Loss-Tolerance for Electronic Wallets

Michael Waidner, Birgit Pfitzmann

Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe,
Postfach 6980, D-7500 Karlsruhe 1, F. R. Germany
Phone: ++49-721-608-4024, Fax: ++49-721-608-4290, CSnet: Waidner@IRA.UKa.De

Abstract. If a payment system is to be used both over a communication network and off-line, *electronic wallets* are needed, i.e. tamper-resistant devices guaranteeing the correctness of digital payments.

Besides the trustworthiness of the physical security techniques, the main problem with such devices are complete failures or losses. These would usually imply a loss of money. On the assumption that ideally tamper-resistant devices exist, we discuss the possibilities for *loss tolerance* measures, taking into account the preservation of both the security of the payment system and the participants' privacy.

The only loss tolerance technique known from literature, the *simple distributed audit trail*, is shown to be insecure. However, we derive several classes of secure protocols for distributed auditing, above all the *distributed account list protocol* and the class of *marked standard value protocols*. Both can provide high fault tolerance and leave the security, the untraceability, and the ability for off-line payments of electronic wallets almost unchanged.

Keywords. Audit trail, digital payment system, electronic wallet, loss tolerance, privacy, tamper-resistant device.

1 Introduction

1.1 Why and when electronic wallets?

The old-fashioned **payment system using cash** has two important features: Each received piece of money can immediately be used for a new payment, without any communication with a third party, and payments are almost untraceable, i.e. the privacy of payers and payees is guaranteed.

The first feature is called **autonomy**, the second one **untraceability**. The **security** of this payment system depends on the hardness of making acceptable reproductions of banknotes.

Using *cryptographic techniques*, one can construct payment systems guaranteeing unconditional untraceability, and security based on complexity theoretical assumptions [Cha8_85, BüPf_86, Chau_89, PWP_87]. These assumptions are usually thought to be more likely than the assumption that forging banknotes is hard. Hence digital payment systems can be more secure and more untraceable than cash.

Transactions are performed by exchanging digital messages only, e.g. using a public communication network. But since digital messages are arbitrarily reproduceable, every payment must be confirmed by a third party. This third party must be involved in every transaction, i.e. digital payment systems cannot have the autonomy feature; they are necessarily on-line systems.

All actions of the payment protocol, i.e. storage of relevant information, generation and test of messages, are performed by small computers with built-in keyboards and displays. Each such computer manages the money of its owner.

It is natural to try to combine the advantages of these two types of payment systems, i.e. to aim for autonomous "digital" transactions. There are two possibilities:

Firstly, one can simply ignore the danger and permit autonomous payments [ChFN_88, CBHM1_89]. Obviously, it is impossible to prevent a dishonest payer from spending his money twice, but, using cryptographic techniques, it is possible to detect such double-spending afterwards and to prove it, e.g., to a court (whereas payments of honest participants remain untraceable). Since it is not *guaranteed* that detected swindlers can satisfy all their creditors, such a payment system is insecure in a certain sense. However, it is at least as secure as today's credit card systems.

Secondly, one can replace the confirming third party by the small computers themselves, i.e. one assumes that they do not permit their owners to double-spend their digital money. This idea, and the name **electronic wallet** for these computers, were proposed in [EvGY_83, Even_89]. There, untraceability is not considered; schemes guaranteeing untraceability were proposed in [BüPf_86, PWP_87].

1.2 Critical remarks on tamper-resistant devices

Obviously, electronic wallets must be **tamper-resistant** devices. This means:

- For reasons of security of the whole payment system, they must keep some of their stored data secret even from their owners (e.g. secret keys chosen by the bank), even if they are destroyed.
- For reasons of untraceability, they must keep their owners' payment behaviours and identities secret from all other parties (e.g. other participants, manufacturers, system manager).

The first point can only be ensured by physical measures (like the unforgeability of banknotes). Since security technique looks like a never-ending race between measurement engineers and security engineers, where no party definitely takes the lead, this assumption seems far less realistic to us than the complexity theoretical assumptions used for on-line payment systems. (In particular, we do not believe that today's chip-cards are really tamper-resistant; apart from the fact that they cannot serve as electronic wallets, because they have no keyboards and displays of their own, and therefore cannot ensure the security of their owners.)

The second point is a special case of the more general problem that electronic wallets must be implemented correctly in a very strong sense. In particular, they must not contain a Trojan Horse [PoK2_78], which, e.g., sometimes deletes or generates money, or leaks sensitive information to an unauthorized third party. Hence the design and implementation of the soft- and hardware of an electronic wallet must be verifiable. If the wallets are tamper-resistant, one cannot do so for a single wallet. Thus the only possibility is to design and manufacture electronic wallets under public control. For this it is also necessary to control the design and implementation of all tools used for designing and manufacturing an electronic wallet [Thom_84, Desm_87].

