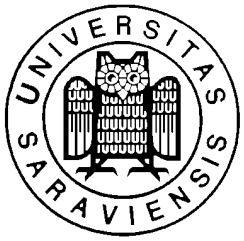
UNIVERSITÄT DES SAARLANDES
FACHBEREICH 14, INFORMATIK



Payment Systems

(Ch. 17 of Course Material "Security")

Birgit Pfitzmann

Dept. of Computer Science
Saarland University
pfitzmann@cs.uni-sb.de

Version Feb. 2, 2000

Contents

17 Payment Systems*

In contrast to most of the system classes in Chapters 2 to 6, there is in principle an enormous number of subclasses of payment systems with somewhat different properties. I.e., even the definitions, if there were any, would be different, not just the degrees of security and the realizations.

On the other hand, most of the systems that are currently known in practice (e.g., have at least been used in a field trial) come from very few of these classes. Moreover, you sometimes only need to change the actual implementations a little bit to get a different class. Hence there is a certain temptation to present only the most usual systems. But as you can also find those in about one new book every two weeks, I decided that a university course should try to look at a somewhat wider picture.

17.1 Properties

We first consider the functionality of payment systems, i.e., what you can do with them (similar to first only showing the algorithms of cryptographic systems in Chapter 2, before making more precise definitions of the security). Afterwards, we first consider security against fraud, i.e., integrity and to some extent availability goals, and finally privacy, i.e., confidentiality goals.

17.1.1 Functionality

A. Flow Models

A standard way to give an overview of payment systems is shown in Figure 17.1 (see, e.g., [BüPf_89, AJSW_97]). These figures give a good first impression if one does not question the meaning of the arrows in them too much:

A. A cash-like system is a bit like real cash, i.e., coins and bank notes:
   1. In a transaction called withdrawal, you get the “cash” from a bank, who subtracts the value from your account.
   2. In the actual payment, you hand such money over to the recipient.
   3. In a deposit, the recipient puts money into his bank account.
   4. In digital systems, a settlement (or clearing), i.e., some interaction between the two banks, might finally be needed (in contrast to real cash).

B. In a cheque- or credit-card-like system,
   1. the payer first gives something to the recipient (payment),
   2. then the recipient hands that in at his bank (this is called capture or deposit; in a real credit-card system the actual institution doing this is called acquirer),

---

* This is the last chapter of a course primarily intended for third-year undergraduate students. The course is called “Security”, but is actually mainly on cryptology, and most other chapters are German slides. In comparison to an overview paper, I’ve used fewer references, historical notes etc.
Thanks to Michael Waidner and Ahmad-Reza Sadeghi for interesting discussions.

3. this bank turns to the payer's bank and demands real money (clearing),
4. and the payer's bank indicates to the payer that the money has now been deducted (indication).

C. In a money-transfer- or remittance-like system, the payer sends a transfer order to his bank, who does the settlement with the bank of the recipient, which then indicates to the recipient that money has arrived.

D. In a debit order system it is the other way round: The recipient tells his bank that he wants money, and the bank gets it from the proposed payer's bank, which tells the proposed payer that money has gone. Here the proposed payer is obviously allowed to protest. Typically, to restrict misuse, there is also a previous phase where the payer allows a specific recipient to make such debit orders (e.g., for phone bills).

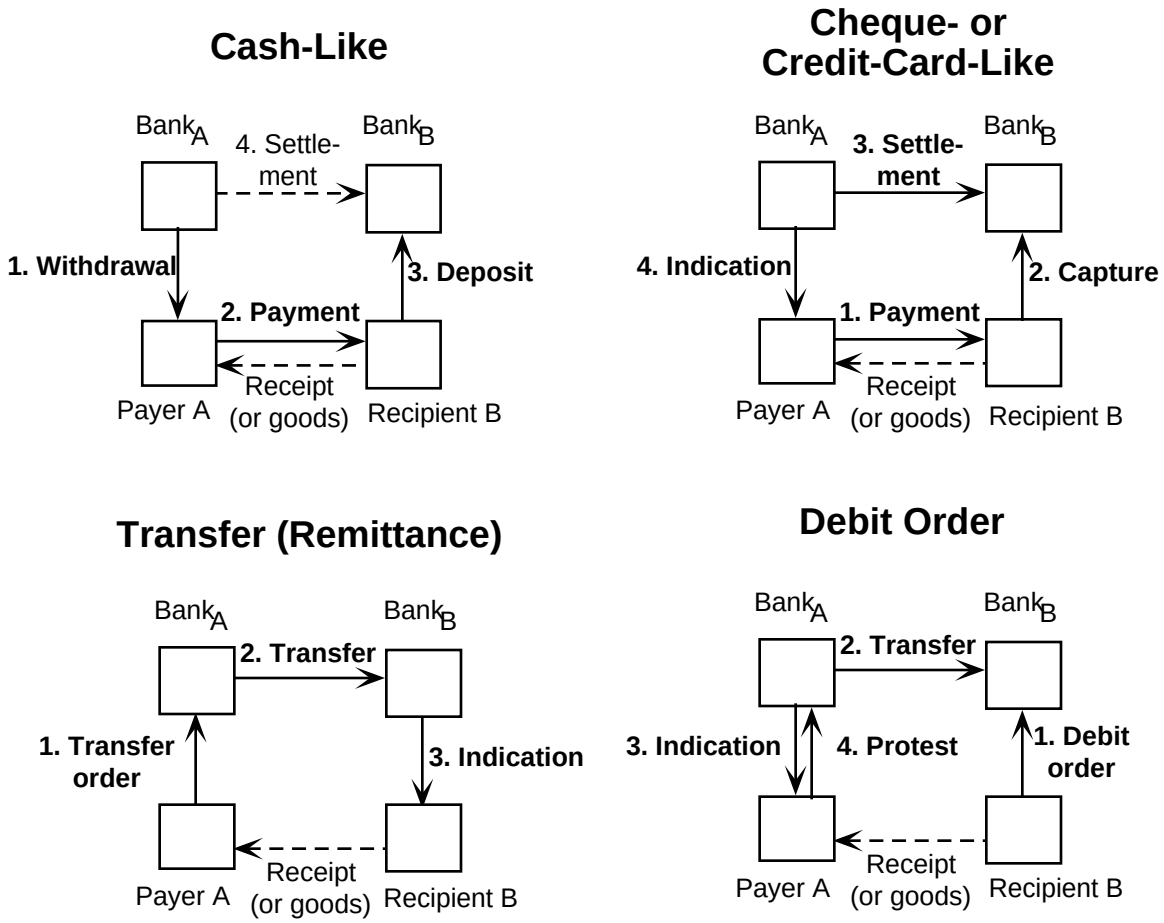


Figure 17.1 Standard payment models in a "money-flow" representation

In all these systems, a **receipt** for the payment may be given — this should be possible in a fair way (i.e., the payer gets the receipt if and only if the recipient gets the money), but actually not many current systems have it. On the other hand, some descriptions of payment systems already include a description of how the money can be more or less fairly exchanged for digital goods. However, it better to consider **fair exchange** of money for goods as a different kind of system with its own

goals, and usually it is also better to separate the modules in the implementation. (Not all goods are digital, nor are they always sent at the same time as the payment, and you don't want to think about the same exchange questions again for each payment system.)

B. More Scientific Treatment

If one looks more closely at the arrows in Figure 17.1, some of them seem to represent a flow of "real" money (typically the settlements, which are usually assumed to be between two normal bank accounts), some a flow of "electronic money" (in particular the withdrawal), and some a simple message (e.g., the indications to the payer). Thus it would be pretty hard to define a formal notion "a cheque-like payment system is … " based on such a figure, in contrast to the figures in Chapter 2 where all elements were clearly either algorithms and messages.

One problem is that "money flow" is not defined at all, and it depends on cultures and communities (bankers, politicians) what is considered "money". For instance, you can lead long discussions whether you really get "money" into your hands if you withdraw electronic cash or whether the "money" is still at the bank and will only leave the payer's bank at the moment of settlement. Such discussions are interesting for economic and legal purposes. Some examples:

- Do issuers of electronic cash need to be banks in a legal sense?
- Should one get interest on electronic cash if it is actually only a representation of money in a bank account?
- Who loses the money if some devices fail?
- In the cheque-like model, is a "credit" in the usual sense given, or are you only allowed to spend as much money as you have?

However, for building the technical systems, you don't need to decide most of this. Hence it is useful to have a more technical view without unclear notions like "money". There are two possible things that one can consider instead: The in- and outputs to the system, or the message flows inside the system. For a high-level view, it is better to consider in- and outputs first, because individual transactions like "payment" may internally consist of an exchange of several messages. Nevertheless, in simple non-anonymous systems the message flow is typically very similar to the "money flow" above, except for the withdrawals and settlements.[1]

The main in- and outputs between a digital payment system and its environment are shown in Figure 17.2.

---

[1] Actually, you can already guess that all systems except the cash-like ones can be built pretty much like their non-digital counterparts and following Figure 17.1., because the main elements are signed messages. The difference with cash is that physical cash is supposed to be hard to copy, while digital information is not. So here, even if you don't care about anonymity, you need special measures. We'll come back to this soon.

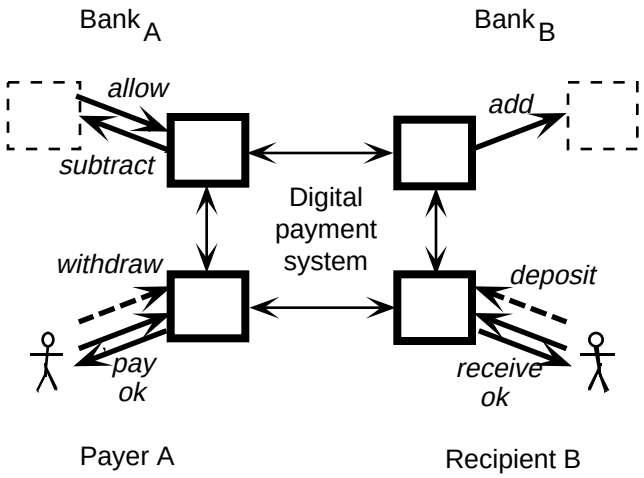


Figure 17.2 Main in- and outputs of payment systems

In Figure 17.2, the bold-face inner boxes are the actual digital payment system; for illustration some environment is also shown: Often the payer and possibly the recipient program would have a direct GUI (graphical user interface), while the bank programs will interact with the old system that handles the normal bank accounts.

The most important inputs are:

- *pay* by a payer: Here the payer authorizes the digital system to make a certain payment.
- *allow* by the payer's bank: Here the bank indicates to the digital system that the payer has enough real money that can be used for one or more digital payments.

The main outputs are:

- *ok* at the recipient: This indicates that he now has received money, so that he can deposit or spend it.
- *subtract* at the payer's bank, which indicates that real money must in fact be taken from the payer (typically from the normal account, but it could also be cash),
- *add* at the recipient's bank, which indicates that real money must be given to the recipient.

The remaining in- and outputs shown in Figure 17.2 mean the following:

- *receive* at the recipient is useful if he is not supposed to accept arbitrary payments made to him, in particular in systems with automatic generation of receipts.
- *ok* at the payer indicates that the initiated payment was successful (similar for all other transactions)
- *withdraw* and *deposit* are specific inputs of certain classes of systems (mostly cash-like) and start withdrawals and deposits.

These were the money transfer in- and outputs; there may be others for purely informational purposes ("how much money do I still have"), for general administration (initialization etc.), and for disputes.

Now you might say that notions like “authorizes to make a payment” mean just as little as “money flow” above, and you are right. I.e., unless you add a specification with what outputs the system should react to what inputs, you haven’t gained much by considering the in- and outputs, except being nearer to an API (application programming interface). Such a specification can in fact be made for in- and outputs, see [PfWa_96], but it is still non-trivial to get complete (e.g., with all possible disputes that can occur afterwards) and anyway it would lead too far here.

Of course, before really writing such a specification, one must first define what parameters all these in- and outputs have. The most important ones are amounts of money and names of partners.

Different payment models, similar to Figure 17.1, can now be distinguished by the order in which the in- and outputs occur, in particular how they are grouped into transactions. The classes get a bit finer than above:

- *Pay-now* systems have just one money-transfer transaction, called a (pay-now) *payment*. It is started by inputs *pay*, *receive*, and *allow* from the three respective participants, and yields all the four outputs (*subtract* and *add* at the two banks and *ok* for payer and recipient) if all inputs fitted together.

  Compared with Figure 17.1, this is a subclass of cheque-like systems where the capture happens immediately, integrated with the payment.

