

Security

Books

The Code Book, by Simon Singh. Interesting history of cryptography.

Web Security and Commerce, by Simpson Garfinkle. Use and ramifications of cryptography on the Internet.

Secrecy

Bytes are intercepted -- prevent the interceptor from learning anything from the bytes

Bytes could be on a socket

Bytes could just be a file

Authenticity

Identify -- someone is who they claim to be

An Internet connection -- who is it?

A file -- who authored it?

Tech: Symmetric Cryptography

Classical encryption with a "key"

Key is a shared secret known to both parties

The key is used for both encryption and decryption

vs. Interception (secrecy)

vs. Forged document (authentication)

Problem: key distribution -- how do you arrange the shared secret in the first place? Historically, this is a huge problem -- e.g. the German Enigma encryption machine and its daily keys in WWII

Application: Simple Cryptography

Terms

p = plain text (readable)

$\text{crypt}()$ = encryption/decryption function

c = cipher text (garbled)

k = key (essentially, a password)

Alice sends message to Bob

Alice computes $c = \text{crypt}(p, k)$;

Alice sends c over Internet, store in file etc. to Bob

Bob computes $p = \text{crypt}(c, k)$

The point: works because only Alice and Bob know the shared secret k

Application: Password Login

A wants to log in to their account on B

A and B both know the shared secret -- the password

A sends the password to B, to prove A's identity -- authenticity

Problem: password can be intercepted -- "sniffing" -- this is the source of most security problems at Stanford

Solution1: challenge-response scheme

Don't send the password itself, the A just needs to prove that they have the password. B sends random R to A. A computes $\text{hash}(R + \text{password})$ and returns to B, proving that client knows the password.

Solution2: create a secure channel between the A and B, then do the password dialog (see below)

Many old protocols (telnet, FTP), send the password in the clear. There are newer variants that avoid this problem.

Tech: "Public Key" Cryptography

One of the most amazing developments in applied mathematics in recent memory. Instead of one, secret key, there are two keys which go together as a pair: one "public" and one "private"

Diffie and Hellman at Stanford sketched the original idea. Rivest, Shamir, Alderman (RSA) came up with the mechanics (patented until recently).

Property #1: A message can be encoded with one key and decoded with the other. Either key may be used to encode, and the other key decodes. Aka -- asymmetric cryptography

Property #2: someone can know the public key, but knowing that will not allow them to deduce/compute what the private key is. As a practical matter, the scheme arranges that computing priv from pub requires solving a known difficult problem, such as factoring. If someone figured out a way to do factoring efficiently, then Property 2 would break.

Application: Public Key Secrecy

Bob generates a random pub/priv key pair on his own. Bob publishes the pub key.

Secure communications works as before, but the public key can be published.

Alice can send secure mail to someone without first arranging a secret key --

Alice just looks up their public key knowing that their private key will decode the message. Like secret key encryption, but without the hassle of key distribution.

Alice looks up Bob's pub key...

Alice computes $c = \text{crypt}(p, \text{pub})$ and sends c to Bob

Bob computes $p = \text{crypt}(c, \text{priv})$ and reads the message.

The asymmetric property -- the pub and priv keys go together -- a message encoded with pub can only be decoded with priv, and only Bob knows what priv is. In fact, no one but Bob ever knew priv.

Tech: Hash Function

A 1-way hash function, that computes a short "digest" or "hash" of some bytes.

$h = \text{hash}(\text{input})$

1. Every bit in the input affects every bit in the output -- changing one bit in the input results in a completely different output.

2. The function is not invertible -- given h , it is difficult (or believed to be difficult) to compute what the input was

Application: Error Detection

Suppose you are sending a large message.

Compute a "digest" $h = \text{hash}(\text{message})$ and include h at the end of the message.

The recipient can check that the message and the digest still match.

Application: Digital Signatures

Bob has a document, doc

Bob computes $h = \text{hash}(\text{doc})$

Bob uses his private key, to encode the digest -- $h_2 = \text{crypt}(h, \text{priv})$ -- this is the signature

Bob attaches the signature to the document

Alice can look at the $\text{doc} + \text{signature}$, and compute that it really comes from Bob...