In the following, in spite of all our doubts, we assume the existence of ideally tamper-resistant and correctly implemented electronic wallets, and that their computational power and storage capacity is similar to that of a PC.

Every payment system using electronic wallets needs a secure (not necessarily public key) authentication scheme. Thus its security depends on some cryptographic assumptions, too. Especially for our loss-tolerant schemes, we assume the existence of a secure private key cryptosystem and a secure digital signature scheme.

But we explicitly recommend to use easier, more secure, and more untraceable on-line payment systems whenever possible!

1.3 Outline of the paper

Besides the trustworthiness of the physical security techniques, the main problem with electronic wallets is how to tolerate complete failures or losses. These would usually imply a loss of money.

In the following, we present some solutions to this **loss tolerance problem**.

All our loss tolerance measures modify a basic payment system sketched in §2.

In §3, we first show that the loss tolerance scheme suggested in [EvGY_83, Even_89], the **simple audit trail**, results in an insecure payment system, i.e. dishonest participants can use this scheme for frauds. We then present several new distributed audit trail schemes, some of them almost equally efficient as the simple audit trail, and prove that they preserve the security of the payment system. They provide high fault tolerance, and leave the autonomy of electronic wallets almost unchanged.

In §4, we present implementations of the proposed loss tolerance schemes which preserve the untraceability of the original payment system, too.

The following is described in more detail in [WaP1_87]. Parts of it were also presented in [WaPf_87, WaPf2_89]. For a survey of cryptography, see [Denn_82, Bras_88].

2 Payments using electronic wallets

2.1 Notations and definitions

For each device D, let P_D denote its possessor, i.e. the participant using D. Each device has only one possessor, but one participant can possess several devices. The amount of money stored by D is denoted by A_D and called the **account** of D.

A payment system is called …

….**secure**,
- if a device D changes A_D only if P_D has given a corresponding transaction order,
- the sum of all accounts A_D is invariant during transactions of devices, and
- if devices are lost, the sum of all accounts cannot be increased, even if the payment system offers fault tolerance measures.

….**untraceable**, if nobody obtains any additional information about the relation "P_D has paid amount a to $P_{D'}$ at time t" by observing the payment system.

….**autonomous**, if a device can perform an arbitrary sequence of transactions without having to communicate with parties other than its current transaction partner.

The manager of the payment system is called the **Bank**. Neither untraceability nor security should depend on the Bank's trustworthiness.

2.2 Basic payment system using electronic wallets

The **basic payment system**, the basis of all following loss tolerance measures, is similar to the original system in [EvGY_83, Even_89]. But since untraceability is not considered there, we sketch an equally efficient system according to a suggestion in [BüPf_86]. To understand the loss tolerance measures, a superficial description suffices.

The system consists of three sub-protocols, the **initialization** of a device, the **transaction protocol**, and the **termination** of a device.

During the initialization of a device D, both the Bank and P_D verify that D is one of the devices manufactured under public control (e.g., using a signature function known to these devices only).

The Bank provides D with a common secret key k for the mutual authentication of secure devices, and possibly a termination date, and P_D provides D with information for the mutual authentication of P_D and D. Finally, the account A_D is initialized.

For a transaction, two devices must be connected, either directly or via a network. Mutual authentication takes place between them (using k only), and between each of them and its possessor. The possessors enter the orders "pay a" and "receive a", respectively. The devices then exchange corresponding messages, and if the orders match, they change the accounts accordingly. To ensure the atomicity of a payment, a device which does not receive the last message enters a special state and must be shown to the Bank. (At previous points the transaction can just be aborted.)

At its termination date, D becomes "invalid"; it may only transfer A_D to a new device of P_D or a bank account.

We will assume that D and the Bank use the normal transaction protocol to initialize A_D and to transfer A_D after termination.

The payment system is *untraceable*, since neither the participants nor their devices are identified during a transaction. (But of course the payment system cannot hide information which is known from other sources. For example, if payer and payee are connected via a network which does not guarantee untraceability (cf. §4), the parties can be identified by observing the network.)

3 Loss tolerance

3.1 Overview

Obviously, in the basic payment system in §2, participants P_D who lose their devices D also lose their amounts A_D.

The same happens if D becomes faulty: To avoid insecure behaviour of faulty devices (e.g., multiple spending of the same money), and to make tampering as difficult as possible, the devices must show fail-stop behaviour: as soon as D detects a fault, or that someone tampers with it, it deletes all secret information and never interacts with its environment again. Again P_D loses A_D. The most natural way of dealing with this problem is to apply standard techniques of fault tolerance in designing the devices. An approach on a higher level would be, e.g., to use a 2-out-of-3 system, instead of a single device D. Thus the failure of one device could be tolerated.

Real *losses*, however, cannot be tolerated by any of these techniques: Fault-tolerant design cannot prevent a participant from being forgetful, and in the example of a 2-out-of-3 system, the losses of the devices are not statistically independent, since at least 2 together are needed for transactions. Thus fault-tolerant design of devices is not further considered here.

