weak anonymity from theorems 1, 2, and 4. A system is defined to have F_c fairness if it contains an entity called the judge which can identify the owner of the coin. The trace protocol of the SAF_cT system allows the judge to do this because he holds the master private key for the ACJT group by which every coin is signed. A system is defined to have transferability if a coin can be respent by its recipient without the need to deposit it and purchase another at the bank. This follows from the completeness of the *transfer* protocol.

6 SAF_cT protocol extensions

6.1 Micropayments

Application of payments systems such as purchasing web content often generate a large volume of very small payments called micropayments, pennies or even fractions of pennies. In such a case, the computational and network resources of the bank must be considered so that the work of handling these payments does not outweigh the value of the payments themselves. A payment system capable of efficiently dealing with small payments is called a micropayment system.

Rivest [16] was first to propose the idea of using a probabilistic scheme for micropayment as follows. Assume that each coin is valued at c dollars and let p be a positive value much less than 1. Then to handle a payment of value pc dollars, the system allows a user to pay a merchant a coin with probability p . Thus, the expected value of a payment is pc , but the bank is only involved in one out of every $1/p$ transactions on the average, a significant load reduction. In fact, the bank handles about one transaction per c dollars. Thus, by adjusting c , the bank can adjust how much work it must do per value handles.

6.1.1 MR1

Although there are many probabilistic payment systems, we need a model that does not require repeated transactions between the sender and receiver, and that does not force either party to reveal information that could potentially identify them. This is because our system aims to allow a probabilistic transfer of a coin between two anonymous users who may or may not engage in multiple transfers.

We shall model our extension based on MR1, a recent probabilistic micropayment system introduced by Micali and Rivest [12]. Let $p < 1$ be some probability value. MR1 makes use of a fixed public function $F()$ that takes as input arbitrary bit strings and outputs a random value between 0 and 1. Then if a user U wishes to pay a user V , they agree on a string T which contains information about their transaction. This is signed by U and given to V , who then signs that signature to generate $\sigma_V(\sigma_U(T))$. If $F(\sigma_V(\sigma_U(T))) < p$, then the transaction is successful. Thus, the expected value of a payment in MR1 can be less than one coin. Furthermore, the bank need only be involved in some small portion p of the total payments made.

MR1 relies on the use of a signature function that produces signatures indistinguishable from random bit strings and is also deterministic. Micali and Rivest suggested using the RSA signature [15] which satisfies these properties. A deterministic signature is defined strictly by the signing key and message, with no additional randomness. Because of this U cannot cheat V because it must commit its signature before knowing what final value will result from V 's signature. Likewise, V cannot cheat U because it cannot know what U 's signature will look like before committing the message, a description of the transaction T .

As a probabilistic micropayment protocol, MR1 is well suited to the SAF_cT framework since, unlike other non-probabilistic micropayment systems [7, 14], it does not require a user to obtain a large number of "microcoins" that get collected into a macropayment, eliminating the need to change the types of coins in circulation. Another advantage of MR1 is that a single transaction passes between a pair of users. This

Protocol Microtransfer:

1. l is a security parameter controlling the precision of probabilities of successful payment (l should be chosen such that $l = O(\log L)$ where L is the length of the RSA public keys in use)
 2. p , the probability of successful payment, is agreed upon by U and V
 3. U chooses a coin pk_{C_U} with which to pay and sends it to V
 4. User V computes $(pk_{C_V}, sk_{C_V}) = RSASetup(1^k)$
 5. V computes $\sigma_{sk_V}(pk_{C_U}, pk_{C_V}, p) = ACJTGroupSign(pk_{C_U}, (pk_{C_V}, p), sk_V)$ and sends it to U
 6. U checks that $ACJTVerify(\sigma_{sk_V}(pk_{C_U}, pk_{C_V}, p)) = true$
 7. U computes $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p) = RSASign((pk_{C_U}, pk_{C_V}, p), sk_{C_U})$
 8. V checks that $RSASign((pk_{C_U}, pk_{C_V}, p), sk_{C_U}) = true$
 9. V computes $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p) = RSASign((\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p), sk_{C_V})$, let l be the length of this signature
 10. If $F(\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p), l) \geq G(p, l)$ then the payment is rejected. V informs U that he may keep his coin, and the transfer ends.
 11. Otherwise, V informs U that the transfer was successful and sends $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)$ and $\sigma_{sk_{C_U}}((pk_{C_U}, pk_{C_V}, p), \sigma_{sk_V}(pk_{C_V}))$ to B .
 12. B checks that $F(\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p), l) < G(p, l)$, and that $RSASign((\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)) = true$
 13. B checks that $RSASign((pk_{C_U}, pk_{C_V}, p), sk_{C_U}) = true$
 14. B checks that $ACJTGroupVerify(\sigma_{sk_V}(pk_{C_V}, p), pk_J) = true$
 15. B keeps $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)$ and $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p)$ as proof this coin has been relinquished
 16. The bank now computes $\sigma_{sk_B}(pk_{C_V}) = RSASign(pk_{C_V}, sk_B)$ and returns it to V
 17. V keeps $(\sigma_{sk_V}(pk_{C_V}), \sigma_{sk_B}(pk_{C_V}))$ as a coin.
-

