# Electronic Cash

Matthijs Mekking

June 2005

# Index

# Introduction

The Internet is widely used e.g. for payment transfers. In some online payments you need to give free your credit card credentials. Paying by credit card is currently very popular, but it's vulnerable with aspect to the security and privacy issues. Internet payment systems can provide anonymity and other drawbacks of credit card payment (e.g. off-line payment), but there are still issues that can be considered, such as traceability and double-spending problems.

Payment systems control transfers of money between different accounts. In general, there are three stages. First of all, a customer requests an amount of money at the bank. The bank verifies some requirements, such as has the person an account and is the balance above a certain limit. If the bank agrees with the request, he lowers the account of the customer and gives him or her the money. This first phase is referred to as the withdrawal. The second stage is the transfer stage. After the withdrawal, the customer wants to spend his money, for example at a shop. The shop owner will ask money for the services or products he provides. The customer pays the employee and he verifies if the money is valid. After the payment, the final stage follows. The money is send to the bank. The money will be verified by the bank again and will deposit it on the account of the shop owner. This is called the deposit stage. In Figure 0.1 the three stages are illustrated.

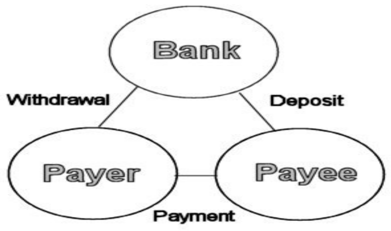

**Figure 0.1 General electronic payment system scheme**

Normally, the accounts and the value involved in the transfer have to be kept confidential [2]. This means that the bank needs consent of the payer and/or the payee to trace a particular transaction. We can identify four different consent conditions:
payer only (payer untraceability)
payee only (payee untraceability)
payer and payee (mutual untraceability)
payer or payee (bilateral untraceability)

Payer untraceability provides privacy by preventing payments from being traced back to the client's account. With payee untraceability, people can receive cash (e.g. change) without permitting these payments to be traced. The final two conditions can be constructed by combining and modifying the first two conditions.

The first chapter will discuss the payer untraceability protocol. It will also consider the security of this protocol. We will see it is secure under all conditions. The payee untraceability is considered in chapter two. Also the final two consent conditions come back in chapter two. Another issue is the time value of money, such as interest earnings and charges. We will see how this works in chapter three. The efficiency and performance in large-scale consumer payment systems is considered in chapter four. In chapter five we will discuss the advantages and disadvantages of unconditional privacy, which is provided by the protocols discussed in this paper. We will see a protocol that provide revocable privacy, referred to as a fair tracing method. The paper will close with a brief overview and some conclusions in chapter six. I finally want to mention that no encryption is considered. This is not the essence of the paper, and it is clear that it can be introduced easily to the proposed systems. We also assume that there exists hardware that is able to put the protocols in effect, although we will consider some computational and storage requirements in chapter four.

# 1. Payer Untraceability

The payer untraceability protocol is based on *blind signatures*. Before the actual protocol is described, this concept will be introduced.

## *1.1 Blind Signatures*

Blind signatures are an extension of the RSA digital signature [3]. A digital signature is an electronic signature that can be used to authenticate the identity of the sender of a message. The signatures use a secret key to sign the messages and a public key to verify them. To produce a forged signature in reasonable lifetime is impossible. Digital signatures are first proposed by Whitfield Diffie and Martin E. Hellman. To understand blind signatures, we first will look at how RSA works.

The RSA system is proposed by R.L. Rivest, A. Shamir and L. Adleman in 1978. [4] It makes use of the following three facts:

Exponentiation modulo a composite number $n$, for example computing $c$ from $c=m^e \bmod n$, is a relatively simple operation.
The opposite problem of taking roots modulo a large, composite number $n$, for example computing $m$ from $c=m^e \bmod n$, is believed to be infeasible. If the prime factorisation of $n$ is known, the problem of taking roots modulo $n$ becomes feasible.

To use the RSA system, a person must first choose two large prime numbers, let's say $p$ and $q$. We calculate $m=p*q$. The person must now choose an integer $e$ between $1$ and $\Phi(m)$ and $\gcd(e, \Phi(m)) =1$, where $\Phi(m)=(p-1)*(q-1)$. The pair $(e, m)$ is the public key of the person. Now the private part d can be computed by the following equation: $e*d=1 \bmod \Phi(m)$, where $d$ also lies between $1$ and $\Phi(m)$. This ends the set-up of the system. The system can be used for privacy, because you're able to send secret messages to other people. It can also be used for signatures, to ensure that you send a certain message or to verify that the message was send by the expected person. We will look at the second ability.
To sign a message, the sender computes $c=n^d \bmod m$. The receiver can easily retrieve the original message by computing $c^e \bmod m$, because $e$ and m are public. Also the receiver knows it must come from the expected sender, because only he knows $d$. The RSA set-up and digital signatures are summarized in Figure 1.1.

**RSA set-up for user A**

```
Select p, q (both large prime numbers, $p \neq q$)
Calculate $m_A=p*q$
Calculate $\Phi(m_A)=(p-1)*(q-1)$
Select integer $e_A$, $(\gcd(e_A, \Phi(m_A)) =1$, $1<e_A<\Phi(m_A))$
Calculate $d_A$, $e_A*d_A=1 \bmod \Phi(m_A)$
Public key = $(e_A, m_A)$
Private key = $(d_A, m_A)$
```

**Digital signatures**

```
A → B: $c = n^{dA} \bmod m_A$
B: $c^{eA} = n \bmod m_A$

Signature = (n, c)
```

**Figure 1.1 RSA Digital signatures**

To see how the digital signatures provide security in a payment system we consider the three stages: the withdrawal, the transfer and the deposit. In the withdrawal phase the payer sends a request to the bank. The request contains a value that represents the amount of money to be withdrawn, and it is signed with the private key of the payer.
The bank receives the request and decrypts it with the public key of the payer. It could not be forged, because of the digital signature. It debits the payer’s account and returns the electronic cash signed by the serial number. The withdrawal is illustrated in Figure 1.2.

1: payer → bank: $(€1)^{dA}$
2: bank: $((€1)^{dA})^{eA} = €1$
3: bank → payer: $(€1)^{dB}$ (dB the private key of the payee)

**Figure 1.2 Withdrawal with digital signatures**

When in the transfer phase, the payer goes to a shop and buys something, the payee asks the payer for money. The payer sends the received cash signed by the bank. With the public key of the bank, the shop can verify that the electronic cash is not forged. If it is valid cash, the payee transmits the note to the bank, which again verifies the message and checks if this cash is not already spent. The bank can send a signed deposit slip to the payee, and the payee gives the payer a signed receipt, which completes the protocol. The transfer is illustrated in Figure 1.2.

