

Digital Money

Edition 1.0

1995-06-07

Lars Johansson

AU-System

Contents

1. Abstract	1
2. Introduction	2
3. Public-Key Cryptography	3
3.1. Encryption	3
3.2. Digital Signatures	4
4. Overview of Payment Models	5
4.1. The Credit Card Model	7
4.2. The Electronic Purse Model	8
4.3. The Cheque Model	9
4.4. The Cash Model	10
5. Preliminaries	11
5.1. Algorithms and Complexity Theory	11
5.2. Finite Groups and Number Theory	12
5.2.1. Discrete Logarithms	13
5.2.2. Roots in RSA Groups	14
6. Cryptographic Primitives Based on RSA	15
6.1. RSA Encryption	15
6.2. Digital Signatures with RSA	16
6.3. Public-Key Certificates Based on RSA	17
7. David Chaum's ecash	19
7.1. Blind RSA Signatures	19
7.2. Withdrawal of \$1 in ecash	21
7.3. The Payment and Insertion Protocol	22
7.4. Additional Denominations	23
7.5. Withdrawal of two \$15 bills in ecash	25
7.6. Cookie Jar Change	26
7.7. Declared Note Value	28
7.8. Distributed Change	29
7.9. Hidden Note Value	30
7.10. Related Work	31
8. Cryptographic Primitives Based on D-L	32
8.1. Diffie-Hellman Key Exchange	32
8.2. D-H Authentication	33
8.3. Zero-Knowledge Proofs of Knowledge	34
8.4. Schnorr's Authentication Protocol	35
8.5. Schnorr Signatures	36
8.6. Secret-Key Certificates	38
8.7. Blind Schnorr Signatures	40
8.8. Restrictive Blinding of Secret-Key Certificates	42
9. The Cash Systems of Stefan Brands	44
9.1. The Representation Problem	44
9.2. Protection Against Double Spending	45
9.3. Proving Knowledge of a Representation	46
9.4. Off-Line Cash Without Observer	48
9.4.1. The Withdrawal Protocol	48

9.4.2. The Payment Protocol	50
9.4.3. The Deposit Protocol	51
9.5. Brands Off-Line Cash With Observers	52
9.5.1. The Withdrawal Protocol	53
9.5.2. The Payment Protocol	55
9.5.3. The Deposit Protocol	56
10. Acknowledgements _____	57
11. References: _____	58

Figures

Error! No table of contents entries found.

1. Abstract

This thesis gives a compact overview and classification of digital payment systems that involve public-key cryptography. It also includes a short introduction of that topic to the not so familiar reader. The main part of the thesis is devoted to a description of two proposed systems of digital transactions that share the property of being anonymous, i.e. payments can be made without revealing the identity of the customer. The two examined systems are:

ecash

CAFE

ecash is a system that is proposed by the Dutch company DigiCash bv and it is based on scientific papers by David Chaum. CAFE is a European project that utilises a protocol that is due to Stefan Brands at CWI in Amsterdam, The Netherlands. The major difference between ecash and CAFE is that CAFE uses smart cards and is a so-called off-line system whereas ecash is entirely based on software and is on-line.

2. Introduction

The purpose of this thesis is to give a short overview of proposed payment systems for the Internet and for electronic purse applications using smart card technology. There is a wide range of systems, both in literature and practice, so this paper does by no means aim at being complete. The focus is on systems that involve public-key cryptography.

There may also, admittedly, be a slight emphasis on anonymous payment systems, but this is mostly due to the overweight of such proposed protocols in the cryptographic literature. A possible reason for that would be that privacy-protecting payment systems with multi-party security is tremendously more difficult to design and henceforth a much greater challenge for the research. The author of this paper does not, however, take any political standpoint on this matter.

A brief evaluation of the efficiency of the presented payment protocols will also be given. Anonymous systems does in general suffer in terms of efficiency. A great deal of progress in designing efficient privacy-protecting systems have nevertheless been achieved in recent years and hopefully this paper will provide a clear explanation of some of these rather complex systems.

Lately there's been great many security schemes proposed for the Internet (see [45], [48] and [63]) which, although not intended to be used for electronic commerce in the first place, may seem appropriate as basic building blocks of such systems. An even greater flood of specific payment models for the Internet have been proposed (see [3], [4], [9], [25], [28], [32], [34], [40], [52], [53], [54], [57], [58], [68] and [70]). Obviously all these systems could not be evaluated or presented in this paper but a good overview can be found in [50].

There is also a parallel evolution in designing smart card systems that will be a replacement of today's cash, a so-called electronic purse (see [5], [24], [33], [55] and [65]). Those models could possibly also be used in payments on the Internet and, if special care is taken, they may offer the best security.

The disposition of this thesis is such that public-key cryptography is first given a brief summary and with this basic knowledge, especially about digital signatures, some different payment models are described. These parts are intended to be readable for anyone, without any further background in discrete mathematics or, in particular, cryptography. The rest of the report is devoted to describing two anonymous payment systems, David Chaum's ecash (see [32] and especially the paper [16]) and Stefan Brands' off-line system (in [12]) that forms the basic foundation of CAFE (see [5]). Those descriptions does however require quite some mathematical embodiments. Therefore some introductory chapters on complexity theory and modular arithmetic have been included.

3. Public-Key Cryptography

The concept of public-key crypto-systems was introduced by Whitfield Diffie and Martin E. Hellman in [31]. It is also called asymmetric cryptography, in contrast to symmetric secret-key crypto-systems such as DES (Data Encryption Standard, see [56]).

It is based on the assumption that there exists *one-way functions* that are simple to compute in one direction but impossible to invert without possession of some secret information, the secret key. The idea is very simple. Every person, i , has two (personal) functions, $E_i(\cdot)$ and $D_i(\cdot)$, which are each others inverses so that:

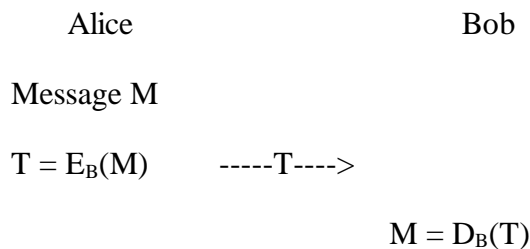
$$D_i(E_i(x)) = x = E_i(D_i(x)) \quad , \text{ for } \forall x \in \mathbf{Z}_n.$$

The function $E_i(\cdot)$ is made publicly known to everyone who wishes to communicate privately with i . $D_i(\cdot)$, on the other hand stays secret, known only to person i . There are two major uses of this setting, namely encryption and digital signatures. For plain cryptography, $E_i(\cdot)$ is the encryption function and $D_i(\cdot)$ is used for decryption. Alternatively, if $D_i(\cdot)$ is used to encrypt a message, a digital signature by person i is created that can only be verified with the corresponding public verification function $E_i(\cdot)$.

3.1. Encryption

Now, suppose the cryptographers favourite characters, Alice and Bob, have their own pair of secret and public functions, such that $E_A(\cdot)$ denotes Alice's public function and $D_A(\cdot)$ denotes her secret function. Similarly, let Bobs corresponding functions be denoted by $E_B(\cdot)$ and $D_B(\cdot)$ respectively.

If Alice wants to send an encrypted message, M , to Bob, this is the way it would work:

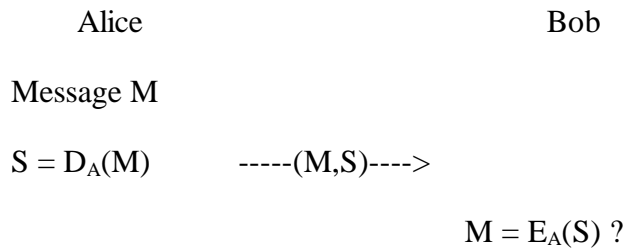


1. Alice encrypts the message, M , by using Bob's public encryption function $E_B(\cdot)$ and then she sends $T = E_B(M)$ to him over an insecure channel (such as the Internet).
2. Bob then decrypts Alice's message with his private function $D_B(\cdot)$ such that her message is recovered as $M = D_B(T) = D_B(E_B(M))$

Since it is assumed that only Bob knows $D_B(\cdot)$, Alice can be assured that only he can read her message and thus they can have a conversation in private. Of course, Bob will use Alice's encryption function $E_A(\cdot)$ when he wishes to transmit a message to her.

3.2. Digital Signatures

On the other hand, if Alice wants to sign a message M , such that Bob (or anyone) can verify that it is Alice and nobody else who have put their signature on M , this is how it can be done:

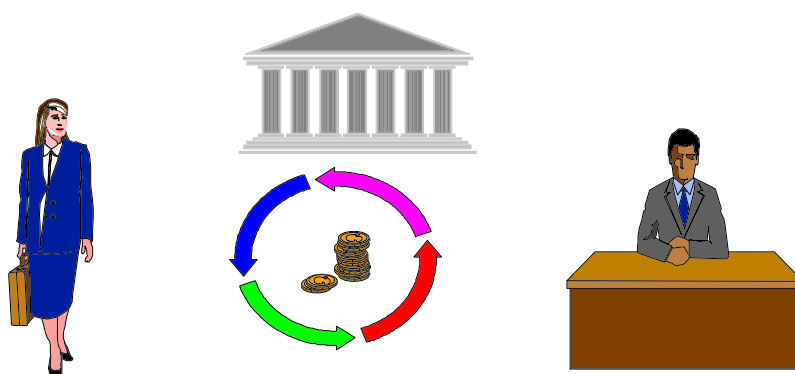


1. Alice encrypts the message with her secret function $D_A(\cdot)$, such that the signature becomes $S = D_A(M)$. Then she sends the signature, S , to Bob (together with M , unless Bob already knows it).
2. Now Bob can verify Alice's signature on M by decrypting S with Alice's public function $E_A(\cdot)$ and check that the following condition holds: $M = E_A(S) = E_A(D_A(M)) ?$

In order to understand how these functions can be constructed it is necessary to have some background knowledge of number theory, finite groups and complexity theory. Therefore these topics will be given a short coverage in a forthcoming section. First though an overview of some generic payment models will be given, which can use almost any kind of public-key cryptographic algorithm.

4. Overview of Payment Models

The simplest model of an economical system (excluding barter) only involves three participants, namely the financial institution, a customer and a service provider. The financial institution issues the medium of exchange (the money) to the customer who uses it as a payment to the service provider in exchange for some service or product. To make the following descriptions more concrete, the financial institution will simply be called the Bank, the customer will be named Alice and the service provider's name will be Bob.



Although the assumption of only having one Bank may seem a bit unrealistic, it isn't in general a major drawback since usually any Bank can serve as issuer or acquirer of money. The Banks internal clearing process facilitates the development of models with several different Banks which accepts payment means from each other. This paper will for simplicity not deal with this matter.

The essential thing to take notice about is that the system should be open. That means that Alice and Bob are not assumed be aware of each other before they start doing business, i.e. it should be possible to do trading spontaneously. Good examples of open systems is the Internet and the "real-world". Closed systems on the other hand would be the traditional telecom networks where there are central telephone exchanges which handles the debiting of all its subscribers.

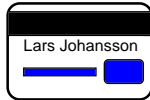
If Alice knew that she would make a purchase at Bob's shop beforehand, then she could buy a credit note, a coupon ticket or a token by Bob and later use these in the actual transaction. That is a pre-paid system and in such there really is no need for a Bank. It won't be further considered here.

It will also be assumed that the money is only used for one single payment before it is returned to the Bank. This is because a majority of the proposed payment systems in literature and practise share this property. It isn't such a big disadvantage as it first may seem though.

This paper won't either be concerned with the issue of having several different currencies. It doesn't have to be difficult to incorporate such a facility but it was excluded from the scope of this thesis mainly due to time limitations. It is also conceivable that the conversions may be performed by the Bank during the withdrawal. Having one single global currency on the Internet (or in the real world), although desirable, may unfortunately seem a bit utopian.

There are several ways to categorise digital payment systems. One example is how the verification of the payment is performed. Most systems for the Internet utilises on-line verification which means that the Bank has to be contacted for every transaction. Systems with off-line verification is in general much more efficient but also harder to design. Of the two systems that is examined in closer detail in this paper, ecash and Brands systems, the first one is an example of an on-line system and the other is off-line.

An excellent overview of payment models on the Internet can be found in [50] and [17] is also a good introduction to cryptographic payment schemes. [47] gives a nice overview in Swedish.



4.1. The Credit Card Model

A vast majority of the currently proposed systems for the Internet uses plain credit card transactions by sending the credit card number in an encrypted form to the service provider (e.g. [28]). There are some slight variations, some which provides anonymity (see [52]) and some that never reveals the credit card number to anyone but to the central Bank (see [70]).

Those systems have the advantage that they are already incorporated into the existing Banking systems and does not even require public-key cryptography (a secret-key algorithm together with a method of key exchange, such as Diffie-Hellman, might as well be used).

The major drawback is that only on-line payments are possible (i.e. the Bank has to verify the validity of the credit card number before the service provider can accept it). This could negatively affect the efficiency of the transactions. Specifically, if this scheme is used in a large scale on the Internet there may be too much traffic to the Bank's server for it to be practical. The Internet will most likely only involve low-value payments and for such, the communication costs might get larger than the actual value of the transaction.

The fact that a credit card number in a way identifies the customer and thus prohibits anonymous payments might not be such an alarming fact to most people, since we use these payments schemes in our daily life already. What's worse (in view of today's public-key cryptography with multi-party security) is that the customer can't disapprove to have *not* bought something from a service provider who tries to withdraw a larger amount than the customer have admitted to (remember that the service provider and the customer never meet face to face). The whole system is based on mutual trust between its participants.

Public-key cryptography, and in particular digital signatures, offers a way of legally signing documents that can be used in a trial if a dispute would arise (see [51]). This leads to payment schemes which can digitally sign every transaction (see the chapter on The Cheque Model).



4.2. The Electronic Purse Model

A model which is cryptographically similar to the credit card model is what will be called the electronic purse model. The idea of electronic purses comes from the early seventies and is thought to be a replacement of ordinary purses or wallets, which means that they will contain electronic information that can be used instead of cash (i.e. coins and bills). Traditional medias such as magnetic cards are not enough for this purpose. Active tamper-resistant hardware such as a smart card would necessarily have to be used.

Smart cards are already in use in many countries (e.g. Sweden) in payphones. Those cards are however too easily tampered and can't normally be reloaded with new money. An electronic purse should be able to be loaded in a withdrawal process similar to withdrawing cash from today's automatic teller machines (ATM). It can thereafter be used to pay for goods and services in shops and supermarkets.

The European Committee for Standardisation (CEN) is working on a standard for electronic purses (see [24]) that is currently only in a draft version. It doesn't specify what cryptographic algorithms that is to be used but the system requires tamper-resistant modules at both the customer's purse and the merchant's terminal. The payment protocol is designed to guarantee that the customer's balance is decreased by the same amount that the merchant's balance is increased. That's why it can be said to be based on mutual trust between the purse and the terminal. Cryptographically there's no big difference with this approach and the credit card model since the only purpose of the encryption is to prohibit eavesdropping, such as a man-in-the-middle attack.

Electronic purses, similar to this model have been developed independently by e.g. Danmont, Proton (by Banksys), Express (by Europay) and Mondex (see [55]). The Swedish transport card project (see [65]) also specifies an architecture of an electronic purse which, as opposed to CEN, uses digitally signed transactions. This system will be given a better coverage in the next chapter. The European ESPRIT consortium have developed an electronic wallet in the CAFE-project (Conditional Access For Europe, see [5]) which has the unique characteristic that it offers protection of the users anonymity. The CAFE wallet is also using digital signatures, although blinded (to be explained later). It utilises an improved version of a protocol by Stefan Brands which will be covered in this report.

Electronic purses were not originally designed to be used on the Internet but may very well be shown to offer appropriate security for such payments as well. Payments on open networks (such as the Internet) puts special demands on the security issues since the customer and the merchant (Alice and Bob) never meet face to face.



4.3. The Cheque Model

As been seen in the previous chapter, digital signatures has all the characteristics of written signatures, including legality (see also [51]), and can be constructed with public-key cryptography. This tool makes a payment system similar to cheque payments conceivable.

The Swedish transport card project, run by KontoCentralen (KC), have described an architecture for such a system (see [65]). It is thought to be used with smart card electronic purses but could possibly be used on the Internet as well.

In this model, Alice receives a specification file by Bob which uniquely identifies him and also includes some other information such as time, a transaction number, the payment terminal identity and Bob's acquiring Bank. Alice extends this file with more information about her own (and her purse's) identity and then signs it with her secret key.

Alice's signature gives legal proof to Bob that she has accepted the transaction. Now Bob can send the signature to the Bank and require to get credited for it. The crediting and debiting is usually thought to simply be increasing or decreasing the balance of Bob's or Alice's accounts.

The KC-model uses a tamper-resistant electronic purse that holds a counter for the balance of the purse. It is hence aimed at being a replacement for cash, i.e. coins and bills. The purse will only sign amounts less than the current balance. It is thus crucial that the balance may never be changed directly by Alice, nor will she be able to find out the secret key, in which case she could make illegal payments of amounts greater than she has due.

The same architecture could however also be used for plain software based cheque payments. In such a model Alice never has any money in her pocket. It always stays at the Bank. When Alice wants to pay Bob, she simply sends him a signature that the Bank may debit her account and credit Bob's. Alternatively, Alice may send this signed payment order directly to her own Bank, which will credit Bob and notify him afterwards (preferably with a signed receipt).

The major disadvantage with the cheque model, unless tamper-resistant hardware is incorporated, is that all the payments has to be performed on-line if it should be really safe. The KC-model is however designed to be an off-line system by using electronic purses and certificates (to be explained later). The purpose of the certificate is to guarantee Bob that the Bank have granted permission to Alice to sign cheques as valid payments. Just as merchant's today only accept written signatures on special pieces of paper (i.e. cheques) that the Bank have issued, Bob will also only accept a digital signature that have been certified by the Bank.

The cheque model can also be described as being a signature generating system. That alludes to the fact that it is the customer, not the Bank, who generates the signatures. The next model that will be described is a signature transporting system, where the user only transfers the Bank's signatures (the cash) to the merchant.



4.4. The Cash Model

The whole idea of digital cash is that, just like today's Banks authenticates coins and bills by putting a signature on them (the print), having electronic Banks authenticate strings with digital signatures in replacement of physical money. A digital bank-note can be thought of as a pair of strings, where the first element represents a serial number and the other represents the Bank's digital signature on the serial number.

In the same way as physical cash is authenticated by itself, thanks to the print, so is digital cash by the digital signature. Ideally, this would make off-line payments possible. There is however one major difference between paper and digital cash and that is the fact that any digital representation may easily be copied whereas copying physical cash requires quite a skilful counterfeiter.

If the crucial point in the cheque model was how to make it impossible to sign amounts that overdraw the user's account, the difficult thing to prohibit with digital cash is double spending of the same coin or bill at several merchants. Just as with the cheque model, there are two different methods of attacking this problem, namely by either using tamper-resistant hardware or by making it an on-line system. For instance, DigiCash's payments system for the Internet, ecash, utilises on-line verification to prevent double spending (to be described later).

When smart cards are used, the cheque and the cash model are essentially equivalent. Just as the balance and the secret key have to be protected in the cheque model, so is the Bank's signatures in the cash model. The major difference though is that a secret key and a corresponding certificate in the cheque model can be used in innumerable transactions, while the cash model preferably only allows a Bank signature to be used once.