Therefore, assuming that a participant P_D has definitely lost the control of the device D, we define the payment system to be …

… **loss-tolerant**,
- if $A_D \geq 0$ always holds (i.e. P_D cannot lose more money than in the system without loss tolerance), and
- if there is a nonzero probability that the monetary loss of P_D is less than A_D.

Loss tolerance can only be achieved by recovery schemes. Because of autonomy, the only way to collect recovery information is to use **distributed audit trail** schemes, i.e. every transaction is used to exchange some recovery information. The development of such schemes is the main subject of this paper.

Obviously, a loss tolerance scheme is only useful if it **preserves the security** of the basic payment system.

An additional, although less important, security problem is posed for the owners of lost devices: They must be sure that the recovering party does not embezzle the money of lost devices. This is called **security against recovery fraud**.

In §3.2, we repeat the simple audit trail scheme of [EvGY_83, Even_89], and demonstrate that it does not preserve the security of the payment system. In §3.3, we describe a class of secure extensions of the simple audit trail, called passing-on protocols. They are rather inefficient, but in §3.4, we use them to derive two efficient loss tolerance schemes, the distributed account list protocol and the marked standard value protocol.

Remember that considering the untraceability of payments is postponed to §4.

3.2 Simple audit trail and its insecurity

The simple audit trail is a very common fault tolerance scheme. As a loss tolerance measure for electronic wallets it is proposed in [EvGY_83, Even_89]. In this section, we describe it in more detail and show why it destroys the security of the basic payment system. (The insecurity does not depend on untraceability.)

The goal is to guarantee that a transaction is valid for both partners if at least one partner survives. Thus each partner must store an entry which unambiguously describes the transaction.

Since they are necessary for the following security preserving protocols, too, we introduce some notations:

If a transaction T is a payment of an amount $a > 0$ from $D1$ to $D2$, we call it a **payment** from $D1$'s point of view, and a **receipt** from $D2$'s point of view.

Each of $D1$ and $D2$ describes T by an **entry**: Each of them chooses a unique transaction number T_{D1} and T_{D2}, resp., and they form the **payment entry** ($D1$, $-a$, T_{D1}) and the **receipt entry** ($D2$, $+a$, T_{D2}), respectively.

If the simple audit trail is added to the basic scheme of §2, during each transaction T, $D1$ stores the receipt entry of $D2$, and $D2$ stores the payment entry of $D1$ (simultaneously with changing the accounts). After termination, a device D sends the list L_D of all stored entries to the Bank.

The *recovery* protocol is very simple: For each lost device D', the Bank collects all entries (D', a, $T_{D'}$), and defines $A_{D'}$ to be the sum of the initial value of $A_{D'}$ plus the amounts of all entries found.

The *insecurity of the simple audit trail* is shown by the following example (Fig. 1):

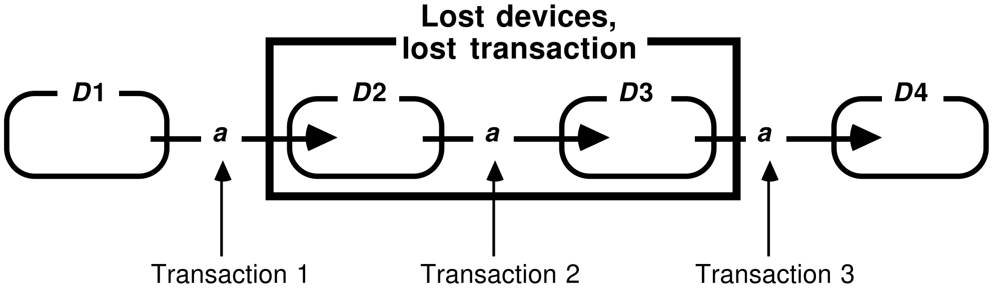

Figure 1 Example of the insecurity of the simple audit trail

Assume there are 4 devices $D1$, $D2$, $D3$, $D4$, and that, after initialization,

- $A_{D1} = a,\ a > 0$
- $A_{D2} = A_{D3} = A_{D4} = 0.$

In three subsequent transactions, Di pays a to $D(i+1)$, $i = 1, 2, 3$. Afterwards, $D2$ and $D3$ get lost. According to the recovery information stored in $D1$ and $D4$, the accounts of the lost devices are restored to

- $A_{D2} = a$
- $A_{D3} = -a$.

Obviously, this is an *impossible* result, since the basic payment system guarantees $A_{D3} \geq 0$. It also contradicts the definition of loss tolerance, because participants can lose more money by the recovery procedure than they had stored in their lost devices. The natural idea to define $A_{D3} := 0$ in such cases is also unsuitable, since then, the sum of all accounts is increased. This is absolutely intolerable, too, since analogous situations can easily be generated by dishonest participants.

