On Electronic Payment Systems

Ronald Cramer, Ivan Damgård and Jesper Buus Nielsen
CPT 2009

April 22, 2009

Abstract

This note is an introduction to the area of electronic cash (eCash) schemes. The note presents an informal definition of security of an eCash scheme and presents two examples of eCash schemes, each along with an informal analysis of its security.

Contents

1 Introduction

In this note, we look at ways to securely transfer money electronically. In such a game, there are always at least 3 players that we will call the User, the Shop and the Bank. The basic goal is for the User to pay money to the Shop, in a scenario where they both have accounts in the Bank.

You should not over-interpret the names or the particularities of this model. In real life, the User may be a company, the Shop may be a private person, and several Banks or other financial institutions will be involved in the transaction. We stick to the scenario above for simplicity.

2 Check-based Systems

As the headline says, this type of system uses an electronic equivalent of a check. The user is assumed to have a key-pair pk_U, sk_U, where the Bank and Shop can get hold of authentic copies of pk_U. This can be ensured, for instance by a standard public key infrastructure. Danish home banking systems use this approach, where PBS (Pengeinstitutternes Betalingssystemer) plays the role of Certification Authority.

The User then simply issues a signed payment order. This consists of a data string d specifying the amount, date and time, and the receiver of the money. This, together with the signature $S(sk_U, d)$ is sent, either to the Shop who forwards it to the Bank later, or directly to the Bank, if the Bank can be assumed to be on-line. In any case, the receiver of the message can check the signature. And the Bank can move the money from one account to the other, assuming, of course, that the signature was valid and that this payment order has not been processed before.

Assuming that the Bank behaves correctly, this clearly prevents a User from spending other Users' money, and the Shop is guaranteed to get payed if it checks the payment order on-line. Otherwise it has to trust that the User has money on his account, just as with paper based checks.

Some variants of this system allow some alternative methods for authenticating the identity of the User, and/or provide some form of anonymity. For instance, the SET protocol (Secure Electronic Transactions, the standard of the Credit Card companies) assumes that the Bank has a public key pk_B. Then the parts of the payment order d that can identify the User, plus the User signature are encrypted under pk_B. Then only the Bank will know the identity of the User.

Also, SET allows for the User not having a key pair. Then the signature is replaced by the credit card data (card number, expiration date) of the User, and all communication is then encrypted between User and Shop and Shop and Bank.

Although most systems in real life work along these lines, they are relatively uninteresting from a protocol design point of view, because their security follow quite trivially from security of the signature scheme involved. What is perhaps more important is that the anonymity this type of system can provide is limited: the Bank *must* know the identity of the payer in order to move money from the right account. So there is no way such a system can be equivalent to the way ordinary cash functions.

3 Cash-Based Systems

The goal in this section is to look at systems that provide functionality equivalent to real cash: The User *withdraws* money from his account, he *pays* money to the Shop, and the Shop *deposits* the money in its own account. Moreover, just like with real cash, we will require privacy or anonymity for the User: there must be no way a particular deposit transaction can be linked to any particular withdrawal. This should hold, even if the Bank and all Shops cooperate. Thus, there is no way that any "Big Brother" can collect information on how a particular user spends his money[1].

Implementing this electronically must mean that the data the User gives to the Shop in the payment transaction is somehow worth money, it is an equivalent of a coin, if you will: the Shop can afterward do a deposit and increase the balance on its account. Saying that a string of bits is worth money immediately raises some questions: Couldn't the user make such electronic coins himself without talking to the bank? and even if he couldn't, why couldn't he spend the same coin several times? And finally, if the bank has to issue these coins and therefore sees them, how can we prevent the bank from recognizing them when they get deposited?

This may seem like difficult, if not impossible problems at first sight, nevertheless, some quite satisfactory solutions exist, as we shall see.

Of course, one can prevent Users from making fake coins by requiring that the Bank signs them when they are issued. But this seems to immediately violate anonymity because the Bank will be able to recognize the signatures it computed when the coins come back and are deposited. But this can be solved by so called blind signature schemes, where the Bank does a protocol with the User, and as a result the User obtains the Bank's signature on some message, nevertheless the Bank has no information about this message. We will look at how this is done below.

Another problem is that simply signing coins does not prevent users from double spending. Of course, if the Bank is on-line at every payment, it can check on the fly that a given coin has not been used before. But this is often unrealistic. However, it turns out that some security can still be provided, even if the bank is not on-line and is only involved when money are deposited: it is possible to "bake into" the coin the identity of the user in such a way that this identity will be revealed immediately if one tries to spend the coin more than once. We will look at this later in detail.

3.1 Basic Set-up

We will assume that the Bank has a public-key pair pk_B, sk_B, and all players know pk_B. We then define how the system should work on a high level:

Withdrawal A protocol for the User and the Bank. The User identifies himself to the bank, and some interaction happens. If the protocol is successful, the User obtains

[1]of course, in the extreme cases where there is only one user, one trivially knows who did all transactions, but this can never be prevented

the following.

- An *electronic coin* c.
- A private *validation key* v. This key depends on the User's identity $ID(U)$ and on the coin c.
- The Bank's *signature* σ on the coin c, with respect to pk_B.

We assume that there is a general rule saying what the value is of such a coin. For instance, its denomination could for instance be encoded in the Bank's signature. Finally, the Bank withdraws the corresponding amount from the User's account.

Payment A protocol for User and Shop. The User sends c, σ to the Shop, who checks the signature. Then some interaction happens, where the User uses v as private input. The Shop uses a string pid_S as (public) input. Basically, pid_S is a payment identifier and contains the identity, $ID(S)$, of the Shop (say, its account number), and a unique transaction identifier, such as a large random number.

If the protocol is successful, the Shop obtains *deposit key* d. Without loss of generality we may assume that the string pid_S is part of d.

Deposit A protocol for Shop and Bank. This may be interactive, but usually just consists of the Shop sending c, σ, d to the Bank. The Bank checks whether the signature σ on the coin c is valid and whether the deposit key d is valid for c.

If so, and if c has not been deposited before, the Bank inserts money on the account of the Shop, and enters (c, d) into its database.

If it has been deposited before, the database will already contain an entry of form c, d'. If the payment identifier pid_S encoded in d and the payment identifier pid'_S encoded in d' is the same, then the shop used the same payment identifier twice or tried to deposit the same coin twice, and (c, d) is simply rejected. If $pid_S \neq pid'_S$, the system should now be constructed in such a way that from c, d, d' the Bank can efficiently compute $ID(U)$ and so can identify the User who has tried to cheat. Depending on the rules of the system, the Bank may still transfer the money to the Shop in this case.

