NCash – an experimental digital cash system

Niels Möller

January 28, 1998

Abstract

An experiment with digital cash, being performed as an M.Sc. project at the Linköping Institute of Technology. The goal is to implement a bare bones digital cash system and try it out in The Real World^{TM}. The cash protocols follow Stefan Brands' ideas [2] rather closely.

Contents

1 Introduction

NCash is my master of science project at Linköping Institute of Technology. NCash is also an implementation of an off-line digital cash system, designed following Stefan Brands' report [2]. From the start, it was my intention to get the system running, and with cooperation from some traditional bank link it to the Real World$^{\text{tm}}$ money system. This has not yet happened, perhaps the time is not yet ripe.

This introductory section gives the background needed to state the goals of the NCash project. Before we can formulate the goals, we have to consider what money is, and which parties are involved in a cash system. In the next section, 2, I will give a personal account on the progress of the project and the choices that have been made.

The main part of the report, describing the mathematical tools, the design, and the protocols used, is found in sections 3 – 6. Section 7 gives an overview of the implementation, and section 8 concludes the report and also points out some possible enhancements and directions for further development.

1.1 What is money?

In order to understand what digital cash is, and what is so interesting about it, one must first consider what traditional money is. Those items made of metal or paper, what *are* they? The questions are not trivial. What does the money represent? How does it *work*? What properties does it have?

As I see it, money possess no intrinsic value. People want to exchange goods or services, to which I attribute *value* . Those things whose values are roughly equal can be exchanged, even among people who do not know or trust each other. As trade was growing, simple exchange became impractical, and *reference items* started to be used when trading. There was a consensus on the value of the reference items, and they were widely accepted in exchange for all sorts of valuable things. Let us call these reference items "coins", as metal coins is how they are usually implemented. They were invented in ancient times. Some of the properties of these coins are:

Convenience Coins provide widely standardized units of value. While large amounts of coins may be heavy to transport, they are still more easily transported than the things you buy for them.

Anonymity As a coin can be exchanged over and over again, it is practically impossible to trace.

Security Coins make robbery easier, as stolen coins are easy to use, easy to hide, and difficult to identify, compared to stolen goods. Forging coins is possible but fairly difficult, as a coin must consist of the correct weight of the correct metal in order to pass a careful examination.

As trade continued to grow, using heavy metal coins also became impractical. This is why the bank note was invented. The system is quite simple: you give the bank your coins, and in return the bank gives you a receipt where it promises to give the coins back to anyone who can produce the receipt[1]. You can use the receipt as payment for goods, as long as your trading partner trusts the bank and accepts the value of the underlying coins. For banks to be regardedas trustworthy, rules for their operation were developed, and in many countries the right to issue bank notes was given exclusively to one special bank, usually a public institution.

Bank notes were invented during the last few hundred years. Their properties are similar to those of coins. The important difference is that they are even more convenient to use, and that forging is easier as well. But there is a new risk in the system: The bank might try to defraud its customers, or simply go bankrupt.

So far, the value of coins and bank notes derives from the value of the reference items, for example fixed amounts of gold. But this has changed recently. When the banks grew, it was realized that the gold kept in the banks' vaults was pretty useless. Gold reserves are being sold out, the possibility of giving bank notes back to the bank in exchange for gold is vanishing. Even the coins are made of cheaper metals that no longer reflect their denomination.

This is a confusing situation, because in practice, nothing in the usage of money has changed, while at the same time the ancient relationship between money and reference items of known value is disappearing. When you accept a bank note in exchange for some goods or service, you do not accept it because you know that you can go to the bank and receive an amount of gold of roughly equal value. You do it because you believe that in the near future you will be able to exchange the bank note for other valuable goods. Thus, the "value" of a bank note derives from the value of the things you can exchange it for, the difference from ancient times is just that the "reference items" have been abolished.[2]

1.2 Why digital cash?

Cash is a very convenient means of exchanging goods, but it has one serious drawback in the information age: It cannot be transferred electronically. Customers have to visit a bank or an automatic teller machine to withdraw money, and shop employees have to carry the cash to a bank or a night-depository, which may not only be cumbersome, but also dangerous. The goal when developing a digital cash system is to remove this restriction,

[1] Of course, banks do more than this; they lend money, pay and receive interest, and so on. But in our quest for the nature of money, only the issuing of bank notes interests us.

[2] It could be argued that there is no real difference; that gold does have as little intrinsic value as bank notes, and that its "value" in ancient times was derived from the goods it could be exchanged for, just as bank notes today. For such a system to work, it is essential that the amount of bank notes in circulation, like the amount of gold, is limited and can increase only slowly over time. "Value" is a philosophically difficult concept.

while retaining all the good features of traditional cash. Of course, not everyone agrees on which features are "good", in particular anonymity is controversial.

The goals for the NCash project are as follows:

Convenience A digital cash system should ultimately be as convenient to use as traditional cash. However, for this project convenience of users is sacrificed for simplicity. Optimal user interfaces, smart card implementations, etc, are not considered.

Anonymity A digital coin is anonymous. When paying, the payer can remain anonymous while the payee can not. The bank should be unable to extract any information about where a user spends her money.

Efficiency It is an *off-line* system, which means that payment can be made without connecting to the bank.

Security The system should be at least as secure as traditional cash. Unfortunately, before-the-fact prevention of double spending requires some extra hardware, which is *not* implemented in this project, but can be added following Brands' description (see section 6 on preventing double-spending).

Openness The protocols used must be public, such that all parties involved can examine the protocols themselves (or ask someone they trust to examine them), and the same applies to all source code. Where practical, well known algorithms and programs are used as building blocks.

Simplicity To prove that a digital cash system providing all basic features, while not sacrificing the privacy of users, is pretty simple from a technical point of view.

It is important not to unnecessarily assume trust between different parties. Of course, some trust is necessary; if you do not trust the bank not to take your money and disappear the next day, you should not use it, and if you do not trust a shop to really give you what you pay for, you should not buy anything there. But users should not need to trust the bank in order to preserve their privacy, and shops need only trust the bank, not their customers.

I chose to start from Brands' cash system [2], for several reasons. The system satisfies NCash design goals for anonymity, efficiency and security. As I see it, it is a basic design which aims at simplicity and flexibility. Brands' system is easy to extend, for example, to several currencies and denominations, hardware observers, or checks. The protocols lack the kind of hacks or "cleverness" one would expect in a a protocol designed for one precisely defined context; there are no obscure optimizations which sacrifice simplicity and flexibility for efficiency, and which might also create unexpected vulnerabilities.

1.3 Setting the stage

In the cryptographic literature, it is customary to describe the persons participating in the protocols as individuals, rather than using anonymous labels like A, B and C. I therefore proudly present the following cast:

Bob Owner and operator of the bank.

Sarah Owner of one of the shops.

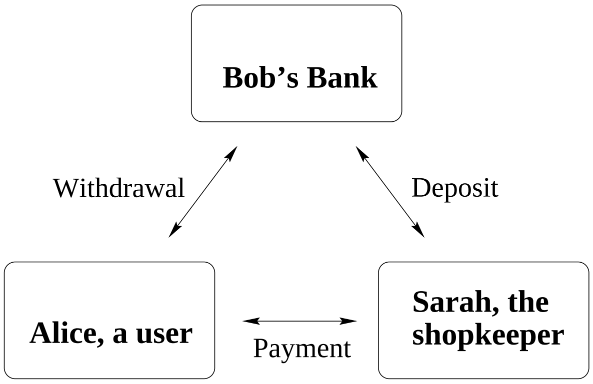
Alice One user and customer in the system. Sometimes, she wants to stay anonymous. At these times, she uses the secret pseudonym Anne.

