HILDESHEIMER INFORMATIK-BERICHTE

ISSN 0941-3014

Birgit Pfitzmann
Michael Waidner

Strong Loss Tolerance for Untraceable Electronic Coin Systems

15/95 (Juni 1995)

UNIVERSITÄT HILDESHEIM

Dieser Bericht ist herausgegeben vom

Institut für Informatik

Postfach 10 13 63
31113 Hildesheim

Strong Loss Tolerance for Untraceable Electronic Coin Systems

Birgit Pfitzmann

Universität Hildesheim
Institut für Informatik
Marienburger Platz 22
D-31141 Hildesheim, Germany

☎ ++49-5121-883739
Fax ++49-5121-883732
pfitzb@informatik.uni-hildesheim.de

Michael Waidner*

IBM Research Division
Zürich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon, Switzerland

☎ ++41-1-724-8220
Fax ++41-1-710-3608
wmi@zurich.ibm.com

* work mainly done at the University of Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz.

4. 6. 1995

Abstract: Untraceable electronic cash means prepaid digital payment systems, usually with off-line payments, that protect user privacy. They have recently received great attention in both theory and development projects. However, with most current schemes, loss of a user device containing electronic cash implies a loss of money, just as with real cash. Compared with credit schemes, this is considered a serious disadvantage.

We presented basic ideas for loss tolerant untraceable electronic cash in 1990, and mainly applied them to systems based on balances in tamper-resistant devices. We now show how they can be applied to the more advanced systems based on the concept of electronic coins. Security against fraud and privacy are preserved by these measures. The main new developments in this paper are a stronger form of loss tolerance (not relying on the bank's goodwill) and an improvement in privacy, so that almost the optimal conceivable combination of security, privacy, and loss tolerance is achieved. We present general design principles and options and their instantiation in one concrete protocol suite. The measures are practical.

1 Introduction

So-called electronic cash systems have received great attention in the last few years. A frequently heard brief description is that they are just like paper cash, only easier to handle and potentially more secure. In fact, they are intended for similar applications: the mass market of low-value payments, e.g., for daily shopping, public transport, phone calls, and sometimes similar transactions over networks. Nevertheless, the analogy to paper cash can be quite confusing, since all electronic cash systems proposed so far have some properties seriously different from paper cash; moreover, many different types of systems are used under this or similar names. Hence, we start by describing properties of electronic cash in Section 1.1, and in Section 1.2, we sketch the construction principles of electronic coin systems. Section 1.3 describes the goals of loss tolerance, and Section 1.4 gives an overview of the rest of the paper.

1.1 Properties of Untraceable Electronic Cash

The main participants in any electronic payment systems are payers, payees, and issuers of electronic money. The latter will briefly be called banks. More precisely, we use these terms for human users in certain roles. The device of a payer is called an *electronic purse* and that of a payee an *electronic till*. More precisely, electronic purses and tills are logical entities, and several of them can be implemented in the same physical device. Traditionally, an electronic purse corresponds to a smartcard, and an electronic till to a point-of-sales terminal. However, both for functionality and security, more advanced electronic purses with a small user interface and a battery of their own are useful. For example, a functional advantage is that users can always check how much electronic cash they still have and what their previous transactions were. Two security advantages are that fake-terminal attacks are prevented since nobody has to enter a PIN into another person's device, and that the amount to be paid is shown on the display of the electronic purse so that the payer can explicitly agree to it or reject it. Examples are super smart cards [TAMc_89], so-called electronic wallets looking like pocket-calculators [Rolf_94, BBCM1_94], or combinations with more functionality such as mobile telephones or personal digital assistants.

There are three basic types of transactions:

- *withdrawal*, where a payer receives electronic money into an electronic purse in exchange for real money, usually from a bank account,
- *payment*, where a payer passes electronic money to a payee, and
- *deposit*, where the payee exchanges the electronic money for real money again, usually on a bank account.

A payment system where electronic money received in a payment can be used for another payment without intermediate deposit is said to offer *transferability*. In contrast to paper cash, most electronic cash systems do not offer transferability.

If there are several banks, *clearing* transactions are needed.

A payment system is said to be *offline* if the payment transaction only involves the electronic purse and the electronic till, i.e., without online verification by the bank. In practice, this is an important requirement in the near future (except for payments for services over networks), since communication for each payment is still quite costly and almost impossible in public transport.

Payment systems should offer *multilateral security*, i.e., security requirements of all parties have to be considered. A rough description of these requirements is that no party wants to lose money. The security of a party X should rely as little as possible on the dependability of other parties; it does depend on the devices controlled by X and the software in these devices, of course. Proposals for payments systems in the scientific literature usually aim at multilateral security in this sense. From a theoretical perspective, payment systems are therefore a special case of fault-tolerant distributed computing with Byzantine faults. In contrast, many electronic payment systems currently in use require everyone to trust the bank and thus, given current operating systems, many of its employees, suppliers and maintenance personal. In less abstract terms, “trust” in such a context means that fraud by bank insiders or programmers (or even outsiders successfully attacking bank equipment) and fraud by customers is indistinguishable. Disquieting recent court cases resulting from this situation can be found in [Ande3_94].

Payments with real cash are de facto *untraceable*, i.e., a payer does not have to identify himself to the payee in a payment, and the bank cannot trace who paid to whom. Therefore, untraceability is considered a necessary property of electronic cash, too.[1,2]

1.2 Principles of Coin-Based Constructions

So far, we sketched the specification of what an electronic cash system should achieve. Actual constructions of off-line untraceable electronic cash systems essentially fall into two classes:

- classical *balance-based* systems, i.e., with balances handled by tamper-resistant devices, following [EvGY_83],
- and the newer class of *coin-based* systems (see below).

In the following, we concentrate on coin-based systems. One reason is that we described loss tolerance measures for the first class in detail earlier [WaPf_90] (with even more details in grey literature, [WaP1_87]).

To see why coin-based systems are of increasing interest, note that tamper-resistance is only a relative notion: Although there is no unclassified literature on the subject, it is widely believed that criminals with sufficient resources and knowledge and willing to spend a certain amount of time per device can break the tamper-resistance of smartcards and similar devices. Electronic cash systems of the first class, however, rely heavily on tamper-resistance: Electronic money is simply represented by balances in the tamper-resistant devices, and payments are performed by decreasing the balance of the purse and increasing the balance of the till quasi-simultaneously. If tamper-resistance is broken, balances can be increased fraudulently. The bank can at most notice this afterwards, and it cannot find out how the fraud happened unless a major part of the transactions is traced and sorted by identities, i.e., not in the untraceable versions of this class of systems.

This problem with tamper-resistance is minimized by coin-based systems like [ChFN_90]. Here, untraceable electronic cash is represented by a set of individual *electronic coins* stored in the purse. An electronic coin is information of a certain form, *digitally signed* by the coin issuer. Its validity can therefore be verified both by tills in payments and by the bank in deposits. Thus users cannot forge new coins, independent of the tamper-resistance of their devices, provided a secure digital signature scheme is used.[3] Different denominations can be distinguished by different keys used for signing. To make the system untraceable, the electronic coins are signed

---

1 Where deemed necessary, upper limits on the amount and frequency of such payments can be set or possibilities for tracing under specific circumstances offered. Political discussions are, of course, outside the scope of this paper. However, whatever we personally think about this, loss tolerance measures can be used for conditional tracing.

2 Untraceability for *payees* is not required in most cases: In systems without transferability, the payee is usually a retailer or a provider of an online information service, i.e., a well-known party.

with a *blind signature scheme* (introduced in [Cha8_85]). “Blind” means that the purse receives a signed coin, but the coin issuer does not know what this coin looks like. Hence, when a payee deposits a coin, the bank cannot link it to the payer who withdrew it.

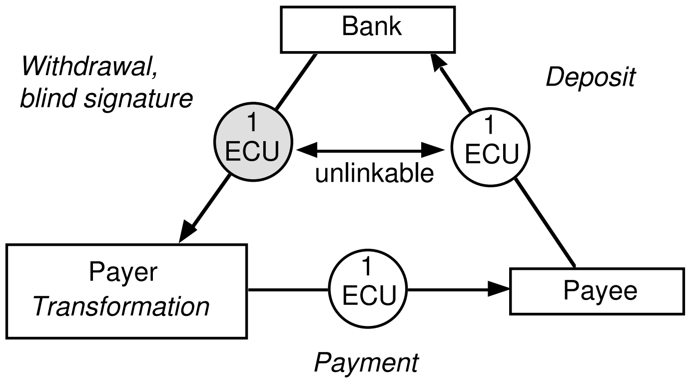

Figure 1 Life-cycle of an electronic coin.
Note that only the transformed (white) form of a coin is acceptable in payments and deposits. Furthermore, all transactions may be interactive.

Of course, this leaves the problem that a payer might try to spend the same coin twice, since it is only signed digital information and can be copied. There are essentially four solutions (all covered by our loss tolerance measures):

1. *On-line verification* was assumed in all early proposals for untraceable coin-based systems, e.g., [Chau_83, Cha8_85, Damg1_90, PfWa2_92]. As mentioned above, this is currently considered unacceptable for normal low-value payments. However, it is reasonable in payments for services in networks, e.g., with Digicash’s ecash [http://www.digicash.com/ecash/ecash-home.html], and may be advisable for all payments above a certain threshold.
2. *Identification of double-spenders* by coding identities into coins such that anyone who spends a coin twice can be identified. The surprising fact that this is possible while preserving the untraceability of honest payers was first shown in [ChFN_90]. For some of the further theoretical and practical developments, see [FrYu_93, Bran_94, Ferg_94]. The basic technical idea, *restrictive blind signatures*, is explained in Section 3.3.
3. *Combination with tamper-resistance*: Identification of double-spenders alone may be unacceptable to banks, since a bank may nevertheless be unable to recover the money. Thus it is natural to combine it with tamper-resistant electronic purses. This does not change the cryptographic protocols, but gives the banks the optimal level of security achievable with off-line payments:

---

[3] The dependence on cryptography is not a special disadvantage of coin-based systems, since balance-based systems also need cryptography for mutual authentication between the tamper-resistant devices. Of course, simple balance-based systems only use symmetric authentication.

- no possibility of fraud as long as the tamper-resistance is unbroken
- and identification of double-spenders otherwise

in combination with untraceability for honest participants.

4. *Wallets with observers*: The electronic purse is separated into one part (called wallet) owned and controlled by its user and another part (called observer) that is owned by the bank and tamper-resistant against the user. The protocols are changed so that only both parts together can spend electronic coins, and the part owned by the bank cannot breach privacy, see, e.g., [Chau_92, ChPe1_93, CrPe_94]. A security advantage is that the bank need not publish the design of the tamper-resistant part, but the customers need not trust a device whose design has not been publicly verified. A functional advantage is that the user-owned, outer part can come from various manufacturers, with many different designs and additional functionality. The observer may be a standard smartcard that can be inserted into all these devices. A system with some aspects of this solution will be submitted to a field trial by the EU project CAFE at the end of 1995 [BBCM1_94].

1.3 Loss Tolerance: Requirements and Fault Model

Electronic cash systems are prepaid systems: payers have actual values in their electronic purses. Hence, in the absence of additional measures, the payers lose the money if a purse is lost, stolen, broken, or stops working. *Loss tolerance* means that the customer gets the money back. Note that this is a property that paper cash does not have. This problem was investigated in [WaPf_90], and the basic solution principles of distributed recovery were mainly worked out for balance-based systems with transferability. In the following, we sketch how the same principles work with coin-based electronic cash systems without transferability, and then derive detailed protocols with particularly high loss tolerance and privacy.[4] Note that the customers will not get the electronic coins back that were lost with an electronic purse, but the equivalent value, usually on a bank account.

We only consider loss tolerance measures for electronic purses, not for electronic tills and the central devices at the bank. The reason is that those devices are stationary and not so limited in size as electronic purses. Hence standard fault tolerance measures can and should be applied to them (see, e.g., [LeAn_90]), such as regular backups or even 2-out-of-2 or 2-out-of-3 systems for the most important components of the bank's devices. In contrast, loss of electronic purses must be tolerated by specialized measures considering the whole distributed system. Moreover, backups of electronic tills do not pose a security risk for the banks: Electronic tills only *collect* electronic coins. Hence, if a payee decides to use a backup instead of the current state of his till, he can only lose coins. (Of course, the bank has to prevent double-depositing of the same received coins, but this is done in any case.) In contrast, electronic purses *spend* coins, hence a backup would usually contain more money than the current state. This is related to the reasons for making

[4] Loss tolerance measures derived from the same basic ideas from [WaPf_90] will be an important feature of the CAFE system mentioned in Section 1.2, see [BBCM1_94], and will thus be submitted to a field trial. In comparison, CAFE in general and its loss tolerance measures in particular concentrate on efficiency improvements (and on all the small snags between ideas and a real system), in contrast to the extended security considered here.

electronic purses tamper-resistant (see Section 1.2, Items 3 and 4), and in those cases no backups are possible in the first place that have not been explicitly provided for in the protocols.

The basic requirements on a loss tolerance scheme for untraceable electronic cash are:

1. *Security of the bank*: Even if arbitrarily many customers cheat and "lose" their devices according to an optimal strategy, they cannot obtain altogether more money than they possessed. (Note that this requirement has to comprise the situation that payees "lose" their tills.)
2. *Security of customers who do not lose their devices*: Nobody should lose money by loss tolerance measures for other payers, nor by provisions for loss tolerance for their own purse as long as this purse does not get lost.
3. *Loss tolerance*: The owner of a lost electronic purse gets the money stored in the purse back after a while. An upper bound on this period of time must be known.
   a) We speak of *strong loss tolerance* if this is guaranteed in the same sense of multilateral security as the other requirements, i.e., if it does not rely on the dependability of other parties.
   b) We sometimes mention *weak loss tolerance*, which means that loss tolerance is only guaranteed if "the bank" is honest, i.e., if bank insiders could embezzle lost money, but are assumed to not to do so.