There also exists hybrids of these systems, such as CAFE which uses disposable keys that may not be used for more than one single payment. CAFE does however require a balance counter. DigiCash have also recently announced an electronic purse (the blue-chip, see [33]) that is specially designed to be able to store a large number of signatures in an extremely compact form which enables numerous payments between the refills.

In its cleanest version, the cash model doesn't require any balance counter at all. The value of the coins or bills is inherent in the signature itself. Different denominations can be implemented by having the Bank sign the digital coin with different secret keys. That is the technique that is used in ecash. Stefan Brands, who is the author of the protocol that is used in CAFE, have also outlined how such a characteristic can be achieved (see [6]). Why this is seldomly used in practice is due to the huge storage requirements.

Both ecash and CAFE offers anonymous payments just as ordinary cash. The rest of this paper will be devoted to describing these systems but first there is a need to look into the mathematics in closer detail.

5. Preliminaries

5.1. Algorithms and Complexity Theory

All current, “practical” use of public-key cryptography is based on assumptions from the complexity theory. For a more formal and extensive introduction to this subject, see [30] or [26]. Only a brief summary of the basic assumptions that is of interest to modern cryptography will be given here, such as the intractability of factoring or computing discrete logarithms.

The basic concept in public-key cryptography is that of one-way functions. Such functions are believed to be simple to compute in one direction but impossible to invert without knowledge of a secret. Although it is not known today whether or not such functions exist (i.e. no proof of their existence have been given), many aspiring functions have been suggested. In practice, however, it is often enough to use functions that are believed to be *hard* to invert. *Hard* in this context means that computing the inverse is infeasible with all currently known methods.

Algorithms are usually classified in terms of their running time. By viewing the running time of an algorithm as a function of the size of its input one gets a simple way of comparing the efficiency of different algorithms that is independent of the task they perform. For example, a factoring algorithm takes as input a positive integer, with any number of bits, and returns its prime factors. In this context the actual result is of no interest, but the time it takes to find it. Neither the input integer is of any interest, but the number of bits needed to represent it (actually, the logarithm of the input, taken with base 2).

If the running time can be expressed as a linear function of the number of bits of the input, then the algorithm is said to linear. This set of algorithms is denoted by $\Theta(|n|)$, where $|n|$ is the magnitude (number of bits) of the input integer n ($\approx 2^{|n|}$). More generally, if the running time is polynomial in the size of the input, then the algorithm is also said to be polynomial and then it belongs to $\Theta(|n|^p)$, where p is the order of the polynomial. Algorithms of polynomial running time are usually considered “feasible” in cryptographic applications.

Infeasible are those algorithms of exponential running time, taken from the set $\Theta(e^{|n|})$. Notice that it isn't the absolute running time that is of interest, but the growth of the *running time function* in terms of the size of the input, $|n|$. A slight increase in the input size (i.e. additional input bits) causes the running time of an exponential algorithm to grow very drastically, whereas the performance of a polynomial algorithm is only slightly worsened.

Factoring integers or computing discrete logarithms are examples of problems that no known algorithm can solve in polynomial time. The best algorithms known to date are “sub-exponential”, meaning that they are only slightly faster than plain exponential. The fact that no polynomial-time factoring algorithm (or one that computes discrete logarithms) have been found is no valid proof that such algorithms doesn't exist. It may be possible that factoring or computation of discrete logarithms becomes feasible sometime in the future, and hence all practical crypto-systems presented here will break down.

Before we look at how one-way functions based on factoring and discrete logarithms can be constructed we need to know a little bit of number theory.

5.2. Finite Groups and Number Theory

The scope of this chapter on number theory and group theory will be limited to what's needed for the rest of the paper. Understanding this text is significantly easier if the reader is familiar with discrete mathematics, at least at undergraduate level (such as [43]). For a more concise introduction to abstract algebra (and group theory) see [2]. There is also a good chapter on number theory and number theoretic algorithms in [26].

The formal definition of a group is that it should be a set, \mathbf{G} , (e.g. the set of integers, \mathbf{Z}) together with an operation, $*$, (usually called multiplication) which should be closed in \mathbf{G} . That means that for any $x, y \in \mathbf{G}$, their combination, $x*y$, must also lie in \mathbf{G} . The operand, $*$, would also have to satisfy three axioms:

1. (Associativity). For all $x, y, z \in \mathbf{G}$, it follows that $(x*y)*z = x*(y*z)$.
2. (Identity). There exists an element $e \in \mathbf{G}$ such that $x*e = x = e*x$, for all $x \in \mathbf{G}$.
3. (Inverse). Each $x \in \mathbf{G}$ has a (unique) inverse $x^{-1} \in \mathbf{G}$ such that $x^{-1}*x = e = x*x^{-1}$.

The most publicly well-known group is the integers under the addition operator, $+$. The set of integers is commonly denoted by:

$$\mathbf{Z} = \{ \dots, -3, -2, -1, 0, 1, 2, 3, \dots \}$$

The associativity law is trivially verified and the identity element is simply 0. The inverse of an integer x is $-x$, so that $x+(-x) = 0 = (-x)+x$ becomes a true statement.

Since computers are bad at using infinite sets, a common approximation of the (positive) integers is:

$$\mathbf{Z}_n = \{ 0, 1, 2, 3, \dots, n-1 \}$$

for some large number n . In abstract algebra, \mathbf{Z}_n is also an example of a finite group under addition modulo n . That means that all sums are reduced by multiples of n , such that only the remainder after a (integer) division by n is left. In this way, \mathbf{Z}_n becomes closed under the addition operator (mod n).

A more interesting group for cryptographic applications is:

$$\mathbf{Z}_n^* = \{ x \in \mathbf{Z}_n : \gcd(x,n)=1 \}$$

under multiplication mod n . \mathbf{Z}_n^* consists of all integers from \mathbf{Z}_n which are relatively prime to n , i.e. those that don't share any common divisor with n (other than 1). In the case that n is a product of two (equally) large primes ($n = p*q$), then \mathbf{Z}_n^* is said to be an RSA-group after the popular and well-known RSA crypto-system (see [64]).

5.2.1. Discrete Logarithms

Other common crypto-systems alternatively use the group \mathbf{Z}_p^* , where p is a large prime such that $p-1$ doesn't have any small factors. Because p is prime, all numbers in \mathbf{Z}_p , except 0, will be relatively prime to p and hence:

$$\mathbf{Z}_p^* = \{ 1, 2, 3, \dots, p-1 \}$$

Let $g \in \mathbf{Z}_p^*$. Fermat's little theorem then states that:

$$g^{p-1} \equiv 1 \pmod{p}, \text{ for all } g \text{ which are relatively prime to } p.$$

This means that the order of g is at most $p-1$ (can also be a factor of $p-1$). In the case that the order of g is $p-1$ (denoted by $O(g) = p-1$), then g becomes a *generator* of \mathbf{Z}_p^* . That means that all elements in \mathbf{Z}_p^* can be written as $g^x \pmod{p}$, for some integer x . Thus, \mathbf{Z}_p^* can be written as:

$$\mathbf{Z}_p^* = \langle g \rangle = \{ h : h = g^x \pmod{p}, x \in \mathbf{Z}_p \} = \{ g, g^2, g^3, \dots, 1=g^{p-1} \}$$

The problem of finding a discrete logarithm is that, given $h, g \in \mathbf{Z}_p^*$, find an exponent $x \in \mathbf{Z}$ such that:

$$h = g^x \pmod{p}$$

This is usually written:

$$x = \log_g h, \text{ (sometimes it's required that } x \in \mathbf{Z}_p)$$

It may seem a bit strange that computing h , given g and x can be performed rather quickly as opposed to calculating x , given h and g . It is for this reason that exponentiation in prime groups is believed to be a one-way function, as is squaring in RSA-groups (the infeasible underlying problems are thus discrete logarithms and factoring). Discrete logarithms is used in Diffie-Hellman Key-Exchange, ElGamal Crypto-System and Schnorr- and DSS-Signatures.

Some researchers have an intuitive feeling that discrete logarithms is a harder problem to solve than factoring, although the truth of this matter is better left unsaid. The scientific skills of solving these problems have often been improved in the same pace. When a great step in factoring is taken, the same advance is soon followed in computing discrete logarithms.

For efficiency reasons, the subgroup $\mathbf{G}_q < \mathbf{Z}_p^*$ is sometimes used, where q is a prime that divides $p-1$. To construct \mathbf{G}_q , one first start by finding an element $g \in \mathbf{G}_q$ for which $O(g) = q$ (i.e. $g^q \equiv 1 \pmod{p}$). Then \mathbf{G}_q is created as:

$$\mathbf{G}_q = \langle g \rangle = \{ h : h = g^x \pmod{p}, x \in \mathbf{Z}_q \} = \{ g, g^2, g^3, \dots, 1=g^q \}$$

At the time of this writing, recommended sizes of p and q may be; $p \geq 2^{640}$ and $q \geq 2^{160}$. When evaluating the efficiency of crypto-systems that are based on discrete logs, these are the approximate values of p and q that will be used.

5.2.2. Roots in RSA Groups

Now, back to the RSA-group \mathbf{Z}_n^* , where n is a composite ($n=p*q$) consisting of two large primes, p and q , of approximately the same size. The order of \mathbf{Z}_n^* is denoted by $\varphi(n) = |\mathbf{Z}_n^*|$ and $\varphi(\cdot)$ is called Euler's totient-function. Euler's theorem, which can be viewed as a generalisation of Fermat, then states that:

$$x^{\varphi(n)} \equiv 1 \pmod{n}, \forall x \text{ such that } \gcd(x,n)=1$$

If p is prime, then $\varphi(p) = p-1$. So Fermat's little theorem can easily be derived from Euler's theorem. But if $n=p*q$, as in RSA, then $\varphi(n) = (p-1)*(q-1)$. Notice that most crypto-systems share the fact that the basic group operations should be taken mod n (n can be either prime or composite), whereas exponents should be performed mod $\varphi(n)$.

The RSA system (see [64]) was designed at MIT in 1978 by Ronald Rivest, Adi Shamir and Leonard Adleman and it was the first public-key crypto-system which could offer both encryption and digital signatures with the same set of functions. It doesn't use the discrete logarithm problem, but the problem of computing roots in composite groups (\mathbf{Z}_n^*). It is designed such that a public exponent (the public key), $e \in \mathbf{Z}_{\varphi(n)}$, is chosen (preferably, it may be rather small, typically $e=3$) and then the secret exponent (the secret key), $d \in \mathbf{Z}_{\varphi(n)}$, is calculated such that:

$$e*d \equiv 1 \pmod{\varphi(n)} \Leftrightarrow d \equiv e^{-1} \pmod{\varphi(n)}$$

From now on, d will for simplicity be denoted by $1/e \pmod{\varphi(n)}$. The public and secret functions in RSA then becomes:

$$E_i(x) = x^e \pmod{n}$$

$$D_i(x) = x^{1/e} \pmod{n}$$

where (n, e) is the user i 's public key and $(p, q, 1/e)$ is the secret key. Obviously, $n = p*q$, and $1/e = d = e^{-1} \pmod{\varphi(n)}$ and $\varphi(n) = (p-1)*(q-1)$. This clearly shows that if someone is able to factor the public key, n , into its prime factors, p and q , then the secret key, $1/e$, could easily be computed from the public key e .

Factoring is thus at least as hard as computing RSA roots, since if one were able to efficiently factor integers, RSA could be broken. Unfortunately though, there is no evidence that computing RSA roots is as hard as factoring. Therefore there might exist an algorithm that can break RSA without factoring the modulus. No such algorithm have yet been found though and most researchers believe that it isn't even possible to find one.

Comparing the security of RSA with systems based on discrete logs is very complicated since there is no easy relation between the complexity of factoring versus computing discrete logs. For the sake of evaluating the efficiency of different systems, however, it will be assumed that the moduli, n , will be approximately $n \geq 2^{640}$. Its factors (p and q) should be almost equal and about half the size of n . In order to prevent certain known attacks on RSA, the size of the message will be assumed to be about 320 bits. It may be increased by using a hash-function but in that case the output of the hash would also be about 320 bits long.

6. Cryptographic Primitives Based on RSA

RSA is perhaps the most widely spread public-key crypto-system in use today. That's because it's been around for some time and therefore been very well studied and evaluated. Another, not insignificant reason worth pointing out is that RSA is only patented in the US so it may be used "for free" in other countries. Many governments (e.g. USA, France and Russia) does however have export restrictions on cryptographic systems and products which imposes great difficulties for the acceptability of these systems (e.g. for Internet standards).

6.1. RSA Encryption

Everything that have been said about public-key in general can be said about RSA as well. For explicitness, let Alice's public key (exponent) be denoted by a and her corresponding secret key (exponent) be $1/a$. Also let Bob's public key be denoted by b and his corresponding secret key by $1/b$ (of course Alice's modulus is $n_A = p_A * q_A$, while Bob's is $n_B = p_B * q_B$). So now if Alice wants to send a message, M , to Bob this is how it would work:

Alice	Bob
Message m	
$T = E_B(m) = m^b \pmod{n_B}$	----T-->
	$m = D_B(T) = T^{1/b} \pmod{n_B}$

1. Alice encrypts the message, m , by using Bob's public key (n_B, b) to calculate the encryption $T = m^b \pmod{n_B}$ and send it to him.
2. Bob then decrypts Alice's message with his private key $1/b$ such that her message is recovered as $m = T^{1/b} \pmod{n_B} = (m^b)^{1/b} \pmod{n_B}$.

For an eavesdropper who overhears this communication to find out the original message, M , the following RSA-root would have to be computed:

$$m = T^{1/b} \pmod{n_B}$$

This is by no means a problem for Bob, since he possesses knowledge of the factors of the modulus n_B (i.e. p_B and q_B). The obvious attack on RSA is to factor the public modulus so that the secret rational exponent (that is used to compute the root) can efficiently be computed. As been pointed out in the previous chapter though, this is not easy.

Special care must be taken when the message is chosen from a small set of possible messages. If an attacker is able to guess what messages that might have been used he can easily encrypt all the possible messages with Bob's public key and compare it with Alice's encrypted message to see which one matches. This trick is called a 'plaintext attack' and RSA is definitely not well suited for this kind of attacks.

6.2. Digital Signatures with RSA

Now, in order for Alice to digitally sign a message and send it to Bob, this is how she would proceed:

Alice	Bob
Message m	
$S = D_A(m) = m^{1/a} \pmod{n_A}$	---(m,S)-->
	$m = E_A(S) = S^a \pmod{n_A} ?$

1. Alice uses her secret key $1/a$ to calculate the signature $S = m^{1/a} \pmod{n_A}$ on the message, m , and sends it to Bob (together with the message unless it is already known by him).
2. Bob then verifies Alice's signature on m with her public key (n_A, a) by checking whether the condition $m = S^a \pmod{n_A} = (m^{1/a})^a \pmod{n_A}$ holds.

This is an RSA signature in its simplest form. Provided that the message, m , is less than approximately 320 bits (in general about half as many bits as the modulus), a signature based on RSA is typically 640 bits long (80 bytes).

To sign messages (or encrypt for that matter) consisting of additional bits, it may be split in several smaller parts where each part is fed into the signing function. A better approach is however to use a hash function which produces a fixed length message digest (e.g. MD5, see [66]) of a message of any length. The signature is then computed on the message digest instead. It is often desired that such a hash function should be one-way and collision-free.

As long as the message consists of meaningful information such as written text, is it very difficult to forge RSA signatures without knowing the secret key. Forging RSA signatures on random messages is however extremely easy as will be seen in the chapters on e-cash. In those cases, a one-way hash function is necessary to make forging sufficiently complicated.

6.3. Public-Key Certificates Based on RSA

Now the question of how Bob can be sure that the public function $E_A(\cdot)$ really belongs to Alice might arise. If Alice and Bob can't meet face to face to exchange their public keys, what is preventing an intruder from masquerading Alice by sending Bob a different public key while claiming it belongs to Alice? This can be solved by using a central trusted third party which can certify Alice's and Bob's public functions. That party is usually called the certification authority (CA) but in this context it will often be identical with the Bank. What's relevant is that both Alice and Bob can fully trust the Bank's certificates.

From now on, the modulus will be excluded from the formulas in order to simplify the presentation. It should hopefully be clear from the context (and especially from what exponent that is used), which modulus that is meant.

The following protocol shows how Alice receives an RSA based public-key certificate on her public key a . Let Alice's identifying information (e.g. her name, social security number and IP-address) be denoted by ID_A and let $\text{hash}(\cdot)$ denote the hash function that is used. Also denote the Bank's public key by x and its corresponding secret key by $1/x$ (the Bank's modulus is obviously $n_x = p_x \cdot q_x$). This is how it works:

Bank		Alice
[SK: $1/x$]		[SK: $1/a$]
[PK: x]		[PK: a]
	<---(a, ID_A)---	identity ID_A
$C = \text{hash}(a, ID_A)^{1/x} \pmod{n_x}$	-----C-----	
	>	
		$C^x = \text{hash}(a, ID_A) \pmod{n_x}$?

1. Alice sends her public key, a , and her identifying information, ID_A , to the Bank for certification with its secret key, x .
2. The Bank may additionally reassure that it's actually talking with Alice. Then it computes a public-key certificate on Alice's public key, a , by signing a hash on a and ID_A , so that the certificate becomes $C = \text{hash}(a, ID_A)^{1/x} \pmod{n_x}$. This certificate is then sent back to Alice.
3. Alice may check that the Bank's signature is valid by verifying that $C^x = \text{hash}(a, ID_A) \pmod{n_x}$. She can then send her identity, ID_A , her public key, a , and the certificate, C , to Bob (or anyone who wishes to communicate with her).

The certificate, C , is usually stored by Alice so that she can use it everytime she needs to identify herself to somebody. It may also be stored in a public directory by the the Bank so that everyone that wants to communicate with Alice can get access to it.

ISO 9594-8 is a standard for the information structure of a certificate (it is identical to the ITU-, or previously known CCITT-standard X.509).

7. David Chaum's ecash

7.1. Blind RSA Signatures

The basic cryptographic primitive in ecash is the *blind signature*. This concept was introduced by David Chaum in [14] and is a method of having the Bank (or anyone) sign messages that is never seen.

The most common metaphor of a blind signature is that Alice provides the Bank with a written contract in an envelope with carbon paper. The Bank then puts its signature on the envelope and this signature is transferred through the carbon paper to the hidden contract. The unopened envelope is thereafter sent back to Alice. It is in fact impossible for anyone except Alice to open the envelope, but as she knows how to do it, she pulls out the contract which by now holds a (blind) signature by the Bank.

Blind signatures based on RSA is using the fact that RSA encryption possesses a multiplicative homomorphism. That means that encrypting a product of two numbers is equivalent to computing the product of those two independently encrypted numbers, i.e. the multiplicative operation is the same in both the unencrypted and the encrypted space.

Specifically, let m_1 and m_2 be any two integers that after encryption looks like $T_1 = m_1^{1/x}$ and $T_2 = m_2^{1/x}$. Now, if T_1 and T_2 is multiplied together (mod n of course), this is the result:

$$T_1 * T_2 = m_1^{1/x} * m_2^{1/x} = (m_1 * m_2)^{1/x}$$

A product of encryptions is thus an encryption of the product.

David Chaum's idea when he designed his protocol was to use this homomorphic property in such a way that Alice can multiply the original message with a random (encrypted) factor that will make the resulting image look like complete garbage to the Bank. If the Bank agrees to sign this random-looking data and return it to Alice, she is able to divide out the blinding factor such that the Bank's signature on the original message will appear.

