Basic Security of the ecash™ Payment System*

Berry Schoenmakers

DigiCash
Kruislaan 419
NL-1098 VA Amsterdam
The Netherlands
berry@digicash.com

Abstract. ecash™ is a payment system designed and implemented for making purchases over open networks such as the Internet. In this paper we review some of the main cryptographic techniques used throughout the ecash system. We will focus on security aspects as well as some performance related issues. The central notion of an electronic coin is treated in detail, and the basic protocols manipulating coins are described.

1 Introduction

Behind the scenes banks, credit-card companies, and other financial institutions have been processing transactions electronically for several decades now. Two important developments that will open up the field of electronic payment systems are now taking place. First, the prospect of electronic commerce over the Internet is creating a large demand for electronic payment methods for open networks. Second, the introduction of nation-wide electronic purse schemes is creating many more places and situations where smart cards can be used for cost-effective off-line payments.

In this paper we will describe several aspects of the ecash system, mostly security related, and discuss its place among other payment technologies. Ecash finds its roots in the work by Chaum (see, e.g., [Cha83,Cha90]), who invented the notion of electronic (or digital) coins as well as the basic protocols for electronic cash. Electronic coins possess similar properties as metal coins, among which is the unique feature that a payment transaction leaves no trace about the identity of the payer. Currently, ecash technology (as provided by DigiCash, see `http://www.digicash.com` for more details) is used by a number of banks around the globe. These banks issue ecash to their customers, who can then spend it at affiliated merchants on the Internet.

We will focus on the core protocols that make up the ecash system. For brevity, we will omit a lot of details and alternative approaches that are taken into account in the actual system. We do, however, consider some future extensions such as the extension to off-line ecash, where the use of smart cards and the Internet are combined into a highly versatile and secure privacy-protecting payment system.

* Version of April 1997. Appears in B. Preneel and V. Rijmen (eds.) *State of the Art in Applied Cryptography, Course on Computer Security and Industrial Cryptography, Leuven, Belgium, June 3–6, 1997*, vol. 1528 of Lecture Notes in Computer Science, pp. 338–352. Springer-Verlag.

2 Characteristics of Electronic Payment Systems

Although more and more consensus is building up as to which properties are required of a payment system, we are not going to list and describe these properties one by one in this paper. Instead, we take a bottom up approach, and describe some of the basic characteristics of payment systems. From these characteristics one can then infer the possibilities and impossibilities for the numerous variations, and what their impact is on the performance and flexibility of the system.

Payment by instruction vs prepaid electronic cash In so-called *payment by instruction* type of systems, a payer basically orders the bank to move a sum of money from her account into a payee's account. Examples in this category are credit and debit cards as well as many forms of cheques. The moment at which the money is actually moved from the payer's account into the payee's account depends on the system, but at all times banks and credit card companies will try to prevent discrepancies between accounts.

The central security aspect in these systems is to ensure that only legitimate account holders are able to issue payment instructions. Of course, digital signatures are the solution for doing this over a large, open network such as the Internet. Since digital signatures only make sense if there is an infrastructure for certifying public keys, a lot of effort is devoted to just this. See, for instance, the SET (Secure Electroncic Transaction) proposal, a joint effort by MasterCard, VISA, and other influential partners, which specifies a hierarchy of certification authorities on top of the payment protocols, as laid out in the iKP system [BGH+95].

Prepaid systems are conceptually close to electronic equivalents of cash. Telephone cards, smart card based systems, as well as ecash fall into this category. The user's account is debited as soon as the card or device is reloaded with electronic cash. During payments the electronic cash is released again, and only then the payee's account will be credited. In the mean time the issuer keeps a float corresponding to the outstanding cash.

The central security aspect in this type of system is to ensure that cards or representations of cash cannot be forged. When forgery happens, the float will ultimately be insufficient to credit all of the payees' accounts for received payments. Of course, it should also be ensured that only legitimate account holders can reload cash from their accounts. However, this security aspect is now limited to the infrequent withdrawal protocol, and is no part anymore of the more frequent payment protocol.

On-line vs off-line In the field of electronic payment systems, the notions on-line and off-line refer to a specific property of the payment protocol. Although the payment protocol is functionally a protocol between two parties (payer and payee) many payment systems require that the payee contacts a third party (e.g., the bank or the credit-card company acting as an acquirer) before accepting a payment. If that is the case, the system is called an on-line payment system; the

communication between a payee and its acquirer may be using any communication medium (not necessarily the Internet). If such a contact with a third party is not required during the payment protocol, the system is called off-line. In an off-line system payees are required to contact their acquirer on a regular basis for clearing all received payments.