We can then (quite informally) list some properties that are needed to make this type of system secure:

Bank Security:
No false coins: Assume the Bank is honest. If c, σ, d is a valid deposit, then the coin c originates from a particular unique execution of the withdrawal protocol. Moreover, a single execution of the withdrawal protocol results in exactly one coin (with the right denomination). This holds even if all Users and Shops conspire against the Bank. Another way to phrase this requirement is that after any run of the protocols with cheating users and shops, the amount withdrawn from accounts in executions of the Withdrawal protocol is at least as large the the amount deposited on accounts in execution of the Deposit protocols.

Catching double spenders: From valid deposits c, σ, d and c, σ, d', the Bank can efficiently compute the identity $ID(U)$ of the User from whose account the coin c was withdrawn. This is also called *accountability-after-the-fact.*

User Security Only the legitimate (honest) User can withdraw electronic coins from an account, and only he can (double-) spend them.

Shop Security If the Shop accepts a payment, the Bank accepts the subsequent deposit made by the Shop.

User Anonymity Consider all payments made by honest Users. Then the Bank cannot trace any of these payments to any specific User, even if it collaborates with the Shops. This is called *anonymity.* Moreover, it cannot even decide whether two payments were made by the same (anonymous) User or not, i.e., payments are unlinkable.

The role of the string pid_S chosen by the Shop during a payment transaction should now be clear. Inclusion of the Shop's identity ensures that no one else can deposit the coin and claim the money. The unique transaction identifier ensures that the Shop does not accept the same payment, i.e., the same c, σ, d, twice in distinct transactions.

3.2 Hiding the Identity and Blind Signatures

The first anonymous electronic cash scheme was based on the following principles, which also play a role in the concrete scheme we present later on.

3.2.1 Commitment-Based Identity Hiding

A first very simple (but very inefficient) example of how to hide the user identity in the coin is to use an unconditionally hiding commitment scheme, with commitment function *com*, which takes as input a string of the same length as a user identity and some random input. Then the user should created the coin as a set of k pairs of commitments

$$c = ((c_1^0, c_1^1), (c_2^0, c_2^1), ..., (c_k^0, c_k^1))$$

where $(c_2^0, c_2^1) = (com(x_i^0, r_i), com(x_i^1, s_i))$ and x_i^0, x_i^1 are chosen at random, subject to $U = x_i^0 \oplus x_i^1$. Of course a dishonest user may not do this correctly, but it is then the role of the withdrawal protocol to ensure that the user does not get the banks signature on something invalid. The validation key simply consists of all the random strings r_i, s_i that are needed to open the commitments. This coin reveals nothing about U, by the hiding property of commitments.

Now, in the payment phase, the Shop sends the string $e = e_1, ..., e_k$, and the user must now for each i open $c_i^{e_i}$. The deposit key consists of the $x_i^{e_i}$'s and r_i's or s_i' revealed. Note that if the same coin is spent twice, and the string e' is sent the second time, except with negligible probability there will be an i for which $e_i \neq e_i'$, and this means by the binding property of commitments that the two deposit keys contain x_i^0, x_i^1 and so we we can find u easily.

We will not use this principle later, as it is inefficient, but included it to demonstrate how an identity can be hidden in a way to reveal it only after double-spending.

3.2.2 Blind Signature Schemes

A *blind signature scheme* is one in which the receiver gets a signature on a message of his choice while the signer remains ignorant about what he is signing.

A bit more precisely, the receiver has the message m as private input. Then there is interaction between signer and receiver, as a result of which the receiver gets a signature σ on m as private output. The signer has no information on m, σ, in the sense that from his point of view, all pairs m', σ' where σ' is a valid signature, are equally likely.

Exploiting the multiplicative properties of the basic RSA signature scheme, it can be turned into a blind signature scheme as follows.

Let n, e be the signer's public RSA-key, and d his secret key.

- The receiver has message $m \in \mathbb{Z}_n^*$. He selects a *blinding factor* $r \in \mathbb{Z}_n^*$ at random, and computes
$$R \leftarrow m \cdot r^e \bmod n,$$
and send R to the signer.
- The signer computes
$$S \leftarrow R^d \bmod n,$$
and returns S to the receiver.
- The receiver finally computes
$$\sigma \leftarrow S \cdot r^{-1} \bmod n.$$

Note that σ is indeed a valid signature, since

$$\sigma^e \equiv S^e \cdot r^{-e} \equiv R \cdot r^{-e} \equiv m \bmod n\ .$$

This blind signature together with the identity hiding technique discussed above are by themselves are not sufficient for an e-cash scheme, one obvious problem is that if we simply make a withdrawal protocol by having the bank blindly sign a coin of the form we sketched above, the bank cannot be sure that it is signing a coin of the correct form, and we cannot be sure that double spenders can be traced.

In principle the user could prove using general zero-knowledge techniques that he executed the protocol correctly. This would solve the problem, but would be very inefficient. Instead we look below at a more efficient scheme.

3.3 Exercises

The questions below are actually relevant for any eCash scheme. But for concreteness, suppose we use the eCash scheme sketched above with the User identity hidden in commitments. Suppose (for simplicity) that we don't care about efficiency, and the user proves during the withdrawal that what he sends for the Bank to sign was correctly formed, using general zero-knowledge techniques.

Exercise 1 Consider the payment protocol. It was stated earlier that it was important that the challenge or transaction identifier e was chosen to guarantee that different shops use different e-values—this can be done, for instance, by requiring that the shop uses its own name as a part of the challenge. Suppose this precaution is not taken, and shops are just supposed to use random e-values. Explain how malicious users and shops can attack the scheme.

Exercise 2 Considering the previous question, it is indeed necessary that different shops use different challenges. So is it secure, if each shop just uses its own name as challenge? why or why not?

Exercise 3 Assume the Bank is malicious and wants to frame a user, that is, make it seem as if that user had withdrawn a coin and spent it twice. Explain how this is possible with the scheme we assume here. Sketch how the problem might be solved. [Hint: suppose there is a public database where for each user U, there is an entry containing U's user name and $f(R_U)$, where f is a one-way function and R_U is a random string known only to user U. Use this database in some way for the withdrawal protocol instead of only U's username.]

4 A Commitment Scheme

The previous section sketched an eCash, where the user identity was hidden using a commitment scheme.

In section we describes a commitment scheme, which will be used in subsequent sections to create an efficient eCash system.