- There are also cheque-like systems where the capture is delayed. Then one can distinguish
  - *reservation* systems: Here *allow* already occurs in the payment and the money is at least temporarily reserved for this recipient (i.e., a *subtract* also occurs). Digitally there is then no need to wait with the deposit — there is an online connection to a bank anyway.
  - *pay-later* systems: Here *allow* only occurs during the deposit. This is the class of standard credit-card systems. However, in the digital world pay-now systems would be more secure without much more trouble, because in pay-later systems the recipient has no guarantee at the end of the payment that he will really get money.

- *Pre-paid* systems (or “stored-value” or “cash-like”) are those with a specific *withdrawal* transaction. It has an input *withdraw* that does not specify a recipient, and leads to the outputs *subtract* and *ok*. The payment transaction has the remaining two inputs, *pay* and *receive*. We now have to subdivide:
  - In *deposit-now* systems, the payment transaction already produces all the remaining outputs *add* and *ok*. Real-world examples are phone cards if one distinguishes between the phone as the recipient and the phone company as the recipient’s bank.
  - In *deposit-later* systems, the payment transaction only produces the outputs *ok* for payer and recipient, but not *add*. There is a special transaction *deposit* for that, where the recipient inputs *deposit* and the output *add* occurs, and *ok* for the recipient. Real-world examples are cash and traveler cheques.

Where there are withdrawals and deposits, they may be over arbitrary sums, i.e., just as with cash, they are typically for many payments together.

For a concrete general API for payment systems (i.e., in a programming language, here Java), see [AASW_98].

C. Classification by Assumed Environment and Communication

Practically, one can also distinguish

- Internet versus shop payment systems,
- and systems with online versus offline payment transaction.

These distinctions are related to specific security measures, in particular measures against double-spending of the "same" digital money. This is quite fundamental and therefore already discussed here, not only in the sections on realizations.

Environment. In practice currently a system either assumes that payers have smartcards and carry them into shops, or that they have PCs which stay at home. (However, this distinction will almost certainly vanish in the future.)

Moreover, it is often assumed that the smartcards are secure even *against* their user (but see Chapter 12), while this is not assumed for PCs.

Obviously, for a really secure digital payment system, just as for any other cryptographic system, one has to assume that all devices are at least secure *for* their users. However, considering that this assumption is currently not fulfilled for either smartcards or PCs (see Chapter 12) against a moderately determined attacker (and breaking a payment system, even only on the payers' side, may well be worth a certain investment), it is reasonable not to rely entirely on such a system, i.e., limit the amounts that can be paid with it or allow refusals of payments that would digitally be non-repudiable (e.g., with digital signatures). This is, e.g., done in the SET credit-card standard (but unfortunately not in the filed trials, it seems).

Communication during payment. Some advantages of offline payments are obvious: lower communication cost and less time-critical transaction handling at the banks. In some shop payment scenarios, online payments would still be almost impossible, e.g., in buses. (In Internet scenarios "offline" is not so important, but still standard with credit-card payments.)

The problem is whether one can give the recipient a final, non-revocable output *ok* in an offline payment. This presupposes that the payer's bank has input *allow* at some point. Hence an offline system with *ok* during the payment must certainly be prepaid. But even then, in any purely digital system the payer could make a backup before each payment and reset his system to this backup after the payment. In this way, an arbitrary number of payments to different recipients are possible with the "same" money. This is called the *double-spending* problem.[2] There are three kinds of solutions:

---

[2] If the payer already knew the recipient when withdrawing money, the system could somehow make the representation of this money specific to this recipient. Then the recipient only needs to check that *he* does not receive the same money twice. This is called "designated payee"-systems, and they are obviously easier to design than others. (E.g., phone cards are of this type, and there are also some theoretic proposals.) However, such systems are not real payment systems because they are not universal (see below).

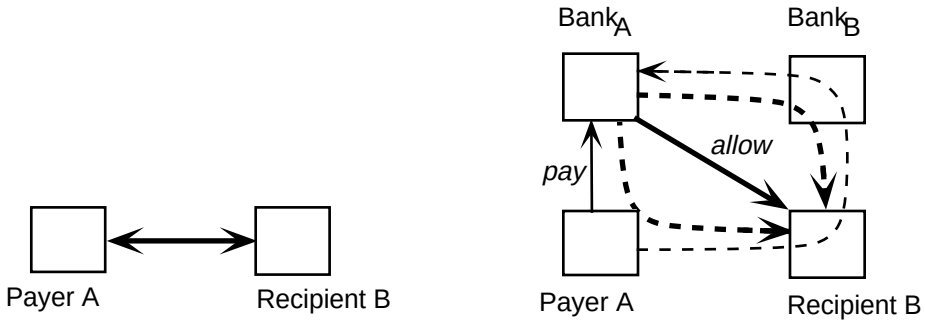


Figure 17.3 Offline (left) versus online payments. The *allow*- and *pay*- arrows, in slight misuse of notation, indicate information related to an input *allow* or *pay*.

- Online payments. Therefore the main arrow (undashed bold-face) in the online payment in Figure 17.3 means that the most important message flow is that the recipient needs a guarantee from the payer's bank that the money is new. Physically, however, the message is usually sent via either the payer (with a freshness measure) or the recipient's bank (dashed bold-face arrows). In addition, the payer's bank needs to know that the payer really wants to pay to this recipient (undashed thin arrow); again the message may be sent in another way (thin dashed).
- If one really wants to prevent double-spending with offline payments, one has to rely on the payer's device to prevent the resetting to a previous state, i.e., it must be tamper-resistant *against* the payer. This is what typical current smartcard systems do assume.
- Another possibility is to rely on detecting and punishing double-spending afterwards. This can even be done in anonymous payment systems — there are tricks that achieve that the anonymity automatically gets lost for double-spent money. However, a doublespender of a really high amount may simply flee the country, and, worse, doublespender detection is digital and will therefore point back to the person from whose device the double-spending was done. Hence a clever doublespender would do it from stolen devices. Thus it can only be used in a legally relevant way if one has highly secure devices where thieves quite certainly have no access.

The second and third measure, which are both not perfect, should usefully be combined.

D. Other Functional Properties

- **Universal?** I.e., can you make all kinds of payments with this system? The two properties make payment-like systems much easier to design, but also non-universal, so that they are no longer *not* payment systems:
  - The recipient is always known at withdrawal time. However, anything called "money" should be spendable everywhere; objects spendable only in one place are called vouchers.
  - Even stronger: The recipient is always equal to the (one and only) bank. Phone cards are of this type because the phone companies can issue them themselves.
- **Open?** Can all potential payers and recipients take part? Typically this is a buzzword for non-digital properties: no need for creditworthiness, ease of use (children, handicapped persons), no

large infrastructure investments for payers and recipients. An interesting question is whether the system allows everybody to be a recipient, too. Often normal people are only allowed to pay.

Other questions are openness for all potential banks and for different software and hardware producers. This is mainly a matter of standardization.

- **Extensions.**
  - Does the system offer receipts for payments? Fairly, i.e., the payer will in some way obtain a receipt if and only if the recipient obtains the money? (Typically, the enforcement somehow goes via the bank, which can see whether a payment took place or not, even in the anonymous case.)
  - Is there a personal log? (No problem digitally, but the fact that one cannot see one's previous payments is a major disadvantage of smartcards for normal users.)
  - Undo? (Typically not — one has to make separate payment back. It is technically difficult because you cannot "give a message back".)
  - For prepaid systems: loss tolerance? I.e., can one get money back if one loses one's device or data? (In non-prepaid system this is typically clear because the "money" stays at the bank; one only loses one of several ways to access it.)
  - For offline systems: transferability? I.e., can you use received money to pay again without having to deposit it at the bank and withdraw it again? This property, which is perfectly normal for real cash, is *not* given in typical digital payment systems, and quite hard to achieve in a secure way.

17.1.2 Integrity and Availability Goals

In systems like payment system, integrity and availability roughly means security against fraud.

The general goal in payment systems is multi-party security, i.e., such requirements from all parties should be considered, and nobody should be required to trust other parties unreasonably. Note that with classical payment systems it was clear that this also concerned the banks, at least that the bank could be held accountable for any irregularities, because everybody held *evidence* in form of signed statements. The same security is also desired for digital payment systems, but many current proposals ignore it.[3] Some proponents argue that fraud by bank insiders (quite a common event in the paper world) becomes less likely in computerized systems due to computer security. However, most banks still use unprotected systems on main-frames with online access and with constant changes. (At least one hears this from many sources — typically the measures are kept secret, which in itself is not a good sign.) Moreover, one only needs to look at how banks handled the security of the PINs of Eurocheque cards ... (see Chapter 12).

More technically, the main integrity requirements are that nobody wants to lose money (where desired payments obviously are not counted as "losing"). We don't go into details here of how this

[3] However, recall that ignoring certain security problems is fine in situations of mere risk management, i.e., if a bank decides not to take an expensive measure and rather to bear the risk *itself*. For example, banks also typically do not verify the handwritten signatures on small money orders, but they bear the risk of acting upon a recognizably forged signature.

can be expressed in terms of the in- and outputs from Section 17.1.1B (see [PfWa_96] in case of interest).

Moreover, payers may also want to be able to specify exactly to whom they pay that money. Not all system proposals currently guarantee this. In systems where double-spending is only detected afterwards the banks require that doublespending can indeed be detected and a responsible user be identified (so there is a special output for this).

For several of the in- and outputs, corresponding disputes are necessary, and all parties require that they win legitimate disputes.

- For example, the usage of receipts at the in-/output level is that if you successfully made a payment (signaled by the payer's *ok*), you can later convince anyone of it in a dispute. Correspondingly, the recipient wants that this convincing only works if he really received this payment (signaled by the recipient's *ok*).
- Similar disputes are needed for the outputs that concern account balances. For example, a payer may deny that a *subtract* occurred which shows up on his statement of account, and will require to win a dispute if he did not make a corresponding input *pay* or *withdraw*.

The technically optimal trust model is that for each of these requirements only the participant himself and, for the correct output in a dispute, the arbiter in this particular dispute is trusted.

17.1.3 Confidentiality Goals

The essential goal with digital payment systems should be to offer about as much privacy as previous payment systems do. This would mean that small payments can be made confidentially and anonymously. For example, with real cash, nobody can collect data on what books a person buys, which buses she takes, what bathroom articles she uses etc. The importance of privacy in such actions is typically not so much in each single purchase (although there are also some sensitive ones among them), but in the fact that very complete user profiles can be generated from them.

Before looking at different models, note some principle limits: For payments over networks, the overall privacy cannot get better than the privacy of the network connection. There are ways to communicate anonymous ly in principle and even in practice (mainly mixes, see [Chau_81], which is also the principle underlying the best remailers), but they are currently not in wide-spread use. Similarly, for payments in shops you cannot get more anonymous overall than you are in the shop. However, such outside knowledge is typically not entered into a digital system and thus not available on a large scale. Hence in the shop case it makes sense to pay anonymously even in a neighborhood shop where you are known.

Technically, we can classify the privacy of payment systems as follows:

A. Confidentiality of Payment Data Against Outsiders

Here one can use encryption. This is fairly independent of the payment system as such. In particular, during a payment a secure channel will usually already be in place anyway, because other steps like

ordering goods preceded the payment, and the confidentiality should be consistent among all related steps. Then the payment protocol will simply run over that channel.[4]

B. Anonymity

The rest of this section therefore concerns anonymity towards the partners, who do know some payment data. The following criteria can be used for classification:

- **Who** is anonymous? In most proposals, it is only the payer ("payer anonymity"), but technically it can also be the recipient or both. The reason why one usually only makes the payer anonymous is a catalogue-like scenario where merchants have far fewer reasons to be anonymous than buyers. However, other scenarios for payments are conceivable. In particular, with coin-like payment systems, i.e., if the money comes in certain "pieces" of fixed amount, one is used to change in the current world, but if a payment system offers payer anonymity only, one cannot give change because then the payer from the main payment would be the non-anonymous recipient.
- **In what actions?** Typically, payers are only anonymous in payments, but not in withdrawals of money. In other words, most proposals assume that there are normal non-anonymous accounts and all digital money must at first be taken from such an account. Technically, however, the accounts could usually just as well be anonymous; see Section 17.3.1. (Of course, the bank would not give credit on such accounts.) Some proposals do not need accounts at all; see Section 17.3.3.
- **Relative to what other actions** is the anonymity?
  - **Unlinkability**. This is the real anonymity in this respect, i.e., no relation between this anonymous action and other actions can be found out.
  - **Linkability**. Here several actions are each anonymous, but one can see a relation. The relation is almost always that one can see that several actions were made by the same person.