Secret key vs public key authentication A basic requirement of a payment protocol is that it allows a payee to receive payments from any payer. A payment can be seen as some sort of authentication of the payer towards the payee (to show that the payment is authentic). Authentication can be based on secret key cryptography or on public key cryptography. In the latter case, the payee only needs to have a public key available in order to verify incoming payments. Although the costs of equipping smart cards with crypto co-processors are expected to become marginal, it is important to note that the property of public verifiability can be obtained using simple smart cards only, provided one applies a method of what we call *signature transport*. In such a system, signatures are created by the issuer only, and later *endorsed* by the payer during the payment protocol, depending on a challenge from the payee. The trick is to achieve that sufficiently many payments can be made between successive reloads, which requires optimal use of the limited amount of EEPROM available on simple smart cards. The added advantage is that the secret key for creating signatures is only used by the issuer.

In case authentication is based on secret key (symmetric) cryptography, however, the payer and payee must have a shared secret key available in order to complete a payment. A straightforward solution is to give all users the same secret key, but this is generally considered insecure, as this would mean that breaking a single smart card (i.e., extracting its secret key) will suffice to break the complete system. The standard solution is therefore to break the symmetry between payers and payees by equipping the merchants with a highly secure tamper-proof box called a SAM that contains a master key. The payers' keys are derived from this master key in a process called diversification by applying a cryptographic hash (e.g., SHA-1) to the concatenation of the master key and the payer's card number. The idea is that the SAM is more difficult to break than a smart card, and also that it is possible to routinely check (as part of the maintenance) if the SAMs have not been tampered with.

In the EMV standard (developed by Europay, MasterCard, and Visa) a first step is made toward including public key authentication. To prevent frauds in which cards with fake card numbers are introduced, each card carries a fixed RSA certificate that shows the validity of the card number. At the start of each payment, the certificate can be verified against the public key stored in the POS terminal. The remainder of the payment protocol again relies on a secret master key stored in the SAM of the POS terminal.

Counters vs coins A direct way of representing electronic cash is to use a counter stored on a smart card. Clearly, this is an efficient and flexible method,

and any amount can be paid from the card as long as it does not exceed the value of the counter. A more involved way is to represent electronic cash by a set of electronic coins. As with ordinary coins, each electronic coin has a fixed denomination. Now, any amount can be paid as long as it can be obtained as a sum of the denominations of a subset of the available coins. By using a suitable distribution of the coin denominations upon reloading, possibly as a function of the expected spending pattern, it can be prevented in most cases that a payment cannot be completed although the total value of the coins is sufficient.

The CAFE project (see, e.g., [BBC+94]) relies on a compromise between these two basic methods. For each payment the required amount is debited from a counter but at the same time *one* special coin is used up. The special coins have no value by itself. Upon reloading the counter is credited with the withdrawn amount and the supply of special coins is replenished.

As we will explain later on in this paper, an important property that separates coins from counters is that electronic coins are the only way to achieve a system that is secure (in the bank's interest) and at the same time protects the users' privacy in a strong sense. Security-wise the nice thing about coins is that no party except the bank is able to create coins. Hence, the only way to attack the system is to duplicate coins that are already in circulation, but this is easily stopped by keeping track of spent coins.

Software-only vs tamper-resistant hardware Assuming that payers and payees need some computer device to take part in the electronic payment system, an important distinction is whether the device contains tamper-resistant hardware or not. If no part of the device is supposed to be tamper-resistant (that is, the security of the system does not rely on such an assumption), we call it software-only. Smart cards and SAMs are examples of tamper-resistant devices.

Some advantages of a software-only system are that it can be distributed easily and at low cost, it can be run on any computer hence there is plenty of computing power and disk storage available, and users do not have to acquire special hardware. Important advantages of the use of tamper-resistant hardware are that storage and use of secret keys is protected well, and that critical parts of the system run in a secured environment.

3 Money Flow

We briefly describe the money flow in the ecash system. Where appropriate we will distinguish between the ecash bank (or issuer/acquirer) and the ecash mint. The mint is the component of the ecash system where coins are created and where the databases of spent coins are held. So-called ecash accounts form the interface between the bank and the mint. In practice, several ways will be provided to transfer money to and from an ecash account. For example, an ecash issuer may provide a home-banking application that allows its customers to move money between their bank accounts and their ecash accounts. Another possibility is

that the bank accepts credit card payments, by means of which users can feed their ecash accounts.

We concentrate on the basic operations that manipulate ecash coins. Other important ingredients of electronic commerce protocols, such as certificates and receipts, are omitted as these parts are more or less independent of the way the core protocols are implemented.

Withdrawal By means of the withdrawal protocol, users are able to convert money from their ecash accounts into ecash coins. Access to the ecash account is only possible if the user is able to sign the withdrawal request, where the signature is checked against the public key registered with the ecash account.[1] The coins obtained in a withdrawal are stored on the user's hard disk. By default the coins are stored in a password-encrypted manner to prevent them from being stolen (copied).

Payment To pay a certain amount, a set of coins is selected such that the values add up to the required amount. In the on-line ecash system, this set of coins is then encrypted for the bank, using the bank's public key, to prevent that the shop or anybody else can steal (copy) the coins. The shop deposits the payment at the bank, who credits the shop's ecash account if all coins are valid and none of the coins has been spent before. Accepted coins are added to the database of spent coins so that double-spending will be detected.

