VarietyCash: a Multi-purpose Electronic Payment System

(Extended Abstract)*

M. Bellare† J. Garay‡ C. Jutla§ M. Yung¶

Abstract

VarietyCash is a new electronic money system which strikes a balance between functionality, security, and user privacy. The system can encompass both network and stored-value card based payment mechanisms, with transferability between them, hence the name.

1 Introduction

Electronic payment systems are well-understood to be an essential step on the road to electronic commerce, and are thus in high demand. These systems need to strike a good balance between a number of different issues. Amongst these are:

- *Anonymity:* The extent to which a third party (bank or other payment service provider) or the merchant has information about the identity of the buyer.
- *Account-based or account-less:* Account based systems are higher cost from the point of view of the service provider. In the US, at least, so-called Regulation E obligates the service provider of account-based systems to provide customers with a level of account maintenance (e.g., periodic statements) which increases the cost of such systems.
- *Network versus card-based:* Some systems are software-based for transactions across the network; in others, like Mondex [14], stored-value cards are used.
- *Atomicity:* The system should be fair and robust in the sense that network failures, for example, do not result in incomplete transactions [22].

We propose a simple, versatile system that is *account-less*, provides some degree of anonymity in the form of *anonymity by trust*, is versatile enough to allow both network-based and card-based payment to inter-operate in the system, and provides atomicity. The system is the result of a thorough system's analysis, design, and proof of concept carried out by the authors for a large financial institution.

1.1 Background

There are many good reasons to have some degree of anonymity: people want to avoid being added to mailing lists, or simply keep secret the kinds of goods they buy; businesses may not want competitors to know what kinds of information they are buying. The strongest form of anonymity (unconditional) is provided by systems based on so-called blind signatures [10]. (These systems were first proposed by Chaum [8]. At this point, there are many of them published.) However, these systems have many well-known drawbacks. Tygar and Yee (as reported in [22]) point out that the protocols lack atomicity: network failures during a transaction will result in loss of the electronic money involved. That double

*Appears in *Proc.3rd Usenix Workshop on Electronic Commerce*, Boston, MA, August-Sept. 1998.

† Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: mihir@cs.ucsd.edu. URL: //www-cse.ucsd.edu/users/mihir. Supported in part by a 1996 Packard Foundation Fellowship in Science and Engineering, and by NSF CAREER Award CCR-9624439.

‡ Information Sciences Research Center, Bell Laboratories, 600 Mountain Ave, Murray Hill, NJ 07974. E-mail: garay@research.bell-labs.com. URL: www.bell-labs.com/user/garay/.

§ IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, New York 10598. E-mail: csjutla@watson.ibm.com.

¶ CertCo, 55 Broad St., New York, NY 10004. E-mail: moti@certco.com.

spending is only caught after the fact is considered too high risk by banks. Another problem (as pointed out in [16]) is scalability: as all coins that have been spent have to be recorded, the coin data base will grow over time, increasing the cost to detect double spending. Attempts to address some of these issues have been made (e.g. via escrow [13]) and stored-value card based settings mitigate others [7], but drawbacks remain.

Mondex [14], on the other side of the spectrum, provides no anonymity. It seems to be a shared key based card design. There are well-known scalability and security problems with such designs. For example the security is totally dependent on the security of a master key, and this master key must be distributed to many users and points of sale across the system. If a single module is penetrated, not only is significant retailer fraud facilitated, but the entire card base may be compromised.

NetBill [19] has a large spectrum of desirable properties, including atomicity. It might be able to provide some limited anonymity via trust in the NetBill server and the use of pseudonyms, but linkage is still possible. Also the NetBill server maintains accounts for buyers and merchants, which is costly. Also, NetBill's transaction protocols include functions that are beyond payment, such as price *negotiation*. The trend nowadays is that these functions should be separated, and standardized; e.g., JEPI [12], SEMPER [20].

Our design (its network version) shares many features with NetCash [16, 15] (NetCash has no "off-line" operation, meaning the ability to also work with stored-value cards). The NetCash protocols also combine symmetric-key cryptography for performance with public-key mechanisms. A consequence of shared-key operations is no non–repudiability in some key functions (e.g., in exchanges with currency server). Anonymity in NetCash is also trust-based, with multiple currency servers which the client selects; this in turn implies an accounting infrastructure in order to maintain consistency accross servers, etc. [17]. Additionally, NetCash is cast as a framework that can accommodate various electronic currency mechanisms,

*i*KP [3], SET [21], etc., are credit card-based payment systems. We are looking for some form of cash.

1.2 Variety Cash

In *VarietyCash*, the issuer functions as a mint. Coins are tokens authenticated under an issuer master key. The system is on-line, meaning the issuer maintains a coin database, and the merchant checks on-line with the issuer that a coin has not been previously spent. Since the system is on-line, the master key is present only at the issuer, so, unlike in Mondex, can be well protected.

Spent coins can be erased from the database, unlike some systems; if this were not possible scalability would be adversely impacted.

VarietyCash provides "trust-based anonymity." At withdrawal time the issuer does note the association of user ID to coin serial numbers, but this user database is separate from the database recording the spent or unspent status of a coin which is looked up when a merchant wants to confirm that no double spending has occurred. The information can be co-related as necessary in case of law enforcement needs, but the issuer is expected to maintain user privacy except in such cases. While not anonymity at the level of digicash, this relies on no more trust than we put in our financial service providers today, and with a well-known, reputable service provider, is likely to be deemed adequate by most people for most things. Furthermore, users can use pseudonyms.

The system is robust, functional and flexible. The protocols for withdrawal and spending will provide atomicity. Coins can be purchased in any number or denomination, and paid for in a variety of ways. The system is account-less, so Regulation E is avoided, and the payment service provider does not have to issue statements to customers or incur other such overhead. The same coins can be used for network based payment transactions or put on stored-value cards in the Mondex style, and the coins are transferable between the two. Novel features include transferability of coins without any specific payment transaction, ie. "making change".

The main concern is cost arising from it being an on-line system. The issuer will have to invest some resources to be able to handle lots of transactions quickly. This, however, appears feasible for reasonable load. Our protocols minimize the overhead. Batching and aggregation can be used to reduce costs (e.g., [4]).

This design originated in response to a request of a certain large financial service corporation to seek a practical, viable e-money system. In discussions with them we found they were more ready to take on the cost of on-line verification than to incur the risks arising from anonymous cash, or to fall under Regulation E. From this it appears that there is a market for a system like *VarietyCash*.