      If only weak loss tolerance is required, one has to require *security of owners of lost purses against bank insiders* separately (whereas strong loss tolerance automatically implies this type of security): Someone who loses a purse should not lose more money by the loss tolerance measures than he had in this purse, independent of the trustworthiness of the bank.
4. *Untraceability of payers who do not lose their purses*: The loss tolerance measures should not impair the privacy of people who do not lose their purses.

   Slightly weaker versions of this property will also be considered. For instance, some loss tolerance measures reduce the untraceability from information-theoretic (i.e., absolute as far as the digital part of the system is concerned) to computational (i.e., relying on the fact that an adversary does not have arbitrary computational resources). Others make payers a bit more vulnerable to physical attacks on untraceability, such as stealing information out of their homes.
5. *Partial untraceability of owners of lost purses*: For technical reasons, we will allow some payments by owners of lost purses to be traced. However, this tracing should be restricted to payments around the time of the loss. For instance, a requirement that can often be fulfilled is that only the payments between the last withdrawal and the loss become traceable.

Since Requirements 4 and 5 are somewhat flexible, our theorems will specify the precise versions that our protocols fulfill. Recall that all these requirements were for nontransferable electronic cash, where received money can only be deposited. Extensions to transferability are mentioned in Section 7.6; some requirements must be weakened in that case: In particular, it is impossible to correctly consider a payment between two purses that were both lost after this payment and before interacting with other devices again.

We model the loss of an electronic purse as a *crash failure*, i.e., the purse behaves correctly until the loss and then stops interacting with its environment completely. This benign fault model is not in contradiction with the multilateral security against Byzantine, i.e., malicious, adversaries that we advocated in Section 1.1: The crash failure assumption is only made for a person's *own* purse when the security of *this* person is considered (i.e., in Requirements 3 and 5). Nevertheless, the purse design should contain standard fault tolerance measures, such as self-tests, to justify this fault model.

To simplify the presentation, we will first make the unrealistic assumption that losses only occur between transactions. Extensions to losses (usually internal failures) during a transaction, i.e., a withdrawal or a payment, are considered in Section 7.5. This means that the transactions, which so far got their name from business transactions, become transactions in the sense of computer science, in particular atomic. We call this interruption tolerance.

1.4 Organization of this Paper

Section 2 contains a high-level description of our loss tolerance measures with several options. The choice among these options cannot be made once and for all by theoretical quality criteria, but has to be based on the infrastructure available in concrete applications and partly on the wishes of the individual users. In Section 3, we describe one particular electronic coin system in more detail. In Section 4, we show concrete instantiations of the loss tolerance measures in this electronic coin system and demonstrate their security. In Section 5, we explain some further organizational details. In Section 6, we show that the loss tolerance measures do not reduce the efficiency of the underlying systems significantly. In Section 7, we show how the loss tolerance measures can be extended to several variants of the payment systems.

2 High-Level Description

Loss tolerance for electronic purses has to be based on distributed recovery [WaPf_90]: Loss or theft of electronic purses cannot be prevented by internal fault-tolerance measures, and external measures like 2-out-of-3 purse systems seem impractical, apart from the fact that one would usually lose all the purses at the same time. For coin-based systems without transferability, the distributed recovery can be summarized as in following overview.

2.1 Overview

From time to time, usually after withdrawals, one has to make a backup of the information stored in a purse, as far as it represents electronic cash. In coin-based systems based on blind signatures, this *backup information* essentially consists of the signed coins in the form after the transformation (the white form in Figure 1, plus, at least for strong loss tolerance, some secret information that allows a particular user to reclaim the coin). Since this information is sensitive from the point of view of privacy, it must either remain with the user or be in encrypted form. The former means that the user needs a second device, the latter implies a need for key management.

After a loss, the owner of the purse contacts the bank, and they carry out a protocol we call RECLAIMING. The bank is needed in recovery to prevent cheating users from using the backup information although they have not lost their purse, and thus from spending money twice. The owner of the purse is needed to either provide the backup information at all or help decrypting it. Because of Requirement 5, it must be ensured that no unnecessary, out-of-date backup information is divulged.

In most cases, the user has already spent some of the coins that are included in the backup information. The only way to find this out is from the electronic tills. Hence, essentially, the user will get all the money back that nobody else deposits. We call the corresponding protocol parts SETTLING (and reserve the word "recovery" for the whole process of reclaiming and settling). One problem with settling is that the payer's claim that the purse is lost may be wrong, and thus the payer may still try to spend the coins *after* the execution of RECLAIMING. Furthermore, if thieves or dishonest finders can simply spend the electronic cash from stolen purses, loss tolerance cannot work as desired; hence the purses must be able to identify their users. These topics are discussed in more detail in Sections 2.4 and 2.5.

2.2 Storing Backup Information

Backup information can be stored in two different places:

- On a backup device owned by the purse holder, e.g., a smartcard. We call such a device a *backup purse*.
- On a device of another party. The most natural choice of such a party is the bank.

A user may also have more than one backup purse and keep them in different places, or both options can be offered simultaneously.

Backup information on backup purses. If a backup purse is used, backup information should be stored at the explicit request of the user, i.e., there is a separate transaction BACKUP. For this, the user has to connect the electronic purse and the backup purse. The cheapest solution with current infrastructure is that the electronic purse has an internal smartcard reader and the backup purse is a memory card. In future situations where users can be expected to own more than one mobile device with infrared or radio connections, these devices can contain mutual backups. Reasonable moments for backups are as soon as possible after withdrawals. Less concerned users may only do backups occasionally, e.g., after withdrawing a particularly high amount of electronic cash or before going to a place they consider dangerous.

The simplest form of backup is to delete the old backup information and produce a completely new one each time. However, communication between electronic purses and backup purses may be quite slow with current technology. Thus incremental backups may be preferable, since most users will not spend all their coins between two withdrawals. It is then useful for both efficiency and privacy not just to store new coins, but also to delete (or tag as obsolete) backup information about spent ones. In this case, individual coins have to be addressable in the backup information.

The easiest method is that the electronic purse gives the backups of individual coins sequence numbers.

Note that a user cannot cheat by deleting more or fewer coins than the spent ones, even if we do not assume the backup purse to be tamper-resistant: If too many coins are deleted, they cannot be claimed back, and if too many coins are retained, it will turn out that they have already been deposited and are therefore not given back in SETTLING.

Backup information at the bank. If the backup information is stored at the bank, it should be stored there during each withdrawal automatically (unless loss tolerance is an option that costs extra — then the purse should ask its owner during the withdrawal if this option is chosen). This solution has the following advantages and disadvantages, compared with backup purses:

+ The backup information can be stored transparently, i.e., the user need not be involved.
+ Backup information is produced as soon as possible: Whenever electronic cash is withdrawn, there is a backup for it, too.
– Since the backup information has to be stored in encrypted form, the user must store the key somewhere (and not on the purse, since the key and the purse should not be lost together). This organizational issue is discussed in Section 5.2.
– En- and decryption make incremental backups more complicated. Furthermore, it cannot be assumed for privacy that the bank really deletes old backup information. Hence the simple method to send the decryption key to the bank after a loss would violate Requirement 5 from Section 1.3. We discuss the possibilities in more detail in Section 4.4.

2.3 Reclaiming

After a loss, some organizational measures are needed before the technical protocol RECLAIMING can start: First, the user has to retrieve a backup purse or the decryption key, respectively. When the user is in a foreign country and left the backup purse or the key safely at home, this may be inconvenient; passphrases as decryption keys (see Section 5.2) are the only option without this problem. Secondly, the identity of a user who requests a recovery has to be verified; we will also assume that he signs that he wants a recovery. If the lost electronic purse was the user's only powerful mobile device, this cannot be done electronically. Thirdly, we sometimes have to assume that the user procures a new electronic purse or another device that can carry out protocols. (Of course, if backup purses are powerful enough, they can play this role; we will not mention this possibility again in the following.) Users who are fed up with such devices can leave everything to a third party, usually the bank, but then they lose some security and privacy with respect to this party.

Reclaiming with backup purses. After the organizational measures, the user connects the backup purse and the new electronic purse, so that the new purse can take over the protocol. Since old backup information was deleted or tagged as obsolete during backups, the new purse only sends the bank information about coins that were still unspent at the last backup (possibly with additional steps to guarantee strong loss tolerance). Hence Requirement 5, restricted untraceability for the owners of lost purses, is quite easy to fulfil. One may even increase the untraceability now (only as an option, since it will be too complicated for most users) by letting the new purse ask the user which coins should be reclaimed, so that he can exclude coins that he remembers or suspects to have spent. This presupposes purses with user interfaces, which was useful anyway. The user should be shown the denomination and the date and time (or the relative order) of the withdrawal of the coins; this is the only coin-identifying information users can be expected to remember. This yields perfect untraceability for users who kept precise records of which coins they have already spent.

Reclaiming with backup information at the bank. If the backup information is stored at the bank, a key to this information must be given to the bank, or, usually easier[5], the information must be sent to the new purse, decrypted there, and sent back (possibly with additional steps to guarantee strong loss tolerance). Details are discussed in Section 4.4.

2.4 Settling

Essentially, a consistent state after recovery is achieved by giving only those coins back to the owners of lost purses that have not been deposited. Protocol details are described in Section 4.2. As to the timing, it is not clear that all these deposits have already been made when the loss is reported. Moreover, a cheating user can falsely claim that the purse is lost and continue to try spending the original electronic cash on the purse for quite some time. This is not trivial to prevent since payments are offline[6]. Thus we have to deal with two issues:

1. *Expiry of lost electronic cash:* It has to be guaranteed that from some time onward, electronic cash that was claimed to be lost can no longer be spent or deposited — this is a protocol and synchronization issue.
2. *Preliminary recovery:* One has to decide what happens in the meantime — this is an organizational issue.

Expiry of lost electronic cash can basically be achieved in two ways: Either electronic coins expire for offline spending at some point by themselves, or reclaimed electronic coins are blacklisted; see Section 4.5. Preliminary recovery is needed unless electronic cash can be blacklisted very fast: A user who needs money back at once should get an advance from the bank. In particular, this should be mandatory if the purse is not really lost, but no longer working. These options and their limitations are discussed in more detail in Section 5.1.

2.5 Relation with User Identification

A bank cannot be expected to give money back to a user if a thief or dishonest finder has spent it. Hence, if loss tolerance is to be more than a measure for purses that are physically broken,

---

5 Note that a key given to the bank should not allow the bank to decrypt obsolete backup information, because of Requirement 5. It can therefore not be a key stored once and for all by the user.

6 If only identification of double-spenders is required, this may not be seen as a problem, since the bank can also notice afterwards that the same coin was deposited whose value was already given back. However, in solutions where tamper-resistance is used to prevent double-spending, it may not be seen as acceptable.

mislaid, or stop working, one needs user identification for payments.[7] This does not contradict untraceability, since the identification is only made by the payer's own electronic purse. With current technology, this can only mean PINs or passwords. However, it is unwise to use the same PINs for low-value payments and for withdrawals, since payments are often made in situations where the PIN entry can easily be observed. Hence one needs provisions for two PINs: a withdrawal PIN that should be handled with great care, and a payment PIN that people who cannot remember two PINs can set to an easily remembered value or write down. Furthermore, it is useful to allow users to unlock amounts (i.e., to free them for spending without PIN entry) before payments, so that they can enter the PIN where they cannot be observed [Deth_94]. Thus, loss tolerance after a theft refers to the PIN-protected part of the amount only.

3 Sketch of the Brands Electronic Coin System

To present a concrete version of our ideas, we consider the efficient payment system from [Bran_94]. The security of this system as a whole has not been proved, but many subproblems have been, and a proof of any of the efficient schemes seems to be beyond currently known methods. We will also sketch a generalization of our protocols to a more general class of schemes; however, the security proof for our best solution for strong loss tolerance does not necessarily hold in other instantiations of this generalization (whereas an alternative sketched in Footnote 13 does).

Before presenting cryptographic details, we refine the high-level description from Figure 1 to show how double-spenders are identified (see Figure 2): Naturally, such a conditional identification can only work if the electronic purse has to reveal different information in the two payments. To this end, it receives a challenge C from the electronic till in each payment and has to send a response R, which is related to both the coin and the challenge. This response is guaranteed to be different for different challenges. From two such responses for one coin, the bank can compute the identity. Moreover, the bank requires the identity of the payee (or the till) and a transaction number for this payee to be encoded into the challenge C in a predefined way (i.e., it only accepts deposits if this is fulfilled). Thus even a collusion of a payer and two payees cannot perform two payments and deposits with the same challenge and response.

The fact that the identity can be computed from two responses R and R^* presupposes that the identity is somehow in the coin, but in a form usually invisible. For this, it is explicitly contained in the information the bank actually signs in a withdrawal (i.e., in the grey form of the coin in Figures 1 and 2). After the transformation, the coin is information-theoretically indistinguishable from coins for other identities, but the purse's *knowledge* about the coin, which allows the purse to answer challenges in payments, is restricted and still related to the identity. We now show this in detail.

---

7 Of course, insurances might be willing to cover the case of stolen electronic purses with freely spendable electronic cash, but there are no technical measures for distinguishing honest and dishonest claims if there is no user identification (just as with thefts of normal purses). Furthermore, although the loss tolerance measures allow the payments made after the claimed loss to be traced to the purse, this will usually not result in finding who had the purse at that time.