Payment deposit In the on-line ecash system this protocol is part of the payment protocol as executed by the shop. In an off-line ecash system this protocol is executed at a later moment, preferably in batch mode. An important property of the payment protocol is that the payment deposit is made specific to the payee. That is, a payment deposit for a specific payee cannot be deposited to any other account than the account of the specified payee.

Coin redemption It is possible to return coins directly to the mint without using them in a payment. A natural restriction is that the number of coins that a user redeems is not allowed to exceed the number of coins that has been withdrawn by the user; a more refined way is to monitor this per coinage and per denomination. This protocol is used when expired coins are refreshed, or to improve the distribution of the coin denominations.

[1] A simple way to set up ecash clients is to assume that the software and the bank's public key are presented to the user securely (e.g., on a sealed floppy disk). The user also gets an account number and a PIN code from the bank. At home the user installs the ecash client, generates its own private key/public key pair, and registers it with its ecash account by sending it to the bank together with the PIN code (everything encrypted with the bank's public key). The user's private key can be stored in a password-encrypted manner on the hard disk.

Recovery If desired, users are able to resurrect coins that have been lost, for instance, because of a hard disk crash. By means of a special recovery protocol executed between the user and the mint, all the coins that have been withdrawn by the user since the previous checkpoint can be reconstructed. The reconstructed coins are then redeemed at the mint, which will only accept those coins that have not been spent before.

4 Cryptographic Primitives

RSA signatures For authentication of messages we currently use the RSA public key cryptosystem. Each participant picks its own RSA key pairs at random. The public key consists of a modulus $n = pq$ of prescribed size, where p, q are two randomly generated primes of equal size, and an exponent e, which is co-prime with $\phi(n) = (p-1)(q-1)$. The private key d is the multiplicative inverse of e modulo $\phi(n)$, that is, the unique number d satisfying $de \equiv 1 \bmod \phi(n)$, which we denote by $1/e$.

To sign a message $m \in \mathbb{Z}_n^*$, where the signer has private key d and public key (e, n), the signer computes the signature $s = m^d \bmod n$. To verify the signature we check that $s^e \equiv m \bmod n$; this identity must hold as we have the identity $m^{de} \equiv m \bmod n$, since $m^{\phi(n)} \equiv 1 \bmod n$ for all $m \in \mathbb{Z}_n^*$. Actually, it can be proved that $m^{de} \equiv m \bmod n$ for all $m \in \mathbb{Z}_n$.

For various reasons (e.g., because RSA signatures as described above can be existentially forged: select an arbitrary s and take $m = s^e \bmod n$, then s is a valid RSA signature on the "message" m), the actual message is usually first transformed into a related message by applying a one-way hash and/or redundancy-adding function f to it and then signing the result $f(m)$. In the next section we will see an example of such a function f.

Hybrid encryption The RSA cryptosystem can be used for encryption as well. To encrypt a message m, $0 \leq m < n$, for a receiver with public key (e, n), the sender computes the ciphertext $c = m^e \bmod n$ and sends c to the receiver. To decrypt the message, the receiver uses its private key d to compute $c^d \bmod n$, which is equal to $m^{de} \bmod n = m$.

In practice, the use of public key encryption is often limited to the encryption of session keys. A symmetric encryption algorithm is then used to encrypt the actual message with the session key. In the ecash system, we also use such a hybrid encryption method (or, "envelope method" as it is sometimes called), where we combine RSA with 3-DES (with 112 bit keys). For reasons of efficiency, it is often advantageous to use public exponent $e = 3$; encryption of the session key then amounts to two modular multiplications, and poses no security risk as long as some well-known attacks are taken into account.

Other primitives Currently we are using SHA-1 as our cryptographic hash function for the ecash system. Since a cryptographic hash function is both

e	3	5	7	11	13	17	19	23	29	31	37	41
D_e	\$0.005	\$0.01	\$0.02	\$0.04	\$0.08	\$0.16	\$0.32	\$0.64	\$1.28	\$2.56	\$5.12	\$10.24

Table 1. A binary scheme with $k = 12$ different denominations

one-way and collision-intractable, it is a powerful primitive that is used for several purposes throughout the system. We assume that the reader is aware of such constructions, and refer to the literature for notions such as secure commitments, cryptographically-strong pseudo random number generators, and password-encryption schemes.

5 Ecash Coins

We will now have a closer look at the internal structure of ecash coins. For each coinage (short for a "generation of coins"), the mint will randomly generate a fresh RSA modulus $N = pq$, keeping the primes p, q secret by storing them in a safe place. Preferably the mint's private keys are only used within the boundaries of tamper-resistant devices, while backups are kept between several entities using secret-sharing techniques. In this way, it is prevented as much as possible that private keys are compromised through attacks by insiders.

The denominations of a coinage are encoded by using different public exponents (but with the same modulus). Let k denote the number of different denominations, and let $\{e_i\}_{i=1}^{k}$ denote the first k odd primes. In order that each e_i is a valid RSA exponent, we have the condition that each e_i is co-prime with $\phi(N)$, that is, $\gcd(e_i, \phi(N)) = 1$ for $i = 1, \ldots, k$.[2] We denote the denomination that is associated with public exponent e_i by D_{e_i}; see Table 1 for an example.