Anne Alice's secret pseudonym.

Peggy A random user, who needs to prove her identity to Victor.

Victor A conscientious clerk verifying Peggy's identity.

Before the play begins, let us have a look at the stage.



At the back of the stage, we see Bob's bank, where the main server software is running, and where users and shops accounts are registered. In front of the bank, we see a large crowded arena with a lot of market stalls, and surrounded by shops. You suspect that the business of some of the customers and service providers is rather shady. You can feel some hostility and distrust in the air, and you would not want to visit this place after dark.

To the right we see a small but well-managed shop, and its proprietress Sarah. To the left, in the middle of the crowd, we can see Alice, one of our main characters.

2 The story of NCash

2.1 History

I have been working on NCash for quite some time, and I have been doing my studies, math teaching, working and assorted hacking at the same time. In this section, I will try to describe the progress of the NCash project, and relate it to the events that have influenced it.

The idea

My introduction to cryptography was a course given by Viiveke Fåk in the first quarter 1995. At this time, I was approaching the end of the master of science program in physics and electrical engineering, and I was about to look for a suitable master of science project. I do not remember where I got the idea of creating a digital cash system from, but the first article discussing the subject that I read was David Chaum's "Security Without Identification: Transaction Systems to Make Big Brother Obsolete" [3].

Later that spring, in April I think, I visited Mats Näslund at NADA, KTH. He provided me with some more background material and also with Brands' article [2]. It was also Mats that later gave me a copy of the CAFE [1] report.

I now knew what I wanted to do.

The plan

The plan for NCASH was to

- implement Brands' system,
- connect it to a real account in an existing bank.

The second part deserves an explanation. A small isolated cash system may be interesting, but not very useful. You have your account, you can withdraw digital cash, but no one outside the cash system, perhaps a dozen people, will accept it. However, by adding the simple service of ordering payments from your NCASH account, you get something really useful. Today, several Swedish banks have started similar services, but back in 1995, there were no secure and convenient services for making payments across the net.

Several banks offer some method for making payment orders electronically, primarily used by larger businesses. Such an account could be registered and shared by all the users of the cash system. The system would keep track of the balances of individual users. When a user wants to send a payment outside, the system verifies that the transaction is allowed, and if so it creates a payment order for the shared account and sends it to the outside bank.

Best of all: the services needed for implementing this payment service already exist. From the view of the outside bank, the cash system looks exactly like the "group accounts" large businesses use. The only difference is that the group consists not of companies, but unrelated individuals that use Brands' system for the internal transactions of the group. Another consequence is that this construction appears to be unregulated and perfectly legal; the bank legislation applies only to the "real" bank outside the cash system, not to the company or individual running the cash system.

Another factor was that in the student residential area in Ryd, the Academic Computer Society Lysator had provided permanent Internet connections to some hundred of its members. As a result, a lot of people had private real computers permanently connected to the net, which meant that a cash system could be run on the hardware installed.

This plan should not be too difficult, it would be an interesting experience, and a nice opportunity to learn network programmin, as there would be many servers and clients talking to each other.

Marketing

During the late spring and the autumn of 1995, I contacted several companies and tried to sell them the idea. I spoke with Jonas Norrbom at Nexus, Saab Combitech in Uppsala, Arne Lestad at Postgirot in Linköping, and with Leif Nilsson and Nils-Göran Albinsson at ICL in Linköping. Both Nexus and ICL are involved with data protection, development of cryptographic products and payment systems.

However, by the end of the year, it turned out that none of these companies were interested in participating or sponsoring the project, for various reasons. So I had to continue by myself, which also had the advantage that I could do it the way I wanted.

Implementing

I wrote some code during the autumn 1995, mainly to deal with the punch-card oriented message formats used by banks. Around Christmas, I learnt the interpreted language Python, and I decided that it was the right tool for the job, with nice built-in data types including bignums, a simple but straight forward object orientation and nice support for accessing the network.

Next, with help from Arne Lestad, I investigated what services Postgirot offered for transferring payment orders electronically. It turned out that payment orders and other messages are represented as a sequence of fix-sized records. Each transmission includes one record containing a keyed checksum, using the message authentication code defined in ISO-8731-2 [7].[3] Pretty simple to deal with, I even implemented some of it. I would guess that most banks use something similar.

I wrote classes, implementing the operations of the group described in section 3.5, and classes to handle unencrypted network channels as well as encryption according to the protocol of 3.3. I used Andrew Kuchling's Python Cryptographic Toolkit for the various cryptographic functions and algorithms that were needed. As far as I know, this package is not being developed any more; I don't know if anybody but I and Andrew were ever using it seriously. At about this time, March 2, I started the NCASH mailing-list.

In April, the mathematics department bought a dozen PC:s for their graduate students. It happened that I was given access to one of them. A few weeks later I got a room in the math department where I could put this computer and connect it to the net. I installed Redhat's Linux distribution on it, and moved development to this machine, from my account at Lysator which I had been using up to that point. Soon I upgraded the kernel to

[3] Some attacks on this MAC are known. Bart Preenel and Paul C. van Oorschot [9] describe how to forge MAC:s, and how to recover the secret key, using quite a large number of known and chosen messages.

version 1.99.7, which offered a randomness device based on the ideas in RFC 1750 [5], as a good random number generator is essential in most of the protocols.

I left Linköping for the summer. I returned in the middle of August and continued working. By early September, I had a simple server and client that could be used to create new accounts.

Reimplementing

Next I had to consider consider how to store information about an account, and withdrawn cash. Earlier, I had considered issues like file locking, and I had decided to do it the simple way and use the file system as a database, one or a few files for each account. But I had not really designed a format for representing the data to be saved in the files. I also realized that the problem of converting assorted objects into a byte sequence was the same whether I was storing the bytes in a database on disk, or sending them across the network while executing some protocol. Ideally, one would use some general mechanism to handle distributed and persistent objects. Python offered some support for this, but using that was out of the question due to the paranoia level of the application.

I realized that the mechanism I had been using for packaging messages for transmission across the network was not very good (I used fields separated by newline characters and colons). A large proportion of the code was making conversions of various objects to and from byte sequences, and this code was messy. (It still is, but to a much lesser degree).

I had also grown tired of Python. I didn't quite like the syntax, lacking many of the convenient constructions I was used to from C. The built-in bignums were *slow*. The module supporting the GMP bignum library was buggy, and so were some of the classes in Andrew's toolkit. I was also curious about the Pike language, created by Fredrik Hübinette. As a result, I decided to rewrite it all, this time in Pike.

A problem was that Pike was not at all as mature as Python. A first version of a class interfacing to the GMP library had just been written. This used operator overloading, another feature that had just recently been added to the language. No one had yet written any support for cryptography, like Andrew Kuchling had done for Python.

Informationsvävarna

After a few days of learning basic Pike programming, I had to learn how to extend the Pike interpreter, by writing new classes in C. I took the Python modules for the IDEA block cipher and the SHA hash function and adapted them to Pike. With this new knowledge, I applied for a job at Informationsvävarna AB[4], the company developing Pike. I started working there November 11.

[4] In October 1997, the company changed name, and is now known as Idonex AB.

Consequently, NCash was put on hold, while I was working approximately half time at Informationsvävarna developing cryptographic tools and some other things. The first goal was to write a general cryptographic toolkit. I and Henrik Grubbström designed it, and I think we created a nice object oriented framework, better than all other crypto toolkits that I have heard of. Henrik had already written a few classes, which were adapted to the new framework. By early February 1997, I and Henrik had written most of the toolkit.

