DNSSEC Tutorial: Public / Private Key Refresher

Hervey Allen
Phil Regnauld

26 February 2009
Manila, Philippines

http://nsrc.org/tutorials/2009/apricot/dnssec/

Public-Private Keys Refresher

- Ciphers
- Ciphertext
- Symmetric Cipher / Private Key
- Public Key
- Hashing functions
- Hash / message digest
- Digital Signatures

Why rsa vs. dsa, sha-1 or sha-256 vs. md5?

Ciphers ==> ciphertext

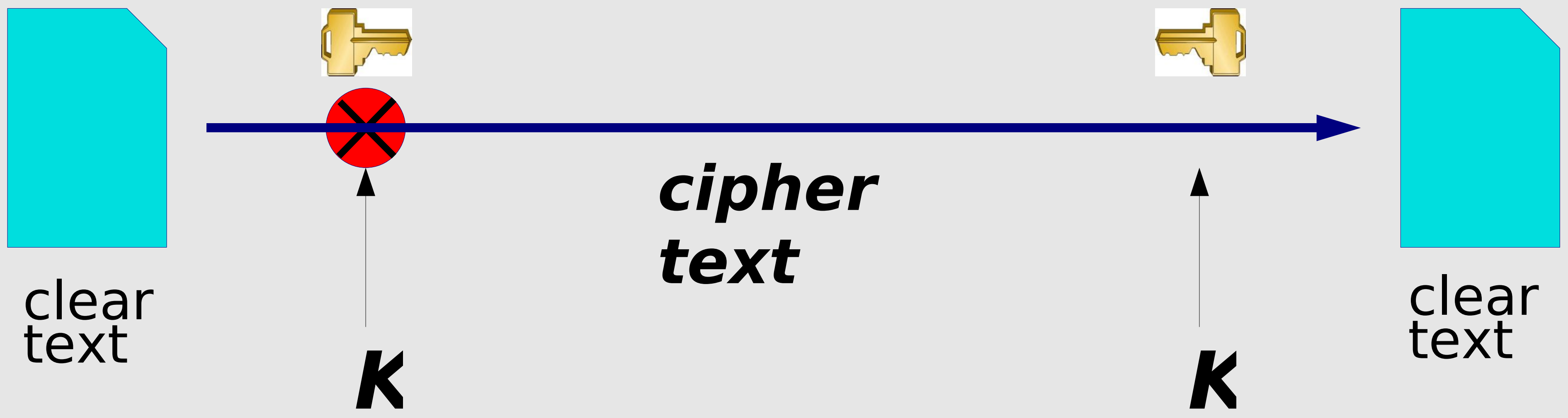
- We start with *plaintext*. Something you can read.
- We apply a mathematical algorithm to the *plaintext*.
- The algorithm is the *cipher*.
- The *plaintext* is turned in to *ciphertext*.
- Almost all *ciphers* were secret until recently.
- Creating a secure *cipher* is *HARD*.

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 obtains 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