To limit the storage space required for ecash coins we take full advantage of the message recovery facility of RSA signatures. This is particularly useful because ecash coins are in fact RSA signatures on small messages. Thus we get the following form, where an ecash coin C of denomination D_e consists of an RSA signature only:

$$C = f(x)^{1/e} \bmod N. \tag{1}$$

For concreteness we assume that x is 160 bits long, and let $\mathcal{H}$ denote a one-way hash function whose output length is 160 bits as well (like SHA-1). Function f is a redundancy-adding function defined by

$$f(x) = x_t \parallel \cdots \parallel x_1 \parallel x_0,$$

[2] This condition may be tested efficiently by precomputing $E_k = \prod_{i=1}^{k} e_i$, and verifying whether $\gcd(E_k, p-1) = 1$ and $\gcd(E_k, q-1) = 1$. Note that this condition can easily be met by choosing p, q from the set of safe primes ($p = 2p' + 1$, $q = 2q' + 1$ with p', q' prime), but we do not want to limit the set of candidate primes more than necessary.

with $x_0 = x$ and $x_{i+1} = \mathcal{H}(x_0 \parallel \cdots \parallel x_i)$. Parameter t is fixed such that the total length of $f(x)$ is about equal to the size of the modulus N. (Actually, the block x_t is truncated such that the integer corresponding to $f(x)$ is always less than N.) Function f may alternatively be defined by $f(x) = y_t$, with $y_0 = x$ and $y_{i+1} = \mathcal{H}(y_i) \parallel y_i$. Note that f itself is clearly not a one-way function, since $f(x)$ contains the input x as a substring.

In current ecash implementations RSA moduli used for ecash coins are at least 768 bits long; for this size forgery of ecash coins is considered entirely infeasible—certainly within the limited time-frame that a coinage is valid. Hence, a storage of about 100 bytes per coin on the user's side is required. With today's hard drives and memory chips there is absolutely no problem of storing any sensible number of coins. Over time this will only improve as the required key length for RSA is expected not to double every year or so, while the storage capacity of modern devices does.

As for the coin storage at the mint, we have the important observation that after checking the validity of a coin signature, and checking that the coin has not been spent before, it suffices to store the coin number only. As described above, we have fixed the size of the coin number at 20 bytes. The only condition on the size of the coin numbers is that it is large enough to prevent that the same coin number is accidently generated for coins of the same coinage. (As a slight refinement, note that it suffices that the coin numbers are unique per denomination.) By the standard result of the birthday paradox, the probability that no two coins will be equal when the coin numbers are picked uniformly at random from $\{0,1\}^{160}$ is bounded by approximately $e^{-B(B-1)/2^{161}}$, as long as at most B coins of the same type are generated. This shows that the probability that two coin numbers collide is truly negligible for any practical number of coins B. Since the number of coins per coinage is limited anyway (see Section 8), this analysis in principle shows that the size of the coin numbers can also be limited to 64–80 bits (8–10 bytes), say, if desired. Hence, a 1Gbyte hard disk can already store in the order of a 100 million coin numbers.

6 Protocols

We consider the withdrawal of ecash coins, the payment protocol (which includes payment deposit), and the recovery protocol. Each protocol is described at a level of detail that permits us to explain the main security features of the ecash system.

6.1 Withdrawal

For each ecash coin to be withdrawn from a user's bank account, the user and the mint execute an instance of Chaum's blind signature protocol [Cha83]. This protocol is executed in parallel as many times as required to withdraw the desired amount. The distribution of the coin denominations is chosen in such a way that

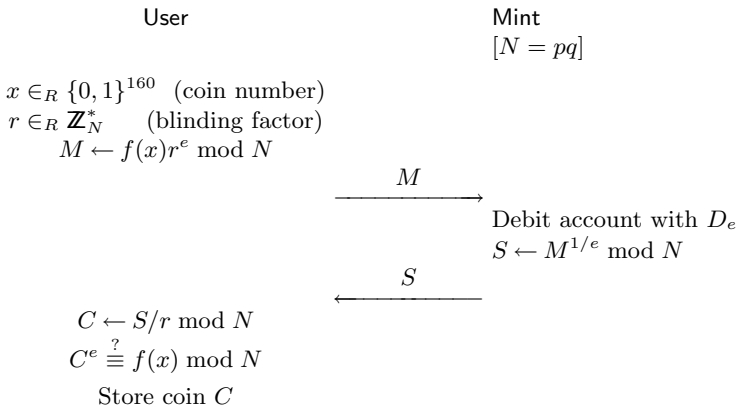

Fig. 1. Withdrawal of a coin $C = f(x)^{1/e} \bmod N$ of denomination D_e

it is guaranteed that—no matter how the money is spent—at least a certain number of payments can be made.

