Lesson 6 – Cryptography

Hardware security review

- Hardware security protection is achieved by
  - Memory isolation
  - Segregating user mode and kernel mode instructions
    - Only secure programs (operating system) get to execute kernel mode instruction
  - Segregation of system resources into non-secured and secured
- In general, hardware security is based on preventing malicious code from accessing unauthorized resources

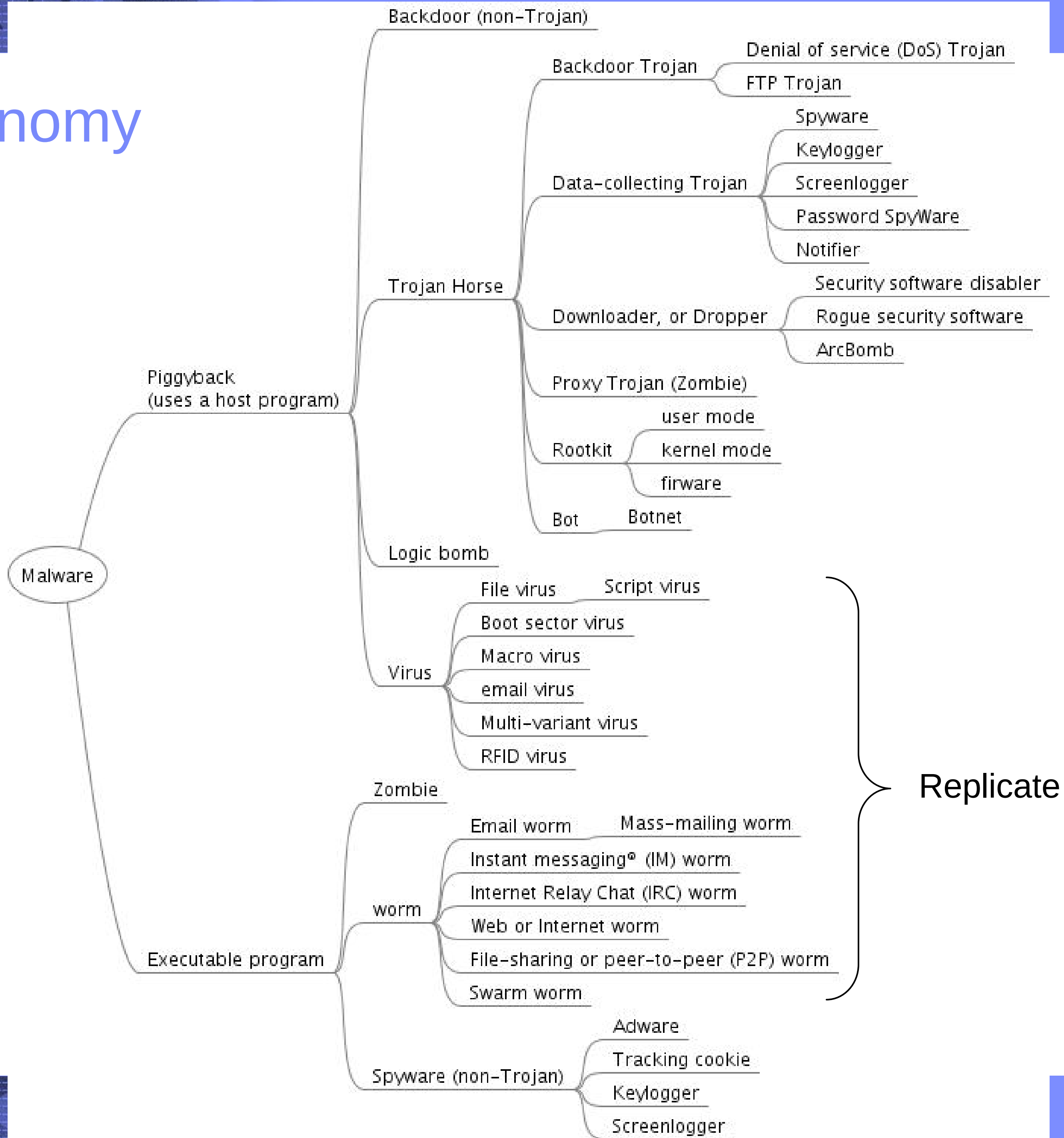
Malware characteristics

- Delivers a payload
  - How the malware affects its target
- Uses an attack vector
  - How the malware infects or spread to its targets
- May use a replicating algorithm
  - How the malware makes copies of itself

