Linux System Administration

Security through Cryptography

Three things “to get”

1. Public / Private Keys
2. Hashes
3. Digital Certificates

What's our Goal with all this?

- -- **C**onfidentiality
- -- **I**ntegrity
- -- **A**uthentication
  - Access Control
  - Verification
  - Repudiation
- -- **A**vailability

Ciphers and Keys

Cryptography Plays a *BIG* Role

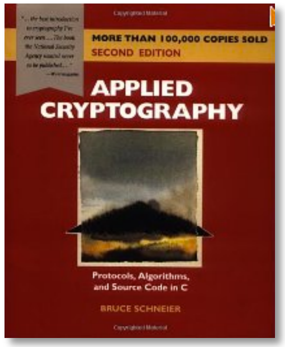
We may take it for granted, but cryptography is behind much of what we do today:

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

“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.schneir.com/crypto-gram.html
- Counterpane Security
    - http://www.counterpane.com/
- A voice of reason around much of the security hysteria we face today.


MORE THAN 100,000 COPIES SOLD
SECOND EDITION
APPLIED CRYPTOGRAPHY
Protocols, Algorithms, and Source Code in C
BRUCE SCHNEIER

Terminology

For some boring...

For others fascinating...

We understand the terminology to use the tools.

Terminology Cont.

We have ➔

- hashes/message digests
  - md5/sha1/sha2/sha3
  - collisions
- entropy (randomness)
- keys
  - symmetric
  - asymmetric (public/private)
  - length
  - distribution
  - creation
- ciphers
  - block
  - stream
- plaintext/ciphertext
- password/passphrase

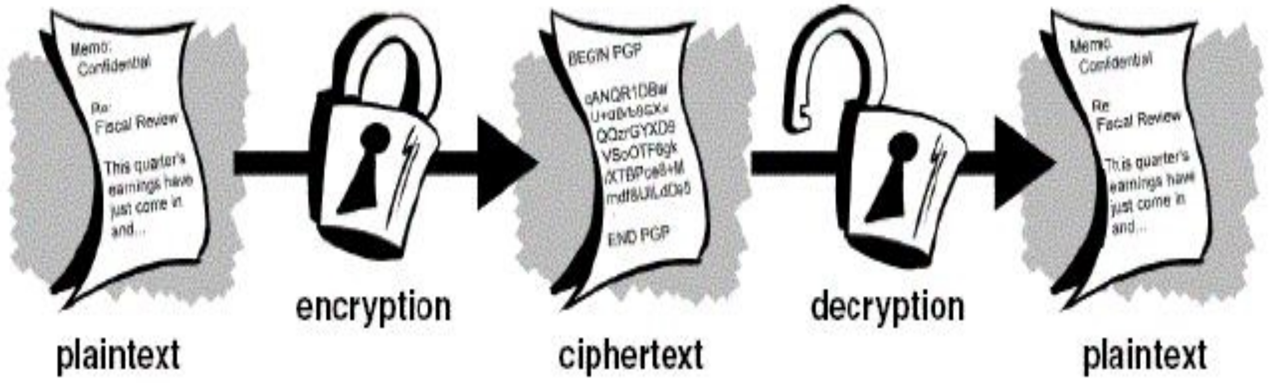
...which lead to...

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

Ciphers ➔ ciphertext:
The foundation

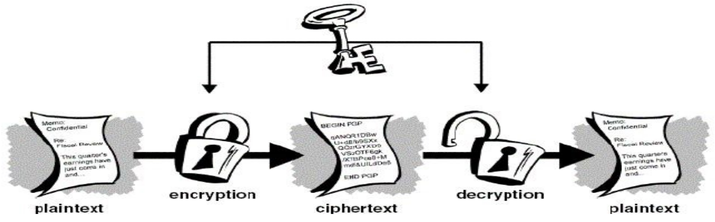
- We start with *plaintext*. Something you can read.
- We apply a mathematical algorithm to the plaintext.
- The algorithm is the *cipher*.
- The plaintext is turned in to *ciphertext*.
- Almost all ciphers were secret until recently.
- Creating a secure cipher is *HARD*.