Informationsvävarna's main interest in the cryptography toolkit was to support SSL, Netscape's Secure Socket Layer protocol, in their www-server Roxen Challenger. This was my next task, and also the first serious use of the crypto toolkit.

The return to NCash

By Easter, which occurred at the end of March, the SSL project at Informationsvävarna and the related cryptographic tools were finished, and I returned to working on NCash. I worked on the protocols and data storage for opening accounts and withdrawing coins, and I wrote a significant part of this report during May and early June.

During September, I implemented the remaining protocols and clients. So far, all databases had been implemented by using one file for each stored object. I partially abandoned this method when I implemented the bank's database of used coins, needed to detect double spending. Luckily, Per Hedbor and Marcus Comstedt at Informationsvävarna had developed a database that suited my needs. I incorporated it into NCash, although it is currently used only for managing the double-spending detection database. It is probably a good idea to use the same database for other objects that have to be stored on disk.

2.2 The future

At the time of this writing, there is still no connection between NCash and the outside world. There are vague plans to start using NCash for "Lysfika", the food and chocolate supply at Lysator.

Several outsiders have shown interest in the project, and it was mentioned in the January 1997 GNU Bulletin (something I did not notice until several months later). It will be interesting to see what happens when I release the code.

Export restrictions

On the negative side, I have recently been alarmed about the Wassenaar arrangements and the corresponding legislation that applies in Sweden, the European Union, and in many other countries. In short, the Wassenaar arrangements restrict export of "dual use" products: products that can be used both for mass-destruction and for non-military purposes. It is accompanied by an appendix listing sensitive products that can not be exported without a license from the authorities. This list includes missile technology, chemicals suitable for chemical weapons, and other technology that can be used for the manufacture and use of nuclear, chemical and biological weapons.

Up to this point, I have no objections. However, this list also includes any software capable of cryptography, with a few exceptions. What makes it even more bizarre, is that it includes *any* electronic computer that can be used for cryptography. Yes, you did not misread the last sentence: As far as I can interpret the law, you can not legally export any microprocessor or personal computer without a license.

What makes me feel really uneasy about this legislation are the following circumstances:

- The silence about it. The relevant legislation passed 1995. I had never heard of it until this autumn, 1997.

- Cryptography is not mentioned at all in the main text. There is not a single word explaining why cryptography is considered a tool for mass-destruction. There are a few exceptions for certain applications of cryptography and certain classes of software. However, to me these exceptions look both fuzzy and arbitrary. And they are not explained by a single word in the text.

- The sheer absurdity of it. Ftp-sites managed by Swedish authorities, for example `ftp.sunet.se`, are offering source code for strong and high quality cryptographic tools and applications. This is most likely legal, as it falls under the exception of export in "non-tangible form", and possibly others. However, if I download the same software, install it on a low-end PC, and carry it across the border to Denmark or Norway without an approved export license, I am breaking the law.

- The intent of the law is not and cannot be enforced. The intention is to make cryptography unavailable to countries that are considered "enemies". However, ftp-sites can still do this perfectly legally, and even if for example `ftp.sunet.se` was shut down, unauthorized export of software is extremely difficult to stop.

- The letter of the law is not enforced, and will most likely never be enforced. For instance, stock PC hardware, hand-held computers and cheap microprocessors cross the borders everyday, without the licenses that the legislation requires. The law *invites* capricious application.

For the time being, the Wassenaar rules seem not to apply to NCash, as distribution via anonymous ftp is one of the obscure exceptions the legislation makes. However, anyone trying to sell software with the capabilities of NCash on the international market, in "tangible" form (for example on a CD), has to apply for an export license. And licenses may legally be denied "for reasons of national security", which for most practical purposes is the same thing as arbitrarily.

3 Primitives

In this section, I give the mathematical background needed to describe NCash. I also describe some of the building blocks that the cash system and its protocols use.

3.1 Groups for Dummies

The mathematical notion of a *group* is very simple but remarkably powerful. There is plenty of literature examining and proving their properties, for example Topics in Algebra [6]. As the cash system works with group elements rather than ordinary numbers, some familiarity with groups is needed. In this section, I will define the notion of a group, give a few examples, and mention the basic properties that are needed in the sections to come.

Definition 1 (Group) *A group* $G = \langle M, \oplus \rangle$ *is a set* M *with a binary operation* $\oplus$*, such that*

- *For all* $a, b \in M$*,* $a \oplus b \in M$*.*
- *For all* $a, b, c \in M$*,* $(a \oplus b) \oplus c = a \oplus (b \oplus c)$*.*
- *There exists a unique* unit *element* $e \in M$*, such that* $e \oplus a = a \oplus e = a$ *for all* $a \in M$*.*
- *For each* $a \in M$*, there exists a unique inverse* b*, such that* $a \oplus b = b \oplus a = e$*.*

If all pairs of elements in the group satisfy the commutativity condition $a \oplus b = b \oplus a$, the group is called *Abelian* . We will not use any non-Abelian groups. If the set M has a finite number of elements, the group is said to be *finite*, and the number of elements is called the *order* of the group, and is written $O(G)$. In computer science, in cryptography, and in this report, finite groups are the ones we are interested in.

Examples

The group of integers under addition, modulo n, is defined as $Z_n = \langle \{0, 1, 2, ..., n-1\}, \oplus \rangle$, where $\oplus$ is defined by $a \oplus b = (a + b) \mod n$. We usually use the symbol $+$ instead of $\oplus$, but for clarity I will keep the $\oplus$-notation throughout this section. The unit element is zero, $e = 0$.

For example, in Z_{17}, $10 \oplus 11 = (10 + 11) \mod 17 = 21 \mod 17 = 4$, and the inverse of 10 is 7, because $10 \oplus 7 = (10 + 7) \mod 17 = 17 \mod 17 = 0 = e$.

The order of Z_n is $O(Z_n) = n$

The group of non-zero integers under multiplication, modulo p, is defined as $Z_p^* = \langle\{1, 2, 3, ..., p-1\}, \oplus\rangle$, where we define $a \oplus b = (a \cdot b)$ mod p (note that 0 is excluded). It is customary to write $\oplus$ as ordinary multiplication, but I prefer to think of the group operation as addition, mainly to get a convenient notation for exponentiation (see below).

The unit element $e = 1$. For all elements to be invertible, p must be a prime number. In this case, the inverse of any element can be computed using Euclid's algorithm, and this is quite efficient even for large p.

For example, if $p = 17$, $10 \oplus 11 = (10 \cdot 11) \mod 17 = 110 \mod 17 = 8$, and the inverse of 10 is 12, because $10 \oplus 12 = (10 \cdot 12) \mod 17 = 120 \mod 17 = 1 = e$.

The order of Z_p^* is $O(Z_p^*) = p - 1$.

Definition 2 (Subgroup) *We say that* $H = \langle N, \oplus\rangle$ *is a subgroup of* G *if* N *is a subset of* M*, and* H *is a group in itself, using the same operation* $\oplus$ *as* G.

In the finite case, this definition is equivalent to the condition that $N \subset M$ and $a \oplus b \in N$ for all $a, b \in N$.

Definition 3 (Exponentiation) *If* $a \in G$ *and* n *is a positive integer, we define the exponential*

$$n \cdot a = \underbrace{a \oplus a \oplus a \oplus \cdots \oplus a}_{n}$$

We also define $0 \cdot a = e$*. Negative powers of* a *are defined by first inverting* a*, then raising the inverse of* a *to the absolute value of* n*. If we let* $-a$ *denote the inverse of* a*, this can be expressed as*

$$(-n) \cdot a = n \cdot (-a)$$

This notation is consistent with my preference of letting addition denote the group operation. If instead the group operation were written as multiplication, one would write exponentiations as a^n.