4.1 Set-up

We assume that some values have been set up and that all parties know these values. Specifically we assume that a group G_q is known with prime order q. Furthermore, two generators g_1 and g_2 of G_q are known. These values have been set up in such a way that nobody knows $\log_{g_2} g_1$. I.e., we assume that no one can efficiently compute x such that $g_1 = g_2^x$.

4.2 The Commitment Scheme

The commitment scheme will commit to secrets $w_1 \in \mathbb{Z}_q$.[2]

Given $w_1 \in \mathbb{Z}_q$ we pick a uniformly random randomizer $w_2 \in \mathbb{Z}_q$ and commit as

$$com = \texttt{commit}(w_1, w_2) = g_1^{w_1} g_2^{w_2} \ .$$

Assume now that we have any commitment $com \in G_q$ and that we have any secret $w_1 \in \mathbb{Z}_q$. Then by letting $w_2 = \log_{g_2}(com \cdot g_1^{-w_1})$ we have a value $w_2 \in \mathbb{Z}_q$ such that $com = g_1^{w_1} g_2^{w_2}$. It is straight-forward to verify that there exists only one such value w_2. This proves the following claim.

[2]In particular it can commit to a k-bit string $w_1 \in \{0,1\}^k$, where $2^k < q$, by interpreting w_1 as a number in $\mathbb{Z}_q$ and committing to this number. To commit to longer strings, we cut it in k-bit blocks and commit to the blocks in parallel.

Claim 1 *For every $com \in G_q$ and $w_1 \in \mathbb{Z}_q$ there exists a unique $w_2 \in \mathbb{Z}_q$ such that $com = \texttt{commit}(w_1, w_2)$.*

This in particular implies that `commit` is perfect hiding.

Assume now that we have two different openings of the same commitment. I.e., we have values $com, w_1, w_2, w_1', w_2'$ such that $w_1 \neq w_2'$ and $com = \texttt{commit}(w_1, w_2)$ and $com = \texttt{commit}(w_1', w_2')$. By definition this means that

$$com = g_1^{w_1} g_2^{w_2}$$

and

$$com = g_1^{w_1'} g_2^{w_2'} ,$$

from which it follows that

$$g_1^{w_1} g_2^{w_2} = g_1^{w_1'} g_2^{w_2'} .$$

From this we can conclude that

$$g_1^{w_1 - w_1'} = g_2^{w_2' - w_2} .$$

Since g_2 has prime order q it follows that if we set $x = (w_2' - w_@)(w_1 - w_1')^{-1} \bmod q$, then

$$g_1 = g_2^x ,$$

where $(w_1 - w_1')^{-1} \bmod q$ exists because $w_1 \neq w_1'$.

Since $x = (w_2' - w_2)(w_1 - w_1')^{-1} \bmod q$ can be computed efficiently from w_1, w_2, w_1', w_2' this means that if one can efficiently compute a commitment *com* and two different openings of *com*, then one can also efficiently compute x such that $g_1 = g_2^x$. Since we have assumed that one can*not* efficiently compute x such that $g_1 = g_2^x$, we have proved the following claim

Claim 2 *The commitment scheme* `commit` *is computationally binding.*

4.3 The Σ-Protocol

Assume now that we have a commitment $com = g_1^{w_1} g_2^{w_2}$. We want to have a Σ-protocol for proving that one knows an opening (w_1, w_2) of *com*.

To simplify the notation a bit we assume that we have $h = g_1^{w_1} g_2^{w_2}$ and we want a Σ-protocol for the relation $R = \{(h, (w_1, w_2)) | h = g_1^{w_1} g_2^{w_2}\}$. I.e., we just rename *com* to h.

The Σ-protocol proceeds as follows:

- The Prover selects v_1, v_2 at random from $\mathbb{Z}_q$, computes

  $$a \leftarrow g_1^{v_1} g_2^{v_2},$$

  and sends a to the Verifier. Here $r = (v_1, v_2)$ is the randomness used to compute the first message.

- The Verifier chooses e at random from $\mathbb{Z}_q$ and sends e to the Prover.

- The Prover computes
$$z_1 \leftarrow ew_1 + v_1 \bmod q$$
$$z_2 \leftarrow ew_2 + v_2 \bmod q,$$
and returns $z = (z_1, z_2)$ to the Verifier.

- The Verifier checks whether
$$g_1^{z_1} g_2^{z_2} = ah^e \ .$$

We verify that this is indeed a perfect witness-indistinguishable Σ-protocol.

Correct: This is straight-forward to verify as
$$g_1^{z_1} g_2^{z_2} = g_1^{ew_1+v_1} g_2^{ew_2+v_2} = (g_1^{w_1} g_2^{w_2})^e g_1^{v_1} g_2^{v_2} = h^e a \ .$$

Special honest verifier zero-knowledge: Given h and e, pick $z_1, z_2 \in \mathbb{Z}_q$ uniformly at random, let $z = (z_1, z_2)$ and let $a = g_1^{z_1} g_2^{z_2} h^{-e}$. Then it is straight-forward to verify that (h, a, e, z) has exactly the same distribution as in the protocol.

Special soundness: Clearly, from $g_1^{z_1} g_2^{z_2} = ah^e$, $g_1^{z_1'} g_2^{z_2'} = ah^{e'}$ with $e \neq e' \bmod q$, we get
$$g_1^{\frac{z_1 - z_1'}{e-e'}} g_2^{\frac{z_2 - z_2'}{e-e'}} = h \ .$$

Perfect witness-indistinguishable: This means that no matter how e is chosen by the Verifier, all possible pairs (w_1, w_2) consistent with h are equally likely from the point of view of the Verifier.

To see this, let $h \in G_q$ be any value. The verifier sees $a \in G_q$ and sees $z_1, z_2 \in \mathbb{Z}_q$ such that
$$g_1^{z_1} g_2^{z_2} = ah^e \ .$$

So, the verifier knows $(h, a, e, (z_1, z_2))$. This is however consistent with the prover knowing any witness (w_1, w_2).

Let namely $w_1, w_2 \in \mathbb{Z}_q$ be any of the q pairs for which $h = g_1^{w_1} g_2^{w_2} \bmod q$.

Could this be the witness used by the prover?

Yes, if the prover had the witness w_1, w_2 and used randomness $r = (v_1, v_2)$ in the first step with $v_1 = z_1 - ew_1 \bmod q$ and $v_2 = z_2 - ew_2$, then the prover would exactly send
$$g_1^{v_1} g_2^{v_2} = g_1^{z_1 - ew_1} g_2^{z_2 - ew_2} = (g_1^{z_1} g_2^{z_2})(g_1^{w_1} g_2^{w_2})^{-e} = (ah^e)(h)^{-e} = a$$
in the first step, and would send
$$ew_1 + v_1 = ew_1 + (z_1 - ew_1) = z_1$$
and
$$ew_2 + v_2 = ew_2 + (z_2 - ew_2) = z_2$$
in the last step, as observed by the prover.