The withdrawal protocol for a single coin is depicted in Figure 1. Apart from the random coin number x, the user also picks a random number $r \in \mathbb{Z}_N^*$, whose e-th power is used to blind the "message" $f(x)$ to be signed by the mint. Since $r \in \mathbb{Z}_N^*$ its inverse $1/r$ exists, hence the blinding factor can be removed again to obtain the coin $C = f(x)^{1/e} \bmod N$. The complete withdrawal protocol also consists of two moves, and basically runs as follows. First, the user sends an authenticated withdrawal request message to the mint, which contains among other things a list of M-messages, one for each coin to be withdrawn. Then, assuming that there is enough money in the user's ecash account, the mint will answer with a withdrawal reply message, which contains the corresponding list of S-messages.

Regarding the security of this part of the ecash system, the important question that needs to be answered is whether it is infeasible to obtain more or other coins than prescribed by the withdrawal protocol of Figure 1. The answer is that an in-depth security analysis shows that (in a reasonable model) any way to obtain more or other coins than prescribed by the withdrawal protocol is equivalent to breaking the RSA assumption. (The RSA assumption roughly states that it is infeasible to compute $y^{1/e}$ for randomly selected $y \in \mathbb{Z}_n^*$.) For the scope of this paper, we will confine ourselves to two aspects of this analysis that are specific to the ecash system.

First, there is the fact that we use not just one, but a number of public exponents with the same RSA modulus to encode the different denominations within a coinage. This raises the question whether, for instance, coins of lower denominations cannot be combined into coins of higher denominations. More formally, the question is whether for randomly selected $M \in \mathbb{Z}_N^*$, it is feasible

to compute, say, M^{1/e_k} from the values $\{M, M^{1/e_1}, \ldots, M^{1/e_{k-1}}\}$. Recall that the e_i's are pairwise co-prime. For a very different purpose, but equally applicable to our setting, Shamir [Sha83] has shown that this question is answered in the negative: he showed that given the values $\{M, M^{1/e_1}, \ldots, M^{1/e_{k-1}}\}$ the computation is just as difficult as if these values weren't given at all.

Second, there is the fact that the mint will just accept *any* message M, and return $S = M^{1/e} \bmod N$ to the user—but each time debiting the user's account with D_e, even if the user picks the message M of the wrong form. So, there is no guarantee that the message M is of the prescribed form $f(x)r^e$, where the user knows x, r, which contrasts with the usual setting for RSA signatures, where the signer will ensure that each message signed is of the required form $f(m)$, say. In fact, by means of an example we will now show that it is possible to obtain valid coins while deviating from the prescribed protocol. Still, the user's account is debited with the total value of the obtained coins, so there's no security problem.

Consider the following approach. We are going to obtain two coins $C_1 = f(x_1)^{1/e_1}$ and $C_2 = f(x_2)^{1/e_2}$ of denominations D_{e_1} and D_{e_2}, respectively. However, instead of just asking the mint to sign $f(x_1)$ and $f(x_2)$, respectively, we do it as follows.

1. Ask the mint to sign $M_1 = f(x_1)^{e_2} f(x_2)^{e_1}$ for denomination D_{e_1}, which results in $S_1 = M_1^{1/e_1}$.
2. Subsequently, ask the mint to sign $M_2 = S_1$ for denomination D_{e_2}, which results in $S_2 = M_2^{1/e_2} = M_1^{1/e_1 e_2} = f(x_1)^{1/e_1} f(x_2)^{1/e_2}$.
3. Finally, using the fact that $\gcd(e_1, e_2) = 1$, hence there exist integers t_1, t_2 such that $t_1 e_1 + t_2 e_2 = 1$, we extract the coins C_1 and C_2 from S_2:

$$S_2^{t_2 e_2} f(x_1)^{t_1} f(x_2)^{-t_2} = f(x_1)^{(t_2 e_2 + t_1 e_1)/e_1} = C_1,$$

$$S_2^{t_1 e_1} f(x_1)^{-t_1} f(x_2)^{t_2} = f(x_2)^{(t_1 e_1 + t_2 e_2)/e_2} = C_2.$$

So, although we do not follow the protocol as described in Figure 1, we are able to obtain two valid coins anyway. (If desired, it is possible to blind the second request to the mint (step 2), such that the mint cannot detect the deviation.) The net result, however, is not very encouraging: we have obtained two coins for a total value of $D_{e_1} + D_{e_2}$, but also the account has been debited with exactly this amount.

Extending work by Shamir [Sha83] and work by Akl and Taylor [AT83], Evertse and van Heyst showed that for a very general class of deviations from the prescribed protocol, nothing is to be gained [EH92,EH93]. As shown by Chaum [Cha90], "deviations" as exemplified above can even be used to improve the efficiency of the withdrawal protocol. For example, it is perfectly safe to collapse steps 1 and 2 in the above deviation of the protocol, where the mint issues a signature w.r.t. exponent $e_1 e_2$ (charging, of course, $D_{e_1} + D_{e_2}$ for this service). This cuts the communication costs with the mint by a factor of two, and also saves the mint from performing one signature. Needless to say, this method can be extended to collapse the withdrawal of any number of coins, as long as each denomination does not occur more than once.