  Typically one can see some kind of pseudonyms in anonymous actions (e.g., a key used or a card number). Unlinkability then essentially means that the same pseudonym is used only once (transaction pseudonym), in contrast to many times (long-lived pseudonyms). It is clear that the same pseudonym should be used for several steps of the same business transaction (e.g., an order, a payment, and a delivery of goods). But different business processes, even with the same partners, can be unlinkable.

  The danger with linkability is reidentification: After some time, there is usually enough information available about the person performing all these actions to identify her. E.g., if a phone card is bought in perfect anonymity, patterns of calls to one's home might identify the owner. Even more clearly, if one can make Geldkarte-like card payments under a number (pseudonym), but one is more or less forced to reload the card from one's own account and the card number is shown to the bank in reloading, all the payments are no longer anonymous from the bank.

---

4 For actual implementation, this may not be quite so simple, because if one is given an arbitrary payment system implementation with direct communication (in contrast to token-based, see Chapter 11.0) one has to "catch" that communication to divert it over the new secure channel.

There is no social reason for having anonymity without unlinkability, but a number of proposals have it for technical simplicity.

So far we have identified goals. The next criteria are the trust model. (Compare Chapter 1; anonymity of payment systems is an example where some requirements engineering is really needed, while for small systems like encryption systems and signatures systems it was a bit of an overkill.)

- **From whom** is the anonymity?
  - **Real anonymity**. This means that anonymity holds against all other parties (even if they cooperate), similar to real coins.
  - **One party only**. The anonymity only holds against one of the canonical two partners. For a payer in a payment, this means that he is only anonymous from either the recipient or the banks, or at least only those two do not cooperate. (One typically just talks of "the banking systems" in such contexts, but distinguishing the payer's and the recipient's bank could also be useful.)
  - ***k*-out-of-*n* central parties** (typically with $k = n$). There are several central parties. If k of them cooperate, they can identify someone. There can be two reasons for this, which have different names:
    - **Anonymizers**: The anonymity is achieved by passing money (or a representation of it) through several parties, who each can link the old form with the new form, but not beyond.
    - **Escrow**: One starts with a fully anonymous system and, for reasons of law enforcement, now adds additional parties and shares some private information among them such that they can identify.

      Then one can further distinguish in what granularity they identify: per payment, or all payments of one person together, also future payments etc.

    A problem with these approaches is (apart from the natural question whether one trusts the parties as such), that one attacker might break into all their computer systems.

Finally, there are degrees of security with which anonymity can hold.[5]

- **Among how many** people is one hidden when anonymous? Obviously, the more the better. If an anonymity group, i.e., the set of people who could have performed an action, becomes too small, there is a risk both of reidentification and that any bad luck directed to the real actor happens to the whole group instead.

  In particular, one must take care that different actions that are linkable by outside data are not performed in different anonymity groups, because then an attacker would know that the real actor is in the intersection of the two groups. E.g., this is a problem with many remailers, because a different anonymity group is given for each message. Now if several messages belong to the same business transaction …
  - The optimum with a payment system is "among all clients of your bank".

---

5 In a larger context, the first of the following points would be considered part of the goals because it is a "degree of security" that *only* occurs with anonymity goals, i.e., it did not occur in Chapters 1-3.

  - Where communication anonymity plays a role, these groups are typically intersected by groups that communicate in the same local area or at the same time.
  - Furthermore, there are natural links by outside data like time and amount. E.g., if only very few payers withdraw “electronic coins” of very high value, and some recipients deposit them, there is only a small set of possible payers who can have made that payment.
- **Standard degree of security** like information-theoretical versus computational.

Finally, note that integrity and anonymity are no contradiction. For the payment systems alone this will become clear by the constructions, even with fair receipts. Fair exchanges of goods and money can also be done anonymously, see [BüPf_90]. Where identification during a purchase is needed for other purposes (e.g., buying dangerous goods), it can be done by normal tools for this purpose, e.g., a digital signature. (A more general study of what legally relevant actions can be anonymous and how can be found in [PWP_87].)

17.2 Non-Anonymous Online Systems

We now look at concrete systems. The criteria by which they are primarily sorted were chosen for technical similarity: The main technical difficulties are to achieve anonymity and to achieve offline payments. Hence non-anonymous online systems are the easiest class. This is the class where, to get a secure system, you (more or less) simply need to take one of the classical paper-based systems with handwritten signatures and use digital signatures instead. However, not all current systems are actually so secure.

In principle, all these system are state-of-the-art of around 1980, but they only come into real practice now because of Web applications.

17.2.1 Cryptoless Systems

Some payment systems do not use any cryptography at all. The best-known examples are simple credit-card payments on the Internet, i.e.,

$$A \text{ — credit card number —> } B$$

This is totally insecure from the cryptographic point of view, but actually secure for the payer in real life![6] The reason is that so-called MOTO transactions (mail order/telephone order), where the supposed recipient of a payment has no signature of the payer, are not binding by the credit card conditions in Europe, i.e., the client can cancel them when they appear on his statement of account. This is reasonable because the recipient has nothing really in hand to prove that the payer wanted to pay.[7]

Of course, then someone else bears the risk of the technical insecurity, here the recipients. In some cases they may be quite willing to do so out of a certain risk analysis, e.g., companies that deliver books in small quantities to physical addresses. However, the fraud with this type of payment seems to be going up; I heard numbers about 25% wrong Internet payments in the US Christmas sale

---

6 The only danger is that credit card numbers might spread more widely to people who can forge handwritten signatures, and thus the combined security of number and signature is lost.

7 In real life, credit card companies nevertheless seem to often try to refuse the cancellation at first.

1998, while credit card fraud usually "only" makes up 1-2% of the total sum. In the end, of course, consumers pay for this by higher prices.

17.2.2 Secure Symmetric Channels Only

The typical example here is simple credit card payments, but via SSL-channels that the browser offers. Any other simple password scheme with any other secure channels would achieve the same effect.

Recall from Chapter 11.4 that SSL achieves confidential and authenticated channels, but is symmetric. In particular, the recipient still gets the credit card number or other password, and no signatures. If one still considers such transactions as MOTO (which one should, and currently does), it makes no difference for the payer compared with Section 17.2.1, but the overall risk for recipients is reduced because it is again about as hard for attackers to collect credit-card numbers as in the non-digital world.[8]

Home banking schemes relying on PINs or TANs (transaction numbers) and used together with SSL or other secure channels are somewhat similar, i.e., they are symmetric with the bank, but the risk that other recipients also see the "secret" is gone. Of course, symmetric authentication would then be the cryptographically better solution.

17.2.3 Real Symmetric Systems

These systems are now somewhat out of date, but still sometimes proposed instead of the systems with signatures under the name of micropayments because they are faster but less secure. (Typically, however, "micropayments" is used for systems with more distinct features, see Section 17.2.5.) The basic principle is to put symmetric authentication where you would typically expect signatures. E.g., with a money transfer system this can be done with a simple 2-message flow as in Figure 17.1.C. With a cheque-like message-flow it only makes sense with a pay-now model, because the recipient cannot even verify the authentication on the payment message (which must be for $Bank_A$), and thus he does not know whether he has anything in hand. The original NetBill system was of this type, but newer versions use signatures.

17.2.4 Simulating Paper System with Signatures

As mentioned, here you can in principle just simulate cheques, credit cards, money transfers — technically all rather similar. These are the cryptographically secure alternatives to Sections 17.2.1 to 17.2.3.

Note, however, that as a payer you don't have an advantage over Section 17.2.1. In fact, as long as you don't have a really secure device, you have a big disadvantage if you now become liable for any payments made with your key, because you have no full control over this key. Hence the SET standard mentioned below actually intended *not* to make payers liable. In the other case, the investment (e.g., in certificates) should reasonably not be paid by the payers, and the need to limit liability (cf. Section 8.2.2) becomes strong.

---

8 Or maybe still a little bit easier because it may be easier to set up a small fraudulent company that gets a lot of small orders and thus the numbers.

- **Home banking.** An important example is the German standard HBCI (home banking computer interface), see [HBCI_98]. This standard closely reflects all paper payment models where payers and recipients only interact with their respective banks, e.g., money transfers and debit orders.

  It is primarily a protocol standard (cf. Section 11.0). However, some requirements on the security of products implementing the client part are made. In particular, some not yet fully specified security of key storage is required (but they think maximally of smartcards) and some requirements on the user interface. Furthermore, registration is fixed: The client registers directly with the bank by sending a handwritten signature. (Usage of external certificates may be considered in the future.)

  The message formats follow paper forms quite closely and seem designed in a robust way at a first glance. (In particular, the freshness measure is sequence numbers, and there are no optimizations in the message format.) The communication is somewhat connection oriented, i.e., there is a dialogue start message and a corresponding end message, but mainly to exchange parameters, not for session key establishment — encryption, if used at all, is hybrid encryption on a per-message basis. Between the dialogue start and end messages, the client can send one or more orders, which are immediately answered by the bank.

  Cryptography is rather fixed (RSA, key length 768). For backward compatibility with certain smartcard systems, symmetric authentication is also still allowed (unfortunately also called signatures).

- **Secure credit cards**. These are systems like CyberCash, IBM's iKP (actually only 3KP from this series), newer versions of NetBill and, most importantly, the credit card companies' standard SET. The latter was designed as a compromise between all the former and some more, and got quite complicated in the process.

  These systems essentially follow the message flow in Figure 17.1.B. The formats get a bit more complicated than for home banking, because the message to the merchant includes the authorization from the payer to her own bank. Often there are initial messages for setting up parameters or for agreeing on a price. (However, such agreements do not really belong inside the payment system because they might be integrated with a catalogue, or for more complicated purchases with negotiations of other features.)

  iKP is a rather short variant for reading, but not a simple robust design, but extremely optimized without a clear proof. See the exercises.

- **Cheques**. As mentioned before, there is no real difference between the payment models of credit cards and unguaranteed cheques. The main concrete proposal under the name of cheque is that of FSTC (Financial Services Technology Consortium) in the US. They also recognized that with more or less the same formats one can implement other payment models, too.

17.2.5 Micropayments with One-Way Chains

Micropayments in the narrow sense are more efficient payments for the special case where a payer makes many successive payments to the same recipient. The underlying business model is paying for many individual web pages from the same server (although business analysts currently tend to say that other models will prevail). Another application is phone ticks, i.e., paying a network provider for

small units of communication; in this context (but for an anonymous system) the idea first came up [Pede_97].

The following system is cheque-like. The idea is to set up all important parameters in a first message, which is signed in a normal way. The following messages can be interpreted as fresh signatures on always the same message "I agree to pay one more unit". This can be done with an identification system, see Section 9.4; specifically the system with one-way chains is used.

Setup message: Concretely, the first message might be

$$sign(sk_A, (prot, \text{“}setup\text{-}msg\text{”}, seqno, B,$$
$$price\_per\_unit, pk_{chain})$$

The fields have the following meanings:

- sk_A is *A*'s normal signing key.
- The next three fields are normal robust design, i.e., *prot* is the name of this payment protocol, "*setup-msg*" denotes that this is the first message (maybe there are others in the overall protocol), *seqno* is a sequence number to avoid replay of such messages.
- Then comes an identifier of the recipient, here *B*.
- Now comes the price per unit of the payments. Thus there must be a fixed price per web page on this server (or one has to pay several units per page) or per a certain number of seconds of communication. (However, the description of the goods need not be included here.)
- Finally comes a new session-public-key for the chain system. This has been generated by the payer as follows and is only used in this one payment: Let *f* be a one-way function (even better a one-way permutation, but typical practical one-way functions like $f\colon k \to \mathrm{DES}(k, m_0)$ for a fixed message m_0 are not). *f* is fixed for *prot*. Let $\{0, 1\}^l$ be the message space for the current security parameter *l*. Then

  $$sk \in_R \{0, 1\}^l,$$
  $$pk := f^x(sk),$$

  where *x* is the maximum number of units one expects to have to pay. (Fix it?)

Unit payments:

For each unit to pay, the payer simply sends the next preimage in the chain to the merchant. Hence the payer keeps a counter *i* initialized with 0, and in the *i*-the micropayment he sends $sig_i := f^{x-i}(sk)$.

To avoid having to recompute this from *sk* in each micropayment, he may either store the complete chain or make a time-memory tradeoff, e.g., by storing every 10th value initially, and then always recomputing chains of length 10.

The recipient always has to store the previous value sig_{i-1} he got, and he verifies the next value sig_i via "is $f(sig_i) = sig_{i-1}$.

17.3 Anonymous Online Systems

The best-known subclass of anonymous online systems is what is now called ecash-like systems, see Section 17.3.3. They are also called coin-like online cash systems. However, we will first look at a weaker form of anonymity, anonymous accounts, and then at another system that achieves a rather similar form of anonymity with fewer cryptographic tricks.