Figure 8: SAF_cT Microtransfer Extension

is in contrast to some micropayment systems [14, 11] that require a history of payments between customer and vendor which are eventually collected into a large macropayment. Finally, the MR1 protocol does not require users to reveal their identities to each other, hence preserving anonymity.

6.1.2 Protocol

Our protocol makes use of a public function $F(s, l)$ which takes as input any arbitrary bit string s and based on the low-order l bits of s returns an integer evenly distributed in the range $[0, 2^l]$. It also uses a public function $G(p, l)$ which takes as input an arbitrary value p in the range $(0, 1)$ and returns $2^l p$ rounded down to the nearest integer.

Figure 8 shows the microtransfer protocol. This protocol is intended to be used in addition to the existing SAF_cT system, so the setup is the same. When U makes a micropayment to V , they must first agree on p , a probability that dictates the quantity of the payment. U then chooses a coin with which to pay, and sends pk_C from that coin to V . V randomly generates a public/private key pair pk_{C_V} and sk_{C_V} and holds sk_{C_V} . He signs both the sending coin and his new receiving coin with his group secret key and transmits it

to U . This serves as proof both that the judge can identify him, and as a commitment to use this coin for this transaction in resolving later disputes.

U checks this signature, then signs and sends a transfer request identifying pk_{C_U} , pk_{C_V} , and p , the sending and receiving coins and probability of transfer. This commits him to use this coin for this transaction.

V now signs this quantity with sk_{C_V} . This final signature is passed to the public function $F()$, and compared to the result of passing p to the public function G , both of which map to integers in the same range. If the result is less, then the transfer succeeds, and the protocol continues. Otherwise V the transfer fails, and V accepts the attempted transfer as payment (U possesses his group signature and can ask for dispute resolution if he refuses to recognize this as payment).

This final signature, along with a signature of the transaction by sk_{C_U} are sent to B . B confirms that this request was authorized by a valid coin's owner by verifying this with pk_{C_U} and checking that pk_{C_U} is on the coin list. B also verifies that the recipient is a traceable user by verifying the signature of pk_{C_V} with the group public key. Finally, B verifies in the same manner that V did that the probabilistic transfer was a success. B then removes pk_{C_U} from the list and adds pk_{C_V} . B also stores a record of this transaction and returns to V a signed proof of coin creation for dispute resolution.

6.1.3 Analysis

The first thing to note is that since the buyer must show a coin to the seller regardless of whether or not the transaction is successful, and that coin may be reused later if it was not, there is some anonymity is lost since the seller might link multiple failed transactions to the same buyer. Because of this, this system can only meet the weak anonymity property. However, a buyer can partially protect himself from this by using different coins in multiple transactions with the same entity, and occasionally transferring his coins to himself to refresh their keys.

In addition to the strict securities described in the basic *SAFT* properties which protect coin ownership and regulate the existing coins in the system, it is necessary that neither the sender nor the receiver can manipulate the probability of the success of the transaction unfairly.

Theorem 6 *Under the strong RSA assumption, relative to a randomly chosen public key pk of size L , the last $\log L$ bits of an RSA signature using sk are computationally indistinguishable from a string of random bits of the same size.*

Proven by Alexi et al. [2].