Hence the simple audit trail does *not* preserve security and must not be used for loss tolerance!

3.3 Passing-on protocols

The insecurity of the simple audit trail is caused by the fact that a payment can be restored without restoring the receipt of the money used for the payment.

This can be avoided by passing on entries about previous transactions:

For example, in every transaction the devices could exchange and store *all* entries known to them, including the two new ones. In the example of §3.2, $D4$ would store the entries about each of the 4 transactions. The Bank would therefore correctly restore $A_{D2} = A_{D3} = 0$.

This example is called the **maximum passing-on** protocol. It *preserves the security* of the payment system:

- The two entries about a transaction are always stored together. This ensures that the sum of all A_D is invariant.
- Entries about payments of D are only stored together with all previous receipts of D. Thus after recovery, $A_D \geq 0$ holds.

The only *security against recovery fraud* guaranteed to an owner of a lost device D is $A_D \geq 0$. To prevent the Bank from forging entries about payments of D, D can sign each entry using a signature function chosen by P_D. To prevent the Bank from forgetting entries about receipts of D, the recovery procedure can be performed simultaneously by several mutually independent organizations.

The protocol guarantees *maximum loss tolerance*, since each device which has a physical chance to hear something about a previous transaction really stores the entries about it.

On the other hand, it causes *maximum storage expenditure* in the devices, and at present, this would be too much for realistic applications.

One way to reduce the storage expenditure is to check which entries must necessarily be passed on. This leads to the **general passing-on protocols with cover functions**, which are only sketched here, since we derive more efficient protocols in the following:

Considering the analogy of electronic and traditional wallets, it is intuitively obvious that each payment T of D must be *covered* by a set $\mu(T)$ of previous receipts. A function μ with suitable characteristics is called cover function. One then sees that security is preserved if

- each device which stores an entry about a payment T also stores all the entries about the receipts $\mu(T)$, and
- the two entries about a transaction are always stored together.

One can easily form effective protocols which guarantee these conditions.

The only critical performance parameter is the *storage* needed for D. It is proportional to $|L_D|$, the number of entries stored in D. This depends on μ and the participants' payment behaviour. Since every passing-on protocol includes the simple audit trail, $|L_D|$ is at least the number of transactions of D. An upper bound is given by the maximum passing-on protocol, where $|L_D|$ can be the number of all transactions made by all devices. This upper bound is hard for *all* passing-on protocols [WaP1_87] (although only in rather special cases of payment behaviour). Thus these protocols are still not very practicable.

Simultaneous termination of all devices of the payment system is necessary for the security of any sensible protocol based on passing on entries (not only for those just described) [WaP1_87].

3.4 Efficient derivates of the passing-on protocols

In this section, we present two loss tolerance schemes which are efficient enough to be applicable, i.e. our goal is to decrease the storage requirements for devices as close as possible to the lower bound given by the simple audit trail.

3.4.1 Distributed account list

Consider the maximum passing-on protocol of §3.3.

If D' contains an entry about a transaction of D, it also contains entries about *all* previous transactions of D. Thus for restoring A_D, one could just ask the device D' which contains the most recent entry about D to *compute* the value of A_D. To answer this question, D' need not know each single entry about D; it suffices if D' knows the sum of these entries, i.e. the latest value of A_D.

Hence we replace the entries about transactions of D by pairs, consisting of a value of A_D and a transaction number T_D which allows D' to determine which of two values of A_D is more recent. Each device D' need only store the most recent value of A_D it knows.

The *preservation of security* and the *maximum loss tolerance* of this **distributed account list** protocol follow from the corresponding statements about the maximum passing-on protocol. The *security against recovery fraud* can be achieved as in §3.3.

The upper bound on the *storage* capacity of a device is decreased to the total number of devices. The computational expenditure of the transaction protocol and the recovery procedure are decreased, too.

If, e.g., each entry consists of 32 bits (= sequence number & value of A_D), and there are 100 000 devices, each device needs about 3 Mbit memory. However, if electronic wallets were introduced nation-wide, and not combined with an on-line payment system, at present the storage requirements could be too large.

3.4.2 Marked standard value transactions

In this section, we present a loss tolerance protocol with even smaller storage requirements.

It can also be derived from a restriction of a passing-on protocol to transactions with standard values (which already has some advantages over a general passing-on protocol). That approach is more systematic and avoids a new security proof, but here we choose the shorter direct presentation.

The protocol works like a system with individual pieces of money which are of certain standard values, say 1, 2, 4, …, 2^ν. Each piece is called a **marked standard value** (**MSV**) and identified by a **standard value number** N, which also denotes its denomination.

Each device D contains a **possession list** PL_D and a **payment list** QL_D. PL_D contains entries about the MSVs D possesses. Each entry in PL_D is a pair, consisting of a standard value number N and a counter z which denotes how often N had been transferred before D received it.

