CS 161 Computer Security
Fall 2005 Joseph/Tygar/Vazirani/Wagner **Notes 20**

In this lecture we will explore some issues in implementing a digital form of cash - ecash. We normally think of cash as paper money or coins issued by the treasury or a central bank. Can the assurances normally assumed for cash be made to carry over to the digital domain in the form of a sequence of bits? To explore some of the issues that arise in this context, let us consider a protocol involving three players: the customer, the merchant and the bank. The outline of the protocol we would like to implement is as follows:

The customer, Alice, interacts with the bank to withdraw some cash. She then interacts with the merchant, exchanging the cash for some goods. The merchant, Bob, then interacts with the bank to deposit the cash in his account.

A first attempt at implementing this protocol might look like this:

1. The bank sends Alice a digital \$1 bill. This might be a message signed with the bank's RSA private key saying "serial number x. This is a \$1 bill."

2. Alice sends the cash to Bob in exchange for goods.

3. Bob later deposits the cash into his account at the bank.

There are several problems with this protocol.

- It is not anonymous. The bank gets to keep a record of all of Alice's spending.
- Double spending. Since the cash is digital in nature, Alice can easily duplicate it and spend the same \$1 bill again with another merchant, Carol.

Blind Signatures:

One of the key building blocks to achieve anonymity is a blind signature. Recall that RSA signatures require the signer to compute $m^d \bmod N$, where (d,N) is the private key and (e,N) is the public key.

A blind RSA signature is carried out as follows:

- Alice sends Bob $s = (r^e m) \bmod N$ where r is a random number $\bmod N$.
- Bob computes $t = s^d \bmod N$ and sends the result to Alice.
- Alice computes $t/r \bmod N = m^d \bmod N$.

The point is that $t = s^d = (r^e m)^d = rm^d \bmod N$. A blind signature allows Alice to obtain Bob's signature on a message of her choice, without Bob having any idea what the message being signed is.

Proposal 1:

Alice and the bank will use blind signatures to collectively create a \$1 bill signed by the bank, but one which the bank will not recognize as coming from Alice.

In this scheme, a valid \$1 bill is a pair (x,y) where $y = f(x)^d \bmod N$. Here $f()$ is a one-way function (ideally a one-way hash function). (e,N) is the bank's public key and (d,N) is the private key.

- To withdraw the \$1 bill, Alice picks x, computes $f(x)$ and runs the blind signature protocol with the bank on $f(x)$. i.e. Alice picks a random $r \bmod N$, sends the bank $s = r^e f(x) \bmod N$. The bank sends back $t = s^d \bmod N$, and Alice recovers $y = f(x)^d = t/r \bmod N$.

- To pay Bob \$1, Alice sends him (x,y).

- When Bob later deposits (x,y), the bank checks that $y^e = f(x)$, and that (x,y) is not on its list of previously deposited bills.

The main feature of this scheme is that the bank cannot connect the bill (x,y) with Alice, since the blind signature was performed on a perfectly random string s.

The reason for the one-way function $f()$ is to prevent forging. For example if instead a valid \$1 bill were (x,y) where $y = x^d \bmod N$, then Alice could forge a bill by first picking $y \bmod N$ and then computing $x = y^e \bmod N$. Instead, in the protocol presented above, $y = f(x)^d$, and forging in this way would require Alice to invert the one-way function. This is because when Alice selects y at random and encrypts it, she obtains $f(x)$, instead of x which she needs to successfully forge.

One way to create ecash with several denominations is for the bank to use different RSA keys to create bills of different denominations. There is also an elegant way for the bank to achieve this while using the same composite N, but different encryption exponents e for different denominations: e.g. $e = 3$ for nickels, $e = 5$ for dimes, $e = 7$ for quarters, and $e = 11$ for dollar bills. One subtle point is that choosing $e = 9$ for a dollar bill would be a mistake. This is because under this scheme (x,y) is a valid dollar bill whenever $y^9 = f(x) \bmod N$. Now, $(x, z = y^3)$ is a valid nickel, since $z^3 = y^9 = f(x) \bmod N$. Thus choosing $e = 0$ would allow Alice to forge an extra nickel for every dollar she withdraws from the bank. Choosing the encryption exponents to be prime numbers gets around this potential problem.

One issue with this ecash scheme is that the bank has to be online at all times to identify bills, and the merchant, Bob, must cash the bill immediately, in case Alice tries to double spend. These days this is not considered a major problem, but for completeness we describe an offline scheme below.

”Offline ecash:”

The basic idea of this scheme is that instead of preventing double spending, it enables the bank to detect it. If the user does not double spend, the bank does not learn her identity. If the user double spends, the bank can compute her identity, and take suitable action.

- Alice generates $2k$ messages of the form $f(x_i)$ and $f(x_i \oplus Id)$, where x_i is selected at random. Here Id is identifying information about Alice. She sends all these values for blind signature to the bank. The point of this construction is that revealing either x_i or $x_i \oplus Id$ reveals nothing about Alice, but revealing both completely gives away her identity. The rest of the protocol is designed to check that Alice really does encode her true identity in these pairs, and that double spending reveals both of at least one pair of messages with high probability.

- The bank asks Alice to unblind k randomly chosen pairs - by revealing the corresponding x_i's and $x_i \oplus Id)$'s and the random number r_i used in the blinding protocol. The bank checcks the k chosen pairs, and checks that Id really is Alice's identifying information.

- If the k randomly chosen pairs check out, then the bank assumes that most of the remaining pairs must also be correctly created (an event that holds with very high probability), and it signs them.

- Alice thus obtains blind signatures on k pairs of messages of the form: $f(x_i)$ and $f(x_i \oplus Id)$.

To pay the merchant, Bob, Alice goes through the following protocol with him:

- Alice sends Bob the k pairs of signatures: $f(x_i)^d$ mod N and $f(x_i \oplus Id)^d$ mod N).
- Bob sends k bits $b_1,\ldots,b_k$ to Alice.
- If $b_i = 0$, Alice sends Bob x_i and if $b_i = 1$, she sends him $x_i \oplus Id$.
- Bob checks each against the signature, and accepts the bill only if they all check out.

To deposit the bill, Bob sends the bank the challenge sequence $b_1,\ldots,b_k$ and Alice's messages. If the coin is double spent, the bank also gets a second set of challenges from another merchant. With probability $1-1/2^k$, the two challenge sequences differ on some bit, say the $j-th$. But now the bank has both x_j and $x_j \oplus Id$. The bank can thus recover Id, which is the identity of the double spending Alice.

An issue that we must still address in the protocol given above is that Alice might double spend by permuting the order of the signed pairs in a single bill, or by selecting signed pairs from several bills. This can be fixed as follows: the $2k$ messages Alice generates in the first step now consist of a triple $(f(z_i), f(x_i), f(x_i \oplus Id))$, where z_i is a random number in a sufficiently large range so that collisions are unlikely. When the bank asks Alice to unbind k messages, she must reveal both x_i and z_i, and the bank checks that the z_i are distinct from each other and all previously chosen values in past bills. Finally, when Bob issues the challenge $b_1,\ldots,b_k$, Alice must reveal z_i in addition to either x_i or $x_i \oplus Id$. The point is that the z_i's provide unique identifiers for the triples, that allow the bank to identify any triple that was used before. This allows the bank to correctly reconstruct the Id of the double spender Alice.