Some fundamental security concerns...

Confidentiality - could someone else read my data?

Integrity - has my data been changed?

Authentication - is this who they claim to be?

*Cryptography offers **genuinely secure** solutions to these problems*

We'll look briefly at four main components

1. Hashes (one-way encryption)

Munging a document (of any size) gives a short, fixed-length ***hash*** or ***message digest***

Examples of Hash algorithms

MD5 - 128 bits of output

SHA1 - 160 bits

RIPEMD-160 - 160 bits

SHA256 - 256 bits

Properties of Cryptographic Hashes

- Running the same hash algorithm on the same document always gives the same result
- It is infeasible to modify the document whilst keeping the hash the same - or even to find *any* other document with the same hash
- Hence a powerful check of **integrity**
- Important: MD5 is now BROKEN!
  - all it takes is 3 days and 200 playstation3's *
  - SHA1 not yet, but has known weaknesses

* Google for "MD5 considered harmful today"

Quick Exercise

On your PC: `cat /etc/mailcap`

Compare the result on your neighbour's PC

Are they exactly the same? Can you be sure?

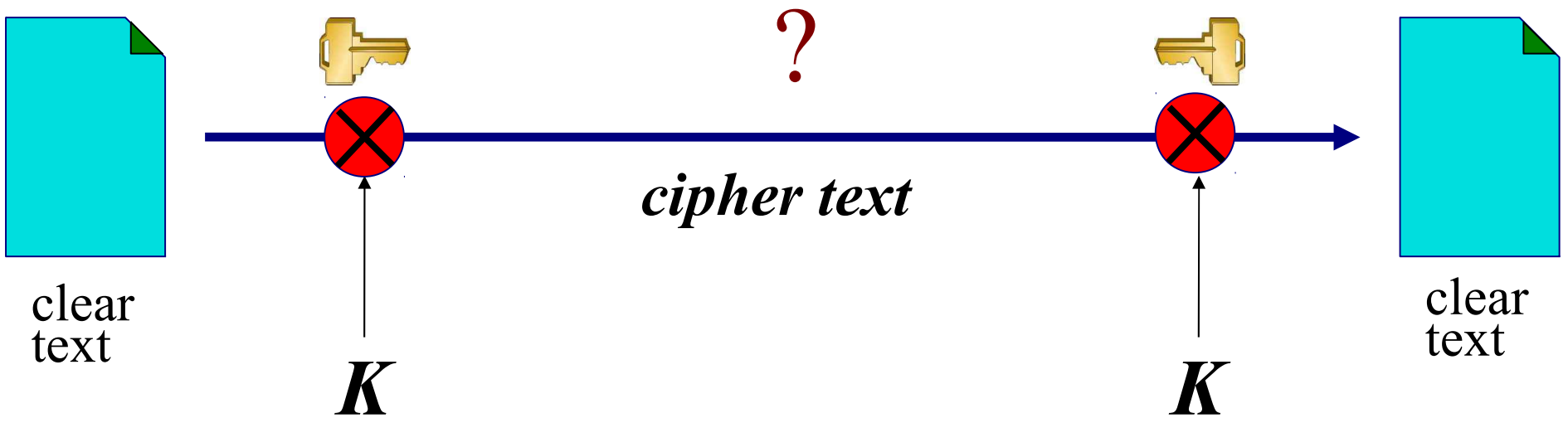
Now do: `sha1sum /etc/mailcap`

Is your neighbour the same now?

Edit the file (`joe /etc/mailcap`) and change just one character

Do again: `sha1sum /etc/mailcap`

2. Symmetric Cipher (private key cipher)



The same key is used to encrypt the document before sending and to decrypt it once it is received