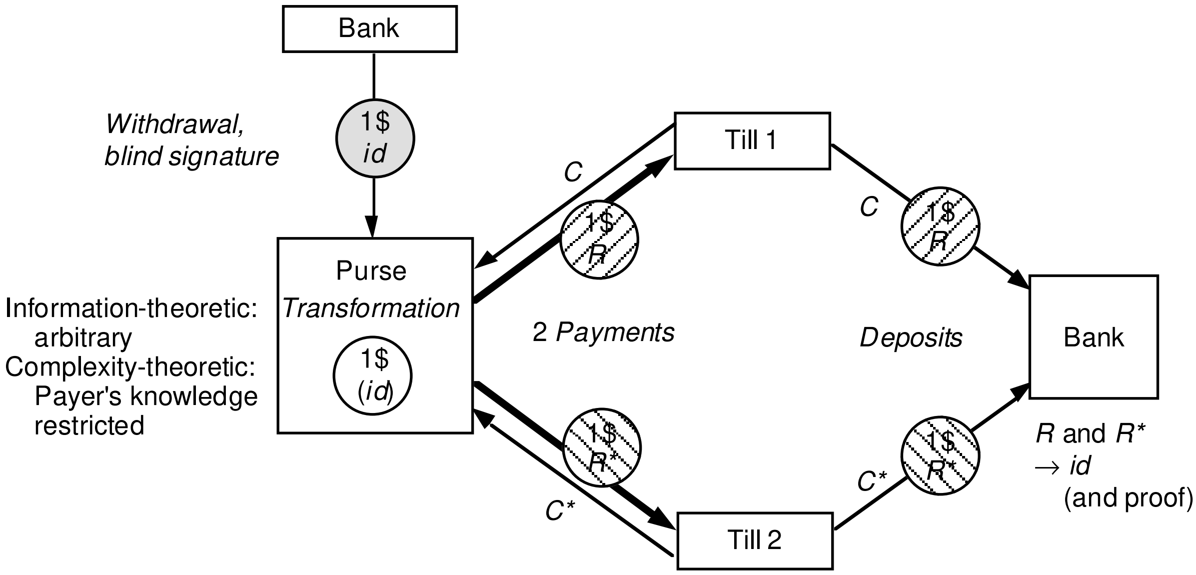

Figure 2 High-level illustration of the identification of a double-spender. See the explanation in the text above.

3.1 Cryptographic Preliminaries

The payment system in [Bran_94] is based on the assumption that it is infeasible to compute discrete logarithms in certain families of large groups. Computing a discrete logarithm means computing a value x with $g^x = h$, where g and h are given elements of the group and the group is written multiplicatively. Here and in the following, we omit details of cryptographic definitions of infeasibility for brevity. The best-known choice of groups are multiplicative groups $\mathbb{Z}_p^*$ of integers modulo large primes p and their subgroups.

In the given scheme (like in several others, most noticeably [DSS_92]), the order of each group, i.e., the number of its elements, must additionally be a prime number q. The best-known choice of groups is now achieved by generating a prime q, then a second prime p with $q \mid (p - 1)$, and using the unique subgroup G_q of order q of $\mathbb{Z}_p^*$. For simplicity, we will always use this special case in the following. A random element of G_q can easily be generated by exponentiating a random element g' of $\mathbb{Z}_p^*$ with $(p - 1)/q$.

In a group of prime order, every element $g \neq 1$ is a generator, i.e., its powers g^x cover the whole group. More precisely, the exponentiation function exp: $x \to g^x$ is an isomorphism between the additive group of integers modulo q and G_q.

Additionally, the scheme relies on a family of hash functions *hash*, which map strings of arbitrary length into the groups G_q. These functions must at least be one-way, i.e., it should be infeasible to find a string s with $hash(s) = s'$ for given s'. In fact, much stronger assumptions are made about *hash*, which are closely linked to the details of the scheme. This use of hash functions is the main reason why one is currently unable to prove efficient electronic cash systems secure.

3.2 The Blind Signature Scheme

The blind signature scheme used in Brands's payment system was defined in [ChPe1_93] as an extension of [Schn5_91, ChEG_88]. The bank chooses an instance of the discrete logarithm problem, i.e., primes q and p, a generator g of G_q, a number x modulo q, and $h := g^x$. The secret key is x, and the remaining parameters are published. Protocol 1 shows how a message is signed. We now describe the five parts of this protocol. (For most of this paper, it is sufficient to understand Parts 1 and 2.) When this protocol is used in a withdrawal of electronic money from a bank account, it must be accompanied by a *withdrawal order*, i.e., a message signed by the electronic purse, which later enables the bank to prove that it was allowed to debit the account.

1. The recipient transforms the message m into m' by random operations compatible with the group structure. (Thus m corresponds to the grey form of the coin in Figures 1 and 2, and m' to the white form.)

2. The bank sends the value $z = m^x$. This is called a signature commitment on m: It is like a signature in the sense that it can only be computed with the secret key. However, nobody can verify that z equals m^x without computing a discrete logarithm. Hence the rest of the message exchange is intended to convince the recipient of this fact.

   The recipient transforms z into a signature commitment z' on m' as shown in the protocol.

	Signer = Bank (Secret: x)	Public: $q, p, g, h = g^x$	Recipient = Purse
1. *Message*	m	$— m \rightarrow$	$m' := m^s g^t$ (s, t random mod q; $s \neq 0$)
2. *Signature commitment*	$z := m^x$	$— z \rightarrow$	$z' := z^s h^t = m'^x$
3. *Random variant*	w random mod q ("Random h") $a := g^w$ ("Random z") $b := m^w$	$— a, b \rightarrow$	$a' := a^u g^v$ (u, v, random) $b' := (b^{su} a^{tu}) m'^v$
4. *Challenge*		$\leftarrow c —$	$c' := hash(m', [pk',]z', a', b')$ [pk' explained later] $c := c' u^{-1}$
5. *Response*	$r := w + cx$	$— r \rightarrow$	Verify $g^r = a h^c$ $m^r = b z^c$ $r' := ur + v$

Protocol 1: Blind signature scheme

3. The signer generates a random variant of the secret and everything based on it. Here, the "random secret" is w, and two values a and b are computed with it just as h and z were computed with the real secret x.

   The recipient transforms a and b into values a' and b'. Note that a' and b' equal $g^{w'}$ and $m'^{w'}$, respectively, with $w' = wu + v$. Thus they can play the same role for proving that z' is the correct signature commitment on m' as a and b do for z and m.

4. Basically, the recipient sends a random challenge c to the signer. However, he will later have to convince others (the payees) of the validity of the signature on m'. In principle, they would only be convinced if *they* had chosen the corresponding challenge c' randomly. Here, the hash function is used according to an idea from [FiSh_87]: One hopes that *hash* is sufficiently similar to a random oracle so that computing c' as the hash value of all the values known at this step of the protocol is just as good as choosing c' randomly.

   The blinding transformation from c' to c is chosen so that the recipient can later compute the correct response r' to c' from the response r that the signer gives to c in the next part.

5. The signer responds with the linear combination between the random secret w and the real secret x with the coefficient c. The recipient can verify the correctness of this response on the images under the exponentiation homomorphism as shown in the protocol.

   Finally, the recipient needs a similar response $r' = w' + c'x$ for m'. One can easily see that the transformation in the protocol yields exactly this value r'.

3.3 Encoding Identities into Coins

For the encoding of identities into coins, two further random generators g_1 and g_2 of G_q have to be chosen a priori for the whole system.[8] For one payer, the bank always signs the same message m, which is constructed as follows:

1. The purse chooses a secret random number i (mod q) and gives $h_1 := g_1{}^i$ to the bank. We call i the identity proof of the purse.
2. With an interactive protocol (a simpler version of Protocol 1), the purse proves to the bank that it really knows i with $h_1 = g_1{}^i$. The bank also verifies that this value h_1 is not used by any other purse yet; if it is, it asks the purse to repeat its choice.
3. The payer signs, e.g., with a handwritten signature, that $m = h_1 g_2$ will be used for his withdrawals. If the bank can later find out i with $g_1{}^i g_2 = m$, this will count as a proof that a coin from this purse was spent twice. Note that the bank does not know i in advance.

Since m is constant, Parts 1 and 2 of Protocol 1 only have to be performed once for all the withdrawals with one purse.

---

[8] In principle, the randomness of these parameters and q and p now has to be trusted by everybody. In practice, this can be achieved by generating them from old tables of random numbers, using a standard deterministic procedure. If one believes that computing discrete logarithms is infeasible even for someone who chooses the group and the generator, the bank may still generate all these parameters alone.

The identity is preserved through the blinding transformation as follows: For payments, the purse will have to know a representation of m' as $m' = g_1^{x_1} g_2^{x_2}$. Under the discrete logarithm assumption, it is infeasible to compute two different representations of the same number m' [BoCP_88, Pede_92, ChHP_92]. Hence, if one believes that the blind signature protocol can *only* be used for messages $m' = m^s g^t$, and thus $m' = g_1^{is} g_2^{s} g^t$, the purse must choose $t = 0$ and thus $x_1 :=$ is and $x_2 := s$, and the invariant of all its coins m' is $x_1/x_2 = i$. This invariant is why the scheme was called a restrictive blind signature scheme.

3.4 Payments

The idea behind the following payment protocol (Protocol 2) is that each response yields two linear equations about four variables, two of which are x_1 and x_2. One response, i.e., two equations, do not reveal anything about the identity proof i, whereas two responses yield four linearly independent equations and thus all four variables and consequently $i = x_1/x_2$.

The two additional secret variables y_1 and y_2 for this purpose are chosen during withdrawal and encoded as $pk' = g_1^{y_1} g_2^{y_2}$. This is the unexplained value included in Step 4 of Protocol 1. The pair (m', pk') can be seen as a special public key of the owner of this coin, and answering the challenge C means signing C with the corresponding secret key (x_1, x_2, y_1, y_2).[9] The resulting payment protocol is shown in Protocol 2. The value *Time* can be the real time or a sequence

	Purse (Secret: all the information about one coin)	Public q, p, g, h, g_1, g_2	Till
1. *Send coin*		$— m', z', pk' \rightarrow$ $— a', b', c', r' \rightarrow$	Verify $c' = hash(m', z', pk', a', b')$, $g^{r'} = a'h^{c'}$, $m'^{r'} = b'z'^{c'}$, $m' \neq 1$.
2. *Challenge*	(Verify Id_{Payee}, *Time*, if possible)	$\leftarrow C —$ possibly: $\leftarrow Id_{Payee}, Time —$	$C = hash'(m', pk', Id_{Payee}, Time)$, where *hash'* is another "random" function
3. *Response*	$R := (sig_1, sig_2)$ with $sig_1 = Cx_1 + y_1$, $sig_2 = Cx_2 + y_2$	$— R \rightarrow$	Verify $g_1^{sig_1} g_2^{sig_2} = m'^C pk'$

Protocol 2: Payment protocol

9 Actually, this is the fail-stop signature scheme from [HePe_93]. The fact that a message C containing the payee's identity is signed in a payment also serves to settle disputes between a bank and a payee where the bank claims that it has deposited the same coin to another payee before [PWP_90].

number; for his own security the payee must not use the same value *Time* twice. One can omit *a'* and *b'* in the first message, since the recipient can compute the only acceptable values from the second and third verification equations.

3.5 Deposit and Double-Spending Detection

In a deposit, the till forwards all the information from a payment, including Id_{Payee} and *Time*. The bank first verifies the validity of the coin just as the till did in the payment. Secondly, it prevents double-depositing by verifying that the challenge C was constructed correctly and that the payee has not deposited the same coin with the same challenge before; this is simple because of the value *Time*. Thirdly, it detects double-spending by entering the coin, identified by

$$coin\_id := m',$$

into a large hash table for comparison with previous deposits by other payees within the period of validity of this coin. We will call this table *double_spending_check_table*. (Of course, any other data structure for fast searching and insertion in large amounts of data can be used instead.) The double-spending check can be done off-line. If the same *m'* is already in the table, the corresponding challenge-response pairs (C, R) and (C^*, R^*) are retrieved and the identity proof i is computed as $i = (sig_1 - sig_2)/(sig^*_1 - sig^*_2)$. The bank computes $m = g_1{}^i g_2$ and looks up to which purse this m belongs.

4 Loss Tolerance in a Concrete System

We now provide the payment system described in Section 3 with loss tolerance, still considering several options: with and without backup purses as well as weak and strong loss tolerance. We start with some aspects that are common to all variants.

Note that for brevity, we describe protocols in a slightly idealized way. In particular, in signatures, we omit the key used if it is clear from the context, a type field indicating the protocol step to which the signature belongs, and measures to avoid replay, since these are not special to loss tolerance.

4.1 Backup Information

In Section 2, we just talked about “the” backup information for a certain coin. However, the information is usually different in different steps of the protocol. For a modular presentation of several options, we now introduce notation for these representations. The information that will be input to the protocol SETTLING concerning one coin (see Section 2.4) is called *settling_info*. In our concrete case, it is

$$settling\_info := (m', z', pk', c', r')$$

in all options. Note that this is not the complete information the electronic purse had about the coin. The secret values (x_1, x_2, y_1, y_2) can be omitted, since recovery after a loss does not work like a payment. In particular, the payer's identity is known to the bank during recovery, and hence there is no need to identify him if it turns out that the coin has been both reclaimed and spent. It is not only efficient, but also necessary that the bank never learns the secret values: Otherwise, it could falsely prove double-spending, in contradiction to Requirement 3b (security of owners of lost purses against bank insiders).

More generally, in several electronic coin systems the transformed version of the coin (white in Figures 1 and 2) can be interpreted as a pair (*pk*, *sig*), where *pk* is a public key of some signature scheme and *sig* is the bank's signature on *pk*, and the electronic purse additionally stores the secret key *sk* corresponding to *pk* and uses it to answer the challenge in a payment. In all these systems, *settling_info* := (*pk*, *sig*) is the information needed to settle the claim about one coin.

In practice, an electronic purse will contain coins signed with different keys (to permit regular key change for security reasons and for different expiry dates and denominations). We denote them as different coin types. As long as the keys are independent, coins of different types can be treated as independent systems realized on the same purses. Hence we usually omit references to the coin type, e.g., in *settling_info*.

By *reclaim_info*, we will denote the complete information the owner of a coin needs to reclaim the coin. Where encryption is used, *reclaim_info* denotes the cleartext. Its content depends on the options chosen and is therefore specified later. The information that is actually stored is called backup information; it varies so much according to the options that it does not even need a common variable name.