Attack Vectors

- Social engineering
  - “Make them want to run it”
- Vulnerability exploitation
  - “Force your way into the system”
- Piggybacking
  - “Make it run when other programs run”

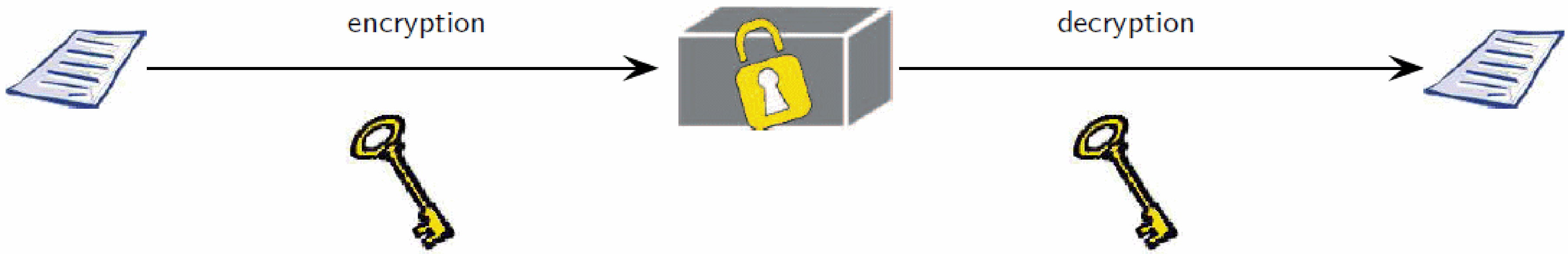