An important feature of exponentiation is that it can be computed by repeated doubling (or squaring in multiplicative notation), and this is quite efficient, even for large n. For example, we can compute

$$\begin{aligned} 555 \cdot a &= (1 + 2 + 8 + 32 + 512) \cdot a \\ &= (1 + 2 \cdot (1 + 4 \cdot (1 + 4 \cdot (1 + 16)))) \cdot a \\ &= a \oplus 2 \cdot (a \oplus 4 \cdot (a \oplus 4 \cdot (a \oplus 16 \cdot a))) \end{aligned}$$

Given a group G and an element $a \in G$, the set of all powers of a, $\{n \cdot a | n \in Z\}$, is a subgroup of G. This subgroup is called the group *generated* by a, and is often denoted by (a). The order of this subgroup generated by a is called the order of a, and is denoted by $O(a)$.

Definition 4 (Cyclic group) *A group* G *is said to be cyclic if there exists an element* $g \in G$ *such that the subgroup generated by* g *is equal to* G*. Such a* g *is referred to as a* generator *of* G*. In other words, if* G *is cyclic, generated by* g*, then every element* $a \in G$ *can be written as* $a = k \cdot g$ *for some integer k.*

For example, Z_7^* is cyclic, generated by 5, as $k \cdot 5 = 1, 5, 4, 6, 2, 3$ when $k = 0, \ldots, 5$. For a simple example of a non-cyclic group, consider the group of pairs of bits, with element-wise XOR as the group operation. There are four elements, the unit $e = (0,0)$ and three non-unit elements $(0,1), (1,0), (1,1)$. As all elements a satisfy $a \cdot 2 = a \oplus a = (0,0) = e$, none of them is a generator.

For a finite group G, g generates G if and only if $O(g) = O(G)$.

Important properties

Some basic but important properties of groups are

1. For every element $a \in G$, $O(G) \cdot a = e$.
2. If H is a subgroup of G, then $O(H)$ divides $O(G)$.
3. As a special case, if H is generated by an element $a \in G$, it follows that $O(a)$ divides $O(G)$.
4. If a prime number p divides $O(G)$, then G has a subgroup of order p. In the case that G is cyclic, this subgroup is *unique* and consists of those elements $x \in G$ that satisfy $p \cdot x = e$.
5. If $O(G)$ is a prime number, then G is cyclic, and every element of G except the unit generates G.
6. If G is cyclic, then every subgroup of G is cyclic as well.
7. Z_n and Z_p^* are cyclic, for all positive integers n and all primes p.

3.2 Discrete logarithms

We have seen in the previous section that if we can compute $a + b$ for all elements a, b in some group G, we can almost as easily compute $n \cdot a$ for $a \in G$ and $n \in Z$. If one group operation $a + b$ takes at most t seconds, we can compute $n \cdot a$ using successive doubling, in $2t \log_2 n$ seconds or less. Here, we will always have $n < O(G)$; otherwise we first reduce n modulo $O(G)$ and take advantage of the fact that $a \cdot O(G) = 0$.

But what about the inverse problem? If we have a, b given, and want to find an integer k such that $k \cdot a = b$? Such a number, if one exists, is called a *discrete logarithm* of b with respect to a. If the group G is cyclic of prime order, and $a \neq 0$, we know that a generates G, so in this case a logarithm k exists, and it is uniquely determined except for multiples of $O(G)$.

However, no efficient algorithm is known for computing discrete logarithms in an arbitrary group. For small groups, one can generate a table of all logarithms, but this gets impractical for larger groups, say 2^{30} elements, and virtually impossible if the group order is increased to 2^{200} or more. So the method has to be adapted to the particular group in question. A particularly easy group is Z_n, where logarithms can be computed using Euclid's algorithm whenever they exist, which is easy as group addition.

When it comes to Z_p^*, there are known algorithms for computing discrete logarithms in some special cases, for example if $p - 1$ has only small factors, but for most large p, discrete logarithms cannot be computed as far as we know.

Thus, exponentiation in many groups is a *one-way function*, which is virtually impossible to invert. As one may suspect, this is an empirical statement, there is no mathematical proof whatsoever. Several public key crypto systems are based on the assumption that computing discrete logarithms is difficult .

A related problem is the so called Diffie-Hellman problem: Given $k \cdot g$ and $l \cdot g$, compute $kl \cdot g$. A solution to the discrete log problem implies a solution to the Diffie-Hellman problem, but the converse is not true, as far as is known. No efficient algorithm for solving the Diffie-Hellman problem is known.

The assumptions that no efficient algorithm exists for solving either the discrete logarithm problem or the Diffie-Hellman problem are formulated and discussed in Brands' paper [2].

3.3 Secure and anonymous network connections

There are several occasions when it may be desirable for a user Anne to connect to the bank or another user Bob, anonymously. For example, Anne may want to buy some controversial data from Bob. Or she may be willing to disclose her identity to Bob, but to Bob only. To do this, we use

- a signature algorithm S, where we assume that Anne knows Bob's public key,
- a symmetric block cipher B, used in CBC mode,
- a group with a generator g, used for a Diffie-Hellman key exchange.

The protocol, almost identical to the station-to-station protocol described in Applied Cryptography [11], looks like this:

Anne generates a random number k_1, and sends $k_1 \cdot g$ to Bob.	$k_1 \cdot g \longrightarrow$	
	$\longleftarrow k_2 \cdot g$	Bob generates a random number k_2, and sends $k_2 \cdot g$ to Anne.
Now, both Anne and Bob compute their shared session key as $k_1 \cdot (k_2 \cdot g) = k_1 k_2 \cdot g = k_2 \cdot (k_1 \cdot g)$. From now on, Anne and Bob encrypt all messages using these keys.		
	$\longleftarrow S(k_1 \cdot g, k_2 \cdot g)$	Bob signs both parties' Diffie-Hellman messages, and sends the signature to Anne (encrypted). This is to make a man-in-the-middle attack infeasible.
Anne verifies the signature, using Bob's public key.		
Anne and Bob now have an encrypted connection, and Anne know that it really is Bob at the other end.		

Known problems

To achieve real anonymity, it is not sufficient for Anne to follow this protocol; she must also be able to prevent the underlying communications channel from being traced. For instance, an ordinary TCP connection discloses the client's ip-number to the server, and the ip-number may in turn be associated with an individual or a small group of individuals. Realizing untraceable connections to the server, by using a chain of anonymizing proxies or a network with anonymous routing or some other technique, is not considered in this report.

I am aware of one rather serious security hole in the current implementation of NCASH: Messages are encrypted with a block cipher, currently IDEA, in CBC mode. Unfortunately, this has the weakness that changing certain bits in the ciphertext changes some other bits in the cleartext message in a predictable way, and this may in some cases go undetected. In order to detect such attacks, a checksum should be added to each message. An HMAC checksum (described in RFC 2104 [8]) will be added to each message next time I work on the network code.

Another option for future versions is to replace NCASH transport and key exchange protocol with the Secure Socket Layer protocol (SSL). SSL is much more complex than the current implementation, but it has the advantages of being fairly well analyzed and a de facto standard.

3.4 The representation problem

The security of the cash system is based on the difficulty of solving the *representation problem*, which is a generalization of the discrete log problem.

