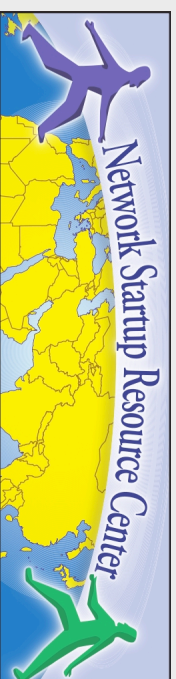


DNSSEC Tutorial

Public / Private Keys



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.

Ciphertext

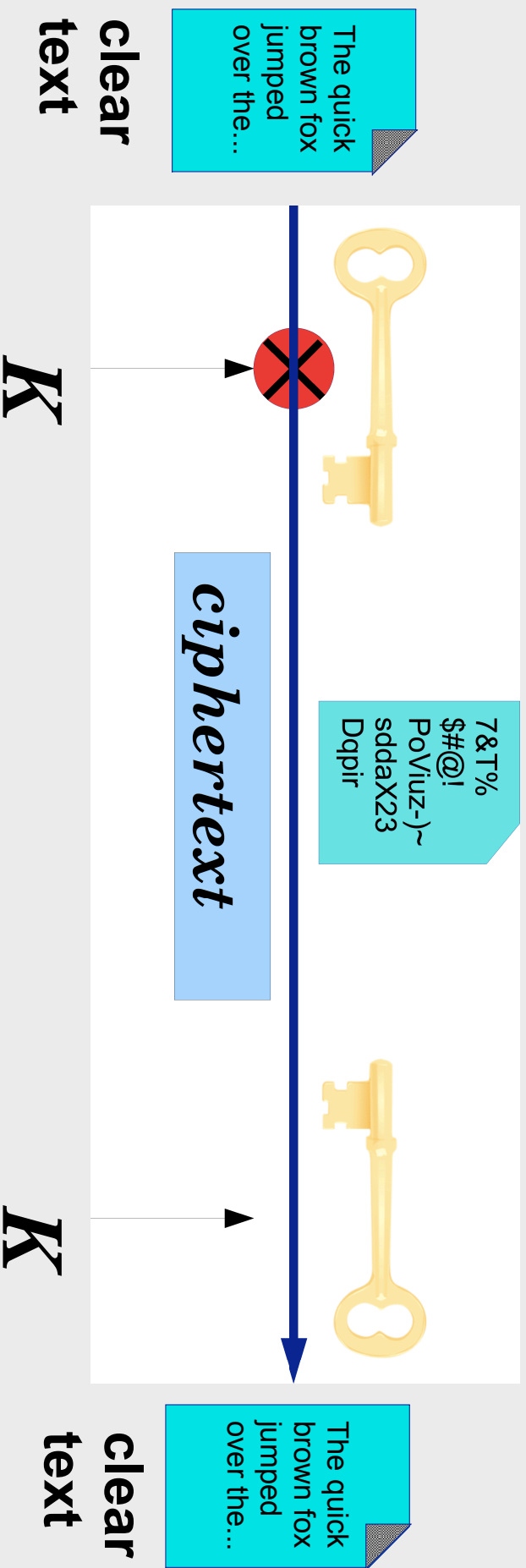
- We start with *plaintext*. Something you can read.
- We apply a mathematical algorithm to the *plaintext*.
- The algorithm is the *cipher*.
- The *plaintext* is turned in to *ciphertext*.
- Creating a secure *cipher* is a *difficult process*.
- The standardization process for AES, the replacement for the aging DES protocol, took 5 years

Keys

- In *symmetric* cryptography, a *plaintext* is transformed into a *ciphertext*, and back into *plaintext* using a key to the *cipher* (the algorithm used) on both ends.
- Assuming that the *cipher* method is known, the security of the *ciphertext* rests with the *key*. This is a *critical* point. If someone obtains your *key*, your *plaintext* is compromised.

