DNSSEC Cryptography Review

Track 2 Workshop

July 3, 2010
American Samoa

Hervey Allen

DNSSEC and Cryptography

Three Key Concepts

- Public / Private keys
- Message digests, checksums, hashes
- Digital signatures

Are at the core of DNSSEC. If these do not make sense, then DNSSEC will not make sense.

Ciphers ==> ciphertext

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

We apply a mathematical algorithm to the plaintext.

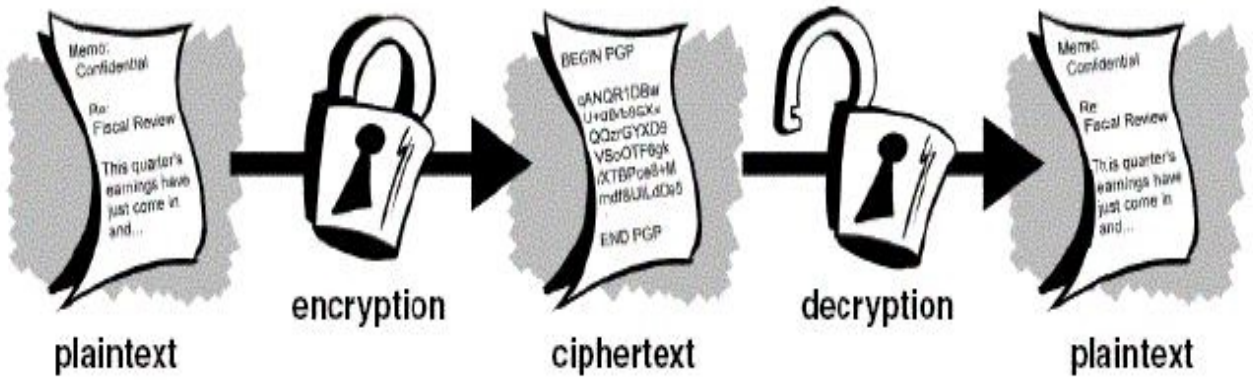
The algorithm is the *cipher*.

The plaintext is turned in to *ciphertext*.

Almost all ciphers were secret until recently.

Creating a secure cipher is *HARD*.

What it Looks Like



Keys

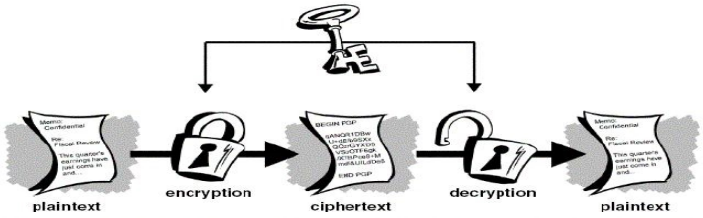
To create ciphertext and turn it back to plaintext we apply a key to the cipher.

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*.

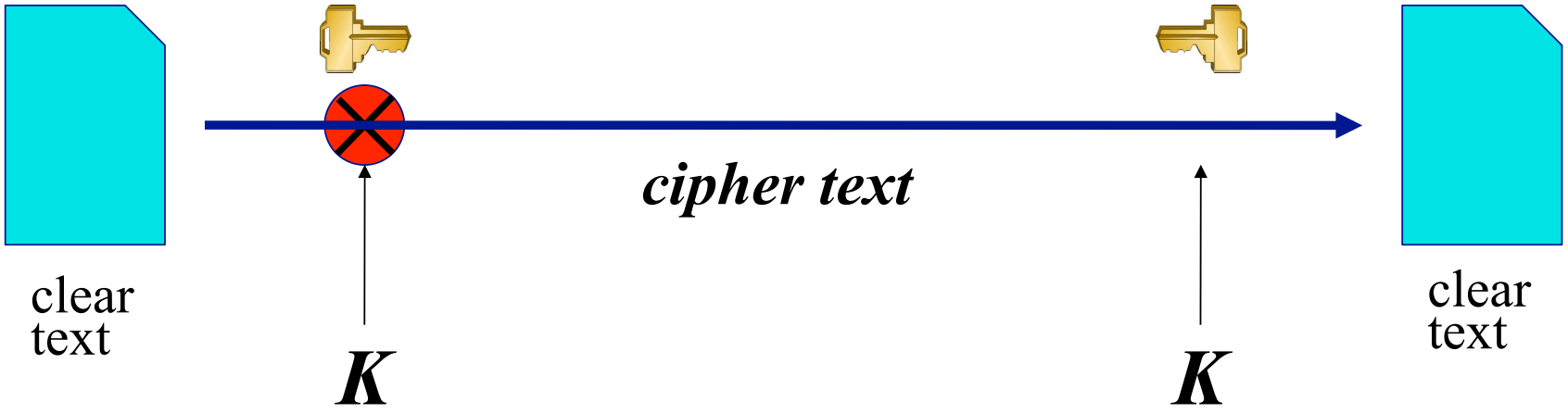
This type of cipher system is efficient for large amounts of data.