If D uses N for a payment to D', it deletes the entry (N, z) from PL_D and enters the payment entry (D', N, z) in QL_D, and D' enters $(N, z + 1)$ in $PL_{D'}$. After termination, each device transfers the remaining MSVs and QL_D to the Bank.

The *recovery procedure* can be carried out for each MSV N individually: If no device has transferred N to the Bank, N must have been lost in a lost device. In this case, the Bank searches for the newest payment entry about N in all lists QL_D, i.e. an entry (D', N, z) with maximal z. The device D', which is necessarily lost, receives N.

One easily sees the *preservation of security*: The only thing that can happen to lost devices is that they receive MSVs, hence $A_D \geq 0$ always holds. Since each MSV is in the possession of exactly one device (lost or not) at the end, the sum of all accounts cannot be increased.

Unfortunately, the *security against recovery fraud* cannot be achieved as for passing-on protocols, since not only entries constructed and signed by D are important for the recovery of D. (For example, if D was the last possessor of N, proved by the entry (D, N, z), the Bank can invent a new entry (Bank, N, z') with $z' > z$.)

This problem can only be solved if only secure devices can perform the transaction protocol, i.e. if even the Bank must use such devices for its transactions: then all secure devices can authenticate entries with a common signature function which they also need during initialization (see §2.2). The embezzlement of entries can be avoided as for passing-on protocols.

Obviously, the restriction to standard values causes the well-known **money-change problem**, i.e. the *autonomy* of the payment system is partly lost, because devices must sometimes communicate with the Bank to change their MSVs.

This problem is solved if standard values may be split, e.g. a value 2^k into two values 2^{k-1}. For an MSV N, each "piece" of N is denoted by a pair (N,α), where α is a binary string (denoting the position of (N,α) in the "splitting-tree"). (N,ε) denotes the original, unsplit MSV. (N,α) can be split into two pieces $(N,\alpha 0)$ and $(N,\alpha 1)$ of equal value. Still just one MSV is needed to cover each single payment. The corresponding change in the recovery procedure is straightforward. The essential statements on loss tolerance and storage requirement are unchanged. However, splitting should be handled with care, since otherwise, one might end up with pieces of the value 1 only. We will not consider splitting further in the following.

The *loss tolerance* of the MSV protocol is clear from the recovery procedure: D receives N during the recovery protocol iff the device from which D had received N has survived, and all devices which possessed N after D are lost, too.

But loss tolerance can be increased arbitrarily if the devices store not only their own payment entries, but also entries about payments of other devices (**MSV protocol with additional passing-on**). For this, each device has an optional list OL_D. The passing-on of entries can be regulated arbitrarily, and each entry in OL_D can be deleted if memory becomes scarce, without loss of security.

The most important advantage of the MSV protocol is that a device only needs *storage* proportional to the number of its own transactions. This is about $v/2$ times more than what is needed for the simple audit trail, since only standard values are allowed.

The complexity of the recovery procedure is similar to that of the simple audit trail (but again, there are more transactions). Rather exact performance characteristics can be found in [WaP1_87].

In the MSV protocol with additional passing-on, the performance parameters should be multiplied by the number of passings-on of each entry.

In any case, since the expenditure of the MSV protocol is comparable to that of the simple audit trail, it seems applicable in real life (provided one finds ideally tamper-resistent devices!).

In contrast to the previous protocols, in the MSV protocol *asynchronous termination* and recovery of devices are possible, if for each MSV a termination date is chosen.

This is based on the observation that each MSV defines its own **small payment system**: Assume that each device is valid for a period p, and each MSV for a period p'. The small system of N is defined by those devices which might have possessed N (cf. Fig. 2) and the Bank.

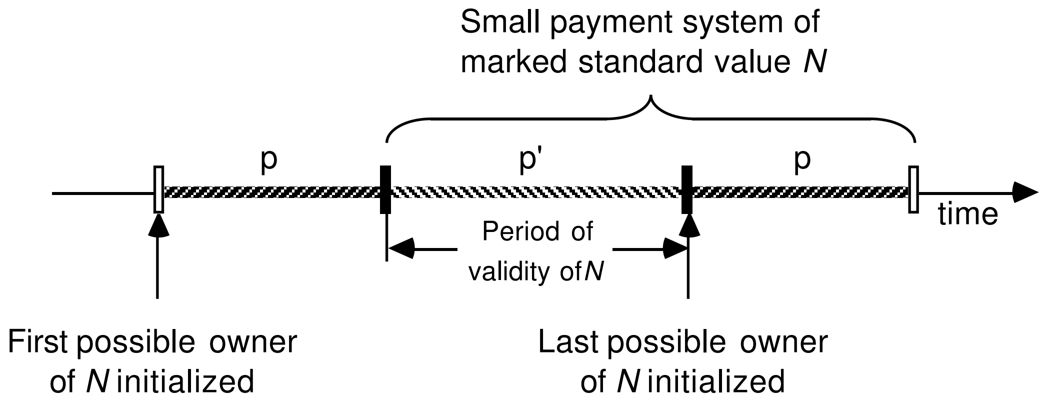

Figure 2 Small payment system of MSV N.

