

# Services and Security through Cryptography

## ccTLD Workshop

November 27, 2007

Amman, Jordan

Hervey Allen



# Securing your Server

So, you wanna secure your server? The usual steps apply:

- Run only what you need – nothing extra.
- Keep it up-to-date.
- Is your kernel “patchable”?
- Run DNS (BIND) in a jail?
- Watch permissions & avoid services as *root*
- Logs: watch them, keep them, tune them.
- Do you want/need a server-based firewall?

# What's our Goal with all this?

(1)-- **C**onfidentiality

(2)-- **I**ntegrity

(3)-- **A**uthentication

- Access Control

- Verification

- Repudiation

(4)-- **A**vailability



# Cryptography Plays a *BIG* Role

We may take it for granted, but cryptography is behind much of what we do today:

- ssh/scp/sftp
- ssl/tls/https
- pgp
- pops/imap
- smtps
- vpn's
- dnssec
- wep/wpa
- digital signatures
- certificates
- pki
- drm
- disk encryption
- etc...

# Do You Need to Know What Makes all this “Tick”?

Let's look at both sides of this...

**YES!**

- You can avoid mistakes and make systems run together more smoothly.
- You might sleep better.
- Implementation of complex items like secure email, DNSSEC might be easier.
- SSL Certificates. Do you know how they work?

# The “YES” Argument cont...

## YES!

- SSH: The initial key exchange. “Man-in-the-Middle” attacks. Endless SSH admin tricks.
- PGP: Do you use it? Very useful for system administration.
- Digital Signatures: Great for legal documents.
- What if you wanna use/test DNSSec?
- Similar to understanding algorithms, software engineering practices, etc.

# The “No” Argument

## What Makes all this “Tick”?

**NO!**

- All these tools still work even if you have no idea how they are working.

OK – I can't think of any other good “No” arguments...

Will you use tools like PGP, digital signatures, unsigned certificates if you don't know what they mean or do?

# Some History

What we use today did not exist for public use just a few years ago (1970's) and was largely just theory for the general public.

Alan Turing published some seminal works on the mathematics behind cryptography. Some are classified and still not available.

The US National Security Administration (NSA) has (probably) used advanced cryptographic methods since the 1960's.



# Some History cont...

**1976 was a big year...**

**DES:** Adopted as an encryption standard by the US government. It was an *open* standard. The NSA calls it “One of their biggest mistakes.”

**public-key cryptography:** Whitfield Diffie and Martin Hellman describe public/private key cryptographic techniques using trap-door one-way mathematical functions. Radical transformation of the cryptographic paradigm.

# “Applied Cryptography”

Written by Bruce Schneier. This is, perhaps, the best book around if you want to understand how all this works.

- Crypto-Gram email newsletter
  - <http://www.schneier.com/crypto-gram.html>
- Counterpane Security
  - <http://www.counterpane.com/>
- A voice of reason around much of the security hysteria we face today.

# Terminology

This is either the “boring” part of this lecture, or if you're me it's the best part of this lecture...

Lots of terminology. Once you know what it means, then you can get on with the business of using these tools well and making your services (more) secure.

# Terminology Cont.

- hashes/message digests
  - md5/sha1
  - collisions
- entropy (randomness)
- keys
  - symmetric
  - asymmetric (public/private)
  - length
  - distribution
  - creation
- ciphers
  - block
  - stream
- plaintext/ciphertext
- password/passphrase

## All these lead to...

- SSL/TLS
  - Digital Certificates
    - + CSRs
    - + CRTs
    - + PEM files
    - + CAs
- SSH
- PGP
- Secure email with:
  - secure SMTP
    - + SSL
    - + StartTLS
  - POPS, IMAPS

So, let's get started!

# Ciphers $\implies$ ciphertext

We start with *plaintext*. Something you can read.

We apply a mathematical algorithm to the plaintext.

The algorithm is the *cipher*.

The plaintext is turned in to *ciphertext*.

Almost all ciphers were secret until recently.

Creating a secure cipher is *HARD*.

# Keys

To create ciphertext and turn it back to plaintext we apply a key to the cipher.

The security of the ciphertext rests with the key. This is a *critical* point. If someone gets your key, your data is compromised.