1.3 Implementation

Our design has been implemented in C++. The core of the system can be viewed as a distributed cryptographic application framework. The issuer and merchant sides have been implemented to run on AIX, whereas the client (buyer) runs on Windows 95. A buyer can access web-sites of merchants through a browser. If the web-site is e-money-enabled, on a purchase it returns a document with a special mime-type, which the browser has been configured to recognize. The browser then invokes an external e-money daemon. The daemon launches (if not already running) a graphical user interface for the client, and rest of the communication between the three parties takes place via the daemon. The transaction databases have been implemented persistently, so that malfunctions such as network breakdowns do not hamper the protocol.

1.4 Organization of the paper

We start in Section 2 by describing the model and system architecture of the main component of *VarietyCash*, namely, a network-based, on-line payment system. Some of the design considerations and requirements for the processes we present may be of wider applicability. In Section 3 we concentrate on two processes (and corresponding protocols) in detail: withdrawal of e-money and payment. Finally, in Section 4 we show how to integrate to the network component a typical card-based payment system.

2 Model and System Architecture

2.1 Security design goals

We start by identifying the basic aspects that compose the security of an e-money system.

PROTOCOL SECURITY. By this we mean liveness and safety guarantees, namely, that the protocols achieve their goals and that every participant gets its information, and is secure in the sense that the other parties which are considered adversaries do not compromise or spoil the system. This aspect is the main focus of this paper.

INTERNAL SECURITY. The security of the internal operation system of the issuer of electronic currency, its capability to withstand insider attacks and abuses. The internal network architecture, operation policies, employment of tamper-proof hardware as well as dual control measures and access-control and physical access limitations should be reviewed. The internal security architecture has to be combined with issues such as availability, reliability, load balancing and back-up requirements.

NETWORK SECURITY. The security of the network (e.g., Internet) of users and the issuer, to prevent attacks not via the protocol but rather through "break-ins;" these attacks exploit the lack of proper protection into the system and software holes. Careful design of the interface to the external network (firewall protection) is required. Both the internal and the network systems have to be evaluated under "Global Security Testing," which includes penetration attempts and security assessment of design and implementation.

USER SECURITY. Security of the user's assets. The user must obviously protect his electronic currency, and the software and procedures supplied to the user have to provide for protection at a proper level (e.g., beyond password-only protection), but at the same time be user-friendly.

In this paper, we deal specifically with the protocol aspects and their security. In this presentation we do not cover all the protocols, but what we cover seems to capture the basic needs of the system. For simplicity, nor do we deal with the temporal aspects of maintaining the system, such as long-term key management and cryptographic policies.

2.2 Parties and roles

We will use the following terminology for the *parties* involved in *VarietyCash*. In a typical transaction there would be three parties involved: the payer, the payee, and the bank or e-money server. However, we would like to be more specific than that:

- Participant – A generic name for a party involved in the system.
- Issuer (I) – Party who issues coins.
- Coin-holder (CH) – A party who holds coins or has the potential to do so.
- Payee (PY) – A party who is willing to accept coins as payment (e.g., a merchant).
- Bank (B) – A number of banks will be involved in moving funds due to conversions between electronic and real money.
- Certification Authority (CA) – The party who can "certify" the public keys of the participants. (In some scenarios, this role can be played by the Issuer itself.)

In turn, a coin-holder may play the following **roles**:

- Coin Purchaser (CP) – purchases coins from issuer
- Redeemer (RD) – turns coins into real money
- Payer (PA)– customer who pays for goods/services with coins
- Refresher (Rf) – gets new coins for old
- Changer (Ch) – makes change

Further, we make the following distinction amongst participants:

- Registerer (RG) – Register a public key at the issuer
- Enroller (EN) – Enroll for a particular role such as coin purchaser or merchant

The idea here is that a participant may wish to be able to "play the game," i.e., accept and use coins as a form of payment, without going through the more elaborate process that enables the purchase or redemption of e-money. Note that parties and roles are not mutually exclusive.

In *VarietyCash* all the participants have distinct identities, as well as public keys. The issuer has a database listing the identities of all participants, and, for each one, its public key. This information is entered at the time of registration. In all transactions directly involving the issuer, there is thus no need for an external certification authority; however, the design leaves an option for such a case.

We assume in this extended abstract a unique issuer, thus dispensing with the presentation of a more elaborate clearing system.

2.3 Adversaries and attacks

We distinguish between two major kinds of attackers: the *abusers* and the *spoilers*. Abusers try to get some specific advantage, such as get coins without properly paying for them; forge or steal coins; etc. Spoilers do not seek any advantage *per se*; they just try to disrupt the system. An example of a spoiler attack is to replay a coin purchase request in an attempt to make an unnecessary transfer of funds (from the withdrawer's bank to the issuer). Another example is the *denial of service* attack in which the attacker tries to tie up issuer resources.

Attackers may be *external* (e.g., on the Internet lines), or they may be parties themselves (for example, a malicious payee or withdrawer trying to get some money for free), or they may be *insiders* (such as an employee at the issuer).

Active attackers are the main problem. Pure eavesdroppers only try to learn information, such as the nature of a transaction. Our protocols will be designed to resist active attacks.

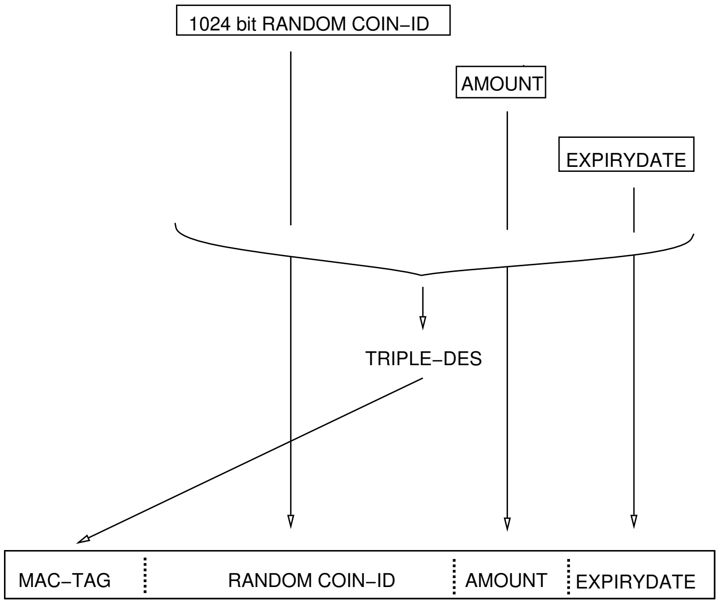
We do not discuss error handling or denial of service attacks. An adversary can always interrupt a flow and thus disrupt a protocol. It is assumed that standard re-transmission and time-out procedures underly the transmission of protocol flows and address these attacks as well as possible.