6.2 Payment

A simplified version of the payment protocol runs as follows. Before the payment actually takes place, payer and payee have to come to an agreement on what the object is that is going to be purchased and for which amount X. We assume that the result of this negotiation is recorded in the string `pay-spec`. A basic property of the payment protocol is that the bank will not learn anything about the string `pay-spec`, except for its hash value. To prevent that the possible values of the string `pay-spec` are limited to a small set (and hence that the bank is able to guess the value of `pay-spec` from its hash value), it is required that `pay-spec` is also randomized (by including some random string, often referred to as a "salt").

Next, the payer will select a set of coins $C_1, \ldots, C_l$ from its coin supply, such that the total value of these coins is equal to the requested amount X. To pay a payee with identity ID_{shop}, the payer then assembles a payment message that consists of the concatenation of `pay-spec`, and an encrypted message Y for the bank:

$$Y = E_{PK_{bank}}(ID_{shop} \parallel \mathcal{H}(\texttt{pay-spec}) \parallel C_1 \parallel \ldots \parallel C_l),$$

where $E(\,)$ denotes a hybrid RSA encryption method (using 3-DES), as explained in Section 4. Upon receiving this message, the payee will then sign and forward a payment deposit message consisting of $\mathcal{H}(\texttt{pay-spec})$, and the encrypted message Y to the bank. Finally, the bank decrypts Y and checks the values of ID_{shop}, and $\mathcal{H}(\texttt{pay-spec})$, and then proceeds to check the validity and freshness of the coins $C_1, \ldots, C_l$. Only if all coins are accepted, the payment is accepted as well, which means that the coins are added to the database of spent coins and that the payee's account is credited with the amount X.

A few remarks regarding security. It is important that neither the payee nor any eavesdropper can extract the coins from the payment message. For this reason, the coins are encrypted with the bank's public key. (Note that public key encryption is required in this case, because the bank and payer cannot use a shared secret key, as the payer needs to remain anonymous.) Also, note that although the string `pay-spec` itself is never shown to the bank, the payee is sure that the payer used the same string, since the bank compares the hash values provided by the payer and payee, respectively. In this way the transaction details remain hidden from the bank; if required, however, either the payer or the payee can later reveal the string `pay-spec` which can then be checked against the data stored at the bank.

Finally, let us briefly describe how interrupted payments can be resolved, in case the payer and payee aren't able to recover from interruptions by normal means. Due to network problems, for example, the payment protocol may be interrupted at several stages, but to the payer it only matters whether (i) the payment wasn't processed by the bank, or in any case it was not credited to the payee's account, or whether (ii) the payment was processed by the bank and credited to the payee's account. To find out about the status of a payment, the payer can either redeem the coins $C_1, \ldots, C_l$ on an individual basis, or find out about the complete payment by submitting message Y to the bank.

In the latter case, the payer proves to be the owner of the payment by revealing the value of `pay-code`, to which the user committed by including $\mathcal{H}$(`pay-code`) in the message Y—this commitment is omitted in the above description. If the payment turns out to be credited to the payee's account, the bank signs a statement to this effect, which the user can then show to the payee to prove that the payment arrived at the payee after all.

6.3 Recovery

In the previous section we have mentioned two ways to recover from interrupted payments. Similarly, there are measures to recover from interrupted withdrawals (applying techniques from the field of transaction processing, see, e.g., [GR93]). In this section we describe the basic idea behind a special recovery protocol that allows users to recover from more severe problems as well. For example, in the exceptional event that all information on the user's hard disk gets corrupted, the recovery protocol allows a user to start all over again, and to get reimbursed for the supply of coins that were stored on the hard disk at the time of the crash.

We are able to do so provided the random coin numbers and blinding factors as used in the withdrawal protocol of Figure 1 are generated in a pseudo-random fashion. As part of the set up procedure of an ecash client, the user will obtain what we call a recovery string, which the user must store in a safe place. The seed of the PRNG is fully determined by the recovery string (and some other information private to the user). As part of the recovery protocol, user and mint than cooperate to reconstruct all coins that have been withdrawn since the previous checkpoint. Subsequently, all reconstructed coins are redeemed and the user gets reimbursed for the coins that have not been spent before.

So much for a brief description of the recovery protocol. At this level of detail the following three remarks regarding security and privacy are in order. First, it is clear that the recovery string must be sufficiently long such that it is infeasible to find it by exhaustive search, say at least 16 bytes. Second, it is important that the mint ensures that the recovery protocol does not yield coins that have never been actually withdrawn by the user. To this end the mint does some additional bookkeeping per withdrawal (since the last checkpoint). And, finally, note that whenever a user requests a recovery some privacy is lost, because the bank learns which coins have been spent since the last checkpoint.