4: payer → payee: $(€1)^{dB}$
5: payee: $((€1)^{dB})^{eB}=€1$
6: payee → bank: $(€1)^{dB}$
7: bank: $((€1)^{dB})^{eB}=€1$
8: bank → payee: $(\text{deposit})^{dB}$
9: payee → payer: $(\text{receipt})^{dC}$ ($d_C$ the private key of the payee)

**Figure 1.3 Transfer and deposit with digital signatures**

The system provides security for the payer, payee and the bank, thanks to the signatures at each stage [3]. The shop can’t deny it received payment, the bank can’t deny that it issued the cash or that it accepted from the shop for deposit, and the customer can’t deny its withdrawal and is not able to spend the money twice.

The system is secure, but it is clear it provides no privacy [3]. When the bank receives messages from a shop, it can identify where and when its customers spend their money, by linking the deposit with the withdrawal. Also, the information related to the transaction can be read by everyone. These problems are solved with blind signatures.

Blind signatures can be illustrated by the concept of *carbonated envelops* [2]. Let’s assume that the bank has special signatures for different amounts of money. When someone wants to withdraw money, they bring such an envelope to the bank. The bank withdraws the requested amount of the right account, puts the money in the envelope and signs it. The person gets the notes out of the envelope, and checks the carbon-imaged signatures of the bank. Now when this person buys something at a shop, the payee can still verify the bank’s signature before accepting the money. When it is send to the bank, the bank can also verify the cash, but it doesn’t know in what envelope it was, so it can’t link the deposit to the payer.

Some vulnerabilities are already identified with this concept, which we have to keep in mind if we describe the protocol. First of all, the system fails if the bank’s special signatures could be forged. Secondly, the amount of money in the envelope could be duplicated. If the bank doesn’t notice this, it issues too much notes, and withdraws not enough from the payer’s account. Another thing we should consider to protect the interest of the payers, is that they should be able to recognize each slip they withdraw and spent. We will see this happening by choosing a random note number.

## *1.2 Protocol Description*

The electronic cash that is transferred in the transactions is from now on referred as notes. The protocol uses the RSA public-key system, described in the previous section. The banks public key will now be referred as `e` and it's private key as 0`d`.

```
payer: $n \leftarrow$ special
payer: $r \leftarrow$ random
1: payer $\rightarrow$ bank: $n*(r^e)$

bank: "withdraw from account of payer"
bank: $(n*(r^e))^d=(n^d)*r$
2: bank $\rightarrow$ payer: $(n^d)*r$

payer: $(n^d)=(n^d)*r*(r^{-1})$
payer: $(n^d)^e=n$
3: payer $\rightarrow$ payee: $(n^d)$

payee: $n=(n^d)^e$
4: payee $\rightarrow$ bank: $(n^d)$

bank: $n=(n^d)^e$
bank: n not used before?
bank: mark n used
bank: deposit to account of the payee
```

**Figure 1.5 Payer untraceability protocol**

To ensure that the payer can recognize the withdrawn notes, they create a blinded note by choosing a random number. We want to make sure that this random number is long enough, so that is not possible to break it in a reasonable amount of time. Breaking the number could be done by factorising it. If a cryptanalyst would succeed, he could determine $d$ , the bank's private key in this case. There are many algorithms known to factorise a number. The fastest known at the moment is an algorithm by Richard Schroeppel. It can factor $n$ in approximately $(\ln(n))^{\sqrt{(\ln(n)/\ln(\ln(n)))}}$. [4] According to this algorithm, if we take a number of 100 digits, it will cost $2.3*10^{15}$ operations and that corresponds with 74 year. In the protocol, the random number exists of 100 digits, but this number is expanded by using a standard public redundancy scheme into a new number $n$, now consisting of two hundred digits. This number corresponds to the random note number.

The user now creates another random number $r \bmod m$. This can be seen as choosing an envelope. This number is encrypted with the public key of the bank, and multiplied with $n$. The result is $n*r^e$ and can be seen as putting the numbered note into the slip. Notice that this is done by the payer, not by the bank. In this case, the payer will recognize the slip at the withdrawal and the transfer phase. The payer signs the result and sends it to the bank. The bank receives the envelope, which holds the note. The bank confirms the request is not forged, reduces the account of the payer by the amount of the notes in the envelope, and sends a signed copy of the request. This is equal to $(n*r^e)^d$ or after rewriting $n^d r$. The payer can verify the issued note by dividing the number by $r$ and verify the signature.

The cash is identified by $n^d$. If you pay with this at a shop, the payee can verify that it is signed by the bank, by raising it to the bank's public key. They now can test if the result is of the correct form that is specified by the redundancy scheme. When the shop forwards it to the bank, the bank can use the same verification method as the payee did, and records the deposit. Before the bank makes the deposit, he has to make it sure that the money was not spend before. This problem rises because numbers are more easily copied than piece of papers. To solute this, the bank needs to maintain a list op already used notes. The algorithm is denoted in Figure 1.5 in protocol notation. Remember that everything is modulo $m$. The signed deposit slip and the signed receipt are left out.

## *1.3 Security Considerations*

First of all, we have to notice that in the protocol no secrecy and or authentication of messages is considered. This is left out, to simplify the protocol. They could of course be provided with one of the many known encryption and authentication techniques.

There are however, some fundamental security considerations [3]. First, we want to prove that the bank is unable to trace a deposit to a withdrawal. This is the main reason that we came up with this protocol. Second, we would not want notes to be double spent. A third requirement is that no information may be revealed by linking between balance changes of the payers and the payees. Finally, note numbers are randomly chosen by the payers. This could lead to note numbers that are not unique, which means one of more payers cannot spend their issued money.

### 1.3.1 Untraceability by the Bank

We want to know if the bank is able to trace notes or in other words, can the bank learn anything about the correspondence between the number that compose $n^d r$ and the elements that compose $n^d$. The bank can select any signed note $n_1^d r$ and deposit note $n_2^d$. From the equation $n_1^d r = n_2^d$, one number $r$ can be computed. [3] But this number doesn't reveal anything, because it is chosen independently and uniformly at random and are unknown to the bank. A more theoretical proof is given by the use of abelian groups. [3]

### 1.3.2 Maintenance of Signatures

By maintaining a list of used notes, it becomes impossible to spend notes twice. Another problem is that if a payer receives several signed notes, he can produce signatures on more note numbers. This is possible because of the multiplicative properties of RSA. [5] These properties say that given two plaintexts $n_1$ and $n_2$ with their corresponding ciphertexts, $c_1$ and $c_2$, than:

Known:
$(n_1^e) = c_1 \bmod m$
$(n_2^e) = c_2 \bmod m$
Consequence:
$(n_1^e) * (n_2^e) = (n_1 * n_2)^e = c_1 * c_2 \bmod m$

Than a logical consequence is that:

$$(c_1^d) * (c_2^d) = (c_1 * c_2)^d = n_1 * n_2 \bmod m$$

This means that if a payer receives two signed notes, he can produce a forged third signed note as follows:

Known:
$(n_1 * (r_1^e)) * (n_2 * (r_2^e)) = n_1 * n_2 * (r_1 * r_2)^e$
$(n_1 * (r_1^e))^d = (n_1^d) * r_1 \bmod m$
$(n_2 * (r_2^e))^d = (n_1^d) * r_2 \bmod m$

Consequence:
$(n_1^d * r_1) * (n_2^d * r_2) = (n_1 * (r_1^e) * (n_2 * (r_2^e))^d = (n_1 * n_2)^d * (r_1 * r_2) \bmod m$

We have produced a signed note by multiplying our unsigned blinded notes (resulting in a third unsigned blinded note), and by multiplying the signed blinded notes (resulting in a third signed blinded note). With a simple special-number scheme, this threat is possible. We will see some special-number schemes that are effective to solve this problem in section 2.3.1.2, when we discuss precursors to n.

