

Key pairs

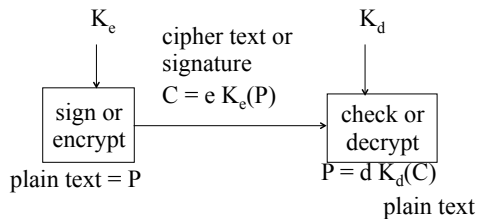
A little bit of information on methods of cryptography. Capabilities use cryptography.

Information is encoded using a key.

It is decoded using either the same or another key.

Symmetric algorithms – same key, must be kept secret.

Asymmetric algorithms use different keys for encrypting and decrypting (one key can be public). The idea is that although the keys are mathematically related it is *impossible* to produce one from the other.



Public key use

We can use public key algorithms in two ways.

Make the encoding key public – then anyone can encrypt messages but only the holder of the private key can decrypt.

Useful for talking to a site without interception.

Make the decoding key public – then anyone can decrypt the messages but only the holder of the private key can encrypt them.

Useful for proving the message came from the holder of the private key.

Capabilities can be of this form.

Digital Signing

We need a way of proving that messages come from who they say they are and haven't been altered on the way.

The sender puts the message through a hash algorithm to produce a message digest (e.g. SHA-256).

Then encrypts the digest with its secret key. This is the digital signature for this message.

Sends the message and the signature.

The recipient uses the sender's public key to decrypt the signature.

Also calculates the message digest on the message (SHA-256). (Note: Microsoft, Google, Mozilla etc will not accept SHA-1 certificates after 2017.)

Checks the local digest value with the one sent.

If they match, the message was sent by the proper subject and was not modified on the way.

Sharing keys

There are problems with distributing keys.

If the key is public the receiver still needs a way of checking that the key is authentic.

If the key is private we need to ensure that only the right domains get the key.

Solutions to checking key authenticity:

- the key can be signed and we have the key to check the authenticity of the signature – a trust chain
- trusted third parties – can certify that the key (or certificate) came from the correct source
- contact the source and check some *fingerprint*

Sharing secret keys

There are methods to cooperatively form secret keys during sessions e.g. the Diffie-Hellman protocol.

The parties then share a secret key but they don't know who the other party is.

This can be solved but requires knowing signature verification keys.

For OSs the usual approach is to use a server that holds secret keys for all domains. Then we can use the Needham-Schroeder Protocol.

Diffie-Hellman Protocol

A and B are the parties that want to communicate securely. They need a shared secret key.

There are two public values (one is a large prime p and the other is related to the prime (primitive root modulo p)).

Both parties generate a random private key. A produces key 'a' and B produces key 'b'.

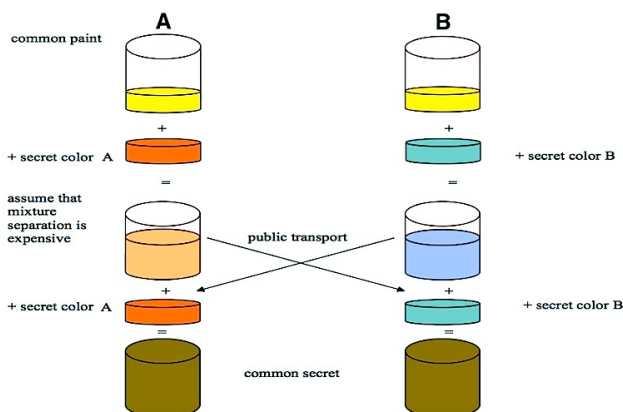
From these keys and the two original public values, both can produce public values they transmit to each other.

A shared secret key can then be produced by combining the public key each gets from the other with their private keys.

This shared key cannot easily be broken, with the public values.

Diffie-Hellman Protocol

<http://en.wikipedia.org/wiki/Diffie-Hellman>



Diffie-Hellman Example

Public prime value $p = 23$, number primitive root mod $p = 5$.

Alice chooses random number 6.

Generates $5^{*}6 \text{ mod } 23 = 8$ (sends to Bob)

Bob chooses random number 15.

Generates $5^{*}15 \text{ mod } 23 = 19$ (sends to Alice)

Alice then does $19^{*}6 \text{ mod } 23 = 2$

and Bob does $8^{*}15 \text{ mod } 23 = 2$

So 2 can be used as the secret key.

In use, p would be a huge number, 300 digits and the random numbers would have about 100 digits.

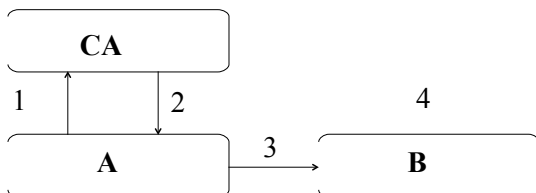
Certification

Diffie-Hellman is susceptible to Man-in-the-Middle attacks.

So identities need to be proved. (MQV (Menezes-Qu-Vanstone) is based on Diffie-Hellman but uses the pre-existing public keys of both parties to include authentication.)

Certification Authorities

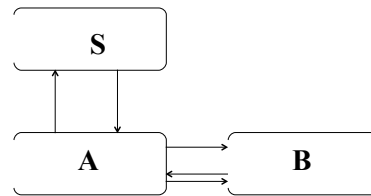
A wants to prove its identity to B.