It is important to note that the blinding factor must be chosen from the set of integers that are relatively prime to the Bank's modulus (i.e. the set \mathbf{Z}_n^*). Otherwise it is not possible to find an inverse and thus the division can't be performed. Inverses can be computed with an extended version of Euclid's algorithm for calculating gcd's, i.e. the greatest common divisor (see [26], [30] or [66]).

The format of a digital bill in ecash is a pair of bit-strings, one that represents the serial number of the Bank-note and one that represents the Bank's signature on the serial number. Using the same notation as in the description of the RSA signatures, a bill will thus look like: $(m, C) = (m, m^{1/x})$.

If the Bank should see the serial number of the bill it issues to Alice, then it could store this number in a database and by comparing this serial number with all the inserted bills it receives from the service providers, Alice's bill will sooner or later be found by the Bank. In this way, the Bank can get knowledge of what Alice is buying for her bills.

Most of the time, this isn't bothering Alice much since she doesn't have anything to hide from the Bank. There are however times when Alice wants to be anonymous when making payments. It doesn't necessarily have to do with something illegal such as trading drugs or buying pornography. It could also be the case that Alice wants to protect her freedom of opinion and hence doesn't want to reveal to the Bank to which political party she has paid her membership fee. Alternatively, Alice might want to use her digital bill when buying medical information which she doesn't want her insurance company to know anything about.

It is therefore essential that the Bank may never see the serial number on the bill it signs, but still every bill have to have a serial number that is unique so that one bill is distinguishable from any other. Chaum suggested that every user should generate their own serial numbers by using a pseudo-random generator. If the output of the pseudo-random generator is distributed well enough over the interval of possible serial numbers, then the risk that two bills should happen to share the same serial number is negligible. It is neither in the user's interest to generate a serial number that have already been used, since then that bill becomes worthless.

The protocols thus starts with Alice randomly choosing a serial number, m , and a random blinding factor, r . The blinding factor, r , should be chosen such that $\gcd(r, n) = 1$, i.e. it should be relatively prime to the Bank's modulus, $n = p * q$. Alice then encrypts r with the Bank's public key, x , so that $E_x(r) = r^x$. The result is then multiplied with the serial number and this number, $T = n * r^x$, is then sent to the Bank for signing.

Due to the homomorphic property of RSA, the Bank's signature becomes:

$$S = T^{1/x} = (n * r^x)^{1/x} = n^{1/x} * r$$

Now Alice simply divides S by r so that the signature becomes:

$$C = S / r = n^{1/x}$$

and Alice gets the bill $(n, C) = (n, n^{1/x})$ which she can use to pay any merchant with.

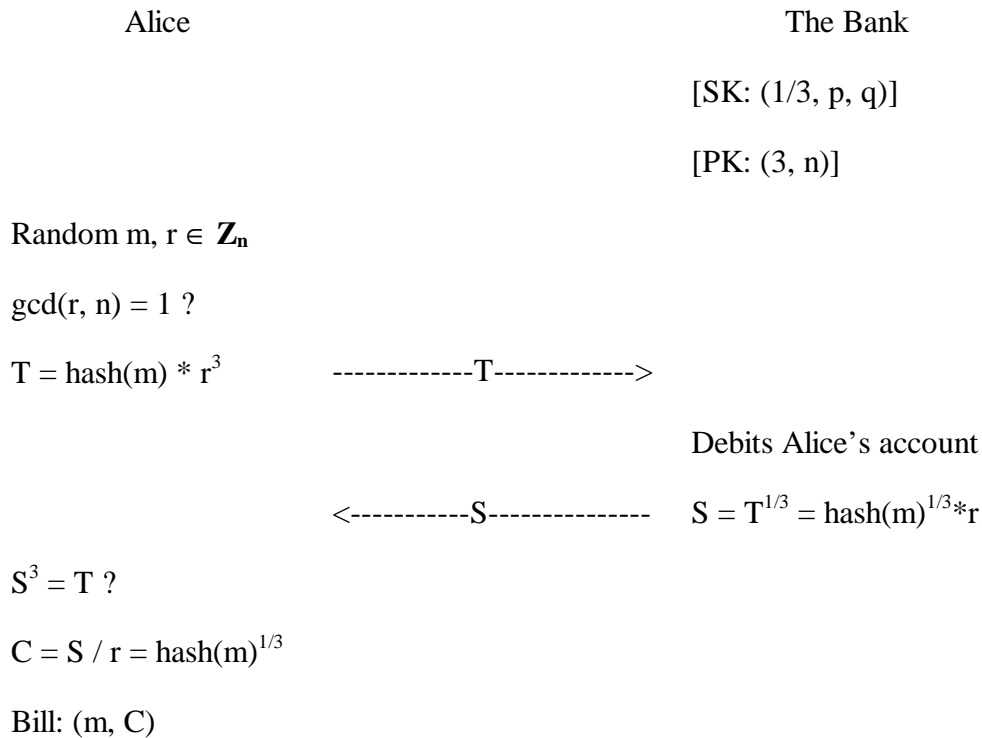
The careful reader may have noticed an alarming fact. Namely, since the serial numbers are freely chosen at random by Alice, she could easily counterfeit new digital Bank-notes by randomly picking a Bank signature, C' , and then compute the corresponding serial number as: $n' = C'^x$ so that $(n', C') = (n', n'^{1/x})$ becomes a perfectly valid digital bill, issued by the Bank!

In order to prevent this kind of forgery attacks, David Chaum included a one-way hash function into the protocol. A digital bill then gets the form: $(n, \text{hash}(n)^{1/x})$ and in order to forge such a bill, the one-way property of the hash function would have to be broken.

In the following protocol, Alice is to receive one digital dollar in a blind signature issuing protocol together with the Bank. The Bank's public exponent, x , will be assumed to be equal to 3 so that the coherence with the next protocols will be clearer.

7.2. Withdrawal of \$1 in ecash

This is exactly how the withdrawal of one digital dollar works in ecash:



1. Alice randomly picks the serial number, m , and the blinding factor, r , where r must be relatively prime to the Bank's public modulus. Then she sends, $T = \text{hash}(m) * r^3$, to the Bank.
2. The Bank debits Alice's account (after it have verified her identity, the authentication protocol is however not included in this description) by \$1 and then sends back the signature, $S = T^{1/3} = \text{hash}(m)^{1/3} * r$, to Alice.
3. Alice verifies that the Bank's signature is valid by checking if $S^3 = T$ and then she divides out the blinding factor so that $C = S / r = \text{hash}(m)^{1/3}$ will make $(m, C) = (m, \text{hash}(m)^{1/3})$ a perfect digital bill of one dollar.

Now if Alice is to pay with her bill, (m, C) , at Bob's shop he has to contact the Bank on-line to check that the serial number, m , haven't been spent before. If it's OK, then Bob accepts the payment and sends Alice the requested service. The Bank, at the same time, credits Bob for the inserted bill but doesn't know that he received it from Alice. In [15], Chaum presents a payment system based on the blind signatures.

The above protocol involves two interactions and since both T and S are about 640 bits long, the total transmitted information is approximately 1280 bits (160 bytes).

RSA computations are always quite time-consuming but since Alice can precompute T and doesn't need to compute C while still communicating with the Bank, the only real-time RSA operation that needs to be done is the computation of the signature, S , by the Bank.

7.3. The Payment and Insertion Protocol

For clarity, this is how the payment protocol works. Since ecash is an on-line system, the bill is immediately inserted at Bob's account in the same protocol.

Alice	Bob	The Bank
$C = \text{hash}(m)^{1/3}$		
Bill: (m, C)	---(m,C)-->	
	$C^3 = \text{hash}(m) ?$	---(m,C)-->
		$C^3 = \text{hash}(m) ?$
		m spent before?
		<---OK---- Credit Bob \$1

1. Alice send her digital bill, (m, $\text{hash}(m)^{1/3}$) to Bob.
2. Bob may additionally verify the Bank's signature by checking if $C^3 = \text{hash}(m)$. In order to reassure that the bill haven't been spent before, he must however send it to the Bank.
3. The Bank can now verify its own signature and also check that the serial number, m, isn't stored in the database of already spent bills. If it isn't, then Bob is credited and a (digitally) signed receipt is returned to him (denoted here as OK).

Notice that the transferring of the service is excluded from the protocol. When Bob has received the receipt from the Bank he will however send Alice the requested service or product.

The above protocol doesn't necessarily involve any time-consuming RSA operations. The only exception would be the verification of the signature, but it may entirely be put on the Bank's responsibility. Bob only needs the receipt from the Bank so he may skip the verification if he wish.

The sending of the digital bill, (m, C), is fast since Alice only has to read it from memory. The size of the bill is however quite large and approximately occupies 1280 bits (160 bytes).

What mainly slows down the protocol is that the Bank has to be contacted for each payment. It is complicated to add other Banks to the system, since each Bank has to verify every transaction of its own issued bills. The crucial thing is that there can only be one database of already spent bills for every Bank. The current ecash trial (see [32]) may show how practical this approach is.

7.4. Additional Denominations

Now what if Alice wants to pay Bob with an amount of, say 10 dollars? Using one dollar bills quickly gets cumbersome, and this calls for a method of incorporating additional denominations in ecash.

David Chaum demonstrates in his article, Online Cash Checks (see [16]), how this can be done with the blind RSA scheme presented above. There are two ways of looking at the following method. In one view, Alice receives a digital bill that she is able to devalue into several smaller entities of lesser value. In the other perspective she is able to store several low-value bills in a compact space such that no more memory is required than what it takes to store one single dollar bill. This RSA-specific signature compression technique, that in a way can be said to have been invented by Chaum, is called ‘intermingling’.

The way additional denominations was introduced in Chaum’s previous systems was to let the Bank sign the digital bills with different keys for every denomination that was needed. It was also realised that the same modulus could be used, so that the only thing that had to be changed when the bill should denote a different value was the exponent. This is on the analogy of having different prints on paper bills with different denominations.

The intermingling compression technique is again using the multiplicative homomorphic property of RSA in such a way that RSA signatures may be multiplied together without losing any information. Because of that, the signatures can be restored to their original shape as long as the used exponents are remembered.

For explicitness, let one dollar correspond to the public RSA exponent 3, just as in the previous protocol. Two dollars should correspond to the public exponent 5, four dollars by the exponent 7 etc. Every subsequent denomination which is a multiple of 2 is thus represented by a corresponding public RSA exponent, taken from the series of (odd) primes. This gives an injective (1-1) relation between the denominations and the exponents.

The only possible denominations that are available in ecash are all multiples of 2. It is however possible to add several ecash bills together in order to form other denominations. David Chaum’s presentation in [16] is slightly different. In that system, any denomination is possible and may also be devalued into terms of multiples of 2 which equals the sum of the bill.

The exponent that corresponds to a denomination that isn’t a multiple of 2, is the product of the corresponding exponents of the (2-multiple) terms in the denomination’s sum. For example, 5 can be written as $5 = 4 + 1$, and since 4 corresponds to the exponent 7 and 1 corresponds to the exponent 3, the exponent that corresponds to 5 is thus $21 = 7 * 3$.

Extra care must be taken when designing this kind of system since otherwise it might be possible to compute new RSA signatures from old ones and hence being able to counterfeit. A result by Evertse and van Heyst (see [35]) implies, loosely stated, that only multiplicative combinations of old RSA signatures can be computed without knowledge of the secret key.

One way of attacking this problem is to only allow denominations up to a maximum value and let this value be the only one that is possible to withdraw from the Bank.

The following table shows the denominations of up to 15 dollars and the corresponding public RSA exponents:

Denomination:	Prime Factors:	Public RSA Exponent
1	3	= 3
2	5	= 5
3	5*3	= 15
4	7	= 7
5	7 *3	= 21
6	7*5	= 35
7	7*5*3	= 105
8	11	= 11
9	11 *3	= 33
10	11 *5	= 55
11	11 *5*3	= 165
12	11*7	= 77
13	11*7 *3	= 231
14	11*7*5	= 385
15	11*7*5*3	= 1155

Assuming that Alice now has a digital bill that is worth 5 dollars, i.e. $(n,C) = (n, \text{hash}(n)^{1/(7*3)})$. By raising the signature to the 7th power, the bill can be devalued into 1 dollar in this way:

$$C^7 = [\text{hash}(n)^{1/(7*3)}]^7 = \text{hash}(n)^{1/3}$$

Now (n, C^7) is a perfectly valid digital bill of 1 dollar.

One method that makes every value distinct is to only allow exponents without any powers. Notice that this has the effect that the bills can only be devalued into smaller denominations that are multiples of 2. For instance, the public RSA exponent of a signature worth 10 dollars is 55. The value of 10 dollars could ambiguously be devalued into e.g. 8+2 or 6+3+1 unless the unique prime factorisation of 55 (=11*5) would tell us that 8+2 are the only possible terms that results in the sum of 10.

In his paper [16], David Chaum demonstrates three ways of returning change to a customer that doesn't have the exact amount. The current implementation of ecash doesn't use this facility however but it will be described here anyway since it may be implemented in the future.

7.5. Withdrawal of two \$15 bills in ecash

In order to demonstrate how change can be returned to the customer we will assume that Alice first have withdrawn two bills of \$15 in ecash from the Bank and then pays Bob 5 and 3 dollars with these two bills. In the first payment of 5 dollars, Bob thus have to return 10 dollars of change to Alice and in the second payment of 3 dollars, the change will be 12 dollars. Finally, Alice will have 22 dollars of ecash.

Alice		The Bank
		[SK: (1/h, p, q)]
		[PK: (h, n)]
		$h = 1155 = 11 * 7 * 5 * 3$
Random $m_1, m_2, r_1, r_2 \in \mathbf{Z}_n$		
gcd(r_1, n) = 1 = gcd(r_2, n) ?		
$T_1 = \text{hash}(m_1) * r_1^h$		
$T_2 = \text{hash}(m_2) * r_2^h$	-----(T_1, T_2)----->	
		Debits Alice's account \$30
		$S_1 = T_1^{1/h} = \text{hash}(m_1)^{1/h} * r_1$
	<-----(S_1, S_2)-----	$S_2 = T_2^{1/h} = \text{hash}(m_2)^{1/h} * r_2$
Divide out r_1 and r_2		
$(m_1, C_1) = (m_1, \text{hash}(m_1)^{1/h})$		
$(m_2, C_2) = (m_2, \text{hash}(m_2)^{1/h})$		

1. Alice randomly picks two serial numbers, m_1 and m_2 , and two blinding factors, r_1 and r_2 , and these two numbers must be relatively prime to the Bank's public modulus. Then Alice sends $T_1 = \text{hash}(m_1) * r_1^h$ and $T_2 = \text{hash}(m_2) * r_2^h$ to the Bank.
2. The Bank debits Alice's account by \$30 (= 15+15) and then replies by sending back the signatures, $S_1 = T_1^{1/h} = \text{hash}(m_1)^{1/h} * r_1$ and $S_2 = T_2^{1/h} = \text{hash}(m_2)^{1/h} * r_2$, to Alice.
3. Alice verifies that the Bank's signatures are valid by checking if $S_1^h = T_1$ and $S_2^h = T_2$. Then she divides out the blinding factors so that $C_1 = S_1/r_1 = \text{hash}(m_1)^{1/h}$ and $C_2 = S_2/r_2 = \text{hash}(m_2)^{1/h}$. This will make $(m_1, C_1) = (m_1, \text{hash}(m_1)^{1/h})$ and $(m_2, C_2) = (m_2, \text{hash}(m_2)^{1/h})$ two perfect digital bills of 15 dollars (remember that $h = 11 * 7 * 5 * 3$).

The performance of this protocol is approximately the same as the performance of two parallel executions of the previous withdrawal protocol of \$1 in ecash.

7.6. Cookie Jar Change

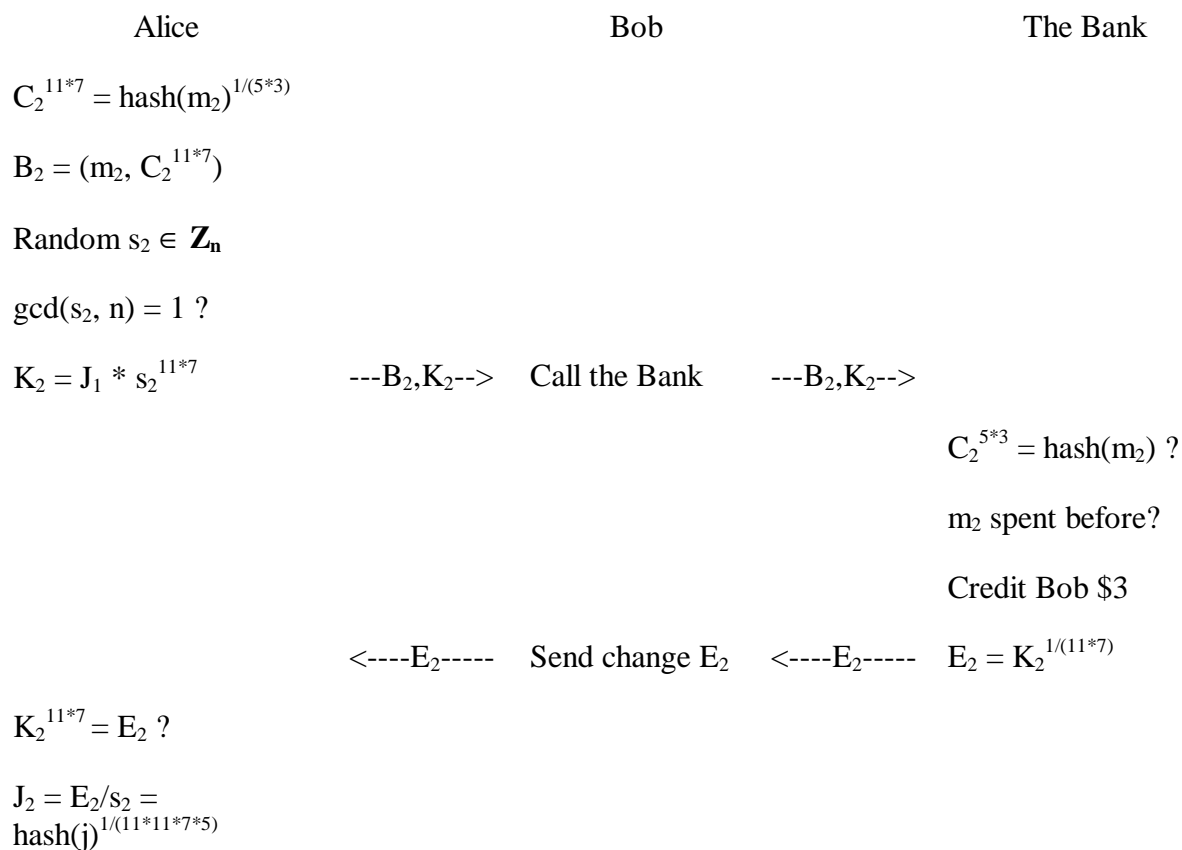
When Alice is to pay Bob 5 dollars with the first \$15 bill, $(m_1, C_1) = (m_1, \text{hash}(m_1)^{1/(11*7*5*3)})$, she devalues it by raising the signature, C_1 , by $55 = 11*5$ (corresponding to the change of \$10) so that the remaining signature exponent becomes $1/21 = 1/(7*3)$, which corresponds to \$5.