This means that for each possible witness (w_1, w_2) of the q possible ones, there is exactly one choice of randomness $r = (v_1, v_2)$ consistent with this witness and the observed conversation (h, a, e, z).

4.3.1 Witness Hiding

We now have a computationally binding commitment scheme and a perfect witness indistinguishable Σ-protocol for proving knowledge of an opening. This can be used to conclude that the Σ-protocol is also witness-*hiding*.

Namely, suppose that a commitment $h = \texttt{commit}(w_1, w_2) = g_1^{w_1} g_2^{w_2}$ is given and that the prover knows (w_1, w_2). Assume the Σ-protocol is not witness hiding, and the malicious verifier after seeing one proof for h extracts a valid witness (w_1', w_2') such that $h = \texttt{commit}(w_1', w_2')$. Then by witness indistinguishability, (w_1', w_2') differs from the known one (w_1, w_2) with high probability. This contradicts the computational binding of the commitment scheme as we then have $(w_1, w_2) \neq (w_1', w_2')$ and $\texttt{commit}(w_1, w_2) = \texttt{commit}(w_1', w_2')$.[3]

5 One-Time Signature Scheme based on a Σ-protocols

As a stepping stone towards the electronic cash scheme we present later, we look at a simple construction for efficient *one-time signatures* from Σ-protocols.

Let a hard relation be given, together with an instance generator. Assume there is a corresponding Σ-protocol that is *witness hiding*, i.e., for an adversary it is as hard to compute a witness after the protocol as before.

In the *Key-Generation Phase*, the Signer first selects a random key-pair (x, w) according to the instance-generator. Next, the Signer generates a first message a for a conversation of the protocol. The verification key vk for the signature scheme is $vk = (x, a)$, while the signing key is $sk = (w, r)$, where r is the randomness used to generate a.

In the Signing-Phase, let m be the message to be signed, where m has the same length as the challenges of the Σ-protocol. The signature is then computed by the Signer as a reply z in that protocol, such that x, a, m, z is an accepting conversation.

This scheme is a secure one-time signature in the sense that an adversary, having seen at most one valid signature cannot efficiently come up with a valid signature on a different message.

This is implied by the assumptions as follows. Suppose not, then from the signature output by the signer and the forgery, the adversary would be able to efficiently compute a witness w' for x, by special soundness. But this contradicts the witness hiding property.

An example construction follows directly from the OR-construction based Σ-protocols for hard relations.

Another example is known as Okamoto's protocol. Assume that the commitment scheme from Section 4 has been set up. Let an instance be $h = \texttt{commit}(w_1, w_2) = g_1^{w_1} g_2^{w_2}$ for uniformly random $w_1, w_2 \in \mathbb{Z}_q$. Then, as discussed in Section 4.3.1, the Σ-protocol for proving knowledge of (w_1, w_2) is witness hiding.

[3]It is straight-forward to verify that if $(w_1, w_2) \neq (w_1', w_2')$ and $\texttt{commit}(w_1, w_2) = \texttt{commit}(w_1', w_2')$, then $w_1 \neq w_1'$. So, we indeed we openings to two different messages.

5.1 Relation to Brands' eCash Scheme

In Brands' eCash scheme, which is the one we will look at later, to generate a coin User samples a random key pair (vk, sk) for a one-time signature scheme based on exactly Okamoto's protocol. We use $(\mathrm{sig}^{\mathrm{OT}}, \mathrm{ver}^{\mathrm{OT}})$ to denote the signing algorithm respectively the verification algorithm of this one-time signature scheme. The coin is then the verification key vk. Then User gets Bank's signature $\sigma_B = \mathrm{sig}_{sk_B}(vk)$ on vk, under a normal signature scheme $(\mathrm{sig}, \mathrm{ver})$—Bank does not use a one-time signature scheme, as it must be able to sign many coins. Here sk_B is the secret key of the bank, and the verification key vk_B is known by all participants. Note the (vk, σ_B) can be seen as a certification of vk, except that we of course did not include User's name in the certificate. To pay, User sends (vk, σ_B) and $\sigma = \mathrm{sig}^{\mathrm{OT}}_{sk}(pid)$ to Shop, i.e., User signs the payment identifier pid with the one-time signature scheme. The shop checks that $\mathrm{ver}_{vk_B}(vk, \sigma_B) = \texttt{valid}$ and $\mathrm{ver}^{\mathrm{OT}}_{vk}(pid, \sigma) = \texttt{valid}$. If so, it considers it a valid spending and records $(vk, \sigma_B, pid, \sigma)$. On deposit, Bank does the same check and additionally checks that the name of Shop is encoded in pid and that Shop did not use pid before.

Bank has security against fake coins as only Bank can sign verification keys vk. As we will see Bank gets security against double spending by ensuring that a user with identity U can only get a signature on a verification key vk, where the witness w of the instance (x, w) used to construct $vk = (x, a)$ and $sk = (w, r)$ encodes U. Since double spending is the same a signing two different pid, which leaks the w of the one-time signature scheme, double spenders can be identified by Bank: it simply extracts U from w.

6 Blind Signature Scheme based on a Σ-protocols

Recall that we can also design normal signature schemes from a Σ-protocol for a hard relation R, with security proved in the random oracle model (ROM). The key generator samples a hard instance $(x, w) \in R$. The signing key is $sk = w$ and the verification key is $vk = x$.

To sign message m, compute the first message a in a proof of knowing w such that $(x, w) \in R$, and compute $e = \mathcal{H}(x, a, m)$, where $\mathcal{H}$ is a cryptographic hash function which is modeled as a random oracle in the proof of security. Then compute the reply z to first message a and challenge e. The signature is $\sigma = (a, z)$. To verify a signature (a, z) on a message m under the verification key x, check that $(x, a, \mathcal{H}(x, a, m), z)$ is a valid conversation. Since also x and a are hashed, the signature in in fact be seen as a signature on x, a and m, which we exploit later.

In Brands' eCash scheme the Bank uses such a signature scheme for signing the coin. The hard relation is the proof of equality of discrete logarithms, for which we already saw a Σ-protocol. The instance is of the form $(H, G, h, g) \in G_q^4$ and the witness is $w \in \mathbb{Z}_q$ such that $H = G^w$ and $h = g^w$.