2.4 Coins and cryptographic terminology

An *e-coin* (*coin* for short) is an object consisting of a unique identifier (serial number, counter) called the *coin ID*, an indication of value (denomination), an expiry date, and an authenticating cryptographic tag. This cryptographic tag can be implemented in different ways. In our design we choose to use a MAC, or message authentication code, on the rest of the information, computed using a symmetric key which is kept in secret by the issuer. Thus, the issuer can compute and verify the tag, but no other party can compute a valid tag. Because it is secret-key based, the tag cannot even be checked by anyone except the issuer, i.e., this is not a digital signature. This is done for efficiency reasons—symmetric key based cryptography is hundreds of times more efficient than public-key operations. A schematic representation of a coin is shown in Figure 1.

The cost of successful tag forgery can be enormous, since if an adversary could forge tags, he or she could manufacture counterfeit coins at will. In order to minimize the possibility of successful tag forgery, we suggest the following:

- First, the tag should be computed in protected, tamper-proof hardware. This minimizes the risk of loss of the secret key.
- Second, the tag-computing algorithm should be "strong." One can of course use MACs based on existing primitives such as DES. This may be acceptable, but we suggest that a Triple-DES based MAC be used. A good choice would be a Wegman-Carter [23] type MAC, meaning that one applies an XOR-universal hash function to the data and then XORs the result with the value of a Triple-DES based pseudorandom function applied to a counter. The size of each tag should be (at least) 128 bits, so the total tag length is 28 bytes. Outside the cryptographic module the tag is encrypted inside the issuer database, and it goes encrypted over the network. It is only available in plain form to the receiver of the coin.



TRIPLE−DES MAC−TAG above is computed using a secret key K1 of the mint .

Figure 1: *Structure of an un-encrypted coin.*

Additional protection is provided by the fact that the coin database itself is protected. So that if an adversary manages to forge the correct tag for a coin which has not yet been issued, it will not help, because the issuer will fail to validate the coin. Thus, the adversary must be able to successfully forge tags of issued, un-spent coins to achieve a meaningful gain.

A coin can have various *states*. For example, spent or not; anonymous or not; split, and if so how; etc. These are marked in the database.

The issuer will expect from a coin purchaser requesting coins, or a change maker wanting change, a specification of exactly what kinds of coins are being requested. The specification takes the form of a list of denominations, and for each denomination, the number of coins of that denomination that is desired. The simplest thing is to just ask for one coin of a certain denomination, for example a single coin of \$0.80. But one could ask for, say, $(2, \$2.50), (1, \$1.25), (3, \$2)$, meaning I want 2 coins of value \$2.50 each, one coin of value \$1.25, and 3 coins of value \$2. The *total value* of the list is the total dollar value, \$12.25 in the example just given. The choice of specification is decided by a combination of the user's needs and the software (*purse*).

The cryptographic primitives used in the protocols of Section 3 are summarized in Figure 2. All the parties have public keys. The issuer has a cache of identities and their corresponding public keys, so that the certification authority is not needed in transactions with the issuer. (But it may be needed for other transactions.)

The encryption function $\mathbf{E}^*_X$ must provide, besides secrecy, some form of "message integrity." Decryption of a ciphertext results either in a plaintext message, or in a flag indicating non-validity. Formally, the property we require of the encryption scheme is *plaintext awareness* [5, 2]. Roughly speaking, this means that correct decryption convinces the decryptor that the transmitter "knows" the plaintext that was encrypted. This is the strongest known type of security, and in particular it is shown in [2] that it implies non-malleability (it is not possible to modify a ciphertext in such a way that the resulting plaintext is meaningfully related to the original one, as formalized in [11]) and security against chosen-ciphertext attacks. A simple, efficient scheme to achieve such encryption using RSA is OAEP [5]. A Diffie-Hellman based solution can be found in [1]. (Note that the RSA PKCS #1 encryption standard can be broken under chosen ciphertext attacks [6],

- **Keys:**

$\mathrm{PK}_X, \mathrm{SK}_X$	Public and secret key of Party X
CERT_X	Public key certificate of Party X, issued by CA. We assume it includes X, PK_X and CA's signature on PK_X.

- **Cryptographic primitives:**

$\mathcal{H}(\cdot)$	A strong collision-resistant one-way hash function. Think of $\mathcal{H}(\cdot)$ as returning "random" values.
$\mathbf{E}^*_X$	Plaintext-aware public key encryption using PK_X
$\mathbf{S}_X(\cdot)$	Digital signature with respect to SK_X. Note the signature of message M does NOT include M. We assume the signature function hashes the message before signing.
$\mathbf{e}_K$	Symmetric key based encryption algorithm, taking key K and a plaintext, and producing the ciphertext
$\mathbf{mac}_K$	Symmetric key based signature, or message authentication code (MAC), taking key K and a plaintext, and returning a short tag.

Figure 2: *Keys and cryptographic primitives used in protocols*

and is thus not suitable for our purposes.)

However we stress that plaintext-aware encryption does not provide authentication in the manner of a signature, i.e., it does not provide non-repudiation. But it prevents an adversary from tampering with a ciphertext.

Also note that the encryption function is *randomized:* $\mathbf{E}^*$, invoked upon message m will use, to compute its output, some randomizer, so that each encryption is different from previous ones.

Finally, the notation $\oplus$ denotes bitwise XOR.

2.5 Databases, modules, components

We briefly mention the components that interact with the processes we describe. Briefly, a participant has a purse, which has a database of coins. The issuer has a coin database and a participant database. These indicate the status of a coin, including the withdrawer ID if anonymity was not requested. A schematic view of the issuer's coin database is shown in Figure 3.

The issuer has a secure cryptographic module for coin generation. The coin-generation key is hardware-protected inside this module.

2.6 General requirements and principles

A global requirement is the *conservation of cash.* This means that the total e-money in the system is equal to the total amount of real money that the issuer's logs show is in e-money.

A general principle is that any coin representation is seen by at most one participant other than the issuer. Thus, after an issuer issues a coin, the withdrawer is the only party who "sees" this coin. When the withdrawer makes a payment with it, the payee doesn't see the coin; it is in a digitally sealed envelope which goes straight to the issuer for validation. This implies "on-line" payments where the issuer is involved in every such transaction. In addition, the internal representation of the coin does not enable insiders with access to its database to use it—a coin is checked by a tamper-proof hardware for its validity.

Coins are treated as bearer instruments, like real currency. The user has the money if s/he has a (valid) coin; no questions asked.