Examples of Symmetric Ciphers

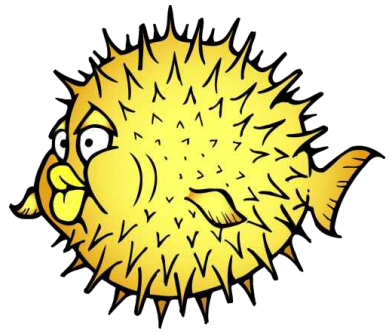
DES - 56 bit key length

3DES - effective key length 112 bits

RC4 - 128 bits

AES (Advanced Encryption Standard) - 128 to 256 bit key length

Blowfish - 128 bits, optimized for fast operation on 32-bit microprocessors



Properties of Symmetric Ciphers

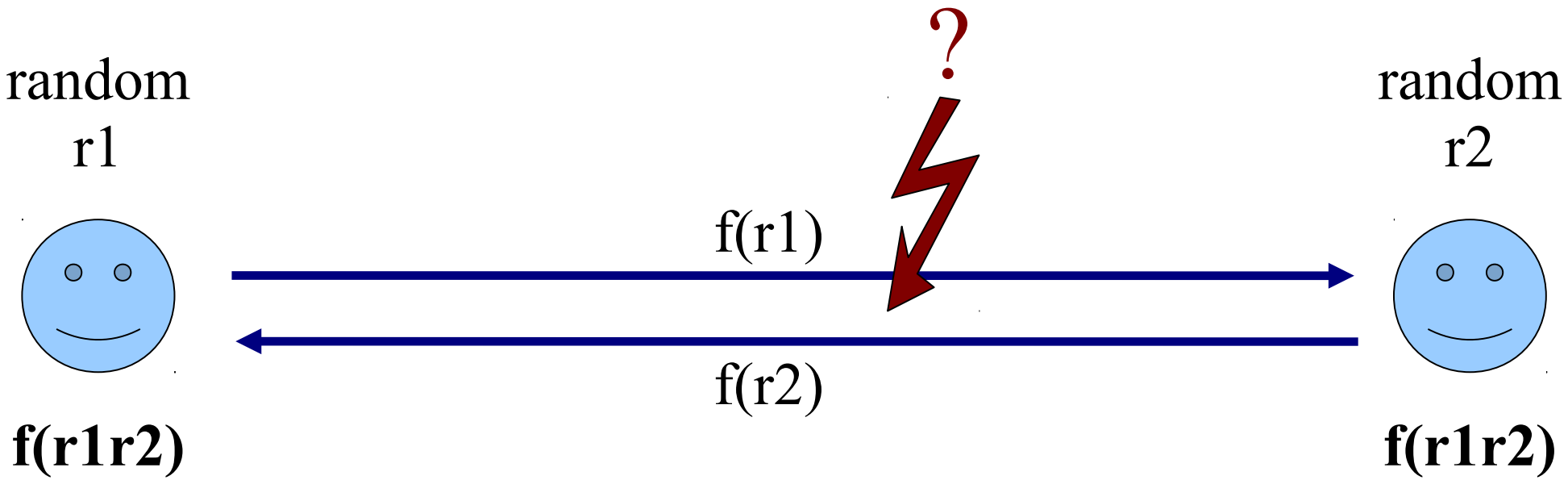
- Provides confidentiality: infeasible to decrypt data without knowing the secret key K
- Provides integrity: a small change to the ciphertext will cause it to decrypt to garbage
- Provides authenticity: if I can decrypt the data with my key K, I know it must have been encrypted by someone who knows K
- Fast to encrypt and decrypt, suitable for large volumes of data

Attacks on Symmetric Ciphers

- Good ciphers resist attacks on the algorithm; brute-force attack is directly related to the key length.
- Current recommendation is a key length of 90+ bits, for data protection of 20 years.*
- Relies entirely on secrecy of the key. How can you distribute it securely to your peer without it being intercepted by an attacker?
- Use a hash to convert a passphrase into a value suitable for a key (passphrase easier to remember)

*See http://www.keylength.com/ for a collection of recommendations

3. Diffie-Hellman key exchange



Two parties agree on the same key - but a sniffer cannot derive it

Properties of Diffie-Hellman

One of the first asymmetric cryptosystems - from 1976

Used for generating temporary keys ("session keys") which are discarded after use

A sniffer who sees all the traffic still cannot work out what key has been agreed upon

Security depends on the number of bits, and on truly random numbers being chosen

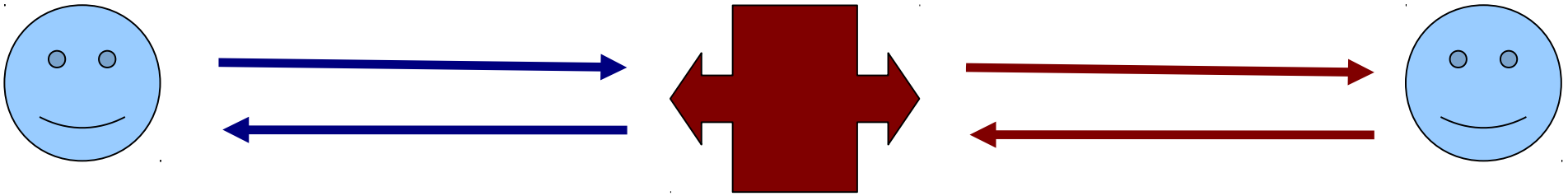
(reasonable minimum: "DH group 2", 1024 bits)

Does Diffie-Hellman solve the key distribution problem?