### 1.3.3 Linking Balance Changes

Depending on how the payment system is used, balance changes could reveal information about the transactions. [3] When withdrawals and matching deposits alternate, the correspondence is completely revealed. Also, if the amounts are distinct and every withdrawal is for a single deposit, then a unique correspondence can be easily found by linking the balance changes. To prevent this threat, we have to use the payment systems on a large scale. This makes it more difficult to link balance changes, because there are more transactions of the same amount. Also we have to have only a few denominations of amounts, because then we have less distinct amounts. Untraceability in our protocol is also improved if payers hold some randomly selected notes between withdrawals. This means that the exact amount that each payer carries with him, can't be determined by others.

### 1.3.4 Uniqueness of Note Numbers

Two people could by accident choose the same note number. There is no advantage by for the payer by withdrawing notes with the same note number, because they can't be spent twice. But if two different persons withdraw the same note number, there are two notes in circulation, but only one of them can be used. If the note number is paid with the first time, it is marked by the bank as used. It will be recorded in a list of spent notes. This is unfair for the second person.

The probability that this situation will occur is very small. A note number in this case is hundred digits long. It's unlikely that two persons choose the same number. Even if they do, the number is expanded by hundred more random digits. If everyone on earth requested a million note numbers a day for a million years, the chance that two note numbers collide is $0.5^{200} * 6\ 533\ 417\ 520 * 365\ 000\ 000 = 1.5^{-42}$.(according to the expected world population on July 1st 2005. [6]

# 2. The Remaining Consent Conditions

We have now seen the properties of the payer untraceable protocol, and under what circumstances they are secure and work correctly. Now we can discuss the other consent conditions. They can be seen as slight different versions of the first consent condition.

## *2.1 Payee Untraceability*

The payee untraceability protocol makes sure that the shop's account can't be traced by the bank or the payer without its consent. For this protocol, not the payer, but the payee needs to select a special note number and puts the corresponding piece of paper in a random envelope [3]. This happens at the same way as in the previous algorithm. The shop supplies it to the customer and it is forwarded to the bank. These are the first two parts in the protocol as also can be seen in Figure 2.1. The third step is identical to the second part of the payee untraceability protocol. The bank withdraws the correct amount of money from the payer's account and returns the signed blinded note. The customer checks the signature and forwards it to the shop. The payee can verify that the note number remained the same by multiplying it with the inverse of `r` and by raising it to the bank's public key. If the protocol is executed well, the shop sends the electronic cash to the bank, and the bank can record the deposit on the same way as discussed in the previous chapter. Note that if in a transaction a payee has to give back change, and the payer wants to stay untraceable, the payer can use this protocol to do so. The payee is now actually the payer (he gives back money), and the payer is now the payee.

```
payee: n ← special
payee: r ← random
1: payee → payer: $n*(r^e)$

2: payer: → bank: $n*(r^e)$

bank: "withdraw from account of payer"
bank: $(n*(r^e))^d=(n^d)*r$
3: bank → payer: $(n^d)*r$

payer: $((n^d)*r)^e=n*(r^e)$
4: payer → payee: $(n^d)*r$

payee: $(n^d)=(n^d)*r*(r^{-1})$
payee: $n=(n^d)^e$
5: payee → bank: $(n^d)$

bank: $n=(n^d)^e$
bank: n not used before?
bank: mark n used
bank: deposit to account of the payee
```

**Figure 2.1 Payee untraceability protocol**

## *2.2 Mutual Untraceability*

In this condition, both the payer and the payee must consent before a transaction can be traced. A possible application could be the flea market or advertisements on a public bulletin board. There are two approaches given to accomplish mutual untraceability. [3] The first is to introduce a third party that is known and trusted to both untraceable parties. This could be the bank. The payer uses the payer untraceability protocol with the third party and this party pays the payee using the protocol for the second consent condition, untraceability of the payee. The third party can't trace the transaction between the two parties. This is because in the first protocol, the bank and the payee couldn't trace the payer, so if the payee is the third party, this party also can't trace him. Then also holds, that if the bank plays the role of the payee, he still can't follow the transaction. In the payee untraceability protocol, the same arguments holds for the payee. That's why it's safe if the bank operates as the third party.

A second approach, referred to as *double blinding*, has the advantage it does not need an intermediary. It is an alternate version of the payee untraceability protocol. In the payee untraceability protocol, the payee knows the blinded note submitted to the bank by the payer for withdrawal. That's why the payer has to modify the protocol and add another blinding to the note before it is signed, so that the blinded note for withdrawal $((n^d)*r*x)$ is different than the payee has chosen $(n*(r^e))$. The payer also strips the layer before returning it to the payee. Now, the payee is unable to determine $x$ or $n*(r^e)*(x^e)$, and this means he's unable to allow the bank to learn about the account of the payer. The protocol is notated in Figure 2.2.

```
payee: $n \leftarrow$ special
payee: $r \leftarrow$ random
1: payee $\rightarrow$ payer: $n*(r^e)$

payer: $x \leftarrow$ random
2: payer $\rightarrow$ bank: $n*(r^e)*(x^e)$

bank: "withdraw from account of payer"
bank: $(n^d)*r*x=(n*(r^e)*(x^e))^d$
3: bank $\rightarrow$ payer: $(n^d)*r*x$

payer: $((n^d)*r*x)^e*(x^{-1})=n*(r^e)$
4: payer $\rightarrow$ payee: $(n^d)*r$

payee: $(n^d)*r*(r^{-1})=(n^d)$
payee: $(n^d)^e=n$
5: payee $\rightarrow$ bank: $n^d$

bank: $n=(n^d)^e$
bank: n not used before?
bank: mark n used
bank: deposit to account of the payee
```

**Figure 2.2 Double blinding**

## *2.3 Bilateral Untraceability*

In the final consent condition, either the payer or the payee have the possibility to allow the bank to follow a transaction. Bilateral untraceability can be used in payments between individuals, where they can seek for help if problems arise during or after the transaction. This payment system can be used for example if a payee refuses to give a receipt for a payment, if the payer (e.g. the police) wants to provide evidence on for example black markets, or if the payer returns unused notes to his own account (which may be important for paying taxes). Payee's might want to give up untraceability in case of audits, for example to show where funds came from.

