Encryption Applications

Encryption and the Internet
Secure Socket Layer (SSL)
Security of the Diebold electronic voting systems
Reading: Brookshear (4.5)

Recap

- Important concepts in encryption
  - the goal is to “randomize” the original text \(\text{plaintext}\)
  - the encrypted text \(\text{ciphertext}\) can be transmitted over the Internet or stored in a database
- Encryption and decryption algorithms use a key
  - \(c = f(m, K)\)
  - different key \(K\) => different ciphertext \(c\)
- Key management is crucial
  - anyone who learns a key can decrypt a piece of ciphertext
    (assuming they know the algorithm, which is usually not a secret)
- State of the art encryption/decryption algorithm:
  - block ciphers created with DES (standard from 1976 to 2002) or AES (current standard)

Recap

- A breakthrough in 1976: \textit{public key cryptography}
  - uses asymmetric keys: one for encryption, one for decryption
- The RSA method is widely used
  - keys are defined in terms of \(N\), the product of two large (2000-bit) prime numbers
  - if someone can figure out how to factor \(N\) they can break the code
- encryption: \(c = m^e \mod N\)
  (where \(e\) is the encryption key)
- decryption: \(m = c^d \mod N\)
  (where \(d\) is the decryption key)
- can be used to encrypt strings of up to 4000 bits (number of bits in \(N\))

Secure Communication on the Internet

- Data encryption is very common on the Internet today
  - entering credit card numbers at amazon.com and other “e-tail” sites
  - typing passwords when connecting to remote servers
  - backup and remote storage (e.g. .Mac)
- Example: secure shell
  - suppose I want to run a program at the UO computer center from my office:
    - shell prompt on fug (my office computer)
    - shell prompt from UOCC system
    - everything I type in my office is encrypted by ssh before it is sent to the UOCC system
Secure Communication on the Internet

- Other examples of encrypted communication include
  - sftp — a secure version of ftp, the file transfer program
    - has a command line interface: connect to remote server, list files, upload or download files, ...
  - https — a secure version of http (hypertext transfer protocol)
    - requests sent to server and responses sent back are encrypted

- Example: using https to buy a book from O’Reilly
  - start by connecting to their home page
  - note this uses the normal (insecure) http protocol

Secure Communication on the Internet

- After I add a bunch of stuff to my shopping cart I click the View Cart icon

SSL

- Secure communication on the Internet is implemented by SSL
  - SSL = secure socket layer
  - an additional layer in the TCP/IP protocols
  - newer version: TLS
  - SSL runs on both the client and the server
  - An application protocol can send a record to SSL, which encrypts it before sending it to TCP
  - On the server the SSL layer decrypts a message received by TCP

browsers often indicate when you are using https
SSL

- SSL begins with a “handshaking” protocol
  - the client sends a message to the server, telling it which encryption methods it knows
  - the server selects one of the methods and returns its public key $K$
  - the client generates a random number $R$
  - the client encrypts $R$ with $K$, returns it to the server
  - client and server use $R$ to create session keys
- Messages in the remainder of the session are encrypted with these keys

Man in the Middle

- There is a problem with this scheme -- how does the client know it is really talking to the server it intends to communicate with?
  - in cryptography this is known as the “man in the middle” problem
  - Example: suppose you click on a link in an e-mail message the text says “click here to update your account at Oregon Community Credit Union” but the URL actually takes you somewhere else

Certificates

- The solution is to have the server send a certificate that proves its identity
  - the server's public key is returned as part of a package that contains a certificate
  - certificates are “signed” by a trusted authority (CA)
  - a business delivers its public key and other identifying info to the CA
  - if it wants to, the client can contact the CA to make sure the certificate is valid
  - Verisign is an example of a CA

OS/X Keychain

- If you have an OS/X system you can find certificates on your system with Keychain Access
OS/X Keychain

- Double-click on a certificate to see its details

Aside: traceroute

- If you’re curious about the path your network packets take on the way from your system to a remote host you can use traceroute
  - should be installed as a command line program on POSIX systems

```
[fugu:conery] % traceroute www.oreilly.com
  1 vl-6.uonet1-gw.uoregon.edu (128.223.6.2) ... 
  2 0.ge-0-1-0.uonet8-gw.uoregon.edu (128.223.3.8) ... 
  3 vl-205.ge-2-0-0.core0-gw.pdx.oregon-gigapop.net ... 
  4 vl-201.abilene-losa-gw.oregon-gigapop.net ... 
  5 so-0-0-0.0.rtr.hous.net.internet2.edu (64.57.28.45) ... 
  14 gig49.dist1-1.sr.sonic.net (209.204.191.30) ... 
  15 ora-demarc.customer.sonic.net (64.142.122.36) ... 
  16 www.oreillynet.com (208.201.239.36) ...
```