The representation problem in a group G is described as follows: Given $n \geq 2$ distinct non-zero elements $(g_1, g_2, \ldots, g_n)$, and a fixed element $h \in G$, find integers $(k_1, k_2, \ldots k_n)$ such that

$$h = k_1 \cdot g_1 + k_2 \cdot g_2 + \ldots k_n \cdot g_n$$

(note that the group operation is denoted by $+$). The set of g_i:s is called a *generator tuple*, and the k_i are called a *representation* of h with respect to the generator tuple.

Suppose G has prime order. Then any of the g_i generates the entire group, and it follows that every h has p^{n-1} distinct representations, where p is the order of G. If p is fairly large, this makes it highly improbable that a representation will be found by chance.

The representation problem has two features that makes it useful for cryptography. First, if it is difficult to compute discrete logarithms in G, then the only practical way to find a representation is to first choose the k_i:s and then compute the corresponding h. In fact, finding a representation for any given element h, or finding two distinct representations of an h of one's own choice, or finding a non-trivial (i.e., some $k_i \neq 0$) representation of zero, is as difficult as computing discrete logarithms in G. A sketch of a proof of these properties can be found in Brands' article [2].

Furthermore, it is possible to prove that one knows a representation, without giving it away. Assume that Peggy knows a representation a_i of h with respect to g_i, and she wants to prove this to Victor.

Peggy		Victor
Peggy chooses n integers ω_i at random, computes $$z = \sum_{1}^{n} \omega_i \cdot g_i$$ and sends z to Victor.	$z \longrightarrow$	
	$\longleftarrow c$	Victor chooses a random challenge c.
Peggy computes the responses $r_i = \omega_i + c a_i$ and sends them to Victor.	$r_i \longrightarrow$	
		Victor verifies that $$z + ch = \sum_{1}^{n} r_i \cdot g_i$$

This protocol is *zero-knowledge* : Victor doesn't gain any information about Peggy's representation whatsoever; in fact, if he makes a fake transcript of an execution of the protocol by choosing r_i and c first and then computing the corresponding z, it cannot be distinguished from a transcript of a real execution of the protocol.

3.5 A suitable group

For the cash system, we need a group with the following characteristics:

- Easy to add and invert elements.
- Elements are easily recognizable, i.e., it should be easy to decide if a given object is a member of the group.
- Prime order, this is to make all elements (except of course the unit element) generators of the entire group.
- Difficult to take logarithms.

One group that satisfies all the requirements but the last one immediately comes to mind: $\langle Z_p, +\rangle$, the group of integers modulo p, under addition. Here, and in the rest of this section, p denotes a prime number. However, taking logarithms in this group is easily done by applying Euclid's algorithm. We need a group that is similar to Z_p, but where it is difficult to take logarithms.

Let p be a (large) prime number, and choose k such that also $q = pk + 1$ is prime (k should be rather large as well). Then $\langle Z_q^*, \cdot\rangle$, the group of non-zero integers modulo q, has order pk, and therefore has a subgroup of order p. Call it G_p (it is uniquely determined because Z_q^* is cyclic).

Represent elements in Z_q with integers in the range $1, ..., q-1$, and use this also to represent the elements of the smaller set G_p. As before, I denote the group operation of G_p by '+', and exponentiation is written as $n \cdot g$, where n is either an integer, or an element of Z_p.

Now G_p satisfies our requirements, under the *discrete log assumption*. Let us investigate this:

- First, G_p is cyclic, because it is of prime order. Every element except the unit element is a generator. Let g be a generator for G_p. Actually finding (the representation of) a generator g is not too costly, as described below.
- G_p is isomorphic[5] to Z_p, the correspondence given by $k \longleftrightarrow k \cdot g$.
- A non-zero number in $x \in Z_q$ represents an element of G_p if and only if $x^p = 1 \pmod{q}$.
- Addition and subtraction is as easy as multiplication and division modulo q.
- Logarithms are difficult, as they are in Z_q^*

How do we find a generator of G_p? We look for a number $x \in \{2, 3, ...q-1\}$ such that $x^p = 1 \pmod{q}$, so that x represents an element of G_p, and $x \neq 1$ ensures that it does not represent the unit element. We start by choosing a random number $a \in \{2, 3, ..., q-1\}$. If $a^p = 1 \pmod{q}$ we take $g = a$ (or more precisely, we let g be the element of G_p that is represented by a). If not, extract the small factors of k, and try $b = a^d \mod q$ for various divisors d of k. If any such b satisfies $b \neq 1$ and $b^p = 1 \pmod{q}$, we take $g = b$. If no such b is found, start over and try another a.

4 Framework of a digital cash system

4.1 Digital cash

Not all systems capable of electronic transactions are digital *cash* systems. It is straight forward to use things like payment orders and credit cards on the net, only proper encryption and digital signatures have to be applied. Several banks and credit card companies are operating such services today. However, these systems should not be confused with digital cash. Cash, whether digital or not, has the following unique properties:

- It is anonymous.
- It is untraceable.

[5]That G_p and Z_p are *isomorphic* means that there is an invertible one-to-one map, in our case described by $k' = k \cdot g$, preserving the group structure. I.e., $k' + l' = (k + l)'$, where $k, l \in Z_p$ and $k', l' \in G_p$.

- It is withdrawn in advance. There is no such thing as an "uncovered coin" (but see section 6 on how to prevent double-spending of coins).

Coins are involved in three primitive operations:

Withdrawal A user withdraws a coin from her account at the bank.

Payment The coin is paid in a shop, in exchange for some goods or service.

Deposit The shop deposits the coin at the bank, and the shop's account is credited.

Physical coins can be used for paying an almost unlimited number of times before they are deposited; they are *transferable* . Transferable digital cash is possible, but since it is easy to connect to the bank to deposit a digital coin and withdraw a new one, one can do without transferability without much trouble.

A digital cash system can be defined as a representation for coins, and three protocols corresponding to the operations of withdrawing, paying and depositing. Of course, the difficult part is to design the representation and the protocols such that their properties correspond to those one expects of coins. Protocol design (see Public Key Cryptography [10] and Applied Cryptography [11]) is a difficult and fascinating art. For this cash system, I follow Brands' design [2].

The labels "user" and "shop" should not be taken too literally; most users will play different parts at different times. A "user" is the one withdrawing or paying, while the "shop" is the one receiving a payment or depositing a coin at the bank.

In this system, there is only one bank. Extensions to several banks are possible, but such for extensions the procedures for clearing will have to be defined. The CAFE report [1] describes one such clearing architecture. Another option is to let each bank issue their own currency, and create currency exchange servers that are connected to the cash systems of two or more banks.

Besides these protocols that are at the heart of the cash system, additional protocols may be needed for book-keeping tasks where no transaction of cash is involved. For example, opening a new account requires some handshaking of parameters.

4.2 Parameters and notation

Their are several parameters needed to describe the cash system. First, a group G_p, see section 3.5, described by the order p, the modulo q and a generator g. The cash system is based on the representation problem in this group (see section 3.4).

For signing coins, the bank has a public key consisting of a pair (g_B, h_B) of elements in G_p. The bank's secret key is the logarithm x such that $h_B = x \cdot g_B$.

Several distinct and random generators for G_p are used in the protocols. g_1 and g_2 are used as a generator pair for the users' identities and for the coins. A third generator g_3 is used when generating new accounts. And for the coins, generators are needed for denoting currencies c_i and denominations d_i.

Several distinct cash systems may be used at once, each with their own parameters and expiration dates.

4.3 Key distribution

In all public key cryptography, the distribution of keys is a key issue. In the cash system, each party has an RSA key pair consisting of a private key and an associated public key. When connecting to the bank or to another user (see section 3.3), you must know the public key of the party you are connecting to[6]. However, in the cash system, the problem of key distribution can be solved quite easily.