Alice	Bob	The Bank
$C_1^{11*5} = \text{hash}(m_1)^{1/(7*3)}$		
$B_1 = (m_1, C_1^{11*5})$		
Random $j, s_1 \in \mathbf{Z}_n$		
$\text{gcd}(s_1, n) = 1 ?$		
$K_1 = \text{hash}(j) * s_1^{11*5}$	---B ₁ ,K ₁ --> Call the Bank	---B ₁ ,K ₁ -->
		$C_1^{7*3} = \text{hash}(m_1) ?$
		m_1 spent before?
		Credit Bob \$5
	<----E ₁ ----- Send change E ₁	<----E ₁ ----- $E_1 = K_1^{1/(11*5)}$
$K_1^{11*5} = E_1 ?$		
$J_1 = E_1/s_1 = \text{hash}(j)^{1/(11*5)}$		

1. Alice devalues her bill by computing $C_1^{11*5} = \text{hash}(m_1)^{1/(7*3)}$ so that $B_1 : (m_1, C_1^{11*5})$ becomes a digital bill of \$5. She then randomly selects the serial number j and the blinding factor s_1 (relatively prime to n), computes the “cookie jar” $K_1 = \text{hash}(j)*s_1^{11*5}$ and sends it together with the digital bill, $B_1 = (m_1, \text{hash}(m_1)^{1/(7*3)})$, to Bob. Bob may additionally verify the Bank’s signature but otherwise he just sends it unchanged to the Bank.
2. The Bank can now verify its own signature and also check that the serial number, m_1 , isn’t stored in the database of already spent bills. If it isn’t, then Bob is credited by \$5 and the change, $E_1 = K_1^{1/(11*5)}$, is returned to him. Bob, in turn, sends the change to Alice.
3. Alice can verify that the change is valid by checking that $K_1^{11*5} = E_1$? If it is, then she computes $J_1 = E_1/s_1 = \text{hash}(j)^{1/(11*5)}$ so that (j, J_1) becomes a digital bill of \$10.

Notice that the only way the Bank can know how much change is to be returned is to assume that only bills of the maximum amount can be withdrawn (\$15 in this case).

In the next payment, Alice devalues her second \$15 bill, $(m_2, C_2) = (m_2, \text{hash}(m_2)^{1/(11*7*5*3)})$, by raising the signature with $77 = 11*7$ (which corresponds to the change of 12 dollars). The remaining signature exponent which corresponds to 3 dollar is thus $1/15 = 1/(5*3)$.



1. Alice devalues her bill by computing $C_2^{11*7} = \text{hash}(m_2)^{1/(5*3)}$ so that $B_2 : (m_2, C_2^{11*7})$ becomes a digital bill of \$3. She then randomly selects a blinding factor s_2 (relatively prime to n) and computes the “cookie jar” $K_2 = \text{hash}(j) * s_2^{11*7}$. The digital bill of 3 dollar, $B_2 = (m_2, \text{hash}(m_2)^{1/(5*3)})$, and K_2 is then sent to Bob. Bob may additionally verify the Bank’s signature but otherwise he just sends it unchanged to the Bank.
2. The Bank can now verify its own signature and also check that the serial number, m_2 , isn’t stored in the database of already spent bills. If it isn’t, then Bob is credited by \$3 and the change, $E_2 = K_2^{1/(11*7)}$, is returned to him. Bob, in turn, sends the change to Alice.
3. Alice can verify that the change is valid by checking that $K_2^{11*7} = E_2$? If it is, then she computes $J_2 = E_2/s_2 = \text{hash}(j)^{1/(11*11*7*5)}$ so that (j, J_2) is digital bill of \$22 (=8+8+4+2).

Now, Alice can deposit her digital bill of \$22 to the Bank in an insertion protocol.

Observe that this scheme of using multiple denominations doesn’t have any negative effect on the efficiency of the protocol. It is solely the modulus that decides the number of bits that has to be used, and the exponents only effects the RSA computation at minimal.

Leaving change does however require the Bank to perform one more real-time RSA computation but notice that Alice can pre-compute K_2 , so it doesn’t effect her contribution to the inefficiency of the protocol.

7.7. Declared Note Value

The other version of leaving change makes it possible to spend the change money without an intervening withdrawal transaction. Alice is still assumed to have withdrawn two bills of \$15. In order to pay Bob 5 dollars, she is again doing the same devaluing procedure by raising the signature of the bill with $55 = 11 * 5$.

In this case it isn't necessary to always withdraw bills of the maximum amount. Alice can claim any amount of change as she wants. To prevent her from cheating, the Bank "locks up" the change in such a way that Alice can only unlock it in case she has claimed a fair amount.

In the payment transaction, not only the digital bill and the "cookie jar" is sent but also the public exponent (55) that corresponds to the amount of change that Alice wants to receive.

Alice	Bob	The Bank
$C_1^{11*5} = \text{hash}(m_1)^{1/(7*3)}$		
$B_1 = (m_1, C_1^{11*5})$		
Random $s \in \mathbf{Z}_n^*$		
$K_1 = \text{hash}(m_2) * s^{11*5}$	$\langle -B_1, K_1, 55 \rangle$	Call the Bank
		$\langle -B_1, K_1, 55 \rangle$
		$C_1^{7*3} = \text{hash}(m_1) ?$
		m_1 spent before?
		Credit Bob \$5
	$\langle \text{----}E_1\text{----} \rangle$	Send change E_1
		$\langle \text{----}E_1\text{----} \rangle$
		$E_1 = K_1^{1/(11*5)} * \text{hash}[\text{hash}(m_1)^{1/(11*5)}]$

Notice that $E_1 = \text{hash}(m_2)^{1/(11*5)} * s * \text{hash}[\text{hash}(m_1)^{1/(11*5)}]$. Now, if Alice is to get a signature that is worth \$10 she can still divide out the blinding factor, s , but in order for her to divide out the protection factor ("the lock"), $\text{hash}[\text{hash}(m_1)^{1/(11*5)}]$, she has to know the signature, $\text{hash}(m_1)^{1/(11*5)}$, which she only does if she has claimed a correct amount of change to Bob.

Provided that Alice has been telling the truth, she can divide E_1 by $(s * \text{hash}[\text{hash}(m_1)^{1/(11*5)}])$ and thus receive the signature, $\text{hash}(m_2)^{1/(11*5)}$, that when multiplied with her next \$15 bill forms a digital bill, $(m_2, \text{hash}(m_2)^{1/(11*11*7*5*5*3)})$ that is worth \$25.

It would be possible for Alice to claim to get a returned change of an amount less than what she deserves, and get away with it. This does however unlikely lie in Alice's interests. The security of this scheme is based on the assumption that $\text{hash}(\cdot)$ is a one-way function.

The efficiency of the above protocol is very close to the previous. The sending of the public modulus (55) only has a limited effect to the number of transmitted bits since it is usually quite a small number. The Bank, in turn, only have to perform one more hashing and yet another RSA computation in real-time.

7.8. Distributed Change

In the third version of how change can be returned to Alice allows her to fill in missing parts that fill in missing denominations in bills not yet spent. This gives better flexibility to the system.

Suppose in the last example, where Alice pays Bob \$5, that she wants the change of \$10 to be in the denominations \$8 and \$2, distributed on her two bills. She then forms K_1 as:

$$K_1 = \text{hash}(m_1)^5 * \text{hash}(m_2)^{11}$$

so that she finally arrives with the change, J_1 , signed by the Bank as:

$$J_1 = K_1^{1/(11*5)} = \text{hash}(m_1)^{1/11} * \text{hash}(m_2)^{1/5}$$

From J_1 , the two signatures corresponding to \$2 and \$8 can be derived by first defining:

$$u = 1/5 \text{ mod } 11$$

$$v = 5*u \text{ div } 11$$

and then computing:

$$C_1' = \text{hash}(m_1)^{1/11} = [J_1^5 / \text{hash}(m_2)]^u / \text{hash}(m_1)^v$$

$$C_2' = \text{hash}(m_2)^{1/5} = J_1 / \text{hash}(m_1)^{1/11}$$

Now C_1' and C_2' can be multiplied with what's left of C_1 and C_2 so that their denominations are added together.

Although the above scheme requires quite some computational capacity of Alice there is really nothing that need to be performed in real-time. The effect on the efficiency is thus negligible.

7.9. Hidden Note Value

There is also a method that eliminates the need for Alice to reveal the desired amount of change she wants to receive from the Bank. The following protocol can be viewed as an improvement of the declared note value protocol above. The payments message is exactly the same except that the public exponent (55) that corresponds to the change is not sent. The protection factor has also been modified by getting rid of the extra hash function and instead been raised to a random power, z , chosen by the Bank.

Alice	Bob	The Bank
$C_1^{11*5} = \text{hash}(m_1)^{1/(7*3)}$		
$B_1 = (m_1, C_1^{11*5})$		
Random $s \in \mathbf{Z}_n^*$		
$K_1 = \text{hash}(m_2) * s^{11*5}$	---B ₁ ,K ₁ --> Call the Bank	---B ₁ ,K ₁ -->
		$C_1^{7*3} = \text{hash}(m_1) ?$
		m_1 spent before?
		Credit Bob \$5
		Random $z \in \mathbf{Z}_{\phi(n)}$
	<---E ₁ ,z----	Send change E ₁ <---E ₁ ,z----
		$E_1 = K_1^{1/(11*5)} * \text{hash}(m_1)^{z/(11*5)}$

Notice that $E_1 = \text{hash}(m_2)^{1/(11*5)} * s * \text{hash}(m_1)^{z/(11*5)}$ and since Alice gets to know z in the returned values, she can divide out the blinding- and the protection factor so that she finally ends up with the signature $J_1 = \text{hash}(m_2)^{1/(11*5)}$.

Observe also that if z were known to Alice before the payment, then she could cheat by including $\text{hash}(m_1)^{-z}$ as a factor of K_1 . In that case she would end up with the system-wide maximum change.

This protocol is almost identical to the declared note value protocol in terms of efficiency. Although Alice doesn't have to send the public exponent corresponding to the desired change, the Bank sends the random exponent, z , which is only slightly larger. The computation of the protection factor that the Bank locks up the change with is modified at minimal.

A combination of the above protocols is quite workable but in the current implementation of ecash, no change is left at all. Although the facility of leaving change improves the flexibility of the system it also slows it down. If low-value payments are considered, there may not be too much of a drawback to exclude this possibility.

7.10. Related Work

In [20], Chaum, Fiat and Naor describes the first anonymous off-line system that is known in literature. Their system is too complex (and unfortunately also very inefficient) to be described here (a simple description of it in plain English can be found in [66] though).

Since then, several similar systems have been proposed (e.g. [21], [46], [61] and [39]), some of which have extensions such as divisibility of bills or transferability.

The transferability property is given a special treatment by van Antwerpen in [1], which presents a generic method of designing transferable payments systems. A result by Chaum and Pedersen in [23] does however imply that all such systems will have the property that the digital bills will be increased in size for every transfer that it is given. Therefore digital cash schemes are necessarily thought to only be used for making single payments before the bill gets returned to the Bank (alternatively a fixed number of transactions would be possible).

Chaum and Pedersen does also suggest a different model in [22] which is called a ‘wallet with observer’. That model gives prior restraint to double spending as opposed to Chaum, Fiat and Naor’s system. It’s privacy protection mechanism was improved in [27] by Cramer and Pedersen.

It is also this model that is used in Ferguson’s systems (see [35] and [37]) and the systems of Stefan Brands (see [6] and [7]). Brands systems are also the foundations of the protocols that are used in CAFE. The next part of this paper will describe an even more efficient protocol by Stefan Brands (see [12]) that have been used by him to design several slightly different systems (e.g. [8] and [9]) which all can be derived from the same basic protocol.

8. Cryptographic Primitives Based on D-L

Before describing the cash systems by Stefan Brands it might be a good idea to first go through some examples of cryptographic primitives that uses the discrete logarithm problem instead of factoring to construct one-way functions. Many of the following protocols are also used by Brands in his rather complex anonymous off-line cash system.

8.1. Diffie-Hellman Key Exchange

Whitfield Diffie and Martin E. Hellman's article "New Directions in Cryptography", [31], was the starting point of the entire modern cryptology with mathematically "provable" security as opposed to previously so-called "scrambling-methods" (DES, [56] would be an example).

In this typical case does Alice and Bob wish to send sensitive information over an insecure channel such as the Internet. They may do so using a traditional secret-key enciphering algorithm such as DES if they both agree on a shared secret key that may never be revealed to any other than Alice and Bob. The problem that is inherent in all secret-key crypto-systems is the exchange of keys. There are in general only two ways for Alice and Bob to agree on the secret key. The first alternative is to meet face to face which may seem very inefficient by means of modern telecommunication. The other alternative is to build a separate secure channel just for the exchange of keys, but this is a paradox; Why not use the secure channel for the transmission of the sensitive data in the first place?

To solve the problem of how to exchange secret keys over an insecure channel, Diffie and Hellman proposed a protocol based on the discrete log assumption. The protocol have been named after the authors, namely Diffie-Hellman (D-H) Key-Exchange. The protocol works like this:

Alice		Bob
Random $x \in \mathbf{Z}$		Random $y \in \mathbf{Z}$
$GX = g^x \pmod{p}$	-----GX-----	
	>	
	<-----GY-----	$GY = g^y \pmod{p}$
	-	
$K = GY^x \pmod{p}$		$K = GX^y \pmod{p}$

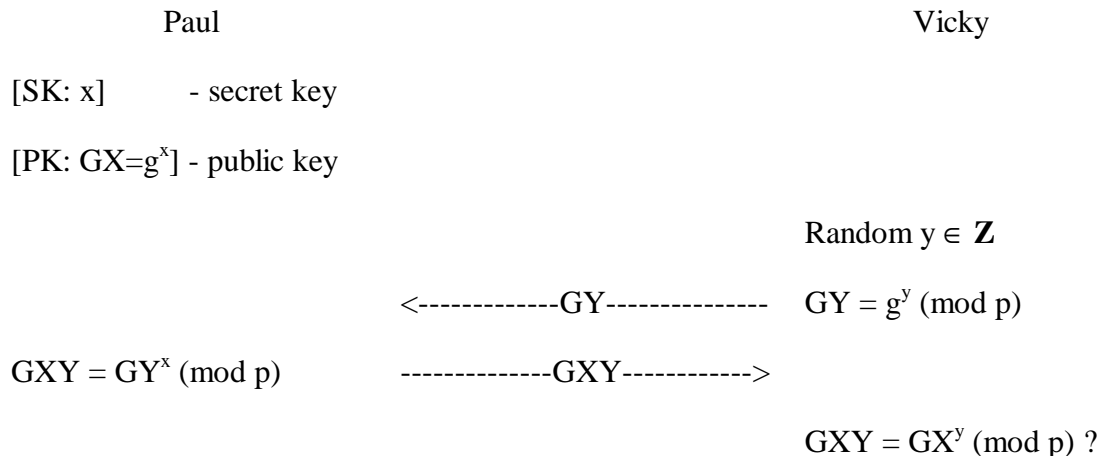
1. Alice randomly selects an integer, x , and sends $GX = g^x \pmod{p}$ to Bob.
2. Bob randomly selects an integer, y , and sends $GY = g^y \pmod{p}$ to Alice.
3. Alice calculates the secret key as $K = GY^x \pmod{p}$. Notice that $K = g^{x*y} \pmod{p}$.
4. Bob also calculates the secret key as $K = GX^y \pmod{p}$, which is also $K = g^{x*y} \pmod{p}$.

Now Alice and Bob both have received a secret key that they can use when communicating.

Notice that even if an eavesdropper would overhear this conversation and get hold of G_X and G_Y it is assumed to be infeasible to calculate K from this data (in polynomial time), even though the generator g would be known. This is usually defined as the Diffie-Hellman (D-H) problem. If the man-in-the-middle (the eavesdropper) knew how to calculate discrete logs efficiently, then the problem would obviously be trivial. As with RSA, it has not been proved that this is the only way to calculate K from G_X and G_Y , but no other algorithm is known to date.

8.2. D-H Authentication

The D-H method can also be used for authentication (secure identification). Now Paul (prover) wants to convince Vicky (verifier) that he knows the secret key, x , that corresponds to his public key, $G_X = g^x$. He obviously doesn't want to tell Vicky what x is, in which case she could forge Paul's signatures. In the following protocol, Vicky lets Paul solve a problem that requires knowledge of x . If Paul can provide Vicky with a correct solution within polynomial time, then she will be convinced that it is actually Paul that she's talking to and that nobody is trying to masquerade him.



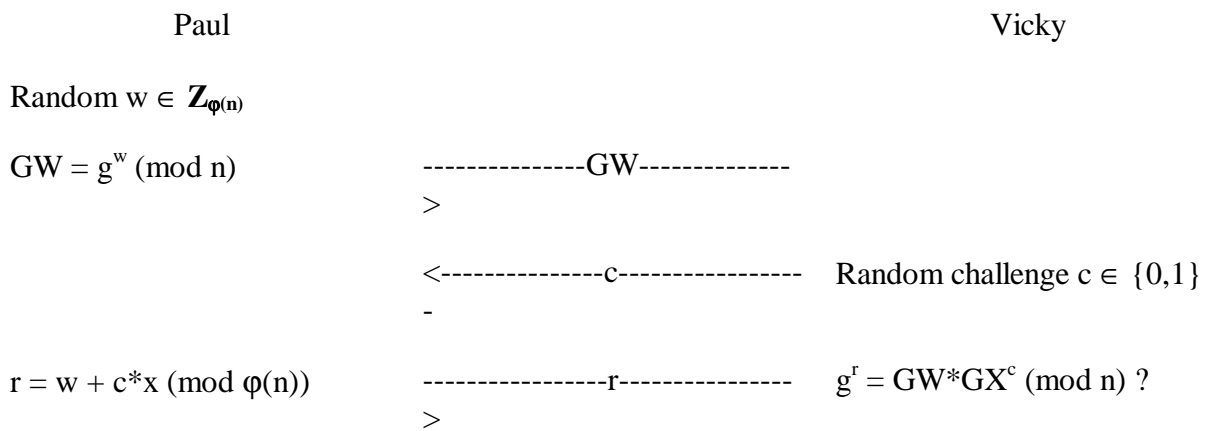
1. Vicky randomly selects a number, y , and sends Paul the challenge $G_Y = g^y \pmod{p}$.
2. Paul calculates $G_{XY} = G_Y^x \pmod{p} = g^{x \cdot y} \pmod{p}$ and sends the result back to Vicky.
3. Vicky verifies that $G_{XY} = G_X^y \pmod{p}$. Remember that she already knows Paul's public key G_X .

This is an example of an interactive proof. It only takes Paul two interactions with Alice to prove to her that he possesses the secret key x . The above protocol is 'zero-knowledge' (ZK) in the sense that the secret (the witness) x is never revealed to Alice. In fact, Alice doesn't get any Shannon-information about x whatsoever. Paul's response doesn't reveal anything to her that she couldn't have computed herself, specifically Alice could have computed Paul's response G_{XY} as G_X^y in advance since his public key (G_X) is already known to her. Although this protocol only involves two interactions, it is still rather inefficient since it involves a modular exponentiation in real-time (performed by Paul). Both Vicky's challenge and Paul's response are also quite large. Using the assumptions of previous chapters, they consist of approximately 640 bits each.