Concretely the signature scheme used by Brands' scheme then works as follows. It uses the same setup (G_q, g_1, g_2) as the commitment scheme in Section 4. A random witness $w \in \mathbb{Z}_q$ has been sampled and $H = G^w$ and $h = g^w$ have been computed, and (G, H, g, h) is known by User and Bank. We later return to who it is that generates these values, for now we just need that User and Bank both know (G, H, g, h) and that

Bank knows w and that User does not know w.

The Σ-protocol looks as follows:

1. The instance is $x = (H, G, h, g) \in G_q$. The witness is $w \in \mathbb{Z}_q$ such that $H = G^w$ and $h = g^w$.
2. The first message a is computed by sampling $v \in \mathbb{Z}_q$ uniformly at random and computing $\bar{H} = G^v$ and $\bar{h} = g^v$. Then $a = (\bar{H}, \bar{h})$.
3. The challenge can be any $e \in \mathbb{Z}_q$.
4. The reply is $z = ew + v \bmod q$.
5. The check is that
$$H^e \bar{H} = G^z$$
and
$$h^e \bar{h} = g^z \ .$$

It is straight forward to verify that this is a Σ-protocol, in particular, it is complete, has special honest-verifier zero-knowledge and special soundness

6.1 Randomizing a Proof

Note that an instance along with a valid conversation for the above equality of discrete logarithm Σ-protocol is a vector of the form

$$(H, G, h, g, \bar{H}, \bar{h}, e, z) \in G_q^6 \times \mathbb{Z}_q^2 \text{ where } H^e \bar{H} = G^z, h^e \bar{h} = g^z \ . \tag{1}$$

There are potentially $|G_q^6 \times \mathbb{Z}_q^2| = q^8$ such values. The values H and G will, however, always be the same. If we fix H and G, then there are potentially q^6 such values. The value h is, however, fixed when g is given, as h should be $h = g^w$ for the instance (H, G, h, g) to be valid, where w is fixed by G and H, as the discrete logarithm of H base G. This brings the potential number of valid conversations down to q^5. Note, however, that once e and z are fixed then $\bar{H} = G^z H^{-e}$ and $\bar{h} = g^z h^{-e}$ are fixed by the validity checks $H^e \bar{H} = G^z$ and $h^e \bar{h} = g^z$. That removes another two degrees of freedom, bringing the total number of valid conversations down to q^3. Put shorter, when H and G are fixed, then for each $(g, e, z) \in G_q \times \mathbb{Z}_q \times \mathbb{Z}_q$, the other components are fixed by the requirement that the instance is a valid instance and the validity checks of the Σ-protocol.

It turns out that given any fixed of these q^3 valid conversations it is possible to efficient compute a new conversation which is uniformly random among the q^3 valid conversations. The trick goes as follows.

First sample $s \in \mathbb{Z}_q$ uniformly at random and compute

$$(H, G, h^s, g^s, \bar{H}, \bar{h}^s, e, z) \ .$$

It is easy to see that this is again a valid instance followed by a valid conversation for that instance. The instance is valid because $h^s = (g^s)^w$ as it should be: $h^s = (g^w)^s = (g^s)^w$, and the conversation is still valid as it is still the case that

$$H^e \bar{H} = G^z$$

and

$$(h^s)^e(\bar{h}^s) = (g^s)^z$$

when $h^e\bar{h} = g^z$. Since g is a generator of G_q, the element g^s is a uniformly random element from G_q. So, now the forth component is random in G_q.

Since the conversation is given by the components (g, e, z) and the g-component is now random, what remains is to randomize e and z. This is done by using the special honest-verifier zero-knowledge simulator. Sample $e' \in \mathbb{Z}_q$ uniformly at random and run the simulator to get a valid conversation for the valid instance $x' = (H, G, h^s, g^s)$ and the challenge e'. This is going to produce a random conversation for x' and e', i.e., a value of the form

$$(H, G, h^s, g^s, \bar{H}', \bar{h}', e', z') \text{ where } H^{e'}\bar{H}' = G^{z'}, (h^s)^{e'}\bar{h}' = (g^s)^{z'} .$$

Recall that the simulator simply picks $z' \in \mathbb{Z}_q$ uniformly at random and computes $\bar{H}' = G^{z'}H^{-e'}$ and $\bar{h}' = (g^s)^{z'}(h^s)^{-e'}$. This means that both e' and z' are uniformly random in $\mathbb{Z}_q$. We use that in a minute, but first we use that two valid conversations for the same instance can be combined into a new valid conversation by simply using the appropriate group operation in each component. In particular, we can combine our valid conversations

$$(H, G, h^s, g^s, \bar{H}, \bar{h}^s, e, z)$$

and

$$(H, G, h^s, g^s, \bar{H}', \bar{h}', e', z')$$

into

$$(H, G, h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}', e + e' \bmod q, z + z' \bmod q) . \tag{2}$$

The validity follows from the validity of the two original conversations. As an example, if we look at the test which has the form $H^e\bar{H} = G^z$ in the canonical form, it now becomes a check whether

$$H^{e+e'}(\bar{H}\bar{H}') = G^{z+z'} ,$$

which is true as

$$G^{z+z'} = G^zG^{z'}$$

and

$$H^{e+e'}(\bar{H}\bar{H}') = (H^e\bar{H})(H^{e'}\bar{H}) = G^zG^{z'} ,$$

where the last step used the validity of the two original conversations. Now the challenge is $e + e' \bmod q$ and the reply is $z + z' \bmod q$. Since e' and z' are uniformly random in $\mathbb{Z}_q$, $e + e' \bmod q$ and $z + z' \bmod q$ are uniformly random in $\mathbb{Z}_q$.

Putting it all together, we now have that $(g^s, e + e' \bmod q, z + z' \bmod q)$ is uniformly random in $G_q \times \mathbb{Z}_q \times \mathbb{Z}_q$, so (2) is a uniformly random valid instance containing (H, G) followed by a uniformly random valid conversation for the instance, i.e., for (H, G, h^s, g^s).

6.2 A Blind Signature Scheme

The ability to perfectly randomize a proof leads to a blind signature scheme for the ROM. We let the signer, who knows w, produce a conversation like the one in (2). We then let the receiver of the this conversation blind it completely, and let the conversation in (2) be the signature. Since it is uniformly random among all valid conversations containing (H, G) it does not leak any information on who received it if all parties use the same (G, H), and they do.