Malware taxonomy



Cryptography

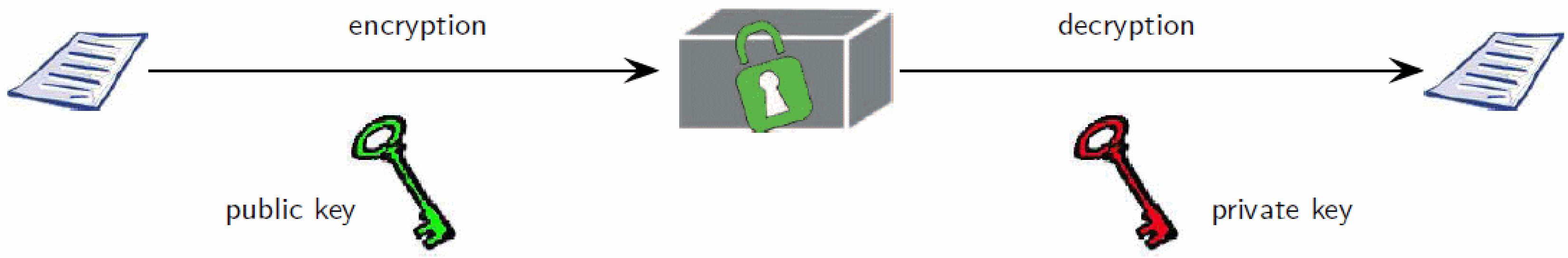
Symmetric encryption

- Same key to encrypt and decrypt
- Key is a shared secret

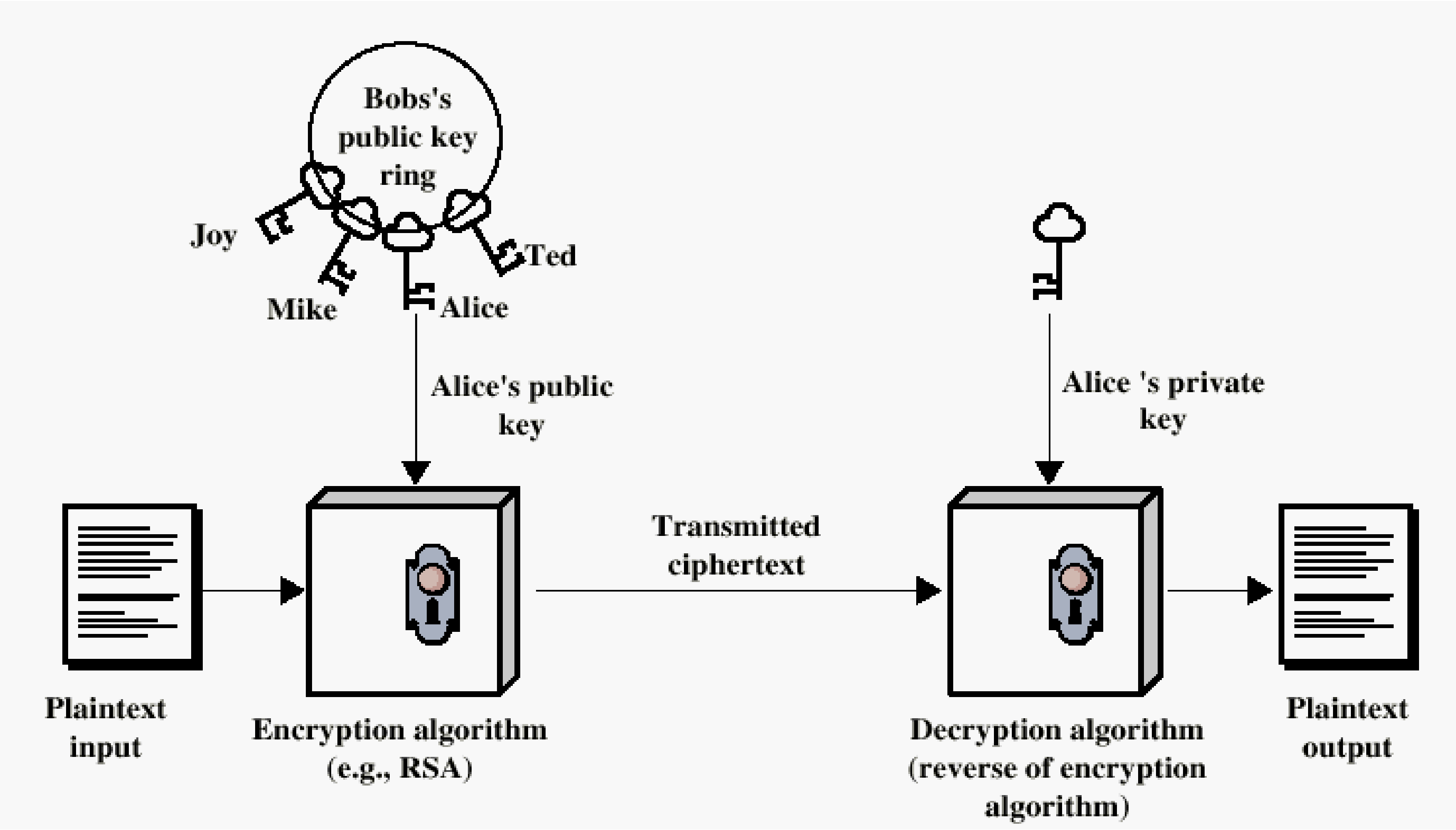


Asymmetric encryption