This system can also be realized by extending the payer and payee untraceability protocols. [3] The untraceable party provides the other with a copy of the data that can be used by the untraceable party to forfeit untraceability. We will see how this works for the payer and payee in the next sections. In chapter seven, we will discuss more payment systems that are not unconditional untraceable.

### 2.3.1 Precursors Protecting Untraceable Party

This section describes what is needed to satisfy bilateral untraceability. Therefore we introduce *precursors* $r'$ and $n'$ that are substances of $r$ and $n$. The different precursors in the situation of the payer and the payee lead to different uses to satisfy the consent condition.

#### 2.3.1.1 Precursors to *r*

We first will look from the payer's point of view. In this case, we need a way to substantiate that a particular note number was actually withdrawn from a payer's account. The payer chooses the precursor $r$. But we have already seen in section 1.3.1 that such an $r$ can easily be computed given a signed blinded and any signed note. This is of course also valid for the payer and means that if the payer reveals his r, this is no strong evidence about the transaction, because the r could be chosen such that the payer's withdrawals seem to relates to the signed note. The solution is the use of precursors. In the withdrawal phase, instead of choosing a random number $r$, we choose another random number $r'$. The $r$ in the protocol will be computed by applying a one-way hash function on $r'$, which we will denote as $r=f(r')$. Since we are working with the RSA system, $f$ should be a bijective one-way function on residues modulo $m$. Given a signed blinded note and any signed note, a $r$ can be easily found, but a corresponding $r'$ can't be easily determined because of the one-way property of the function. This means revealing an $r'$ gives a better proof about the involvement in the transaction. The bank can then look up the corresponding payee, and a trace is found. To be able to look up the payee, a list of blinded notes withdrawn should be maintained. The bank could keep an archive of the notes. Another approach is that the payer obtains withdrawal receipts that are signed by the bank. A proposed form if the withdrawal receipt might be
$f(n^*(r^e), \text{'withdrawal by'}, \text{payer})^d$.

In case of the payee untraceability, the approach of the payer doesn't work. In the payee untraceability protocol, double blinding by the payer can't be prevented by the payee. This problem could also be solved by providing a copy of the withdrawal receipt as mentioned above. The payee can now verify that no double blinding took place and use the precursor $r'$ to sacrifice untraceability.

The one-way hash function actually also provide a fallback system. [3] If a signature forgery was detected, we could adapt the system in such a way that only payments are accepted that include $r'$ and an indication of which archived $n^*(r^e)$ satisfies $n^*(f(r')^e)$. Of course, $n$ still needs to be checked if it holds the right signature and if it's not used before. With this fallback system, users would have to sacrifice untraceability in transactions.

#### 2.3.1.2 Precursors to *n*

In section 1.2 we mentioned a public redundancy scheme that expanded the number $n$ by hundred digits. This was done by concatenating the binary representation of the chosen number to itself and then apply an agreed mixing transformation. [3] Several other schemes are possible. The first half of the digits could be chosen randomly and the second half then would be a one-way function of the first half. Or the first half could be constructed by the bitwise exclusive-or of the original first half and the mapped second half. With these variations, the first half is referred as the *random component* of $n$. $n'$ is the precursor to this random component, or can be called the precursor to $n$.

We will see two examples of the usefulness of supplying precursors to $n$.

First we discuss a situation where a payer gives money to an employee of the payee, for example a waiter or a sales assistant. As a payer you want to make sure that your payment will  get to the right account. As a payer, you would like to have a receipt to prove that you made the payment. If the intermediary refuses to give you one, the payer can deny the payment. To provide this situation, the payer encodes a property of the payee in the note number, such as the name of the account number. This is only possible if the payee is already known to the payer. With an unknown payee, the note can be formed, such that it contains a public key. If the payer has decided on a payee, a signature is made on the property of that payee. This signature is hand over to the payee, together with the signed note. The appellation (the property) can be verified with the public key contained in the note. Also the bank should check the appellation with the same public key, before making the deposit. The signed property shows the payee that the payer has approved in the transaction. If an intermediary tries to deposit the notes to his own account, the bank will prevent this with the final check. The property can be seen as a precursor to `n`.

Another example that illustrates the use of `n'` is referred to as *bankproof payer untraceability*. [3] This protocol described in Figure 2.3, the bank can not falsely claim to already deposit the note to another account.

Here a whole new process is added before the deposit is made. If a shop makes a request for a deposit, the bank will first mark `n` pending. The provide the payee a statement of acceptability (the has function `f(n, e, d)`). The date stamp is to protect the bank from having lots of records of aborted requests for an indefinitely long time. The payee and payer can both verify this statement, and the payer sends his precursor to the payee, who forwards it to the bank. If the hashed value of the precursor is equal to the note number to deposit, then the deposit is made. The bank preserves `n'` to fairly reject to provide a statement of acceptability. The justification is done by showing `n'`, and thereby verifying the note was already deposited, because otherwise the bank couldn't know `n'`. If the bank can't provide this precursor, a judge must decide in favour of the payee.

```
payer: r, n′ ← random
payer: n = f(n′)
1: payer → bank: $n*(r^{e})$

bank: “withdraw from account of payer”
bank: $(n*(r^{e}))^{d}=(n^{d})*r$
2: bank → payer: $(n^{d})*r$

payer: $(n^{d})=(n^{d})*r*(r^{-1})$
payer: $(n^{d})^{e}=n$
3: payer → payee: $(n^{d})$

payee: $n=(n^{d})^{e}$
4: payee → bank: $(n^{d})$

bank: $n=(n^{d})^{e}$
bank: n not pending and not used before?
bank: mark n pending from date e until date d
bank → payee: $f(n, e, d)^{d}$

payee: $(f(n, e, d)^{d})^{e}=f(n, e, d)$
payee → payer: $f(n, e, d)^{d}$

payer: $(f(n, e, d)^{d})^{e}=f(n, e, d)$
payer → payee: n′

payee: $f(n')=(n^{d})^{e}$
payee → bank: n′

bank: f(n′)=n
bank: n pending from?
bank: mark n used
bank: remember n′
bank: deposit to account of the payee
```

**Figure 2.3 Bankproof payer untraceability**

# 3. Time Value of Money

In current payment systems, customers earn some interest over the money on their account. They could be characterized as being a *debit* or a *credit* payment system. In a debit system it is required that the balance of the payer's account stays above zero. In a credit system you can have a negative balance, you only need to stay above a negative limit. In both systems, interest can be earned. In the credit system, you can be charged interest for the negative amount on the account.