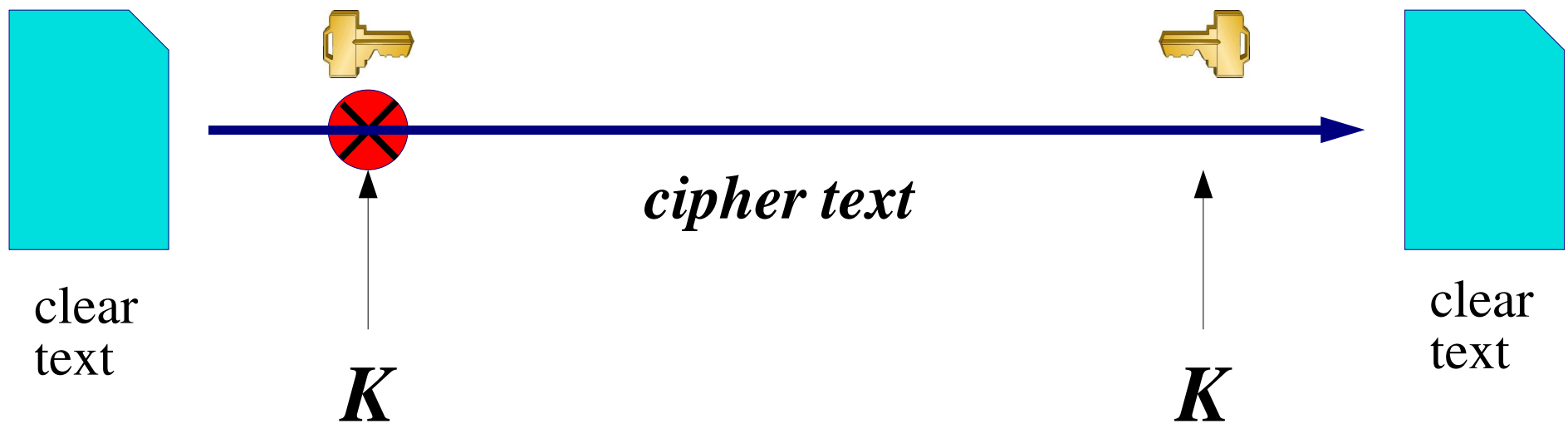
This type of key is called a *private key*.

This type of cipher system is efficient for large amounts of data.

This is a *symmetric cipher*.

# Symmetric Cipher

## Private Key/Symmetric Ciphers



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

# Examples of Symmetric Ciphers

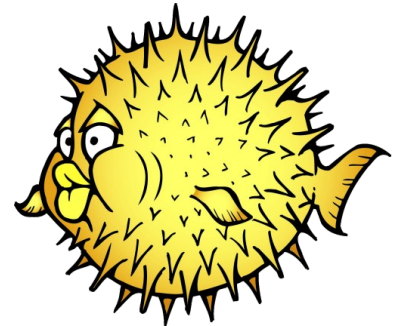
**DES** - 56 bit key length, designed by US security service

**3DES** - effective key length 112 bits

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

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

**IDEA** - 128 bits, patented (requires a license for commercial use)





# Features of Symmetric Ciphers

- Fast to encrypt and decrypt, suitable for large volumes of data
- A well-designed cipher is only subject to brute-force attack; the strength is therefore directly related to the key length.
- **Current recommendation is a key length of at least 90 bits**
  - i.e. to be fairly sure that your data will be safe for at least 20 years
- Problem - how do you distribute the keys?

# Public/Private Keys

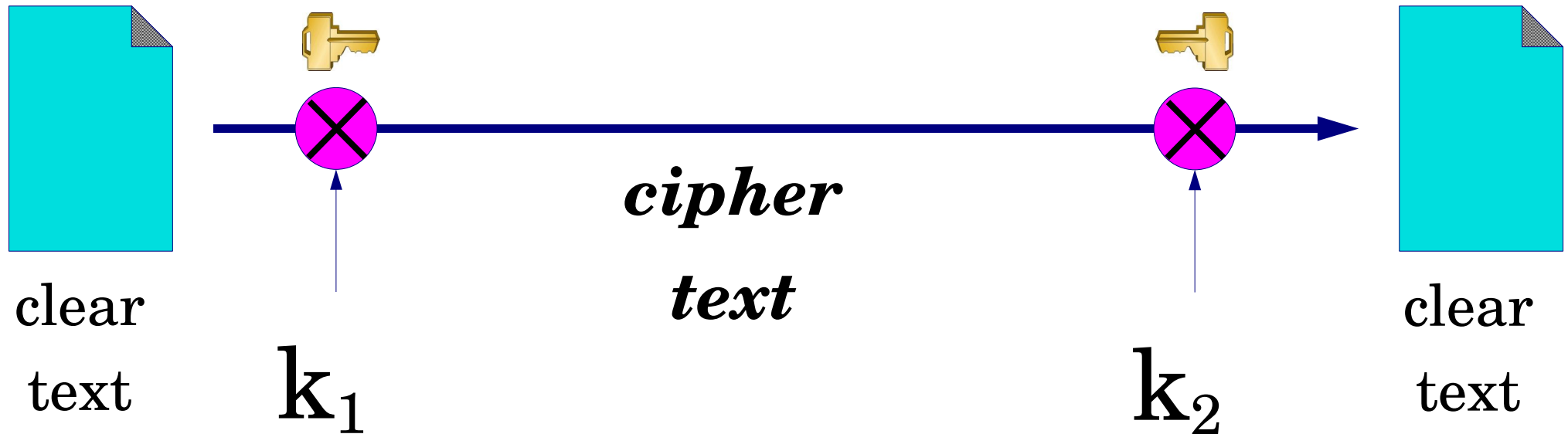
We generate a cipher key pair. One key is the *private key*, the other is the *public key*.