What it Looks Like



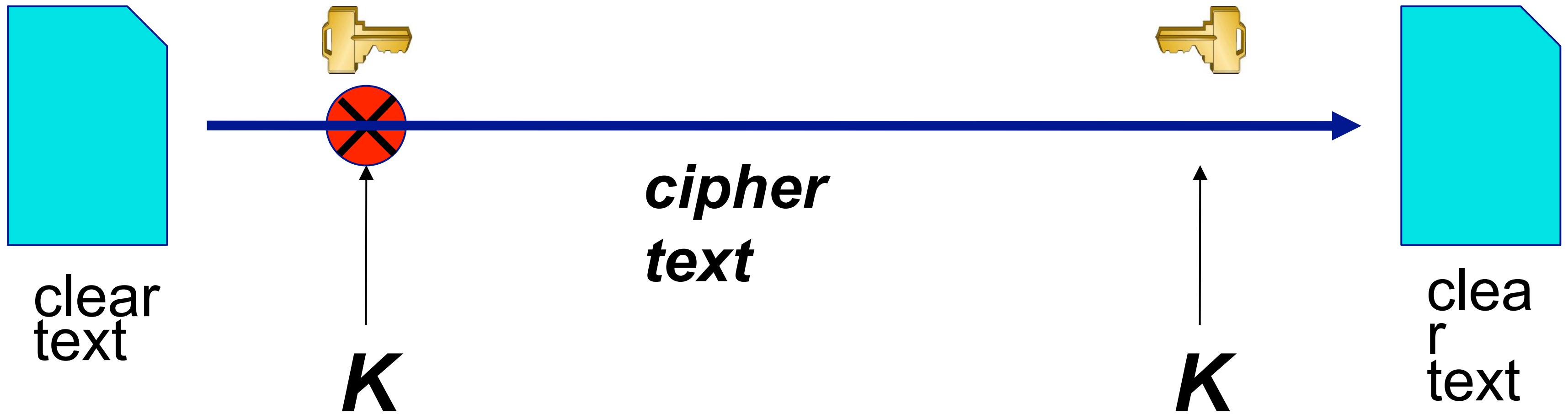
Keys

- Ciphertext and back to plaintext – Apply a key.
- Security of ciphertext with key. Lost key = compromised data.
- This is a *private key*.
- This type of cipher system is efficient for large amounts of da
- This is a *symmetric cipher*.



Symmetric Cipher

Private Key/Symmetric Ciphers



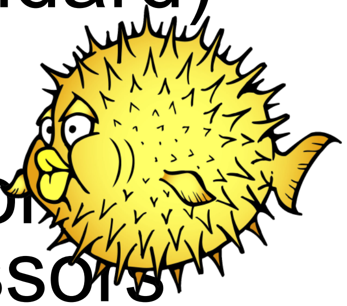
The same key is used to encrypt the document before sending and to decrypt it once it is received

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 processors

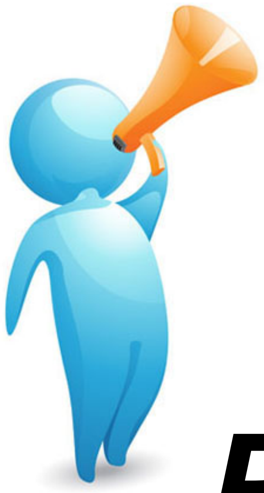
IDEA - 128 bits, patented (requires a license for commercial use)

Features of Symmetric Ciphers

- Fast to encrypt and decrypt, suitable for large volumes of data
- Brute force attack only to crack.
- Problem - how do you distribute the keys?

Problem: How do you distribute the keys?

Answer



Public / Private Keys!

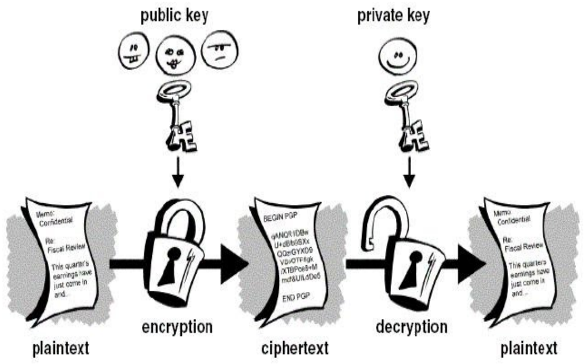
Public/Private Keys

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

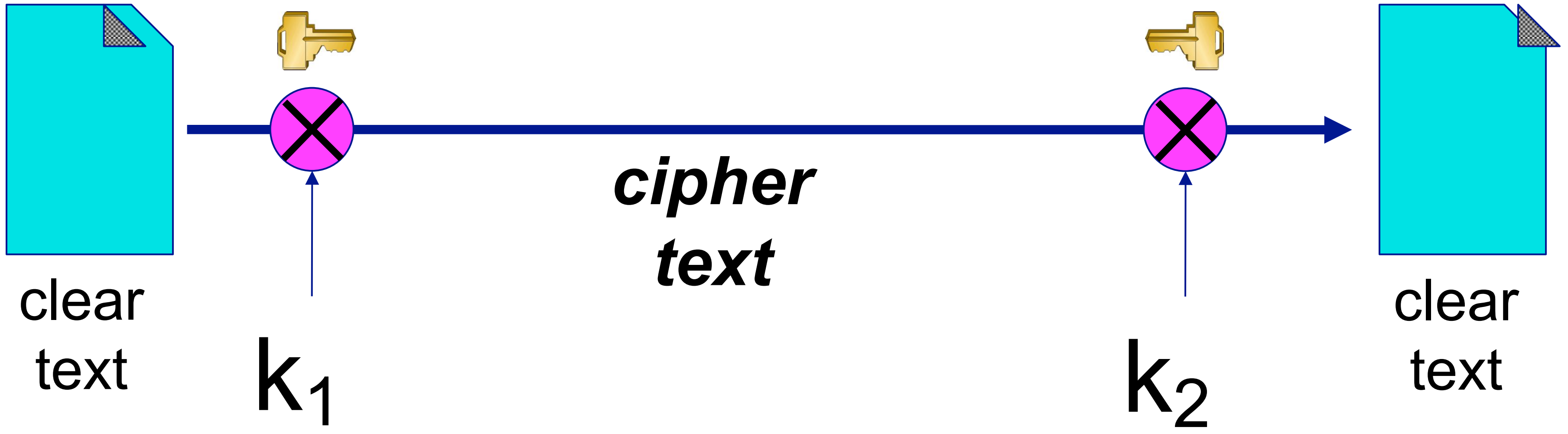
The *private key* remains secret and should be protected.