You could see the protocol as a withdrawal at a ATM cash machine. Once you made a withdrawal, the money is no longer on your account and you don't gain interest over that amount. Another possible implementation is that you earn interest over your money until you spend it. With the first implementation, the untraceable party protocols cause no problems. The accounts and balance are known to the bank, as well as the balance changes. The bank can easily calculate the interest. With the second implementation, it is more difficult, because with the proposed protocols we don't know for how long a payer carries the money. When a note is deposited, we don't know how much and who we should give interest. There are two approaches to cover this problem. *Dynamic note values* changes the value of notes over time. With *fixed note values*, the bank can determine when changes in accounts occur, and then can calculate the interest.

## *3.1 Dynamic Note Value*

The first approach is called dynamic note values. We will first see how it works for interest earnings. Signed notes will carry a new value, the year of issue. [3] If a payer now withdraws for example a euro, he will receive the current value of one euro, which is one euro and some extra little bit of money. Suppose the current value of the euro is € 1.06, then when a payment of one euro occurs, the payer actually pays € 1.06, and the payee will give back € 0.06 change, the exact amount of the interest earned on that euro.

However, in this approach it seems that a payer never earns the correct amount of interest. He will receive a bit more. In the given example, he should receive € 0.06. But the notes that represent one eurocent also have increased in value. If the payee pays back 6 eurocents, than he actually gives too much change back, and the payer gets more interest than expected. This problem does not occur with fixed note values, which is discussed in the next section.

In case of interest charges a different approach is used. Together with the year of issue, a expiration date is included in the note. [3]. Now if the payer withdraws money and the balance is below zero, he should be charged interest. If the notes have not been return to the bank before the expiration date, the bank considers them spent, and the payer will have to pay interest over the current value of that money. Thus, at the moment of determining how many interest should be charged, the payer can have a negative balance at his account and some unmatched withdrawals. The interest of the negative balance can be computed easily. The interest of the unmatched withdrawals is the current value of the time the money was spent minus the (higher) price that must be repaid to the bank on expiration date. This means that if the money was spent after the expiration date, you still owned the notes, and you don't have to pay interest for that notes. This seems fair and doesn't provide a vulnerability.

A problem that does occur, is that credit notes can be held as long as they can by payees to increase the current value of the received cash. This can be discouraged if payers ask for a receipt as described in section 2.3.1.2. Instead of designating a property of the payee, the payer could include the date of payment, so that the bank can see if the payee is cheating. (i.e. if the request for deposit is a long time after the actual payment). The approach with the designation of the payee is ineffective if payers and payee conspire. But if payees wait to long with the deposit stage, they take the risk that the payer will pay with his notes at another place, resulting in that the cheating payee can't deposit his money anymore.

## 3.2 Fixed Note Value

This approach deals with determining when balance changes occur. We will again first look at the debit case. In the transfer stage, the payer supplies an unsigned blinded *receipt note*. [3] This should have the form of $n*(r^e)$. If the deposit is accepted, the bank signs the receipt with signatures that encode the date and corresponding denominations of the payment. This is of course not the same signature as with the withdrawal, so that we're still able to recognize the difference between a withdrawal and a payment. The receipt is returned via the payee to the payer, and he unblinds it.

Because of the blinding, the payer can safely hand over the receipt note to the bank. The bank is unable to learn about the shop where the transaction took place. The bank also doesn't know when the payer had withdrawn the money. The bank should match the receipt with an unmatched withdrawal of the same amount. The only thing he does know is the date of payment (which is encoded in the signature). Thus, the bank can give interest to the payer. A clear drawback is that the timestamp reveals information about when the transactions took place. To prevent that the bank learns too much, the system should be used at a large scale, to make it more difficult to link balance changes. Also if interest is given on a monthly base, the timestamp would only have hold the month of payment. The less detailed the timestamp is, the more difficult it becomes again to link balance changes.

The credit case works a bit similar. Here we work with an issue date and a settlement date. The withdrawal takes place on the issue date. On the settlement date when interest has to be calculated, the payer must hand over all the receipts and unspent notes. The receipts provide information about the date of payment, thus interest can be computed by the time difference between the settlement date and the payment date. If a payer returns receipts and unspent notes that is below the issued withdrawal, the bank will calculate maximum interest over the missing part, considering it was spent on the issue date.

We suggested in the debit case that the bank should choose an unmatched withdrawal with a receipt of the same amount. But does it matter which withdrawal the bank chooses? If the bank chooses the least recent unmatched withdrawal first, the interest of the notes will be more constant than if the bank picks the most recent first (when some notes will produce less interest than others because the oldest payment is matched to the latest withdrawal, and vice versa). But these two approaches will always yield the same interest. We will give a proof.

Let's assume there are $n$ withdrawals of the same amount. We don't consider different amounts, because we always choose an unmatched withdrawal of the same amount. To calculate the interest we have to subtract withdrawal dates from the payment dates. Thus we say that $w_i$ denotes the day of the $i^{th}$ withdrawal ($1 \leq i \leq n$). To determine all interest, all n withdrawals should be linked to a payment. Thus there are n payments. The date of payment $i$ is denoted as $p_i$ ($1 \leq i \leq n$). Interest is defined as: [7]

- *Interest = Amount * Interest rate * Time*

Suppose our amount is one euro and $r$ denotes the interest rate.

Interest in first case (most recent unmatched withdrawals) is than:

$$\sum_i (r*(p_i - w_{(n-i)}))$$

En in the second case:

$$\sum_i (r*(p_i - w_i))$$

But this is exactly the same!:

$$\sum_i (r*(p_i - w_{(n-i)})) = r*(\sum_i (p_i - w_{(n-i)})) = r*(\sum_i p_i - \sum_i w_{(n-i)})$$
$$= r*(\sum_i p_i - \sum_i w_i) = r*(\sum_i (p_i - w_i)) = \sum_i (r*(p_i - w_i))$$

This means it doesn't matter which unmatched withdrawal the bank chooses to link to a payment (as long as the amount is the same as at the receipt), the interest doesn't change.

# 4. Efficiency and Performance

It is now clear how and why the protocol works, and which expansions can be introduced. This section deals with the efficiency and the performance of the system. We will measure the efficiency of different naming schemes, look at possibilities to prevent abuse of the payment system, and discuss the time and space requirements.

## *4.1 Denomination Schemes*

This section discusses what denominations we should use. If each note was signed by the exact amount of payment, it becomes very easy to link balance changes. We've already seen that using less denominations of notes makes it more difficult to link balance changes in section 1.3.3. But paying everything with a number of eurocents is also not desirable. We wish to use a reduced number of notes. There is a middle course between one sort of amount and all possible amounts that brings the most effective denomination scheme.

Let's illustrate this by taking cent notes based on the powers of two. [3] Then we get cent notes of amount 1, 2, 4, 8, 16, …. A denomination is then one bit in a binary representation of a payment. Only if the bit is one, the note is sent. € 6,35 would then be represented by 1101110010 (least significant bit at the left), and with $j$ different denominations we can make payments up to $2^j - 1$ cents, so with 10 bits we can pay up to € 10.23 and with 20 bits we can buy something of € 104 857.60. Uniformly distributed amounts and a maximum amount of $2^j - 2$ means that the average number of notes in a payment is j/2. [3] But cheap purchases (e.g. food, birthday cards) occur more often than expensive ones (e.g. flat screen television, tuition fees), thus the average number will probably be below j/2.