4.2 Settling

The protocol SETTLING for a certain lost purse with identity Id_{Purse} (and a certain coin type) starts with the following preconditions (achieved by the protocols RECLAIMING described below for the different options): The bank has a list *Reclaimed_list*(Id_{Purse}) of the values *settling_info* = (*m'*, *z'*, *pk'*, *c'*, *r'*) for coins that the payer claims to have possessed in this purse. Moreover, if strong loss tolerance is required, the payer has a confirmation by the bank that these coins are considered as lost by exactly this purse, in particular, that they were not reclaimed before for other purses. For concreteness, let this confirmation consist of signatures *sign*(*settling_info*, Id_{Purse}, *Time*) for individual coins, where *Time* is the time when the bank signed.

The format of the data attached to the coin identifiers *m'* in *double_spending_check_table* is extended by an attribute *type* ∈ {*deposited*, *reclaimed*}, and in the latter case, the identity Id_{Purse} of the lost purse, see Figure 3. In the more general case with *settling_info* = (*pk*, *sig*), the values can be stored in *double_spending_check_table* according to *coin_id* = *pk*.

The following steps are carried out at once when SETTLING is called:

SETTLING, PART I

1. The bank verifies that the values *settling_info* in the input list *Reclaimed_list*(Id_{Purse}) are valid coins, just like the till in the first step of a payment (recall that *a'* and *b'* can be recomputed during this verification). More generally, if *settling_info* = (*pk*, *sig*), the bank verifies that *sig* is its own valid signature on *pk*.
2. The bank only retains the valid coins in *Reclaimed_list*(Id_{Purse}) and gives each of them an attribute *found* := *false* (see Figure 3).
3. The bank looks each of the values *m'* up in *double_spending_check_table* (see Section 3.5).

- If *m'* is not found there, it is now entered with *type* = *reclaimed*.
- If *m'* is already there with *type* = *deposited* (i.e., this coin was spent and deposited), the attribute *found* of this coin in *Reclaimed_list*(Id_{Purse}) is set to *true*. The entry in *double_spending_check_table* is not changed.
- If *m'* is already there with *type* = *reclaimed*, and if only weak loss tolerance is required, the bank simply refuses the new recovery attempt by setting *found* in *Reclaimed_list*(Id_{Purse}) to *true*. The entry in *double_spending_check_table* is not changed.
- For completeness, we have to define: If *m'* is already there with *type* = *reclaimed*, and if strong loss tolerance is required, the bank has to record both recovery attempts, i.e., both identities are stored under *m'* in *double_spending_check_table*, and the attributes *found* in both lists, say *Reclaimed_list*(Id_{Purse}) and *Reclaimed_list*(Id_{Purse^*}), remain *false*[10].

double_spending_check_table

coin_id	*type*	Id_{Purse}	(*rest*)
…			
m'	*reclaimed*	•	—
…			
*m**	*deposited*	—	*C**, *R**
…			

Reclaimed_list(Id_{Purse})

coin_id	*found*	(*rest*)
…		
m'	*false*	*z'*, *pk'*, *c'*, *r'*
…		

Figure 3 Data structures for settling.
For instance, *m'* is an identifier of a coin that can be reclaimed successfully.

Note that the simple refusal to carry out a second recovery for the same coin in the case of weak loss tolerance is secure, assuming that the user had stored the backup information safely, since only the owner of the lost purse and the bank could have known this information about an unspent coin and reclaimed the coin before.

Deposit is modified as follows:

MODIFICATION TO DEPOSIT

- When the bank finds the newly deposited coin in *double_spending_check_table* with *type* = *reclaimed*, it sets *found* := *true* in the corresponding entry (or entries) in *Reclaimed_list*(Id_{Purse}), and updates the entry in *double_spending_check_table* to *type* = *deposited* with the corresponding challenge-response pair.

---

[10] This case is not strictly excluded by the precondition, which was that the bank had *signed* that the coin was not previously reclaimed, not that it *was* not in the table. Instead, we show below that this case cannot occur if the cryptographic mechanisms are secure (and as long as the bank is honest, but the bank would only harm itself if it gave the same coin back to two payers). The reason why we did not make the simpler precondition is that it must be possible to settle disputes about whether a coin has been reclaimed before, see Sections 4.3 and 4.4.

At some time $t_{deposit\_expiry}$, coins from a certain list *Reclaimed_list*(Id_{Purse}) can no longer be deposited in the normal way (see Sections 2.4 and 4.5). At this time, the settling protocol for this list is finished:

SETTLING, PART II

4. The bank gives the monetary value of those coins in *Reclaimed_list*(Id_{Purse}) that still have the attribute *found* = *false* irrevocably back to the owner of the purse Id_{Purse}.
5. If strong loss tolerance is desired, the bank proves to the payer that the other coins from *Reclaimed_list*(Id_{Purse}) were deposited by showing the challenge-response pairs from the deposits.
6. *Reclaimed_list*(Id_{Purse}) remains stored just as long as *double_spending_check_table* for this coin type.[11] Conversely, the payer can delete the bank's signatures that the coins were not previously reclaimed as soon as he has got the money back or a proof that it was deposited.

In the case of strong loss tolerance, a court can decide a dispute whether SETTLING was carried out correctly as follows:

DISPUTE

- The payer shows the signed confirmations by the bank to prove what coins were in the input list. The court verifies that they were valid. Now the bank either has to give the value of these coins back or prove the deposit to the court just as in Step 5 above. (The question whether the bank has already given the value of a coin back can be decided if this was done by a statement of account indicating precisely which coin was given back.)

The "proof" of deposit in Step 5 is sufficient, since the response is a signature with a key that is still only known to the owner of the lost coin.

4.3 Backups and Reclaiming with Backup Purses

The protocol for storing backup information on a backup purse simply consists of the high-level description from Section 2.2, where the information stored per coin is the value *reclaim_info* introduced in Section 4.1. The content of this information and the protocol for reclaiming vary.

Weak loss tolerance. The information needed for reclaiming is simply

$$reclaim\_info := settling\_info.$$

For reclaiming, this information is simply sent from the backup purse to the bank with the organizational measures from Section 2.3, and the bank inputs it to SETTLING.

Strong loss tolerance. In this case, the precondition for SETTLING is that the bank has a list *Reclaimed_list*(Id_{Purse}) of values *settling_info* and the user has the bank's signature that these values are considered as lost by exactly this purse. For any coin that the bank rejects as previously reclaimed, it must be able to prove this fact. Essentially, the bank therefore needs a signature that

[11] Instead, it can be deleted now if the payer signed at the end of RECLAIMING that he has now finished all reclaiming for this lost purse. Instead of *Reclaimed_list*(Id_{Purse}), only this signature remains stored.

is provably by the owner of the coin and denotes for which lost purse the coin is reclaimed.[12] Thus the information needed for reclaiming will usually contain secrets, say

$$reclaim\_info := (settling\_info, reclaim\_secrets).$$

For efficiency, one can often omit parts of *settling_info* when storing *reclaim_info* and recompute them from *reclaim_secrets*. We use

$$reclaim\_secrets = s,$$

i.e., the blinding exponent that relates m' to m.[13] The reclaiming protocol starts with the organizational measures from Section 2.3 and then proceeds as follows:

RECLAIMING (FROM A BACKUP PURSE)

1. For each coin to be reclaimed, the new purse sends (*settling_info*, s) and a signature on this pair with its normal public key. (This key must have been certified as belonging to the same payer as the lost purse, Id_{Purse}.)
2. The bank verifies that the same coin was not reclaimed for the same Id_{Purse} before, by looking m' up in the current *Reclaimed_list*(Id_{Purse}).[14] If m' is found there, the bank shows the purse's signature from Step 1 of the old execution of RECLAIMING to prove this fact. (And if the user claims that the previous reclaiming attempt was interrupted, the bank sends its own signature from Step 3 of that attempt again.)
3. The bank verifies the new signature by the purse and that $m' = m^s$, where m is the message signed in withdrawals with the lost purse. If all these verifications were successful, the bank inserts *settling_info* into *Reclaimed_list*(Id_{Purse}) and stores the new signature there, too. It gives the user a signature that this coin is considered lost by this purse, i.e., *sign*(*settling_info*, Id_{Purse}, *Time*), and also stores its own signature.

A court can settle a dispute about an execution of RECLAIMING as follows:

DISPUTE

- The new purse and the bank repeat the steps of RECLAIMING, sending their messages via the court. The court verifies all the signatures, too, and that $m' = m^s$. If the value of m for the lost purse is disputed, the bank has to show the user's signature on it (from Step 3 in Section 3.3). The verifiable result is that the user obtains the signature that this coin is considered lost by this purse.

---

[12] A signature with the normal public key of the owner of the *purse* is not sufficient, since as soon as the bank sees *settling_info*, it could give it to other people who then sign that they reclaimed it earlier. Conversely, the bank cannot be forced to give the money to anybody who knows *settling_info*, since two colluding users might both present it.

[13] An alternative is *reclaim_secrets* := (x_1, x_2, y_1, y_2). Reclaiming can then be signed similar to a payment. An advantage is that this can easily be generalized as *reclaim_secrets* = *sk*. The disadvantage is that the backup purse has to be equally well protected (by tamper-resistance and user identification) as the main purse, since it contains enough information for the payment protocol. So far, we assume that it can be a cheap memory card.

Yet another possibility works without any secrets: The purse first sends a one-way hash value of *settling_info*, obtains a promise by the bank that no coin with this hash value has been reclaimed before, and then sends *settling_info* itself. However, this solution requires more complicated timing, see the deposit protocol in [BüPf_89, Section 4.3].

[14] If *Reclaimed_list*(Id_{Purse}) is deleted at the end of an execution of SETTLING (see Footnote 11), the bank first checks if the signature that all coins have been reclaimed is already there; if yes, it aborts.

The rest of this section demonstrates the security of this protocol in conjunction with SETTLING from Section 4.2. In some places, we need one explicit assumption about the payment system:[15]

BLIND SIGNATURE ASSUMPTION: It is infeasible for any number of colluding participants (not including the bank) to obtain k different values m' with signatures by the bank that designate them as coins, unless they carry out at least k withdrawals. ◊

Informally, this assumption means that the values m' can in fact be seen as coin identifiers.

THEOREM 1: Under the usual cryptographic assumptions about the security of the original payment system, the described loss tolerance measures with backup purses fulfil the requirements from Section 1.3, with the following precise versions of Requirements 4 and 5:

- *Requirement 4:* The untraceability of a payer who does not lose his purse is only reduced as follows: If his backup purse is less protected (by tamper-resistance and user identification) than his main electronic purse and someone steals it, succeeds in using it, and colludes with the bank, the untraceability of this payer is lost.
- *Requirement 5:* By the recovery for a lost purse, at most the payments between the last withdrawal with this purse and the loss become traceable. If the payer remembers that he already spent certain coins, he can avoid their being traced.
- The strong loss tolerance is of course restricted to those coins that were withdrawn before the last backup. ◊

We separate two cryptographic lemmas from the proof of this theorem. Informally, the first one shows that s is a kind of proof to which purse a coin belonged.

LEMMA 1: Under the discrete logarithm assumption for the chosen family of groups, it is infeasible for any number of colluding participants (not including the bank) to show nonzero values s and s^* with

$$m' := m^s = m^{*s^*}, \qquad (\ast)$$

where m and m^* are the messages that the bank always signs in the blind signature protocol with two different purses. ◊

PROOF: First recall that the bank enforces $m \neq m^*$ in Section 3.3. This implies $s \neq s^*$ in Equation ($\ast$), since otherwise, exponentiating with s^{-1} would yield $m = m^*$. (Here we need the fact that the exponents are numbers modulo a prime q, so that every nonzero element has an inverse.) Furthermore, the bank enforced $m = g_1{}^i g_2$ and $m^* = g_1{}^{i^*} g_2$, where i and i^* are known to the purses. Equation ($\ast$) therefore implies $g_1{}^{is} g_2{}^s = g_1{}^{i^*s^*} g_2{}^{s^*}$, and thus that the discrete logarithm of g_2 with respect to g_1 can be computed efficiently as $(i^*s^* - is)(s - s^*)^{-1}$. Thus fulfilling Equation ($\ast$) is as hard as computing discrete logarithms. ❑

The second lemma will be used to show that no cryptographic properties are endangered when s becomes known to parties who would not have known it in the system without loss tolerance. We

[15] "Explicit" means that it is not a trivial consequence of the assumption that the payment system is secure: one might conceivably be able to forge signed values m', but not know the corresponding secrets needed for paying. However, the same assumption was used and derived from a more basic assumption by Brands [Bran_94, Prop. 7].

refer to Schnorr identification [Schn5_91]. In our context, it can be described as the signer's part of Protocol 1 if one omits everything to do with a specific message, i.e., m, z, and b. By its security, we mean the weak statement that it is impossible for anyone else to find out the secret x.

LEMMA 2: Under the assumption that Schnorr identification is secure even in parallel executions (a necessary condition in [Bran_94][16]), it is infeasible to find out the identity proof i of a purse even if one knows the values s used in the withdrawals of any number of coins, in addition to all the data the payees and the bank obtain about the coins. ◊

PROOF: We assume that there is an attacker algorithm A that contradicts Lemma 2. We show how to construct an equally successful, almost equally efficient attacker algorithm $A_{Schnorr}$ against parallel Schnorr identification. In more technical terms, we perform a blackbox reduction between the two problems, the usual type of reduction between interactive algorithms (see, e.g., [GoOr_94]).

Without loss of generality, we assume that the data that A obtains about each coin is s, m', pk' and a response R on a challenge C, where A can choose C after it knows the values s, m', pk' about *all* the coins.[17] (The remaining values, c, z', a', b', c', and r', can be omitted, since we can first assume that A obtains u and v instead, from which all others can be derived, and then simulate the choice of u and v, which is random and independent of anything else, inside A.)