The *private key* remains secret and should be protected.

The *public key* is freely distributable. It is related mathematically to the private key, but you cannot (easily) reverse engineer the *private key* from the *public key*.

Use the *public key* to encrypt data. Only someone with the *private key* can decrypt.

# Example (Public/Private Key pair): Not Efficient – Not as Secure



*(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!***

# Why Not Efficient/Secure?

- Symmetric ciphers (one private key) are *much* more efficient. About 1000x more efficient than public key algorithms for data transmission!
- Attack on the public key is possible via chosen-plaintext attack. Thus, the public/private key pair need to be large (2048 bits).

# Why Not Efficient/Secure cont.

Mathematically we have:

$E$  = the encryption function

$C$  = ciphertext

$P$  = plaintext

$$C = E_K(P)$$

So, if you know one  $P$  encrypted by  $E$ , then you can attack by guessing all possible plaintexts and comparing with  $C$ .  $E$  is public in this case. Thus, you can recover the complete original text.

# Hybrid Systems

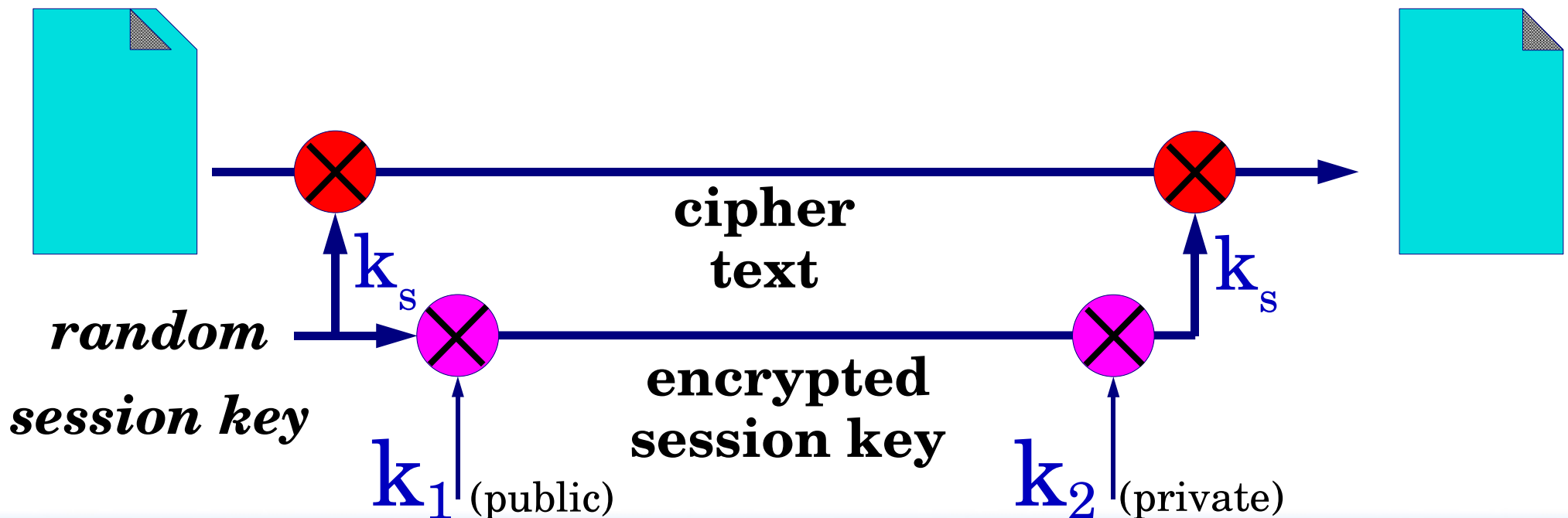
Symmetric Ciphers are not vulnerable in the previous way. The key length can be much shorter.