17.3.1 Anonymous Accounts

Anonymous accounts are just like non-anonymous accounts, except that they are not linked to a real identity, and that the bank will not allow the balance on such accounts to drop below zero.

Account opening: To open an anonymous account, a person simply generates a key pair of a signature scheme,

$$(sk, pk) \leftarrow gen_{sig}(l),$$

and sends *pk* to the bank; *pk* is called a pseudonym. No prior certificates are needed for the key pair. The bank assigns an account number *N* and certifies this fact, e.g., with $sign(sk_{Bank}, (\text{“}Account\ opened\text{”}, N, pk))$.

Payment between anonymous accounts: The recipient *R* tells the payer his account number N_R. Now any of the non-anonymous message flows is possible. For instance, the payer can send the bank a signed transfer order with a sequence number:

$$sign(sk, (\text{“}transfer\text{”}, N, N_R, seq\_no, amount)).$$

Money can be paid into such an account with any normal cheque, cash or transfer order. To take it out again, a signature with *sk* is needed, similar to the one above.

Relative to the person’s normal account, such a system is prepaid: There are withdrawal and deposit transactions. However, the bank could pay interest for money on anonymous accounts. If one only considered anonymous accounts, the system would count as pay-now. The loss-tolerance problem of prepaid systems exists in a minimal form: If one loses the secret key *sk*, one loses access to the account, and one cannot prove ownership in another way unless one has taken specific precautions.

The degree of anonymity is low, because all transactions on one account are linkable. On the other hand, both payers and recipients are anonymous. Note that an anonymous “Geldkarte” can be seen as a symmetric analog of this scheme.

17.3.2 Anonymously Transferable Standard Values

The scheme in this section is from [BüPf_89], Section 4.4. It was reinvented in [Simo_96] and patented by Microsoft, but the patent must be invalid given the older publication. A standard value is like a “one-time” account whose ownership is transferred with its value, to avoid the linkability of the scheme in Section 17.3.1.

We will immediately look at a payment, assuming the payer already has a standard value *v* (where *v* is a number, like a banknote number).

Payment:

1. The recipient chooses a new key pair $(sk_R, pk_R) \leftarrow gen_{sig}(l)$. We call pk_R a pseudonym. He sends the payer a message like "I want to receive standard value v for purpose Z under the pseudonym pk_R" and signs it with a key that is already known to the payer. (This is a precaution for later disputes whether the recipient got the money.)

2. The payer sends a transfer order to the bank, e.g.,

   $$sign(sk_P, (\text{"transfer standard value"}, v, \text{"to"}, pk_R)).$$

   The secret key sk_P used is from the key pair the payer chose when receiving v, i.e., from Step 1 of the previous transfer.

3. The bank looks v up in a database and retrieves the public key pk_P of the current owner. It verifies that the transfer order is correct and signed with respect to this key. If moves pk_P to a list of former owners of v and stores the transfer order (again as a precaution for later disputes). Finally, it enters pk_R as the public key of the current owner.

   The bank signs a transfer confirmation, e.g.,

   $$sign(sk_{Bank}, (\text{"new owner of"}, v, \text{"is"}, pk_R)),$$

   and sends it to the payer, who forwards it to the recipient.

   If the recipient does not obtain this confirmation, he contacts the bank himself to see if the transfer of v to his pseudonym went through.

4. Receipt: In a correct protocol execution, the recipient R confirms the payment under the pseudonym or name under which P knows him. However, if R refuses, P can use the bank's confirmation from Step 3 together with the recipient's message from Step 1 to prove that he transferred the money to a pseudonym of the correct recipient.

Depositing a standard value to a normal account is done with a similar message as in Step 2. Withdrawing a standard value from the bank for normal money can be realized as a payment from the bank, who entered itself as the initial owner of v, to the first real owner.

In this scheme, both payers and recipients are anonymous, and payments of the same person are not linkable. However, there is a certain link between all the payments with the same standard value v, just like with current banknote numbers.

17.3.3 Schemes with Blind Signatures

This class contains the best-known anonymous payment schemes, those which are typically called coin schemes. However, the "coin" terminology is unclear. One could almost better call a standard value from Section 17.3.2 a coin. (For example, both schemes are online in contrast to real coins, and the standard values are transferable and anonymous for both parties like real coins.) The really distinctive concept in this section is that of blind signatures.

A well-known product in this class is e-cash (from the former company DigiCash), which realizes the scheme with payer anonymity and non-anonymous accounts as described in Subsections A and B below. Several banks, among them Deutsche Bank, made a field trial with it. Most of the ideas in this section are from [Chau_83, Cha8_85, Chau_89] (a first sketch, a well-known overview, and the full

paper). The coin keys and general dispute security, and the scheme with maximum anonymity are from [PWP_87].

The scheme was originally invented for shop scenarios and thus anonymous communication was no problem; e-cash, however, is for an Internet scenario.

The basic concept is shown in Figure 17.4: A bank somehow gives a coin, which is a signed message, to a payer, and the payer can somehow transform the coin before giving it to a recipient. The recipient then deposits it. Because of the transformation, the bank cannot link which deposited coin corresponds to which withdrawn coin. Thus it cannot find out who paid to whom.

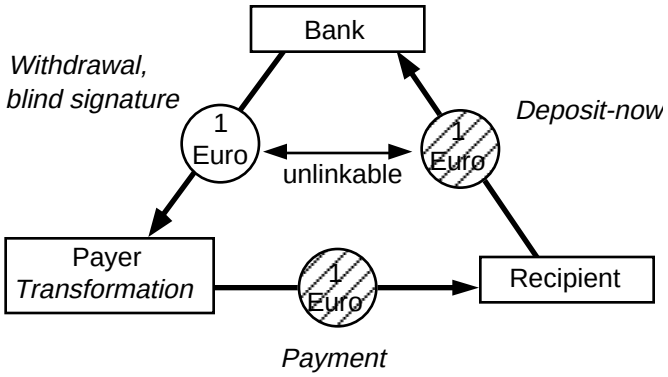


Figure 17.4 Basic concept of an online system with blind signatures

A. Basic System in Abstract Form

Now we show the basic version of this system in more detail, but still in abstract form. This basic version is prepaid and deposit-now, and it provides

- payer anonymity with unlinkability,
- no recipient anonymity, and
- no security in disputes.

An overview is given in Figure 17.5.

- **Setup:** Some parameters have to be distributed initially by the bank.
- **Withdrawal:**
  1. The payer generates a coin *coin* with an operation *gencoin* (probabilistic and based on parameters from the setup). This is the form later used in the payment (striped in the figure).
  2. He transforms it with an operation *blind*. We call the result *blindcoin* (white in the figure).
  3. He sends the blinded coin to the bank together with a withdrawal order stating what amount he wants, e.g., 1 Euro, and from which account.
  4. The bank subtracts the amount from the account and signs *blindcoin* with a special key belonging to this amount.
  5. The bank sends the resulting signature, *sigblind*, back to the payer, who tests it.

- **Payment with deposit:**
  6. The payer uses an operation *unblind* on the signature so that the result *sig* fits the original (striped) coin. He needs stored parameters from *blind* for this.
  7. He sends (*coin*, *sig*) to the recipient.
  8. The recipient simply forwards this to the bank to make an on-line verification against double-spending. (He could verify the signature, but it makes no difference.)
  9. The bank verifies the signature and checks in a database that this coin was not deposited before. If all is ok, it enters the coin there and adds the amount to the recipient's account.
  10. The bank tells the recipient the result of the tests.
  11. If it was ok, the recipient typically signs a receipt or gives the payer goods.

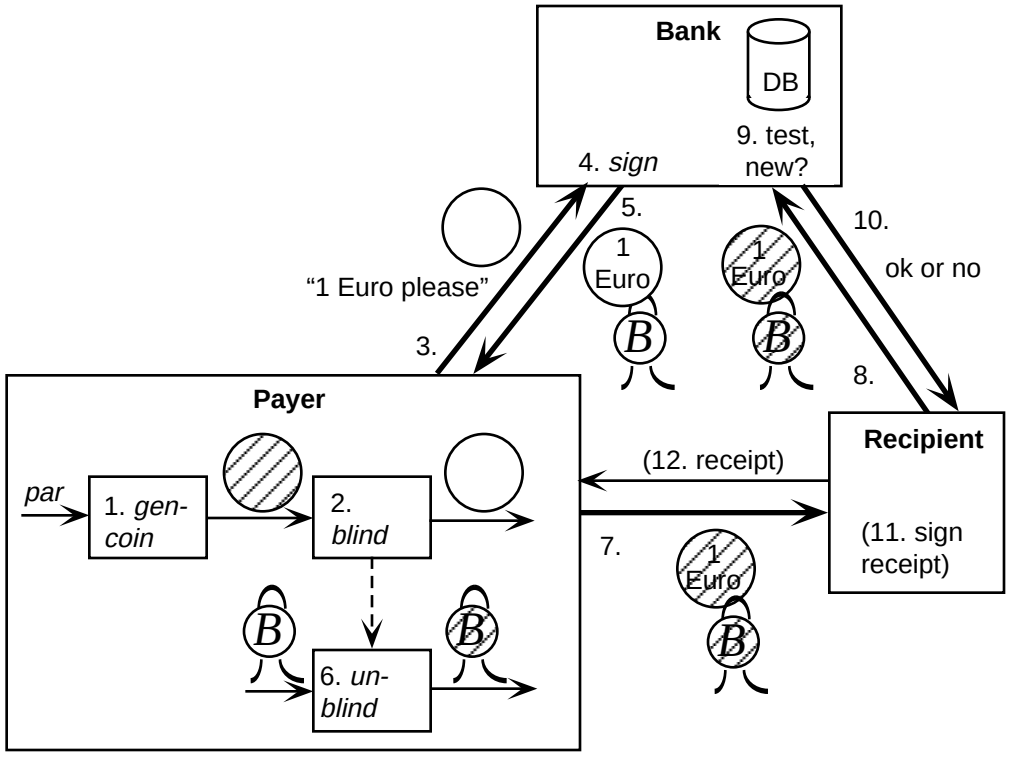


Figure 17.5 Abstract representation of basic blind RSA signatures

Note where the inputs and outputs from Figure 17.2 occur: *withdraw* triggers Step 1; *allow* is needed in Step 4, and *subtract* also occurs there. Later, *pay* triggers Step 6, and if the recipient has to enter *receive*, it is needed in Step 8. Then *add* occurs in Step 9, and *ok* to the recipient in Step 10. An *ok* to the payer at the end is not provided in the basic system, because the receipts are already an extension. (Without receipts, this *ok* would come in Step 6 when the device finds that it indeed has a coin.)

Two non-cryptographic issues are important for real-life anonymity:

- Payers should typically wait some time before using a withdrawn coin. Otherwise the bank can link withdrawals and deposits simply by their relation in time.

- There should not be too many different amounts, because anyone paying with a certain coin is only anonymous among all people who have withdrawn coins of the same amount at the same bank.

The coins would typically have a limited validity (some months or years) to limit the problems if signature forgery becomes easier, and to limit the size of the coin database. This database is large, but not infeasible. In other payment systems, there is one record per payment, and here one per coin used in a payment, i.e., about a factor of 5 more.

B. Cryptographic Realization

For a concrete system, we only have to fill in the operations in Figure 17.5. The original and best-known implementation is with RSA blind signatures. This can be seen as a positive usage of the active attacks on RSA signatures (see Section 6.4.1A, in particular the last attack).

We denote the bank's RSA key pair for signing coins by

$$sk_B = (n, d_B),\ pk_B = (n, e_B).$$

In reality, it needs one such key pair for every coin amount like 0.1 Euro, 1 Euro, 10 Euro etc., or rather binary values. It distributes the public keys during setup. The operations are now:

a) Coin generation *gencoin*: Here we need a redundancy scheme for RSA signatures. E.g., the payer generates a random value c, which is 160 bits shorter than n, and sets $coin = (c, hash(c))$ for a hash function fixed in setup.

b) Blinding *blind*: Choose $r \in_R \mathbb{Z}_n$ (called blinding factor) and set

$$blindcoin := coin \cdot r^{e_B} \bmod n.$$

c) Signing is naive RSA signing:

$$blindsig := blindcoin^{d_B} \bmod n.$$

Thus we obtain $blindsig = coin^{d_B} \cdot r^{e_B d_B} = coin^{d_B} \cdot r \bmod n$.

d) Unblinding signatures, *unblind*: Given r from blinding, let

$$sig := blindsig/r.$$

Thus $sig = coin^{d_B}$.

e) Test: The signature test is the normal RSA test, and the redundancy provided in Step 1 is verified.