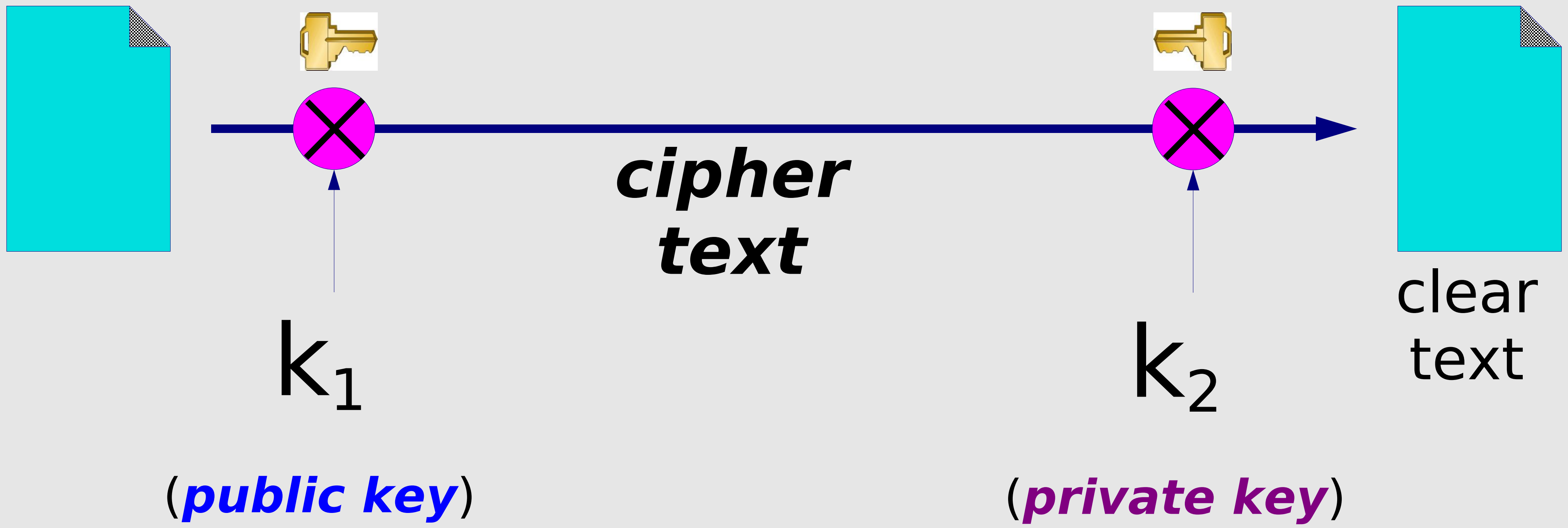
Features of Symmetric Ciphers

- 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. Symmetric Ciphers in use exceed this length:
  - 3DES = 112 bits, Blowfish = 128 bits, etc.
- 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 the encrypted data.

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!

Issues

- *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).
- For larger data transmissions than used in DNSSEC we use *hybrid systems*. That's another discussion.

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.

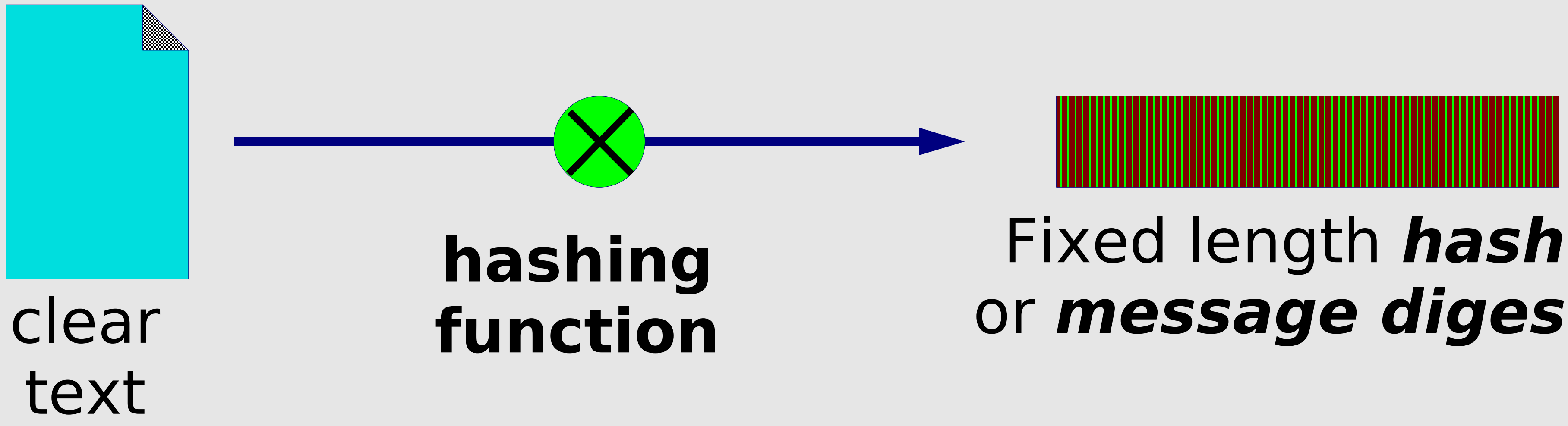
Some popular hashing functions include:

- **md5**: Outputs 128 bit result. Fast. Collisions found.
- **sha-1**: Outputs 160 bits. Slower. Collisions in 2^{x69}.
- **sha-2**: Outputs 224-512 bits. Slower. Collisions expected (2^{x80} attack).
- **sha-3**: TBA: Currently in development via a new *NIST Hash Function Competition.*

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

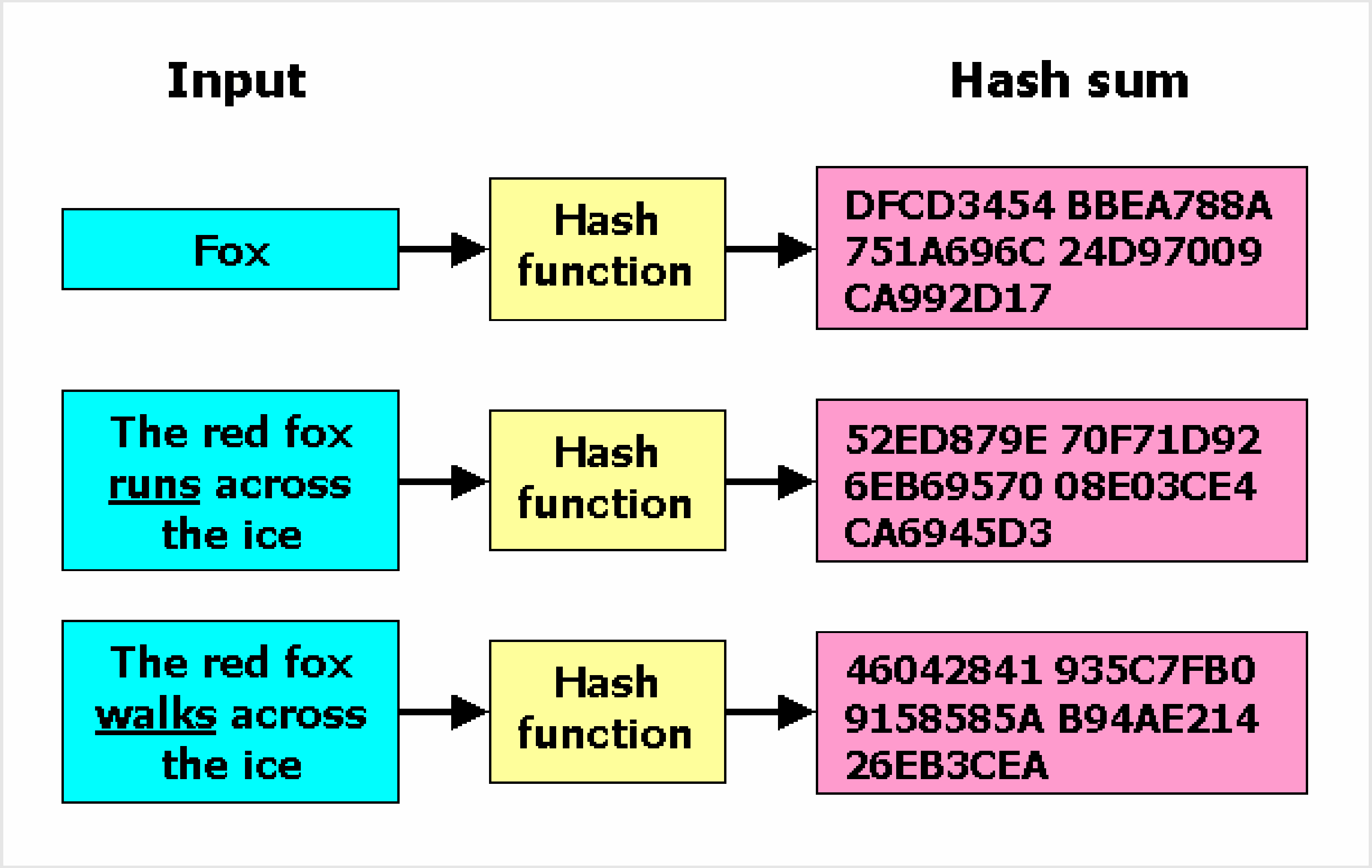
The fixed-length result is referred to as a *checksum, fingerprint, message digest,* etc.