So, we do this:

- Use a symmetric cipher.
- Generate a one-time private key.
- Encrypt the key using a public key.
- Send it to the other side, decrypt the one-time key.
- Start transmitting data using the symmetric cipher.

# Hybrid Systems

Use a symmetric cipher with a random key (the "session key"). Use a public key cipher to encrypt the session key and send it along with the encrypted document.



# Hybrid Systems cont...

Two things should (imho) stand out:

- 1) “*Send it to the other side, decrypt the one-time key.*” How?
- 2) What about protecting your private key?

Any ideas?



# Hybrid Systems cont...

1) “*Send it to the other side, decrypt the one-time key.*” How?

Use your private key.

2) What about protecting your private key?

Encrypt it using a hash function.

# One-Way Hashing Functions

A mathematical function that generates a fixed length result regardless of the amount of data you pass through it. Generally very fast.

You cannot generate the original data from the fixed-length result.

Hopefully you cannot find two sets of data that produce the same fixed-length result. If you do this is called a *collision*.

# One-Way Hashing Functions cont.

Two popular hashing functions include:

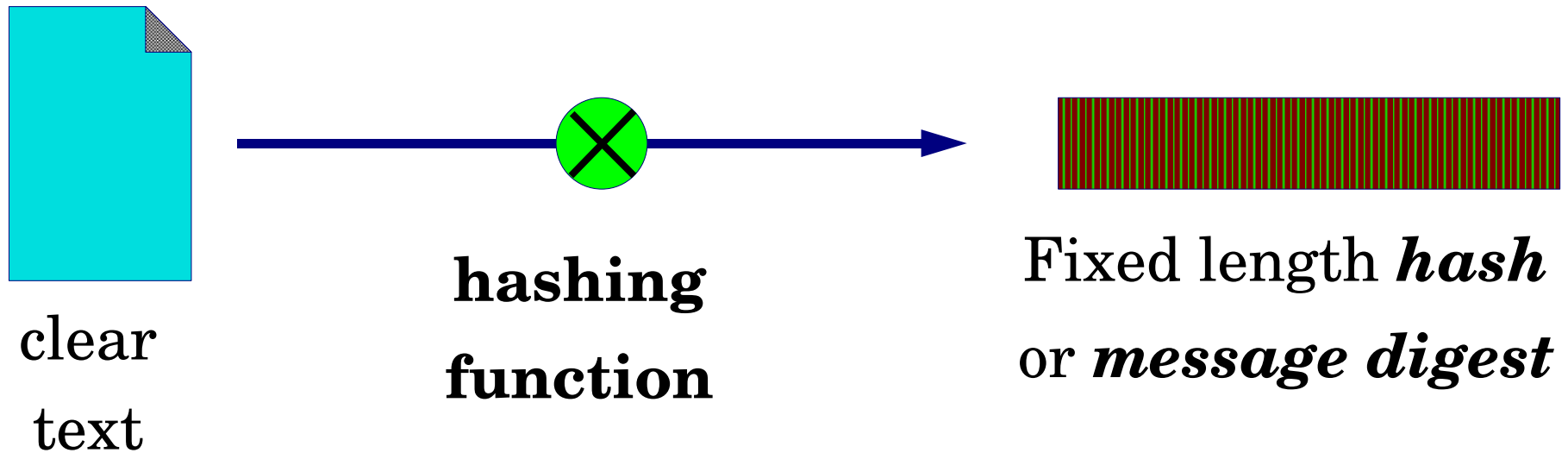
- md5: Outputs 128 bit result. Fast. Collisions found.
- sha-1: Outputs 160 bits. Slower. No collisions yet.

Applying a hashing function to plaintext is called *munging the document*.

The fixed-length result is referred to as a *checksum, fingerprint, message digest, etc.*