## *4.2 Abuse of Payment Systems*

Payment systems that guarantee complete anonymity, will attract criminal activities, such as bribes, black market sales and launder money. So it's obvious that sometimes we don't want to provide unconditional untraceability. A possible threat is that a payer pretends to make a withdrawal like in the payer untraceability protocol, but in reality it could be a withdrawal like in the payee untraceability protocol. [3] The bank can't tell the difference. This could be prevented by requiring payer account holders to submit blinded notes in advance. The notes will be kept safe by the bank, and return signed copies during the withdrawal phase. In the next chapter we will discuss a system that is designed to prevent such fraud and other criminal activities.

## *4.4 Online and Offline Systems*

Payment systems can be online or offline. In online systems, banks are in continuous contact with the payers and payees. Balancing of the accounts can happen after a transaction (thus, after the confirmation of the deposit from the bank). [3] This is typical in online systems, and doesn't happen in offline systems. Offline systems are adequate for payments that can be performed later, such as loans, prepaid orders or donations.

A drawback of online systems is that there should be a continuous connection between banks, payers and payees. This might be expensive, especially for countries that not yet have a good technical infrastructure. It's inefficient in communication costs, computing costs and storage. To reduce the costs, the system might be online during busy hours, otherwise offline (i.e. at night).

## *4.5 Memory Requirements*

Several data storages are need. The untraceable party must store signed note numbers before provided to the other party or the bank. Remember that the proposed RSA system required $n$ to be 200 (decimal) digits long. This can be encoded in estimated 111k bits ($2^{11025} \approx 8.1^{199}$). This can easily be stored on a computer. A card might not have enough capacity. Most smart cards carry 64k bits of EEPROM memory [8]. This is sufficient for at least eight payments of € 655.35. But nowadays, cards are being developed that can store much more data.

The bank also needs storage space to keep track of the account balances. A traceable payment system would only require one access to the payer's account and one to the payee's account for each payment. With the proposed protocols, the balancing of the payer's account is done at the withdrawal stage and can be cover several payments. We can probably be more efficient in the storage of account balances. Also with payee anonymity, several payments can be combined in one single deposit.

Sometimes clearing is required. Notes that are expired, can be removed from the bank's record of clear notes as they expire. This expiration date may not be too far in the future, otherwise the bank is not sure how many still-valid uncleared notes are outstanding, because destroyed or lost notes will be on the list forever. In bankproof payer untraceability, the precursors also need to be stored. The storage should be no problem: Chaum talks about 7.5 gigabyte, that is a more than feasible capacity these days. This of course varies if the system is used on a larger (or smaller) scale.

## *4.6 Computation Requirements*

The computers of today could easily compute the transactions between the payee and payer. If small public exponents are used by the bank to sign the notes, then some simple modular calculations are needed, and no special devices are needed. But to ensure that the payment system is safe, large public keys are needed, that results in large exponential calculation in modulus. However, these calculations should also be no problem these days.

# 5. Fair Tracing Methods

We have seen several protocols that provide untraceability for a payer or payee. We also have noticed that sometimes these systems is not desirable, because it is easy to abuse the system for criminal activities. Therefore, some other systems are designed to prevent these situations. These systems are referred as *fair tracing systems*. Here, transactions may only be traced legally (with the permission of the payer or the judge). Tracing without the permission of the judge or the customer, or illegal tracing, is prohibited.

There are three aspects that are important to fair tracing payment systems: coin or note tracing, owner tracing and self-deanonymization. [1] Note tracing refers to some special piece of information that is stored in the note. This can be used to detect blackmailing or bank robbery. A blackmailed person who is forced to make a withdrawal, can inform the bank to add note tracing. The notes can bow be recognized if the criminal is trying to pay with them. Also in bank robbery, before delivered to the criminal, the bank will add note tracing.

Owner tracing refers to the revealing of the identity of the withdrawer during a deposit. The trace is made at the deposit stage. This aspect detects money laundry and black market selling.

We have already seen some forms of self-deanonymization in the bilateral untraceability protocol. This also could be used to detect black market selling and bribes. One special form of bribes is a kidnapping. In this form, the harmed person is not able to contact a third party to apply fair tracing. In this case, self-deanonymization is an important aspect.

We will discuss one fair tracing systems. The three discussed aspects will be analysed for this approach. This protocol makes use of a trusted third party.

## *5.1 Fair Tracing by Trusted Third Party*

This method provides fair tracing by the use of some publicly trusted third parties. The idea is that the payer's spending history can be revealed, including the amount and the payees of the transactions. This system is developed by Brickell, Gemmell and Kravitz. [9] It makes use of the algebraic properties of a large subgroup of prime order $q$, embedded in the multiplicative group $GF(p)^*$, where $p$ is also a large prime. We will briefly describe the properties of groups, so that the protocol becomes more comprehensible. Furthermore, we now will have different types of signatures, denoting different denominations. This means the bank has not on public exponent $e$, but for example $e_{cent}$, $e_{dime}$ and $e_{euro}$.

### 5.1.1 Groups and subgroups

A *group* is an abstract structure in which one can multiply and divide, and it always contains a neutral element $u$, such that for all $x$, $u*x = x*u = x$. [10] In fact, the modulus where we worked with in the RSA system, to describe Chaum's protocols, can be seen as a group. Associativity, the existence of a neutral element and an inverse, form the three axioms of group theory. The neutral element and inverse are unique.

We say that a group has *order* $n$, if it has $n$ elements and has the form $\{1, g, g^2, \ldots, g^{n-1}\}$, with $g^n = 1$ and for all $i$ between zero and $n$, $g^i \neq 1$. $g$ is called a generator of this group. We can multiply the elements in the exponent, just as we have seen in the RSA system. Thus, $g^i * g^j = g^{i+j \bmod n}$. The *inverse* of an element $g^i$ can be computed by $g^{-i \bmod n} = g^{n-i}$.

Some properties of groups are:

- if $x$ is an element of group $G$ with order $n$, and $x = g^i$, then $x^n = g^{i*n} = 1$. If we find a $d$, $0<d<n$ , such that $x^d = 1$, than $d$ is called the order of $x$. This can only be true if $d$ divides $n$.
- If there exists a $d$ that divides $n$, than there exists a subgroup $H$ of order $d$.

A subgroup of the group $G$ can be defined as a smaller group with the operation induced by that of $G$. [10] A subgroup can not be empty and has the following property:

- For all elements $x$, $y$ that are contained in the subgroup, $x*y^{-1}$ must also be in the subgroup.