The algorithm $A_{Schnorr}$ first obtains an input $g_1{}^i$ and values $a_k = g_1{}^{w_k}$ ($k = 1, \ldots, K$) from K parallel executions of Schnorr identification. To prepare a suitable input for its subroutine A, it sets $m := g_1{}^i g_2$. Then it chooses values s_k and $y_{2,k}$ randomly for $k := 1, \ldots K$ and computes $m'_k := m^{s_k}$ and $pk'_k := a_k g_2{}^{y_{2,k}}$. It starts A with the inputs m and all the values s_k, m'_k, and pk'_k. The interactive subroutine A now iteratively outputs a challenge C_k for some k and waits for a response R_k, which should consist of $sig_{k,1} = C_k i s_k + w_k$ and $sig_{k,2} = C_k s_k + y_{2,k}$. To compute this, $A_{Schnorr}$ continues the k-th execution of Schnorr identification by sending $c_k := C_k s_k$ as its challenge. Thus it obtains the response $r_k = w_k + c_k i$. This is just the desired $sig_{k,1}$. Since $A_{Schnorr}$ can compute $sig_{k,2}$ on its own, it can now give R_k to A. When A has obtained all the K responses, it should output the value i, and if it does, $A_{Schnorr}$ outputs it, too. ❏

PROOF OF THEOREM 1: *Requirement 1* is the security of the bank. If no tamper-resistance is used to prevent double-spending, the usual cryptologic aspects of the bank's security are not weakened, since the bank does not perform any new computations with its secrets. Moreover, the Blind Signature Assumption guarantees that "coins" with just *settling_info* are no easier to forge than coins that one can pay with. Thus we only have to show that the bank does not lose money by accepting coins with the same identifier m' in more than one recovery, or a deposit and a recovery. The use of *double_spending_check_table* in SETTLING obviously prevents a deposit

[16] Although "in parallel executions" is not mentioned in [Bran_94], this condition is needed in Prop. 16 about the protocol for wallets with observers. It is equivalent to the infeasibility for the user to find out the value o_1, which corresponds to the identity proof of the observer. As mentioned by Brands, his observers perform exactly Schnorr identification, where sending the random variant (a in our notation) happens in the withdrawal and the challenge and response in the payment. Since many withdrawals may take place before a payment, the executions are parallel, i.e., an attacker may, in the worst case, base the challenge in one execution on the random variants of all other executions.

[17] Actually, A's choice of C is quite restricted, and hence a somewhat weaker assumption than fully parallel executions would be sufficient, but the freedom left in Id_{Payee} and $Time$ is not negligible.

and a recovery, and Step 2 of RECLAIMING prevents two recoveries for the same purse. It remains to prove that it is infeasible even for colluding payers to reclaim the same coin m' successfully for two purses (compare Footnote 10). This is just Lemma 1, since these payers would have to show values s and s^* with $m' = m^s = m^{*s^*}$.

If tamper-resistance is used (Solution 3 in Section 1.2), we also have to show that double-spending does not become easier by the fact that the tamper-resistant purse outputs s to the backup purse, which was not assumed to be tamper-resistant. The only data a cheating payer now obtains about his coins are those the bank also sees and s. If he could spend one of his coins for a second time with these data, he could also compute the value i of his purse, just like the bank in Section 3.5. This would contradict Lemma 2.

Requirement 2: The payees' security is obviously unaffected, since there are no new steps where deposits can be rejected.

Similarly, none of the new steps have any effect on the owners of other purses.

It remains to show that the loss tolerance measures do not decrease the security of payers who do not lose their purses. The only loss tolerance measure for these payers is storing the backup information. If nobody finds the backup purse or if it is equally well protected as the main purse, there is no problem. Thus we now assume that the backup purse is a simple memory card and is stolen. We have to show that the thief cannot (a) make withdrawals from the payer's account, (b) make payments or deposits with the payer's coins, (c) wrongly brandish the payer as a double-spender, or (d) reclaim the payer's coins. (a) is clear: no information about the key needed to sign withdrawal orders is on the backup purse. Lemma 2 applies to the thief; this immediately excludes (c). If the thief could make payments or deposits (even if only for one challenge), this would yield double-spending together with the payments that the payer makes. Thus i could be computed, which contradicts Lemma 2 again. This proves (b). For (d), first note that the thief cannot reclaim the coins under the payer's name (which, although of no direct advantage to the thief, might put the payer in an embarrassing position), since this requires a signature by a real purse of the payer. If the thief could reclaim the coin in another name, this would not prevent the honest payer from reclaiming it, too; see the last item of Step 3 in SETTLING and the corresponding dispute. (However, this would harm the bank, and was shown to be infeasible under Requirement 1.)

Requirement 3, in our case, is the strong loss tolerance. First note that all the information that the owner of a lost purse divulges in RECLAIMING would also be known to a thief of the backup purse, and we have shown for Requirement 2 that this information does not enable anyone else to cheat the payer. It remains to show that each unspent coin for which a backup was made can be reclaimed successfully. This follows from the impossibility of the types of cheating called (b) and (d) above, since the bank can only refuse to give the value of a valid coin back if it either has the challenge-response pair from a deposit or a signature by the same payer under a previous attempt at reclaiming, see the two disputes. The only problem would be if an honest payer used the same s twice, but this only happens with an exponentially small probability.

Requirement 4: The untraceability of people who do not lose their purses is obviously only restricted by the possibility that the backup purse falls into the wrong hands. This immediately yields the claim made in the theorem.

Requirement 5: For the partial untraceability of the owners of lost purses, it suffices to show that all the payments with coins that are not reclaimed remain untraceable. Here we use that the untraceability in Brands's system is information-theoretic, i.e., independent of the computing power of the attacker, and that the coins are unlinkable, i.e., even if it becomes known by context information who made some of the payments, this does not help in tracing other payments (see [Bran_94, Prop. 11]). Thus the pairs (m, $m' = m^s$) for some coins do not help in tracing other coins. An information-theoretic attacker can compute s from such a pair, and hence divulging s for some coins during reclaiming does not help tracing other coins either. ❑

4.4 Backups at the Bank and Corresponding Reclaiming

The encryption needed if the backup information is stored at the bank can be carried out with an almost arbitrary encryption system. The only constraint is that one should be able to decrypt encrypted messages individually and in a different order. Since we allow incremental backup, these messages are the values *reclaim_info* for individual coins. We assume here that the decryption key, and thus also the encryption key, is constant over many withdrawals (see Section 5.2 for alternatives). Special provisions are needed for the partial untraceability of the owners of lost purses (Requirement 5) and, if strong loss tolerance is required, against the bank embezzling backup information.

Weak loss tolerance. A first, simple modification is that each withdrawal is now preceded by a withdrawal order that contains the complete list of sequence numbers *seq_no* (in the sequence of all coins withdrawn by this purse) of the coins that will be on the purse after this withdrawal. At the end of the withdrawal of an individual coin with sequence number *seq_no*, the purse encrypts *reclaim_info* = *settling_info*. We denoted this by *ciph* := *enc*(*reclaim_info*). The purse computes a signature *sign*(*seq_no*, *ciph*) and sends *ciph* and this signature to the bank.

During reclaiming, the new purse requires a withdrawal order as described above and the signed individual ciphertexts from the bank. It only accepts ciphertexts whose sequence numbers are contained in the withdrawal order. Note that we assume here that the public key of the lost purse is known to the new purse; it should be input by the user. (Users who do not want to bother may have it input via the bank with a certificate from an independent certification authority.)

Thus, the new purse only decrypts coins that were present after *one* withdrawal. This is already a reasonable form of partial untraceability for owners of lost purses. Moreover, if the bank refers to the wrong withdrawal, the user may notice this (although not be able to prove it) and refuse to decrypt coins if he thinks they were already spent.

Strong loss tolerance. First note that with the protocol for weak loss tolerance, the user does not lose money if the bank declares an old withdrawal order to be the newest one: Any coins withdrawn in the deleted withdrawals cannot appear on statements of account and thus cannot be deducted from the account.[18] However, the bank can embezzle one coin (and recall that "coins"

[18] Here we need the fact that the electronic cash is withdrawn from an account, in contrast, e.g., to paying real cash for it. Anyway, if one pays cash, storing the backup information at the bank is not so natural. In that case, it seems impossible to guarantee that the bank uses the newest backup information. Thus, one has weak loss tolerance only, and both the cash-to-electronic-cash machine and the bank must be trusted for it.

are not restricted to small denominations) by claiming that the withdrawal order arrived and the coin was sent, but the withdrawal was interrupted before the purse sent the backup information. Since this claim might be true and the purse might have spent the coin, such a withdrawal cannot simply be declared invalid.[19] Essentially, we now have to ensure that the withdrawal order and the backup information become valid in the same step. We therefore integrate storing the backup information into the withdrawals as follows:

MODIFIED WITHDRAWAL

1. The complete withdrawal starts with the same signed withdrawal order as for weak loss tolerance, i.e., it contains the complete list of withdrawal sequence numbers of the coins that will be on the purse after this withdrawal.

2. In the withdrawal of an individual coin with withdrawal sequence number *seq_no*, the purse prepares *reclaim_info* before Step 4. More precisely,

$$reclaim\_info := (s, u, v, pk'),$$
$$ciph := encrypt(reclaim\_info),$$
$$ack := sign(seq\_no, a, b, c, ciph),$$

   and the message sent in Step 4 is

$$(c, ciph, ack).$$

3. In Step 5, the bank verifies the signature *ack*, computes *r*, stores

$$backup\_info_{Bank} := (seq\_no, a, b, c, ciph, ack, r),$$

   debits the account, and sends *r* to the purse.

Note that *reclaim_info* is no longer sufficient for reconstructing *settling_info*, for lack of *r'*. However, together with *r*, which the bank has to store for successful withdrawals, it will be. Reclaiming starts with the organizational measures from Section 2.3 and then proceeds as follows:

RECLAIMING (WITH BACKUP INFORMATION AT THE BANK)

1. The bank sends the last withdrawal order from the lost purse. For each withdrawal sequence number *seq_no* listed in this withdrawal order, the bank looks up if the withdrawal *seq_no* took place and was not interrupted before the money was deducted from the user's account. If this is the case, the bank sends the value $backup\_info_{Bank}$ stored for *seq_no*. The bank also sends a signature on all this information.

2. The new purse verifies the bank's signature and the signature by the lost purse on the withdrawal order. It only accepts values $backup\_info_{Bank}$ with correct withdrawal sequence numbers. The user in person verifies now or later that no newer withdrawals with the lost purse turn up on the statements of account, nor withdrawals of coins with sequence numbers that were contained in the last withdrawal order, but for which $backup\_info_{Bank}$ was not sent.

3. For each value $backup\_info_{Bank}$ it received, the new purse acts as follows:

[19] Recall that we do not trust the tamper-resistance of the purse enough to assume that it only starts spending coins after it has sent the backup information.

a) It verifies that the component *ack* is a valid signature with the key of the lost purse.

b) It verifies that r is correct, just as in Step 5 of Protocol 1.

c) It deciphers *ciph* and recomputes *settling_info* = (m', z', pk', c', r') just as in Protocol 1.

d) It sends *settling_info* and s to the bank in a signed message.

4. The bank verifies the new signature and that $m' = m^s$, where m is the message signed in withdrawals with the lost purse. If all these verifications were successful, the bank inserts *settling_info* into *Reclaimed_list*(Id_{Purse}), stores the new signature there, too, and gives the user a signature that this coin is considered lost by this purse, i.e., *sign*(*settling_info*, Id_{Purse}, *Time*).

Note that the bank does not verify in Step 4 that *settling_info* has anything to do with $backup\_info_{Bank}$. A court can settle a dispute about this protocol as follows:

DISPUTE

- The new purse and the bank repeat the steps of RECLAIMING, sending their messages via the court. The court tests all the signatures, performs the verifications of Step 2 and Step 3b, and verifies that $m' = m^s$. If the value of m is disputed, the bank has to show the user's signature on it (from Step 3 in Section 3.3). The verifiable result is that for all coins that passed the verifications, the user obtains the signature that these coins are considered lost by this purse.

THEOREM 2: Under the usual cryptographic assumptions made about the security of the original payment system, the described loss tolerance measures with backup information at the bank fulfil all the requirements from Section 1.3, with the following precise versions of Requirements 4 and 5:

- *Requirement 4:* The untraceability of a payer who does not lose his purse is only reduced as follows: It is now only computational instead of information-theoretic. Furthermore, if someone steals the decryption key and colludes with the bank, the untraceability is lost.

- *Requirement 5:* By the recovery for a lost purse, at most the coins that were present after *one* withdrawal are traced. Furthermore, if the bank cheats by not making this the *last* withdrawal, it suffers a financial loss, and the owner of the purse can prevent tracing of coins that he remembers to have spent. ◊

The following lemma extends Lemma 2 from Section 4.3 to the extended reclaiming information in this section.

LEMMA 3: If the payment system without loss tolerance is secure, it is still infeasible to find out the identity proof i even if one knows the values s, u, and v used in the withdrawals of any number of coins, in addition to all the data the payees and the bank obtain about the coins. ◊

PROOF: Without u and v, this would just be Lemma 2. We show that u and v can be computed efficiently from the remaining data and therefore do not make any difference. The bank has the data c and r from the withdrawal and c' and r' from the deposit, and given s, it knows that this withdrawal and this deposit belong together. This yields $u = c'c^{-1}$ and $v = r' - ur$. ❑

PROOF OF THEOREM 2: *Requirement 1:* Several parts are proved just as in Section 4.3: If no tamper-resistance is used to prevent double-spending, the cryptologic aspects of the bank's security are obviously not weakened by the loss tolerance measures. Similarly, the bank does not lose money by accepting coins with the same identifier *m'* in more than one recovery, or a deposit and a recovery. However, we also have to show that the bank's security is not weakened by the fact that it now needs valid data $backup\_info_{Bank}$ to make a withdrawal valid. The only problem would be if the payer could spend the coin although he had not sent valid data *ciph* and *ack*. In this case, the payer has at most received the "random variants" *a* and *b* from the bank. He can generate values with the same probability distribution on his own. Thus the bank is as secure as if the withdrawal had not started at all.