# Hashing

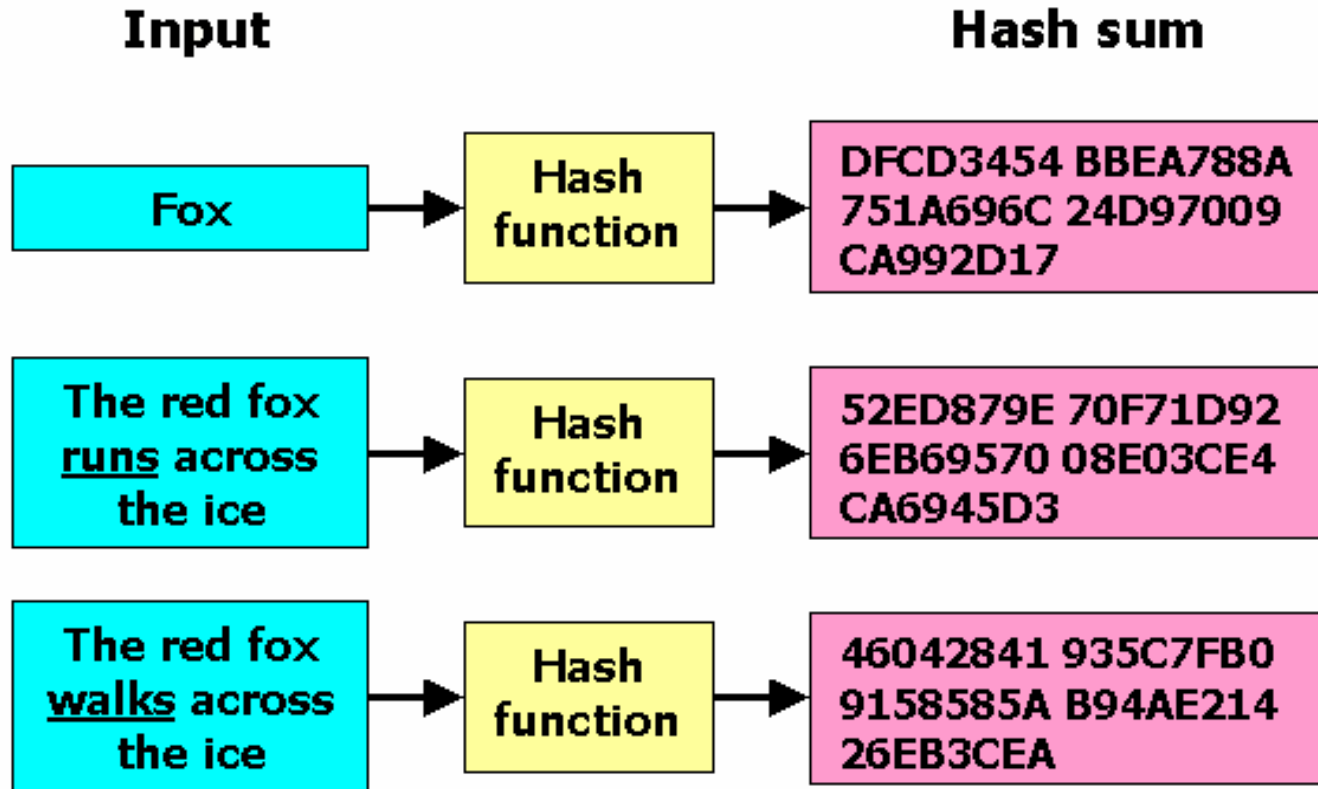
## One-Way Encryption



Munging the document gives a short *message digest* (checksum). Not possible to go back from the digest to the original document.

# Hashing

one-way encryption: another example



Note the significant change in the hash sum for minor changes in the input. Note that the hash sum is the same length for varying input sizes. This is extremely useful.

\*Image courtesy Wikipedia.org.