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

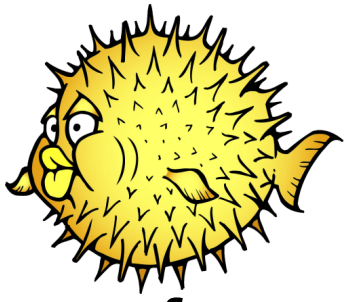
Examples of Symmetric Ciphers

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 around 128 bits, for data protection around 20 years.*
- Problem - how do you distribute the keys?

*See http://www.keylength.com/ for a good and fun discussion.

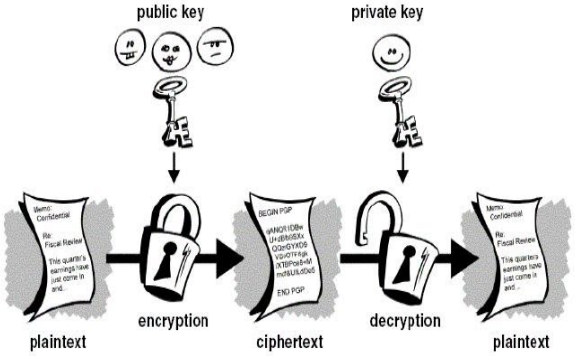
Public/Private Keys

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

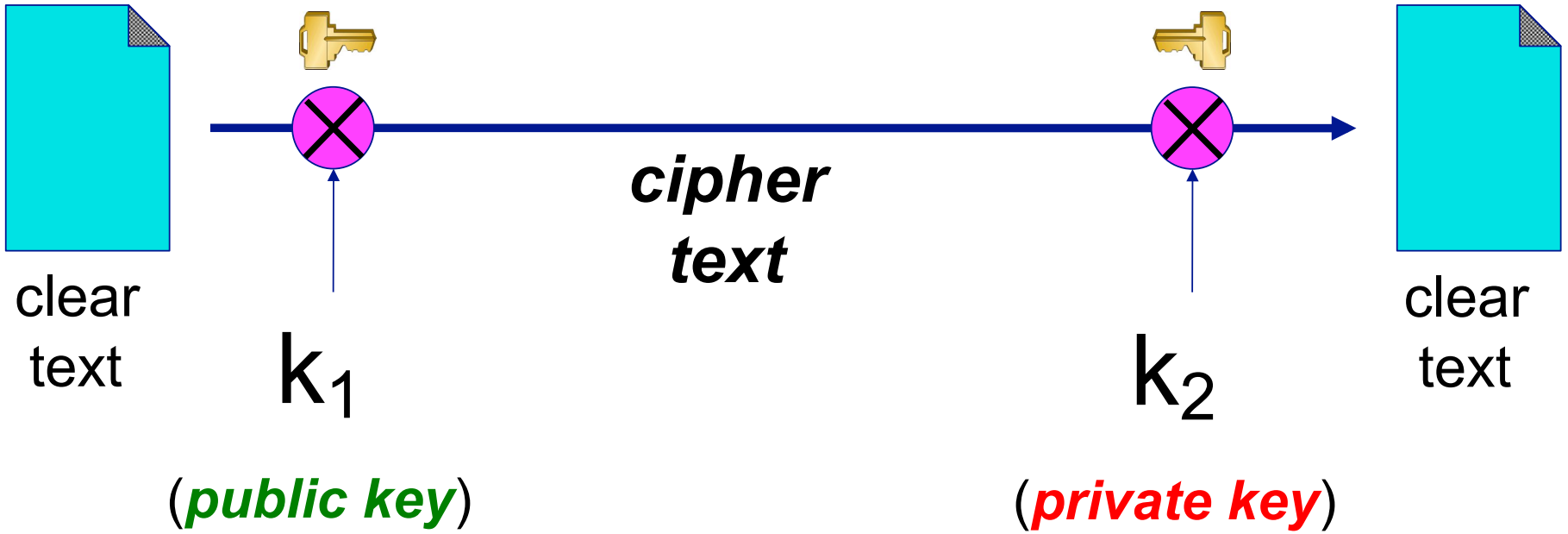
The *private key* remains secret and should be protected.