Symmetric Cipher

Single Key/Symmetric Ciphers



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

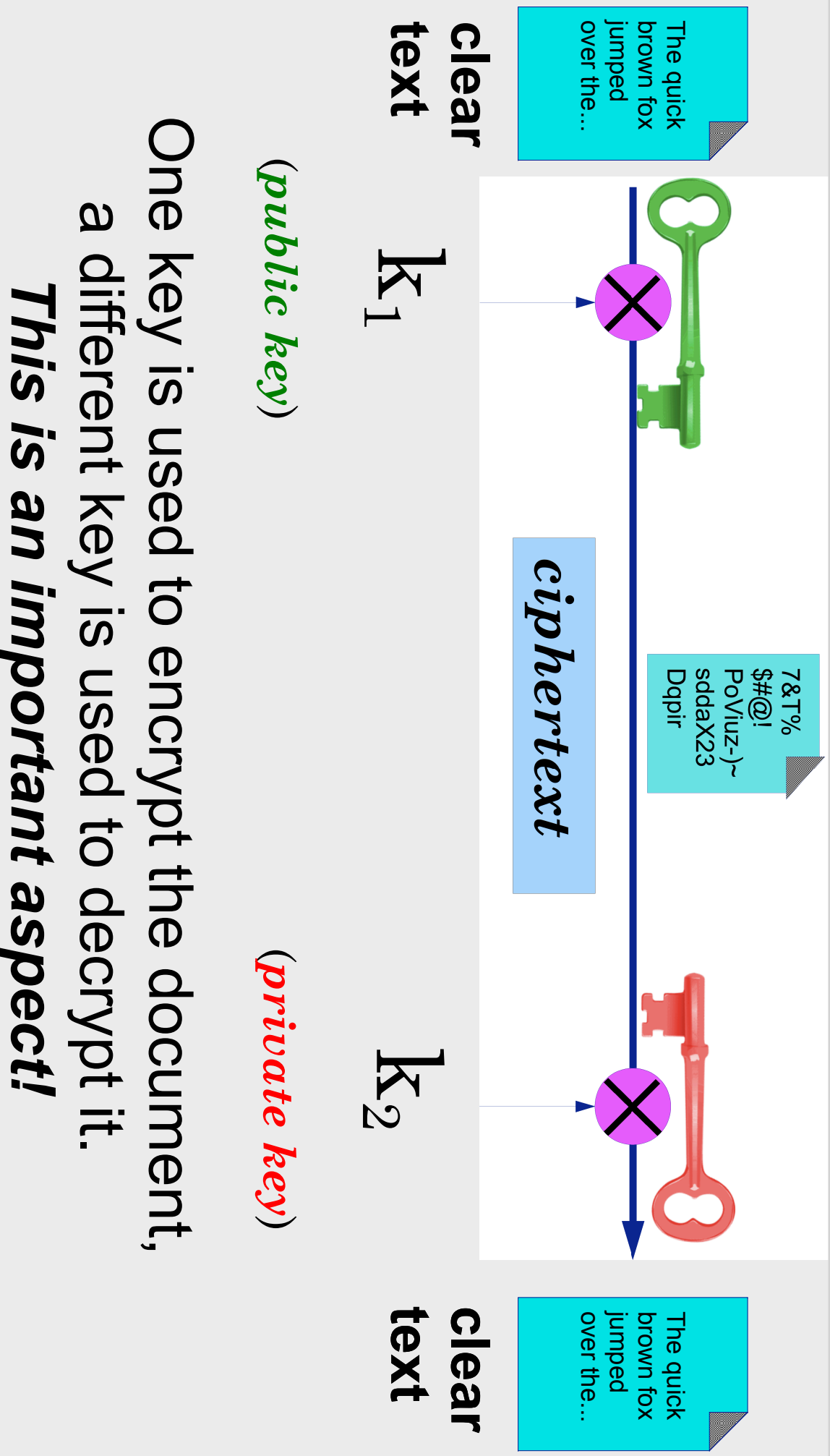
The Big Question...

 **Issue: how do you securely distribute the key to the intended receiving party or parties ?**

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) derive 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 an important aspect!

Issues

- For larger data transmissions than used in DNSSEC we use *hybrid systems*.
- *Symmetric ciphers* (single key) are *much* more efficient than *public key* algorithms for data transmission!
- Attack on the *public key* is possible via chosen-plaintext attacks. Thus, the *public/private key pair* need to be large (2048 bits).
- For instance, SSH uses *public/private* cryptography to setup the initial session, and exchange the dynamically calculated *symmetric session-key*.

Issues

- For larger data transmissions than used in DNSSEC we use *hybrid systems*.
- *Symmetric ciphers* (single key) are *much* more efficient than *public key* algorithms for data transmission!
- Attack on the *public key* is possible via chosen-plaintext attacks. Thus, the *public/private key pair* need to be large (2048 bits).
- For instance, SSH uses *public/private* cryptography to setup the initial session, and exchange the dynamically calculated *symmetric session-key*.