The only trick remaining is that for (2) to be a ROM signature on some message m, we need that the challenge $e'' = e + e' \bmod q$ is of the form

$$e'' = \mathcal{H}(H, G, h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}', m) \ . \tag{3}$$

And, we should ensure this without showing the values $h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}'$ to the signer, or all the trouble we went through to blind them is lost. The crucial observation is that (3) is equivalent to

$$e = \mathcal{H}(H, G, h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}', m) - e' \bmod q \ . \tag{4}$$

The next observation is that the receiver of the signature can actually compute the value e in (4) in time to make it the challenge used when it has the conversation in (1) with the signer. In details the blind signature scheme proceeds as follows:

1. The instance (H, G, h, g) is known by signer and receiver, and the signer knows the witness w.
2. The signer computes $\bar{H}$ and $\bar{h}$ according to the Σ-protocol and sends them to the receiver.
3. The receiver samples uniformly random $s \in \mathbb{Z}_q$, computes $x' = (H, G, h^s, g^s)$ and samples uniformly random $e' \in \mathbb{Z}_q$ and $z' \in \mathbb{Z}_q$ and computes the simulated values $\bar{H}' = G^{z'}H^{-e'}$ and $\bar{h}' = (g^s)^{z'}(h^s)^{-e'}$, and $e = \mathcal{H}(H, G, h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}', m) - e' \bmod q$, and sends e to the signer.
4. The signer returns the reply z to first message $(\bar{H}, \bar{h})$ and challenge e.
5. The receiver takes the signature to by $\sigma = (H, G, h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}', z + z' \bmod q)$.

As already noted, the value σ is a valid conversation for challenge $\mathcal{H}(H, G, h^s, g^s, \bar{H}\bar{H}', \bar{h}^s\bar{h}', m)$, as required to be a ROM signature based on the equality of discrete logarithms Σ-protocol with public key (H, G, h^s, g^s).

6.3 Relation to Brands' eCash Scheme

In Brands' eCash scheme the Bank will use the above signature scheme to blindly sign the coins of the users. The reason that we even randomized part of the public key (H, G, h, g) (it became (H, G, h^s, g^s)) is that Brands' scheme uses a crucial twist, where only G, H is the public key of the bank. The values h^s and g^s are going to be part of the coin, and it is therefore important that also those values are blinded.

7 Brands' eCash Scheme

In this section we describe the main ideas behind an eCash scheme due to Brands. We sketch the protocol construction, while the analysis we provide will most of the time be on an intuitive level.

From a high level point of view, a coin c in Brands' scheme is a verification key vk for the one-time signature scheme $(\mathrm{sig}^{\mathrm{OT}}, \mathrm{ver}^{\mathrm{OT}})$ in Section 5, that is, in the withdrawal, the bank signs such a verification key blindly. The signing key sk serves as the validation key of the coin. Moreover, this secret key is constructed such that the user's identity can be computed from it. Of course, at this point, only the user knows this key, but the blind signature scheme must be constructed such that the user can only get the banks' signature if the user identity is correctly encoded into the validation key. In the following we use vk_B to denote the publicly known verification key of the bank, use sk_B to denote its secret signing key and we use $\sigma_B = \mathrm{sig}_{sk_B}(vk)$ to denote the signature of the Bank on the verification key generated by the user.

To spend a coin $c = vk$, the User signs the payment identifier *pid* sent by the shop (or a hash thereof) w.r.t. the verification key that the coin c represents: User knows sk and sends $\sigma = \mathrm{sig}^{\mathrm{OT}}_{sk}(pid)$ to Shop, along with σ_B. The shop uses σ_B to verify vk and vk to verify σ. By security of the signature scheme $(\mathrm{sig}^{\mathrm{OT}}, \mathrm{ver}^{\mathrm{OT}})$, no one other than the owner of c (not even Bank or Shop) can spend the coin. Since we constructed the coin from a Σ-protocol, in stead of "signing the string *pid*", we could equivalently have said: the user provides the shop with the correct reply in the Σ-protocol to challenge $e = pid$. This is the reason why Accountability-after-the-fact is satisfied, since if the same coin is spent twice, with overwhelming probability, the user will have to reply to a different challenge. On account of special soundness of the Σ-protocol, this will allow the Bank to eventually compute the secret validation key associated to c, and we assumed that knowing this key, the user identity can be computed easily.

The last remaining trick is to exploit a special property of the blind signature scheme in Section 6 and to let it "interact" nicely with the particular one-time signature scheme from Section 5, so that it is guaranteed that a user with identity U can only withdraw coins which encode U.

As it turns out, this blind signature scheme allows User to get one blind signature on a message of the form

$$(g_U^s, m),$$

where $g_U \in G_q$ is common input between User and Bank, and $s \in \mathbb{Z}_q$ and m are inputs given by User and not seen by Bank. For the reason that the first "coordinate" can only be chosen as a known power of g_U, this is sometimes called a *restrictive blind signature scheme.*

7.1 Setup

We now go into the details of the scheme. There is a global setup (G_q, g_1, g_2) as for the commitment scheme in Section 4. No player in the system should be able to compute $\log_{g_2}(g_1)$. In addition, the Bank samples uniformly random $G \in G_q$ and $w \in \mathbb{Z}_q$ and makes makes G and

$$H = G^w$$

public. It keeps w secret, as it will be its signing key sk_B. The value (G, H) is its verification key vk_B.

7.2 User Registration

Let $U \in \mathbb{Z}_q$ represent User's (secret) identity. Define

$$g_U = g_1^U g_2 \ .$$

The value U is picked uniformly at random by User at the time it registers for the eCash scheme and is kept secret. The bank is only given g_U. As part of registration User proves knowledge of $U \in \mathbb{Z}_q$ such that $g_U = g_1^U g_2$. After User registered g_U, the bank computes

$$h_U = g_U^w \ ,$$

and sends h_U to User. Then User stores (h_U, g_U) and U. The bank records that this particular user registered under the public identifier g_U.

7.3 Withdrawal

In each withdrawal, User and Bank use the instance.

$$X = (H, G, h_U, g_U) \ .$$

Note the $H = G^w$ and $h_U = g_U^w$, so X is actually a valid instance for the equality of discrete logarithms relation, and Bank knows the witness w.