When a user opens a new account, she will first fetch the bank's public key from some unreliable source, for example the bank's www or `ftp` server. She uses this key when connecting to the bank for the first time. She gives the bank her name and her public RSA key, and then executes the protocol of section 5.1. So far, neither she nor the bank have any verification that the other party's public key does in fact belong to the right entity.

Next, the user will have to visit the bank's office in person. She will identify herself to the bank. As with any other account, the user and her bank enters a contract detailing the account conditions, and in this case the contract binds both parties to their public keys. When this is done, the bank activates the account that was created earlier.

This "social protocol" solves the key distribution problem for communication with the bank. But not all communication involves the bank. Users and shops have to communicate in the payment protocol, without connecting to the bank. The easiest way to distribute keys for this purpose is to have the bank publish a list of the public keys it has verified, and sign this list with its own public key. When a user need to connect to a shop, it can look up the shop's key in this list.

In other words, the bank acts as a key distribution center. Of course, if users and shops prefer some other way of distribution keys, they can use it, and in this case the shop can ask the bank not to have its key published. And it is not necessary that a shop's key, as used when customers connect to it, is the same as the key the shop uses when communicating with the bank.

[6]Actually, a user does not need an RSA key to connect to the bank, but she needs an RSA key to sign other messages, most importantly she signs receipts for the coins she withdraws.

5 Protocols

5.1 Opening a new account

Alice's identity $I \in G_p$ is an element of which she, and no one else, knows a representation. This is generated in a peculiar way, because the bank wants to ensure that the representation is truly random.

Alice generates a random index triple (x_1, x_2, x_3), computes $I' = x_1 \cdot g_1 + x_2 \cdot g_2 + x_3 \cdot g_3$ and sends it to Bob.	$I' \longrightarrow$	
	$\longleftarrow (y_1, y_2)$	Bob generates a random index pair (y_1, y_2) and sends it to Alice
Alice reveals x_3 to Bob.	$x_3 \longrightarrow$	

Both parts compute

$$I = I' + y_1 \cdot g_1 + y_2 \cdot g_2 - x_3 \cdot g_3$$

of which Alice alone knows the representation
$I = (x_1 + y_1) \cdot g_1 + (x_2 + y_2) \cdot g_2$.

5.2 Withdrawing a coin

Before the withdrawal protocol is executed, Alice sends her identity I to Bob's bank, and proves that she knows a representation of it (using the protocol of section 3.4). Then Alice withdraws one coin at a time using the following protocol.

The denomination and currency of a coin is encoded in the so called "dummy generator" d below. For each coin Alice wants to withdraw, she tells Bob which currency and denomination she wants, and they both compute the appropriate d.

Bob signs coins using a blinded version of the Schnorr signature scheme (see Brands' paper [2] and Applied Cryptography [11]). Alice blinds the message m as $m' = s \cdot m$. The number w in the signature scheme is blinded as well, to $w' = u \cdot w + v$, even though it is unknown to Alice.

The blinded m' is randomly split into two parts A and B, before the blinded and unblinded challenges c' and c are computed and sent to Bob.

	$\longleftarrow z, a, b$	$m = I + d$. Bob chooses a random number $w \in Z_p$, and computes $z = x \cdot m$, $a = w \cdot g_B$ and $b = w \cdot m$, as in the Schnorr signature scheme.
Alice chooses $s \in Z_p^*$ randomly. $m' = s \cdot m = su_1 \cdot g_1 + su_2 \cdot g_2 + s \cdot d$ and $z' = s \cdot z$. m' and its representation is randomly split in two parts; $su_1 = x_1 + y_1, su_2 = x_2 + y_2, s = x_3 + y_3$. $A = x_1 \cdot g_1 + x_2 \cdot g_2 + x_3 \cdot d$, and similarly for B. Alice chooses $u, v \in Z_p$. $a' = u \cdot a + v \cdot g_B$, $b' = su \cdot b + v \cdot m'$. The challenge $c' = H(m', z', a', b', A)$ is unblinded as $c = c'/u \pmod{p}$ and sent to Bob.	$c \longrightarrow$	
	$\longleftarrow r$	Bob responds by sending $r = (xc + w) \mod p$.
Alice checks that the relations $r \cdot g_B = c \cdot h_B + a$ and $r \cdot m = c \cdot z + b$ hold, and then computes $r' = ru + v \pmod{p}$.		

The coin is represented as (A, B, z', a', b', r'), where Alice knows representations of A, B and $m' = A + B$ with respect to (g_1, g_2, d). The tuple (z', a', b', r') acts as a Schnorr signature on (A, B).

The reason for the split in the protocol is to make possible a partial leak of information during the payment protocol. Therefore the bank can compute the numbers x_k, y_k and find out Alice's identity, if and only if she double-spends (see section 6).

5.3 Paying a coin

Before a customer pays a shop, she should first make sure that she is connected to the right shop, and that the shop is committed to delivering what she is paying for. The first problem can be solved by using the shops public key (see section 4.3), while the second problem must be dealt with by some kind of social protocol.

Payment is anonymous for the customer, unless the shop requests an identification outside the payment protocol. However, the shop cannot stay anonymous; its identity is encoded in the challenge sent to the payer, and the bank will not accept the coin for deposit unless the challenge matches the identity of the account holder.

Thus, anonymity for the shop is non-negotiable. Even if a customer for some reason *wants* to pay an anonymous shop, she cannot, if the shop ever wishes to deposit the payment at the bank. This gives the customer a somewhat stronger position, and should, for example, make anonymous black mailing less profitable.

Another advantage of encoding the shop's identity into the payment protocol is that it makes it possible to run the shop's server on a machine that is not fully trusted. A malicious system administrator or a cracker can destroy received coins, but can not steal them. Received coins can never be deposited on any account but the one the payment was meant for.

Before executing the protocol, the customer has to inform the shop which currency and denomination the coin is supposedly worth.

Anne sends A, B and the bank's signature to Sarah's shop.	$A, B, \mathrm{sign}(A, B) \longrightarrow$	
	$\longleftarrow c$	Sarah verifies that $A + B \neq 0$, and that the bank's signature is valid. She replies with a challenge $c \in Z_p^*$.
Anne responds with $r_i = (x_i + cy_i) \mod p$, $i = 1, 2, 3$.	$r_1, r_2, r_3 \longrightarrow$	
		Sarah verifies that $r_1 \cdot g_1 + r_2 \cdot g_2 + r_3 \cdot d = A + c \cdot B$.

The challenge c does not have to be random, but it must be *unique* for each payment. And in order to prevent anonymity for the shop, it must contain the shop's identity. For example, the challenge could be constructed from the shop's name and a counter or timestamp.

5.4 Depositing a coin

Depositing a coin is the simplest of the operations in the cash system.

Sarah sends a transcript of the execution of the payment protocol to Bob.	$A, B, \mathrm{sign}(A, B), c, r_1, r_2, r_3 \longrightarrow$	
		Bob verifies that the coin is not being double-spent, that the signature is correct, and that both the shop's challenge and the customers response are valid.

When accepting a deposit, the bank verifies that the payment protocol was executed properly, and that the coin has not been deposited before (see section 6). Provided that the shop's identity is contained in the payment challenge, and that invalid deposit requests are silently ignored, there is actually no need for the shop to authenticate itself to the bank when depositing a coin.

6 Double-spending

The trickiest issue in an off-line digital cash system is that of double-spending. Anne could make a backup of her electronic purse, and use the same coins over and over again. And as she is anonymous, she will not easily get caught.