The security against double-spending in the case with tamper-resistance follows from Lemma 3 just as it followed from Lemma 2 in Section 4.3.

Requirement 2: Again, the loss tolerance measures for one payer obviously do not endanger the security of payees and other payers, and it remains to show that the security of a payer is preserved even if someone finds the decryption key and colludes with the bank to obtain the encrypted backup information. This is shown just as in Section 4.3 with Lemma 3 in the place of Lemma 2.

Requirement 3: Similar to Section 4.3, all the information that the owner of a lost purse divulges in RECLAIMING would also be known to a thief of the decryption key colluding with the bank, and thus the proof of Requirement 2 excludes the majority of ways to cheat a payer. It remains to show that each unspent coin with a sequence number *seq_no* for which the bank deducted money from the account can be reclaimed successfully. Let us fix such a value *seq_no*. First, the bank has to send some correct withdrawal order as the last one. Since the coin with *seq_no* was unspent at the time of the loss, it is either listed in this withdrawal order or was withdrawn later. However, the latter case can be excluded, since it would be declared invalid in a dispute. Thus the bank is forced to send a value $backup\_info_{Bank}$ for this coin, and this value must pass the tests in Step 3 of RECLAIMING. These tests ensure that $backup\_info_{Bank}$ contains the value *reclaim_info* that the lost purse prepared for *seq_no* and the values *a*, *b*, and *c* from the same withdrawal. Thus the new purse can reconstruct the correct values *settling_info* and *s* needed for Step 4. Finally, the possibilities that the bank wrongly refuses recovery by claiming that the coin was reclaimed or deposited are excluded as in Section 4.3.

Requirement 4: The untraceability of people who do not lose their purses is only restricted by the backup information at the bank and by the decryption key. Hence it is obviously as good as the security of the encryption system if the decryption key remains secret, and lost if someone obtains both the backup information and the decryption key.

Requirement 5: The new purse only decrypts coins with sequence numbers that were present after one withdrawal, and if this is not the newest withdrawal, the bank cannot deduct money for coins that were withdrawn later. The information divulged for a reclaimed coin is just *settling_info* and *s* (and provably the correct values). Hence the other coins remain untraceable for the same reasons as in Section 4.3. ❏

4.5 Expiry of Lost Coins for Off-Line Payments

In the protocol SETTLING, the value of a reclaimed coin is irrevocably given to the payer at a certain time $t_{deposit\_expiry}$. To preserve both the security of payees and that of the bank, the system timing must guarantee that no honest payee will deposit the coin after this deadline, even if the payer is dishonest and tries to spend it long after the claimed loss. This can be achieved by automatic expiry of coins for off-line payments or special blacklists of reclaimed coins.

We have to assume that all devices have partially synchronized clocks, i.e., an upper bound Δ on the difference between two local times is known. This is no serious restriction, since, e.g., Δ = 1 day is quite sufficient for our purpose.[20]

For simplicity, we omit an additional margin needed in transactions where the bank *has* to do something if its local time is at most a value t: In reality, if the bank refuses to do it, its transaction partner must still have time to appeal to a court, and the court must be able to decide before its local time $t - \Delta$. We also omitted the lengths of transactions.

Timing for normal expiry. Apart from $t_{deposit\_expiry}$, each coin type will have dates $t_{withdrawal\_expiry}$ and $t_{payment\_expiry}$ as attributes. The dates belonging to a coin type must be linked to its representation in key distribution (e.g., by a certificate from a central issuer) just like the name of the bank it belongs to and the denomination. The protocols for handling a coin of such a type are modified as follows:

WITHDRAWAL

a) The bank only issues such coins before its local time is $t_{withdrawal\_expiry} - \Delta$.

b) A purse only accepts such a coin before its local time is $t_{withdrawal\_expiry}$.

PAYMENT

a) A purse tries to pay with the coin before its local time is $t_{payment\_expiry} - \Delta$. More precisely, the purse should warn its user that the coin will expire for off-line payments long enough before this date.

b) A till only accepts the coin before its local time is $t_{payment\_expiry}$.

DEPOSIT

a) The payee has to deposit the coin before his local time is $t_{deposit\_expiry} - \Delta$.

b) The bank only accepts the coin in a deposit before its local time is $t_{deposit\_expiry}$.

The period where a payer can spend such a coin is, in the worst case, $t_{pay} := t_{payment\_expiry} - \Delta - t_{withdrawal\_expiry}$. This difference should be at least several months. The period where a payee can deposit a coin is at least $t_{deposit} := t_{deposit\_expiry} - \Delta - t_{payment\_expiry}$. If payees are just shops, this difference may be rather short, e.g., a week. The timing is secure in the sense that, whenever someone wants to carry out a transaction with a coin according to the respective condition a), the transaction partner will accept it according to the corresponding condition b).

[20] If one insisted on purses without any real-time clock, is would even be sufficient if the devices notice that certain deadlines occur with a difference of at most Δ. If these deadlines are easily remembered values such as New Year or the beginning and end of daylight saving time, one can ask the users to input the deadlines themselves, just as they change their calenders or set their watches. If they forget, they only endanger their own security.

If a payer has not spent a coin in time (not even to the bank in exchange for a newer coin), we can still allow him to deposit it directly back onto his account, similar to reclaiming it after a loss: He has to wait until the bank's local time $t_{deposit\_expiry}$. To avoid disputes whether the same coin was deposited in this way before, the purse gives a signature just as in a payment. If the bank refuses, it can prove that the coin was deposited or reclaimed before. Note that these *late deposits* have no precedence over reclaiming. Nevertheless, tills that can make online connections to the bank can also accept late payments by asking the bank for online confirmation that the coin was not deposited or reclaimed before.

Quick deposits. It can be helpful for organizational measures (see Section 5.1) if each coin has to be deposited within a short period after the actual payment, which can be long before $t_{payment\_expiry}$. If we want this period to be of length $t_{deposit}$, the protocols are modified as follows:

1. In a payment, the purse verifies that *Time*, as sent by the till, deviates from its own local time by at most Δ.
2. The payee has to deposit the coin before his own local time is $Time + t_{deposit}$. The bank only accepts a deposit if $Time \le t_{payment\_expiry}$ and before the bank's local time $Time + t_{deposit} + \Delta$. Note that this is not later than $t_{deposit\_expiry}$.

Blacklists. Both for loss tolerance and for cases where the user of a purse suspects that the PIN to his stolen purse is also known (or if no user identification is used in payments) it is useful if lost coins can be blacklisted. If we want to exploit blacklists for giving lost coins an expiry date $t_{deposit\_expiry}{}^* < t_{deposit\_expiry}$, we must assume that a blacklist can be distributed to all tills within a certain period of time. More precisely, we can proceed as follows:

SETTLING, BLACKLIST PART

- When the bank enters a coin into *double_spending_check_table* with *type = reclaimed* at its local time $t_{reclaim}$, it enters *m'* into a local list *blacklist_update* together with $t_{reclaim}$.
- . Furthermore, it enters $t_{deposit\_expiry}{}^* := t_{reclaim} + t_{deposit} + 2\Delta$ into the information attached to *m'* in *double_spending_check_table*.
- The bank can delete *m'* from *blacklist_update* at its local time $t_{deposit\_expiry}{}^*$.

DEPOSIT, BLACKLIST PART

- Each till has to contact the bank in intervals of at most $t_{deposit}$ (according to its local time) even if it did not receive payments, and deposit all payments received in that period.
- Whenever a till contacts a bank, the bank distributes *blacklist_update* to it in a signed message, including the bank's current local time t_{update}.[21] The till verifies the signature and that t_{update} deviates from its own local time by at most Δ. Then it joins *blacklist_update* with its current blacklist. It gives the bank a signed acknowledgment on *blacklist_update*.

[21] In reality, with more than one bank, an additional delay would occur in this step, just as in deposits, because the blacklists have to be distributed through the clearing infrastructure.

- If a deposited coin is found in *double_spending_check_table* with *type = reclaimed* and a certain value $t_{deposit\_expiry}$*, the bank only accepts this deposit if its local time is at most $t_{deposit\_expiry}$*.

PAYMENT, BLACKLIST PART

- After receiving *m'* in Step 1, the till looks it up in its current blacklist.
- In Step 2, the till also sends a signed message whether *m'* was in its blacklist or not. If it was, it aborts the payment.
- The till can delete the coins from its blacklist when its local time equals the original value $t_{payment\_expiry}$ of this coin type.

A dispute whether a payee unduly refuses to accept a coin can be settled as follows:

DISPUTES ABOUT BLACKLISTS

- The payee has to show the signature by the bank on a *blacklist_update* containing the disputed coin. The bank has to show the request for blacklisting, i.e., the signature by the new purse from RECLAIMING.

LEMMA 4: Assuming that the payment system was secure so far, it is still secure with the modifications for blacklisting. ◊

PROOF: What might be affected is primarily the payees' security, since the bank now refuses deposits under certain circumstances. Thus we show that any coin accepted by a till will also be accepted by the bank in the deposit. Let t_p be the till's local time of the payment and $t_{d,1}$ and $t_{d,2}$ the times of the deposits immediately before and after this payment, respectively. At t_p, the coin *m'* was not in the till's blacklist. If it had already been deleted there, the date $t_{payment\_expiry}$ would have passed, and the till would not have accepted the coin. Hence *m'* was not contained in any *blacklist_update* the till received until $t_{d,1}$. This means that the bank either (1) had not entered *m'* there until its local time $t_{d,1} - \Delta$, or (2) that it had entered and deleted *m'* between two successive deposits by this till. In the first case, $t_{reclaim} > t_{d,1} - \Delta$, if the coin is blacklisted at all. The bank's local time of the second deposit is at most $t_{d,2} + \Delta \leq t_{d,1} + t_{deposit} + \Delta < t_{reclaim} + t_{deposit} + 2\Delta = t_{deposit\_expiry}$*. Thus the bank does indeed accept the coin. The second case is impossible, since *m'* remains in *blacklist_update* for an interval of at least $t_{deposit} + 2\Delta$ according to the bank's local time, and thus $t_{deposit}$ in the till's time, and in any such interval, the till makes at least one deposit.

If we consider disputes about refused deposits (which were not described in the original payment system, nor in most other published ones, but should be included [PWP_90]), the bank can use the signed acknowledgment to prove that the payee knew in time that a coin was blacklisted.

The disputes about refused payments are secure (i.e., no one can blacklist coins of other payers) for the same reason that no one else can reclaim these coins, see the proofs of Requirement 2 in Theorems 1 and 2. ❑

5 Organizational Details

5.1 Preliminary Recovery

At least if there is no infrastructure for speedy distribution of blacklists, the time that passes between the loss of an electronic purse and the date $t_{deposit\_expiry}$, where the owner can irrevocably be given the remaining money back, may seem rather long. Thus the bank (or an insurance, but we briefly call any such organization "bank" again) may consider giving some money back to the user earlier, pending future settlement. We can give the following technical support for the necessary risk analysis.

Immediately after the loss. RECLAIMING can be completed at once. The resulting lists *Reclaimed_list*(Id_{Purse}) for all coin types yield an upper bound *max_lost* on the amount the user has lost. With the expected use of prepaid purses, this amount will not be very high on average. The bank may simply decide to give the user a loan up to *max_lost* until $t_{deposit\_expiry}$, and to take no interest or only a very low rate on that part that turns out to have been lost indeed. In particular, this is a reasonable service if the purse seems to have failed without fault of the user.

The bank may ask the user how much of the amount *max_lost* he thinks he has spent. If the purse was stolen or really lost, the unlocked amount must also be subtracted. Of course, only the estimates of honest users will be useful.

With quick deposits. If quick deposits are used (see Section 4.5), no deposit from payments before the loss can be made after the bank's local time $t_{quick} := t_{reclaim} + t_{deposit} + 2\Delta$, where $t_{reclaim}$ is the bank's local time when the coin was reclaimed.

- If the user can show at $t_{reclaim}$ that the purse is physically broken or stopped working, it is clear that he cannot cheat and continue to use it (if it is unclear, one can use a hammer). Thus the bank can irrevocably give the lost money back to the user at t_{quick}.
- If the user claims that the purse was lost or stolen, and if we trust the user identification of the purse (and, if unlocking is used, if the user is certain about the unlocked amount), deposits of the locked amount can only come in after t_{quick} if the user lied and continues to use the purse himself. Far fewer users will do this than wrongly remember how much money they spent since the last withdrawal. (Note that they are not anonymous here and that the lie can be proved.) Thus the bank may now decide to give a loan even to users who where not given a loan at $t_{reclaim}$.

5.2 Storing Decryption Keys

The decryption keys needed if backup information is stored at the bank must be generated by the user or the purse. They can be stored on electronic devices, on paper, or as passphrases.

1. Electronic devices would be memory cards, similar to backup purses. An advantage over storing the whole backup information on the backup purse is that the key only needs to be stored once, during initialization of the electronic purse. However, this solution may be

considered too expensive, and it presupposes that the electronic purse has an interface to a memory card.

2. A piece of paper can be seen as a very simple backup purse. The paper may be printed out, but at least the owners of lost purses will probably have to type the backup key into their new purse. Hence the key should be encoded with a code that detects at least one error and the output format should be user-friendly.

3. If the encryption system uses fairly short keys, one may consider encoding them in or deriving them from passphrases that the users can remember. (Some users might still write theirs down on paper.) If a user forgets the pass phrase, only his loss tolerance is lost. If he notices this, he can choose a new passphrase, and the purse should make a non-incremental backup at the next withdrawal, so that full loss tolerance is restored.