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, thus the term “one-way”.
- Hopefully you cannot find two sets of data that produce the same fixed-length result. If you do, this is called a *collision*. (Example, md5).
- The fixed length result is known as a *Message Digest* or a *checksum* or a *hash*.

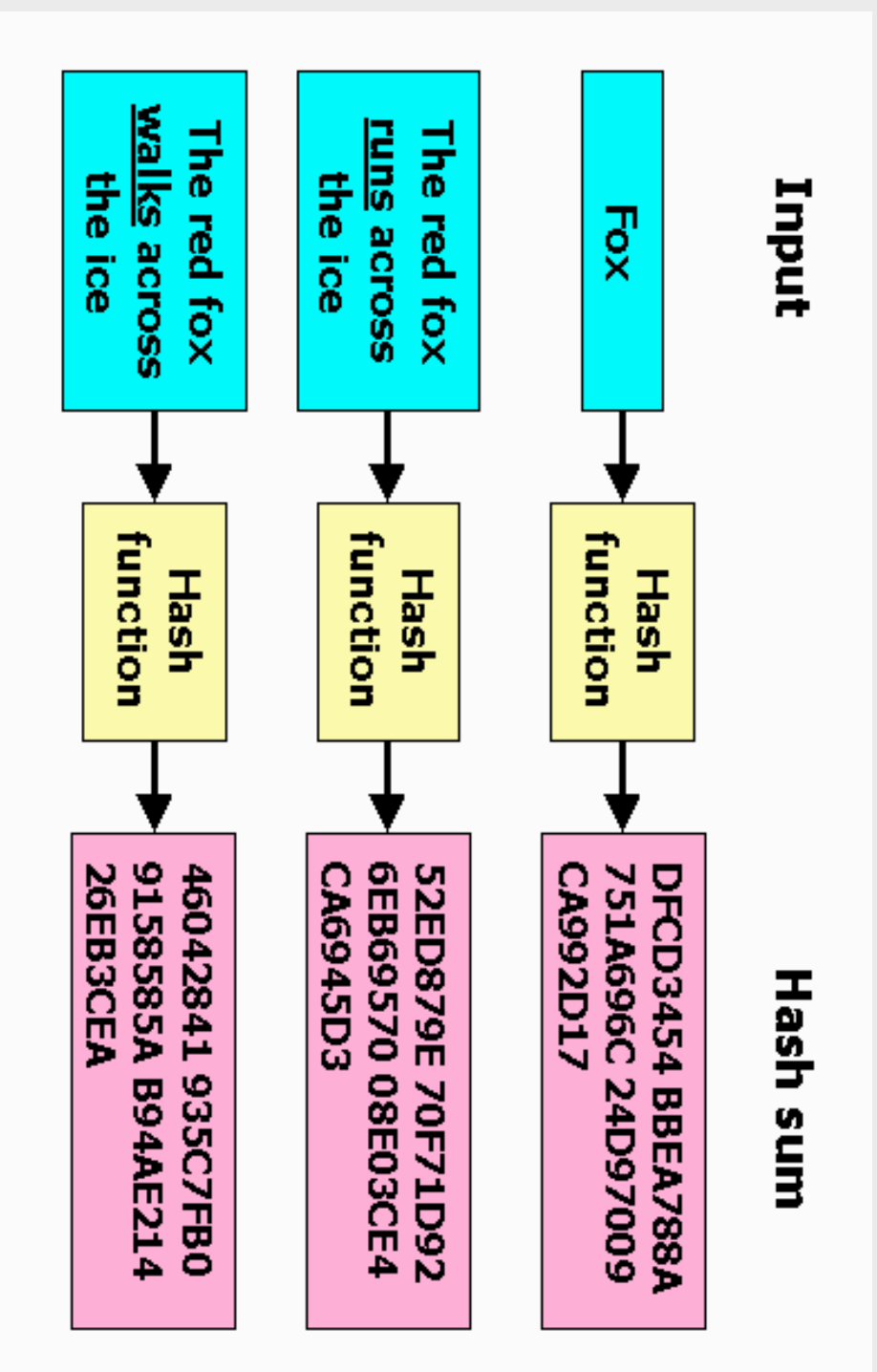
One-Way Hashing Functions cont.