Proof of information-theoretic anonymity ("sketch" for the active part): The claim is that every coin as seen in payment and deposit can come from any blinded coin as seen in withdrawal, with equal probability. More precisely, we have to compare

- the entire observations (called "view") of the bank and recipient in payment and deposit
- with the bank's entire observation in a withdrawal.

In our case, we can easily reduce this to only the coins and the signatures in both forms: The only other information from the payer is the withdrawal order, but the additional information there is totally unrelated to the payment.

Passive attacks: Let us first consider only passive attacks, in particular the bank carries out its protocols correctly. Then each signature *sig* or *blindsig* is uniquely determined by *coin* or *blindcoin*, respectively, and hence we only need to consider the coins. (The signatures give no additional information.) Consider any values *blindcoin* and *coin*. They can belong together if there exists r such that

$$blindcoin = coin \bullet r^{e_B} \bmod n.$$

This is equivalent to $r^{e_B} = blindcoin / coin$, and thus $r = (blindcoin / coin)^{d_B}$. Hence there is precisely one such r for every pair, except if $coin$ has no inverse mod n, which happens only with exponentially small probability. The result corresponds to the sufficient criterion for the generic secrecy formula in Section 6.1.1B:

1. The "key", here the blinding factor r, is random and uniformly distributed.
2. For every possible secret, here the blinded coin, and every observation, here the unblinded coin, there exists exactly one key r such that they can belong together.

Active attacks: Now let us consider active attacks. They can only come from the bank, which could choose a wrong n and e_B and give wrong signatures. We show that if it does, this can almost certainly be detected soon.[9]

First we can see that as long as $\gcd(e_B, \phi(n)) = 1$, exponentiation with e_B is still a permutation (similar to the proof for correct RSA keys). In this case, the test in Step 5 of the withdrawal still guarantees that signatures are unique, and we still only need to consider the coins. The rest of the proof is also still valid, except that if n has more prime factors, the probability that *coin* is not invertible may increase. Here a test of $\gcd(n, coin)$ could be inserted, or someone could run trial division and try to find small factors of n.

If $g := \gcd(e_B, \phi(n)) \neq 1$, the payers cannot notice this directly (except if $2|e_B$, which we can exclude). However, one can then see that every value x^{e_B} has g roots (similar to the proof that squares have two roots in Section 6.6.4), and thus at most $1/g$ of all values *coin* have an e_B-th root. If one assumes that these values were uniformly distributed (which is not quite true because of the hash function), this means that the bank will in more than half the cases not be able to sign a coin, which will be noticed very fast.

Unforgeability? A minimum requirement is that the bank does not lose money in a payment system. For example, if someone could existentially forge RSA signatures with the given redundancy predicate in a passive attack, this would not be fulfilled. As such security is not proven for RSA under a normal assumption, the entire security of this payment system is certainly not provable at the moment.

What one would need to prove is so called "one-more unforgeability". This means that no probabilistic polynomial-time attacker who carries out k blind signing protocols (withdrawals) with the signer (bank) can afterwards present $k + 1$ valid signatures with significant probability. This has not been proven for any efficient scheme under a normal assumption yet. A very inefficient provable

---

9 Hence we use the trust model where the bank is not willing to take a risk, see Section 1.3.2. An alternative where nothing can go wrong at all is that the bank must give a so-called zero-knowledge proof of the correctness of n and d_B to each payer at the beginning (see the course "Cryptographic Protocols").

scheme was presented in [PfWa2_92], a rather efficient scheme provable in the so-called random-oracle model (which is a heuristic, but better than nothing) in [Poin_98].

C. Adding Security in Disputes

Now we want to provide secure for disputes between the bank and a payer or recipient. Such disputes can in particular occur if the clients say that their statements of account are not correct, i.e., either a withdrawal (the output *subtract*) appears there but they did not get the corresponding coin, or a recipient misses a deposit (the output *add*). Another possibility is that the bank refuses to accept a coin in Step 9, but the payer claims that he never spent that coin before.

We add the following measures:

a) **Withdrawal order:** We require that the withdrawal order, i.e., the message in Step 3 of the withdrawal, is signed and includes the coin. (To protect only from attacks by outsiders, a similar order with symmetric authentication would be sufficient.) Hence it might be

$$sign(sk_{account}, (\text{“coin withdrawal”}, N, seq\_no, amount, blindcoin)),$$

where $sk_{account}$ is a non-anonymous key belonging to the account number N, and seq_no a sequence number.

In a dispute about a statement *subtract* on the account, the bank has to show this withdrawal order. Then:

- If the payer claims that this withdrawal order is replayed, the dispute can be solved with the sequence number like any other dispute about a normal account (see Section 9.3).
- If the payer claims that the bank did not sign the coin, the bank has to sign it again and send *blindsig* via the court to the payer. The court tests the signature.[10]

b) **Coin keys:** What can we do if the bank refuses to deposit a coin, saying that this coin is already in the database, but the payer disagrees and says he never spent it before? The bank might have to prove that the coin's value was added to *some* account, but what if the payer says this is not the recipient for whom he intended the coin? The problem is that a possibility for the payer is missing to specify the destination of a coin in an unforgeable way. (Clearly he cannot sign this under his normal identity because the payment should be anonymous.)

The solution is to introduce “coin keys”, i.e., a specific key with which only the owner of the coin can sign. (This is similar to the pseudonyms under which one owns standard values in Section 17.3.2.) Hence we modify the protocols as follows:

- *gencoin*: We choose *coin* as the public key of a signature scheme, say

$$(sk_{coin}, pk_{coin}) \leftarrow gen_{sig}(l);\ coin := pk_{coin}.$$

- Payment: In Step 7, the payer not only sends (*coin*, *sig*), but also a transfer order, e.g.,

$$sign(sk_{coin}, (\text{“pay”}, N_R, \text{“for”}, ref)).$$

---

10 If the payer lied and the bank has to sign the coin twice, this is no problem, because naive RSA signing is deterministic and the payer just gets the same bitstring twice. With a probabilistic signing algorithm, the bank would have to store the exact signature it gave for the time in which such disputes are allowed.

Here N_R is the account number of the recipient, and *ref* may be some reference string used to link this payment to something else (as a precaution for disputes between payer and recipient). No freshness measure is necessary because each coin is only spent once.

Both the recipient in Step 8 and the bank in Step 9 can verify this signature with respect to $coin = pk_{coin}$.

c) **Deposit confirmation:** In Step 10, the bank not only tells the recipient the result, but gives a signed deposit confirmation (containing the coin, the transfer order and the current time).

In a dispute, the bank can now prove any earlier deposit of a coin using the transfer order. This is secure for the payer if the signature scheme is secure, because he never reveals any information about sk_{coin} except for this one signature.

The scheme is also secure for the recipient because he only gives out a receipt or goods after Step 11, and at that time he has the deposit confirmation which he can use if the money is not added to his account.

If the recipient refuses a receipt, the payer can get (and even enforce) a confirmation by the bank that his coin was deposited. If it was not, he can at that point pay it to himself instead.

D. Recipient Anonymity

We now show how to modify the scheme such that it provides recipient anonymity instead of payer anonymity. The basic idea is that now the recipient must transform the coin. Figure 17.6 shows the resulting scheme. It is very similar to Figure 17.5, only coin generation, blinding and unblinding have been moved from the payer to the recipient. The payer simply forwards the blinded coin to the bank (Steps 3-5) and the blind signature back (Steps 7-9). The bank does not notice the difference at all.

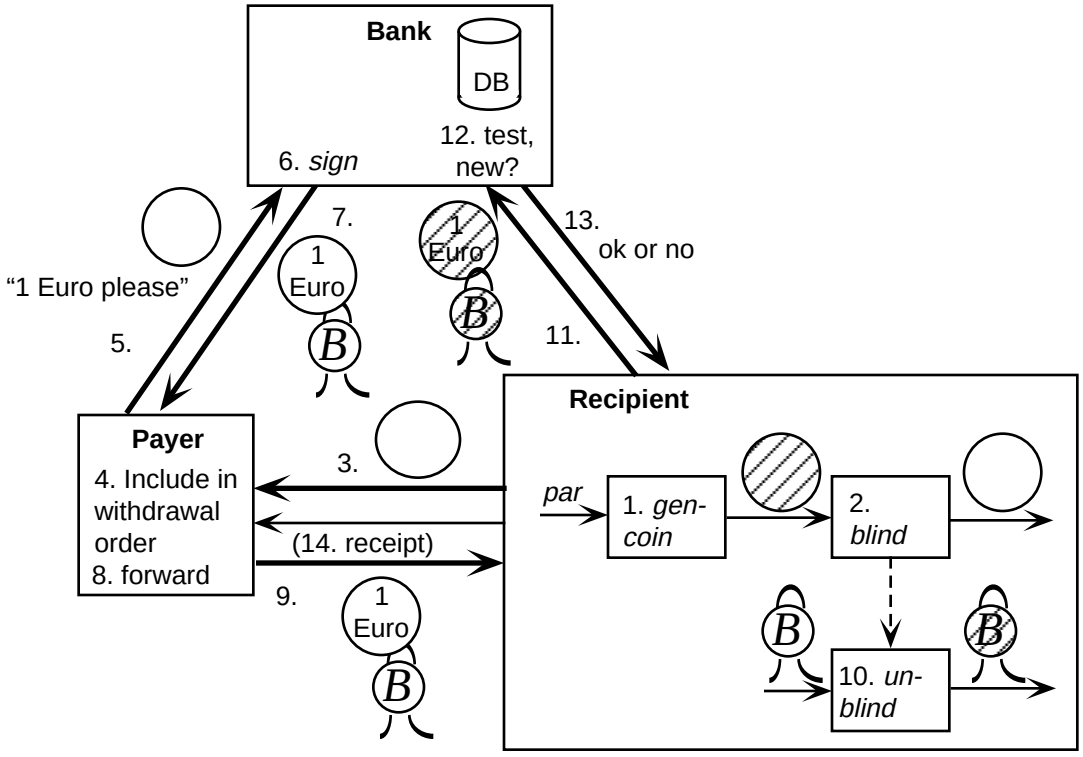


Figure 17.6 Basic blind RSA signatures used for recipient anonymity

Properties of the scheme are:

- It is not prepaid, because the payer does not know in advance to whom he will pay (compare "universality" in Section 17.1.1D).
- To avoid linking by the times of withdrawal and deposit, the recipient should therefore defer the deposit. As he is now the only person knowing the unblinded coin, the payer cannot double-spend it in the meantime. In terms of Section 17.1.1B, we therefore get a reservation system (which made not much sense without the anonymity).
- To provide security in disputes about deposits, the recipient should again produce the coin with a coin key.

E. Maximum Anonymity

We now show that one can combine the different types of anonymity of the schemes from Section 17.3.2 and 17.3.3A-D into a scheme where nothing at all is linkable (beyond what is naturally linked by events outside the payment scheme).

The basic idea is not to pay a received coin into an account, but to immediately exchange it for a new coin. Technically, this is based on the recipient-anonymity system.

We will immediately look at a payment, assuming that the payer P has an undeposited recipient-anonymous coin (as in Subsection D) with a coin key. Let us call this $coin_1 := pk_{coin,1}$ with the secret coin key $sk_{coin,1}$ and the signature sig_1.

Payment:

- The recipient generates a new coin, i.e., a key pair ($sk_{coin,2}$, $pk_{coin,2}$). He sets $coin_2 := pk_{coin,2}$, blinds it into $blindcoin_2$ and sends this to the payer.
- The payer, instead of a deposit order for his old coin to an account, signs an exchange order, e.g.,

$$sign(sk_{coin,1}, (\text{“exchange”}, blindcoin_2)).$$

- Upon receiving ($coin_1$, sig_1) and the exchange order, the bank makes the usual tests: It verifies both signatures (i.e., that the coin is valid and the exchange order was given by the coin owner) and checks in its database that the coin was not deposited or exchanged before. If all is ok, it gives a signature $blindsig_2$ on $blindcoin_2$ and sends this back to the payer.
- The payer now forwards $blindsig_2$ to the recipient, who can unblind it. Then he is in the same situation as we assumed before a payment.

Now there is neither the linkability between successive owners of the same coin that existed with standard values, because the coin is exchanged in each step, nor does all money have to be paid into an account between any two payments. (Hence this is more like metal coins, while standard values were more like banknotes and “coins” with blind signatures have no real analog among classical payment systems.) Note, however, that everything is still online to prevent double-spending. Hence there is not the full independence that one was used to with metal coins (pocket-money, tips etc.).

One can also use exchange transactions to change one coin anonymously into several coins of smaller value or vice versa.