For example, $\{0,2,4\}$ is a subgroup of $GF(6)$, because it's smaller than $GF(6)$ .

### 5.1.2 Proving Combined Knowledge of a Representation

The set-up, withdrawal and transfer protocols are extensions of some protocols introduced by S. Brands, to prove knowledge of a representation. [9] So, we'll need to describe these protocols first.

In proving knowledge of a certain representation, a person might know some value $y$, that is constructed by $h=g_1^{e1}*g_2^{e2}$. To prove he knows how $y$ is constructed, the person computes two random variables $r_1$ and $r_2$ that are elements of the group that is used (denoted as $Z_q$), and sends $h$, $(g_1, g_2)$ and $g_1^{r1}*g_2^{r2}$ to the verifier. The verifier provides a challenge by sending back a number $c$, also an element of $Z_q$, to this person. Now the person can really show his knowledge about $h$, by calculating $x_i = r_i+c*e_i \bmod q$ for $i=1$ and $i=2$ and send both $x$'s to the verifier. This party can check the knowledge by $g_1^{r1}*g_2^{r2}*h^c = g_1^{x1}*g_2^{x2}$. This must be the same because $g_1^{r1}*g_2^{r2}*h^c = g_1^{r1}*g_2^{r2}*(g_1^{e1}*g_2^{e2})^c = g_1^{r1}*g_2^{r2}*g_1^{c*e1}*g_2^{c*e2} = g_1^{(r1+c*e1)}*g_2^{(r2+c*e2)} = g_1^{x1}*g_2^{x2}$. The protocol is summarized in Figure 5.1.

```
P: knows $h=g_1^{e1}*g_2^{e2}$
P: computes $r_1, r_2 \in Z_q$, such that $y=g_1^{r1}*g_2^{r2}$
P → V: $h, y, (g_1, g_2)$

V: chooses $c \in Z_q$
V → P: c

P: computes $x_i = r_i+c*e_i \bmod q$ for i=1 and i=2
P → V: $x_1, x_2$

V: checks $y*h^c = g_1^{x1}*g_2^{x2}$
```

**Figure 5.1 Proving knowledge of a representation**

Now we can give a prove of combined knowledge of a representation. This is going to be used by the trusted parties in the following protocol, to prove that the trusted parties know the representation of the value of the note. Thus, two persons (or parties) claim to have combined knowledge of a representation of $h$, of $g_1$ and $g_2$. Let's assume that the first person knows $g_1^{e1,1}*g_2^{e1,2}$ and the second one knows $g_1^{e2,1}*g_2^{e2,2}$. They both can perform the proving knowledge of a representation used before. If they both succeed, we say that the two persons have combined knowledge of the representation $h$, if the verifier can see that $h = (g_1^{e1,1}*g_2^{e1,2})*(g_1^{e2,1}*g_2^{e2,2})$.

## 5.1.3 Protocol Description

We shall now continue with the protocol description. This method works with $G$ with order $q$, as a subgroup of $GF(p)^*$, where $p$ and $q$ are large primes. The bank chooses randomly and publishes some generators of $G$, called $g$, $g_1$, $g_2$, $g_3$, $g_4$ and $d$. In the set-up, the user gives information to the trusted parties, which makes them able to trace any payment back to its withdrawal, including every note used. To open an account, the payer first generates two random numbers $r_1$ and $r_2$ that are in $G$ and computes $g_1^{r1}*g_2^{r2} \bmod p$. This is send to the bank to serve as an identity. $r$ is kept secret, so remains unknown to the bank, until the payer will double-spend his note. The bank now computes for each note to be withdrawn a random $d_i$, that is also kept secret. The bank sends $g$ and $h_i=g^{di}$ back. The user can randomly choose for every note $i$, a $y_{3,i}$ and $y_{4,i}$, such that $y_{3,i}=s^i_{1,1}+s^i_{2,1}$ and $y_{4,i}=s^i_{1,2}+s^i_{2,2}$. We're splitting the values. For each note, we send back the value of the note $n'_i=g_3^{y3,i}*g_4^{y4,i}$. The $s$ values are send to the trusted parties. They can now prove combined knowledge over $n'_i$.

```
payer: $r_1$, $r_2$ ∈ G ← random
payer → bank: $g_1^{r1}*g_2^{r2}$ mod p

for every note i:
  bank: $d_i$ ← random
  bank: $h_i=g^{di}$
  bank → payer: g, $h_i$

  payer: $y_{4,i}$, $y_{4,i}$ ∈ G ← random
  payer: split $y_{3,i}$, $y_{4,I}$: $y_{3,i}=s^i_{1,1}+s^i_{2,1}$ ,$y_{4,i}=s^i_{2,1}+s^i_{2,2}$
  payer: $n'_i$ ← $g_3^{y3,i}*g_4^{y4,i}$

  payer → trusted party 1: $s^i_{1,1}$, $s^i_{2,1}$
  payer → trusted party 2: $s^i_{2,1}$, $s^i_{2,2}$
  payer → bank: $n'_i$

trustees can prove combined knowledge over representation of $n_i$
```

**Figure 5.2 Set-up with the trusted parties**

At this moment, the payer is able to make withdrawals. The bank provides the payer with a blind signature of the form $(A, B, z', a', b', r')$, where $(A, B)$ denotes the note. It is believed that is it hard to create such an tuple, because it is believed that calculate the discrete log of $h_i$, as said in section 1.1. Furthermore, a hash function is used to calculate a part of $r'$. The somewhat complicated withdrawal protocol is denoted in Figure 5.3.

After withdrawals follows the payment protocol. The payer provides (A, B) and $(z', a', b', r')$ to the payee. He also has to reveal $y_{3,i}*s$. If the trusted party reveals $y_{3,i}$ of the withdrawal protocol to the judge (who has already the values of $m'$, $y_{3,I}*s$ of the transfer protocol). With these two values he can link the payment to the withdrawal. Also the transfer protocol is described in Figure 5.3. The deposit is done by sending a transcript of the transfer protocol to the bank and the judge.

## 5.1.4 Fair Tracing Analysis

With respect to the three aspects of fair tracing methods, let's see if this is a good fair tracing system.

### 5.1.4.1 Note Tracing

We have seen that note tracing is possible, because in the set-up, the trusted parties have combined knowledge of the representations of the notes. If a judge wants to know if a customer spends a note, he can ask the two trusted parties for all values of $s_{j,i}$, that are provided to them by that user. We can now compute all $y_{3,i}$ from the withdrawal. If the note is recorded in the list of used notes, then this note was spent, and we proved it was spent by the suspected user. The note can be found by computing $n'^{(l3)\verb|^|-1} = n^{(y3,i)\verb|^|-1}$.