Security and Diebold

- Diebold was sharply criticized for the way they handled security in their GEMS software
  - see sections 4.4 and 4.5 of the paper by Kohno, et al
- Security experts who analyzed the system found many places where encryption or other security measures should have been used but were not
  - example: the system did not look for checksums or digital signatures on the “smart cards”
  - anyone could write a program on a card, insert the card, and have their program control the system
- Where Diebold did use encryption it was very poorly implemented

The Diebold software downloaded from the Internet was written in C

- The symmetric encryption key was defined inside the software
  - the C program had the equivalent of this Ruby statement:
    ```ruby
des_key = “F2654hD4”
```
- There are several problems with this design:
  - every system in the country used the same key
  - a hacker who discovered this key could gain access to data encrypted by any system
  - "hardwiring" passwords and other security information is very poor practice
    - the digital equivalent of leaving a note for your friends on your front door, saying “the key is under the welcome mat”
  - the same key had been used since 1998
    - we’ll talk about how Kohno et all figured this out next week, when we talk about software engineering and the CVS source code control system
Security and Diebold

- Even if the source code is not available, it is very easy to find strings in compiled C programs
  - an example, using a program I wrote for one of my research projects:

```c
[fintan:src] % strings realign.o
vector::M_fill_insert
column lists:
Usage: realign [options] < file
md_syscmd(.//version.rb)
vector::M_insert_aux
specify either -g or -i but not both
gamma must be between 0 and 1
alpha must be greater than 0
--help
Copyright (C) 2007 University of Oregon.
...
```

Potential Future Problems

- Suppose Diebold fixes these flaws and any others that remain
  - assume GEMS'08 and other software is up to date and uses the latest in encryption standards
  - There are still potential security problems
  - Here is one example:
    - to keep the voting machines secure they are configured and then disconnected from the internet
    - on election day votes are encrypted with RSA and written to a storage device
    - at the end of the day the encrypted votes are sent to the county office for tabulating
    - It might still be possible to figure out how a person voted
      - if the votes are recorded in order, someone monitoring the voting place might associate a voter with their encrypted vote
      - they could watch the order in which people voted, and correlate that with the order of votes in the file

Potential Future Problems

- What good would that do?
  - if the votes are encrypted they just look like random numbers
  - The only thing the watcher has learned is
    - A's vote is 8171277103293069225
    - B's vote is 4589046621106005910
    - etc
  - But there only a few candidates in each election
    - RSA.encrypt("clinton")
    - 8171277103293069225
    - RSA.encrypt("obama")
    - 4589046621106005910
    - These will be the only two (or two most common) numbers in the file

Potential Future Problems

- This is a variation on what cryptographers call a "chosen plaintext attack"
  - the code-breaker initially doesn't know how each vote is encrypted
  - but there are a limited number of plaintexts (candidate names)
  - the code-breaker eventually learns which code corresponds to each candidate (e.g. one might get 55% of the vote, so find out which number appears 55% of the time)
  - This scenario is easily fixed
    - encrypt other information along with the candidate name
      - e.g. prefix a candidate's name with the voter number
        - RSA.encrypt("1obama")
        - 2126310943527971245
        - RSA.encrypt("2clinton")
        - 26851402470649252
        - RSA.encrypt("3obama")
        - 348014133149898019
Potential Future Problems

- Another solution is to write the encrypted votes in a random order
  - this problem was one of the issues mentioned in the Kohno paper
  - the Diebold system recorded votes serially
- The main point:
  - security is a very subtle problem
  - there are very strong technical solutions for each individual part of the system
  - **secure parts do not guarantee a secure system**
  - security and privacy have to be considered at each step in the design and verification of the entire system
- More on this topic next week when we look at software engineering....

A Peek Ahead

- One of the software engineering methods we’ll discuss is known as **open source software**
  - instead of a group keeping their source code private they release it to the world
  - others can look at it, test it, modify it, ...
- Diebold and other organizations don’t want to release source code
  - they are worried others will steal their company secrets
  - they also argue hackers will have an easier time breaking in
- Which approach is right?
- My opinion: open source
  - better to have “friendly” opposition find flaws now than have them exposed later
  - “security by obfuscation” is not a viable approach to such a critical process