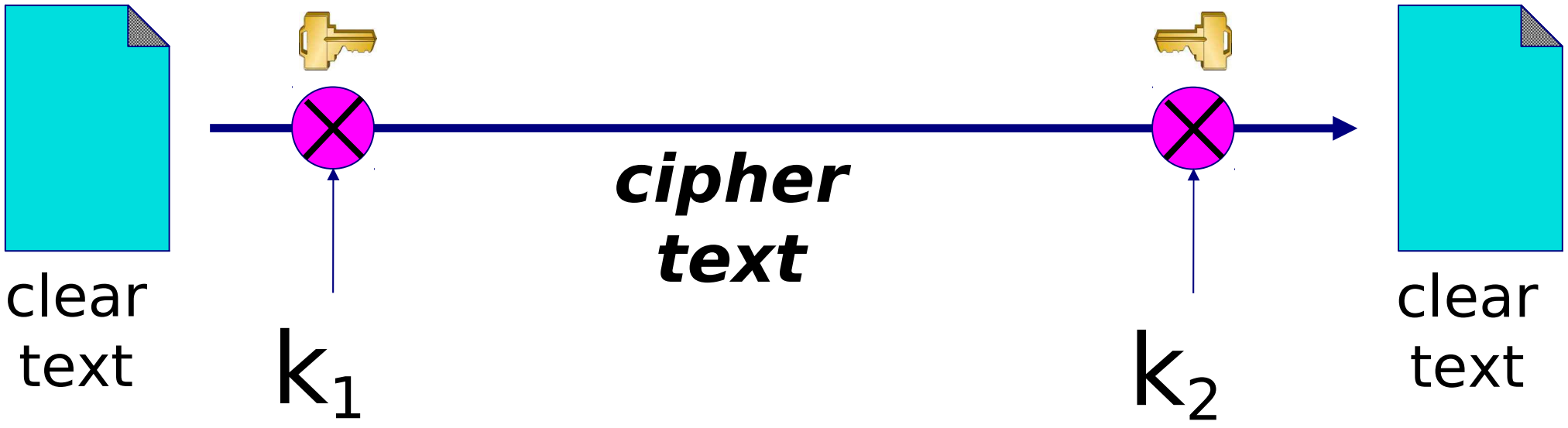
Unfortunately not :-(

An **active** attacker can intercept traffic and perform key exchange with either or both sides: a **man-in-the-middle** attack

In short: you don't know who you're talking to



4. Public key cipher



(***public key***) (***private key***)

One key is used to encrypt the document,
a different key is used to decrypt it.

This is a big deal!

Properties of Public Key Ciphers

- The keys are mathematically related
  - Easy to convert private key into public key
  - Infeasible to convert public key into private key
  - You can safely post your public key anywhere!! (That's why it's called "public")
- Can provide confidentiality: encrypt with public key, decrypt with private key
- Can provide authenticity: encrypt with private key, decrypt with public key

Properties of Public Key Ciphers

Mathematical attacks, not just brute force

- 512-bit RSA broken
- 1024-bit RSA may be insecure against highly-resourced opponents
- 2048-bit is recommended for good security

MUCH slower than symmetric ciphers (like, 1000 or 10,000 times slower)

Hence are *combined* with symmetric ciphers to build a practical system

Example application: gpg

gpg lets you:

- generate a public/private key pair
- encrypt messages with any public key, and/or
- sign messages with your private key

Used for sending encrypted E-mail, verifying integrity of software packages, etc

Use for authentication

If you have my public key, I can prove to you that I own the corresponding private key (without sending it to you)

My public key is therefore a form of identity

Similarly, you can prove your identity to me

Solves the man-in-the-middle problem, as long as we both know each other's public keys

If not, we can use a third party - a Certificate Authority - to confirm identity of key owner

Example: ssh with private key auth

A simple way to get rid of passwords entirely!