The security requirements that we pursue are those of "strong cryptography" for the protection of financial transactions; the use of "weak cryptography" only (e.g., 40 bit-long keys and passwords) is insufficient for e-money. International usage of the system is possible if the encryption is not made "general purpose," but is rather restricted to the use

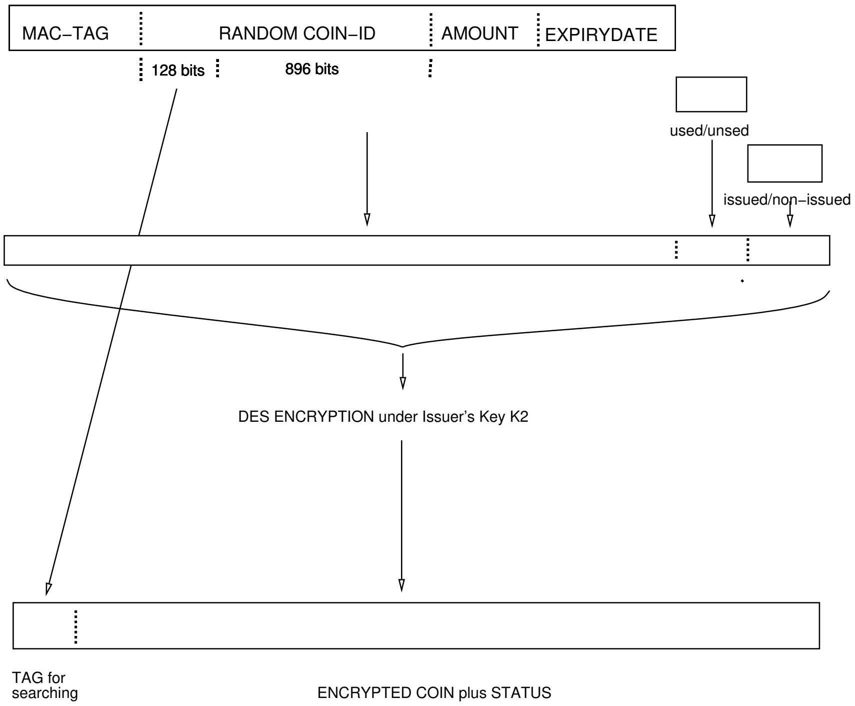


Figure 3: *Encrypted coin in the Issuer's coin database.*

inside the user's software.

3 Processes and Protocols

We first enumerate the basic processes taking place in *VarietyCash*, and the parties involved in them. Later we describe in detail the requirements and protocols that realize them for the two more relevant processes: *Coin Purchase* and *Payment*.

- Registration – A party chooses an identity, and has his/her public key registered at the issuer.
- Enrollment $(PY, I$ or $PA, I)$ – An already registered participant enrolls for a role such as coin purchaser or redeemer.
- Coin Purchase (CP, I, B) – The withdrawer specifies what kinds of coins s/he wants, and a corresponding set of coins is then issued and sent to the withdrawer. The coins are paid for by funds of value equal to the total dollar value of the issued coins, which the withdrawer authorizes the issuer to get from his/her bank account.
- Payment (PA, PY, I) – The payer pays to the payee a requested sum, using one or more coins totalling to the requested value. The payee immediately (i.e., on-line) validates the coins with the issuer, and may either obtain new coins in return, redeem the coins, or aggregate them for later redemption.
- Change (CH, I) – The coin-holder gives the issuer a set of coins, and also specifies what kinds of coins he wants in change, the total dollar value of the requested coins being the same as that of the provided coins. Coins corresponding to the request are then issued to the coin-holder.
- Redeem (RD, I, B) – The redeemer gives some set of coins to the issuer, and the latter turns them into real money in the redeemer's bank account.
- Refresh (Rf, I) – The refresher turns in old (expired) coins for an equivalent value in new coins.
- Refund – A non-anonymous coin holder can request the issuer to resend him his coins in case of a failure (e.g., disk crash).

Again, we note the distinction between the Registration and Enrollment processes. Intuitively, Registration is a simpler, on-line process that will let a user participate in the transactions. It basically consists of chooshing and ID (possibly a *pseudonym*) and a secret/public key pair, and making sure that requirements such as availability and security (e.g., minimizing spoiler attacks) are met. On the other hand, Enrollment is accomplished by a combination of on-line and out-of-band steps. It comprises steps such as providing DDA/credit card information, validation of this information, and issuance of initial amount. Being enrolled allows a user to play an "active" role in the system, in the sense of generating money conversions from regular to electronic, and viceversa. We leave the details of these two important processes for the full version of the paper, and will assume in the following description that they have already taken place.

3.1 Coin Purchase

The Coin Purchase process involves the coin purchaser, the issuer and the coin purchaser's bank. The coin purchaser specifies how much he wants in e-coins, and in what denominations, and provides the information to make the coin purchase from the bank. The issuer makes the coin purchase and then issues the coins.

The requirements are as follows:

W1– *Valid transactions go through.*

W2– *Can't get coins for nothing.* It is not possible to get coins without paying for them: if a party ends up getting a certain dollar value of valid coins, then the issuer has the corresponding funds from the same party's bank account.

W3– *Can't create false debits.* An adversary may want to play spoiler: it doesn't want coins, but wants the coin purchaser's account to be unnecessarily debited. This should not be possible. That is, it is not possible for an adversary to create a fake coin request which leads the issuer into debiting the coin purchaser's account.

A protocol for the coin purchase process is shown in Figure 5. The field W-DESC contains two things. First, the type and number of coins he wants: this is a list of denominations, and, for each denomination, the number of coins of that denomination that are desired (see Section 2.4). Second, information necessary to enable the issuer to get paid, in real funds of equal value to the total dollar value of the coins. Here we take the case that this payment is made by coin purchase from the coin purchaser's bank, so this information includes the bank name and address, and the coin purchaser's account number.[1]

[1] Another possibility is that this payment is made by credit card, in which case an iKP/SET-type protocol [3, 21] may be used, instead of the protocol we are describing here. The issuer would play the role of the merchant in SET.

Note that this protocol does not make use of a certification authority. It assumes that the parties have each other's public keys and certificates already cached. The coin purchaser has the issuer ID and public key in his purse from enrollment, and similarly the issuer has the coin purchaser ID and public key in his database from enrollment.

There are three transactions: the coins request, in which the coin purchaser asks for the coins and provides the bank information; the execution, in which the ACH transaction is done; and the final issuance of coins. The protocol is designed to guarantee both privacy and authenticity of the data. This is to protect the coin purchase information and the coins that are issued. It must also provide freshness. For efficiency's sake we use a key exchange protocol to get a session key K under which later messages are encrypted or MACed. However, the coin purchase information is digitally signed for non-repudiability. We now go over the transactions in more detail.