6.1 Approaches to prevent double-spending

There are several more or less effective approaches to detect and prevent double-spending , some of which are:

Let the shops take all the risk. If a shop accepts a double-spent coin, it loses. Some mechanism may be provided to let shops connect to the bank to verify that coins they are about to receive are not being double-spent. This violates the definition of an "off-line" digital cash system, but could still be quite efficient if it is used only when values are large, or the customer is unknown.

Use blacklists of double-spent coins, and require shops to update their lists regularly. This should limit the amount of misuse possible with a single coin. Blacklists are not a very effective solution by themselves, but could be useful when combined with the other methods.

After-the-fact identification, which means that a user who double-spends will eventually be identified by the bank. As far as I know, this is the best that can be done as long as one does not resort to special trusted hardware or on-line payments, and it is the approach that is chosen for this project.

Trusted observers. An observer , as described by David Chaum [4] and Stefan Brands [2], is a piece of tamper-proof hardware, trusted by the bank, that is assigned to each user. The observer participates in each payment, and signs an assertion that the coin being paid has not been spent before, if and only this is the case. Interfacing the observer device to the rest of the user's system is somewhat delicate, as the observer does not trust the user it is assigned to, and vice versa. However, it can be done, without sacrificing the privacy or anonymity of users. Another cash system using observers is described in [1].

As no *tamper-proof* devices exist in the Real World$^{\mathrm{tm}}$, only *tamper-resistant* ones, one should make sure that after-the-fact identification is possible if an observer is ever successfully tampered with.

In NCash, I have chosen to use after-the-fact identification only, for simplicity. How this is done is explained below. If desired, the protocols can quite easily be extended to involve observers; this is described in Brands' paper.

6.2 After-the-fact identification of double-spenders

Recall how coins are represented (see section 5.2): The element $I + d$ is blinded to $m = s \cdot (I + d)$, where the user's public identity

$$I = u_1 \cdot g_1 + u_2 \cdot g_2 \tag{1}$$

is derived from the user's secret representation (u_1, u_2), and d is a "dummy" generator. m is then partitioned into two parts $m = A + B$, and these elements are blindly signed by the bank. In order to use a coin for paying, the user must know representations of A and B with respect to the generator triple (g_1, g_2, d), and according to Brands, the only way for the user to achieve this is to execute the withdrawal protocol properly.

When paying (see section 5.3), the shop verifies the banks signature on (A, B), and then it challenges the user's knowledge of representations. The user must produce a representation for $A + c \cdot B$; i.e. three numbers r_i such that

$$A + c \cdot B = r_1 \cdot g_1 + r_2 \cdot g_2 + r_3 \cdot d \tag{2}$$

where c is the shop's challenge.

This challenge-response pair is reported to the bank when the coin is deposited (section 5.4).

Now, let us assume that the same coin is deposited twice. The values of A and B are obviously the same. Denote the two challenges c, c' and the corresponding responses r_i, r_i', where $i = 1, 2, 3$.

First, the bank compares c and c'. If they are equal, this is not a case of double-spending by a user, it is the *shop* trying to deposit the coin twice, and the deposit is simply denied. If c and c' really are distinct, a user has double-spent a coin, and the bank has to compute that user's identity.

If x_i and y_i are the user's representations of A and B, we have that

$$s \cdot u_1 = x_1 + y_1 \tag{3}$$
$$s \cdot u_2 = x_2 + y_2 \tag{4}$$
$$s = x_3 + y_3 \tag{5}$$

Above, and for the rest of this section, we work in the field Z_p, i.e. all computations are done modulo p.

Thus, computing x_i and y_i reveals not only the user's identity, but also her private key. And the two executions of the payment protocol makes exactly this possible. From the challenge-response relations

$$r_i = x_i + cy_i \tag{6}$$
$$r_i' = x_i + c'y_i \tag{7}$$

we get

$$r_i - r_i' = (c - c')y_i \tag{8}$$
$$cr_i' - c'r_i = (c - c')x_i \tag{9}$$

Adding these equations, we get

$$(c - c')(x_i + y_i) = (c - 1)r_i' - (c' - 1)r_i \tag{10}$$

Combined with equation 1, this reveals u_1 and u_2:

$$u_i = \frac{(c-1)r_i' - (c'-1)r_i}{(c-1)r_3' - (c'-1)r_3}, \text{ for } i = 1, 2 \tag{11}$$

The denominator, s, is always invertible, as $s = 0$ implies that $A + B = m = 0$, which makes the coin invalid.

Double-spending is therefore a very unwise thing to do, not only will a double-spender get caught, but she also reveals her private key to the bank.

7 Overview of the implementation

NCash consists of several servers and clients talking to each other, and a few primitive databases for storing information about accounts and coins.

7.1 The ncashd server

At the bank, all communication with users is taken care of by a single process, ncashd. The server lets clients connect and set up an encrypted channel as described in section 3.3. After this handshaking is completed, a client requests to talk with one of the services provided:

- "echo" echoes the messages sent to it.
- "open" is used for opening new accounts.
- "withdraw" lets a user withdraw coins from her account.
- "deposit" receives used coins to be credited to the user's account.

The server uses non-blocking I/O, and can serve many clients simultaneously.

7.2 The charged servers

To be able to receive payments from users, each shop runs a charged server. Connecting to this server is similar to connecting to ncashd, except that there is only one service provided, "pay". A payment is made up of a message to the payee, and one or more coins. Each coin is paid following the protocol of 5.3 and they are stored by the shop in the file `$HOME/.ncash/paid_coins` for later deposit at the bank. The user may also request a receipt from the shop, which will include the user's message, the shop's name and the amount paid.

7.3 Databases

An account at the bank

I use the file system as a database. The bank uses a file `$NCASH_HOME/users/17` for account number 17, and a corresponding log file `$NCASH_HOME/users/17.log`. The log file is only appended to, and I use lock files to make sure that the other files stay consistent.

The information saved in the account file includes

- The type of the account (eg. "test").
- The name of the owner of the account.
- The identity I, as used in the protocols of section 5.
- An RSA public key, which the owner of the account has committed to for signing receipts.
- The current balance.
- The current status of the account.

The last item deserves an explanation. Possible values are defined in `account.h`:

```
#define ACCOUNT_DISABLED 0
#define ACCOUNT_NEW 1
#define ACCOUNT_ALLOW_DEPOSIT 2
#define ACCOUNT_DEFER_WITHDRAWAL 4
#define ACCOUNT_ALLOW_WITHDRAWAL 5
```

When a new account is created, by executing the proper protocol, the status of the account is set to ACCOUNT_NEW. After all paperwork is completed, the bank raises the account status to ACCOUNT_ALLOW_WITHDRAWAL. The bank will refuse to service withdrawal requests for accounts with a status less than ACCOUNT_ALLOW_WITHDRAWAL. While a withdrawal is in progress, the status is temporarily lowered to ACCOUNT_DEFER_WITHDRAWAL, preventing simultaneous withdrawals from the same account. The status is raised again after the bank has received a signed receipt for the withdrawal transaction. These signed receipts and other information is appended to the log file.

Networks go down, and machines crash occasionally. What happens if the connection between a user and the bank is interrupted in the middle of a withdrawal transaction? In this case, the account status will stay at ACCOUNT_DEFER_WITHDRAWAL, and the bank and user will have to work out the situation before the account is reactivated. The amount of money that could possibly be lost is limited; the user can lose at most one coin, and the bank can lose at most n coins where n is the number of accounts, or the number of simultaneous withdrawals allowed (which are currently the same).

Depositing is safer and easier than withdrawal, and is allowed if the account status is ACCOUNT_ALLOW_DEPOSIT or higher. It is possible to have an account for deposits only.