The small system of N can be recovered (i.e. N given to some device) independently of all other small systems.

The account A_D of a lost device D can be restored as soon as all small systems D has participated in can be recovered (Fig. 3). (In real life, P_D must wait a little longer, due to the expected deviations of local clocks).

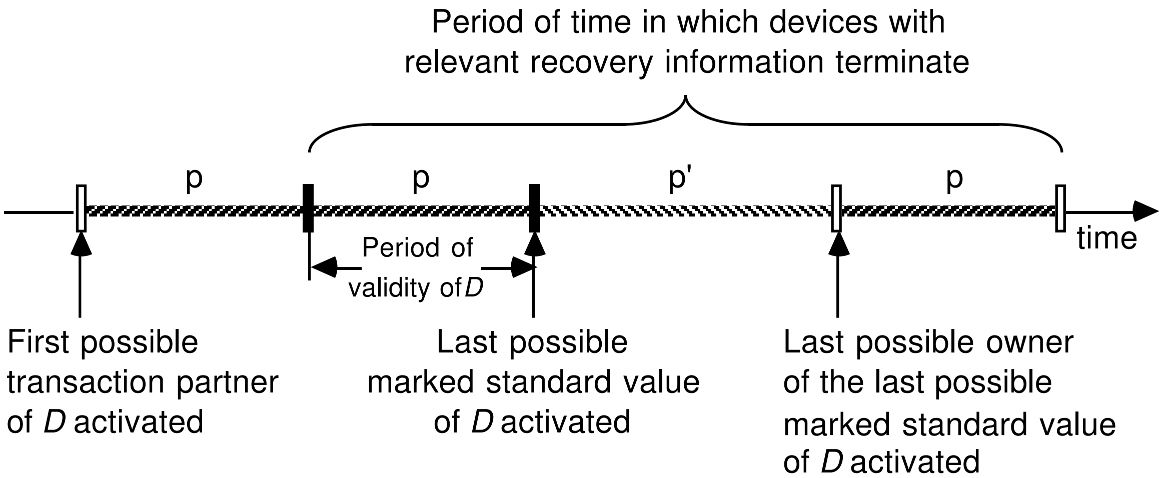

Figure 3 Period of time where devices with information relevant for the restoration of A_D terminate.

4 Combining loss tolerance and untraceability

In §4.1, we discuss how much untraceability can be expected, and what assumptions must be made. In §4.2, we present basic measures to achieve untraceability, as far as they are the same for all loss tolerance schemes. In §4.3-4.4, we describe untraceable versions of the distributed account list protocol and the MSV protocol, respectively.

4.1 Models of untraceability

All loss tolerance schemes described in §3 are based on the fact that a restoring party can redo transactions of lost devices. Therefore this party must receive as much information about these transactions as possible.

Thus we think the transactions of *lost* devices must be traceable by the restoring party by definition. Our goal is to make the payment behaviour of the other devices untraceable.

In all schemes of §3, the restoring party receives all entries, not just those about lost devices.

If one assumes that the recovery procedure is performed by a *secure recovery device*, this does not harm untraceability. But this assumption is considerably more unrealistic than the assumption that the electronic wallets are secure, since such a secure recovery device would be more complex, would need more memory, and should be more reliable.

Therefore, in the rest of this chapter, we will consider the case where the recovery procedure is performed by *normal computers*.

As usual with untraceability, it is assumed that every party is a potential attacker, i.e. might try to trace payments, and that all actual attackers collaborate. However, it is assumed that nobody can break the tamper-resistance of the secure devices, or the cryptosystems used. For simplicity, we assume that there is just one restoring party, and that this is the Bank.

At some points, we will consider different cases of how the Bank performs the payment protocol:

Case 1: The Bank, too, needs secure devices to perform the payment protocol.

Case 2: Since the Bank must perform more transactions than usual participants, it is allowed to perform the payment protocol using normal computers.

Case 2 can be split further:

Case 2a: In the transaction protocol, a secure device can distinguish whether its partner is another secure device or the Bank (e.g., using the same procedure as during initialization).

Case 2b: The Bank can act exactly like the secure devices.

These cases can also be seen as different assumptions on the attacker. Case 1 corresponds to the strong assumption that all devices taking part in payments are secure from the attacker; Case 2b corresponds to the weaker (and therefore nicer) assumption that participants need only trust their own devices.

(Note that for security, we normally assumed Case 1. However, Case 2 is not unrealistic, because the Bank could only endanger security by increasing the amount of money in the system, not by taking money away from individual participants. If the Bank itself is liable for this increase, it has no advantage from this attack.)

In Case 1 and 2a, we assume that all communication between secure devices is encrypted. Hence outsiders need not be considered.

