CS 336 Lecture 35– 5/15/98

Announcements

- I will not accept work after the last day of classes.
- I could really use a few more good 105 TA's next semester. If you might still be interested in applying, don't let the fact that the deadline has passed bother you.

Secret vs Public Key Encryption

1. Last time, we saw how to use secret keys and a KDS to ensure private communications.
2. We also discussed how public keys could be used to provide privacy and irrefutably signed digital messages if we could ensure that one could determine everyone else's public key reliably.
3. With this background on the use of public key encryption to produce signed digital documents, we can sketch out how a certified authority can allow one to determine another's public key.
    - As with a KDS, to get connected an individual or organization has to register with a certified authority by offering:
        - proof of identity
        - your public key
    - In a secret key system, each time Alice wants to talk to a new Bob, she has to make up a private key and ask the KDS to pass it on to Bob.
    - By contrast, when anyone wants to talk to Bob in a public key system they will use the same public key which Bob presented when he registered.
    - As long as Alice receives a message from the CA stating that the CA has verified that Bob is Bob and including Bob's public key she can send messages to Bob.
    - As long as the message is signed by the CA, Alice will know it came from the CA no matter how it is actually delivered. So, the CA can leave that up to Bob by handing him a "signed certificate" (what used to be called a letter of introduction) including his public key and details of his identity.
    - This means the CA is spared from becoming a bottleneck in the system. In fact, people can establish communications even while the CA is down.

Widely used encryption standards

- In case it hasn't become clear yet, what I really like about the subject of encryption is how much you can say without actually discussing how the encryption is done.

  So far, the only properties of the encryption functions we have been discussing that have been relevant are:

    - The distinction between public and secret key systems.
    - The claim that in some public key systems either key can be used for encryption as long as the other key is used for decryption.

  If you think about it, we've gone quite a distance without asking much about how the encryption is done. I think this is very good.

- Alas, I think I owe you some discussion of actual encryption functions used in the real world. So, here goes. Needless to say, I will keep it short.
- Secret Key Encryption Systems.
    - The two most common systems for secret key encryption are

      DES which stands for **data encryption standard**. This is an algorithm developed by IBM in the early 70's and then mysteriously weakened before becoming a US government standard. (There are those who suspect it was weakened just enough to make sure certain intelligence agencies could crack DES codes when necessary.)

      IDEA is a more recent (1991) standard of European roots.

    - Both DES and IDEA encrypt blocks of data 64-bits in length.

      If (as is likely) a message that is not 64-bits long is to be encrypted it must be broken up into 64-bit chunks (and padded with extra bits if necessary). In this case, it is not considered safe to simply apply the encryption process independently to each block. A more complex procedure is used so that each block is encrypted somewhat differently from the proceeding blocks. We will discuss this if time permits.

    - DES uses a 64-bit key 8 bits of which are parity bits for a "real" length of 56-bits. IDEA uses a 128-bit key. All other things being equal, it should be clear that it is harder to break IDEA than DES by brute force. In fact, while $2^5 6$ may seem like a lot of keys to us normal people, it is not considered enough for highly secure encryption given current hardware speeds.
    - When one looks at the guts of these encryption methods one comes to understand why I don't like to talk about the details much. They basically both involve phase after phase of highly unpredictable bit shifting and substring substitutions controlled in peculiar ways by the bits of the keys.