8.3. Zero-Knowledge Proofs of Knowledge

The concept of zero-knowledge proofs was explored and defined in the early eighties (e.g. Goldwasser, Micali and Rackoff, [42]). It has been shown to be a very powerful tool in many ways. General results by, for instance, Goldreich, Micali and Wigderson (see [41]) imply that almost anything that can be proved in polynomial time can be proved in a polynomial time zero-knowledge interactive proof such that no information about the particular solution (the witness) is released. The only thing that the verifier needs to know is that a solution *does* exist.

We will not be concern with the general concept in this paper, only specific protocols that serves the purpose of simplifying the explanation of certain applications in digital cash. One such protocol for demonstrating possession of a discrete logarithm was presented in [19] by Chaum, Evertse and van de Graaf. The situation is the same as before, Paul wants to convince Vicky that he knows the discrete logarithm to GX, without telling her what it is.



1. Paul randomly selects a number, w , and sends the commit $GW = g^w \pmod{n}$ to Vicky. Notice that this doesn't reveal any information about x .
2. Vicky randomly selects a binary challenge, c , and sends it to Paul.
3. Paul computes the response $r = w+c*x \pmod{\phi(n)}$ and sends it to Vicky.
4. Vicky verifies that $g^r = GW*GX^c \pmod{n} = g^{w+x*c} \pmod{n}$.

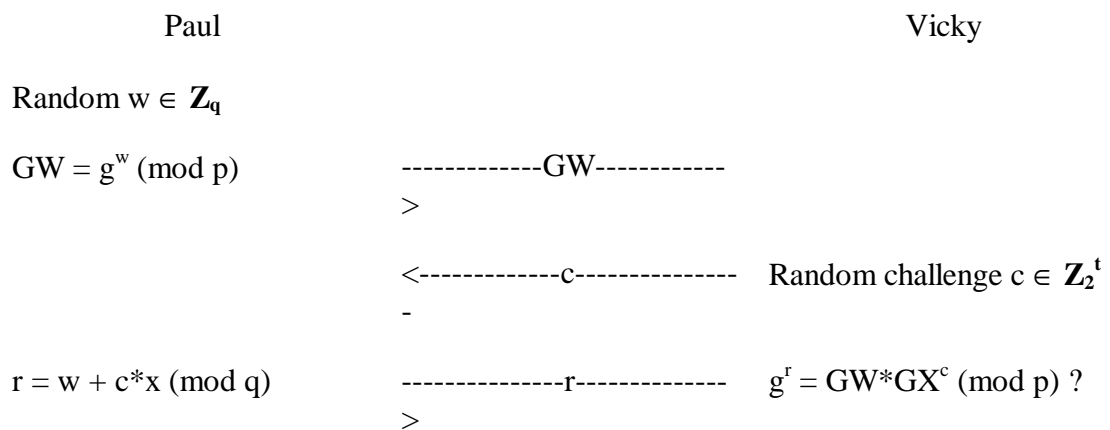
If w is completely random and since it's never revealed to Vicky (assuming that she is unable to compute discrete logs), then no information about x is released in Paul's response r . In other words, x (and w) may be anything, given c and r .

Also note that a cheater (i.e. one who doesn't know x) has 50% chance of succeeding in masquerade Paul by guessing Vicki's challenges in advance. If $c = 0$, then the response is simply $r = w$ and if the cheater expects that Vicky will send $c = 1$, then the cheater can start by choosing the response r in advance and then compute the commit as $GW = g^r / GX$.

In order to limit these attacks, the protocol can be performed sequentially as many times as Vicky desires to become ascertain that Paul couldn't have guessed all her challenges in advance. This method is called 'cut-and-choose' and is very inefficient in practice.

8.4. Schnorr's Authentication Protocol

C. P. Schnorr improved the efficiency of Chaum, Evertse and van de Graaf's protocol in order to for it to be implementable on smart cards (see [67]). He suggested that the arithmetic should be performed in the subgroup \mathbf{G}_q (as described above) and also that the challenge c possibly could be picked from the interval $\{0, \dots, 2^t - 1\}$, for a security parameter t . Schnorr used $t = 72$ in his paper, but here will we assume that t is approximately 128 bits in view of the ever ongoing advances in algorithm design and computing. This way, the cut-and-choose methodology of the previous protocol is removed since it's a lot harder for a masquerader to guess a random integer than just a random bit. Apart from that, Schnorr's protocol is practically identical to Chaum, Evertse and van de Graaf's version.



1. Paul randomly selects a number, w , and sends the commit $GW = g^w \pmod p$ to Vicky. Notice that this doesn't reveal any information about x .
2. Vicky randomly selects a challenge, c (with $0 \leq c \leq 2^t - 1$), and sends it to Paul.
3. Paul computes the response $r = w + c*x \pmod q$ and sends it to Vicky.
4. Vicky verifies that $g^r = GW*GX^c \pmod p = g^{w+x*c} \pmod p$.

Notice that Paul's smart card only has to perform one real-time modular multiplication (and a modular addition but that's a lot faster) since GW can be computed in advance of the actual authentication. The communication efficiency is also very high, only three interactions is needed and GW is 640 bits, c is 128 bits and r is about 160 bits so the total number of bits that is transmitted is: 928 bits (116 bytes).

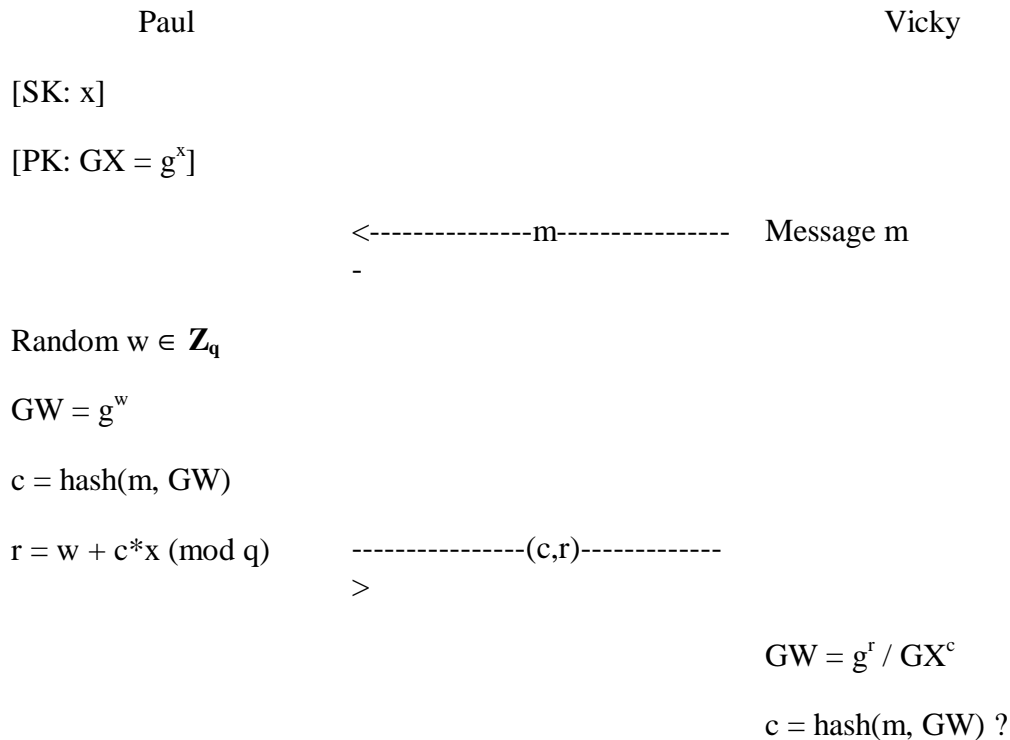
From now on, unless something else is stated, all the arithmetic will be assumed to be performed in \mathbf{G}_q , so $\pmod p$ will be left out from the formulas. Hopefully, this will make them look cleaner. At the same time, all the calculations with exponents will be performed $\pmod q$ but that will still be explicitly written out though.

8.5. Schnorr Signatures

Most interactive proofs can be transformed into non-interactive proofs by using a standard technique due to Fiat and Shamir (see [38]). This makes them more similar to ordinary digital signature schemes, such as the RSA-scheme.

The idea is to replace Vicki's challenge in the protocol by a cryptographically secure pseudo-random hash-function which allows for Paul to compute the challenge all by himself. The only important thing to beware of is to ensure that he cannot predict the challenge, in which case he could easily cheat. The hash function will also preferably be assumed to be collision-free.

Let $\text{hash}(\cdot)$ denote the hash-function. By applying this function on the values that are known to Paul before Vicky should have sent her challenge (i.e. the message, m , and the commit GW), then Paul can use the result of the hash as a replacement of Vicki's challenge. Specifically, it should be $c = \text{hash}(m, GW)$. Since both Paul and Vicky can compute c on their own, it doesn't have to be sent along with the rest of the data in the signature. For efficiency reasons, c is sent anyway, instead of GW , which is a lot larger number.



1. Vicky sends the message, m , to Paul to be signed with his secret key x .
2. Paul randomly chooses a number, w , and computes the commit $GW = g^w$, the challenge as $c = \text{hash}(m, GW)$ and the response $r = w + c*x \pmod{q}$. Then he transmits the signature (c, r) on m to Vicky.
3. Vicky computes the commit as $GW = g^f / GX^c$ and verifies that $c = \text{hash}(m, GW)$.

Notice that a Schnorr signature is significantly more compact than an RSA signature that is assumed to provide the same security against forgery. Using the above assumptions, an RSA

signature is typically 640 bits (80 bytes), while a Schnorr signature, (c, r) , is merely $128 + 160 = 288$ bits (36 bytes).

8.6. Secret-Key Certificates

Stefan Brands introduces the concept of secret-key certificates in [10]. Just as with public-key certificates, it is only possible to retrieve a triple consisting of a secret key, a corresponding public key and a certificate on the public key in an issuing protocol together with the Certificate Authority (CA). In contrast to public-key certificates however is it possible for anyone to generate a pair of a public key and a secret-key certificate on the public key without the interaction with the CA.

Nevertheless does secret-key certificates offer the same functionality as do public-key certificates, since there is no point in verifying anyone's certificate unless that person demonstrates possession of the secret key in an authentication or signature protocol with the verifier.

The concept may be a bit confusing at first, perhaps the name secret-key certificates is also badly chosen since it is still a certificate *on the public key*. Just verifying the certificate on the public key by itself does not prove anything to the verifier unless the prover also demonstrates possession of the corresponding secret key. For this reason, secret-key certificates can be viewed as a signature by the CA on the users secret key and thus the name becomes a little more natural.

Brands demonstrates how secret-key certificates can be constructed from any Fiat-Shamir signature scheme (see [38]). This paper will only deal with secret-key certificates based on Schnorr, since that is the scheme which Brands finally uses in his cash-system. From now on the user will once more be called Alice and the CA will be called the Bank. That is so that the similarities with the cash scheme will become clearer.

The way the protocol works is that the Bank certifies Alice's public key with an ordinary Schnorr signature. In order for the certificate to be useful to Alice, she has to modify it afterwards so that she later can prove knowledge of her secret key (let's call it a).

Alice		Bank
[SK: a]		[SK: x]
[PK: $GA = g^a$]		[PK: $GX = g^x$]
	-----GA-----	
	>	
		Random $w \in \mathbf{Z}_q \Rightarrow GW = g^w$
		$c = \text{hash}(GA, GW)$
	<----- (c, r_0) -----	$r_0 = w + c * x \pmod{q}$
	-	
$c = \text{hash}(GA, g^{r_0} / GX^c) ?$		
$r = r_0 + c * a \pmod{q}$		

Cert(GA): (c, r)

1. Alice sends her public key, GA, to the Bank for certification with its secret key x.
2. The Bank randomly chooses a number, w, and computes the commit $GW = g^w$, the challenge $c = \text{hash}(GA, GW)$ and the response $r_0 = w + c * x \pmod{q}$. Then the signature $\text{Sign}(GA) = (c, r_0)$ is sent to Alice.
3. Alice computes the commit as $GW = g^{r_0} / GX^c$ and then verifies that $c = \text{hash}(GA, GW)$. Then she computes $r = r_0 + c * a \pmod{q}$ such that $\text{Cert}(GA) = (c, r)$ becomes a secret-key certificate on GA.

Notice that (c, r) is a secret-key certificate iff the following relation holds:

$$g^r = g^{w+c*(x+a)} = GW*(GX*GA)^c$$

A secret-key certificate is hence a (Schnorr-) signature on GA and GX, i.e. the public keys of both Alice and the Bank.

It is often desirable to include more information about Alice in a certificate (e.g. social security number, IP-address etc.). Let this information be denoted by ID. The way that is done is to also include ID in the hash of the challenge c. Now ID has to be sent along with GA and (c, r) as yet another public key in order for the verification of the certificate to succeed.

Now assume that Alice have sent her public key, GA (and possibly ID), together with the certificate (c, r) to Bob so that he can verify that the Bank have certified Alice's public key. Bob then verifies the certificate by first computing the commit as $GW = g^r / (GX*GA)^c$ and then check if the challenge equals $c = \text{hash}(GA, GW)$. That's not enough though. Bob also demands of Alice that she shall prove knowledge of the secret key corresponding to her public key GA (which is easily done with Schnorr's authentication- or signature-scheme).

It is very important to point out the last statement since anyone actually can compute a public key and a certificate that matches it, without needing to interact with the Bank! Such a certified public key is not possible to receive and also knowing the corresponding secret key to it however. This is the way it would work:

Alice randomly chooses a number, s, and computes the "false" public key as $GA' = g^s / GX$. Then she selects another random number, v, and computes $GV = g^v$. These values are then used to calculate the challenge $c' = \text{hash}(GA', GV)$ and using that, Alice can easily compute the response $r' = v + c' * s \pmod{q}$. Now (c', r') is a perfectly valid secret-key certificate on GA', signed with the secret key of the Bank! Alice's problem though is that she doesn't know the secret key (the discrete logarithm) corresponding to GA', because it includes the Bank's secret key x. Specifically, $GA' = g^s / GX = g^s * g^{-x} = g^{s-x}$, so that $\log_g(GA') = s-x$.

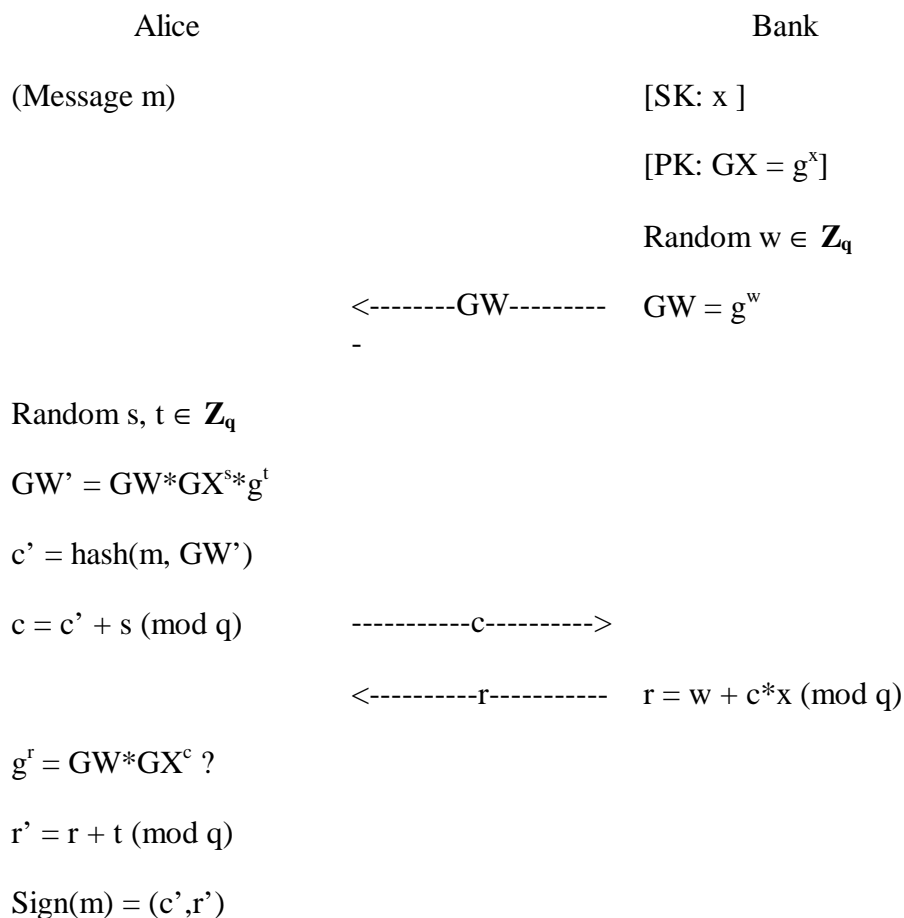
With secret-key certificates it would be possible to construct non-anonymous cheque-systems such as the Swedish transport card project (see [65]) where a payment is a signature on a file in a special format, provided by the service provider. A payment, i.e. signature (128+160 bits), public key (640 bits) and certificate (128+160 bits) would then be about 1216 bits (152 bytes) in total.

8.7. Blind Schnorr Signatures

By applying a general technique due to Okamoto and Ohta (see [59]) all signature protocols of Fiat-Shamir type (such as Schnorr) can be transformed into blind signature issuing protocols similar to the blind RSA-signatures by Chaum (see [14]).

The way it works is to keep the interaction in the protocol and let Alice decide the challenge to the Bank (the signature issuer) from a hash on the (blind) message she wishes to get signed. When the protocol is finished, Alice has received a non-interactive signature on the message of her choice. The Bank on the other hand signs with an interactive signature scheme but is not concerned with what it actually signs. It can be viewed as a reversed authentication-scheme, where the Bank identifies itself to Alice while she uses the received data to calculate a (Bank-) signature on her secret message. This clearly shows that one should be careful not to mix authentication- and signature-schemes, since it's easy to unnoticedly sign messages while identifying oneself to somebody.

The following example may not be that useful in practice for precisely that reason. It mainly serves as a tutorial example which will hopefully improve the understanding of Brands' full cash-system. It can also demonstrate how Schnorr's authentication-scheme can be abused. Brands uses a protocol by Guillou and Quisquater (see [44]) to demonstrate this technique but it may simplify things to see how it works with Schnorr.



1. The Bank randomly selects a number, w , and sends the commit $GW = g^w$ to Alice.
2. Alice randomly selects s and t and blinds the commit to $GW' = GW * GX^s * g^t$. She then uses this number together with her secret message, m , in the hashing of the challenge as $c' = \text{hash}(m, GW')$. Alice sends the blinded challenge $c = c' + s \pmod{q}$ to the Bank.
3. The Bank responds with $r = w + c * x \pmod{q}$.
4. Alice checks that $g^r = GW * GX^c$ and then she computes $r' = r + t \pmod{q}$ so that (c', r') becomes a (blind) Schnorr-signature on m .

Why does the above protocol work? Well, since the Bank never sees the message it signs, nor the final signature (c', r') , is it clear that the signing protocol is a blind signature issuing protocol. By verifying that the relation $c' = \text{hash}(m, g^{r'} / GX^c)$ holds it can be seen that (c', r') is actually a Schnorr signature on m . That this is the case becomes clear by checking that:

$$g^{r'} = g^{r+t} = g^{w+c*x} * g^t = GW * GX^c * g^t = GW * GX^{c'+s} * g^t = GW * GX^s * g^t * GX^{c'} = GW' * GX^{c'}$$

and thus $GW' = g^{r'} / GX^{c'}$, which is what Bob has to compute before he verifies the signature by checking that $c' = \text{hash}(m, GW')$.