17.4 Non-Anonymous Offline Systems

As explained in Section 17.1.1.D, the main problem with offline operation is to prevent or handle doublespending. Typical non-anonymous offline systems rely mostly on (hopefully) tamper-resistant user devices, e.g., smartcards, for this purpose.[11] Then the "electronic money" on the user devices does not need a complicated representation like coins, but can simply be held in a balance variable. Thus the user device is a bit like a "pocket bank" handling one electronic account and taking care that only permitted operations on the account are carried out.

In addition to the tamper resistance, the devices also need cryptography because they must recognize when they talk with other secure devices and are therefore allowed to change their balance.

The main disadvantage is that tamper resistance is very doubtful, see Chapter 12. Another disadvantage is that two different parties, the user and the bank, must trust in the same user device. For instance, the banks might prefer the design to be secret, while for the users' security public verification would be better.

17.4.1 With Signatures

Conceptually, the easiest way to realize a balance-based system is to use a signature system [EvGY_83]: Each device D generates a key pair (sk_D, pk_D). The bank gives an attribute certificate $cert_D$ on pk_D with an attribute like "this is one of my secure devices". For the offline operation, each device contains its own certificate. Each device has a balance bal_D initialized with 0.

- In a withdrawal, the device gets a signed fresh message (e.g., using a nonce) from the bank that it can now increase its balance, e.g.,

  $$m = sign(sk_B, (\text{“}increase\text{”}, D, nonce, amount)).$$

- In a payment, the main message m is from the payer's device P to the recipient's device R. It contains the identities of both devices and the amount x of the payment. Again, a freshness measure is necessary. E.g., in a prior message, R could send P its identity and a nonce, and then

  $$m = sign(sk_P, (\text{“}pay\text{”}, P, R, nonce, amount)).$$

  It must also attach the attribute certificate $cert_D$.

  Before sending off m, the payer's device P sets $bal_P := bal_P - x$. After receiving m, the recipient's device R sets $bal_R := bal_R + x$. This must happen in this order; otherwise the two parties could deliberately lose the message to increase their joint amount of money.

We don't go into details of enforcing receipts and of disputes about withdrawals and deposits here.

This is a prepaid deposit-later system (like all offline systems). It can be transferable without additional cryptographic problems, i.e., devices may be used for both paying and receiving, and received money can be added into the same balance from which one pays.

---

[11] There are almost no proposals that one should entirely rely on punishing double-spenders afterwards. One reason is that in particular in a network scenario, one can do a lot of double-spending in a short time. Another is that most of the systems are not entirely secure against theft (of a computer or card), and measures against double-spending typically also limit the amount a thief can spend.

One well-known system in field trials might be of this type, Mondex. They keep their specifications secret, even the high-level descriptions. It is assumed that the field trials are simple systems as in Section 17.4.2., but that they hope to progress to an asymmetric system because they at least considered transferability by giving the users not only smartcards, but real devices that can interact. The EMV (Europay, Mastercard, Visa) standardization efforts for an offline systems allow either signatures or symmetric systems, but they do not define a full system. The implementation CLIP by Europay is asymmetric.

17.4.2 With Symmetric Authentication

In practice, the typical offline systems do not yet use signatures, but only symmetric cryptography. Apart from the standard disadvantages of offline systems that there are no proofs, this leads to a technical problem: How can one distribute the keys?

a) One can give all the devices the same key. If the devices really were perfectly tamper-resistant, this would be no problem. (E.g., the payment messages similar to Section 17.4.1 contain the device names, hence a cheating user cannot make two devices increase their balance by sending the same message to both of them. Actually, even nonces alone would ensure this.) However, nobody trusts at least smartcards that much.

b) It is not possible to have a different key for each pair of devices (as one would usually expect in a symmetric system): All payer devices should be able to talk to all recipient devices. This is offline, so one cannot use the standard technique with a central server and master keys (see Section 8.2.1). And there are so many devices that one cannot assume that the keys are distributed in advance.

c) Hence the usual **solution** is as follows: The recipient devices (point-of-sales terminals, more expensive and hopefully more tamper-resistant) all get the same key K, which is called a master key. However, the payer devices are not allowed to see K. Instead, each one has its own device key k_D. The trick to allow these devices to communicate is to derive the device key k_D from the device number n_D with a secret operation involving the master key, e.g., $f(K, n_D)$ where f is a one-way function. (A pseudo-random function seems useful cryptographically.) Hence each recipient device, once it gets the number n_D of a payer device, can use the master key to derive k_D. Then they can use k_D to communicate securely in the usual way (i.e., a symmetric version of Section 17.4.1).

This solution was adopted in most trials with smartcards (e.g., Danmønt and Proton) and standardized as “CEN Intersector Electronic Purse” (CEN is a European standardization organization, composed of national bodies like DIN). The German Geldkarte is such a system.

Obviously the systems with master keys only in special recipient devices cannot be transferable. Hence it is assumed that Mondex (cf. Section 17.4.1) had the same key in all devices and that this is why they kept the design secret.

17.5 Anonymous Offline Systems

Now we combine both technical difficulties, anonymity and offline operation.

17.5.1 Relying on Tamper-Resistance

If one fully relied on tamper resistance, one could use the system from Section 17.4.2a, where all devices have the same key. If they also have no numbers etc., such a system can be anonymous. (Recall that unique use of a message "pay" can be guaranteed by nonces alone.)

However, as mentioned above, one typically does not trust the devices so far, i.e., one wants to have some fall-back mechanism by shadow accounts where the bank monitors the balance on each card in order to detect doublespending. This means that whenever the devices have contact with the bank (for withdrawals or deposits), they send all their transaction records to the bank, who sorts them by accounts and checks whether no device spent more money than it withdrew. This cannot be done without device identifiers.

Anonymity also aggravates the problem with tamper-resistant devices that the banks might want to keep the design secret — how does one know that the devices do not contain trapdoors that transfer the identity after all?

17.5.2 Basic Ideas for Systems with Doublespender Identification

Now we consider systems that are anonymous but where doublespenders can be identified and punished. As mentioned in Section 17.1.1.D, this could also be combined with tamper resistance so that doublespending becomes hard in the first place.

A. Basic Ideas

The systems are based on the coin concept (the anonymous standard values are even more intrinsically an online system). If a simple coin system were used offline, the bank could see afterwards that a coin was spent several times, but how can it react if the coins are anonymous?

The idea in [ChFN_90] was to take cryptographic measures such that doublespenders are no longer anonymous. For this, two payments with the same coin (and the subsequent deposits) must actually give the bank more information than one payment — otherwise it would be impossible that one is anonymous after one payment, but no longer after two. This is done by a challenge-response mechanism: In each payment, the recipient asks a question and the payer has to send a certain answer, see Figure 17.5. With one such answer, the payer is still anonymous, but answers to two different questions identify him.

The answers can only reveal the identity if the identity is somehow encoded into the coin in the first place, i.e., during the withdrawal where the bank can verify this. If we keep the information-theoretic anonymity, the identity can no longer be present in the transformed coin in a strict sense, because every paid coin should in principle fit to every withdrawal. However, the payer will need to "know" something about a coin for paying (somewhat like the secret coin key in Section 17.3.3). This knowledge will still be linked to the identity.

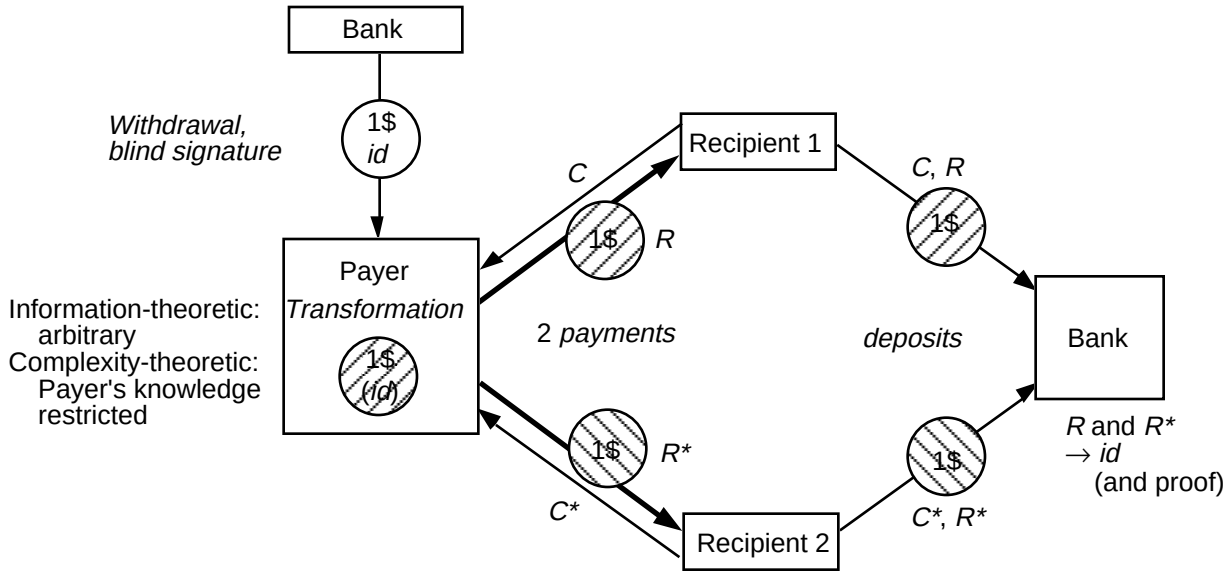


Figure 17.5 Basic ideas for doublespender identification

B. On the Challenges

One more problem can be considered abstractly: What happens if a payer and two recipients collude and pose the payer the same challenges, i.e., $C^* = C$? The bank can certainly not give the money to both recipients in the deposit. One could say that only the first recipient who deposits gets the money. But then a payer and Recipient 2 could cheat Recipient 1 by reusing C and depositing faster. Therefore each challenge will include the recipient's identity, and deposit will only be possible to the account of this recipient. (This is a special case of the coin key mechanism from Section 17.3.3: The response is at the same time a signature on the recipient's identity.)

C. Realization Ideas for the Responses

A first idea what to use as responses would be that the first answer R is a one-time pad k, and the second answer is $R^* = id \oplus k$. Then obviously each answer alone gives no information about id, but both answers reveal id in full. However, this would only allow two challenges.

The next idea is that a response is a linear equations in two variables, one of which is id, e.g., $c_1 id + c_2 k$. This give al large set of possible challenges (c_1, c_2). Then one response still leaves every id possible, while two equations determine both variables uniquely (assuming the equations are linearly independent.) This idea is from [FrYu_93]. Essentially, it is also used in Brands' system, which we describe below. Only there, each response gives two equations in four variables — this allowed Brands to come up with an efficient verification whether the responses are correct.

17.5.3 Brands' System

Brands' system is the best-known anonymous offline system because it is quite efficient. It is not provable, just like any other of these efficient systems.

It uses a blind signature scheme based on discrete logarithms, an extension of the Chaum-Pedersen scheme [ChPe1_93], which can be seen as an extension of Schnorr signatures. We already

saw Schnorr signatures in Section 6.4.2.E.γ. However, for understanding the following modifications, it is useful to reconsider the Schnorr's system from another point of view.

A. Schnorr Signatures Revisited

Recall that key generation for Schnorr signatures results in a group G_q of order q, say a subgroup of $\mathbb{Z}_p^*$, a generator g of G_q, a secret key $x \in \mathbb{Z}_q$, and $h = g^x$ as the main part of the public key. The signature scheme was originally invented as a variant of an identification system. In the identification system, the owner of x simply "proves that he knows x". (More details on what this means below.) The proof protocol is shown in Figure 17.6.

	Prover (Secret: x)	Common input: $q, p, g, h = g^x$	Verifier
1. Random variant	$w \in_R \mathbb{Z}_q$ $a := g^w$	— a →	
2. Challenge		← c —	$c \in_R \mathbb{Z}_q$
3. Response	$r := w + c\,x$	— r →	Verify $g^r = a\,h^c$

Figure 17.6 Schnorr identification protocol

The idea is the following, and this will be the same in the modified protocols: In Step 1, the owner of the secret, here x, chooses another secret of the same form, here w. It will more or less serve as a one-time pad for x. He also computes the equivalent of the public value (here h) with w; here this is $a = g^w$.

In Step 2, the verifier sends a challenge c.

In Step 3, the prover has to respond with a linear combination between x and w (his real secret and his one-time secret) determined by c. The verifier can test whether this linear combination is correct by verifying its homomorphic image for the public values: If all was correct, then g^r should be $g^{w+cx} = g^w(g^x)^c = a\,h^c$.