If someone else finds the electronic device or the piece of paper with the key, he can only use it in collusion with the bank, and only to violate privacy. (Passphrases or tamper-resistant electronic devices avoid this risk.) For users who worry about this, these solutions can be improved by secret sharing.

Note that none of these solutions really works if the electronic purse has no user interface, i.e., if it is just a smartcard (unless the user has some other device he trusts that can mediate this communication). The smallest evil in such a case would be special devices from independent sources, e.g., consumer organizations, that are placed at the bank and output pieces of paper.

Updating decryption keys. In Section 4.4, we assumed that the decryption key is constant over many withdrawals, which implied that the new purse must carry out the decryption. The alternative is to use a new key per coin and to give the bank only the decryption keys for the lost coins during RECLAIMING. Of course, this cannot be done by outputting new decryption keys to the user in each withdrawal, because that would destroy the advantages of storing the backup information at the bank. Instead, the purse updates the encryption key pseudo-randomly and the new purse recomputes the necessary values. This does not make much difference in the protocols from Section 4.4: A withdrawal order and the corresponding backup information still have to become valid in the same step, and thus the modified withdrawal is still needed. The new purse still needs a signed withdrawal order from the old purse containing the withdrawal sequence numbers of all coins on that purse after that withdrawal. It now recomputes the corresponding decryption keys and sends them to the bank. If a dispute arises where the user is not satisfied with the bank's decryption results, the bank has to show the signed ciphertexts, i.e., one returns to Step 3 of the RECLAIMING protocol of Section 4.4. Finally, the bank still verifies the decrypted coins as in Section 4.4 (even though it now decrypted them itself), because it neither trusts the user to supply the correct decryption key nor the old purse to supply correct ciphertexts.

6 Efficiency

The security of our loss tolerance measures was immediately considered after each option. We now show that the measures are reasonably efficient, too. More precisely, we show that (except for the optional blacklists in Section 4.5) all the complexity parameters are dominated by the complexity of the system without loss tolerance.

The storage space needed at the bank is dominated by *double_spending_check_table*, which was needed in the system without loss tolerance, too, and was only augmented by one bit per entry and an attribute Id_{Purse} in the entries for lost coins. The lists *Reclaimed_list*(Id_{Purse}) only contain lost coins, which should only be a small percentage of the coins in *double_spending_check_table*. If the backup information is stored at the bank, it is only needed for coins that were present at the last withdrawal of the respective purses, and these, too, are only a small percentage of the coins in *double_spending_check_table*. (Furthermore, even in the system without loss tolerance, the bank has to store at least the withdrawal orders and some information to solve disputes about a possible interruption of the last withdrawal.)

Storage space on backup purses is only needed for coins that were present at the last backup, and the information *reclaim_info* for each coin is less than what the main purse has to know about the coin. (However, one can optimize the representation on the main purse at the cost of additional recomputation more easily than on the backup purse.) Anyway, since the backup purse need not store any other data, in contrast to the main purse, it seems clear that a cheap backup purse will be sufficient.

The communication complexity of storing backup information (which is much more important than the protocols carried out after a loss only) on a backup purse is just one round with less data per coin than in a payment. If the backup information is stored at the bank, there is no new round of communication in the part carried out for each coin, and the additional data sent is much shorter than in a payment.

The time complexity of storing the backup information on backup purses (not considering the communication) is negligible. Storing backup information at the bank implies one encryption, which is certainly dominated by the rest of the computation if a fast symmetric cipher is available, and an additional signature per coin. If *sign* means, e.g., Schnorr signatures (which is reasonable since Protocol 1 is constructed from them) this again is dominated by the other exponentiations in a withdrawal.

The time and communication complexity of RECLAIMING is similar to that of the withdrawals of the lost coins, and that of SETTLING is dominated by the standard checks after deposits.

The optional blacklists described in Section 4.5 require additional storage in the tills and additional communication between tills and the bank. Whether this increased complexity pays off for the improved user friendliness cannot be decided in general.

7 Extensions

We now present briefly how the loss tolerance measures can be adapted to several variants of the payment systems, in particular the major known extensions to electronic coin systems.

7.1 Exploiting Pseudo-Randomness

In practice, one may let the purse generate the random numbers it needs in withdrawals pseudo-randomly only (i.e., s, u, v, y_1, and y_2). This is particularly reasonable if the backup information is stored at the bank, so that the untraceability is only computational anyway[22]. In this case, we can reduce the size of the backup information and the time needed to store it. We assume that the number w used by the bank is also generated pseudo-randomly. Hence all coins could be reconstructed given just the seeds (i.e., the secret initial values) of the two pseudorandom generators. However, these initial seeds must remain secret even after a loss. Several options are conceivable with decreasing size of the backup information and increasing risk if it falls into the wrong hands. We only present the details of a rather extreme case and with backup information stored at the bank. Furthermore, we first describe a protocol for the case where no tamper-resistance is used to prevent double-spending.

The format of *settling_info*, i.e., the information the bank finally needs about a coin, and the protocol SETTLING remain unchanged. The information the purse needs about a coin to be reclaimed becomes

$$reclaim\_info := (seed_{Purse}, distance_{Purse}),$$

where $seed_{Purse}$ is the seed that the purse used for its pseudorandom generator, and *distance* denotes the number of bits generated with the pseudorandom generator at the beginning of the withdrawal of this coin.[23]

MODIFIED INITIALIZATION

Instead of sending an encryption of $seed_{Purse}$ to the bank and outputting the decryption key to the user, the purse can directly output $seed_{Purse}$ to the user, who stores it secretly. (If the seeds used are much longer than the decryption keys one would have used, one can still do it in the old way.) Since at least the purse's key for signing withdrawal orders should not be endangered by the backup information, we assume that $seed_{Purse}$ is exclusively used for generating the numbers used in withdrawals. Thus, if there is only one real random seed in the purse at the beginning, one should first use it to generate several other seeds.

The bank stores its own seed $seed_{Bank}$ used for generating the numbers w.

MODIFIED WITHDRAWAL

1. The complete withdrawal still starts with a withdrawal order containing the withdrawal sequence numbers of the coins that will be on the purse after this withdrawal.
2. In Step 4 of the withdrawal of an individual coin, the purse computes
   $$ack := sign(seq\_no, a, b, c, distance_{Purse}),$$
   and sends the message
   $$(c, distance_{Purse}, ack).$$

---

22 The reasons are that a key for a one-time pad would be no easier to store than the backup information itself, and that partial information about the encrypted coins becomes known, since the bank sees them in cleartext after the deposits and may sometimes even know who made those payments.

23 If each withdrawal needs the same number of pseudorandom bits, *seq_no* can be stored instead of *distance*. However, this excludes trial-and-error subroutines and needs some care in combination with interruption tolerance.

3. In Step 5, the bank verifies the signature *ack*, computes *r*, stores $backup\_info_{Bank}$:= (*seq_no*, $distance_{Bank}$, *c*, $distance_{Purse}$, *ack*), where $distance_{Bank}$ indicates how far the bank's pseudorandom generator had been run before this withdrawal. Then it debits the account and sends *r* to the purse.

Reclaiming starts with the organizational measures from Section 2.3. Naturally, this time, the user inputs $seed_{Purse}$ instead of the decryption key. Then it proceeds as follows:

RECLAIMING

1. The bank sends the last withdrawal order from the lost purse. For each withdrawal sequence number *seq_no* listed in this withdrawal order, the bank looks up if the withdrawal *seq_no* took place and was not interrupted before the money was deducted from the user's account. If this is the case, the bank retrieves the value $backup\_info_{Bank}$ stored for *seq_no*. Using $seed_{Bank}$ and $distance_{Bank}$, it regenerates the value *w* it used in that withdrawal. Now it recomputes
$$reclaim\_info_{Bank} := (seq\_no, a, b, c, distance_{Purse}, ack, r)$$
and sends it to the purse. The bank also sends a signature on all this information.

2. The purse and the user in person verify that values $reclaim\_info_{Bank}$ for the correct sequence numbers have been sent just as in Section 4.4.

3. For each value $reclaim\_info_{Bank}$ it received, the new purse acts as follows:
   a) It verifies that the component *ack* is a valid signature with the key of the lost purse.
   b) It verifies that *r* is correct, just as in Step 5 of Protocol 1.
   c) From $seed_{Purse}$ and $distance_{Purse}$, it regenerates *s*, *u*, *v*, y_1, and y_2, and then recomputes *settling_info* = (*m'*, *z'*, *pk'*, *c'*, *r'*) just as in Protocol 1.
   d) It sends *settling_info* and *s* to the bank in a signed message.

4. The bank proceeds as in Section 4.4.

Under the usual cryptographic assumptions made about the security of the original payment system with pseudorandom number generation, the described loss tolerance measures fulfil the requirements from Section 1.3, except that someone who finds the value $seed_{Purse}$ can both break the untraceability of the owner of this purse and spend the coins that this owner has already withdrawn. A proof of this statement is very similar to the proofs of Theorems 1 and 2.

If tamper-resistance is used to prevent double-spending, the bank cannot allow $seed_{Purse}$ to become known to the user. In this case, we use secret sharing. The purse initially splits $seed_{Purse}$ into two pieces with $seed_1 \oplus seed_2 = seed_{Purse}$. It outputs $seed_1$ to the user. It signs $seed_2$ and sends it to the bank, encrypted with a public key of the bank. At the beginning of RECLAIMING, the new purse recombines $seed_{Purse}$ after verifying the signature on $seed_2$.

7.2 Wallets with Observers

Among the four ways of protecting the bank against double-spending, our measures so far were suitable for electronic purses with and without tamper-resistance. If payments are verified on-line, the protocols stay essentially the same, except that one can give the value of all reclaimed coins back at once, without the measures from Section 4.5, since payees will be notified that a coin was reclaimed in the online verification.

It remains to adapt the protocols to wallets with observers. The basic change in the payment system is usually (see the references in Section 1.2), and in particular in [Bran_94], that the secret key belonging to a coin (i.e., x_1, x_2, y_1, y_2 in the concrete example) is shared between the purse and the observer in such a way that only both of them together can sign in a payment. The till's part in a payment and the deposit are unchanged. In a withdrawal, the purse still learns all the public information about a coin, i.e., in our case *settling_info* = (m', z', pk', c', r'), or, more generally, (pk, sig), and it also knows s, u, and v, which we used as *reclaim_secrets* in the variants with strong loss tolerance. Hence essentially, the purse alone can store backup information just as in Section 4, and RECLAIMING and SETTLING remain unchanged.

7.3 Divisibility

Divisibility of coins was introduced in [OkOh_90] (and applied to the Brands scheme in [EnOk_94]). It means that a coin can be split into several pieces of smaller denominations. In all its current implementations, it is no problem for loss tolerance, since all the parts that a coin is divided into have the public part in common, i.e., in our case, they are linkable by the same value *settling_info*. The entries in *double_spending_check_table* now collect the individual pieces that a coin may have been divided up into, and only the part of a reclaimed coin that is unspent at the time $t_{deposit\_expiry}$ is given back.

7.4 Balance-and-Cheque Protocols

A measure that increases efficiency significantly at the cost of a significant reduction in security is described in [BoCh_90] (and applied to the protocols from [Bran_94] in [Brand_94]). The electronic cash is stored in two forms in the purses: a balance and so-called cheques. A cheque is constructed just like a coin, but it does not signify a fixed amount. Instead, the amount is signed when the cheque is spent (like *Time* and Id_{Payee}), and this amount is deducted from the balance. Of course, such protocols only make sense if the bank trusts the tamper-resistance of the electronic purses to a certain extent, because the balance must not be manipulated. We therefore assume that the bank also trusts the electronic purses in recovery.

The information input into SETTLING now has to consist of a value *settling_balance* of the balance and the values *settling_info* for all the cheques that the purse could have used to spend this amount. The cheques are handled in *double_spending_check_table* just as the coins were so far, but whenever a deposit of a reclaimed cheque is found, the amount is stored in *Reclaimed_list*(Id_{Purse}). At the time $t_{deposit\_expiry}$, all these amounts are subtracted from the original value *settling_balance*.

It is security-critical that exactly those cheques that the purse had not yet spent when *settling_balance* was recorded are entered into SETTLING: If cheques are entered that were spent earlier, the user loses money when they are deducted from the reclaimed amount, and if cheques are missing, the bank has to give money from *settling_balance* back although it was spent. We now show how this can be achieved in the case with backup information at the bank.[24]

MODIFIED WITHDRAWAL

- The withdrawal starts with a withdrawal order that contains the number *w_seq_no* of this withdrawal and the numbers *seq_no* of all the cheques that will be on the purse after this withdrawal.
- At the end of a complete withdrawal, the purse encrypts *settling_balance* and *settling_info* for each new cheque. It signs the ciphertexts, together with *w_seq_no* for the balance and the respective *seq_no* for each cheque, and sends all this to the bank.
- The bank verifies the signatures, stores the message, finally deducts the amount from the user's account, and sends a signed acknowledgment.
- The purse waits for the acknowledgment before carrying out payments. (Note that the bank now trusts the purse to wait indeed, in contrast even to model underlying the solution with additional tamper-resistance for a real electronic coin system.)

RECLAIMING

1. The bank sends the last withdrawal order from the lost purse and the corresponding signed ciphertexts, plus a signature by the bank.
2. The purse verifies the bank's signature and the signatures by the lost purse on the withdrawal order and the ciphertexts. It also verifies that the sequence numbers of the ciphertexts fit those in the withdrawal order. The user in person verifies now or later that no newer withdrawals with the lost purse turn up on the statements of account.
3. The purse decrypts all the ciphertexts and signs the message with all the resulting cleartexts.
4. The bank verifies the new purse's signature and enters *settling_balance* and the values *settling_info* into SETTLING.