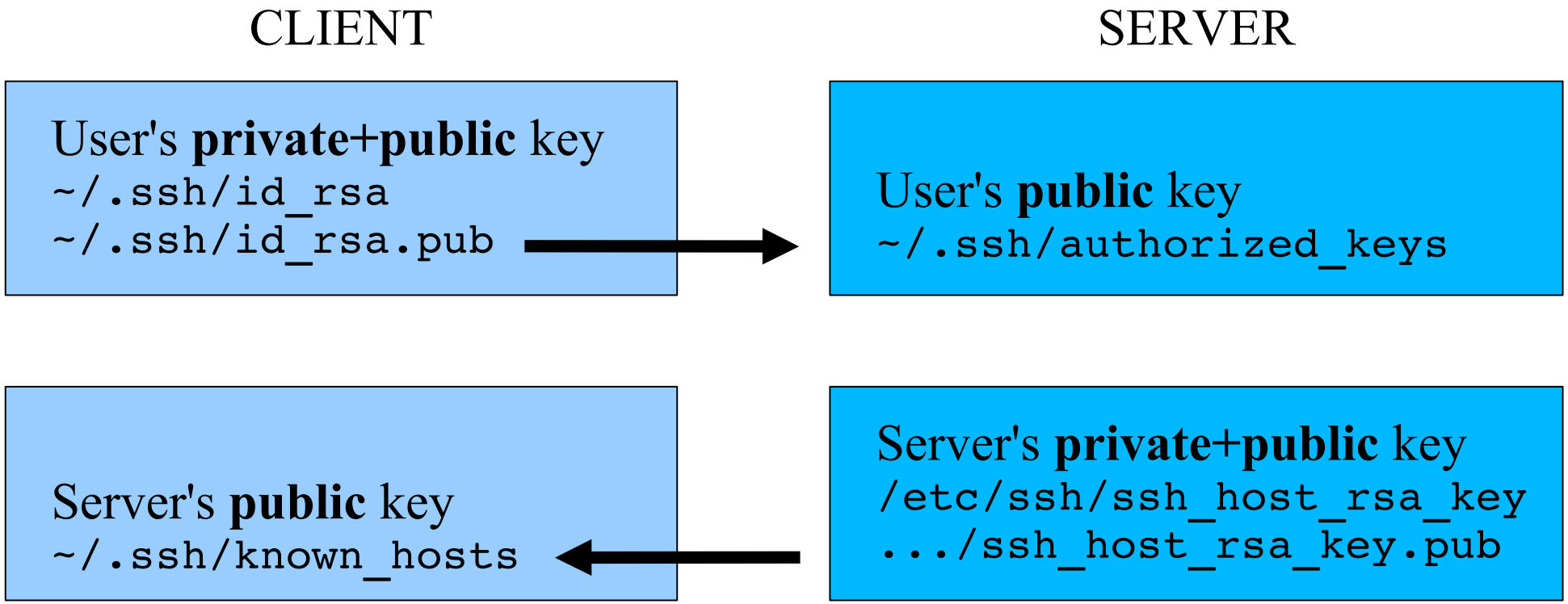
Generate an ssh key pair (just once)

Keep the private key safe - encrypt it with a passphrase

Install the public key on each server you want to login to

When you login, you use the passphrase to decrypt ("unlock") your private key locally - it never gets sent to the remote system

Proving identities with ssh keys



(ssh has become popular partly because it *doesn't* use certificates)

Man-in-the-middle and ssh

- How does the server know it's really me?
  - Because my public key is in ~/.ssh/authorized_keys
  - How did it get there? Trusted person put it there
- How do I know the server is really them?
  - The server's public key is in ~/.ssh/known_hosts
  - How does it get there? I *could* do it manually. If not, ssh prompts me to install it the first time I connect
  - This works fine, as long as a man-in-the-middle attack is not taking place at the time!

Exercise

Create ssh public/private key pair (RSA, 2048 bits)

Copy the public key onto your Unix server PC

Use your private key to login without password

Discussion

What are the pros and cons of this approach, compared to passwords?

WARNING!

This is the most basic introduction to crypto

DON'T try to write your own cryptographic tools - there are many pitfalls

  Look at what happened with WEP

DO use widely-used and reviewed tools: they are written by people who understand the maths and the possible weaknesses

And don't forget the human element