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 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

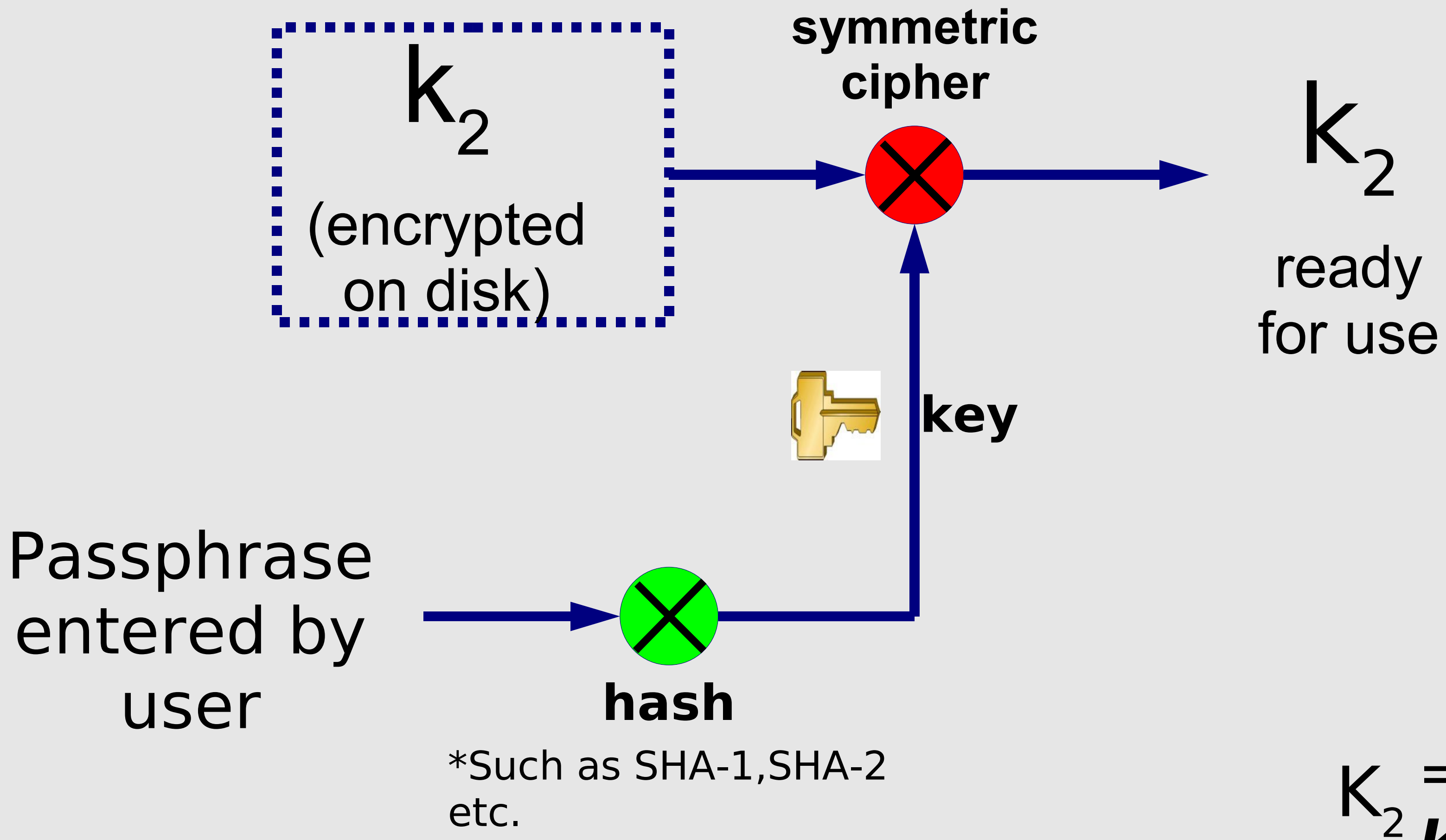
Note the significant change in the hash sum for minor changes in the input. Note that the hash sum is the same length for varying input sizes. This is extremely useful.