The same account from the user's point of view

At the user's end, there iss again one file for each account. Symbolic names (of the user's choosing) are used to identify the accounts. The information about an account called "foo" is stored in the file `$HOME/.ncash/foo.$$$`. This file includes all the information found in the corresponding file at the bank (as described above), except for the account status.

In addition, the bank's account number, the secret half of the RSA key, the secret representation of the user's identity, and the public and secret information associated with each withdrawn coin is stored in this file. As this information is obviously very sensitive, it is encrypted before it is stored, using a key derived from a pass-phrase.

7.4 Representation of coins

NCash supports the use of several currencies and denominations. One way to implement this is to treat each class of coins as a distinct cash system, using distinct parameters. Another way, and the way used in NCash, is to encode this information in the so called "dummy generator" d. To do this, d is chosen as follows.

There is one generator c_i for each supported currency. Denominations are powers of 2, and there is one generator d_k for each bit of the exponent. For example, for a coin representing 32 units of currency 1, we note that $32 = 2^5$, that $5 = 1 + 4 = 2^0 + 2^2$, and we therefore use the dummy generator $d = c_1 + d_0 + d_2$. Using three generators for denominations, available denominations are $1, 2, 4, \ldots, 128$. All the generators used in a particular cash system should be independently and randomly chosen.

Coins are represented as tuples (A, B, z', a', b', r'), where the last four items act as a Schnorr signature on the first two. But there is more information that should be associated with the coin. The coin should be associated with a particular bank and a particular cash system at that bank.

Thus, the public representation of a coin (i.e. that which is transmitted when paying or depositing it) includes

- The cash system it is associated with.
- The currency used.
- The exponent of the denomination.
- (A, B) and the bank's signature (z', a', b', r').

To pay with the coin, the user also has to associate some secret information with the coin. Namely the two triples x_i, y_i that are representations of A and B with respect to g_1, g_2, d, where d is the generator representing the currency and denomination, as described above.

8 Conclusions

8.1 NCash from a technical point of view

Technically, I think that NCASH went off well. There remain a few bugs that should be ironed out. Perhaps it has to be improved in order to handle large number of users and transactions; this has not been tested. But the basic system works according to the stated goals: it is a secure, anonymous and off-line digital cash system. And it is quite simple, the implementation consists of less than 7000 lines of pike code.

8.2 Non-technical problems

On the other hand, connecting NCASH to the Real World^{tm} turned out to be more difficult than I had expected. And this connection is crucial in making the system attractive to users. I did not put enough time or effort into these matters. Banks have just started Internet services, and from the admittedly few contacts I have had with them it seems they are still not ready for true digital cash.

Another important issue is making the system trustworthy. NCASH is still not well tested. I do not trust it myself, and I cannot expect other people to trust it yet. On the other hand, a larger user base is necessary both to get a higher level of testing, and to make it useful. To get a system used, one need either find a niche where it is really useful, or give new users some amount of digital cash for free.

8.3 How can digital cash succeed?

I really hope to see real digital cash being used, primarily for net commerce, and later for everyday use in ordinary shops. It would be sad if the "digital revolution" implies that anonymity is lost in a maze of different charge and credit card systems, all of them collecting information about customers' habits for marketing or other more dubious purposes.

To create the infrastructure needed for digital cash to be useful and convenient, there are at least two models.

- Technically easiest is to have a rather centralized system with only a few operating banks. The banks trust each other, and use some clearing architecture for their transactions. An example is given in the CAFE report[1].
- It is also conceivable to take a more distributed approach to operating. One may have many smaller "banks", without any central point handling clearing. Instead, currency exchange points are created: Servers that offer currency exchange between the cash of two or more banks (and the exchange rates are not necessarily 1:1).

For exchange servers to be convenient, transferability of digital cash is highly desirable (if coins are not transferable, many users would need to open accounts at several banks). It is also crucial that protocols are *open* and standardized; both user agents and exchange servers should be able to inter-operate with different banks without having to implement one set of protocols for each bank.

8.4 The future of NCash

So where does NCash enter into this picture? Perhaps NCash can show that anonymous cash is indeed possible, and it could be a useful as an example when considering a standard for transactions of digital cash. Possible extensions include use of observers (see section 6.1), and adapting it to a more general set of communication protocols, which could cope with several cash systems and other cryptographic algorithms.

Another interesting direction for future development is transferable digital cash. If one considers what transferability would imply in an off-line and anonymous system similar to NCash, one sees that the implications are non-trivial. Recall that double-spending prevention (section 6.2) requires each coin to be associated with its owner, and that some of this information is revealed in the payment protocol, in such a way that the identity is revealed completely if and only if the coin is paid twice.

Now consider a transferable coin, when it is finally deposited to the bank. The payment history of the coin handled by honest users is a linear path, and it has been received and repaid n times. When it is deposited, the bank should not be able to extract any information about any of the users that have handled it, except the identity of the user requesting the deposit.

However, if someone along the payment path of the coin double-spends, the path is no longer linear, and in general it may be a tree, with the root representing the user that originally withdrew the coin, the nodes are users, the edges are payments, and the leaves are users depositing the coin at the bank. Double-spending has occurred if and only if some node in the tree has more than one child, which happens if and only if there is more than one leaf.

Thus, if a coin is double-spent, the bank will eventually see two leaves of a payment tree. From this information, it has to find out at which node the linear payment path forked to become a tree. It should still not be able to find out the identities of the other users that have handled the coin, who are presumably honest.

To realize after-the-fact identification of double-spenders, it seems that the coin has to be transformed in each payment transaction. The secret information corresponding to the new transformed coin will encode the receiver's identity, and it is this information which will be challenged next time it is paid. And of course, the transformed coin must also include the public information of all previous incarnations of the coin, and the challenge-response information of previous payments.

I have not investigated if there are any known protocols that offer this kind of after-the-fact-identification of double-spenders for off-line anonymous cash, but I see no reason why it could not be done. It is likely that observers would make transferability easier (but after-the-fact identification is still highly desirable, as a tamper-resistant observer will be broken into sooner or later).

It is also possible that k-time transferability (for some small k) is easier than general transferability. This would mean that there can be at most $k+2$ users on the payment path before the coin must be deposited. Note that $k=1$ is sufficient for currency exchangers to operate conveniently.

References

[1] A. Bosselaers, R. Cramer, R. Michelsen, S. Mjølsnes, F. Muller, T. Pedersen, B. Pfitzmann, C. Radu, P. de Rooij, B. Schoenmakers, and M. Schunter. Functionality of the basic protocols, cafe public report. Technical report, Centrum voor Wiskunde en Informatica, October 1995.

[2] Stefan Brands. An efficient off-line electronic cash system based on the representation problem. Technical report, Centrum voor Wiskunde en Informatica, 1993.

[3] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28, October 1985.

[4] David Chaum. Achieving electronic privacy. *Scientific American*, August 1992.

[5] S. Crocker and J. Schiller. *Randomness Recommendations for Security*. Network Working Group, December 1994. RFC 1750.

[6] Herstein. *Topics in Algebra*. Wiley, second edition, 1975.

[7] ISO. *Banking – Approved algorithms for message authentication – Part 2: Message authenticator algorithm*, second edition, 1992. ISO 8731-2.

[8] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. Network Working Group, February 1997. RFC 2104.

[9] Bart Preenel and Paul C. van Oorschot. On the security of two mac algorithms.

[10] Arto Salomaa. *Public Key Cryptography*, volume 23. Springer, 1990.

[11] Bruce Schneier. *Applied Cryptography*. Wiley, second edition, 1996.

Index