Notice that what Alice did when she blinded the signature by the Bank was to transform the Bank's random number, w , into $w' = w + s*x + t \pmod{q}$, where s and t are random blinding numbers chosen by Alice and x is the Bank's secret key. Alice could do this without knowing either w nor x by modifying the commit GW into $GW' = GW * GX^s * g^t$.

Although the above blind signature issuing protocol makes it possible to get blind Bank signatures on freely chosen messages, is it not possible however to receive such signatures without the interaction with the Bank, i.e. it is not possible for Alice to forge the Bank's signature all by herself.

For this reason it would actually be possible to use the above protocol in designing an anonymous on-line payment system, such as David Chaum's ecash. The Bank would then have to keep all the used signatures in a database in order to prevent double spending.

8.8. Restrictive Blinding of Secret-Key Certificates

In [11], Brands demonstrates how secret-key certificates can be constructed such that they can be blinded by Alice, but in a restrictive fashion so that there is always a blinding-invariant left on the received certificate which can be used to trace Alice in case she uses her signature more than once. The whole thing can be seen as a battle between Alice and the Bank, where Alice wants to blind her identity so that the Bank won't recognise it while the Bank on the other hand wants to prevent Alice from doing it unrestrictedly.

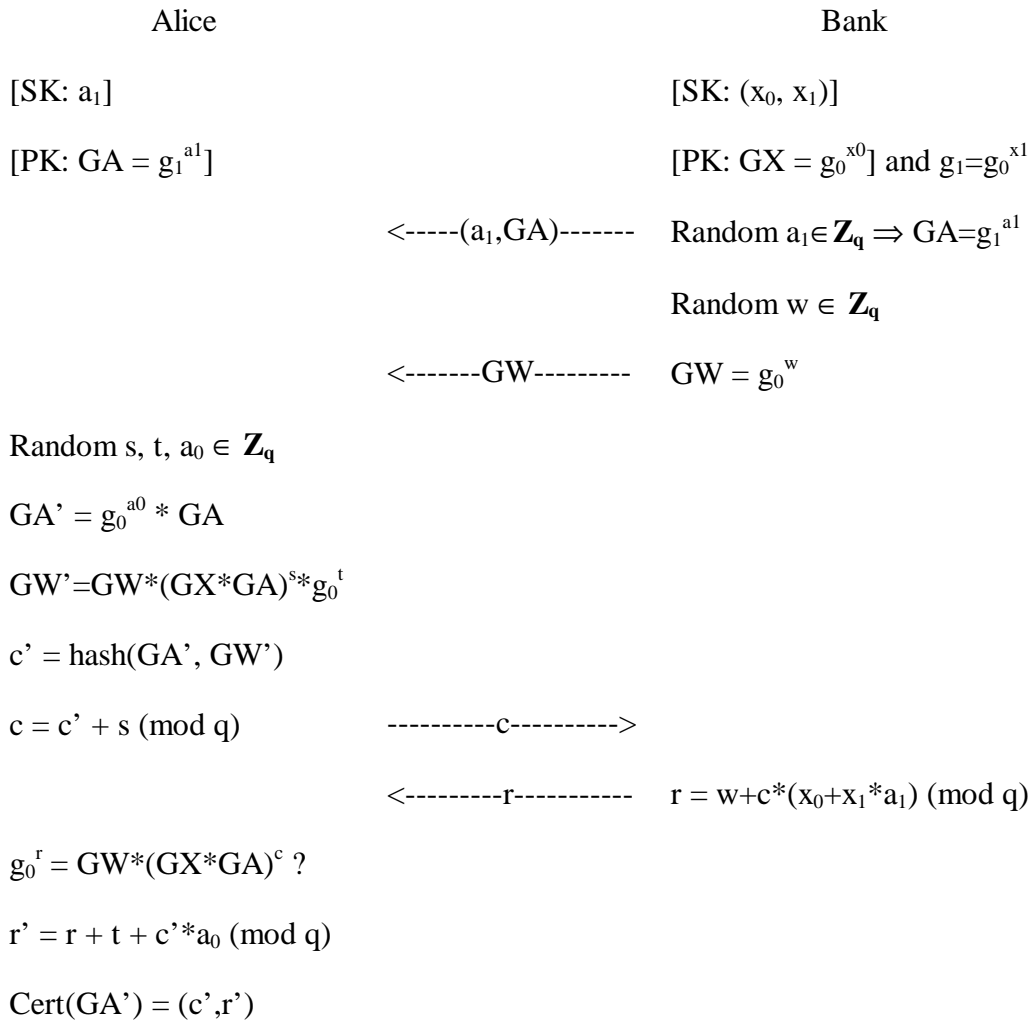
In applications such as anonymous payments, restrictive blinding may be used in a way that prevents Alice from double spending her withdrawn coin (Bank signature). If she does, the Bank receives enough information to trace Alice after the fact. Brands still uses the Guillou-Quisquater scheme to demonstrate this technique but this paper will only deal with the case of Schnorr signatures.

In the setting of this system, the Bank's secret key is the pair (x_0, x_1) and the corresponding public key is $GX = g_0^{x_0}$ (For simplicity, that is the only thing that will be called the Bank's public key, although the generators, the hash-function and the primes can also be viewed as public keys issued by the Bank). The Bank's secret key is a pair because in this case we need two generators, g_0 and g_1 , where $g_1 = g_0^{x_1}$, and only the Bank may know the correspondence between g_0 and g_1 (i.e. x_1).

It is now the Bank that picks a random secret key, a_1 , for Alice (which theoretically would allow the Bank to forge Alice's signatures) and then computes the corresponding public key as $GA = g_1^{a_1}$. The Bank then sends the key-pair (GA, a_1) to Alice. Notice that Alice only knows the g_1 -logarithm of GA (i.e. a_1) while the Bank also knows the g_0 -logarithm (i.e. $x_1 * a_1$) of GA .

The idea is now to let the Bank certify GA *in respect to* g_0 and since Alice doesn't know the discrete logarithm of GA with base g_0 she can't completely blind GA as she wants. What Alice can do is to blind GA into $GA' = g_0^{a_0} * GA$, for a randomly chosen blinding number a_0 .

Most of all, Alice should have wished to be able to blind GA into $GA * R$, where R is a random blinding factor (as in Chaum's e-cash) because then she could get a certificate by the Bank on any message, M , as she wants ($M = GA * R$). The reason why she can't do that is based on the assumption that Alice doesn't know how to compute the discrete logarithm of R in respect to g_0 . The best Alice can do to blind GA is to randomly select an a_0 and see what $GA' = g_0^{a_0} * GA$ becomes.



1. The Bank randomly selects w and sends the commit $GW = g_0^w$ to Alice.
2. Alice picks s, t, a_0 at random and then blinds her public key GA into $GA' = g_0^{a_0} * GA$. She also blinds the commit $GW' = GW * (GX * GA)^s * g_0^t$ and computes $c' = \text{hash}(GA', GW')$. Then she sends the blinded challenge $c = c' + s \pmod{q}$ to the Bank.
3. The Bank responds with $r = w + c * (x_0 + x_1 * a_1) \pmod{q}$.
4. Alice verifies the Bank's certificate on GA by checking that $g_0^r = GW * (GX * GA)^c$? She then forms the blinded response $r' = r + t + c' * a_0 \pmod{q}$ so that (c', r') becomes a secret-key certificate on GA' .

The fact that the last statement is correct can be controlled by the following calculation:

$$g_0^{r'} = g_0^{r+t+c'*a_0} = GW * (GX * GA)^c * g_0^t * g_0^{a_0 * c'} = GW * (GX * GA)^{c'+s} * g_0^t * g_0^{a_0 * c'} = GW * (GX * GA)^s * g_0^t * (GX * g_0^{a_0} * GA)^{c'} = GW' * (GX * GA')^{c'}$$

which shows that $GW' = g_0^{r'} / (GX * GA')^{c'}$ and thus $c' = \text{hash}(GA', GW')$.

9. The Cash Systems of Stefan Brands

This is practically all the details in the withdrawal protocol of Stefan Brands paper “Off-Line Electronic Cash Based on Secret-Key Certificates” (see [12]). The system that is described there uses the ‘wallet with observer’-model of Chaum and Pedersen (see [22] and [17]) and is an improvement of Brands previous systems (in [6] and [7]) in terms of efficiency, since Schnorr signatures are used instead of the signature protocol by Chaum and Pedersen. There are also systems, described by Brands, that further improves the efficiency with a slight drawback in anonymity (see [8] and [9]). Those systems are however all based on the very same basic foundation, namely secret-key certificates.

Instead of moving straight into describing Brands system (of [12]) with the Observer it might be a good idea to look at a slightly simpler system first. Therefore we shall start by describing an off-line payment system that doesn’t need any external hardware to protect against double spending (such as Chaum, Fiat and Naor’s system in [20]). But first we need to look at a concept which Brands calls the representation problem.

9.1. The Representation Problem

In the above described protocol, Alice receives a blinded public key, GA' , with the corresponding secret key (a_0, a_1) . The pair (a_0, a_1) is said to be a *representation* of GA' with respect to the generator-pair (g_0, g_1) , which means that $GA' = g_0^{a_0} * g_1^{a_1}$.

The first mentioning of the representation problem for groups of prime order was done in [19], although the term used there was ‘relaxed discrete logs’ (RDL). Brands extends the use of this problem in his CRYPTO’93 paper, [7], and specifically [6] includes a comprehensive study of the representation problem with various proofs. For groups of prime order, the representation problem is basically a generalisation of the discrete logarithm problem (DL).

Remember that the assumption about the discrete logarithm problem was it is computationally infeasible (in polynomial time) to find an x such that $h = g^x \pmod{p}$, for a given prime p and the elements g and h in the group (e.g. \mathbf{G}_q). The more general representation problem is to, given a generator-tuple $(g_1, g_2, \dots, g_k) \in \mathbf{G}_q^k$, and an element, $h \in \mathbf{G}_q$, find a representation-tuple $(x_1, x_2, \dots, x_k) \in \mathbf{Z}_q^k$, such that:

$$h = g_1^{x_1} * g_2^{x_2} * \dots * g_k^{x_k} = \prod g_i^{x_i} \quad (1 \leq i \leq k)$$

Brands provides a proof that for all $h \in \mathbf{G}_q$ and for all generator-tuples of length k , there are exactly q^{k-1} number of (different) representations of h . For the discrete logarithm problem, when there’s only one single generator, this implies that there is really only one single representation (in \mathbf{Z}_q) for every element in the group. The interesting special case would be when the generator-tuple consists of two generators. In that case there are as many as q number of different representations (remember that $q \geq 2^{160}$) for the very same element!

Brands also proves that it is as difficult to find more than one representation to the same element as it is to solve the discrete logarithm problem. The security of his system relies on this statement.

9.2. Protection Against Double Spending

The most crucial point in designing anonymous payment systems have been to prohibit multiple spendings of the same coin or cheque. Since digital money is merely zeroes and ones it can easily be copied to many different service providers. The most common ways of preventing double spending, is either to use on-line verification of the payment (e.g. ecash), or to use some sort of tamper-resistant hardware protection, such as a smart card.

The first solution is more flexible (and possibly with better “provable” security) but is very expensive in terms of communication costs, if small amounts is to be transferred. There is also a potential risk that such a solution will be rather slow when scaled up to a large system, such as the Internet. The smart card solution on the other hand, enables off-line verification but is usually not very suitable for anonymous systems since the user no longer has any control of the transmitted information.

The first entirely software based off-line payment system that was described in literature was due to Chaum, Fiat and Naor (see [20]). It is rather complicated to describe here but the way it works is that Alice gets told by Bob to answer some queries when spending her coin anonymously at his shop. The answers she gives him are enough to convince Bob that she isn't trying to cheat him and in case she does it gives the Bank enough information to trace her after the fact. Their system is unfortunately very inefficient because it uses cut-and-choose methods (as previously described).

Brands idea is to code Alice's identity into the coin in the form of a straight line (or in the general case of multiple spendable cheques, as a polynomial). When Alice then spends her coin at Bob's shop she is forced to reveal a point on that line that Bob decides. If she should have used the same coin at another shop it is most likely to have used a different point than Bob and from these two points the Bank is able to trace Alice's identity. It's a well known fact that two points is enough to unambiguously decide the equation of a line. From elementary school it is known that the equation of a straight line is:

$$y = k \cdot x + l$$

where k is the slope, l is it's intersection with y -axis and x and y are real coordinates. Knowing one point on the line, say (x_1, y_1) doesn't reveal any information about k or l , whereas knowing yet another point, (x_2, y_2) , then k and l are easily solvable as:

$$k = (y_2 - y_1) / (x_2 - x_1)$$

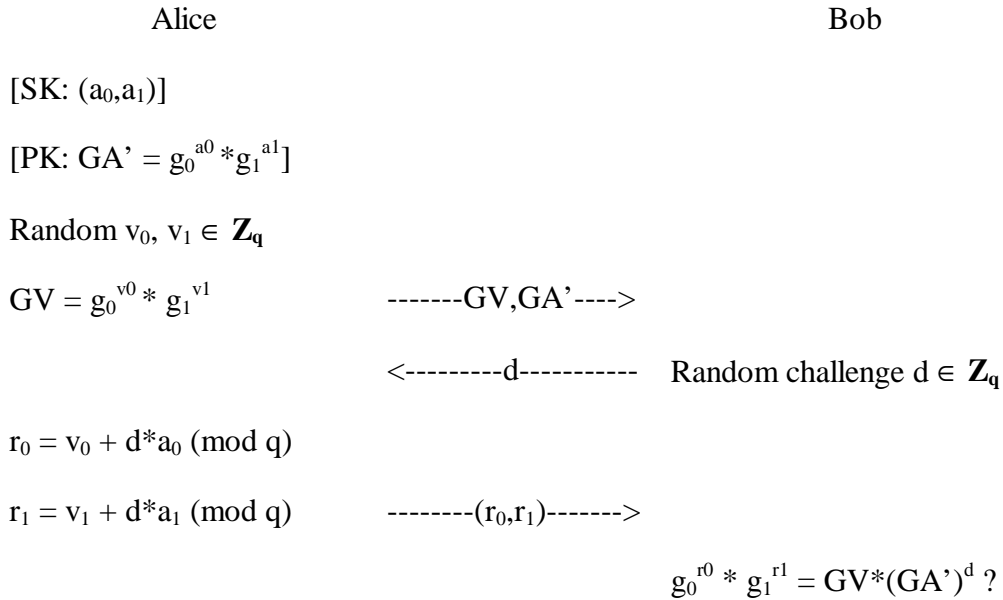
$$l = y_1 - k \cdot x_1$$

Note that the response $r = w + c \cdot x \pmod{q}$ in Schnorr's authentication- and signature-schemes is really the equation of a line with the challenge, c , acting as the 'x-value' and the response, r , as the 'y-value' (and x and w as the slope and y -axis intersection respectively).

So, how is one to go about designing a system that forces Alice to reveal such points on a line that holds her identity? What is needed is yet another protocol for demonstrating possession of discrete logarithms (possibly in zero-knowledge), but in this case for more than one generator. Such a protocol can be found in [6] which is really a generalisation of the Schnorr authentication-scheme for proving knowledge of a representation of more than one generator. We will only deal with generator-pairs in this paper however.

9.3. Proving Knowledge of a Representation

Now Alice wants to prove to Bob that she possesses a representation (secret key) that corresponds to her (blinded) public key GA' . She does not want to reveal this representation to Bob however. This is an example of a method that will work:



1. Alice picks v_0 and v_1 at random and commits to them by sending $GV = g_0^{v_0} * g_1^{v_1}$ to Bob (along with GA').
2. Bob replies with the challenge c .
3. Alice responds with $r_0 = v_0 + d * a_0 \pmod{q}$ and $r_1 = v_1 + d * a_1 \pmod{q}$.
4. Bob verifies that $g_0^{r_0} * g_1^{r_1} = GV * (GA')^d ?$

The last statement can be checked by:

$$g_0^{r_0} * g_1^{r_1} = g_0^{v_0 + a_0 * d} * g_1^{v_1 + a_1 * d} = g_0^{v_0} * g_1^{v_1} * (g_0^{a_0} * g_1^{a_1})^d = GV * (GA')^d ?$$

It may simplify things to think about the representations as a matrix, where the public keys (GA' and GV) are computed row-wise and the responses (r_0 and r_1) are taken column-wise:

	$r_0 = v_0 + d * a_0 \pmod{q}$	$r_1 = v_1 + d * a_1 \pmod{q}$
$GV = g_0^{v_0} * g_1^{v_1}$	v_0	v_1
$GA' = g_0^{a_0} * g_1^{a_1}$	a_0	a_1

Of course, this technique can be extended to include n by m matrices as well, which is also demonstrated by Brands. Such matrices makes it possible to construct cheques that can be used a fixed number of times without getting traced. Unfortunately though, all the payments a user does with such a cheque are linkable even though the user's identity is never revealed.

Anyway, Brands uses the response r_1 in the above protocol to trace a double spender since r_1 involves the blinding invariant part of Alice's secret key, i.e. a_1 . Remember that the Bank also knows a_1 .

Suppose that Alice has spent her coin (the blinded public key) at both Bob's and Cecilia's shops. The Bank then discovers that the same public key, GA' , have been used in both payments when Bob and Cecilia have requested to get credited by the Bank for the payment transcripts. Bob's and Cecilia's transcripts are however most likely distinguished by having used different challenges (d and d' respectively) which in turn implies that Alice's responses to Bob and Cecilia will differ (r_1 and r_1' respectively). The Bank has kept these numbers in its database and does hence have the equations:

$$r_1 = v_1 + d * a_1 \pmod{q}$$

$$r_1' = v_1' + d' * a_1 \pmod{q}$$

From these equations, one can easily see that if Alice could be forced to use the same random numbers in the commitment, GV (i.e. if $v_1 = v_1'$), then the Bank should be able to trace Alice's unblinded identity, a_1 , from these payment transcripts.

The way Brands solves this is to also include GV in the hashing of the challenge in the secret-key certificate as yet another (blind) public key. Notice that GV can be freely chosen (completely blinded) by Alice as opposed to GA' , which could only be formed in a certain way (restrictively blinded). That may seem a bit dangerous at first, especially if one remembers that there are as many as q number of representations of the very same number GV with respect to (g_0, g_1) ! Hence, if Alice only should happen to know more than one representation of some number GV , she could use that to make untraceable double spendings.

This is where Brands' proofs around the representation problem comes at hand. Alice simply *can't* find more than one representation for some number in polynomial time. It is as hard as solving the discrete logarithm problem (in fact, if she could find at least two representations of any given number, then she would know a way to compute discrete logarithms). The only thing that Bob has to be aware of when he accepts Alice's payments is that GV mustn't equal 1, since if it does then there are the trivial representations with respect to (g_0, g_1) , namely, $(0, 0)$, or in general, $(k*q, l*q)$, for some integers k and l .

9.4. Off-Line Cash Without Observer

The following system can be seen as Brands system with the Observer removed. It is presented here just so that his final system will seem more understandable. Notice however that the following system can't be found in any of Stefan Brands papers, although he provides all the necessary tools to construct it.

Just as with all digital payment systems, it consists of three basic protocols, namely withdrawal, payment and insertion of the digital coin. Such systems are unfortunately quite impractical by today's means due to the huge memory requirements if several coins are needed in a payment (although they possibly offer the best security).

9.4.1. The Withdrawal Protocol

The withdrawal protocol is almost identical to the protocol for restrictive blinding of secret-key certificates based on Schnorr signatures as was presented previously. The only difference is the way the challenge, c' , is computed (the parameters to the hash function).