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):



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 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).

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

Public Key Attack Method

Mathematically we have:

E = the encryption function

C = ciphertext

P = plaintext

K = private key

$C = E_K(P)$

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

Hybrid Systems

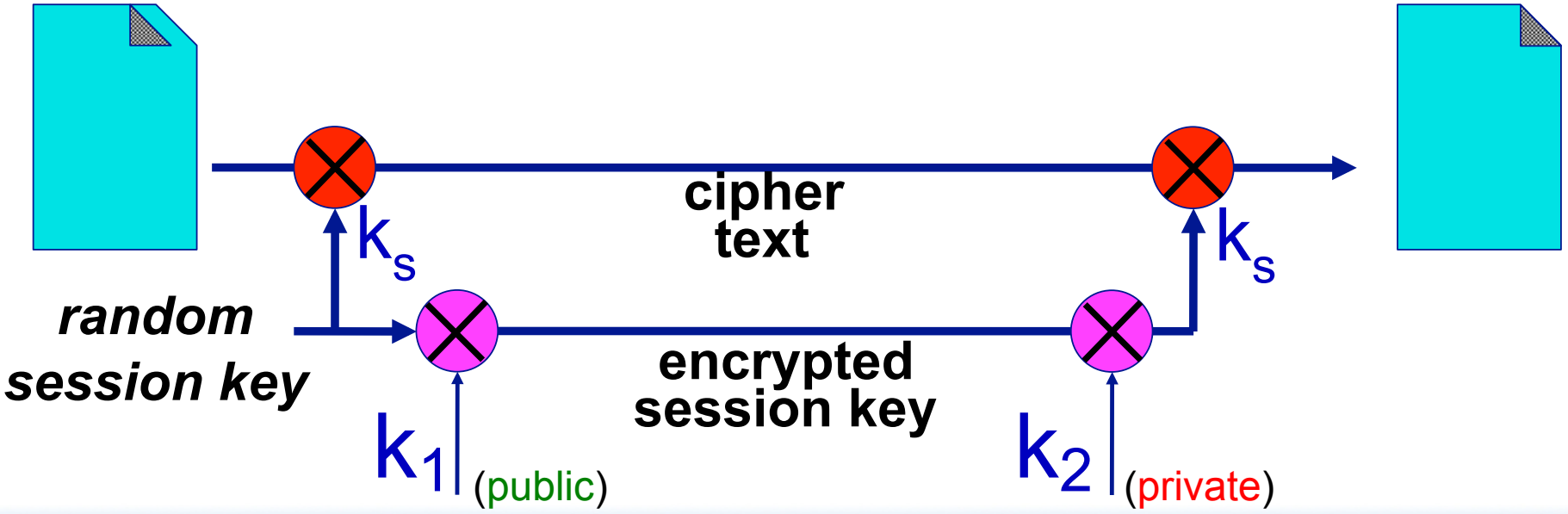
Symmetric Ciphers are not vulnerable in the previous way. The key length can be much shorter.

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...

- "*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.

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*.