(1) Coin Request. The coin purchaser requests that a certain amount in coins be returned to him, and authorizes the issuer to withdraw this amount from his bank account. The protocol begins with a key exchange which issues the key $K = K_{CP} \oplus K_I$ to both parties:

  (1.1) WRequest1. The coin purchaser chooses a random number K_{CP}, and then encrypts his identity ID_{CP} and K_{CP} under the public encryption key of the issuer, using the plaintext aware encryption algorithm. The ciphertext is passed to the issuer.

  (1.2) WRequest2. The issuer applies the plaintext-aware decryption algorithm to the received ciphertext. If this algorithm rejects the text as non-authentic then he rejects; else be obtains and records the identity ID_{CP} of the withdrawer and K_{CP}. Now he chooses a random number K_I and also a random nonce R_I. He uses ID_{CP} to retrieve PK_{CP} and then encrypts ID_I, K_I, R_I under PK_{CP} using the plaintext aware encryption algorithm. The ciphertext is sent to the coin purchaser. The value $K = K_I \oplus K_{CP}$ is stored as the session key.

  (1.3) WRequest3. The withdrwawer applies the plaintext-aware decryption algorithm to the received ciphertext. If this algorithm rejects the text as non-authentic then he rejects; else be obtains and records the identity ID_I, and the numbers K_I, R_I. He checks that the identity is really that of

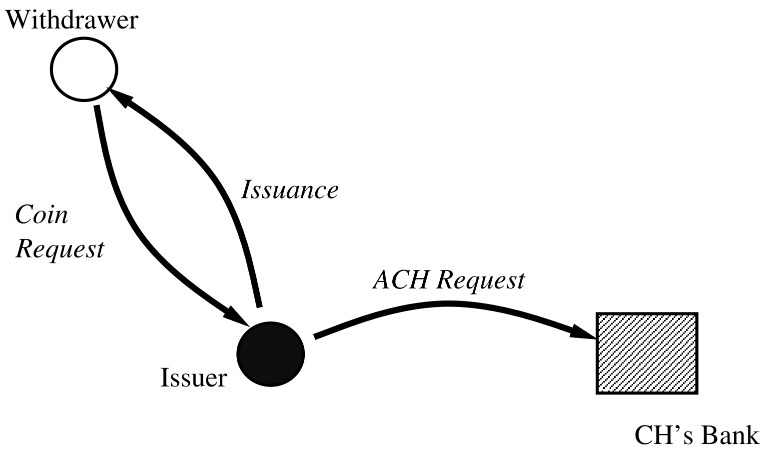


Figure 4: *The Coin Purchase process.*

the issuer by matching it with the value in his purse. He forms the session key $K = K_{CP} \oplus K_I$. Now he forms the indicated flow, which contains W-DESC and a signature, the whole encrypted under the shared session key K to ensure privacy of the bank coin purchase information. Note the nonce R_I is included in the signature to ensure freshness.

(2) Execution. The issuer now uses the coin purchase information and authorization provided by the customer to make the ACH transaction of coin purchase from the bank.

(2.1) WExecution. The coin purchaser uses K to decrypt the ciphertext and obtain W-DESC and the signature. He checks that the signature is valid, and stores it. Now he uses the information in W-DESC to make the ACH request. The issuer then waits a suitable period (which can range up to the order of days). If there is anything wrong, the bank sends a reject within this period; else the funds are in the issuer account. Now the issuer is ready to issue the coins.

(3) Issuance. The issuer forms e-coins $\text{Coin}_1, \ldots, \text{Coin}_n$ of the denominations requested in W-DESC. (These coins may have been created earlier and are archived, or may be formed at this time.) Then:

(3.1) Issuance1. The issuer encrypts the coins under the session key K to get EncC. This ciphertext is then authenticated, also under K, by computing $\mathbf{mac}_K(\text{ID}_I, \text{EncC})$. The ciphertext and the MAC are sent to the coin purchaser.

(3.2) Issuance2. The coin purchaser checks that the MAC is correct. (This means the coins are really from the issuer.) Then he decrypts the ciphertext to get the coins, which go into the purse. He now issues a final acknowledgment, consisting of the issuer nonce R_I MACed under the session key.

In the full paper we show how requirements **W1–3** are met. We now turn to the description of the Payment process.

3.2 Payment

The Payment process involves the payer (e.g., a customer), the payee (e.g., a merchant) and the issuer. The requirements are as follows:

P1– *Valid payments go through.* If the payer transfers a certain amount in valid coins, and if these coins are as yet unspent, then, after checking with the issuer, the payee accepts the payment. (He may end up with refreshed coins, or have redeemed them, or aggregated, as he wishes.)

P2– *Accepted payments are valid.* If after checking with the issuer a payee accepts a payment, then he knows that the refreshed coins he has obtained are valid. In particular, already-spent coins are detected: If a payer uses an already-spent coin then (by checking with the issuer) the payee will detect it, and the payee will not accept the payment.

- **Fields:**

W-DESC	*Coin Purchase description*– amount, denominations of desired coins, information and authorization required to make debit at bank.
R_X	Random challenge chosen by party X
K_X	Random number chosen by party X
K	$K_{CP} \oplus K_I$

- **Protocol Flows:**

$$\text{WRequest1} \quad : \quad CP \xrightarrow{\quad \mathbf{E}_I^*(\mathrm{ID}_{CP}, K_{CP}) \quad} I$$

$$\text{WRequest2} \quad : \quad CP \xleftarrow{\quad \mathbf{E}_{CP}^*(\mathrm{ID}_I, K_I, R_I) \quad} I$$

$$\text{WRequest3} \quad : \quad CP \xrightarrow{\quad \mathbf{e}_K(\,\text{W-DESC},\ \mathbf{S}_{CP}(\mathrm{ID}_I, R_I, \text{W-DESC})\,) \quad} I$$

$$\text{WExecution} \quad : \quad I \xrightarrow{\quad \text{ACH-Req} \quad} B$$

$$\text{Issuance1} \quad : \quad CP \xleftarrow{\quad \overbrace{\mathbf{e}_K(\mathrm{Coin}_1, \ldots, \mathrm{Coin}_n)}^{\mathrm{EncC}},\ \mathbf{mac}_K(\mathrm{ID}_I, \mathrm{EncC}) \quad} I$$

$$\text{Issuance2} \quad : \quad CP \xrightarrow{\quad \mathbf{mac}_K(\mathrm{ID}_{CP}, R_I) \quad} I$$

Figure 5: *Coin Purchase rotocol*

P3– *Payment is for the goods or services the parties have agreed on.* An adversary A cannot divert a payment by the payer to A's advantage, or even change the order description in "spoiler" ways. This is an optional requirement, which can be provided given an external certification authority.