Remark. In an even more basic protocol from [ChEG_88], the challenge c is only one bit, but then one has to repeat the protocol several times. The advantage is that the security is more provable (a stronger property called zero-knowledge then holds instead of the informal Property 2 below).

Security ideas. We would like the identification protocol to have two properties. Both cannot be defined formally in the available space and time (see the course on higher cryptographic protocols):

1. "Proof of knowledge": If a prover can successfully pass the verification with significant probability, then he must "know" x.

   In our case, the proof idea is the following: Consider a prover that has sent a. Assume that he can answer at least two out of the many possible challenges c that he might now get. Let these challenges be c_1 and c_2 and the answers that he would send be r_1 and r_2, respectively. The values x and w are uniquely defined as the discrete logarithms of h and a. By Proposition 1 from Section 5.5.1, r_1 and r_2 pass the verification only if in fact

$$r_1 = w + c_1 x \text{ and } r_2 = w + c_2 x \pmod q.$$

Hence, from (c_1, c_2, r_1, r_2) one can easily compute x by solving this equation system, i.e.,

$$(r_1 - r_2) = (c_1 - c_2)x \mod q.$$

This counts as having known x. (This was only a sketch because, e.g., the probabilities were not considered and the question why "being able to compute" is a suitable notion of "knowledge".)[12]

2. "No useful knowledge leaks": Even if the prover carries out many such proofs, this will not make it easier for the recipient to find out x or do other useful things with x. The basic idea here is that for any specific c, the value w is a one-time pad on x in the response r that the attacker gets. However, this is not a real proof, since w is not completely hidden because a also belongs to the attacker's view.

Signatures from identification protocols. There is a general trick by Fiat and Shamir to transform 3-step identification protocols where the verifier only sends a random challenge c into a signature scheme [FiSh_87]. However, the quality of the trick is not proven.

The trick is that the signer computes c himself as

$$c = hash(m, a),$$

where m is the message to be signed and *hash* a hash function that is at least one-way. The hope is that the hash function is somehow pseudorandom, so that the resulting c is as good as the real random c in the identification protocol. However, this is only a heuristic because *hash* is public (otherwise the signature could not be tested), and thus of course anyone can distinguish it from a real random function.

The signature test is now

$$g^r = a\, h^c \text{ for } c = hash(m, a).$$

Remark (Comparison with the presentation in Section 6.4.2): Instead of sending a and r and letting the recipient recompute c, one can also send c and r and let the recipient recompute a as h^c/g^r and then verify the hashing. The version in Section 6.4.2 corresponds to this: The triple (z^*, r, s) there is our (w, c, r). The remaining difference is that Schnorr used a "–" instead of a "+" in Step 3.

B. Chaum-Pedersen Signatures Without Blinding

As a second step, we consider Chaum-Pedersen signatures without blinding. This protocol is shown in Figure 17.7. The changes with respect to Figure 17.6 are highlighted.

---

12 Check as an exercise that it is really necessary to argue with two challenges here, i.e., how could someone cheat who did not know x but who did know c in advance?

	Signer (Secret: x)	Common input: $q, p, g, h = g^x$	Recipient
0. Signature commitment	$\boldsymbol{z := m^x}$	$\boldsymbol{— z \rightarrow}$	
1. Random variant	$w \in_R \mathbb{Z}_q$; $a := g^w$; $\boldsymbol{b := m^w}$	$— a, \boldsymbol{b} \rightarrow$	
2. Challenge		$\leftarrow c —$	$c \in_R \mathbb{Z}_q$
3. Response	$r := w + c\,x$	$— r \rightarrow$	Verify $g^r = a\,h^c$; and $\boldsymbol{m^r = b\,z^c}$

Figure 17.7 Chaum-Pedersen signatures, interactively and without blinding

The main novelty is the value $z = m^x$ in Step 0. We call it a signature commitment on m because, like a signature, it can only be computed with the secret key; however the recipient cannot verify its correctness on his own. The rest of the message exchange is intended to convince the recipient that z is correct.

This proof is exactly of the same structure as above: In Step 1, the random variant w of x is chosen. As there is now an additional public value z based on x, there is also a similar value b based on w. In Step 3, the verification is done both for the powers of g (Line 1) and for the powers of m (Line 2).

The security considerations are as before. In particular, if one can answer two challenges for the same a and b, one can again easily compute a value x such that both $h = g^x$ and $z = m^x$ hold. Thus the protocol proves both that the sender "knows" x and that z is correct.

Finally, one can again replace the random c by a hash value

$$c = hash(m, z, a, b).$$

The general rule is to use all the values from the previous steps as the inputs to *hash*.

C. Blind Chaum-Pedersen Signatures

Now we add the steps where the recipient transforms the signature. Compared with Figure 17.4, the message m will be the white form of the coin and m' the striped form. The blind signature is the values that are sent, i.e., $\sigma = (z, a, b, c, r)$ and the unblinded signature is $\sigma' = (z', a', b', c', r')$. The idea that makes only the striped form valid for paying is that only in that form, $c' = hash(m', z', a', b')$ will hold. Again the changes to the previous protocol are highlighted.

	Signer (Secret: x)	Common input: $q, p, g, h = g^x$	Signature recipient
0. Signature commitment	$z := m^x$	— z →	$\mathbf{s, t, u, v \in_R \mathbb{Z}_q}$ **with** $\mathbf{s \neq 0}$; $\mathbf{m' := m^s g^t}$; $\mathbf{z' := z^s h^t}$
1. Random variant	$w \in_R \mathbb{Z}_q$ $a := g^w$; $b := m^w$	— a, b →	$\mathbf{a' := a^u g^v}$; $\mathbf{b' := b^{su} a^{tu} m'^{,v}}$
2. Challenge		← c —	$\mathbf{c' := hash(}m', z', a', b'\mathbf{)}$; $\mathbf{c := c' u^{-1}}$
3. Response	$r := w + c\,x$	— r →	Verify $g^r = a\,h^c$; and $m^r = b\,z^c$; $\mathbf{r' := u\,r + v}$

Figure 17.8 Blind Chaum-Pedersen signatures

The values (s, t, u, v) are called blinding factors. Later for the Brands system we will always set $t = 0$, but it is useful for security to know that blinding with $t \neq 0$ is also possible. We now look why this scheme works:

- **Correctness**: We show that the result σ' is a correct Chaum-Pedersen signature on m' just as above. It is clear by the construction that $c' = hash(m', z', a', b')$. We have to verify that

$$g^{r'} = a'\,h^{c'} \quad \text{and} \quad m'^{r'} = b'\,z'^{c'}.$$

  We can do this simply by substituting all equations into each other. However, the underlying ideas become clearer by showing that all the values on the right side have the same structure as those on the left. First,

$$z' = z^s h^t = m^{xs} g^{xt} = (m^s g^t)^x = m'^x,$$

  just as it should be. Next

$$a' = g^{wu+v}.$$

  Thus $wu + v$ plays the same role for a' as w for a and we abbreviate it as

$$w' = wu + v.$$

  Next we show that also $b' = m'^{w'}$:

$$b' = b^{su} a^{tu} m'^{,v} = m^{wsu} g^{wtu} m'^{,v} = (m^s g^t)^{wu} m'^{,v} = m'^{wu+v} = m'^{w'}.$$

  Finally,

$$r' = u\,r + v = u(w + cx) + v = (wu + v) + ucx = w' + c'x.$$

  This is also as it should be, and hence we know by the correctness of normal Chaum-Pedersen signatures that our signature also fulfils the test equations.

- **Security against one-more forgery:** Here nothing is really known. We simply have to assume that it is impossible to get more valid signatures than the number of blind signing protocols one executes. The basic idea is that one hopes that although one might be able to manipulate all the other values in different ways, one will not end up with $c' = hash(m', z', a', b')$ except by choosing everything exactly in the order as above.
- **Blindness**: Here we show that any message with a valid signature (m', σ') fits any view (m, σ) that the signer gets in a blind signature protocol. This will later mean that any coin could result from any withdrawal protocol. The proof is again by the sufficient criterion for the general secrecy formula, see Section 6.1.1.B: First, the blinding factor quadruples (s, t, u, v), which play the role of the key, are uniformly distributed. Secondly, we show that for each pair $((m', \sigma'), (m, \sigma))$ there are exactly q blinding factor quadruples that transform (m, σ) into (m', σ').

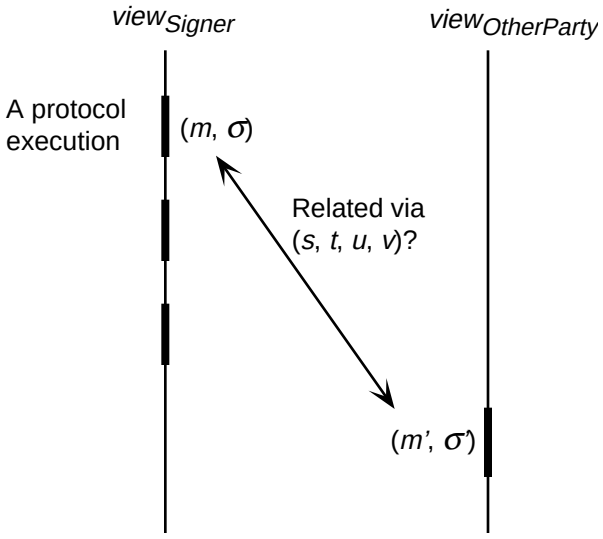


Figure 17.9 Situation for the blindness proof: A signer has given many blind signatures. One of them, (m', σ') is now shown to another party. We want to prove that it could come from any of the executions of the signing protocols, e.g., that where the signer's view was (m, σ).

We first show that there are *at most q* solutions:

- First u can only be c'/c (see Step 2).
- Then v can only be $r' - ur$ (see Step 3).
- Now we write m as g^{α} and $m' = g^{\beta}$. Then $m' = m^s g^t$ means $\beta \equiv \alpha s + t \bmod q$. This equation has q solutions for (s, t).

In the rest of the proof, we show that each of these tuples (s, t, u, v) indeed leads to exactly the signature σ'. (It is clear by construction that it leads to the same m'.) Fix such a tuple and call the resulting signature σ'' (because we must have a name for it as long as we haven't finished the proof that it equals σ').

We first show that a, b and z in any protocol run where the recipient accepts are of the correct form (except with negligible probability) although the signer may be among the attackers:

- We can assume that $z = m^x$ because the protocol proves this fact. (Actually, here is a gap in the proof that nobody seems to have noticed before: It was only shown that the protocol proves this fact if *hash* is random-like. Now, however, we would like a real information-theoretic argument, and this does not seem to be true. Probably a rather weak assumption on *hash* is sufficient for what we need here, though.)
- As to a, we simply define w such that $a = g^w$.
- Now let $b = g^{w*}$. Then indeed $w* = w$ because the recipient verifies $g^r = ah^c$ and $m^r = bz^c$, which is equivalent to $r = w + xc$ and $r = w* + xc \bmod q$.

Given this, the proof of the correctness implies that the signature σ" is constructed like a correct σ starting with the values m' and $w' = wu + v$. Similarly, σ' was constructed in some withdrawal (probably not the one we currently consider) from the same m' with a value w'_0. Thus, to show that σ" = σ', it is sufficient to show that $w'_0 = w'$. We have

$$\begin{aligned} w' &= wu + v = wu + r' - ur = r' + u(w - r) \\ &= r' + (c'/c)(w - w - xc) = r' - c'x. \end{aligned}$$

By the correct construction of σ' from w'_0, we therefore obtain

$$w' = (w'_0 + c'x) - c'x = w'_0.$$

- **Blindness for $t = 0$**. We easily see in the proof above that among the q solutions, there is always exactly one with $t = 0$ unless $\alpha = 0$, i.e., as long as we exclude $m = 1$. This will be important in Brands' applications where honest recipients always use $t = 0$. So we still have information-theoretic anonymity in that case.

D. Ideas for Encoding Identities into Coins

So far, we have a blind signature scheme and can use it for an anonymous online payment system just like the blind RSA signatures. Now we want to add doublespender identification as shown in Figure 17.5. Thus we now have to encode identities (more precisely: numbers linked to identities) into the coins and devise the challenge-response mechanism for the payments.

As the bank can only verify that the correct identity is present in the coin m it signs, while in the payment m' will be used, the encoded identity must in some way survive this blinding process. This property is called restrictiveness.