# Examples

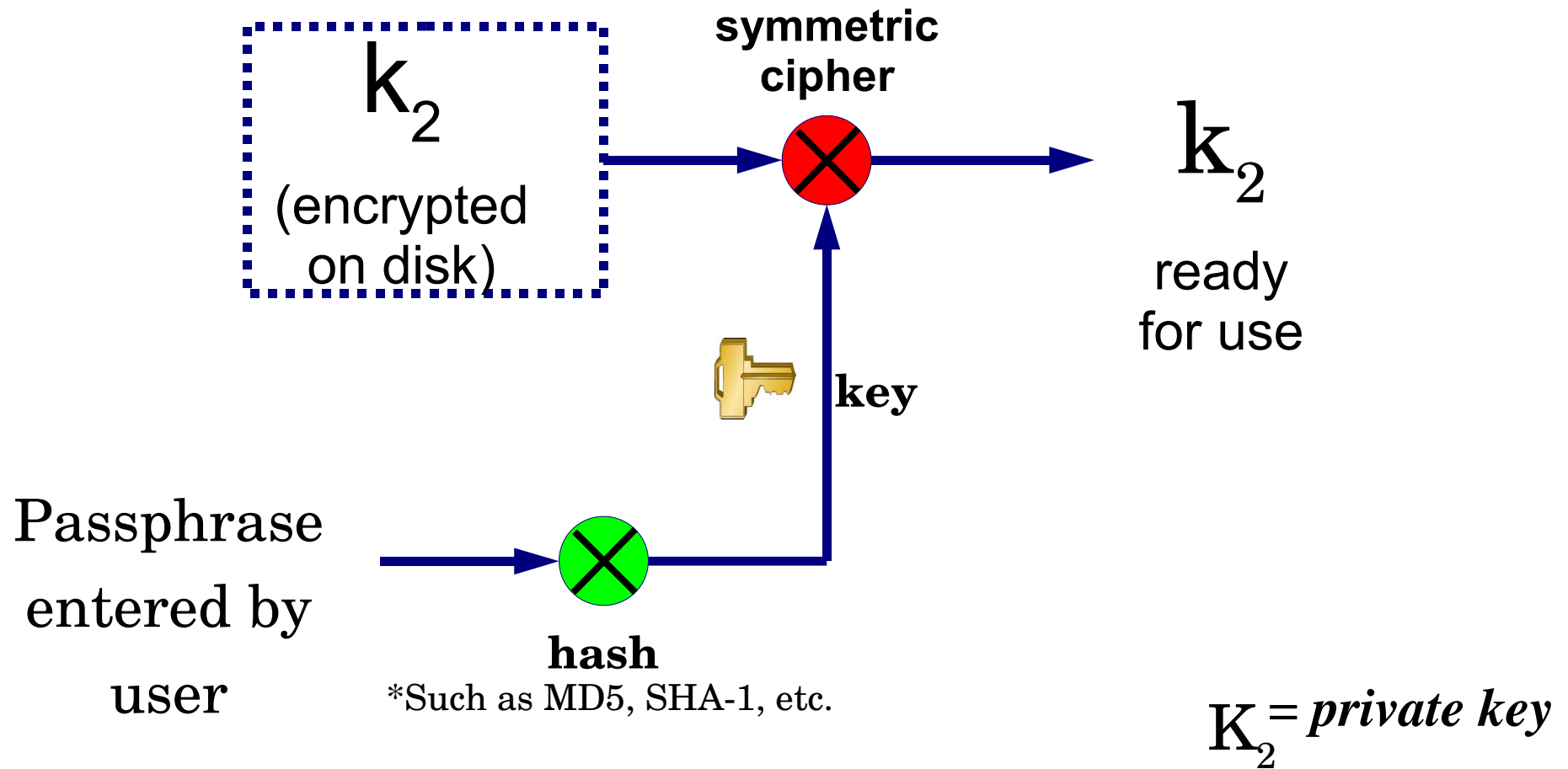
- Unix ***crypt()*** function, based on DES (***not secure!***)
- ***MD5*** (Message Digest 5) - 128 bit hash
- ***SHA1*** (Secure Hash Algorithm) - 160 bits
- Until August 2004, no two documents had been discovered which had the same MD5 digest!
  - Such "collisions" are not a major problem as yet
  - No collisions have yet been found in SHA-1
- Still no feasible method to create any document which has a given MD5 digest

# What use is this?

- You can run many megabytes of data through a hashing function, but only have to check 128-160 bits of information. A compact and *unique document signature*.\*
- You can generate a *passphrase* for your data – such as your encrypted private key. If someone gets your private key, they still must know your passphrase to decrypt anything using your private key.

\* Even with the recent attack, at best the attacker could add some corruption and leave the MD5sum unchanged. They could not insert any data of their own choosing.

# Protecting the Private Key





# Checking passphrases/passwords

Q.) How do you do this?

A.) It's very simple.