- The fixed-length result of a hashing function is referred to as a *checksum*, *message digest* or *hash*.
- Some popular hashing functions include:
 - **md5**: Outputs 128 bit result. Fast. Collisions found. <http://www.mscs.dal.ca/~selinger/md5collision/>
 - **sha-1**: Outputs 160 bits. Slower. Collisions in 2^{63} .
 - **sha-2**: Outputs 224-512 bits. Slower. Collisions expected (2^{80} attack).
 - **sha-3**: TBA: Currently in development via a new *NIST Hash Function Competition*:

<http://csrc.nist.gov/groups/ST/hash/sha-3/>

Hashing

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.

What use is this?

There are several:

- Passwords encryption (in Linux, Unix and Windows), using multiple rounds of hashing (MD5 or other)
- 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). This is used to create a *digital signature*.

Digital Signatures

Reverse the role of public and private keys.

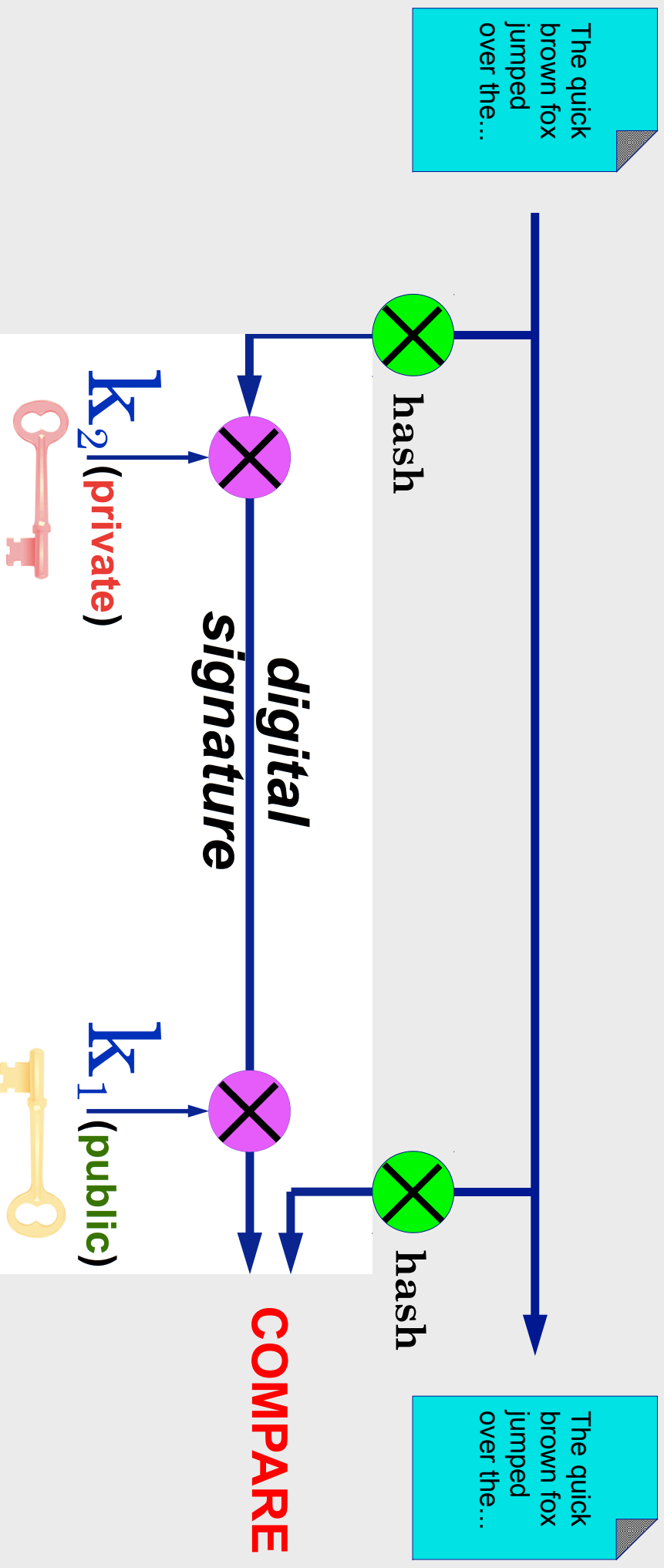
To create a digital signature on a document do:

- *Hash* a document, producing a *message digest*
- 1. Encrypt the *message digest* with your *private key*.
 - Send the document plus the encrypted *message digest*.
 - On the other end *hash* the document and decrypt the encrypted message digest with the person's *public key*.
- 1. If the results match, the document is authenticated.

This process creates a *digital signature*.

When Authenticating:

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



Conclusion

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

Are at the core of DNSSEC.