

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? Has it been altered?

Alice and Bob

Alice wants to send a message to Bob

Authenticity: Alice can determine that the other party really is Bob

Secrecy: Alice sends Bob a message. A bad guy intercepting the message cannot read it.

Authenticity: Bob receives the message, and can tell that it really was written by Alice and has not been changed.

Security: Secret Key

Security depends on a secret "key" known only to the parties

If the key is well protected, the protocol can be public

Security Through Obscurity

Security through obscurity -- not a good idea

e.g. a detail of the protocol is not publicized, but Alice and Bob know and depend on it

Inevitably, "secret" parts of the protocol become known, and then the whole thing is blown

Real security depends on secret keys, not an obscure method

Obscurity can be somewhat effective in a low-tech way -- e.g. keep your passwords in a file named "resume.txt".

Can be effective against robotic attacks that are set up only to deal with a "standard" installation of, say, a web server.

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

Bob gets the message, and we have both security and authenticity

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 used to be the most common source or problems on campus when "password in the clear" protocols were in wide use. Now Microsoft IE/email holes are the larger source of problems.

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 (or the other way around), 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 resulting in a signature -- $\text{sig} = \text{crypt}(h, \text{priv})$

Bob attaches the signature to the document

Alice can look at the $\text{doc} + \text{sig}$, and compute that it really comes from Bob and goes with this document...

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

Only Bob has priv, and so only Bob could have computed sig -- 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 a token encrypted with Bob's priv key

Alice can verify that the certificate is valid, and use the public key to decrypt the token. If the token decrypts cleanly, Alice verifies that the other party really is Bob.

Alice can make up a one-time session secret, encrypt it under Bob's pub 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.