P4– *Payer is informed of double spending.* In case the issuer detects double spending, the payer should be told his coins are bad, and be sure that the issuer thinks so.

We provide two basic kinds of payment protocols: payment with refresh (i.e., the payee obtains new coins) and payment with redemption (the payee is enrolled, and obtains real funds). In this abstract we only describe the former; payment with redemption has the same flavor. The same applies to payment with *aggregation*. The flows involved in payment with refresh are shown in Figure 6, and the protocol in Figure 7. The V-DESC field indicates which option is being used. In addition, it includes whatever information is needed for the option being used. For example, if it is refresh, the V-DESC field includes the type and number of the desired coins; if it is redemption, the V-DESC field includes the bank name, address and the account number.

The payment protocols have certain optional flows. They are indicated in square brackets, for example $[\mathbf{S}_{PY}(\mathcal{H}(\mathrm{Com}))]$ means providing this signature in the first flow is an option. The issue here is certificates. The basic protocol does not need a certification authority: it is enough that the issuer have the public keys of the participants. But for the extra functionality of order protection and receipt,

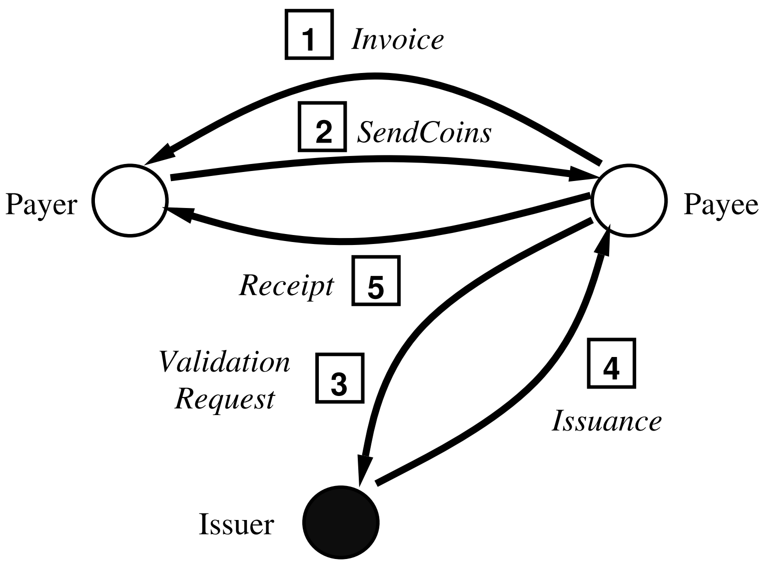


Figure 6: *Payment with Refresh: Flows*

an external certification authority is needed to provide the payer with the public key of the payee. We now describe how the flows are computed.

(1) Invoice. This transaction consists of a single flow in which the payee provides the transaction ID. The latter is a randomly chosen number which uniquely identifies the transaction. For confirmation of amount and order information, it is suggested that this be accompanied by a signature of the common information.

(2) SendCoins. The payer picks from his purse a collection of coins $\mathrm{Coin}_1, \ldots, \mathrm{Coin}_n$ whose total dollar value equals the amount to be paid. (If the purse happens to not currently hold this amount, but holds coins of total dollar value which is larger, the payer can go through a change transaction to get change, and then resume the payment. If the purse has insufficient funds, the payer will have to make a coin purchase, and, since this is a lengthy process, he will probably stop the payment here and re-start when he has the funds.) The coins are put in an envelope by encrypting them (and the identity of the payee) under the public key of the *issuer*. The ciphertext is transmitted to the payee.

(3) Validation Request. The payee cannot open the envelope; he never sees the coins. Instead, he forwards them to the issuer for validation, along with V-DESC which indicates whether he wants refresh, redemption or aggregation. The payee also includes in V-DESC the amount, to guard against the payer paying less than the agreed amount. This is done, for privacy, under cover of an encryption under the issuer's public key. Also in the scope of the encryption go the payer identity, the transaction id, and a number K_{PY}, chosen at random, which will be used to derive a session key.

At this point the process is different depending on whether we are doing refresh or redeem. We continue to describe the refresh case.

(3) Issuance. The issuer decrypts the ciphertext to obtain $\mathrm{ID}_{PY}, K_{PY}, \mathrm{TID}_{PY}, \text{V-DESC}, \mathrm{EncC}_1$. He then decrypts EncC_1 to get the coins which were sent by the payee. The validity of these coins is checked, as also the fact that the total value of these coins matches the amount claimed by the payee that is present in V-DESC. Now new coins are issued, of the type specified in V-DESC, via two flows:

(3.1) Issuance1. The issuer picks a number K_I at random and forms the session key $K = K_I \oplus K_{PY}$. The session key, together with ID_I, are encrypted under the public key of the payee, and the resulting ciphertext is transfered to the payee. Also the issuer encryptes the new coins under K; then MACs this ciphertext and some other stuff as shown. The second ciphertext and the MAC are also sent to the payee.

(3.2) Issuance2. The payee acknowledge having received the new coins by sending a message signed under the session key K.

(4) Receipt. The last flow is optional, and consists of a receipt, from payee to payer, that the payer's payment was accepted by the issuer.

We have omitted from the protocol picture the flows related to error conditions, such as the issuer informing the payer if his coins are bad, or the issuer informing both parties if the claimed and paid amounts do not match. The issuer would sign the bad coins and the error statement, and pass this to the payee, who in turn passes it to the payer.

Several spoiling attacks are possible. For example an attacker could flip some bits in the MAC in the Issuance1 flow, making the payee reject. In a seemingly more sophisticated attack, he can remove the ValidationReq flow sent by the payee and substitute a fake one which contains the same information except that the value of K_{PY} is different. (Note he is in possession of all information except K_{PY} so can indeed do this.) Then the payee will again reject the Issuance1 flow since he will recover the wrong session key. However, such attacks do not really help the attacker. These kinds of spoiling attacks are unpreventable, and handled by appropriate error handling and re-transmission.

4 Integrating Card Cash

4.1 Card-based systems

Typically, a card-based e-money system is an electronic payment system based on a tamper-proof device, with the properties of being pseudo-anonymous, offline, and non-circulating. The term non-circulating refers to the fact that the monetary value once paid cannot be reused offline (by the payee of the first transaction) and must be redeemed. Being an offline system, it requires tamper-proof smartcards holding monetary values (for making payments), as well as tamper-proof devices for accepting payments. The devices for accepting payments require even higher level of protection as they may contain certain global cryptographic keys.