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 dat Only someone with the *private key* can decrypt.



Example: Public/Private key pair



(***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!

Less Efficient & Attackable

- Symmetric *much* more efficient. About 1000x > public 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).
- We’ll see how to use this combination.

Remember, symmetric cipher attack is to steal the private key...

Hashing – Checksums - Digests

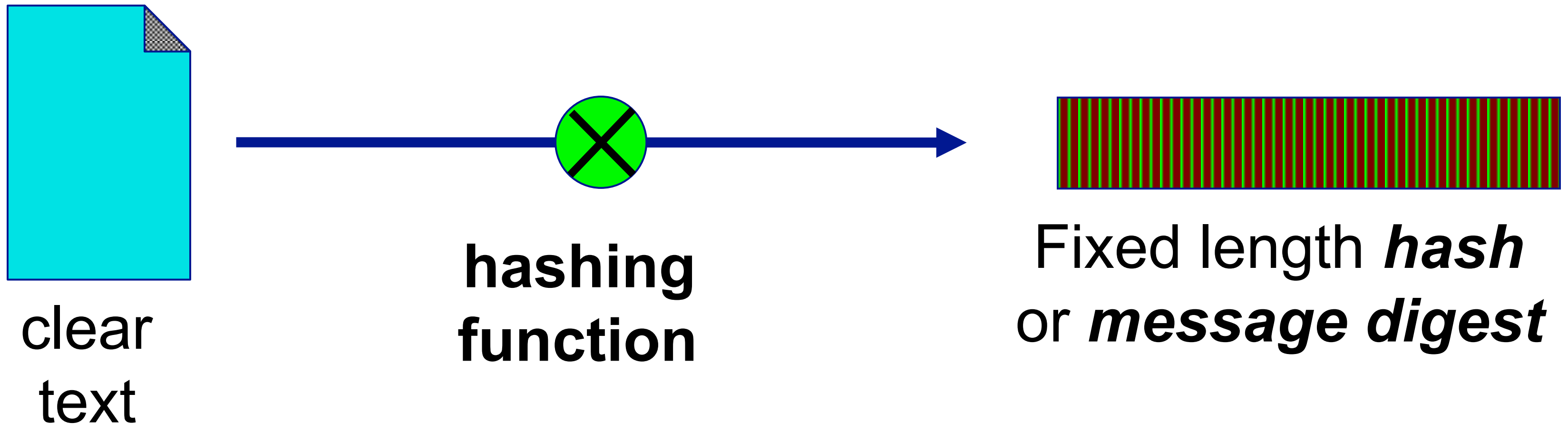
One-Way Hashing Functions

- Mathematical function that generates a fixed length result regardless of amount of data used.
- Cannot generate original data from fixed-length result.
- Two sets of data that produce the same fixed-length result. are called *collisions*.

Hashing Function Examples

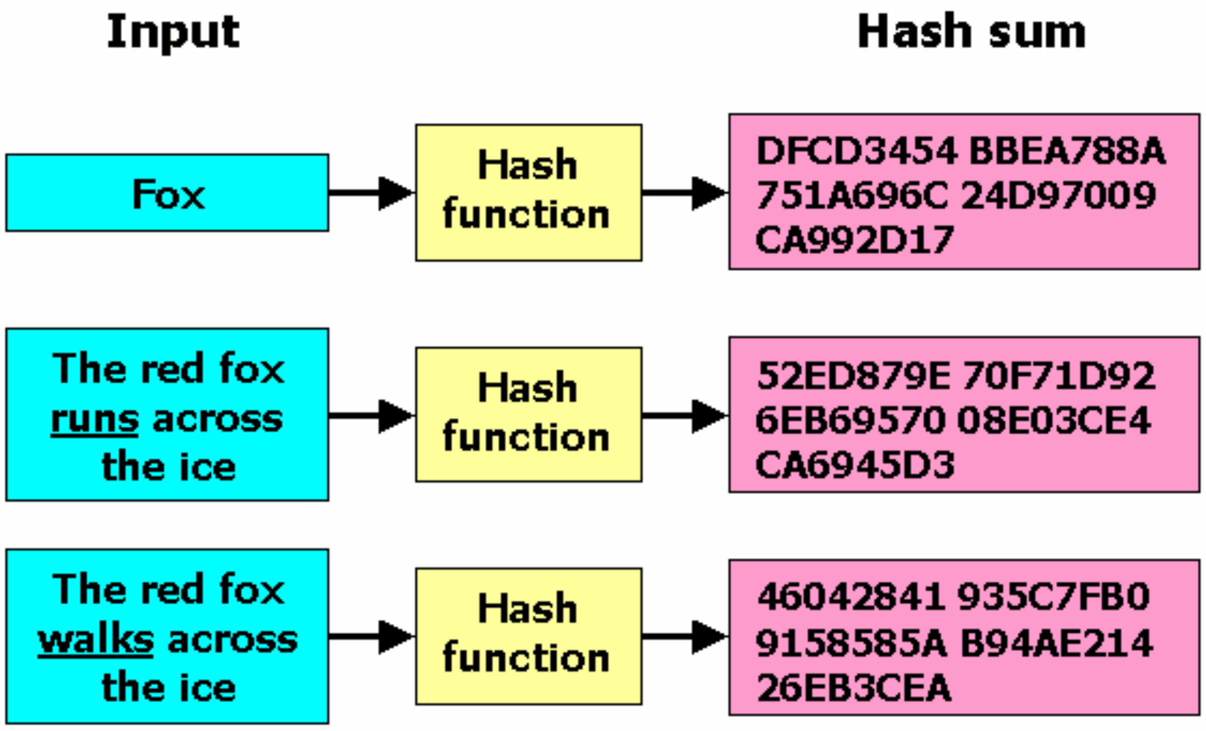
- Unix ***crypt***() function, based on DES, 56 bits (***not secure)***
- ***MD5*** (Message Digest 5) - 128 bit hash (**deprecated**)
- ***SHA-1*** (Secure Hash Algorithm) - 160 bits (**deprecating**)
- ***SHA-3*** (Secure Hash Algorithm 3) – 256-1024 bits (**upcoming)**
- Still no feasible method to create any document which has a given digest (hash sum, checksum).