Alice		Bank
[SK: a_1]		[SK: (x_0, x_1)]
[PK: $GA = g_1^{a_1}$]		[PK: $GX = g_0^{x_0}$] and $g_1 = g_0^{x_1}$
	<-----(a_1, GA)----->	Random $a_1 \in \mathbf{Z}_q \Rightarrow GA = g_1^{a_1}$
		Random $w \in \mathbf{Z}_q$
	<----- GW ----->	$GW = g_0^w$
Random $s, t, a_0, v_0, v_1 \in \mathbf{Z}_q$		
$GA' = g_0^{a_0} * GA$		
$GW' = GW * (GX * GA)^{s * g_0^t}$		
$GV = g_0^{v_0} * g_1^{v_1}$		
$c' = \text{hash}(GA', GV, GW')$		
$c = c' + s \pmod{q}$	----- c ----->	
	<----- r ----->	$r = w + c * (x_0 + x_1 * a_1) \pmod{q}$
$g_0^r = GW * (GX * GA)^c ?$		
$r' = r + t + c' * a_0 \pmod{q}$		
$\text{Cert}(GA', GV) = (c', r')$		

1. The Bank randomly selects w and then sends the commit $GW = g_0^w$ to Alice.
2. Alice picks s, t, a_0, v_0, v_1 at random and blinds her public key to $GA' = g_0^{a_0} * GA$, the Bank's commit to $GW' = GW * (GX * GA)^s * g_0^t$ and also forms $GV = g_0^{v_0} * g_1^{v_1}$. She hashes $c' = \text{hash}(GA', GV, GW')$ and the blinded challenge $c = c' + s \pmod{q}$ is then sent to the Bank.
3. The Bank responds with $r = w + c * (x_0 + x_1 * a_1) \pmod{q}$.
4. Alice verifies the Bank's certificate on GA and GV by checking if $g_0^r = GW * (GX * GA)^c$? Then she compute the blinded response $r' = r + t + c' * a_0 \pmod{q}$ so that (c', r') becomes a secret-key certificate on GA' (and GV).

As before, it might be useful with the following extra check:

$$g_0^{r'} = g_0^{r+t+c'*a_0} = GW * (GX * GA)^c * g_0^t * g_0^{a_0*c'} = GW * (GX * GA)^{c'+s} * g_0^t * g_0^{a_0*c'} = GW * (GX * GA)^s * g_0^t * (GX * g_0^{a_0} * GA)^{c'} = GW' * (GX * GA')^{c'}$$

Hence $GW' = g_0^{r'} / (GX * GA')^{c'}$ and $c' = \text{hash}(GA', GV, GW')$, which clearly demonstrates that (c', r') is a (Schnorr) secret-key certificate on GA' and GV .

The withdrawal protocol only involves three interactions of, apart from GW , rather small numbers. The computations that Alice has to perform may seem quite complex but many of them can actually be computed in advance. Brands systems in [8] and [9] uses these facts to improve the efficiency.

In fact, both GA' and GV can be pre-computed by Alice and “half” of GW' (the only thing missing is the Bank's commit GW). So when Alice receives GW , all she has to do is one modular multiplication, one hashing and a modular addition before she can send the challenge, c , to the Bank. Of these operations, it's only the modular multiplication that is significantly time-consuming so the other operations can be neglected. Even that operation is fast in comparison with an RSA computation, however, or compared with a modular exponentiation.

In the same manner is it possible for the Bank to have pre-computed $(x_0 + x_1 * a_1) \pmod{q}$. In [9], Brands suggests that Alice gets the id-number $ID = (x_0 + x_1 * a_1) \pmod{q}$ when she opens an account at the Bank. This means that the Bank only needs to perform one modular multiplication (the modular addition can be neglected) in real-time as well.

As before, GW is assumed to be about 640 bits and c 128 bits and r may be about 160 bits long so the total number of bits that is transmitted is approximately 928 bits (116 bytes).

9.4.2. The Payment Protocol

When Alice is to pay at Bob's shop, this is a way they could proceed. Notice that Alice is assumed to find out the format of Bob's file, Spec, on her own.

Alice	Bob
[SK: (a ₀ ,a ₁)]	
[PK: GA', GV]	
[SKC: (c',r')]	
Get Spec.	
$d = \text{hash}(GA', GV, \text{Spec})$	
$r_0 = v_0 + d * a_0 \pmod{q}$	
$r_1 = v_1 + d * a_1 \pmod{q}$	--GA',c',r',d,r ₀ ,r ₁ ->
	$GV = g_0^{r_0} * g_1^{r_1} / (GA')^d$
	GV ≠ 1 ?
	$d = \text{hash}(GA', GV, \text{Spec}) ?$
	$GW' = g_0^{r'} / (GX * GA')^{c'}$
	$c' = \text{hash}(GA', GV, GW') ?$

1. Alice gets Bob's file Spec (for instance, from a public directory) and hashes the challenge as $d = \text{hash}(GA', GV, \text{Spec})$ and then computes the responses $r_0 = v_0 + d * a_0 \pmod{q}$ and $r_1 = v_1 + d * a_1 \pmod{q}$. Finally, Alice sends the blinded public key GA', the secret-key certificate, (c', r'), on the public key and the payment signature, (d, r₀, r₁) to Bob.
2. Bob verifies Alice's signature on Spec by first computing $GV = g_0^{r_0} * g_1^{r_1} / (GA')^d$ and make sure it doesn't equal 1. Then he checks whether $d = \text{hash}(GA', GV, \text{Spec}) ?$ and when that is done he verifies the Bank's secret-key certificate on GA' and GV by comparing the challenge, c', with the expression $\text{hash}(GA', GV, g_0^{r'} / (GX * GA')^{c'})$.

This payment scheme differs slightly from Brands original protocol (see [12]) in such a way that he lets Alice send GV to Bob instead of just d. Because d is a much smaller number than GV, the above protocol is more efficient and since both GV and d contain the same information it can be argued that the security won't be affected.

In [8], Brands discusses various efficiency improvements of his protocol. He suggests a method that even avoids sending the hash value by replacing GV by (c',r') in the hashing of d and hence defining it to be $d = \text{hash}(GA', (c', r'), \text{Spec})$. This way, Bob can compute d all on his own, as well as GV.

Notice that the only seriously time-consuming operations that Alice has to perform is two modular multiplications (the hashing and the modular additions can be neglected).

In Brands original protocol (which transmits GV instead of d), the data that is sent occupies about 236 bytes ($2*640 + 128 + 3*160 = 1888$ bits). The above protocol have decreased it to be around 172 bytes ($640 + 2*128 + 3*160 = 1376$ bits) and with Brands most efficient version, only approximately 156 bytes ($640 + 128 + 3*160 = 1248$ bits) needs to be transmitted. This is also the storage requirement for the payment transcript.

9.4.3. The Deposit Protocol

When Bob have verified Alice's signature and certificate, he stores the payment transcript $[GA', (c', r'), \text{Spec}, (d, r_0, r_1)]$ in a database. Sometime later (at the end of the day or before the weekend) he wants to get credited by the Bank for the money he has received from various customers (Alice in this example).

Bob deposits his money by simply sending the payment transcript to the Bank which then is able to verify it's own certificate on Alice's (blinded) public key and that the payment signature is all right. The Bank also checks whether $(GA', GV), (c', r')$ isn't in it's database of already received coins. If it isn't, then these numbers are stored together with (d, r_1) in the database (again, GV wouldn't have to be stored since it may be derived from the other numbers).

The insertion may alternatively be performed via an Acquirer such as is described in [65].

In case Alice should have spent the same coin at both Bob's and Cecilia's shops, then they both will claim to get credited by the Bank with the same public keys and certificate. Remember that there is nothing stopping Alice from doing that since her coin is just a string of zeroes and ones. The Bank will easily detect this however in case it happens (by just comparing the public keys of the payment transcripts in it's database).

If it is assumed that every service provider (and thus also Cecilia and Bob) have unique specification files (these files need also uniquely specify each payment), then it is always possible to distinguish different transactions. Let Cecilia's specification file be denoted by Spec' and Alice's payment signature to her by (d', r_0', r_1') . Observe that Alice's payment signature to Cecilia necessarily would have to be different from Bob's if Cecilia should accept it in the first place (we also assume that the hash function used should be collision-free).

Now if the Bank have discovered that both Cecilia's and Bob's payment transcripts uses the same GA' and GV (but different challenges, d and d' respectively, and responses, r_1 and r_1'), then the Bank have two equations with two unknowns (v_1 and a_1):

$$r_1 = v_1 + d*a_1 \pmod{q}$$

$$r_1' = v_1 + d'*a_1 \pmod{q}$$

and from these Alice's secret key is easily derived as:

$$a_1 = (r_1 - r_1') / (d - d') \pmod{q}$$

This way Alice can be traced after the fact in case she double spends her (anonymous) coin.

9.5. Brands Off-Line Cash With Observers

In the system described above, Alice can anonymously pay with her digital coin as long as she doesn't spend it more than once. There is nothing preventing Alice from double spending however since she can freely copy her digital coin to as many service providers as she wishes. Even though the Bank is able to trace her after the fact, this may not be appropriate by security means. Alice can always escape before she gets caught, since it takes some time until a double spending is detected by the Bank.

The problem is even worse if this kind of system is to be used in a global community, e.g. on the Internet. How does a Bank catch a double spender that sits somewhere on the other side of the globe? There is also a problem with how to distribute the blacklisted keys to every service provider in a world-wide network.

There doesn't seem to be any other solution but to use some kind of tamper-resistant hardware, such as a smart card, to prevent double spending in advance (unless of course an on-line system is used instead) if the system should have acceptable security. The drawback of this approach is usually that now Alice no longer can be assured that her payments are anonymous, since she can't control the communication to and from the smart card and the other participants (the Bank and the service providers). Theoretically, the Bank could put some information on the smart card that reveals Alice's identity every time she makes a payment.

In order to combine the security of the smart card method with the privacy of the software solution, Chaum and Pedersen introduced a model which they called a 'wallet with observer' (see [22] and a more informal description in [17]).

An observer is basically a tamper-resistant chip or a smart card that is issued by the Bank to the users in order to prevent double spending. The way it works is that the users have to have every payment signed by the Observer and since it refuses to sign any coin it has previously signed, double spending isn't possible anymore.

The protocols are arranged in such a way that the user always have full control of the inflow and outflow of the Observer. This is to reassure that the users identity isn't revealed. The user also blinds the signatures made by the observer so that they can't be recognised and hence traced to Alice, not even by the Bank.

In Chaum and Pedersen's original protocol, there is shared information between the Observer and the service providers (or the Bank for that matter). This means that if the Bank could get hold of the Observer of some user (e.g. if it was returned for service), then all the users payments made with that Observer could be traced. If a user, be it unlikely, should be able to break the tamper-resistance of the Observer in such a way that its contents is revealed, then double spending would be possible without ever getting traced.

Brands protocol is based on the off-line system presented above which means that even if Alice should be able to break her Observer, it isn't possible for her to double spend a coin without being traced after the fact. There is also no shared information which can reveal Alice's identity in case the Observer is ever returned to the Bank.

In order to incorporate an Observer in the previous system, it's really only to let it contain Alice's private key, a_1 . Alice gets to know the public key $GA = g_1^{a_1}$ but have no idea whatsoever what the corresponding secret key, a_1 , is. For Alice now to sign payments with respect to GA , Alice have to ask the Observer of help. This is also the case when Alice is to authenticate herself to somebody (e.g. the Bank) but that goes outside the scope of this paper.

Put it simple, when the Bank certifies Alice's (blinded) key-pair it makes a signature *with respect to* g_0 , while when making a payment, her Observer signs it *with respect to* g_1 .

Alice blinds her public key, GA , into GA' and the Bank's commit, GW , to GW' , just as before but she may no longer freely choose GV . Instead, the Observer randomly selects a number, v , and sends $GV = g_1^v$ to Alice. What Alice can do though is to blind the Observer's commit, GV , into $GV' = g_0^{v_0} * g_1^{v_1} * GV * GA^u$, for some random numbers, v_0, v_1 and u .

Since Alice doesn't know the representations of GA' , nor GV' with respect to the generator tuple (g_0, g_1) , she can't sign payments with these without consulting help from the Observer. The reason why Alice doesn't know these representations is because she doesn't know a_1 or v and even if she should, she can't double spend without being traced.

The following protocols comes from [12]. Although the notations have been modified, functionally there shouldn't be anything that differs.

9.5.1. The Withdrawal Protocol

Observer	Alice	Bank
[SK: a_1]		[SK: (x_0, x_1)]
[PK: $GA = g_1^{a_1}$]		[PK: $GX = g_0^{x_0}$] and $g_1 = g_0^{x_1}$
Random $v \in \mathbf{Z}_q$		Random $w \in \mathbf{Z}_q$
$GV = g_1^v$	$GV \rightarrow$	$\leftarrow GW$ $GW = g_0^w$
(Store v)	Random $a_0, v_0, v_1, s, t, u \in \mathbf{Z}_q$	
	$GA' = g_0^{a_0} * GA$	
	$GV' = g_0^{v_0} * g_1^{v_1} * GV * GA^u$	
	$GW' = g_0^t * GW * (GX * GA)^s$	
	$c' = \text{hash}(GA', GV', GW')$	
	$c = c' + s \pmod{q}$	$\rightarrow c \leftarrow$
		$\leftarrow r \rightarrow$ $r = w + c * (x_0 + x_1 * a_1) \pmod{q}$
	$g_0^r = GW * (GX * GA)^c$?	
	$r' = r + t + c' * a_0 \pmod{q}$	

1. The Bank randomly selects a number, w , and sends the commit $GW = g_0^w$ to Alice.
2. Alice's Observer (smart card) selects v at random and sends the commit $GV = g_1^v$ to Alice. It then stores v for later use in the payment protocol.
3. Alice picks a_0, v_0, v_1, s, t, u at random and blinds the public key as $GA' = g_0^{a_0} * GA$, the Observer's commit to $GV' = g_0^{v_0} * g_1^{v_1} * GV * GA^u$ and $GW' = g_0^t * GW * (GX * GA)^s$ is her blinded version of the Bank's commit. Then she hashes the $c' = \text{hash}(GA', GV', GW')$ and sends the blinded challenge $c = c' + s \pmod{q}$ to the Bank.
4. The Bank responds with $r = w + c * (x_0 + x_1 * a_1) \pmod{q}$.
5. Alice checks that $g_0^r = GW * (GX * GA)^c$ and then computes $r' = r + t + c' * a_0 \pmod{q}$ such that (c', r') becomes a secret-key certificate on GA' (and GV').

The last statement can be verified just as before by the check:

$$g_0^{r'} = g_0^{r+t+c'a_0} = GW * (GX * GA)^c * g_0^t * g_0^{a_0 * c'} = GW * (GX * GA)^{c'+s} * g_0^t * g_0^{a_0 * c'} = GW * (GX * GA)^s * g_0^t * (GX * g_0^{a_0} * GA)^{c'} = GW' * (GX * GA')^{c'}$$

which shows that:

$$c' = \text{hash}(GA', GV', g_0^{r'} / (GX * GA')^{c'})$$

If we assume that Alice can activate her Observer in advance of the actual withdrawal protocol, then the same pre-computations as in the previous system can be performed as thus this protocol is as efficient as that one (i.e. Alice and the Bank performs one real-time modular multiplication each and the total number of exchanged bits is 928).

9.5.2. The Payment Protocol

Observer	Alice	Bob
[SK: a_1]	[SK:(a_0, v_0, v_1)]	
[PK:GA= $g_1^{a_1}$]	[PK:(GA',GV')]	
	$d = \text{hash}(GA', GV', \text{Spec})$	
	$\leftarrow d' \leftarrow d + u \pmod{q}$	
	-	
v stored?		
$y = v + d' * a_1$	$\rightarrow y \rightarrow$	
(Delete v)	$g_1^y = GV * GA^{d'}$?	GA', GV', (c', r'), (r ₀ , r ₁) - ----->
	$r_0 = v_0 + d * a_0 \pmod{q}$	
	$r_1 = v_1 + y \pmod{q}$	
		$GW' = g_0^{r'} / (GX * GA')^{c'}$
		$c' = \text{hash}(GA', GV', GW')$?
		$d = \text{hash}(GA', GV', \text{Spec})$
		$g_0^{r_0} * g_1^{r_1} = GV' * (GA')^d$?

1. Alice finds out how Bob's specification-file Spec is formed and then hashes the challenge $d = \text{hash}(GA', GV', \text{Spec})$. After that she sends the blinded challenge $d' = d + u \pmod{q}$ to her Observer (smart card).
2. The Observer checks if v is stored in memory and if it is, it replies by sending the response $y = v + d' * a_1 \pmod{q}$ to Alice. It then deletes v from memory.
3. Alice verifies that $g_1^y = GV * GA^{d'}$ and then computes the responses $r_0 = v_0 + d * a_0 \pmod{q}$ and $r_1 = v_1 + y \pmod{q}$. These are then sent by her together with the (blinded) public keys (GA', GV') and the secret-key certificate (c', r') as a payment to Bob.
4. Bob verifies the secret-key certificate by $c' = \text{hash}(GA', GV', g_0^{r'} / (GX * GA')^{c'})$ and the payment by hashing $d = \text{hash}(GA', GV', \text{Spec})$ and then check if $g_0^{r_0} * g_1^{r_1} = GV' * (GA')^d$.

Define the Observer's blinded random number to be $v_1' = v_1 + v + u * a_1 \pmod{q}$. Then:

$$r_1 = v_1 + y = v_1 + v + d' * a_1 = v_1 + v + (d+u) * a_1 = (v_1 + v + u * a_1) + d * a_1 = v_1' + d * a_1 \pmod{q}$$

$$g_0^{v_0} * g_1^{v_1'} = g_0^{v_0} * g_1^{v_1 + v + u * a_1} = g_0^{v_0} * g_1^{v_1} * GV * GA^u = GV' \dots \text{and hence:}$$

$$g_0^{r_0} * g_1^{r_1} = g_0^{v_0 + d * a_0} * g_1^{v_1 + d * a_1} = g_0^{v_0} * g_1^{v_1} * (g_0^{a_0} * g_1^{a_1})^d = GV' * (GA')^d$$

9.5.3. The Deposit Protocol

The deposit is basically identical to the corresponding protocol of the previous system. When Bob have verified Alice's signature and certificate, he stores the payment transcript, consisting of $[GA', GV', (c', r'), Spec, (r_0, r_1)]$ in a database. Sometime later (at the end of the day or before the weekend) he wants to get credited by the Bank for the money he has received from various customers (Alice in this example).

Bob deposits his money by simply sending the payment transcript to the Bank which then is able to verify it's own certificate on Alice's (blinded) public key and that the payment signature is all right. The Bank also checks whether (GA', GV') , (c', r') or $Spec$ isn't in it's database of already received coins. If it isn't, then (GA', GV') , (c', r') , $(Spec, d)$ are stored in the database (the same efficiency improvement as discussed above can be applied here as well).

In case Alice should have been able to find out her Observer's secret key, a_1 , and thus being able spend the same coin at both Bob's and Cecilia's shops, then the Bank would discover that both Cecilia's and Bob's payment transcripts uses the same GA' and GV' (but different challenges, d and d' respectively, and responses, r_1 and r_1'), so the Bank then have two equations with two unknowns (v_1 and a_1):

$$r_1 = v_1 + d * a_1 \pmod{q}$$

$$r_1' = v_1 + d' * a_1 \pmod{q}$$

and from these Alice's Observer's secret key is easily derived as:

$$a_1 = (r_1 - r_1') / (d - d') \pmod{q}$$

So even breaking the Observer is not enough to double spend undetected.