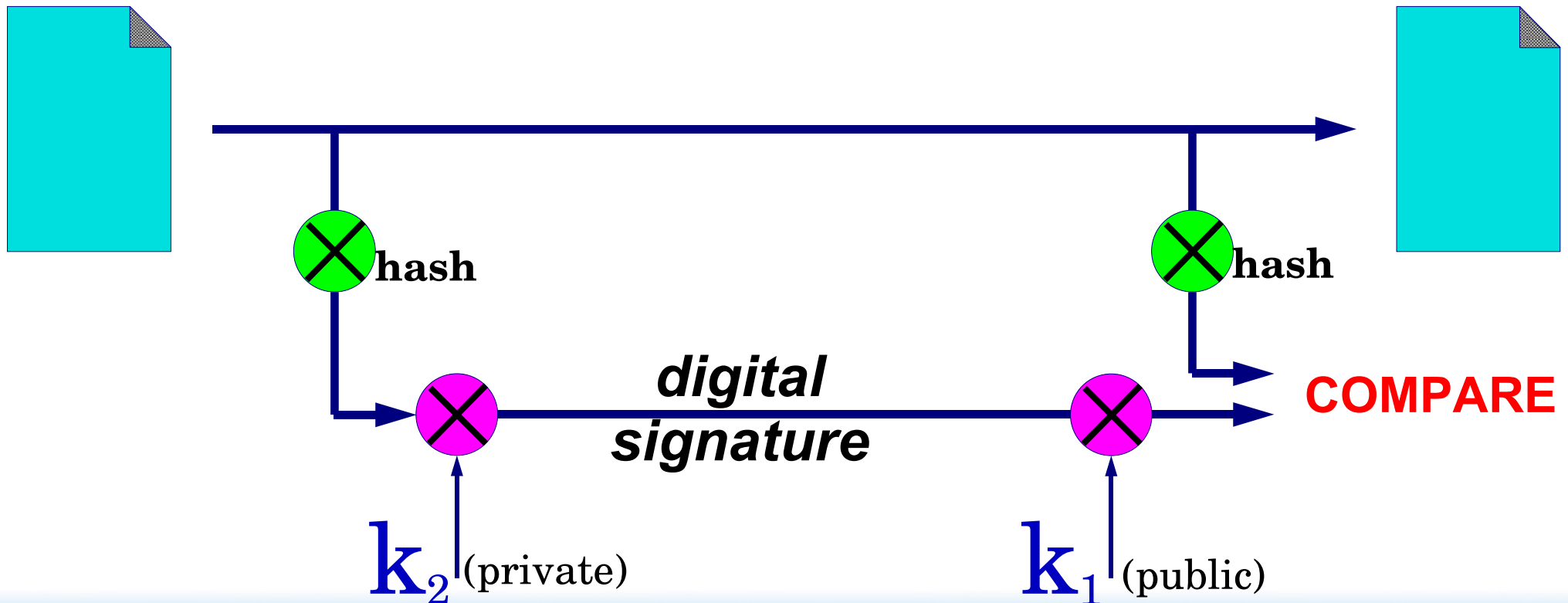
- Type in a passphrase/password.
- Run the hashing function on the text.
- If the message digest matches, you typed in the correct passphrase/password.

# Digital Signatures

- Let's reverse the role of public and private keys.  
To create a digital signature on a document do:
- *Munge* a document.
  - Encrypt the *message digest* with your private key.
  - Send the document plus the encrypted message digest.
  - On the other end munge the document *and* decrypt the encrypted message digest with the person's public key.
  - If they match, the document is authenticated.

# When authenticating:

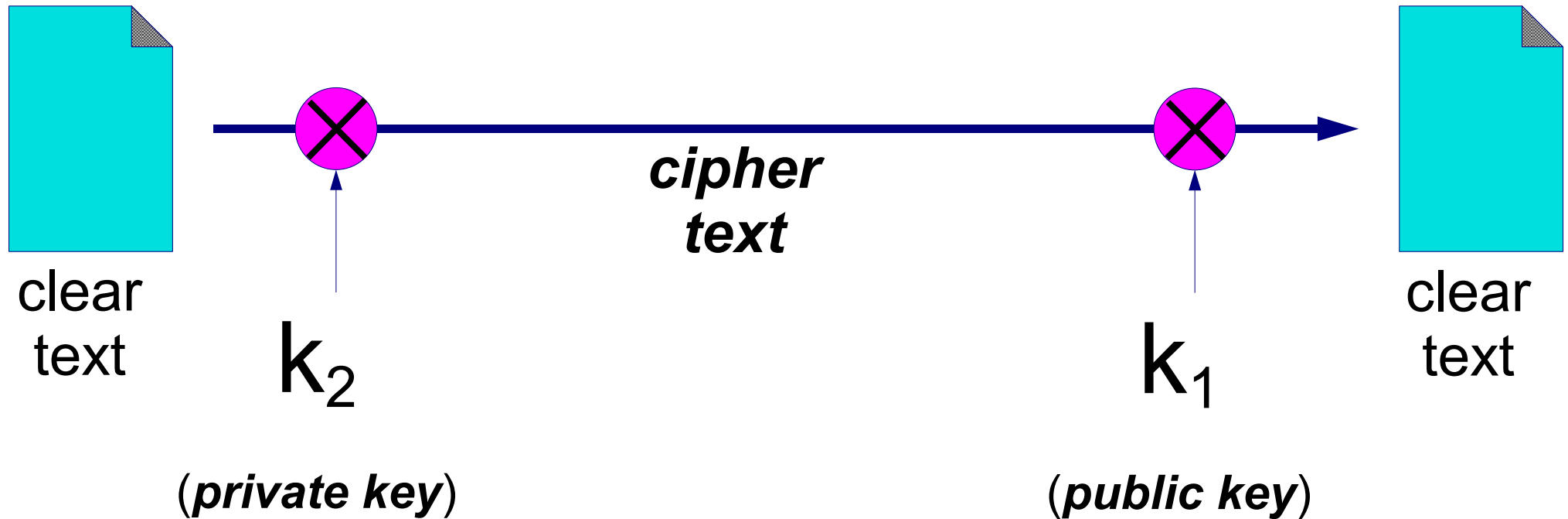
Take a hash of the document and encrypt only that. An encrypted hash is called a "digital signature"