### 5.1.4.2 Owner Tracing

This protocol also provides owner tracing. If the judge provides the trusted parties with $n'$, $l_3$ and the identity of the user $g_1^{r1}*g_2^{r2}$ (and of course sign the message, so the trusted parties can verify that they're talking to the judge), the trustees can try to prove if they have combined knowledge of $(g_1^{r1}*g_2^{r2})*d*n'_i$ $(=n)$, that's relative to $n^{(y3,i)\verb|^|-1}$.
They do this of course using their knowledge of $s^i{}_{j,1}$ and $s^i{}_{j,2}$ (for trusted party $j$).
If they succeed, we can assume that this user was involved in the payment where the note related to $n'$ was spent (by this user).

Another nice property of this protocol is that users that try to double spend their money, easily can be detected. If the bank has two records of the same note spent twice, then two different set of $l$'s are made during the transfer phase: $(l_1, l_2, l_3, l_4, l_5)$ and $(l_1', l_2', l_3', l_4', l_5')$. We're only interested in $l_i$ and $l_i'$ for $1 \le i \le 3$, because with these values we can reveal the identity of the user, $(g_1^{r1}*g_2^{r2})$. [9]

### 5.1.4.3 Self-deanonymization

To give up our anonymity in a transaction, we could use the same protocol and pretend that the payer is taking the roles of the two trusted parties. But with one person, no combined knowledge over a representation can be proved, thus the system is too weak. If the payer wants to give up his untraceability, he has to broadcast $r_1$ and $r_2$ to reveal his identity, and also $n'$ (that is only known to the bank and the user), to show that he used the note in a payment.

**Withdrawal**

```
payer → bank: prove knowledge of $g_1^{r1}*g_2^{r2}$ mod p

bank: $n \leftarrow g_1^{r1}*g_2^{r2}*d*n'_i$
bank: choose $w \in G$
bank: $z \leftarrow n^{di}$, $a \leftarrow g^w$, $b \leftarrow n^w$
bank → payer: z, a, b

payer: $s, x_1, x_2, x_3, x_4, x_5 \in G \leftarrow$ random
payer: $n' \leftarrow n^s$, $z' \leftarrow z^s$
payer: $j_1 \leftarrow u_1*s-x_1$ mod q, $j_2 \leftarrow u_2*s-x_2$ mod q
       $j_4 \leftarrow y_{4,i}*s-x_3$ mod q, $j_5 \leftarrow s-x_5$ mod q
       $A \leftarrow (g_1^{x1})*(g_2^{x2})*(g_3^{y3,k*s})*(g_4^{x4})*(d^{x5})$
       $B \leftarrow (g_1^{j1})*(g_2^{j2})*(g_4^{j4})*(d^{j5})$
payer: choose $u, v \in G$
payer: $a' \leftarrow (a^u)*(g^v)$, $b' := (b^{s*u})*(m'^v)$,
       $c' \leftarrow f(m',z',a',b',A)$
payer: $c \leftarrow c'/u$
payer → bank: c

bank: $r' \leftarrow d_i*c + w$ mod q
bank → payer: r'

payer: $g^{r'}=(h_i^c)*a$ mod p, $m^{r'}=(z^c)*b$ mod p
payer: $R \leftarrow r'*u + v$ mod p
payer: $sign_B(A,B) \leftarrow (z',a',b',R)$
```

**Transfer**

```
payer: $l_3=y_{3,i}*s$
payer → payee: A, B, $sign_B(A,B)$, $l_3$

payee: verify $sign_B(A,B)$
payee: $A*B\neq 1$
payee: $c_1 \leftarrow f(payee, time, A, B, y_{3,i}*s)$
payee → payer: $c_1$

payer: $l_1 \leftarrow x_1+c_1*j_1$ mod q, $l_2 \leftarrow x_2+c_1*j_2$ mod q,
       $l_4 \leftarrow x_4+c_1*j_4$ mod q, $l_5 \leftarrow x_5+c_1*j_5$ mod q
payer → payee: $l_1$, $l_2$, $l_4$, $l_5$

payee: $(g_1^{l1})*(g_2^{l2})*(g_3^{l3})*(g_4^{l4})*(d^{l5})=A*B^{c1}$ mod p
payee → bank, judge: transcript of transfer protocol
```

**Figure 5.3 Withdrawal, transfer and deposit with trusted parties**

# 6. Conclusions

We have seen many protocols that deal with the traceability of payment systems. We have provided the underlying cryptographic that must be used to create the anonymity. First we presented unconditional untraceable systems presented by David Chaum. [3] The protocols provide the basics of anonymity in electronic payment systems. The protocols know lots of drawbacks that can be solved by extending the protocols. With these extensions, the algorithms become more unclear. Another problem is that if a person is fully untraceable, criminals can perform unethical actions like money laundering or bribes. That's why we presented a fair trace system in chapter 5. More fair trace systems can be found in the paper of X. Hou, C.H. Tan. [1] These protocols are significantly more complex, but they fit better in our society, because they prevent and detect crimes, that could not be prevented or detected in untraceable payment systems.

I haven't looked too deeply into the requirements of hardware in order to run these systems. My attention went out to the mathematical behind the systems. How to implement it, and to integrate it with the hardware, is something that can be done in the future.

## References

[1] X. Hou, C.H. Tan - *Information Communication Center, Singapore 639798*
'On Fair Traceable Electronic Cash' 2005
http://csdl.computer.org/dl/proceedings/cnsr/2005/2333/00/23330039.pdf

[2] David Chaum - *Scientific American*
'Achieving Electronic Privacy' August 1992
http://ntrg.cs.tcd.ie/mepeirce/Project/Chaum/sciam.html

[3] David Chaum - *Centre for Mathematics and Computer Science, Kruislaan 413, 1098 SJ Amsterdam, the Netherlands*
'Privacy Protected Payments Unconditional Payer and/or Payee Untraceability' 1989

[4] R.L. Rivest, A. Shamir, L. Adleman
'A Method for Obtaining Digital Signatures and Public-Key Cryptosystems' February 1978
http://theory.lcs.mit.edu/~rivest/rsapaper.pdf

[5] A. Menezes, P. van Oorschot, S. Vanstone
'RSA public-key encryption' (Section 8.2 from the Handbook of Applied Cryptography) 1996
http://www.cacr.math.uwaterloo.ca/hac/about/chap8.pdf

[6] http://www.ibiblio.org/lunarbin/worldpop

[7] Needles, Powers, Crosson
'The Time Value of Money' (Appendix C from the book 'Principles of Accounting') 2005

[8] Erik Poll – *SoS Group, RU Nijmegen*
'Smart cards' 2004

[9] Ernie Brickell, Peter Gemmell, David Kravitz – *Symposium on Discrete Algorithms, Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms, San Francisco, California, United States.*
'Trustee-based tracing extensions to anonymous cash and the making of anonymous change' 1995
http://portal.acm.org/citation.cfm?id=313790

[10] Dick van Leijenhorst – *Afdeling Grondslagen, RU Nijmegen*
'Cyclic groups and subgroups, the possible orders of the elements' & 'Some elementary facts about groups' from the course Cryptography 2005