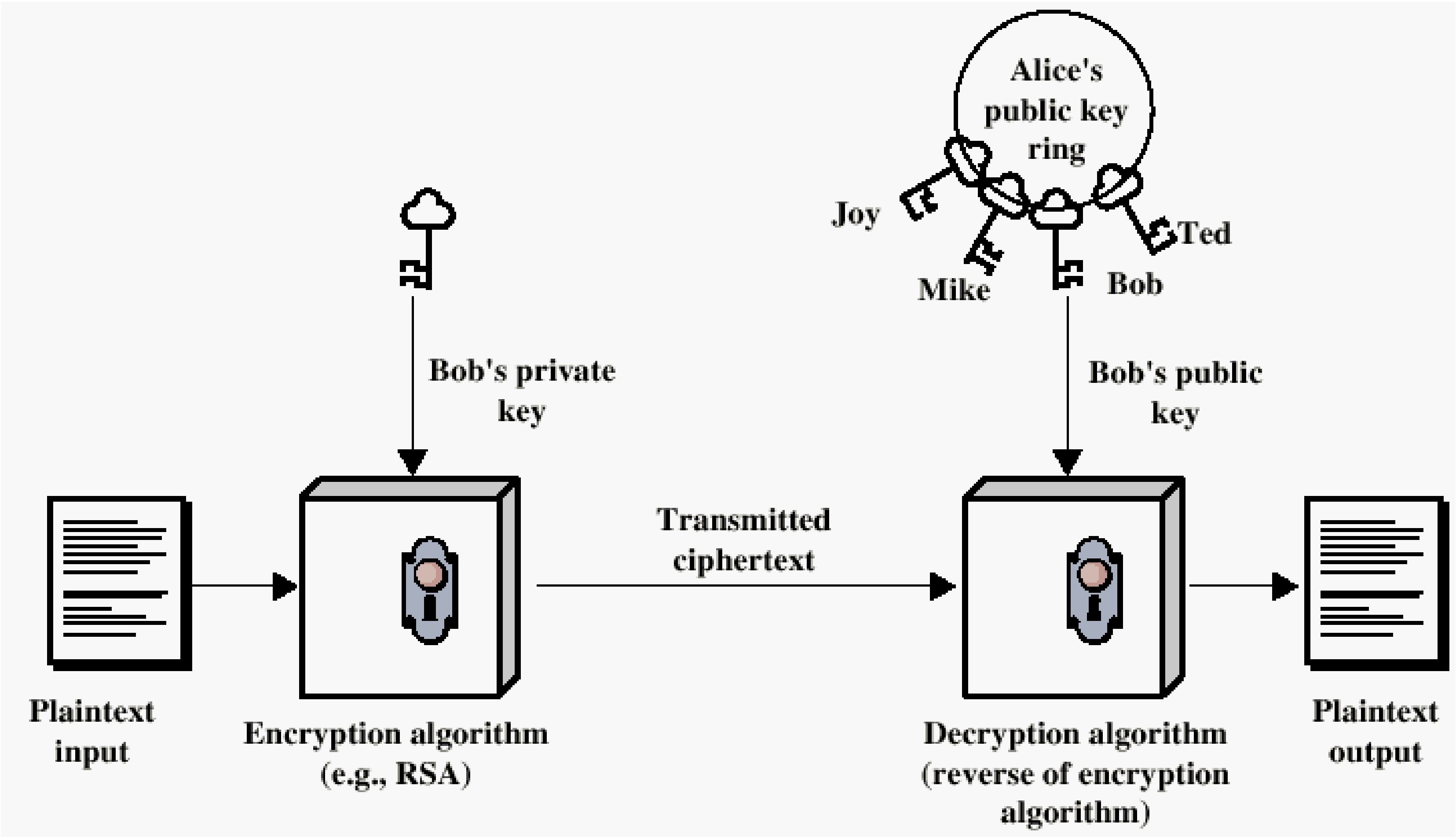
- Uses a mathematically related key pair
  - Public key to encrypt
  - Private key to decrypt
- Bob gives away the public key to all his friends so they can encrypt messages for him
- Are considered slow