Theorem 7 *Under the strong RSA assumption, no input by the payer to the Microtransfer algorithm after agreeing on the probability of success will alter that probability.*

Proof: The only inputs U , the payer, are given in the algorithm are the choice of pk_C from among his coins to pay with, and the signature $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p)$. It is clear that if the value of the input $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)$ to $F()$ can be modelled as a random oracle from the point of view of U , then the probability of transfer remains fair.

U must pick his input pk_C , which is the only portion of the signature $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)$ he can affect, without yet knowing the values pk_{C_V} or sk_{C_V} . Since sk_{C_V} is unknown to U , by the security of RSA the resulting signature is random to U and thus the probability of transfer remains fair.

Theorem 8 *Under the strong RSA assumption, no input by the payee to the Microtransfer algorithm after agreeing on the probability of success will alter that probability.*

Proof: Again, it is clear that if $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)$ can be modelled as a random oracle from the point of view of V , the payee, then the probability of transfer remains fair. V can only possibly influence this signature with his choice of pk_{C_V} , which influences the value $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p)$ which is signed, and his choice of sk_{C_V} which is used to compute the signature.

Since sk_{C_U} is unknown to V , the resulting signature $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p)$ is random or the strong RSA assumption is false. It is proven by Alexi et. al. [2] that relative to a randomly chosen public key pk of size L , the last $\log L$ bits of a signature using sk are computationally indistinguishable from a string of random bits of the same size. Thus, since V must choose the keys (pk_{C_V}, sk_{C_V}) before seeing the random input $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p)$, the final output of the signature $\sigma_{sk_{C_V}}(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}, p), p)$ must be random as well. Therefore, the probability of transfer remains fair.

6.2 Offline transfer

We would also like to extend our system to allow for transfers which do not involve contacting the bank. This can be useful when transactions need to occur in low connectivity situations. For example, a user might bring his coins on removable media with him to a real store that is equipped to accept offline transfer as payments.

However, it also potentially increases the risk of double spending, since there is no longer a central authority in contact for each transaction to check if a coin has already been used. We will propose an extension to our system that keeps track of offline transfers with a list of signatures for each transaction. Since the coin contains a list of signatures, and the payment system is a fair system, the bank will eventually be able to detect double spending once the coin is deposited, and the judge can identify the perpetrator.

6.2.1 Protocols

Our protocol is once again an extension of the basic *SAF_cT* system. The extension includes two protocols. The first is an *OfflineTransfer* protocol, that is intended to allow a number of transfers to occur offline. It removes either a regular online coin or an existing "offline coin" from the payer and generates an offline coin for the payee.

This offline coin can be passed to a final *Resolve* protocol which resolves all of the offline transfers that occurred with the bank records and results in a normal coin that can be used in normal online transfers.

Our approach is straightforward; we append to the coin a list of signatures for each user that transfers it to the next. In other words, a coin C has a *history* C_h which is a list that may be appended to. Once the coin is finally deposited, the bank must verify each signature in C_h . In terms of computation, this does not reduce work for the bank, but in terms of network communication, much is saved.

Figure 9 shows the offline transfer protocol. As with the microtransfer protocol, this is an extension to the *SAF_cT* system to allow a number of transfers to occur without contacting the bank.

To begin with, V must generate a new RSA signature pair to represent his new ownership of the coin. He signs the public key with his group signature and transmits this to U . U chooses a coin with which to pay V , and if this coin is not already an offline coin, he makes it one by adding to it an empty "history". He transfers the coin to V by signing V 's new signature pair with the coin's old one. This signature along with the group signature by U are appended to the coin's history which is given in full to V .

To resolve this coin and return it to a normal coin without history, a user transfers the coin's entire history to B . This history will contain a group signature by each owner, which can be uniformly verified with the group public key. It will also contain a RSA signature by each public key that has been used to transfer the coin. The current public key of the coin should be signed by the last signature in the history, which will then

Protocol OfflineTransfer:

1. User V computes $(pk_{C_V}, sk_{C_V}) = RSASetup()$
2. V computes $\sigma_{sk_V}(pk_{C_V}) = ACJTGroupSign(pk_{C_V}, sk_V)$
3. User U chooses coin C_U with which to pay
4. C_U is made into an offline coin: If it is an online coin (i.e. it has no history C_{Uh}), then C_U is transformed by adding an empty history C_{Uh} . Thus $C_U = (\sigma_{sk_U}(pk_{C_U}), \sigma_{sk_B}(pk_{C_U}), C_{Uh})$
5. U computes $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}) = RSASign((pk_{C_U}, pk_{C_V}), sk_{C_U})$
6. U sends $\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V})$, C_{Uh} , and $\sigma_{sk_U}(pk_{C_U})$ to V
7. V checks that $RSASignVerify(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}), pk_{C_U}) = true$
8. V checks that $ACJTGroupVerify(\sigma_{sk_U}(pk_{C_U}), pk_J) = true$
9. V sets $C_{Vh} = C_{Uh}$ and appends $(\sigma_{sk_{C_U}}(pk_{C_U}, pk_{C_V}), pk_{C_U})$, $(\sigma_{sk_U}(pk_{C_U}), pk_J)$
10. V keeps as a coin $C_V = (\sigma_{sk_V}(pk_{C_V}), \sigma_{sk_B}(pk_{C_U}), C_{Vh})$ (note that the signature by sk_B remains for the original coin, which can be traced back to in C_{Vh})

Protocol Resolve:

1. U sends an offline coin $C = (\sigma_{sk_U}(pk_C), \sigma_{sk_B}(pk_C), C_h)$ to B
2. B verifies all signatures in C_h , if a RSA signature is invalid, or the bank records show that it originated from a false coin, then the signer's group signature is traced. If that signature is unavailable or false, then the user who accepted it is traced and held responsible for paying the final recipient. This continues until a valid signature is found, or all of the signatures including the final recipient's are invalid in which case the coin is valueless and ignored.
3. B checks that $ACJTGroupVerify(\sigma_{sk_U}(pk_C), pk_J) = true$
4. B now computes $\sigma_{sk_B}(pk_C) = RSASign(pk_C, sk_B)$ and returns it to U
5. B stores all signatures as if each transaction had been an online transfer
6. U keeps $(\sigma_{sk_U}(pk_C), \sigma_{sk_B}(pk_C))$ as a coin.

Figure 9: Offline Transfer

recursively be signed by each signature before it until the first signature in the history which was produced by either a normal transfer or purchase and thus will be held by the bank. Since these intervals contain each public key, each RSA signature should be verifiable.

If the same coin is used more than once, or if a payment is made with a fake coin, the bank's records will show this in the same way it would detect it with online transfers. If a coin is double-spent, then the first time it is resolved, the transfer will be successful. At this point, the bank records will be updated to show that the originating coin, and all subsequent coins in the coin history, are no longer valid coins. Thus, when the second coin is resolved, it will be handled in the same way as a forged coin.

Since each user is required to obtain and verify a group signature from the user paying him a coin, we can always hold the owner of the earliest valid group signature responsible for forgeries. It is necessary to hold this person responsible, otherwise a user can always invent a history of transactions leading to himself and claim that he was the one who was cheated. If a user wishes to protect himself, he must always ensure that the person he receives an offline transfer from has given him a valid group signature.

Once all signatures have been verified, the bank gives its own signature, and the coin's history is removed and stored at the bank records, making the coin a valid online coin.

6.2.2 Analysis

In terms of security, allowing offline transfers weakens our system into a *weaklysecure* system, since double spending will not be immediately prevented, but rather detected when a coin is eventually resolved before depositing. We must show that improper behaviors will be eventually detected.

Theorem 9 *Under the strong RSA assumption, a group of users conspires to transfer offline an invalid coin to an honest user, one of them will be detected.*

Proof: For a coin C , the coin history C_h contains a list of group signatures of every owner the coin was transferred through since last it was a valid online coin, in order. The initial coin contains a signature by sk_B which is verified by pk_B during the *Resolve* protocol. Thus, if the transfer begins with an invalid coin, the verification must fail and the bank will detect this or the strong RSA assumption is false. At this point, the bank will possess a group signature by the initial transferer U_0 , who must be one of the conspiring users. By the correctness of ACJT group signatures, this signature can be opened to identify U_0 .

If the group signature is invalid, then U_1 , the user that the coin was next transferred to, would not have accepted the coin, according to the *Offlinetransfer* protocol. If he did accept it, then by definition he too is a conspiring malicious user, and his group signature will be in C_h as well. Either his signature is correct and he can be traced, or the person he transferred it to is also a conspiring malicious user. This continues until we have a valid signature of a malicious user U_i , or until the coin is transferred to the malicious user U_n who initiated the *Resolve* protocol, or U_h , an honest user. If we have the signature of U_i , by the correctness of ACJT group signatures, this signature can be opened to identify U_i .