7 Privacy

We present the by now standard argument why the ecash system protects the privacy of its users [Cha83]. To fully protect the users' privacy, the system must satisfy the requirement of *unlinkability*; a system in which each user gets a pseudonym is not sufficient to protect the privacy of its users.

Unlinkability This property says that it should be impossible to determine whether any two payment transactions originate from the same user or not.

A moment's thought will show that the property of unlinkability implies that *individual* payment transactions are not traceable to the user who acted as the payer in such a transaction: if they were traceable, two payment transactions are linked whenever they can be traced back to the same user.

To appreciate the strength of unlinkability, consider the following scenario. When you buy a prepaid telephone card you can do this completely anonymously at a newsstand (paying cash). Later when you use the card in a public phone the telephone company will have no clue that it is you making the phone call because you bought it anonymously. That is, the individual telephone calls are untraceable, as they cannot be connected to your identity. Suppose however that the telephone company gives every card a unique number, which is quite realistic as this is a basic mechanism to detect fraud (i.e., to find cards on which the total spent is larger than the card's value). Then it is easy to keep a file per card of all phone numbers called from that card (and possibly the time and date of the calls as well). Since a similar file is kept per home-phone as well, a simple pattern matching procedure will in many cases reveal the identity of a card's owner. Thus, although the card is obtained anonymously (and the card number acts as a pseudonym), the identity of the card's owner can be revealed anyway because all calls from the same card are linkable.

We now argue why the ecash protocols protect the users' privacy. Consider a fixed denomination D_e. Referring to Figure 1, we will show that *any* of the signatures S issued by the mint to any of its users will match equally well with *any* of the coins C spent by any of the users. So, let S_A denote a signature of denomination D_e that has been issued to some user A, and let C_B denote any coin of denomination D_e that has been spent by some user B (users A and B are not necessarily different). The mint knows that signature S_A has been issued to user A, but the mint does not know that coin C_B belongs to user B. Unlinkability now holds on account of the following reasoning: since for any pair (S_A, C_B) there exists a *unique* blinding factor $r_{AB} \in \mathbb{Z}_N^*$ that satisfies $C_B = S_A/r_{AB} \bmod N$, we have that each signature issued by the mint matches equally likely with any of the coins spent. There is no bias between the case that $A = B$ and the case that $A \neq B$, which implies that there is no way the bank will be able to link coins that belong to the same user.

8 Coinages

In practice, an ecash mint will work with several *coinages* at the same time. For each coinage the mint generates a fresh RSA modulus, as described in Section 5. Hence, there is a one-to-one correspondence between the RSA moduli and the coinages. Apart from the RSA modulus, a coinage has at least the following attributes: the identity of the mint, the sequence of denominations, the currency, and a key schedule (i.e., expiration dates) as in Figure 2. These attributes are distributed with each coinage. Note that the private key can in fact be removed as soon as withdrawals are deactivated, while the public key must be available until the coinage becomes invalid.

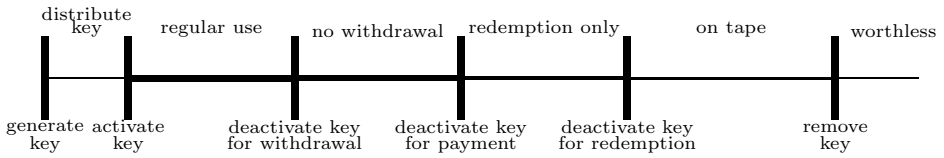

Fig. 2. Life-cycle of a coinage

There are two main reasons why coinages are refreshed on a regular basis, say every six months. First, there is the standard reason for refreshing keys, namely to limit the risk that the secret key is compromised. There are two direct ways this may happen. In the first method, the attacker will try to find the secret key from the public key only, possibly using the mint as a signing oracle. In the second method, the attacker simply tries to get hold of the secret key by breaking into the bank or into its computer network, possibly assisted by insiders.

The other reason is to limit the size of the "spent coin" databases. Using a scheme like that of Figure 2, spent coins are first stored on disk to enable fast checking for duplicates. After some time the coins are moved to tape, and during that period it will still be possible to check for duplicates, but using a slower procedure (which is only available so that users can return coins when they haven't used their ecash for a longer period of time). Eventually, a coinage will become invalid and the tapes can be removed.

Apart from a division in time, where coinages are refreshed every six months say, we can also use a division in space. That is, instead of viewing all users as one big set of users, it makes sense to divide the users in clusters. Each cluster is chosen sufficiently large such that the behaviour of the individual users is not visible. The advantage is that by limiting the size of a cluster that the corresponding parts of the "spent coin" database are independent of each other, which enhances the scalability of the system.

The way the bank divides its users into clusters should be publicly verifiable, and not at the discretion of the bank (to prevent the bank from introducing a few small clusters). There are many ways to accomplish such a fair division into clusters. A simple idea is to first take a cryptographic hash of the user's identities, and then define 2^t clusters, $t \geq 0$, by looking at the first t bits of the hash value. Assuming that the bank cannot influence (the representation of) the user's identities, the users are thus evenly spread over the clusters.