In all these cases, if one uses a scheme of §3 directly, untraceability is completely lost.

4.2 How to hide unnecessary information

There are two sources of information for attackers:

- the entries themselves, and
- the way the attacker receives them.

The importance of the first source is obvious.

A simple example shows that the second one also endangers untraceability: If the entry about a transaction T of a lost device D is stored by just one device D', and if the Bank knows that it receives this entry from D', the Bank knows that, with high probability, T was a transaction between D and D'. This happens with some transactions, in particular late ones, in the passing-on protocols and the distributed account list protocol (§3.3, 3.4.1), and with all transactions in the simple MSV protocol (§3.4.2).

4.2.1 Encrypting entries

In principle, the first source of information is closed by encrypting the entries about a device D before showing them to a potential attacker. (Depending on the case and the loss tolerance scheme, this may happen during the recovery protocol only or during every payment.)

For this, a key k_D is used which is given to D by P_D during initialization.

For recovery, if D is lost, P_D sends k_D to the Bank, which can then read all relevant information. (It must check that it does not receive a key twice, but not that the keys are correct.) Now the recovery procedure proceeds as in §3.

To prevent the Bank from checking how often it received an entry, the encryption algorithm should be indeterministic.

To make the recovery procedure effective, once the Bank knows k_D, it must be able to find the entries about D, i.e. to decide whether a ciphertext was produced using k_D or not. For this, one can add some redundancy to the cleartext. Instead, if the entries are already digitally signed (which was a measure against recovery fraud in some of the cases), one can check whether a decrypted entry is correctly signed (but this is less efficient). In the following, for simplicity, neither the redundancy nor the signatures are shown. Note that the security is exactly that of the corresponding schemes in §3; the additional encryption has no authentication purpose.

(The recovery procedure can be sped up by prefixing the encrypted entries with pseudo-randomly chosen labels of the devices [WaP1_87].)

4.2.2 Using an untraceable network

The second source is almost closed by using an **untraceable network**, i.e. a network which allows its participants to send and receive messages anonymously, at least for the communication between the secure devices and the Bank during the termination protocol.

For a survey of such networks, see [PfPW_88, PfWa_86, WaPf_89]. (If untraceable payments are to be made not only off-line, but also over a network, one needs an untraceable network anyway, cf. §2.2)

To prevent the Bank from guessing who is sending an entry, for each terminating device D, a sufficiently large set of devices should terminate simultaneously with D.

Whether a device must send its list of entries in one piece, or whether each entry can be sent alone, depends on the loss tolerance scheme.

4.3 Untraceability for the distributed account list protocol

In the distributed account list protocol, each device must be able to decide which entries (= accounts) belong to the same device and which of two entries about a device is more recent. Thus the devices must see the entries in an unencrypted form. Hence for complete untraceability, all secure devices must be trustworthy, i.e. Case 1 or 2a must be assumed.

Among the secure devices, instead of the "name" D, the key k_D is used to identify D, i.e. entries are of the form (k_D, A_D, T_D), instead of (D, A_D, T_D).

If two secure devices communicate, the rest of the transaction protocol remains unchanged.

Otherwise, instead of forwarding an entry (k_D, A_D, T_D) to an insecure device, D' encrypts it as $k_D(A_D, T_D)$.

Apparently, the information which entries are stored together by a single device cannot be hidden: To preserve security, the Bank must either use a list $L_{D'}$ completely or ignore it. Therefore, D' must send it over the untraceable network in one piece, since otherwise, some of the entries could be intercepted and deleted.

4.4 Untraceability for the marked standard value protocol

The simple MSV protocol can be made untraceable in all cases: Each entry (D, N, z) is only passed on as $k_D(N, z)$ by D . The same holds for the MSV protocol with additional passing-on, if each device passes on entries about itself only.

If other devices, too, pass on entries about D, we must assume Case 1 and 2a. Then, as in §4.3, if device D' communicates with a secure device, the "name" D is replaced by k_D, and if D' communicates with an insecure device, an entry (k_D, N, z) is encrypted as $k_D(N, z)$.

(Some untraceability can also be achieved in Case 2b [WaP1_87].)

In contrast to §4.3, each entry can be sent to the Bank independently, without loss of security. Thus untraceability is perfect.

5 Summary

Assuming the existence of tamper-resistant devices with computational power and storage capacity similar to PCs and secure cryptosystems, we have presented loss tolerance schemes which leave the security, the autonomy, and the untraceability of the basic payment system with electronic wallets (§2.2) almost unchanged: the *distributed account list* protocol (§3.4.1, §4.3) and the *marked standard value (MSV)* protocol (§3.4.2, §4.4).

The two schemes are compared in Figure 4.

