6.857 Computer and Network Security	Fall Term, 1997
Lecture 24 : Decmeber 2, 1997	
Lecturer: Yoav Yerushalmi	*Scribe: Eytan Adar*

1 Today's Topics

- Digital Cash
- Fingerprinting

2 Digital Cash

One of our main interests in providing digital cash is that the cash should in some ways remain anonymous. That is, Alice should be able to ask the bank to generate the cash, but the bank should not be able to attribute the cash to Alice when it's done. This effect can be achieved with blinding.

Blinding - review

- Alice generates a message M and sends M to the Bank
  - trick is that M = r^d m (r is some random value)
- The Bank calculates $m_c = (r^d m)^e$ mod n and send m_c back to Alice.
- Alice divides out r: (r m^e mod n)(r^{-1} mod n) = m^e mod n, and she has a bank signed digital cash message.

Problem 1

The first problem in the basic scheme is Alice's ability to generate any message that she wants. She may tell the bank that her message is "this is $5" but instead she sends "this is $5000." The bank has no way of knowing this, and will sign either message gladly (but will be out $4995 in the latter case).

There are two solutions to this problem:

- The bank can have a limited number of possible currency values: \$1, \$5, \$10, etc. The bank will then make different signatures for each value, and will sign Alice's message with the appropriate signature no matter what the message says. This, however, limits the number of choice for currency to some finite set. Ideally, we would like to be able to get any value signed.

- The second, "nicer," solution works as follows: Alice creates 100 messages, each with the same m, but differnt random values r. She sends the messages to the bank. The bank, in turn, asks for 99 r values and decrypts the 99 messages. The bank can then verify that Alice's messages are the same, say "this is \$5." and will sign the 100th message which it did not open. The probability of Alice fooling the bank is small, and the first problem is solved.

Problem 2

The second problem is how to prevent Alice from double-spending (using the same signed cash twice). The solution to this is to use some unique serial number y, where $y \in \{0,1\}^k$. We can then find out what Alice has double spent, but...

Problem 3

Although we can now find out which bill was double-spent, Alice is still untrackable by either the merchant nor the bank. As a possible solution we introduce a new cryptographic method that allows for "holes" in the message which can be filled in after encryption has occured. So Alice's encrypted message can have 32 holes in it that can be set to either 0 or 1 at a later time. When Alice purchases something from the merchant, the merchant can ask Alice to encode a special message which the merchant generates in her payment. If the merchant keeps track of the money he receives he can make sure that Alice doesn't double spend. But this doesn't solve the general problem of Alice double-spending.

There are a number of extensions to this scheme, that can help

- The merchant can always check back with the bank before completing a transaction. However this doesn't necessarily scale because the merchant would have to keep talking to the bank for every miniscule transaction. This "check" might cost more than the value of the sale.

- We can also have a hardware solution to the problem in which we use a money card. It is then very hard to double spend if the card is smart enough

- A better method uses a scheme in which a message is divided into two halves. The message can not be reconstructed without both halves. So for example, we can take the message "I am Alice" and break it up into h_1 and h_2. If you don't have h_1 or h_2 you can't read the message. We do this by using XOR, and each h_1 and h_2 are one-time-pads. $m \oplus h_1 = h_2$.

Our new scheme now works like this:

- Let Alice's message, m = "I am Alice"
- Pick the left side, $L \in \{0,1\}^l$, and the right side is $R = m \oplus L$
- Alice generates m left messages and m corresponding right messages for each of the 100 money orders (as specified above). She commits each L_i and R_i (called the Identity strings) and sends them to the bank (using a bit-commitement scheme). She also appends the uniquness/serial number string (also discussed above) to each message.
- The bank asks Alice to open 99 of the money orders, and all the identity strings.
- As before, the bank sends back the one unopened, but signed, money order back to Alice.
- Alice goes and spends her money
- The merhant sends back an m length bit stream indicating which halves of the identity strings to open. So say at position 5, the bit stream has value 0, Alice will open L_5. If the value was 1 she would open R_5.
- The merhant takes the money order to the bank. The bank verifies that the uniqueness string is indeed unique. If not, the bank can open the identity strings and find out who's been double-spending (by taking the "spent" value and the newly submitted value, XORing them, and out pops Alice's name').

The final thing we want to add to this scheme is a timestamp to the digital cash. We want to prevent a merchant holding on to money until the bank forgets which money it has seen before (as it may have to do if it becomes expensive to store the messages indefinitely). If the merchant doesn't collect after a certain time, they lose the money.

3 Fingerprinting

Fingerprinting refers to the generalized concept of using identifying marks to identify the owner of some object. For example, mapmakers would like to prevent other mapmakers from copying their hard work. They would create fake streets our map features and hide them somewhere in their maps in some vague spot. If someone were to ever copy their maps it would be easy to point out the probability of the same fake street or feature appearing in another mapmaker's work would be very slim. More generally, we would like to be able to fingerprint maps, books, audio, movies, pictures, programs, chips, and anything else whose ownership (intellectual property we value).

For books we can hide fingerprint information by chaning the kerning (space between letters), spacing between words and lines. However, these methods don't work for things like OCRing where the text is scanned and converted into some other form. For pictures, we can enrypt messages by flipping and modifying pixels in the picture. For example, we can modify the contrast of pixel (5,10) by some value that is undetectable to the eye, but easily detectable by analyzing the picture electronically.

For programs we have a number of methods to fingerprint:

- We can add something to the end of the program that doesn't get executed.
- We can hide something in the code itself, like an array that doesn't mean anything but has values that we remember. The same technique can be applied to computer chips by hiding non-functioning parts in chip design.
- Flip NOPs and real commands in some way that doesn't change program performance but is not the "natural" compiled way.

Ideally we would like to avoid having people know about or modify fingerprint information. There are two types of fingerprinting:

- **C-frameproof** Given C evil users, they can't get together to frame another user not in their group.
- **C-secure** Given C evil users, they can't invent another working fingerprint

Let's say we have three users A, B, and C and each has something that is fingerprinted. The fingerprint is a 3 bit value, and each of A, B, and C has 100, 010, and 001 respectively. If users A and B get together and diff the two objects the have, the

only difference they will find is the first two bits. They won't be able to figure out that a signature is 3 bits, and won't be able to frame user C. Worse, if they generate something that's only two bits different, we can blame it on the only two users who could have figured out that the first 2 bits are part of a print: users A and B. The is C-frameproof but no C-secure because users A and B can generate values outside the set of they know, or can make the fingerprint 000 so that no two people can be blamed. The problem is that we require lots of bits for people (specifically, we need as many bits as we have people). The partial solution is to compress the fingerpring, which isn't perfectly secure, but can help with some probability.

Here's an example of C-secure scheme. Say we have 4 users, A, B, C, and D. Each has the following embeded fingerprint strings.

- User A: 111 111 111
- User B: 000 111 111
- User C: 000 000 111
- User D: 000 000 000

We permute and hide each of the strings randomly in the document we give to each user. If any of them gather together they will only find some limited number of "different" bits. Say users A and C work together. They will only figure out that the first 6 bits are different. If they try to invent some new message we will be able to at least catch one of them.