According to the *Offlinetransfer* protocol, as an honest user U_h will attempt to verify the signature of U_{h-1} the malicious user who sent it to him without providing a valid signature. By the security of ACJT group signatures, this verification will fail and thus U_h will not accept the coin. If U_n as well did not provide a valid group signature, then when the bank attempts to verify it, as in the *Resolve* protocol, by the security of ACJT group signatures, the protocol will fail and the coin will not be converted into a valid online coin. Thus, they cannot transfer this coin any further to an honest user.

7 Performance

7.1 Implementation

Our server, client, and judge for the *SAFT* payment system are implemented in Java. We have chosen this platform because it has a number of existing libraries for many of the basic signature systems we need to use as building blocks for our payment system, and is easy to develop in. Source code is available at

<http://web.mit.edu/bdv/www/saft>

The server and client are coded to run the online protocol without offline transfer and micropayment extensions. It consists of three entities:

- *Bank*: A server application that acts as the bank. On startup, it engages in the *Setup* protocol. It then maintains a list of valid coins in a hash table using their public keys as keys, and listens for requests on a single port. Requests arrive flagged as *Register*, *Purchase*, *Transfer*, and *Deposit* protocols. It

runs with a Java GUI open that displays and logs all incoming requests. Users can also select items from the log to have the Bank send a *Trace* request to the judge which will display a user ID in a popup window.

- *User*: A client application that acts as a user. On startup, it must be given an IP and port address for the judge, so that it may request that the *Judge* participate in the *Register* protocol, and for the bank. In this implementation, its ID is simply its own IP address and a port it uses to receive *Transfer* requests. It then maintain a list of coins owned by its user, visible in a GUI, and can request that the *Bank* participate in *Purchase* and *Deposit* protocols. It can also make and listen for requests by or of other users to engage in the *Transfer* protocol, for which it must also interact with the *Bank*. To do these actions it must be given the IP and port address of the Bank or of another user (their ID).
- *Judge*: A server application that acts as the judge. On startup, it must engage in the *Setup* protocol. It then maintains a list of registered users, and listens for requests by the *Bank* to engage in the *Trace* protocol, and requests by users to engage in the *Register* protocol. In the current implementation, it will respond to any *Trace* requests. In a practical situation, it should contain some mechanism to verify the identity of the authority making requests.

7.2 Results

All tests were run on a Pentium 4-m 1.2 GHz processor with 512MB RAM running the Windows XP Professional Operating System and the J2SE Runtime Environment 5.0 Update 2. All users and the bank server were simulated on the same machine. As such, our data does not reflect network transfer times, which in a practical situation would likely scale with coin sizes. Our system was only capable of measuring timing to a precision of 10ms. Times are listed in increments of greater precision because each running time was measured 5 times with an average taken. Results are shown in the table below:

Table 1: Running time (ms) for SAFT protocols

| Users | Bank Setup | Judge Setup | Register | Purchase | Transfer | Deposit | Trace |
|-------|------------|-------------|----------|----------|----------|---------|-------|
| 1 | 668 | 280 | 216 | 584 | 688 | 38 | 52 |
| 10 | 670 | 282 | 216 | 582 | 684 | 40 | 54 |
| 100 | 668 | 278 | 214 | 582 | 684 | 38 | 52 |
| 1000 | 672 | 280 | 216 | 582 | 686 | 40 | 52 |

As the Table 1 shows, the running times of the protocols did not vary much by the number of users. Similarly, running times generally did not vary with number of coins in the system, with coins of up to 1000. We did not have resources to test, but we speculate that the trace and register protocols might slow down marginally once a number of users reaches an amount that would impact the performance of the hashtables used to store them. Similarly, purchase, transfer, and deposit might slow down marginally with extremely large numbers of coins.

The slowest running protocols were the *Purchase* and *Transfer* protocols, which required group signature and verification and also RSA key generation. Also, the one time *Setup* protocol for the bank requires the generation of signatures for both RSA and RSA blind signatures. Finally, *Deposit* and *Trace*, the only protocols which did not involve any sort of key generation or group signatures, were the fastest.