Protocol	Distributed account list protocol	MSV protocol
Autonomy	unrestricted	slightly restricted (money change problem)
Security	guaranteed	
Security against recovery fraud	can be made arbitrarily high	
Untraceability	can almost be guaranteed	can be guaranteed
Loss tolerance	maximum possible loss tolerance	can be made arbitrarily high
Storage capacity needed in each secure device	worst case: ≈ number of devices average case: unknown	≈ number of transactions • number of passings-on of each entry
Termination of devices, waiting time until recovery	simultaneous termination and recovery of lost devices	asynchronous termination, waiting time about 3 • period of the validity of a device **or** as in the distributed account list protocol
Recovery expenditure	acceptable	

Figure 4 Summary

More important than loss tolerance is the problem how really secure tamper-resistant devices can be constructed. Today's chipcards seem unsuitable for this purpose, since their tamper-resistance is doubtful, and their computational power and storage capacity is too small.

As long as this problem is not solved, we cannot recommend to use a payment system based on secure devices, since at present, such a system seems unnecessarily insecure.

Acknowledgements

We are pleased to thank Birgit Baum-Waidner, Klaus Echtle, Winfried Görke, Andreas Pfitzmann, and Karl Rihaczek for helpful critique and stimulating discussions, and the German Science Foundation (DFG) for financial support.

6 Literature

Bras_88 G. Brassard: Modern Cryptology; LNCS 325, Springer-Verlag, Berlin 1988.

BüPf_86 H. Bürk, A. Pfitzmann: Digital Payment Systems Enabling Security and Unobservability; Computers & Security 8/5 (1989) 399-416.

CBHM1_89 D. Chaum, B. den Boer, E. van Heyst, S. Mjølsnes, A. Steenbeek: Efficient Electronic Checks; will be published in: Eurocrypt '89, LNCS, Springer-Verlag, Berlin 1990.

Cha8_85 D. Chaum: Security without Identification: Transaction Systems to make Big Brother Obsolete; CACM 28/10 (1985) 1030-1044.

Chau_89 D. Chaum: Privacy Protected Payments – Unconditioned Payer and/or Payee Untraceability; Smart Card 2000, Proc., North Holland, 1989, 69-93.

ChFN_88 D. Chaum, A. Fiat, M. Naor: Untraceable Electronic Cash; Crypto '88, LNCS 403, Springer-Verlag, Berlin 1990, 319-327

Denn_82 D. Denning: Cryptography and Data Security; Addison-Wesley, Reading 1982, 1983.

Desm_87 Y. Desmedt: Is there an ultimate use of cryptography? (Extended abstract); Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 459-463.

Even_89 S. Even: Secure Off-Line Electronic Fund Transfer Between Nontrusting Parties; Smart Card 2000, Proc., North Holland, 1989, 57-66.

EvGY_83 S. Even, O. Goldreich, Y. Yacobi: Electronic wallet; Crypto '83, Plenum Press, New York 1984, 383-386.

PfPW_88 A. Pfitzmann, B. Pfitzmann, M. Waidner: Datenschutz garantierende offene Kommunikationsnetze; DuD /3 (1986) 178-191; extended version: Informatik-Spektrum 11/3 (1988) 118-142.

PfWa_86 A. Pfitzmann, M. Waidner: Networks without user observability; Eurocrypt '85, LNCS 219, Springer-Verlag, Heidelberg 1986, 245-253; revised version: Computers & Security 6/2 (1987) 158-166.

PoK2_78 G. J. Popek, C. S. Kline: Issues in Kernel Design; in: R. Bayer, R. M. Graham, G. Seegmüller (eds.): Operating Systems, An Advanced Course; LNCS 60, Springer-Verlag, Berlin 1978; reprinted: Study Edition, Springer-Verlag, Heidelberg 1979, 209-227.

PWP_87 B. Pfitzmann, M. Waidner, A. Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen; Computer und Recht 3/10,11,12 (1987) 712-717, 796-803, 898-904.

Thom_84 K. Thompson: Reflections on Trusting Trust; Communications of the ACM 27/8 (1984) 761-763.

WaP1_87 M. Waidner, B. Pfitzmann: Anonyme und verlusttolerante elektronische Brieftaschen; Interner Bericht 1/87 der Fakultät für Informatik, Universität Karlsruhe 1987.

WaPf2_89 M. Waidner, B. Pfitzmann: Loss-tolerant Electronic Wallet; Smart Card 2000, Amsterdam, 4.-6. October 1989.

WaPf_87 M. Waidner, B. Pfitzmann: Verlusttolerante elektronische Brieftaschen; 3rd International Conference on Fault-Tolerant Computing-Systems, IFB 147, Springer-Verlag, Berlin 1987, 36-50; revised version: DuD /10 (1987) 487-497.

WaPf_89 M. Waidner, B. Pfitzmann: Unconditional Sender and Recipient Untraceability in spite of Active Attacks – Some Remarks; Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 5/89, March 1989; revised version will be published in: Eurocrypt '89, LNCS, Springer-Verlag, Berlin 1990.