Most aspects of the security of this protocol are quite easy to see, since all the information is signed by the purses whom everybody trusts for the recovery. Moreover, no additional information becomes public (in contrast to s in Section 4), so that no cryptographic lemmas are needed. An attack not prevented by the signatures is that the bank provides out-of-date backup information. However, this never yields a financial advantage for the bank: Changes to the balance during later withdrawals are cancelled out, since the bank cannot show them on the user's account when it denies those withdrawals. If further cheques were withdrawn, the bank loses all money spent with them. Furthermore, note that the withdrawal and the backup information become valid in the same step, as in Section 4.4, so that it is true that the bank must have backup information for every amount that it deducted from the account.

7.5 Interruption Tolerance

So far, we primarily considered losses between transactions (except for the withdrawals in Section 4.4, where interruption of the loss tolerance measures would have brought about a new

[24] Actually, there are some problems with backup purses for balance-and-cheque protocols, since storing the backup information is not integrated into the withdrawal.

security risk for the bank). Interruption tolerance is a different topic: Even in a payment system without loss tolerance in our sense, one needs additional measures to guarantee that all the other requirements are fulfilled even if transactions are interrupted maliciously or by communication or power failures. Since publications about highly secure payment system never get to this level of detail, we sketch the main points, but describing the full payment system with interruption tolerance would be a complete second paper. Naturally, the idea is to make the transactions atomic, i.e., to guarantee that they are either completed or undone, at least as far as the monetary results of the transactions are concerned. (In contrast, e.g., a pseudo-random generator may have to remain in a modified state after a canceled transaction.)

Subroutines on one device can be made atomic by (more or less) standard measures. A transaction between two devices, however, will always have at least one step where the devices cannot reach a consistent state if they are permanently interrupted in this step; this is known as the two-army problem. Of course, they should first try to retransmit. For a transaction with the bank, the next step should be to try to redo it over a different channel. (This is already nontrivial, in particular, if we require that a purse can be used for payments after an interrupted withdrawal.) If all else fails, one of the participants will have to apply to a third party to enforce a consistent state. A reasonable approach, where nobody has to contact anonymous participants, is as follows (similar to the solutions for the on-line case in [BüPf_89]).

- In payments, we provide one step where the till has obtained the money, i.e., enough information to carry out a deposit, but not yet sent a receipt. If the transaction is interrupted in this step, the purse contacts the bank (either by explicitly asking its user to do so or during the next withdrawal). Of course, the purse cannot distinguish this situation from an interruption one step earlier, i.e., before the till has obtained the money. Therefore, it asks the bank to trace this payment, giving the bank the information needed to do this. If the coin is found in *double_spending_check_table* as deposited in the possible period of time, the payee is forced to give a receipt or the bank gives one instead. If the coin is not found, the owner of the purse gets the money back.
- In withdrawals, we provide one step where the bank has already deducted the money from the account, but the purse has not yet obtained the coin (e.g., as in Section 4.4). If an interruption occurs here, the purse contacts the bank again (and, if necessary, a court), and the bank is forced to continue the withdrawal. This will usually imply that the individual messages of the withdrawal must be signed: For instance, the purse holder might try to trick the bank into signing a different coin in the repetition of the withdrawal (by sending two different values c in the concrete example), and thus the bank must be able to prove that the purse had already sent c in the interrupted withdrawal.
- In deposits and the transactions for recovery, it should also be the till or purse, respectively, that contacts the bank again or, if necessary, a court. Essentially, the transactions are repeated from the first unclear step, just like withdrawals.

7.6 Other Extensions

- **Different banks** are no problem for loss tolerance: If one implements untraceable electronic coin systems with several banks, one needs a clearing structure anyway, either to handle the tables *double_spending_check_table* for each individual coin type (which now includes the issuing bank) or to route deposited coins back to the issuing banks. Thus the settling protocols (Section 4.2) can remain essentially the same. The lists $Reclaimed\_list(Id_{Purse})$ would probably be held at the bank that issued the purse.
- **Accumulation.** A bank might decide to restrict double-spending checks to random samples of the coins, because it trusts tamper-resistance to some extent. In this case, loss tolerant coins would have to be a separate coin type, because they always have to be stored in *double_spending_check_table* after deposits, and the banks would probably charge extra for them.
- **Transferability.** Measures to make untraceable electronic coin systems transferable are described in [Antw_90, ChPe_93]. To make these systems loss tolerant, one has to combine the measures described in this paper with those described in [WaPf_90] for transferable electronic cash in tamper-resistant devices, in particular, Section 3.4.2 called “Marked Standard Values”. Note that the problem of receiving change (if one does not use divisible coins) is related to transferable cash, since the coins received as change should not only be usable in another payment, and not just in a deposit.

Of course, most of the extensions can also be combined.

7.7 Traceable Systems

Many offline payment systems do not offer untraceability at all, or only a very limited form. All these systems are balance-based. Loss tolerance for such systems can be achieved using the same ideas as for untraceable systems. Since systems with transferability were already discussed in detail in [WaPf_90], we only describe the less complicated, but more efficient special case without transferability.

Weak loss tolerance can be achieved using so-called mirror accounts: For each purse, the bank maintains a variable *mirror* that approximates the balance of this purse. It updates *mirror* during each withdrawal and whenever a till deposits a payment received from this purse. Therefore tills must store individual payment records and forward them to the bank (like coins or cheques in the untraceable systems). If the purse is lost, the bank waits until the purse becomes invalid and all its payments must have been deposited (determined in ways similar to Section 4.5). Then it gives the value of *mirror* back to the owner of the lost purse.

The total amount of information that must be processed by the bank is similar to that for the untraceable systems, but each payment record can be deleted as soon as *mirror* has been updated. Mirror accounts are already used for security checks in many proposed smartcard-based systems because of limited trust in tamper-resistance. Hence weak loss tolerance could easily be added to these systems.

Strong loss tolerance requires some extensions: The purse should sign the current balance in withdrawals and the payment records given to the tills in payments. The bank must store the last signed backup of the balance and all the payment records signed after that balance. If the purse is lost, the bank computes the money to be given back from these data. In a dispute, the bank has to show the signed backup of the balance and the signed payment records. It cannot gain anything by showing an outdated copy, since the omitted subsequent withdrawals cannot be shown on the statements of account. (The bank even loses if the new payments cannot be shown either since they indicate that another withdrawal took place in between.)

Now the individual payment records must be stored by the bank for some time.

8 Conclusion

We have shown how untraceable prepaid electronic cash systems based on the construction principle of electronic coins can be made strongly loss-tolerant, with detailed protocols for the example of the system from [Bran_94]. Loss tolerance removes an important disadvantage that these systems would otherwise have in comparison with credit systems. Some positive and negative aspects of our solutions are:

– The measures cannot be entirely transparent — a user at least has to store some information at the start of the system and to retrieve it after a loss of the electronic purse.

– The backup information for a certain user poses a certain risk to this user's untraceability, in particular in the cheaper solutions without additional tamper-resistant devices.

\+ All the other security properties need not be weakened.

\+ The efficiency of the systems is not significantly reduced.

Recall that our protocols are still somewhat idealized. For a concrete implementation, one has to work out the additional interruption tolerance measures sketched in Section 7.5, remove the simplifications in timing mentioned in Section 4.5, and fix precise message formats, in particular with respect to signatures (see the beginning of Section 4).

Note that the loss tolerance problem is no consequence of untraceability, i.e., even a completely traceable payment system would need loss tolerance measures. Moreover, loss tolerance generally requires the processing of individual payment records, and efficient loss tolerance in a system with transferability even seems to require electronic cash to consist of individual pieces [WaPf_90], independently of tamper-resistance and untraceability.

Acknowledgment

We thank the DFG (German Research Foundation) for financial support.

References

Ande3_94 Ross J. Anderson: Why Cryptosystems Fail; Communications of the ACM 37/11 (1994) 32-40.

Antw_90 C. J. van Antwerpen: Electronic Cash; Centre for Mathematics and Computer Science (CWI), Amsterdam, October 11, 1990.

BBCM1_94 Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjølsnes, Frank Muller, Torben Pedersen, Birgit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Vallée, Michael Waidner: The ESPRIT Project CAFE — High Security Digital Payment Systems; ESORICS 94, LNCS 875, Springer-Verlag, Berlin 1994, 217-230.

BoCh_90 Jurjen Bos, David Chaum: SmartCash: A Practical Electronic Payment System; CWI, Computer Science/Department of Algorithmics and Architecture, Report CS-R9035, Amsterdam, August 1990.

BoCP_88 Jurjen Bos, David Chaum, George Purdy: A Voting Scheme; unpublished manuscript, presented at the rump session of Crypto '88.

Bran_94 Stefan Brands: Untraceable Off-line Cash in Wallet with Observers; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 302-318.

Brand_94 Stefan Brands: Electronic Cash on the Internet; Proceedings of the Internet Society 1995, Symposium on Network and Distributed System Security, San Diego, California, February 16-17, 1995.

BüPf_89 Holger Bürk, Andreas Pfitzmann: Digital Payment Systems Enabling Security and Unobservability; Computers & Security 8/5 (1989) 399-416.

Cha8_85 David Chaum: Security without Identification: Transaction Systems to make Big Brother Obsolete; Communications of the ACM 28/10 (1985) 1030-1044.

Chau_83 David Chaum: Blind Signatures for Untraceable Payments; Crypto '82, Plenum Press, New York 1983, 199-203.

Chau_92 David Chaum: Achieving Electronic Privacy; Scientific American (August 1992) 96-101.

ChEG_88 David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf: An improved protocol for demonstrating possession of discrete logarithms and some generalization; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 127-141.

ChFN_90 David Chaum, Amos Fiat, Moni Naor: Untraceable Electronic Cash; Crypto '88, LNCS 403, Springer-Verlag, Berlin 1990, 319-327.

ChHP_92 David Chaum, Eugène van Heijst, Birgit Pfitzmann: Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer; Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, 470-484.

ChPe_93 David Chaum, Torben P. Pedersen: Transferred Cash Grows in Size; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 390-407.

ChPe1_93 David Chaum, Torben P. Pedersen: Wallet Databases with Observers; Crypto '92, LNCS 740, Springer-Verlag, Berlin 1993, 89-105.

CrPe_94 Ronald J. F. Cramer, Torben P. Pedersen: Improved Privacy in Wallets with Observers (Extended Abstract); Eurocrypt '93, LNCS 765, Springer-Verlag, Berlin 1994, 329-343.

Damg1_90 Ivan B. Damgård: Payment Systems and Credential Mechanisms with Provable Security Against Abuse by Individuals; Crypto '88, LNCS 403, Springer-Verlag, Berlin 1990, 328-335.

Deth_94 Jürgen Dethloff: 25 Jahre Chipkarten-Technik – Rückblick und Ausblick (in German); Proceedings 4. GMD-SmartCard Workshop (Bruno Struif, ed.), Feb. 8-9, 1994, GMD, Rheinstr. 75, D-64295 Darmstadt.

DSS_92 The Digital Signature Standard Proposed by NIST; Communications of the ACM 35/7 (1992) 36-40.

EnOk_94 Tony Eng, Tatsuaki Okamoto: Single-Term Divisible Electronic Coins; Pre-proceeding of Eurocrypt '94, May 9-12, 1994, Perugia, Italy, 311-223.

EvGY_83 Shimon Even, Oded Goldreich, Yacov Yacobi: Electronic wallet; Crypto '83, Plenum Press, New York 1984, 383-386.

Ferg_94 Niels Ferguson: Single Term Off-Line Coins; Eurocrypt '93, LNCS 765, Springer-Verlag, Berlin 1994, 318-328.

FiSh_87 Amos Fiat, Adi Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems; Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 186-194.

FrYu_93 Matthew Franklin, Moti Yung: Secure and Efficient Off-Line Digital Money; 20th ICALP, LNCS 700, Springer-Verlag, Heidelberg 1993, 265-276.

GoOr_94 Oded Goldreich, Yair Oren: Definitions and Properties of Zero-Knowledge Proof Systems; Journal of Cryptology 7/1 (1994) 1-32.

HePe_93 Eugène van Heyst, Torben P. Pedersen: How to make efficient Fail-stop signatures; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 366-377.

LeAn_90 P. A. Lee, T. Anderson: Fault Tolerance — Principles and Practice; 2nd rev. ed., Dependable Computing and Fault-Tolerant Systems Vol. 3, Springer-Verlag, Wien 1990.

OkOh_90 Tatsuaki Okamoto, Kazuo Ohta: Disposable zero-knowledge authentications and their applications to untraceable electronic cash; Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, 481-496.

Pede_92 Torben P. Pedersen: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing; Crypto '91, LNCS 576, Springer Verlag, Berlin 1992, 129-140.

PfWa2_92 Birgit Pfitzmann, Michael Waidner: How to Break and Repair a "Provably Secure" Untraceable Payment System; Crypto '91, LNCS 576, Springer Verlag, Berlin 1992, 338-350.

PWP_90 Birgit Pfitzmann, Michael Waidner, Andreas Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen (in German); Datenschutz und Datensicherung DuD 14/5-6 (1990) 243-253, 305-315.

Rolf_94 Richard Rolfe: Here Comes Electronic Cash; Credit Card Management Europe, January/February 1994, 16-20.

Schn5_91 Claus-Peter Schnorr: Efficient Signature Generation by Smart Cards; Journal of Cryptology 4/3 (1991) 161-174.

TAMc_89 Hardy Tichenor, Einar Asbo, Gretchen McCoy: Visa Super Smart Card Applications and Technology; SMART CARD 2000: The Future of IC Cards, Proceedings of the IFIP WG 11.6 International Conference; Laxenburg 1987, North-Holland, Amsterdam 1989, 35-44.

WaP1_87 Michael Waidner, Birgit Pfitzmann: Anonyme und verlusttolerante elektronische Brieftaschen (in German); Interner Bericht 1/87 der Fakultät für Informatik, Universität Karlsruhe 1987.

WaPf_90 Michael Waidner, Birgit Pfitzmann: Loss-Tolerance for Electronic Wallets; Proceedings 20th International Symposium on Fault-Tolerant Computing (FTCS 20), Newcastle upon Tyne (UK), 140-147.