Coin size was static and did not vary with number of users. Table 2 shows the space used by each entity to store information about one coin.

Table 2: Memory use (bytes) to store one coin in the SAFT system

| User | Bank (active coin) | Bank (spent coin) |
|------|--------------------|-------------------|
| 1663 | 901 | 1191 |

8 Conclusion

We outlined the *SAFT* properties of security, anonymity, fairness, and transferability and discussed how a number of current systems satisfied these properties. We then presented a method for achieving fair and transferable ECash based on previously known group signature schemes and provided an implementation and performance data for this method. This ECash system satisfies more of the *SAFT* properties than any other currently known system. This method was implemented as a payment system in Java.

It allows for the flexibility and privacy fundamental to electronic commerce, while providing an avenue for law enforcement to expose those users who abuse the system for illegal activities. Additionally, extensions to this system were discussed to allow for probabilistic payment and offline payment.

References

- [1] S.A. Vanstone A.J. Menezes, P.C. van Oorschot. *Handbook of Applied Cryptography*. CRC Press, ISBN: 0-8493-8523-7, 1996.
- [2] W. Alexi, B. Chor, O. Goldenreich, and C.P. Schnorr. RSA/Rabin functions: Certain parts are as hard as the whole. In *Siam Journal on Computing*, volume 17/2, pages 194–209. Springer-Verlag, 1988.
- [3] H. Burk and A. Pfitzmann. Digital payment systems enabling security and unobservability. *Computers & Security*, 8/5:399–416, 1989.
- [4] D. Chaum. Blind signature system. In *Advances in Cryptology: CRYPTO '83*, pages 153–156. Plenum Publishing, 1984.
- [5] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology: EUROCRYPT '91*, volume 547, pages 257–265. Springer-Verlag, 1991.
- [6] M. Joye G. Ateniese, J. Camenisch and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology: CRYPTO '00*, volume 1880, pages 255–270. Springer-Verlag, 2000.
- [7] S. Glassman and et. al. M. Manasse. The millicent protocol for inexpensive electronic commerce. In *World Wide Web Journal, Fourth International World Wide Web Conference Proceedings*, pages 603–618. O'Reilly, 1995.
- [8] J.M. Piveteau J. Camenisch and M. Stadler. An efficient fair payment system. In *Proceedings of the 3rd ACM conference on Computer and communications security: CCS '96*, pages 88–94. ACM Press, 1996.
- [9] J. Solinas L. Law, S. Sabett. How to make a mint: The cryptography of anonymous electronic cash. *National Security Agency, Office of Information Security Research and Technology*, June 18, 1996.

- [10] J. Piveteau M. Stadler and J. Camenisch. Fair blind signatures. In *Advances in Cryptology: EURO-CRYPT '95*, volume 921, pages 209–219. Springer-Verlag, 1995.
- [11] W. Mao. A simple cash payment technique for the internet. In *Proceedings 1996 European Symposium on Research in Computer Science: ESORICS '96*. Springer-Verlag, 1996.
- [12] S. Micali and R.L. Rivest. Micropayments revisited. In *Proceedings of the The Cryptographer's Track at the RSA Conference on Topics in Cryptology: CT-RSA '02*, volume 2271, pages 149–163. Springer-Verlag, 2002.
- [13] David Pointcheval. Strengthened security for blind signatures. In *Advances in Cryptology: EURO-CRYPT '98*, volume 1403, pages 391–405. Springer-Verlag, 1998.
- [14] R. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *Proceedings of the International Workshop on Security Protocols*, volume 1189, pages 69–87. Springer-Verlag, 1997.
- [15] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public key cryptosystems. In *Communications of the ACM*, volume 21/2, pages 120–126. ACM Press, 1978.
- [16] Ronald L. Rivest. Electronic lottery tickets as micropayments. In *Proceedings of the First International Conference on Financial Cryptography: FC '97*, volume 1318, pages 307–314. Springer-Verlag, 1997.
- [17] S. Micali S. Goldwasser and R. Rivest. A digital signature scheme secure against chosen message attacks. In *SIAM Journal on Computing: SIAM '88*, volume 17(2), pages 231–308, 1988.
- [18] Daniel R. Simon. Anonymous communication and anonymous cash. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology: CRYPTO '96*, volume 1109, pages 61–73. Springer-Verlag, 1996.