Hashing Function Examples

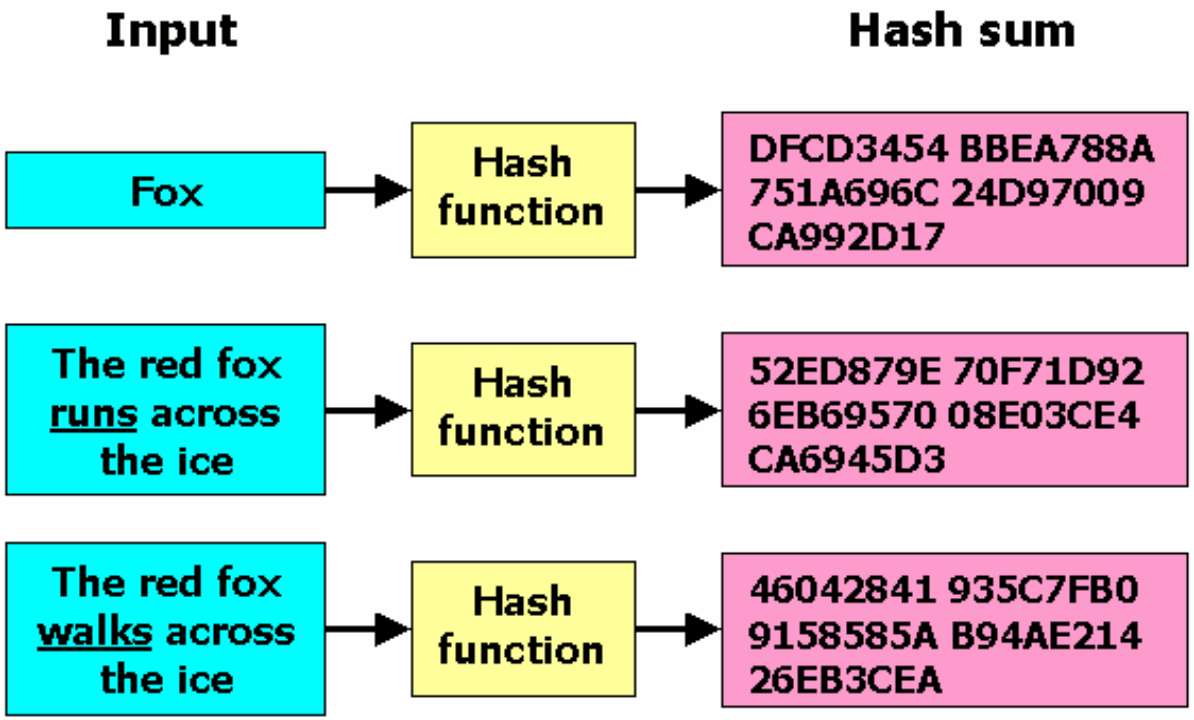
- Unix ***crypt***() function, based on DES, 56 bits (***not secure)***
- ***MD5*** (Message Digest 5) - 128 bit hash (**deprecated**)
- ***SHA1*** (Secure Hash Algorithm) - 160 bits
- Until August 2004, no two documents had been discovered which had the same MD5 digest.
  - *No collisions have yet been found in SHA-1, but it is now known to be compromised and will likely be phased out in the next few years. See http://en.wikipedia.org/wiki/SHA for details.*
- Still no feasible method to create any document which has a given MD5 digest

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.

One-Way Hashing Functions cont.

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

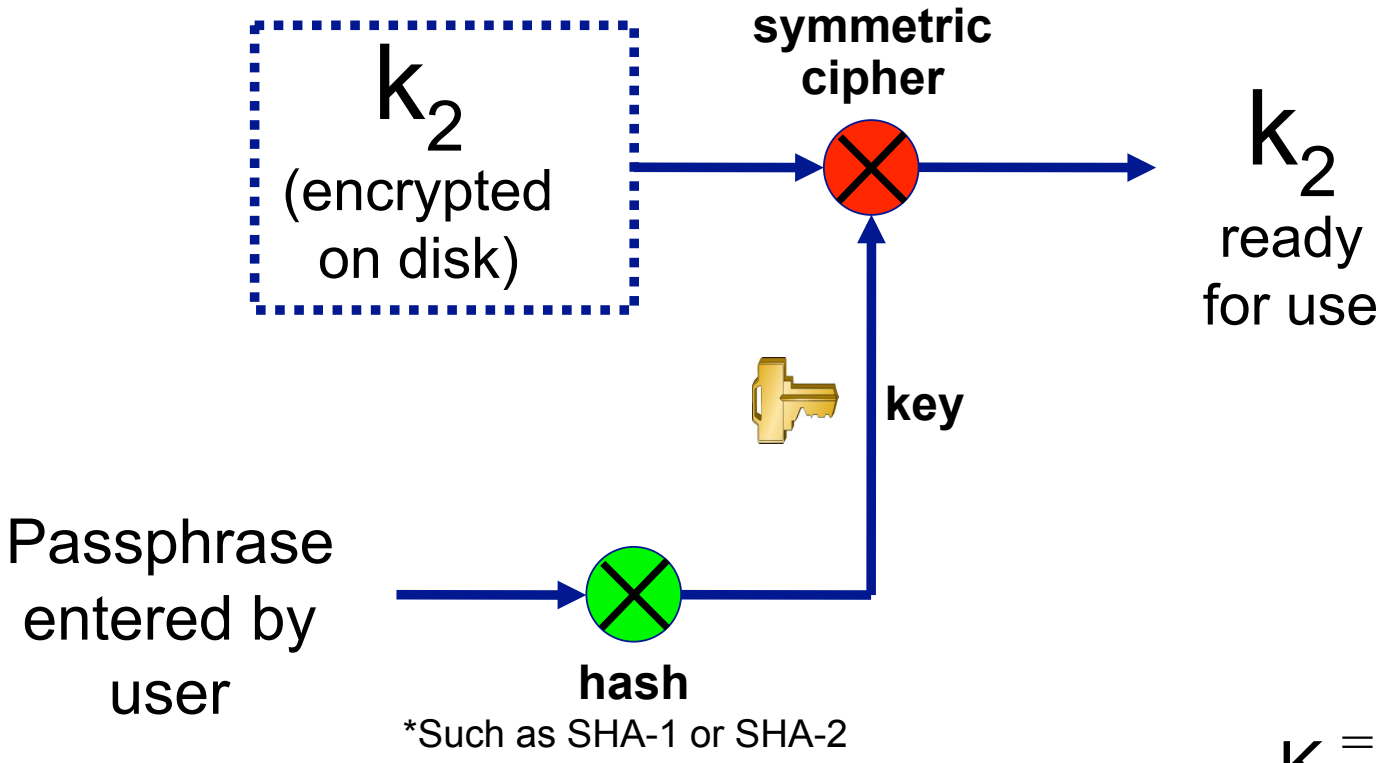
The fixed-length result is referred to as a *checksum, fingerprint, message digest, signature, digest, hash, hash sum…*