1. A makes a request to the certification authority for a signed certificate. Includes its name and public key.
2. CA signs the message with its private key to produce a signature (the message and signature is now the certificate).
3. A sends the certificate to B.
4. B checks certificate using CA's public key.

Needham-Schroeder Protocol

But if we have a server with everyone's secret keys, we can use the Needham-Schroeder symmetric key protocol.



A and B are the parties that want to communicate securely.

S is the server with secret keys for A and B.

So A can communicate securely with S and B can communicate securely with S.

A tells S it wants to talk to B.

S gives A a key for A and B to talk K_{AB} . It also includes a verifying message for B using B's secret key.

A sends the verifying message to B.

B checks the message (which includes A's identity and the key K_{AB})

This algorithm is extended by Kerberos (and hence AFS).

How things go wrong

ch8 of Gollmann's Computer Security

The three "c"s of security failure.

- change
- complacency
- convenience

Change

Odd numbered versions of software (and OSs) commonly fix security errors.

The Mad Hacker (retrofitting security)

ICL's VME/B information on files was owned by :SD
Added security levels. So :SD didn't own classified file information.

To restore from backup a new user :SD/CLASS was added to handle this.

To stop anyone logging in as :SD/CLASS given an empty password by patching the password file.

The wrong field was patched and caused the :SD/CLASS user to have unlimited access.

Could only log on as :SD/CLASS from the master console, but if the master console was turned off the next device to open a connection became the master console.

Complacency

Bounds checking

`fingerd` – process running to handle *finger* requests

Used `gets` library routine to get input into a buffer.

No length check.

If you know the architecture and OS you can overflow the allocated stack space for the buffer and leave a return address to an exec method call on the stack.

A buffer overflow attack.

The same sort of thing is possible with many OSs and there are several cases of unchecked buffers causing security problems under all varieties of Windows and Unixes.

VMS login – users could specify the machine they wanted to access

`username/DEVICE=<machine>`

The length of `<machine>` wasn't checked. The user's privilege mask was on the stack following the buffer. So you could overwrite the privilege mask to provide any desired privileges.

Preventing Stack Buffer Overflow attacks

Buffer length checks before storing data into the buffer/stack.

Canaries - place a random value before the return address. If this value is modified then the stack has been compromised.

Data Execution Protection (DEP) - use the memory management system to not allow execution of locations on the stack.

- Return oriented programming can get around this
- so the OS randomises the location of our code (ASLR - Address Space Layout Randomisation), making it harder to work out where code "gadgets" are

Unthought through interactions

Some programs cause security problems because of unexpected usage.

at

The Unix at command is used to run programs at particular times.

```
at time -f file
```

puts a copy of the file into a spool directory to be executed at the time.

at doesn't check the read permissions when the file is put there.

The user can read files they put in the spool directory (copies of files they were not allowed to read) - and then remove them to hide the evidence.

More complacency

Syntax checking

```
rlogin
```

Can remotely login to a machine with the rlogin command

```
rlogin -l username machine
```

On Linux and AIX some versions did not check the syntax of the username and passed it straight on to the login command.

```
login username
```

if the username was -froot

```
rlogin -l -froot machine
```

```
login -froot machine
```

then the person was logged in without a password required; the -f flag means without a password to the login command.

Convenience

Adding security makes a system less usable.

There is always a trade off between convenience and security.

The most secure systems are very inconvenient.

Any expert will acknowledge that it's simple to create a totally secure computer: you simply unplug every connection, including power, encase the thing in concrete, and surround it with guards. By the same token, a pair of wire cutters provides the perfect network firewall: cut your Internet connections and we guarantee you won't suffer from Internet-based attacks.

from the SideWinder site <http://www.securecomputing.com>

Systems with a superuser are certainly easier to maintain but much harder to keep secure.

Authentication

Even the best security system can be compromised if an intruder can successfully impersonate a legitimate user.

There also need to be policies that stop users *sharing* their identities with others.

From the OS point of view we need a way of allowing authenticated users access to the system but no one else.

We can use:

- possessions
- attributes
- knowledge

and combinations of these.

Possessions

Keys or cards.

Locks can be picked and smart cards can be analysed.

Attackers (if they have access to a card) use techniques such as manipulating the power supply or clock to get secret information.

The hope is that the card will get into an unknown state and produce information it shouldn't.

A type of fuzzing - provide random, invalid, unexpected data and observe what happens. Used to find bugs and security problems.

Even just observing how long it takes to perform computations can be used to cut down the possibilities when trying to guess a secret key.

Of course keys and cards can be easily stolen as well.

The theft must be reported quickly and the locks changed.

Attributes

Physical characteristics of the user.

- palm prints
- finger prints
- iris patterns
- retina patterns
- voice print

Also the way things are done.

- typing patterns (different people type different sequences of characters at different speeds)
- signatures – we include the speeds, directions and pressure of different strokes

All of these biometric methods can suffer from false positives and false negatives.

False positive – let someone have access who shouldn't

False negative – refuse access to a legitimate user

The probabilities of each can be altered by changing parameters. Best to use a combination of techniques.

Before next time

Read from the textbook

15.2 – Program Threats

15.3 – System and Network Threats

15.6 – Implementing Security Defenses

15.8 – Computer Security Classifications

15.9 – An Example Security Model: Windows 7 (XP)