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


Input
Hash sum
Fox
Hash function
DFCD3454 BBEA788A 751A696C 24D97009 CA992D17
The red fox runs across the ice
Hash function
52ED879E 70F71D92 6EB69570 08E03CE4 CA6945D3
The red fox walks across the ice
Hash function
46042841 935C7FB0 9158585A B94AE214 26EB3CEA

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.

One-Way Hashing Functions

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

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

What use is this?

- You can run many megabytes of data through a hashing function, but only have to check 160* bits of information. A compact and *unique document signature*.*
- Generate a *passphrase* for your data – such as your private key. If someone gets your private key, they still must know your passphrase to decrypt anything using your private key.
- This is how Unix, Linux and Windows protect user passwords (but not effectively).

What use is this?

- You can run many megabytes of data through a hashing function, but only have to check 160* bits of information. A compact and *unique document signature*.*
- Generate a *passphrase* for your data – such as your private key. If someone gets your private key, they still must know your passphrase to decrypt anything using your private key.
- This is how Unix, Linux and Windows protect user passwords (but not effectively).

Review

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

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

Let's give it a try

Munge a document...

Connect to your machine and become root:

```
$ sudo bash
```

Copy a file and run the sha1sum hashing function on it:

```
# cp /etc/motd .
# sha1sum motd
```

Make note of the result. Edit the file and change 1 character:

```
# vi motd
```

Save the file and run sha1sum again:

```
# sha1sum motd
```

- How different were the results?
- A good hashing function changes message digest significantly for small differences

Hybrid Systems & Digital Signatures

Hybrid Systems

- Symmetric Ciphers encrypt lots of data securely and quickly.
- Public key systems encrypts lots of data very slowly in order to be secure.
- Public keys, however, let us solve the private key distribution problem.
- How do we take advantage of this...?

Hybrid Systems

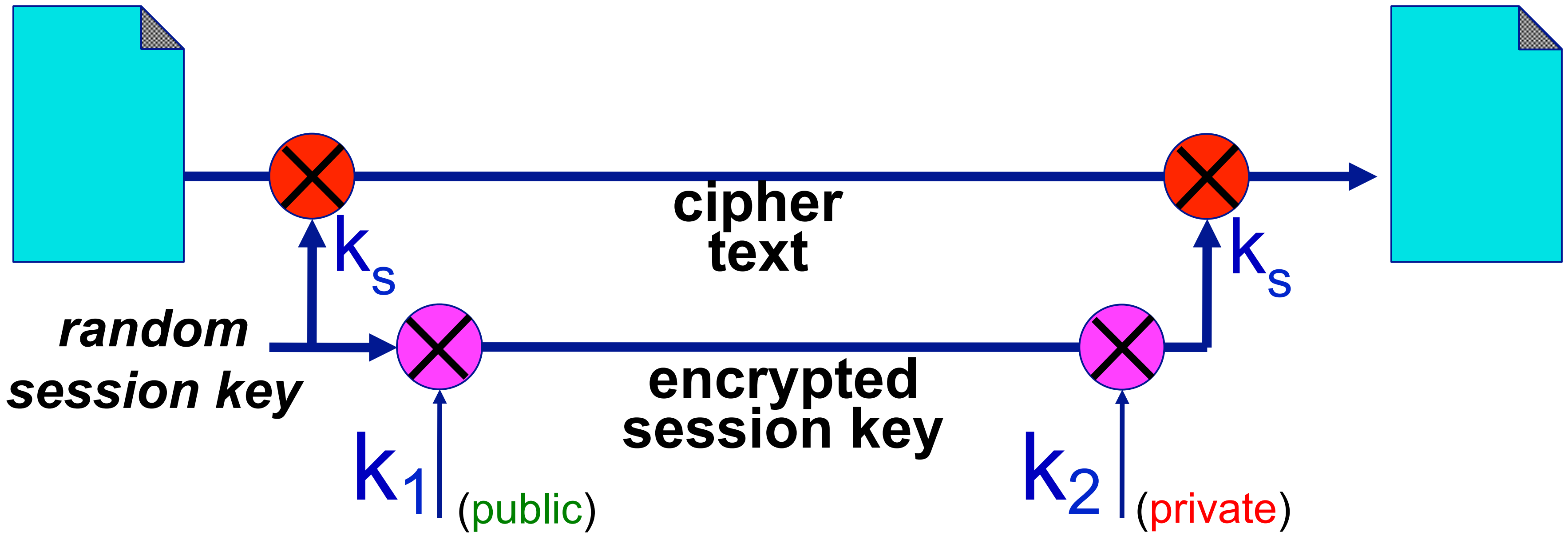
...we do this:

- Start with a symmetric cipher on one side.
- 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.

(more trumpets, fireworks, etc...)

Hybrid Systems

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



Hybrid Systems cont...

Two things should (imho) stand out:

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

Any ideas?

Hybrid Systems cont...

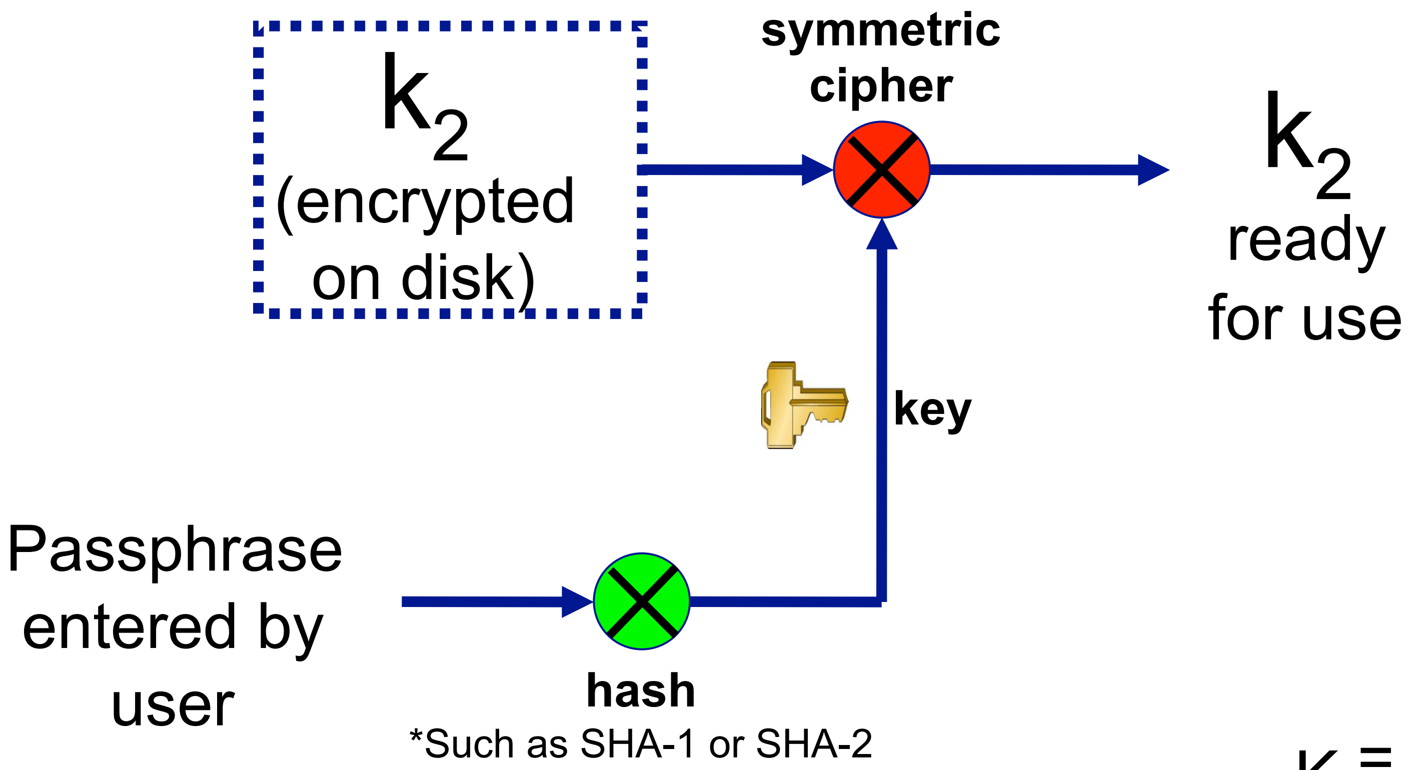
- "*Send it to the other side, decrypt the one-time key.*" How?

  Use your ***private key***.

- What about protecting your private key?

  Encrypt it using a ***hash function***.