*Image courtesy Wikipedia.org.

What use is this?

- You can run many megabytes of data through a hashing function, but only have to check a fixed number of bits of information (160-512 bits). 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.

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.

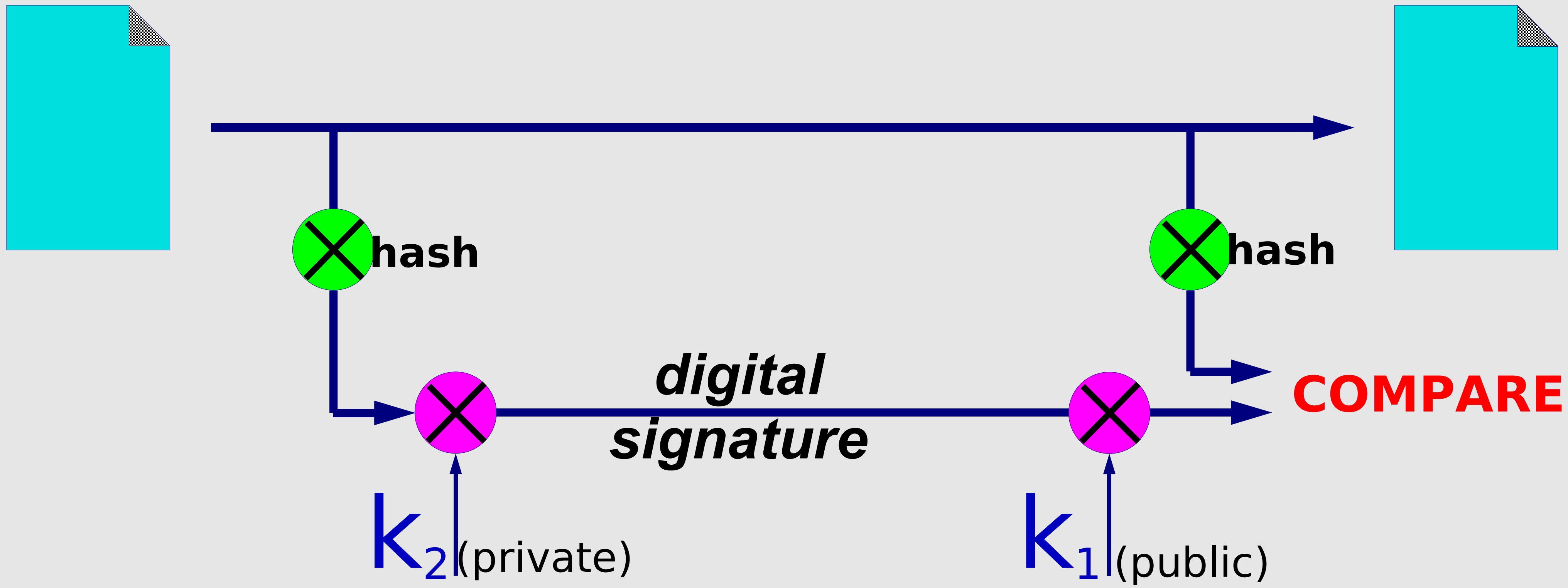
Digital Signatures

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*"



Our Original Question Revisited

Let's answer our original question...

Why rsa vs. dsa, sha-1 or sha-256 vs. md5?

- rsa can do 2048 bits and greater
- dsa max bits is 1024
- md5 has collisions
- sha-1 / sha-2 collisions expected, but more secure for the moment.

Conclusion

- Public / Private keys
- Message digests
- Digital signatures

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