The e-money system outlined in Sections 2 and 3 is a network-based, online payment system. In this section we show how that system can be combined with this card-based system to offer additional services and functionality, i.e., *VarietyCash*. For example, the combined system allows transfer of monetary value from one system to another and vice versa. The advantages of one system can then be used in the other. Fortunately, these transfer protocols can be built at a high level, on top of the two systems.

4.2 Cardcash

We assume the card-based system has the following components:

- Purse smartcard (smartcard for short) (*SM*),
- load/unload device,
- load/unload server (*LUS*),
- purchase device,
- purchase SAM (Security Access Module), and
- collection server.

In addition, there maybe one or more payment and clearing agencies (AG) which are not part of the system, but are needed for transfering the real money to electronic money and vice versa.

The cryptographic protocols used in these systems typically use symmetric keys (e.g., DES). The security is hierarchical in nature. In other words, there are devices which have the capability of generating their own keys, whereas devices higher up in the hierarchy can generate keys of devices immediately lower in rank using identification information of the lower ranked device. For example, a smartcard is lower ranked than a Purchase Security Access Module(PSAM). The PSAM has a global key K, whereas the purse just has a derived key $f(K, ID)$, where f is a predefined one-way function, and ID is the identification number of the smartcard. The PSAM can dynamically generate the derived key once it has the identification of the smartcard.

In general, the keys are "segmented" for further security. In such a scheme there maybe many global keys for a particular device class. In this abstract we will simplify the presentation by assuming no segmentation of keys.

We now turn to the description of some of the "native" card-based system protocols, and the transfer protocol between systems.

4.3 Load protocol

There are many modes of loading value onto the card, such as off-line load using Load Security Access Modules (LSAM), on-line account-based loads, and on-line non-account-based loads. In this abstract we will only cover on-line account-based loads. In the on-line account-based load, there is a payment agency involved with which the card holder has an account (or registration). The load is mediated by the payment agency, and the account is debited.

Informally, when a user with a card wants to load value onto his card (using an on-line account based method), he communicates with his payment agency using the load/unload device. Once a load of a particular value is requested by the card, the payment agency (which acts as a trusted party), instructs the load/unload server to load the monetary value onto the card (while guaranteeing payment). All further messages between the load/unload server and card go via the payment agency (or the client), and the payment agency and the load/unload server commit the transaction based on acknowledgments from the two end parties. The load/unload server logs the transaction, and clears the payment with the payment agency via a clearing system.

A sketch of the protocol is shown in Figure 8. There are three parties invloved: the load/unload server (LUS), the card (SM), and the payment agency (AG). The first step of the Load protocol (not shown) is itself a high level protocol between an account holder and a payment agency. This protocol ensures a monetary value transfer from the smart-card holder to the payment agency. For example, if the payment agency is your bank, this payment protocol is an electronic authorization to debit your account. As to how this protocol is defined is independent of the card-based system, except for the fact that it should be able to associate a SM ID, a unique transaction ID for this SM ID, and an amount with this transaction to be used in the composite transaction.

All the messages between SM and LUS go via the payment agency (or the load/unload client in the payment agency). However, note that in this protocol there is no way for the server LUS to prove to the payment agency that the load into the card actually happened, since the key is symmetric. In other words the signature (i.e., signed ack) in the protocol—it could have been generated by the server itself. Thus, the server is assumed to be a trusted party.

The Unload protocol is symmetrically opposite to the one of Figure 8.

4.4 Transfer protocols and complex payment transactions

We now briefly describe the transfer protocols between card- and software-based systems. Effectively, this is a composition of an online protocol between a user and a load/unload server, followed by an online protocol between the same user and the issuer of Section 2. Specifically, an account-based Unload protocol followed by a Coin Purchase protocol similar to that of Figure 5. The issuer acts as the payment agency in the Unload protocol. Call this combined protocol Cardcash-to-Softcash.

The reverse direction, software cash to card cash, is composed of the Redeem protocol between the user and issuer, followed by the Load protocol of Figure 8 between the same user and a load/unload server. Again, the issuer acts as the payment agency in of the Load protocol. Call this protocol Softcash-to-Cardcash.

In the same way, transactions between users from the same or different payment "media" are also enabled. For example, the payment from one card user to another is achieved as follows. This type of transfer can be accomplished by any intermediary, in particular, the issuer of Section 2:

1. The payer runs Cardcash-to-Softcash;
2. the payer, payee and issuer run the Payment with Redeem protocol (Section 3);
3. the payee runs Softcash-to-Cardcash.

Note that in the card-based system a clearing system is used for transfering money back and forth between the load/unload servers and the payment agencies. Thus, there is not much of an overhead in running these composed protocols. At the end of the three steps above the payment agency (the issuer) and the load/unload server are even.

Similarly, card payers can pay software payees with not much of an overhead, since a clearing system is involved. Here, the unload server owes money to the issuer. In the reverse direction, the issuer owes money to the load server.

Acknowledgements

We are thankful to the anonymous referees for the many comments regarding comparison with other payment systems, as well as spoiling attacks against some of the protocols.

References

[1] M. Abdalla, M. Bellare and P. Rogaway. DHAES: An encryption scheme based on the Diffie-Hellman problem. Manuscript.

[2] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Relations among notions of security for public-key encryption schemes. *Advances in Cryptology – Crypto 98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.

[3] 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. *Proc. First Usenix Workshop on Electronic Commerce*, pp. 89-106, New York, June 1995.

[4] M. Bellare, J. Garay and T. Rabin. Fast Batch Verification for Modular Exponentiation and Digital Signatures. *Advances in Cryptology – Eurocrypt* 98 *Proceedings*, Lecture Notes in Computer Science Vol. 1403, K. Nyberg ed., Springer-Verlag, 1998.

[5] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. *Advances in Cryptology – Eurocrypt* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1994.

[6] D. Bleichenbacher. A chosen ciphertext attack against protocols based on the RSA encryption standard PKCS #1. *Advances in Cryptology – Crypto* 98 *Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.

[7] J.-P. Boly *et al.* The ESPRIT Project CAFE - High Security Digital Payment Systems. *ESORICS '94*, LNCS 875, Springer-Verlag, Berlin 1994, 217-230. `<http://www.informatik.uni-hildesheim.de/FB4/Projekte/sirene/projects/cafe/index.html>`.