Protecting the Private Key



K_2 = ***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

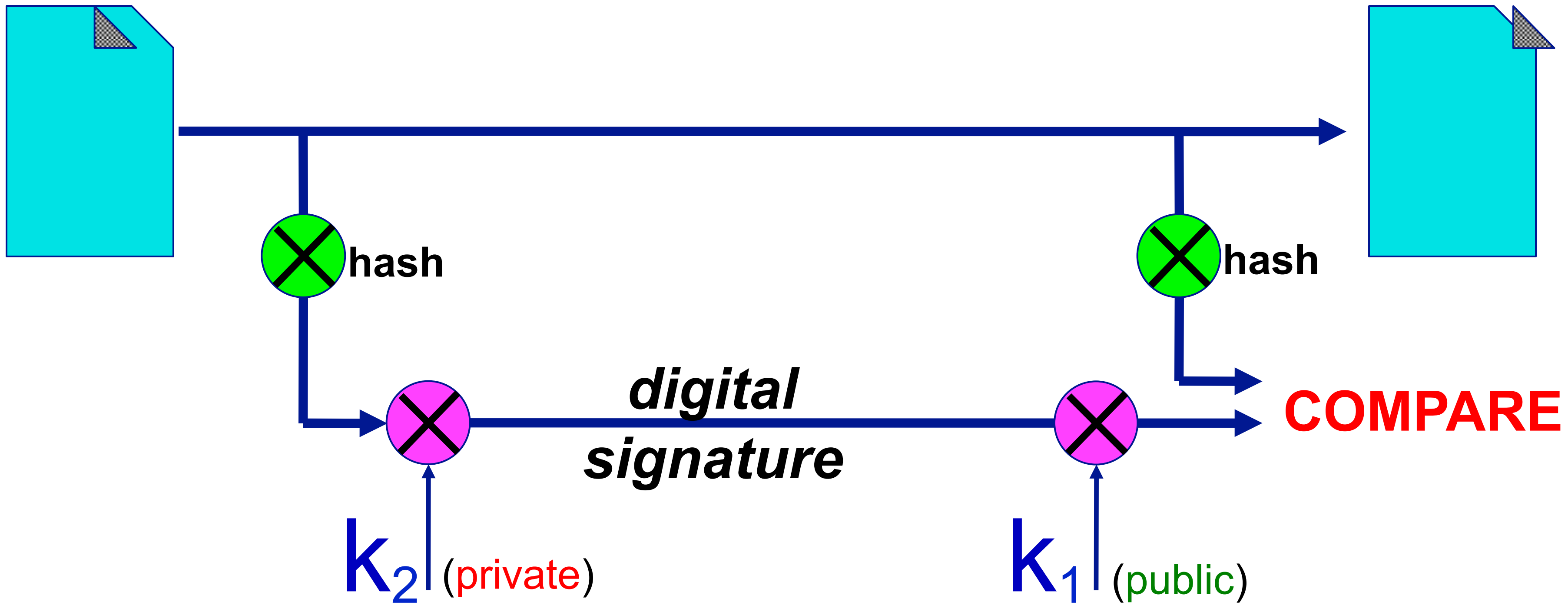
Digital Signatures

Let's reverse the role of public and private keys. To create a digital signature on a document:

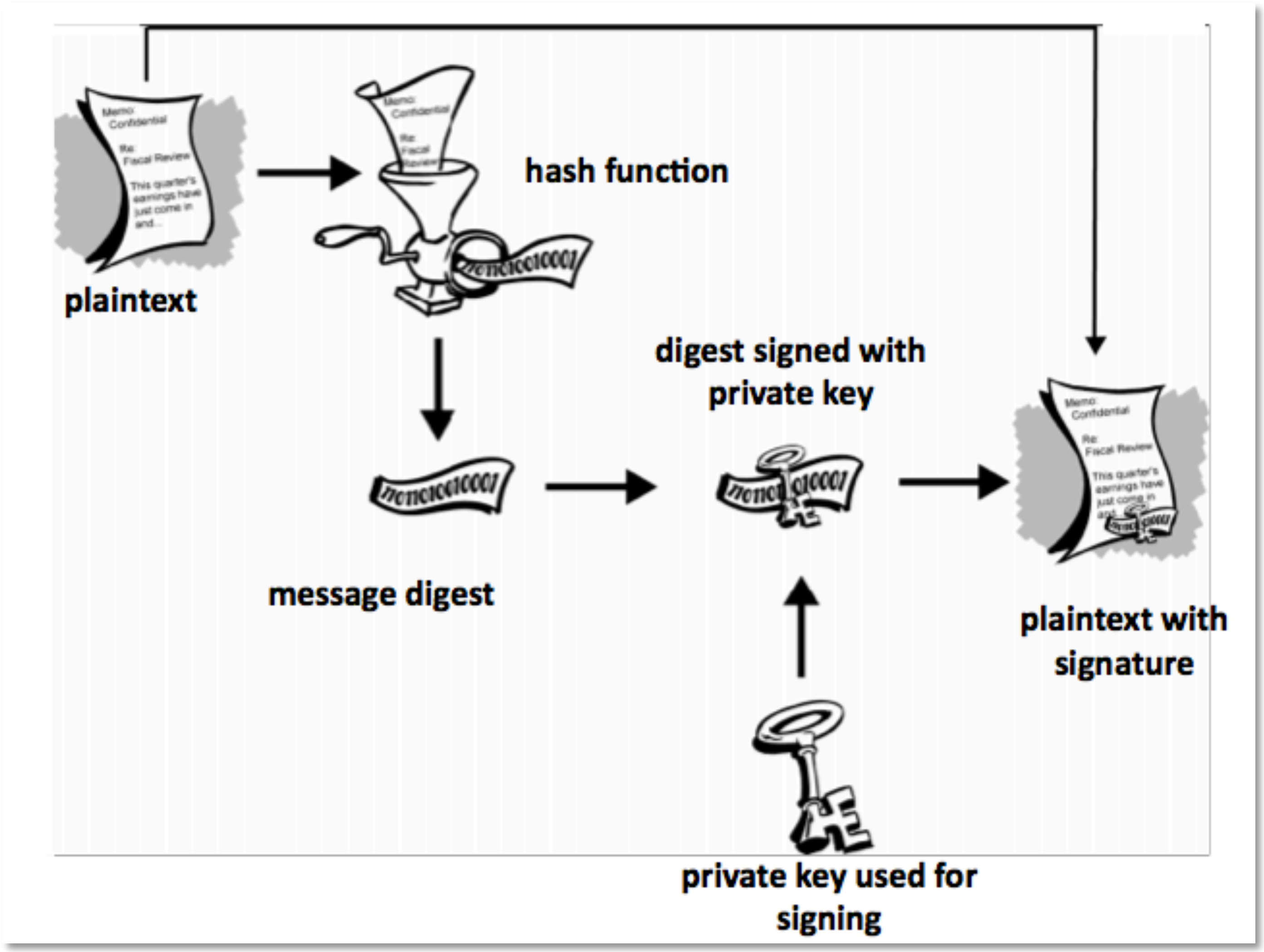
- *Munge* a document.
- Encrypt the *message digest* with your private key.
- Send the document plus the encrypted message digest.
- On the other end *munge* the document *and* decrypt the encrypted message digest with the person's public key.
- If they match, the document is authenticated.

Digital Signatures cont.

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



Another View



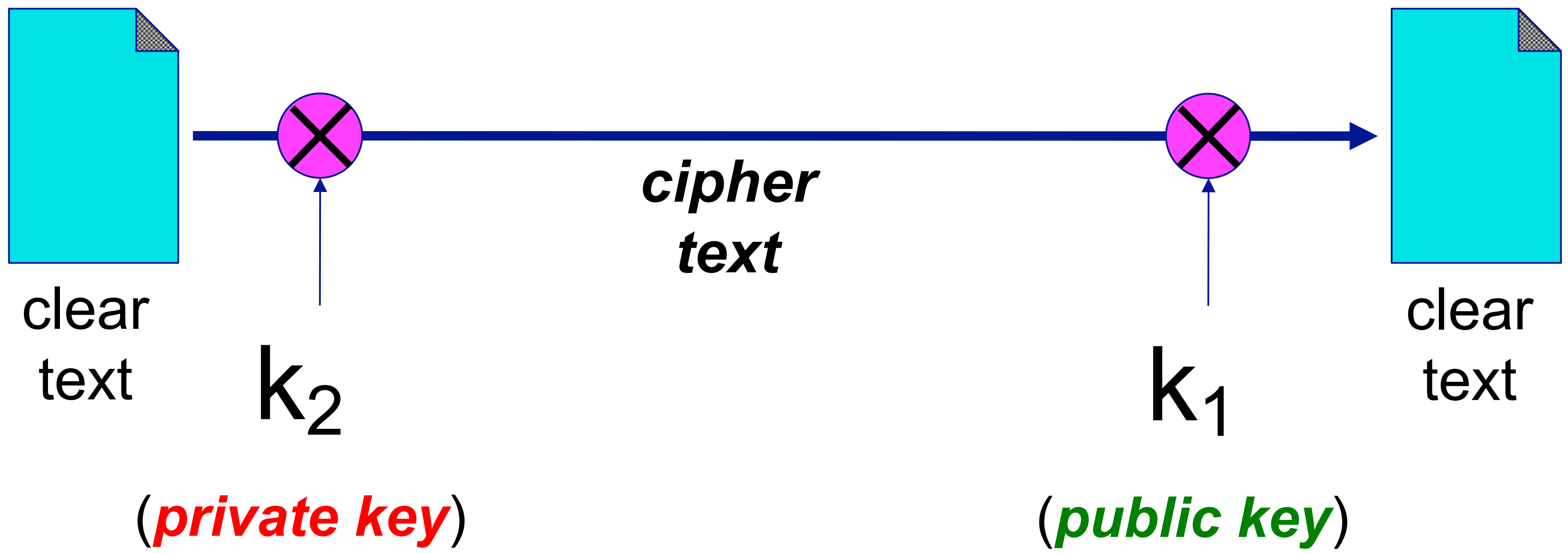
Digital Signatures & many uses

- 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.