Now User and Bank run the blind signature scheme from Section 4. Recall that User in Step 3 in that protocol selects uniformly random $s \in \mathbb{Z}_q$, and that the signature, when the instance is (H, G, h_U, g_U), ends up containing the value g_U^s. Note that

$$g_U^s = (g_1^U g_2)^s = g_1^{Us} g_2^s \ .$$

If we let User compute $com = g_U^s$ and $w_1 = Us \bmod q$ and $w_2 = s$, then User already in Step 3 knows a witness (w_1, w_2) such that

$$com = g_1^{w_1} g_2^{w_2} \ .$$

This is almost a key pair for the one-time signature scheme $(\mathrm{sig}^{\mathrm{OT}}, \mathrm{ver}^{\mathrm{OT}})$ from Section 5. What is missing it the first message a in a proof that User knows a witness (w_1, w_2) such that $com = g_1^{w_1} g_2^{w_2}$. Users knows such a witness and can therefore compute a. Then User lets the message m in Step 3 in the blind signature scheme be $m = a$ and completes the execution of the blind signature algorithm with Bank. This gives User a blind signature σ_B from the bank—it is the value in Step 5 in the blind signature scheme. Notice that both $g_U^s = com$ and $m = a$ are part of the messages being hashed, so σ_B is in particular a signature on both com and a.

Now User lets $vk = (com, a)$ and $sk = (w_1, w_2)$, and thus has a key pair (vk, sk) for $(\mathrm{sig}^{\mathrm{OT}}, \mathrm{ver}^{\mathrm{OT}})$ along with a blind signature σ_B from Bank on vk.

After the execution of the blind signature scheme with User, Bank withdraws one unit from the account of User.

7.4 Spending

To spend the coin, Shop sends a payment identifier *pid* to User, where *pid* encodes the identity of Shop plus a unique transaction identifier, either a counter or a nonce. Then User computes $\sigma = \mathrm{sig}^{\mathrm{OT}}_{sk}(pid)$ and sends (vk, σ_B, σ) to Shop.

Shop checks that $\mathrm{ver}_{vk_B}(vk, \sigma_B) = \texttt{valid}$ and that $\mathrm{ver}^{\mathrm{OT}}_{vk}(pid, \sigma) = \texttt{valid}$. If so, it hands out the goods and stores $(vk, \sigma_B, pid, \sigma)$.

7.5 Deposit

To deposit a coin, Shop simply sends $(vk, \sigma_B, pid, \sigma)$ to Bank. Bank checks that *pid* encodes the identity of Shop and that Shop did not deposit a coin with this *pid* earlier. If so, Bank checks that $\mathrm{ver}_{vk_B}(vk, \sigma_B) = \texttt{valid}$ and that $\mathrm{ver}^{\mathrm{OT}}_{vk}(pid, \sigma) = \texttt{valid}$. If so, Bank adds one unit to the account of Shop, and records $(vk, \sigma_B, pid, \sigma)$.

7.6 Security

The bank has security against fake coins as the only way to generate a signature on some new *vk* is to engage in the blind signature protocol, which means that Bank gets one unit per *vk* signed, and each deposited coin contains a signed *vk*. It actually takes a care analysis in the ROM to see this, but we will not dive into the details.

As to accountability-after-the-fact, from a double-spent coin $vk = (com, a)$ the bank can use special soundness to compute a witness (w'_1, w'_2) such that $com = g_1^{w'_1} g_2^{w'_2}$. Since in the execution of the blind signature scheme the bank uses (H, G, h_U, g_U), it holds under the assumption that the users can only randomize proofs by replacing (h_U, g_U) by (h_U^s, g_U^s) for some known s, that $com = g_U^s$. So,

$$com = g_U^s = (g_1^U g_2)^s = g_1^{Us} g_2^s$$

and

$$com = g_1^{w'_1} g_2^{w'_2} \ ,$$

where Bank knows the opening (w'_1, w'_2) of *com*. Since User by knows s and proved knowledge of U at registration time, it follows that User knows the opening $(w_1, w_2) = (Us \bmod s, s)$ of *com*. By the computational binding of *com* it then follows that $(w'_1, w'_2) = (w_1, w_2)$, except with negligible probability.[4] Therefore Bank can compute

$$w'_1 (w'_2)^{-1} \bmod q = w_1 (w_2)^{-1} \bmod q = Uss^{-1} \bmod q = U \ .$$

It can then compute $g_U = g_1^U g_2$ and look up which user registered g_U. This is the double spender.

It is here important that the only way a user can randomize (h_U, g_U) is by replacing it by (h_U^s, g_U^s) for some known s, since otherwise User could avoid encoding of his identity into a coin. The assumption that User can only randomize (h_U, g_U) by replacing it by (h_U^s, g_U^s) for some known s is closely related to an assumption known as the knowledge of exponent assumption. We will not look further into the details of this.

[4] Or User and Bank together would have efficiently computed a double opening with non-negligible probability.

The security of Shop is clear as Bank does the same checks as Shop, except that it checks that Shop remember to put its identifier in *pid* and that it remember to use a new *pid* for each transaction.

The user anonymity follows from the signature being blind. In fact, the generated signature is uniformly random among all possible signatures as we argued in Section 6. This means that the anonymity is perfect, even an infinitely powerful Bank cannot trace the deposited coin back to the user.

Nobody but User can double spend a coin in the name of User, as being able to double spend in the name of U implies that one can efficiently compute U, and that would imply computing the uniformly random U from $g_U = g_1^U g_2$, which means that you can compute U from $g_U g_2^{-1} = g_1^U$, which is a discrete logarithm problem in G_q, which we have assumed hard.

8 Additional User Security

When defining user security we required that only the user U can withdraw money from $account_U$. If we want this to hold even against the bank B further techniques are needed — as of now the bank can simply withdraw money on $account_U$ and claim that U was given the corresponding coins. The user U has no way to prove that this is not the case.

To prevent this we could let U sends along with each request a signature $\sigma_W = sig_{sk_U}(\texttt{withdraw } den \texttt{ from my account} \| wid)$, where wid is a unique withdrawal identity. If then a dispute occurs on whether a withdrawal on the account $account_U$ was requested by U, we would require from the bank that it can show us a corresponding signature σ_W as proof.

Notice that there is still the possibility that after the bank gets σ_W it does not complete the Withdrawal protocol. If we solve this by requiring that U does not send σ_W until it received a valid coin, then B has the problem that U might not send σ_W. There exists techniques which allows to solve this dispute of who is to go first, known as *fair exchange* or *gradual release.*

9 Sender Anonymity

One generic objection against anonymous payment systems is the following: suppose I use an electronic coin to pay from my home PC. Then, even though the coin itself cannot be traced to me, the fact that the message it is contained in comes from a particular IP address may mean that it can still be linked to my identity, and/or to a particular withdrawal transaction.