[8] D. Chaum. Blind signatures for untraceable payments. *Advances in Cryptology - Crypto 82 Proceedings*, D. Chaum, R. Rivest & A. Sherman eds., Plenum Press, New York, pp. 199-203.

[9] CyberCash. The CyberCash(tm) System - How it Works. `<http://www.cybercash.com/cybercash/cyber2.html>`.

[10] DigiCash. About ecash. `<http://www.digicash.com/ecash/ecash-home.html>`.

[11] D. Dolev, C. Dwork, and M. Naor, Nonmalleable cryptography. *Proceedings of the* 23rd *Annual Symposium on the Theory of Computing*, ACM, 1991.

[12] W3C Joint Electronic Payments Initiative (JEPI). `<http://www13.w3.org/ECommerce/Overview-JEPI.html>`.

[13] M. Jakobsson and M. Yung. Revokable and versatile electronic money. *Proceedings of the Third Annual Conference on Computer and Communications Security*, ACM, 1996.

[14] Mondex. `<http://www.mondex.com>`.

[15] G. Medvinsky. NetCash: A framework for electronic currency. Ph. D thesis, Dept. of Computer Science, USC, 1997.

[16] G. Medvinsky, C. Neuman. NetCash: A design for practical electronic currency on the Internet. *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993. `<ftp://prospero.isi.edu/pub/papers/security/netcheque-requirements-compcon95.ps.Z>`.

[17] C. Neuman. Proxy-based authorization and accounting for distributed systems. *Proc. 13th International Conf. on Distributed Computing Systems*, pp. 283-291, May 1993.

[18] C. Neuman, G. Medvinsky. Requirements for Network Payment: The NetCheque Perspective. *IEEE COMPCON*, March 95. `<ftp://prospero.isi.edu/pub/papers/security/netcheque-requirements-compcon95.ps.Z>`.

[19] M. Sirbu, J. D. Tygar. NetBill: An Internet Commerce System. *IEEE COMPCON, March 95.* `<http://www.ini.cmu.edu/netbill/CompCon.html>`.

[20] Secure Electronic Marketplace for Europe (SEMPER). `<http://www.semper.org/>`.

[21] SET. `<http://www.mastercard.com/set/>`.

[22] D. Tygar. Atomicity in electronic commerce. Invited talk and paper, *Proc. Fifteenth Annual ACM Symposium on Principles of Distributed Computing,*, pp. 8-26, Philadelphia, May 1996.

[23] M. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *J. of Computer and System Sciences* 22, pp. 265–279, 1981.

- **Fields:**

P-DESC	*Purchase description*– amount, description of goods, payment method or mechanism. Assumed part of starting information of payer and payee.
TID_{PY}	*Transaction ID*– a number generated by the payee which is uniquely associated to this transaction
Com	P-DESC, $\mathrm{TID}_{PY}, \mathrm{ID}_{PA}, \mathrm{ID}_{PY}, \mathrm{ID}_I$
V-DESC	Verification and execution request text. Indicates one of three options (refresh, immediate redeem, or aggregate) and provides corresponding data. Includes amount.
K_X	Random number chosen by party X
K	$K_{PY} \oplus K_I$

- **Protocol Flows:**

$$\text{Invoice} \quad : \quad PA \xleftarrow{\mathrm{ID}_{PY},\ \mathrm{TID}_{PY},\ [\mathbf{S}_{PY}(\mathcal{H}(\mathrm{Com}))]} PY$$

$$\text{SendCoins} \quad : \quad PA \xrightarrow{\overbrace{\mathbf{E}_I^*(PY, \mathrm{TID}_{PY}, \mathrm{Coin}_1, \ldots, \mathrm{Coin}_n)}^{\mathrm{EncC}_1}} PY$$

$$\text{ValidationReq} \quad : \quad PY \xrightarrow{\mathbf{E}_I^*(\ \mathrm{ID}_{PY},\ K_{PY},\ \mathrm{TID}_{PY},\ \text{V-DESC},\ \mathrm{EncC}_1\ )} I$$

$$\text{Issuance1} \quad : \quad PY \xleftarrow{\mathbf{E}_{PY}^*(\mathrm{ID}_I, K_I),\ \overbrace{\mathbf{e}_K(\mathrm{Coin}'_1, \ldots, \mathrm{Coin}'_{n'})}^{\mathrm{EncC}_2},\ \mathbf{mac}_K(\mathrm{ID}_I, \mathrm{TID}_{PY}, \mathrm{EncC}_2)} I$$

$$\text{Issuance2} \quad : \quad PY \xrightarrow{\mathbf{mac}_K(\mathrm{ID}_{PY}, \mathrm{TID}_{PY})} I$$

$$\text{Receipt} \quad : \quad PY \xrightarrow{[\,\mathbf{S}_{PY}(\mathrm{ID}_{PA}, \mathcal{H}(\mathrm{Com}), \mathrm{EncC}_1)\,]} PA$$

Figure 7: *Payment with Refresh protocol.*

- **Fields:**

SM_{ID}	Smartcard's unique ID
TID	A unique transaction ID for the smartcard
$Amount$	Monetary value to be loaded
$\mathbf{e}_{Load-key(SM_{ID})}$	Encrypted under the derived Load key of the SM, stored as it is on the SM , and computable by the load server using its own global key and SM_{ID}
R_X	Random challenge chosen by party X
$\mathcal{H}$	A standard one-way hash function

- **Protocol Flows:**

$$\begin{array}{llcll}
\mathsf{LInitialize1} & : & LUS & \xleftarrow{\quad TID, SM_{ID}, Amount \quad} & AG \\
\mathsf{LInitialize2} & : & SM & \xleftarrow{\quad \mathbf{e}_{Load-key(SM_{ID})}(TID, R_{LUS}), TID \quad} & LUS \\
\mathsf{LDebit} & : & SM & \xrightarrow{\quad \mathbf{e}_{Load-key(SM_{ID})}(\mathcal{H}(TID, R_{LUS}), R_{SM}), TID \quad} & LUS \\
\mathsf{LCredit} & : & SM & \xleftarrow{\quad \mathbf{e}_{Load-key(SM_{ID})}(\mathcal{H}(R_{SM}), Amount), TID \quad} & LUS \\
\mathsf{LAck1} & : & SM & \xrightarrow{\quad \mathbf{e}_{Load-key(SM_{ID})}(\mathcal{H}(TID, R_{SM} \oplus R_{LUS})), TID \quad} & LUS \\
\mathsf{LAck2} & : & LUS & \xrightarrow{\quad TID, ``OK'' \quad} & AG
\end{array}$$

Figure 8: *Load protocol*