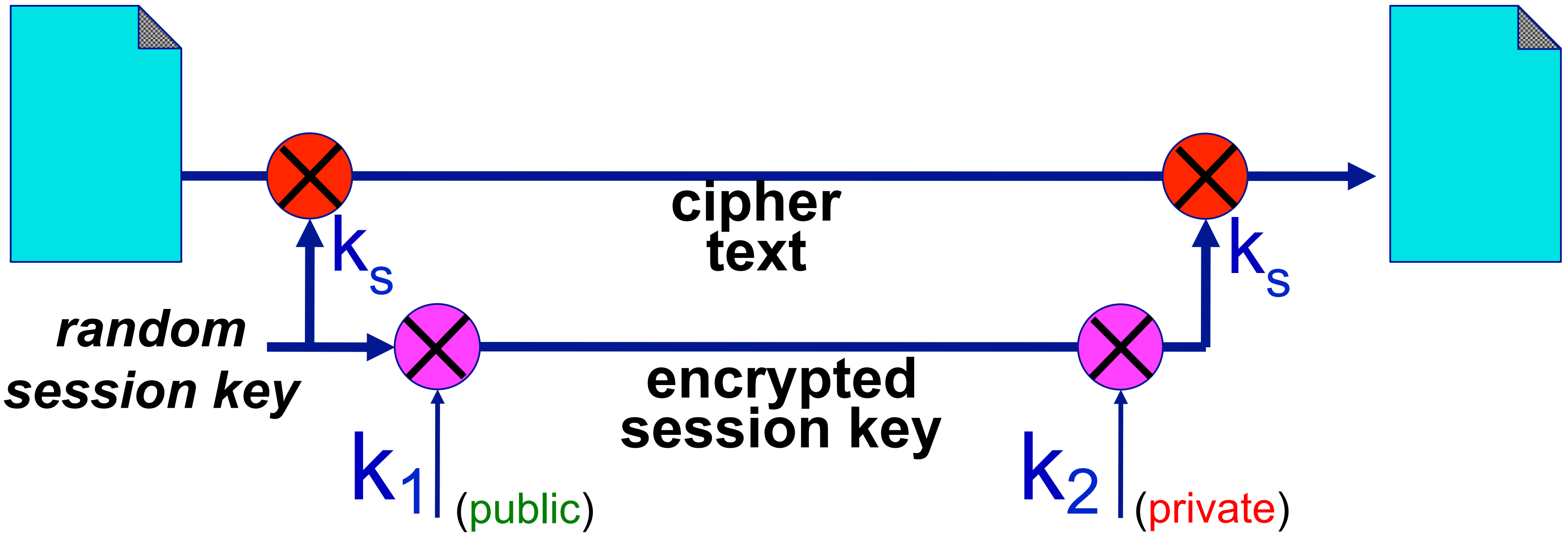
Use for Authentication: Reverse the Roles of the Keys



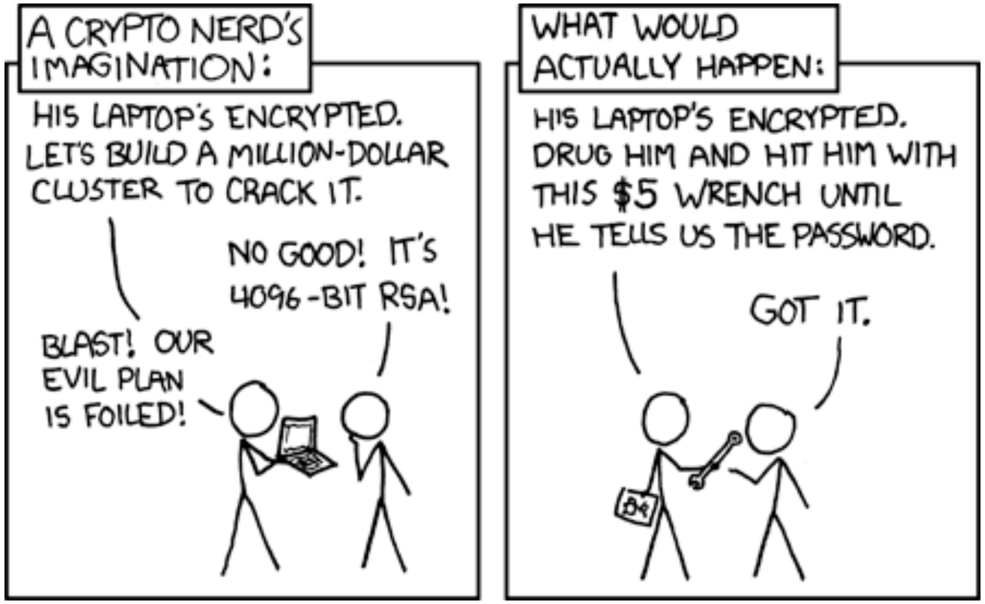
If you can decrypt the document with the public key, it proves it was written by the owner of the private key (and was not changed).

Summary

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



Back in the real world...


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

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 private key, not with keeping the cipher secret.

All Clear? :-)

Questions?

Questions

?