Alice computes $h = \text{crypt}(h_2, \text{pub})$, and checks that $h = \text{hash}(\text{doc})$

Only Bob has priv , and so only Bob could have computed h_2 -- the document must come from Bob

For this to work, Alice needs some independent way to look up Bob's public key -- this is what a Certificate Authorities (CA) does.

Note that this is better than a (ink, paper) signature -- you cannot lift a signature from one document and put it on another. Alice can verify Bob's signature, but Alice cannot pretend to be Bob.

Technology: Certificates

A Certificate Authority (CA) publishes that a particular identity (a person, an email address, a DNS name like store.foo.com), goes with a particular public key.

The CA does not know the private key.

The CA provides a certificate that associates the identity with the public key. The certificate is signed by the CA, so you know it is not just made up (if you trust the CA).

You can tell your browser which CA's to trust, and you can add them. The most common ones are built in.

CA's are a bit expensive and somewhat obnoxious at present, since there is a lack of competition -- Verisign and Thawte are the largest

Application: HTTPS/SSL

This is how "secure" web connections/ ssh terminal connections work

Alice connects to Bob's server

Bob's server returns a certificate, and something encrypted with Bob's priv key

Alice can verify that the certificate is valid, and use the public key on it to verify what Bob sends --> Bob is authentic.

Alice can make up a one-time session secret, encrypt it under Bob's priv key, and send it to Bob. Only Bob will be able to recover the session secret --> now the 2 have a shared secret for simple cryptography for the session.

Misc....

Code = Dangerous

Code is more dangerous than a passive document.

Alice gets code from Bob. Bob's code runs on Alice's machine, deleting all her files, sending email, etc. etc. -- e.g. the many Microsoft email worm problems.

Solution1: code signing -- the code is signed, so you know who it's from (Microsoft is working on this)

Solution2: sandbox -- the code runs in a limited environment where it can't cause much damage (Java does this and also code-signing)

Buffer Overflow Attack

The FTP server is running on Alice's machine

Bob sends an over-sized request to the FTP server, exploiting a bug in the FTP server code that causes the FTP server to come under Bob's control (running on Alice's machine).

There have been many such vulnerabilities -- but they are gradually being closed as programmers learn about that sort of bug.

Server Security vs. Complexity

Set up server with web server, file sharing,

The system is complex, so it

Attack: Replay

Carl observes Alice sending encrypted packets to Bob

Carl cannot read the packets, but he can re-send them later

Problem: Bob gets the "send this order to Alice" 10 times

Solution: include serial numbers, the current time, etc. in each message

Spoof Attacks

Man in the Middle -- a router messes with the content each way

DNS Hijack -- give you the wrong IP addr, so you are not connecting

Packet spoofing -- bad guy inserts their own packets as if they were part of the stream of packets (difficult)

Solutions: certificates, encryption

Denial Of Service Attack (DOS)

Flooding a service with so many or erroneous requests, that it can't serve the people it's supposed to.

e.g. send many SYN packets (to start 3-way handshake) but never complete the handshake

Solution: partial solution is to firm up the internal routing protocols so it is harder to send bad packets -- screen them out before they get to the victim.

DOS attacks are, at some level, not solvable

Distributed DOS

Bad guy takes over a bunch of random innocent machines

Bad guys runs their DOS attack robot on each machine

On the Bad guys's signal, the hundreds of robots send traffic to the victim -- greater effect, and it's harder to trace back to the bad guy

Social Engineering Attack

Tricking people socially into moving a file, revealing a password, etc.

This works very well -- send the message "your password has been compromised, please change it to 'nowsafe'" -- send this message to 10,000 AOL users -- you get some.

Other Technologies

1. Firewall

Separate your internal net from the rest of the internet. Only allow certain port#'s connections to go through in certain directions. Easier than securing your whole internal network, but can cause problems where some services no longer work since they don't use one of the approved port #'s. Also, can provide a false sense of security -- if the HTTP server has a vulnerability (port 80), the firewall is no help.

2. Virtual Private Network

Set up an encryption layer in software between all your computers on the Internet. The VPN traffic is all encrypted/authenticated, but before the traffic is sent on the Internet at large.

IPv6 does this