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*.*
- You can 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).

* May increase after 2012 if a new SHA-3 algorithm is approved for use.

Protecting the Private Key



K_2 = ***private key***

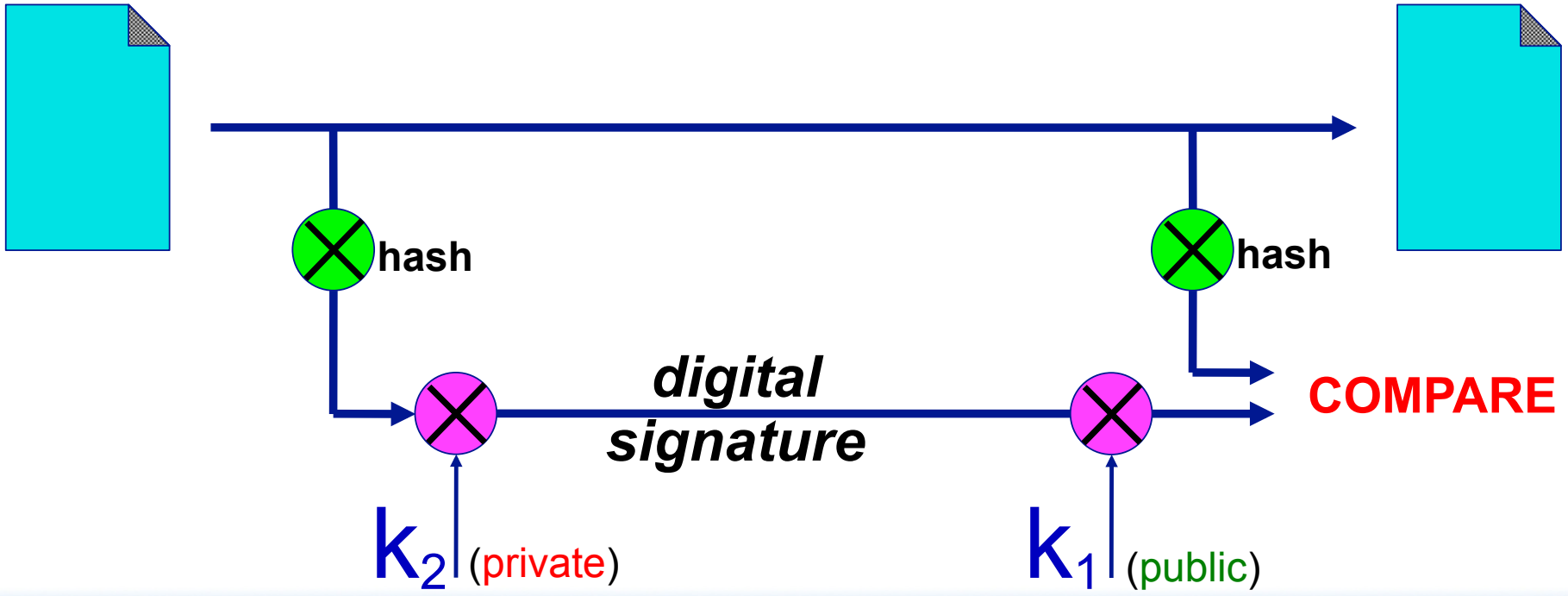
Digital Signatures

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

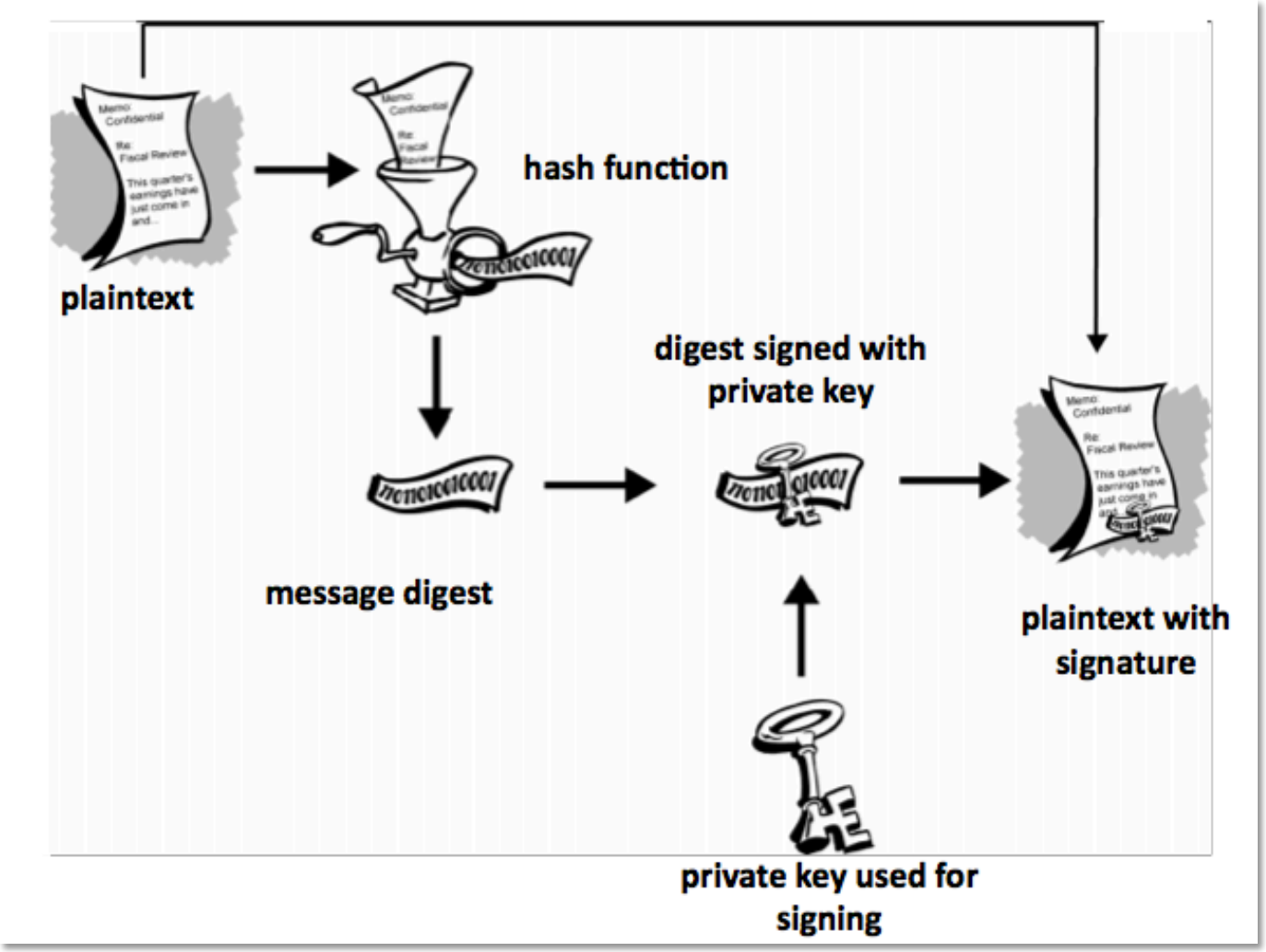
- *Munge* a document.
- Encrypt the *message digest* with your private key.
- Send the document plus the encrypted message digest.
- On the other end munge the document *and* decrypt the encrypted message digest with the person's public key.
- If they match, the document is authenticated.

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 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”
- And, of course, we sign records in the DNS to prove they are authentic, unchanged and come from a trusted source.