It is therefore of interest to look at, whether it is possible to send messages in such a way that it is impossible to tell who sent the message. There are in fact practical solutions around, known as *Crowds* and *Onion Routing* (lots of info on these methods can be found on the Internet). Here, a message is sent around (using various forms of encryption) between members of a certain large group of computers, before it finally goes to the receiver. This guarantees anonymity as long as not too many members of the group are malicious, but of course only w.r.t. the group, i.e., we know the message came from the group, but not from which member.

There is also a theoretical solution that guarantees unconditional sender anonymity. This was proposed by David Chaum, and is known as the *Dining Cryptographers' Problem.* The following exercise is about this:

Exercise 4 Three cryptographers meet for dinner at an expensive restaurant. The waiter tells them that someone who wants to remain anonymous has already paid for the dinner. The cryptographers decide that they would like to find out if one among themselves paid, but they want to respect this person's anonymity. They execute the following protocol:

1. Call the cryptographers A, B and C. Now, A and B flip a fair coin such that only they can see the outcome - a bit b_{AB}. In similar way, random bits b_{AC}, b_{BC} are chosen.

2. Now, every cryptographer has seen the outcome of exactly two coins. Then, each of them announce one bit. For each cryptographer, the announced bit is the XOR of the two coin-flips he has seen if he did not pay for dinner, and it is the complement of the XOR if he did pay. So for instance, A announces either $b_{AB} \oplus b_{AC}$ or $1 \oplus b_{AB} \oplus b_{AC}$.

Show that if all cryptographers follow the instructions, and at most one of them paid for dinner, then one can decide from the announced bits if one of them paid or not. Show that if the result is that one of them paid, then a non-paying cryptographer learns no information on which of his two friends paid.

Sketch how this can be generalized to a protocol for any number of parties where a message (say, of fixed length for simplicity) can be sent anonymously, assuming that only one party wants to send at any given time.

10 Notes and a Brief Historical Overview

The concept of *anonymous electronic cash* was invented by David Chaum [3, 4] in the early 80s. The first examples of such schemes used Chaum's RSA-based *blind signature scheme*, that we presented in the lecture.

The importance of these seminal works is partly explained by the fact that they have brought *anonymity* onto the cryptographic scene, next to *authenticity* and *secrecy*. This significantly expanded the scope of the field.

The principle of *accountability-after-the-fact* was proposed in a paper by David Chaum, Amos Fiat and Moni Naor [5] in 1988.

In the late 80s, Chaum invented the concept of *wallets with observers*, to ensure cryptographic protection of privacy while at the same time preventing misuse of electronic tokens, i.e., double-spending.

Very briefly, a wallet with observer is a combination of two devices: one tamper-resistant processing unit (the observer), such as a smart-card, which operates on behalf of the Bank, and an arbitrary processor under complete control of the User (the wallet). The observer is embedded into the wallet in such a way that it can only communicate with the outside world through the wallet.

On an intuitive level, the main idea is that the observer prevents the User from double-spending electronic money, thereby safeguarding the interests of the Bank, while the wallet takes care of the User's privacy by making sure that no sensitive information about his transactions can leak over the channel between observer and outside world.

Technically, the role of the User in the electronic cash schemes such as the one we discussed in the lecture, is split up between the wallet and the observer in a specialized "two-party secure computation".

This is organized in a such a way that the wallet does not obtain validation keys and can only spent a coin in cooperation with the observer. Once used, the observer simply refuses to engage in subsequent payments with the same coin. In the event an observer is broken into, validation keys may be become available to the attacker. In this case accountability-after-the-fact provides a fall-back mechanism.

The wallet can be seen as a "randomizer" that enforces honest execution of withdrawal and payment protocol, so that privacy is guaranteed. The technical challenge in the design of such systems is in achieving those properties simultaneously.

A very nice overview of this concept, as well as other privacy protection mechanisms proposed by Chaum, such as *electronic credential systems*, is given in his article in Scientific American [6].

Electronic cash was first studied in a theoretical framework by Ivan Damgård, who showed [9] the plausibility of such systems assuming the existence of general cryptographic functions.

David Chaum and Torben Pedersen [7] presented efficient discrete logarithm based protocols for wallets with observers. The discrete logarithm based blind signature scheme we presented in the lecture was proposed in this paper.

Ronald Cramer and Torben Pedersen [8] strengthened the notion of privacy by requiring that it is impossible for the observer to collect information that is sensitive from a privacy point of view, and adapted the protocols from [7] to satisfy this stronger notion. However, the systems from [7, 8] do not provide the important property of accountability-after-the-fact.

This problem was solved by Stefan Brands [1], who devised electronic coins based on the discrete logarithm problem where the User identity is elegantly coded based on his idea of *restrictive blind signatures*. The scheme we presented in the lecture is due to Brands.

The schemes proposed in [1] facilitate easier security analysis and are much more efficient than earlier RSA-based proposals, [5] and in essence represent the state of the art in privacy protecting electronic cash today.

A different extension of the electronic cash idea is to make schemes with so called *revocable anonymity*. These are schemes where anonymity is not unconditional, but can be broken and coins be traced, if a certain secret key is known. This key can then be secret shared between a number of trustees, that would all have to agree in order to revoke anonymity. Such schemes can be designed such that only a single coin (and not all that have been issued) becomes traceable, and can be as practical as the schemes we have seen here. A lot of work on this concept has been done by Jan Camenisch[2].

[5] Improvements have been proposed by Niels Ferguson [10], who found a more efficient solution for accountability-after-the-fact in the case of RSA-based schemes.

11 Acknowledgments

Thanks to Martin Eriksen for comments.

References

[1] Stefan Brands. Untraceable electronic cash in wallets with observers. Proc. CRYPTO '93.

[2] Jan Camenisch. Group Signature Schemes and Payment Systems based on the Discrete Logarithm Problem. PhD thesis, ETH Zurich, 1998.

[3] David Chaum. Blinding signatures for untraceable payments. Proc. CRYPTO '82.

[4] David Chaum. Security without identification: transaction systems to make Big Brother obsolete. Communications of the ACM, 28 (1985).

[5] David Chaum, Amos Fiat, Moni Naor. Untraceable electronic cash. Proc. CRYPTO '88.

[6] David Chaum. Achieving electronic privacy. Scientific American, August 1992.

[7] David Chaum, Torben Pedersen. Wallet databases with observers. Proc. CRYPTO '92.

[8] Ronald Cramer, Torben Pedersen. Improved privacy in wallets with observers. Proc. EUROCRYPT '93.

[9] Ivan Damgaard. Payment systems and credential mechanisms with provable security against abuse by individuals. Proc. CRYPTO '88.

[10] Niels Ferguson. Single-term off-line coins. Proc. EUROCRYPT '93.