9 Some Extensions

Credentials If desired, other types of clusters like age-groups can also be encoded in the ecash system by reserving a different coinage for each age-group. Certain shops could then be made to accept ecash from certain age-groups only. Clearly, this is just an example of the general token functionality that can be

achieved with the ecash protocols. The token functionality is in turn subsumed in the general notion of electronic credentials, which encompasses things as diverse as theatre tickets, driver's licenses, passports, diplomas, etc. (e.g., see [Cha92]). A crucial property of a credential system is that it allows credentials to be shown in such a way that only the part of the credential relevant to the situation is revealed. For example, when using your driver's license to show that you are over twenty, the protocol for showing this fact should only reveal to the doorman that your day of birth is more than twenty years ago, but should give him no clue about your exact age. The cryptography behind these protocols is based on the general notion of proofs of knowledge.

Off-line electronic cash It is interesting to consider the development of privacy-protecting off-line cash protocols. To be precise we are considering off-line prepaid electronic coin systems with public key authentication, where tamper-resistant hardware is used only by the payers (see Section 2). A basic concept of these systems is Chaum's one-show blinding paradigm, which says that the privacy of a user is completely protected as long as the user spends each coin not more than once; however, if a user is able to manipulate its payment device such that some coins are used more than once, the protocols are such that the identity of the double spender can be computed. Since the introduction of the one-show blinding paradigm and the first solutions to the problem ten years ago [CFN90], the protocols have now developed into an efficient and relatively simple system for off-line cash.

A problem with early off-line cash systems was the fact that they heavily relied on computationally expensive cut-and-choose techniques. As a consequence, both the computational and communication complexity of these protocols was quadratic in the security parameter. A system avoiding cut-and-choose was first proposed in [Fer94], which is based on RSA and achieves linear complexity. Linear complexity is also achieved in [Bra94a], this time based on a clean Discrete Log setting: using Schnorr's identification protocol [Sch91] and derived primitives, a relatively simple and efficient system is obtained with a rich set of security properties. Then [Bra94b] presents a general construction (which can be used to built systems either based on RSA or based on Discrete Log). In [Sch95] a system is presented which even withstands parallel attacks by two or more users and is efficient as [Bra94b]; however, the technique only works for a Discrete Log setting. The work on the CAFE system indicates that this type of payment systems can be implemented in a practical way.

References

[AT83] S. Akl and P. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Transactions on Computer Systems*, 1:239–248, 1983.

[BBC+94] J.-P. Boly, A. Bosselaers, R. Cramer, R. Michelsen, S. Mjølsnes, F. Muller, T. Pedersen, B. Pfitzmann, P. de Rooij, B. Schoenmakers, M. Schunter, L. Vallée, and M. Waidner. The ESPRIT Project CAFE – High Security

Digital Payment Systems. In *Computer Security – ESORICS 94*, volume 875 of *Lecture Notes in Computer Science*, pages 217–230, Berlin, 1994. Springer-Verlag.

[BGH+95] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. iKP — a family of secure electronic payment protocols. In *First USENIX Workshop on Electronic Commerce*, 1995.

[Bra94a] S. Brands. Untraceable off-line cash in wallet with observers. In *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318, Berlin, 1994. Springer-Verlag.

[Bra94b] S. Brands. Off-line cash transfer by smart cards. In V. Cordonnier and J.-J. Quisquater, editors, *Proceedings First Smart Card Research and Advanced Application Conference*, pages 101–117, 1994. Also as report CS-R9455, Centrum voor Wiskunde en Informatica.

[CFN90] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology—CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327, Berlin, 1990. Springer-Verlag.

[Cha83] D. Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology—CRYPTO '82*, pages 199–203, New York, 1983. Plenum Press.

[Cha90] D. Chaum. Online cash checks. In *Advances in Cryptology—EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 288–293, Berlin, 1990. Springer-Verlag.

[Cha92] D. Chaum. Achieving electronic privacy. *Scientific American*, pages 96–101, August 1992.

[EH92] J.-H. Evertse and E. van Heyst. Which new RSA-signatures can be computed from certain given RSA-signatures? *Journal of Cryptology*, 5(1):41–52, 1992.

[EH93] J.-H. Evertse and E. van Heyst. Which new RSA-signatures can be computed from RSA-signatures, obtained in a specific interactive protocol? In *Advances in Cryptology—EUROCRYPT '92*, volume 658 of *Lecture Notes in Computer Science*, pages 378–389, Berlin, 1993. Springer-Verlag.

[Fer94] N. Ferguson. Single term off-line coins. In *Advances in Cryptology—EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 318–328, Berlin, 1994. Springer-Verlag.

[GR93] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Francisco (CA), 1993.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sch95] B. Schoenmakers. An efficient electronic payment system withstanding parallel attacks. Report CS-R9522, Centrum voor Wiskunde en Informatica, March 1995.

[Sha83] A. Shamir. On the generation of cryptographically strong pseudorandom sequences. *ACM Transactions on Computer Systems*, 1:38–44, 1983.