10. Acknowledgements

Many people have shown interest and have been involved in the proceeding of this final project and the following master's thesis in computer science. It is impossible to give everyone their fair credit but I will hereby make a serious attempt.

First of all I would like to thank my examiner, professor Dick Schefström at CDT - Centre for Distance-Spanning Technology of Luleå University, who although distant, have provided useful comments on pre-prints of this work via email.

I also wish to thank everyone at AU-System that have been engaged in my assignment. In particular Håkan Persson who I've had the pleasure to co-operate with in an adjacent project, but also Ulf Kilander, Bengt Ohlsson, Fredrik Gustafsson, Ulf Jonströmmer and Hans Nilsson have shown interest in what I have done.

I'm greatly indebted to the people at KTH in Stockholm, especially Mats Näslund who have functioned as my instructor and with whom I've had a lot of enlightening email discussions. Greatly appreciated are also the challenging discussions with professor Johan Håstad.

Stefan Hellberg of Telia Promotor have been involved in these discussions as well and during my collaboration with him, he has contributed with some useful comments and ideas on various parts of my thesis. In particular, the evaluations of the efficiency of different systems have been inspired by his recommendations.

Writing about digital money without mentioning David Chaum is impossible. I had the pleasure to meet him shortly at a conference and he kindly answered all the questions I had prepared for him. Later I got the opportunity to work with Niels Ferguson of DigiCash in writing a tender of the so-called 'Blue'-chip. Unfortunately nothing of that work could be included in this thesis due to its confidentiality but it gave me a lot of valuable insights.

Also Stefan Brands, himself, have provided valuable comments via email which have been very helpful for my understanding of his systems.

If it would be possible to thank such an abstract mass as the Internet community, this is where I want to do it. The discussions on the 'www-buyinfo' mailing list and on the various newsgroups on cryptography, such as 'sci.crypt', have given me faster access to relevant information than no other medium could have provided. I especially owe thanks to somebody by the name Hal Finney, somewhere on the net, for the notation in the sections that describes Brands systems.

Last but not least I would like to take the opportunity to give my appreciation to my professor and academic advisor Josh Benaloh, then at Clarkson University, Potsdam NY, USA, who first introduced me to the subject of modern cryptography. Had it not been for him, this thesis would never have been written, at least not by me.

11. References:

- [1] **Off-Line Electronic Cash**, Hans van Antwerpen, Eindhoven University of Technology, Dept. of Mathematics and Computer Science, 1990, Master's thesis.
- [2] **Groups and Symmetry**, M. A. Armstrong, Springer Verlag, 1988, Undergraduate Texts in Mathematics.
- [3] **First Bank of Internet (FBOI) Opens**, Vinn Beigh, The Risks Digest v. 16, no. 93, March 20, 1995, ACM Committee on Computers and Public Policy. Available from: <http://catless.ncl.ac.uk/Risks/16.93.html#subj11>.
- [4] **iKP - A Family of Secure Electronic Payment Protocols**, Mihir Bellare, Juan A. Garay, Ralf Hauser, Amir Herzberg, Hugo Krawczyk, Michael Steiner, Gene Tsudik, Michael Waidner, IBM T. J. Watson Research Centre & IBM Zurich Research Lab, March 1995, Working Draft. Available from: <http://www.zurich.ibm.com/Technology/Security/publications/1995/ikp.ps>.
- [5] **The ESPRIT Project CAFE - High Security Digital Payment Systems**, Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjolsnes, Frank Muller, Torben Pedersen, Birgit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Vallee, Michael Waidner, Third European Symposium on Research in Computer Security (ESORICS'94), Brighton, 1994, LNCS 875, Springer Verlag, Berlin, pp. 217-230. Available from: ftp://ftp.uni-hildesheim.de/pub/publications/Sirene/publications/BBCM1_94CafeEsorics.ps.
- [6] **An Efficient Off-Line Electronic Cash System Based on The Representation Problem**, Stefan Brands, CWI - Centrum voor Wiskunde en Informatica, 1993, Technical Report CS-R9323. Available by anonymous ftp from: <ftp.cwi.nl/pub/CWIREports/AA/CS-R9323.ps.Z>.
- [7] **Untraceable Off-Line Cash in Wallet with Observers**, Stefan Brands, Advances in Cryptology - Proceedings of CRYPTO'93, 1993, LNCS 773, Springer-Verlag, pp. 302-318.
- [8] **Off-Line Cash Transfer by Smart Cards**, Stefan Brands, CWI-Centrum voor Wiskunde en Informatica, Technical Report CS-R9455, September 1994, Available by anonymous ftp from: <ftp.cwi.nl/pub/CWIREports/AA/CS-R9455.ps.Z>. Also in: Proceedings of the First Smart Card Research and Advanced Application Conference (CARDIS'94), France, October 1994, pp. 101-117.
- [9] **Electronic Cash on the Internet**, Stefan Brands, October 1994, Will appear in: Proceedings of the Internet Society 1995 Symposium on Network and Distributed System Security, San Diego, California, February 16-17 1995, Available by anonymous ftp from: <ftp.cwi.nl/pub/brands/e-cash.ps.Z>.
- [10] **Secret-Key Certificates**, Stefan Brands, CWI-Centrum voor Wiskunde en Informatica, Manuscript (1993), Part (i), Technical Report CS-R9510, March 1995. Available by anonymous ftp from: <ftp.cwi.nl/pub/CWIREports/AA/CS-R9510.ps.Z>.

- [11] **Restrictive Blinding of Secret-Key Certificates**, Stefan Brands, CWI-Centrum voor Wiskunde en Informatica, Manuscript (1993), Part (ii), Technical Report CS-R9509, February 1995. Available by anonymous ftp from: <ftp.cwi.nl/pub/CWIREports/AA/CS-R9509.ps.Z>. To appear in: *Advances in Cryptology - Proceedings of EUROCRYPT'95*, LNCS, Springer Verlag.
- [12] **Off-Line Electronic Cash Based on Secret-Key Certificates**, Stefan Brands, CWI-Centrum voor Wiskunde en Informatica, Manuscript (1993), Part (iii), Technical Report CS-R9506, January 1995. Available by anonymous ftp from: <ftp.cwi.nl/pub/CWIREports/AA/CS-R9506.ps.Z>. To appear in: *Proceedings of the Second International Symposium of Latin American Theoretical Informatics (LATIN'95)*, Valparaiso, Chile, April 3-7, 1995.
- [13] **Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms**, David Chaum, *Communications of the ACM*, v. 24, no. 2, February 1981.
- [14] **Blind Signatures for Untraceable Payments**, David Chaum, *Advances in Cryptology - Proceedings of CRYPTO'82*, 1982, LNCS, Springer Verlag, pp. 199-203.
- [15] **Security Without Identification: transaction system to make big brother obsolete**, David Chaum, *Communications of the ACM*, v. 28, no. 10, 1985. Refined version available from: <http://www.digicash.com/publish/bigbro.html>.
- [16] **Online Cash Checks**, David Chaum, *Advances in Cryptology - Proceedings of EUROCRYPT'89*, 1989, LNCS, Springer Verlag, pp. 288-293. Available from: <http://www.digicash.com/publish/online.html>.
- [17] **Achieving Electronic Privacy**, David Chaum, *Scientific American*, August 1992. Available from: <http://www.digicash.com/publish/sciam.html>.
- [18] **Demonstrating Possession of a Discrete Logarithm Without Revealing It**, David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf, Rene Peralta, *Advances in Cryptology - Proceedings of CRYPTO'86*, 1986, LNCS, Springer Verlag, pp. 200-212.
- [19] **An Improved Protocol for Demonstrating Possession of Discrete Logarithms and Some Generalizations**, David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf, *Advances in Cryptology - EUROCRYPT'87*, 1987, LNCS, Springer Verlag, pp. 127-141.
- [20] **Untraceable Electronic Cash**, David Chaum, Amos Fiat, Moni Naor, *Advances in Cryptology - Proceedings of CRYPTO'88*, 1988, LNCS, Springer Verlag, pp. 319-327.
- [21] **Efficient Offline Electronic Checks**, David Chaum, Bert den Boer, Eugene van Heyst, Stig Mjolsnes, Adri Steenbeek, *Advances in Cryptology - Proceedings of EUROCRYPT'89*, 1989, LNCS, Springer Verlag, pp. 294-301.
- [22] **Wallet Databases with Observers**, David Chaum, Torben Pryds Pedersen, *Advances in Cryptology - Proceedings of CRYPTO'92*, 1992, LNCS 740, Springer Verlag, pp. 89-105.

- [23] **Transferred Cash Grows in Size**, David Chaum, Torben Pryds Pedersen, *Advances in Cryptology - Proceeding of EUROCRYPT'92*, 1992, LNCS, Springer Verlag, pp. 390-407.
- [24] **Identification Card Systems - Inter-Sector Electronic Purse, Draft prEN 1546, European Prestandard**, Committee Europeen de Normalisation CEN/TC224/WG10, 1992-1994, Part 1: Concepts and Structures, Part 2: Security Architecture, Part 3: Data Elements and Interchanges; Brussels, Belgium.
- [25] **CommerceNet Home Page**, CommerceNet, <http://www.commerce.net>.
- [26] **Introduction to Algorithms**, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, The MIT Press & McGraw-Hill Book Company, 1990, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.
- [27] **Improved Privacy in Wallets with Observers**, Ronald J. F. Cramer, Torben Pryds Pedersen, *Advances in Cryptology - Proceedings of EUROCRYPT'93*, 1993, LNCS, Springer Verlag, pp. 329-343.
- [28] **The CyberCash(tm) System - How it Works**, CyberCash, 1995, Available from: <http://www.cybercash.com/cybercash/cyber2.html>.
- [29] **Payment Systems and Credential Mechanisms with Provable Security Against Abuse by Individuals**, Ivan Bjerre Damgård, *Advances in Cryptology - Proceedings of CRYPTO'88*, 1988, LNCS, Springer Verlag, pp. 328-335.
- [30] **Cryptography and Data Security**, Dorothy Elizabeth Robling Denning, Addison-Wesley Publishing Company, 1983.
- [31] **New Directions in Cryptography**, Whitfield Diffie, Martin E. Hellman, *IEE Transactions on Information Theory*, v. IT-22, no. 6, November 1976.
- [32] **Home Page of ecash**, DigiCash bv. <http://www.digicash.com/ecash/ecash-home.html>.
- [33] **DigiCash Announces Cost Breakthrough in Secure Chip Technology for Smart Cards**, DigiCash bv, February 14, 1995, 'Blue' Press Release. Available from: http://www.digicash.com/publish/blue_press.html. Blue is also presented in: <http://www.digicash.com/products/blue/blue.html>.
- [34] **SNPP: A Simple Network Payment Protocol**, Seymon Dukach, *Computer Security Applications Conference*, 1992, Available from: <ftp://ana.lcs.mit.edu/pub/snpp-paper.ps>.
- [35] **Which New RSA-Signatures Can Be Computed From Certain Given RSA-Signatures?**, Jan-Hendrik Evertse, Eugene van Heyst, *Journal of Cryptology*, v. 5, n. 1, pp. 41-52, 1992
- [36] **Single Term Off-Line Coins**, Niels Ferguson, *Advances in Cryptology - Proceedings of EUROCRYPT'93*, 1993, LNCS, Springer Verlag, pp. 318-328. Preprint available by anonymous ftp from: <ftp.cwi.nl/pub/CWIreports/AA/CS-R9318.ps.Z>.
- [37] **Extensions of Single-Term Coins**, Niels Ferguson, *Advances in Cryptology - Proceedings of CRYPTO'93*, 1993, LNCS, Springer Verlag, pp. 292-301.

- [38] **How To Prove Yourself: Practical Solutions to Identification and Signature Problems**, Amos Fiat, Adi Shamir, Advances in Cryptology - Proceedings of CRYPTO'86, 1986, LNCS, Springer Verlag, pp. 186-194.
- [39] **Secure and Efficient Off-Line Digital Money**, Matthew Franklin, Moti Yung, 20th International Colloquium Automata, Languages and Programming, Proceedings (ICALP'93), July 1993, Lund, Sweden, LNCS 700, Springer Verlag, pp. 265-276.
- [40] **Payment Switches for Open Networks**, David K. Gifford, Lawrence C. Stewart, Andrew C. Payne, G. Winfield Treese, Proceedings of IEE COMPCON'95, March 1995, San Francisco, California, USA. Available from:
<http://www.openmarket.com/about/technical/compcon95.ps>.
- [41] **How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design**, Oded Goldreich, Silvio Micali, Avi Wigderson, Advances in Cryptology - Proceedings of CRYPTO'86, 1986, LNCS, Springer Verlag, pp. 171-185.
- [42] **The Knowledge Complexity of Interactive Proof Systems**, Shafi Goldwasser, Silvio Micali, Charles Rackoff, SIAM Journal of Computing, v. 18, 1989. Preliminary version in: Proceedings of the 27:th IEE Symposium on Foundations of Computer Science (FOCS'86), 1986.
- [43] **Discrete and Combinatorial Mathematics - An Applied Introduction**, Ralph P. Grimaldi, Addison-Wesley Publishing Company, June, 1989.
- [44] **A Practical Zero-Knowledge Protocol Fitted to Security Microprocessor Minimizing Both Transmission and Memory**, Louis Guillou, Jean-Jacques Quisquater, Advances in Cryptology - Proceedings of EUROCRYPT'88, 1988, LNCS 330, Springer Verlag, pp. 123-128.
- [45] **SHEN: A Security Scheme for the World Wide Web**, Phillip M. Hallam-Baker, CERN Programming Techniques Group, 1994, Available from:
<http://info.cern.ch/hypertext/WWW/Shen/ref/shen.html>.
- [46] **Anonymous One-Time Signatures and Flexible Untraceable Electronic Cash**, Barry Hayes, Advances in Cryptology - Proceedings of AUSCRYPT'90, 1990, LNCS, Springer Verlag, pp. 294-305.
- [47] **Elektroniska kontantbetalningar off-line**, Stefan Hellberg, TELE 1/94, 1994.
- [48] **Secure Socket Layer (SSL)**, Kipp E. B. Hickman, Netscape Communications Corp., February 9, 1995, Internet Draft, Available from:
<http://www.mcom.com/newsref/std/SSL.html>.
- [49] **Making Electronic Refunds Safer**, Rafael Hirschfeld, Advances in Cryptology - Proceedings of CRYPTO'92, 1992, LNCS, Springer Verlag, pp. 106-112.
- [50] **Electronic Payments over Open Networks**, P. Janson, Michael Waidner, IBM Zurich Research Laboratory, April 18, 1995, Available from:
<http://www.zurich.ibm.ch/Technology/Security/publications/1995/JaWa95.dir/JaWa95e.html>.

- [51] **Juridiskt accepterad digital signatur: utsikter och möjligheter. Examensarbete i offentlig rätt**, Mats Larsson, Juridiska fakulteten, Uppsala Universitet, 1994.
- [52] **Anonymous Credit Cards**, Steven H. Low, Nicholas F. Maxemchuk, Proceedings of the 2nd ACM Conference on Computer and Communication Security, November 2-4, 1994, Fairfax, Virginia, USA. Available by anonymous ftp from: <ftp://research.att.com:/dist/anoncc/collude.ps>.
- [53] **A Secure, Cheap, Scalable and Exportable/Importable Method for Internet Electronic Payments**, Wenbo Mao, Hewlett-Packard, May 4, 1995, Available from: <http://www.hpl.hp.co.uk/projects/vishnu/openbank.ps>.
- [54] **NetCash: A design for practical electronic currency on the Internet**, Gennady Medvinsky, B. Clifford Neuman, Proceedings of the First ACM Conference on Computer and Communications Security, November 1993. Available from: <ftp://prospero.isi.edu/pub/papers/security/netcash-cccs93.ps>.
- [55] **Mondex Home Page**, Mondex, <http://www.mondex.com/mondex/home.htm>.
- [56] **Data Encryption Standard (DES)**, FIPS PUB 46, National Bureau of Standards, January 1977, Washington, D.C. USA.
- [57] **Welcome to NetChex**, NetChex Home Page: <http://www.netchex.com>.
- [58] **Requirements for Network Payment: The NetCheque(tm) Perspective**, B. Clifford Neuman, Gennady Medvinsky, Proceedings of IEEE COMPCON'95, March 1995, San Francisco, California, USA. Available from: <ftp://prospero.isi.edu/pub/papers/security/netcheque-requirements-comcon95.ps>.
- [59] **Divertible Zero Knowledge Interactive Proofs and Commutative Random Self-Reducibility**, Tatsuaki Okamoto, Kazuo Ohta, Advances in Cryptology - Proceedings of EUROCRYPT'89, 1989, LNCS, Springer Verlag, pp. 134-149.
- [60] **Disposable Zero-Knowledge Authentications and Their Applications to Untraceable Electronic Cash**, Tatsuaki Okamoto, Kazuo Ohta, Advances in Cryptology - Proceedings of CRYPTO'89, 1989, LNCS, Springer Verlag, pp. 481-496.
- [61] **Universal Electronic Cash**, Tatsuaki Okamoto, Kazuo Ohta, Advances in Cryptology - Proceedings of CRYPTO'91, 1991, LNCS, Springer Verlag, pp. 324-337.
- [62] **How to Break and Repair a "Provably Secure" Untraceable Payment System**, Birgit Pfitzmann, Michael Waidner, Advances in Cryptology - Proceedings of CRYPTO'91, 1991, LNCS, Springer Verlag, pp. 338-350.
- [63] **The Secure HyperText Transfer Protocol (S-HTTP)**, E. Rescola, Allan Shiffman, EIT - Enterprise Integration Technologies, December 1994, Internet Draft (expires 5/95). Available by anonymous ftp from: <ftp://commerce.net:/pub/standards/drafts/shttp.txt>.
- [64] **A Method for Obtaining Digital Signatures and Public-Key Cryptosystems**, Ronald L. Rivest, Adi Shamir, Leonard M. Adleman, Communications of the ACM, v. 21, no. 2, 1978.

- [65] **Security Architecture for Electronic Purse, Draft version 3, Doc. no: ACAD 1033**, Swedish Transport Card Project: Stefan Santesson, KontoCentralen (KC), 1994.
- [66] **Applied Cryptography**, Bruce Schneier, John Wiley & Sons, 1994
- [67] **Efficient Signature Generation by Smart Cards**, Claus P. Schnorr, Journal of Cryptology, v. 4, no. 3, April 1991, pp. 161-174.
- [68] **NetBill: An Internet Commerce System Optimized for Network Delivered Services**, Marvin Sirbu, J. D. Tygar, Proceedings of IEE COMPCON'95, March 1995, San Francisco, California, USA. Available from:
<http://www.ini.cmu.edu/netbill/CompCon.html>.
- [69] **On Blind Signatures and Perfect Crimes**, Sebastiaan von Solms, David Naccache, Computers & Security, 11, 1992, pp. 581-583.
- [70] **The Green Commerce Model**, Lee H. Stein, Einar A. Stefferud, Nathaniel S. Borenstein, Marshall T. Rose, FIRST VIRTUAL Holdings Inc., October 1994, Available from: <http://www.fv.com/tech/green-model.html>.