Idea for restrictiveness. The essential idea is the following: m must be of the form

$$m = g_1^{id} g_2$$

for two generators g_1, g_2, which are different from the g used so far, and the payer must know id. Then if blinding can *only* be done as in the correct protocol (see Figure 17.8), the result will be

$$m' = m^s g^t = g_1^{id \cdot s} g_2^s g^t.$$

The payment protocol will somehow enforce that the payer must know exponents x_1, x_2 such that

$$m' = g_1^{x_1} g_2^{x_2}.$$

One can show that triple exponentiation, i.e., the function $(x_1, x_2, x_3) \to g_1^{x_1} g_2^{x_2} g^{x_3}$ is collision-free.[13] Thus the two representations of *m'* using the triples $(id \bullet s, s, t)$ and $(x, y, 0)$ must be the same. Hence the payer must choose $t = 0$ in the blinding process and we have

$$(x_1, x_2) = (id \bullet s, s).$$

Hence the "identity" *id* is still present as $id = x_1/x_2$. Now doublespending will reveal all the secret values, in particular x_1 and x_2, and therefore *id*.

The precise restrictiveness assumption. The restrictiveness as just sketched cannot be proved at the moment. One can give some heuristic arguments why it is hard to find any other form of blinding etc., but finally one simply has to make the following assumption (slightly more general than what is needed):

Let a probabilistic polynomial-time attacker *A* be given that can interact with a Chaum-Pedersen signer several times for messages *m* of his choice. However, he has to output representations of all these messages, i.e., pairs (i_1, i_2) such that

$$m = g_1^{i_1} g_2^{i_2}.$$

At the end, he has to output a message *m'* with a valid signature. He also has to output a representation (x_1, x_2) of *m'*.

The actual assumption is: The probability that *A* fulfills all the conditions and that the pair (i_1, i_2) is not a scalar multiple (mod *q*) of one of the pairs (i_1, i_2) is negligible. (The probability is taken over all the random choices of the signer and the attacker.)

E. The Complete System

Now we put all the parts together and fill in the actual payment protocol.

Bank setup. The bank generates keys for the Chaum-Pedersen blind signature scheme, i.e., *p*, *q*, *g*, *x*, and $h = g^x$ where *x* is secret and the rest is published. More precisely, for different denominations it would use the same *p*, *q*, and *g*, but different values x_j and h_j, but we will omit the index *j* in the following. Moreover, expiration dates may be associated with these keys, i.e., coins signed with those keys are only valid for a certain time.

The bank also generates two further random generators g_1 and g_2. Everybody has to trust that even the bank cannot compute discrete logarithms with respect to g_1. In practice, one might believe that this is true even if the bank chooses *p*, *q* and g_1 arbitrarily. However, one can also guarantee their randomness by using an arbitrary trusted random string *r* (typically taken from old tables of random numbers) and generating the desired values from *r* using a standardized deterministic procedure. In particular, g_1 is then chosen as $r_1^{(p-1)/q}$ where r_1 is a substring of *r* interpreted as an element of $\mathbb{Z}_p^*$.

Opening an account. This protocol is executed once for each new payer. "Opening an account" is the usual name, but in reality one would probably use a normal bank account, and this is just a specific registration phase for this payment system.

0. For secure electronic withdrawals, we have to assume that the payer has a key pair (sk_P, pk_P) of an arbitrary signature scheme with which he can authorize actions regarding his bank account.

---

[13] The proof will not be shown in the following. For pairs it was shown for the hash function in Section 6.5.2A. For triples and larger tuples it can be found in [ChHP_92].

This pk_P may also be registered at this moment or may already exist, e.g., from a standard homebanking protocol.

1. The payer chooses a secret random number *id* mod *q* and gives $h_1 := g_1^{id}$ to the bank. We call *id* the identity proof of this payer. If another payer already uses h_1, the bank asks the current payer to repeat his choice.
2. The payer proves to the bank that he knows *id* with $h_1 = g_1^{id}$. For this, they use the Schnorr identification protocol; see Figure 17.6.
3. The payer signs (using sk_P) that $m = h_1 g_2$ will be used for his withdrawals. If the bank can later show the value *id* with $g_1^{id} g_2 = m$, this will count as a proof that a coin from this payer was spent twice.

Registering as a recipient. As mentioned in Section 17.5.2A, each recipient must use a different challenge. Thus it is useful if he registers under a specific digital identity $Id_{Recipient}$. However, recipients do not need keys for this system.

Withdrawal. For one payer, the bank always blindly signs the same message *m* which was agreed upon above. The protocol is that from Figure 17.8 with the following addition:[14]

- The payer generates two additional secret numbers y_1, y_2 mod *q*.
  Together with (x_1, x_2) they are the four secret variables about which each payment will reveal two linear equations. At the same time, one can see (x_1, x_2, y_1, y_2) as a secret coin key sk_{coin} with which the payer, as the anonymous owner of this coin, can later sign to whom he wants to pay it. (Compare the measures for secure disputes about deposits in Section 17.3.3.)
- He sets $pk_{coin} := g_1^{y_1} g_2^{y_2}$.
  Recall that also $m' = g_1^{x_1} g_2^{x_2}$. Together, the values (m, pk_{coin}) can be seen as the public coin key corresponding to sk_{coin}.
- To fix which pk_{coin} belongs to which coin, this value is included in the hashing process in Figure 17.8, i.e., the challenge is actually computed as

$$c' := hash(m', pk_{coin}, z', a', b').$$

For security in disputes about a withdrawal, the payer should sign (using sk_P) a withdrawal order including the desired denomination of the coin and the value *c'* for which he wants a response. This withdrawal order must be sent together with Step 2 of the blind signature protocol.

We call the result of a withdrawal

$$coin' = (m', pk_{coin}, \sigma') \quad \text{with} \quad \sigma' = (z', a', b', c', r').$$

Here *m'* is called the coin identifier and σ' the coin signature.

Payment. First the payer sends *coin'*, whose components are named as above, and the recipient verifies that it is correctly signed with a Chaum-Pedersen signature. The rest of the payment protocol follows the ideas described in Section 17.5.2, in particular that each response reveals two linear

---

14 Moreover, as *z* is now the same in each protocol execution, the payer can store it once and for all so that the message in Step 0 is not needed. But of course, new values *w* and (*s*, *t*, *u*, *v*) must be chosen in each new withdrawal.

equations about the four variables (x_1, x_2, y_1, y_2). The trick is how to verify that the responses are correct. Once again, this is done by verifying corresponding equations in the exponents.

Remark: The protocol follows the same structure as Figure 17.6 except that the "random variant" of (x_1, x_2) is the already given (y_1, y_2). The corresponding public values are $m' = g_1^{x_1} g_2^{x_2}$ and $pk_{coin} := g_1^{y_1} g_2^{y_2}$. The linear combination in the response is done by scalar multiplication and addition of pairs, $R = C(x_1, x_2) + (y_1, y_2)$. If everything was correct, we should have

$$m'^{C} pk_{coin} = (g_1^{x_1} g_2^{x_2})^{C} (g_1^{y_1} g_2^{y_2}) = g_1^{x_1 C + y_1} g_2^{x_2 C + y_2}.$$

The resulting protocol is shown in Figure 17.10.Here, *hash'* is another hash function. The value *nonce* stands for any freshness measure that guarantees to this recipient that he won't accept that same coin twice with the same challenge.

	Payer $coin', sk_{coin}$	Public q, p, g, h, g_1, g_2	Recipient
1. Send coin		— $coin'$ →	Verify: $c' = hash(m', z', pk', a', b')$; $g^{r'} = a' h^{c'}$ and $m'^{r'} = b' z'^{c'}$; $m' \neq 1$
2. Challenge	$C = hash'(m', pk', Id_{Recipient}, nonce)$	← $nonce$ —	$C = hash'(m', pk', Id_{Recipient}, nonce)$
3. Response	$R := (sig_1, sig_2)$ with $sig_1 = Cx_1 + y_1$; $sig_2 = Cx_2 + y_2$	— R →	Verify $g_1^{sig_1} g_2^{sig_2} = m'^{C} pk'$

Figure 17.10 Brands' payment protocol

Deposit and Doublespender Identification. In a deposit, the recipient forwards all his information from a payment, including $Id_{Recipient}$ and *nonce*.

1. The bank first verifies the validity of the coin just as the recipient did in Steps 1 and 3 of the payment.
2. Secondly, it detects double-spending or double-depositing by entering the coin into a data structure *cointable*, say a hash table, for comparison with previous deposits by other recipients. This can be done offline. Coins stay in this table until they expire.
   - If the coin is already there, the bank checks whether the recipient and nonce were the same or not. If yes, the recipient double-deposited and has to pay back.
   - If not, and unless someone found a collision of the function *hash'*, the two challenges C and C^* are also different. If the coin is already in the table, the bank retrieves the corresponding responses R and R^*, computes the identity proof as $id = (sig_1 - sig^*_1)/(sig_2 - sig^*_2)$ and looks up to which payer $m = g_1^{id} g_2$ belongs.

The correctness of the computation of *id* can be seen immediately from the verification equations: We have

$$g_1^{sig_1} g_2^{sig_2} = m'^{C} pk' \quad \text{and} \quad g_1^{sig^*_1} g_2^{sig^*_2} = m'^{C^*} pk'$$

and therefore

$$g_1^{sig_1 - sig^*_1} g_2^{sig_2 - sig^*_2} = m'^{C - C^*}.$$

This implies that the payer knows a representation of m' as $g_1^{x_1} g_2^{x_2}$ with $x_1 = (sig_1 - sig^*_1)/(C - C^*)$ and $x_2 = (sig_2 - sig^*_2)/(C - C^*)$. Now the restrictiveness assumption is used to show that $x_1/x_2 = id$.

Remarks

- **On punishment:** Note that you cannot find out how many coins where doublespent. This could be changed by using a different *id* for each coin, but still one could not prove how often a coin was spent so one cannot exactly prove how much damage the payer caused. (One could check how many different signatures with this coin key exist, but once that payer made two such signatures, the secret coin key can be found out and thus the recipients could also take the opportunity and make additional payments with this coin.)
- **On proofs:** Given the restrictiveness assumption and assumptions that protocols using hash functions instead of random challenges are still proofs of knowledge, it should be possible to prove higher-level properties of the protocol, but we don't do this now.
- **On implementations:** A system that was essentially based on Brands' payment system was implemented in the project CAFE [BBCM1_94].

Literature

AASW_98 Jose. L. Abad-Peiro, N. Asokan, Michael Steiner, Michael Waidner: Designing a generic payment service; IBM Systems Journal 37/1 (1998) 72-88.

AJSW_97 N. Asokan, Phillipe A. Janson, Michael Steiner, Michael Waidner: The State of the Art in Electronic Payment Systems; Computer 30/9 (1997) 28-35.

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 (Third European Symposium on Research in Computer Security), LNCS 875, Springer-Verlag, Berlin 1994, 217-230.

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

BüPf_90 Holger Bürk, Andreas Pfitzmann: Value Exchange Systems Enabling Security and Unobservability; Computers & Security 9/8 (1990) 715-721.

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

Chau_81 David Chaum: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms; Communications of the ACM 24/2 (1981) 84-88.

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

Chau_89 David Chaum: Privacy Protected Payments – Unconditional Payer and/or Payee Untraceability; SMART CARD 2000: The Future of IC Cards, IFIP WG 11.6 International Conference, Laxenburg (Austria) 1987, North-Holland, Amsterdam 1989, 69-93.

ChEG_88 D. Chaum, J.-H. Evertse, J. van de Graaf: An improved protocol for demonstrating possession of discrete logarithms and some generalizations; 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.

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

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

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 International Colloquium on Automata, Languages and Programming (ICALP), LNCS 700, Springer-Verlag, Berlin 1993, 265-276.

HBCI_99 Homebanking Computer Interface - HBCI (Version 2.1); Zentraler Kreditausschuß, March 1999 (http://www.siz.de/siz/hbcispec_e/hbcisp_e.htm).

Pede_97 Torben P. Pedersen: Electronic Payments of Small Amounts; Security Protocols 1996, LNCS 1189, Springer-Verlag, Berlin 1997, 59-68.

PfWa_96 Birgit Pfitzmann, Michael Waidner: Properties of Payment Systems: General Definition Sketch and Classification; IBM Research Report RZ 2823 (#90126) 05/06/96, IBM Research Division, Zurich, May 1996.

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.

Poin_98 David Pointcheval: Strengthened security for blind signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 391-405.

PWP_87 Birgit Pfitzmann, Michael Waidner, Andreas Pfitzmann: Rechtssicherheit trotz Anonymität in offenen digitalen Systemen; Computer und Recht 3/10,11,12 (1987) 712-717, 796-803, 898-904; Überarbeitung und Erweiterung erschien in zwei Teilen in Datenschutz und Datensicherung DuD 14/5-6 (1990) 243-253, 305-315.

Simo_96 Daniel Simon: Anonymous Communication and Anonymous Cash; Crypto '96, LNCS 1109, Springer-Verlag, Berlin 1996, 61-73.

Index