- Public Key Encryption
  - The BIG public key system is RSA (Rivest-Shamir-Adelman) encryption.
  - Unlike the secret key algorithms, RSA encryption is based on something a bit more subtle than lots of permutations and substitutions.
    * RSA encyption depends on the amazing fact that if r is the number of number less than n that are relatively prime to n, then for any number t such that t mod r = 1, $x^t modn = x$.
    * To take advantage of this, one first picks two large prime numbers p and q and chooses n = pq. By large we mean hundreds of binary digits.
    * Luckily, in this case one can conclude that the number of numbers less than n that are relatively prime to n equals (p-1)(q-1).
    * So, all we need is some t such that t mod (p-1)(q-1) is one.
    * Rather than being straightforward about this, we will look for a pair of numbers e and d such that t = ed.
    * If we first find some e such that e is relatively prime to (p-1)(q-1) then we can be sure e has a multiplicative inverse mod (p-1)(q-1). We will call this value d.
    * Given these values, we can conclude that for any x less than n, $x = x^{ed} modn$.
    * Now, we will use these facts by using $cipher = x^e modn$ as our encryption function and $x = cipher^d modn$ as the decryption function. In other words, (e,n) will be the public key and (d,n) will be the private key.
    * If n were easy to factor, it would be simple to determine d from e. However, since n is a VERY LARGE number, factoring it is very difficult. The security of the scheme rests on this fact.
  - Notice, this explains the fact that e and d are interchangeable as encryption/decryption keys.

A word about Cryptographic hashing

- Given that the semester is ending in a few minutes, I won't say a lot about cryptographic hash functions. Having mentioned them earlier, however, I feel obliged to briefly explain what they are.
- Basically, a cryptographic hash function is somewhat like a checksum. The idea is that given a message, one computes a checksum that somehow summarized the message in a way that makes it unlikely that another message produced randomly (or deliberately) will have the same checksum.
- The extra properties that make a cryptographic has different from a checksum are that:
  - Given one messages hash it should be hard to find another message with the same hash.
  - It should be hard to find any two messages with the same hash.
- The best known use of such functions is password encrypting.
  - It is dangerous to store passwords in a file that could later be read (even if reading it would require decryption).
  - Instead, many systems only store "hashes" of the passwords. When a password is entered, it is hashed and the result is compared to the stored hash.
  - Since it is hard to find a password with a given hash, being able to read the file does not help one guess passwords.
- Hashes are also sometimes used to reduce the cost of "signing" messages. Instead of actually signing the message, one can just sign a hash of the message. If hashing is cheaper than the encryption scheme used for signing this saves effort.

Digital Cash

- What I really like about encryption is the fact that clever uses of encryption can make seemingly impossible forms of communication feasible.
- One potentially nasty feature of using credit cards rather than cash is that the credit card company can keep a record of everything you buy using your card.
- As catalogs go on-line, it seems likely that more and more purchases will be paid for using electronic devices and credit-card like systems.
- The interesting question is whether there is anyway to produce the equivalent of digital cash. That is, provide a way that you can go to your "bank", withdraw money from your account, and in return receive a message which you can spend without identifying yourself but which you can only spend once.
- The key to solving this problem is to think of the fact that current paper currency carries both a serial number and a denomination. We will devise a scheme by which a bank can give you a digital message with both a serial number and a denomination. Things just need to be arranged so that the bank knows who you are when it gives you the denomination, but does not know your identity when giving you the serial number.

- Part of the secret to the solution is to have the bank use several keys. One key will be used to sign serial numbers. In addition, the bank will have a series of other key pairs used to sign "bills" of different denominations.

- The final secret is to assume the encryption and decryption processes are communative, even when using several distinct keys. That is, if you and I both encrypt a message, we can apply our decryption algoritms in either order to get the plaintext back.

- The steps are roughly:

  - Have the bank provide a server that will respond to anonymous requests for signed serial numbers.
  - Have the bank also provide a withdrawal server.

    To make a withdrawal a customer must:

    * Identify him/herself.
    * Specify the amound to withdraw (in one of a relatively small number of provided denominations).
    * Present a signed serial number encrypted with a key unknown to the bank.

    The bank must:

    * Verify that you have the money,
    * Sign the encrypted certificate with the key for the denomination requested.

    The customer must then decrypt the message so that it is now simply double signed by the bank.

    Anyone you try to pay with this money can:

    * Use the banks public keys to determine the denomiations.
    * Ask the bank if it is still valid by sending the bank the serial number.
    * Deposit it in their own account by sending the message to the bank and identifying themselves. (At this point, the bank will mark the serial number as "spend/invalid").