# Digital Signatures have many uses, for example:

- E-commerce. An instruction to your bank to transfer money can be authenticated with a digital signature.
- A trusted third party can issue declarations such as "the holder of this key is a person who is legally known as Alice Hacker"
  - Like a passport binds your identity to your face
- Such a declaration is called a "certificate"
- You only need the third-party's public key to check the signature
- We'll talk about this more later.

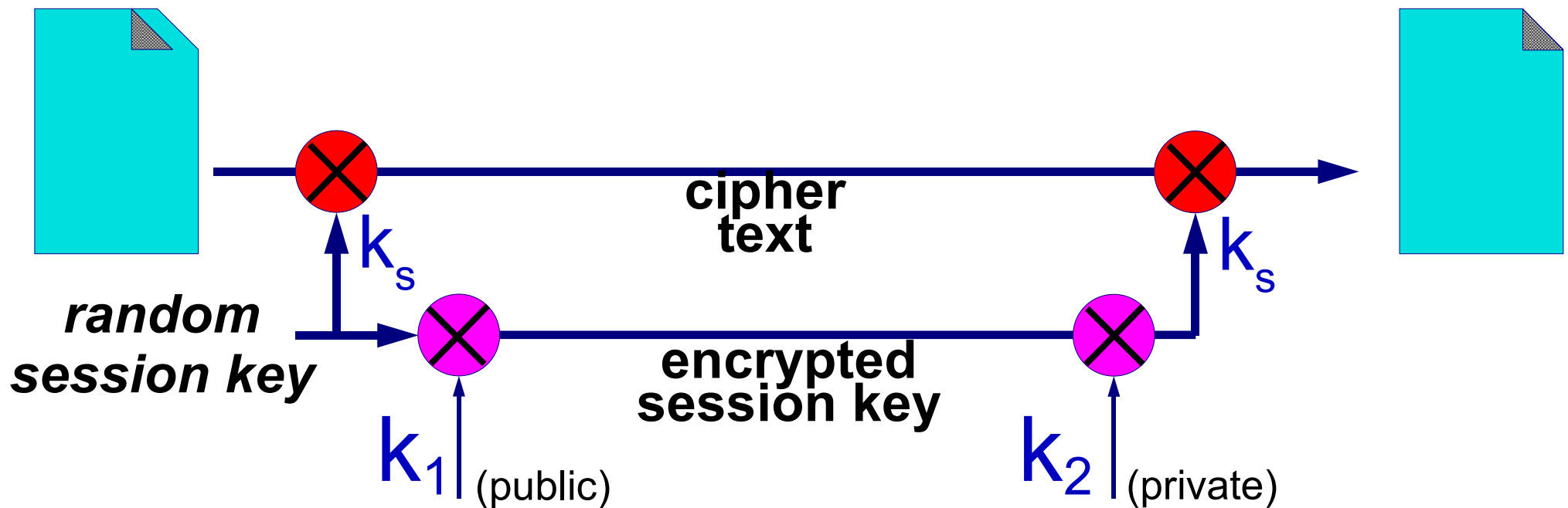
# Use for Authentication: Reverse the Roles of the Keys



If you can decrypt the document with the public key, it proves it was written by the owner of the private key (and was not changed).

# Summary

The core idea you should take away from this is how a hybrid cryptosystem works:



# Summary cont.

To view this mathematically we have:

$E$  = the encryption function

$C$  = ciphertext

$P$  = plaintext:  $M$ =Message (binary data)

$D$  = decryption function

$K_1$  = encryption key (public)

$K_2$  = different encryption key (private)

# Summary cont.

## Symmetric Cipher

$$E_{K_2}(M) = C$$

$$D_{K_2}(C) = M$$

With the property that:

$$D_{K_2}(E_{K_2}(M)) = M$$

And, with different keys (public/private) we have:

$$E_{K_1}(M) = C$$

$$D_{K_2}(C) = M$$

$$D_{K_2}(E_{K_1}(M)) = M$$



# Summary cont.

Finally – Remember, we are using *open* cryptosystems. This means that the cipher algorithm is known and available.

The security of your data rests with the key, not with keeping the cipher secret.

All Clear? :-)

## Questions?