Encryption using Asymmetric Key system



Authentication using Asymmetric Key System

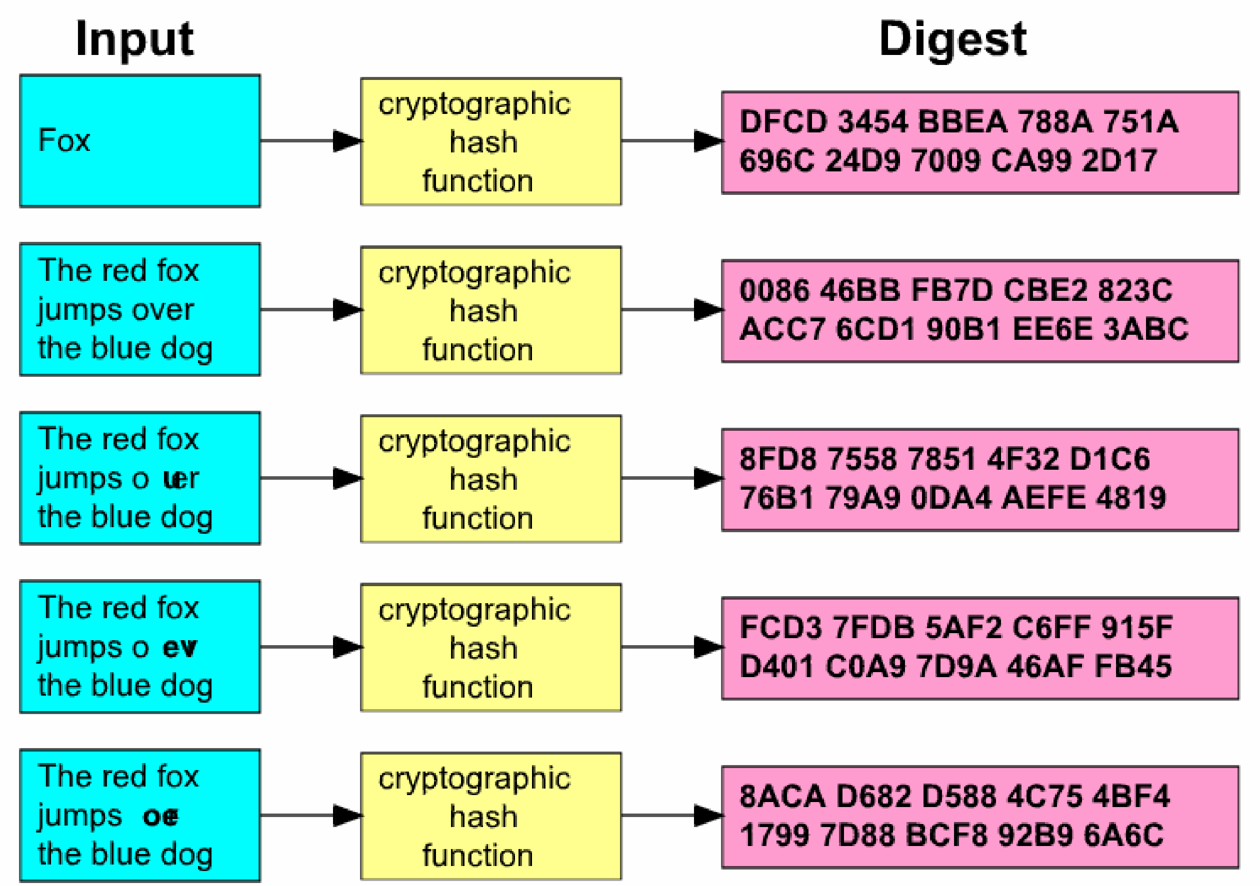


Secure HASH Functions

- Purpose of the HASH function is to produce a ”fingerprint”.
- Properties of a HASH function
  – can be applied to a block of data at any size
  – produces a fixed length digest
  – Should be easy to compute
  – Closely related text should produce different hashes

Common hash functions

Function	Digest size
RIPEMD	128
RIPEMD-160	160
MD2	128
MD4	128
MD5	**128**
SHA-0	160
SHA-1	**160**
SHA-256	256
SHA-512	512
GOST	256
Tiger	192

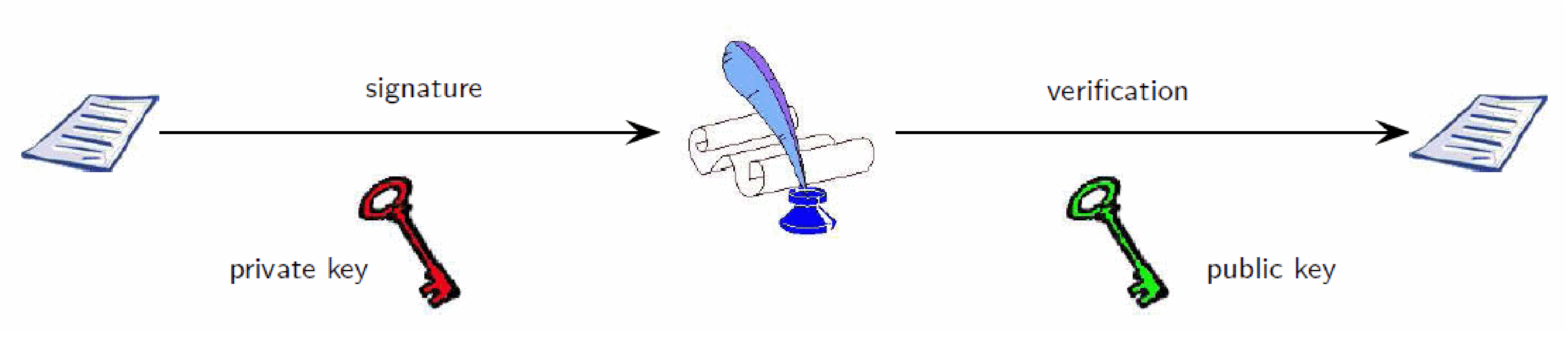


cryptographic hash function (SHA-1) at work.

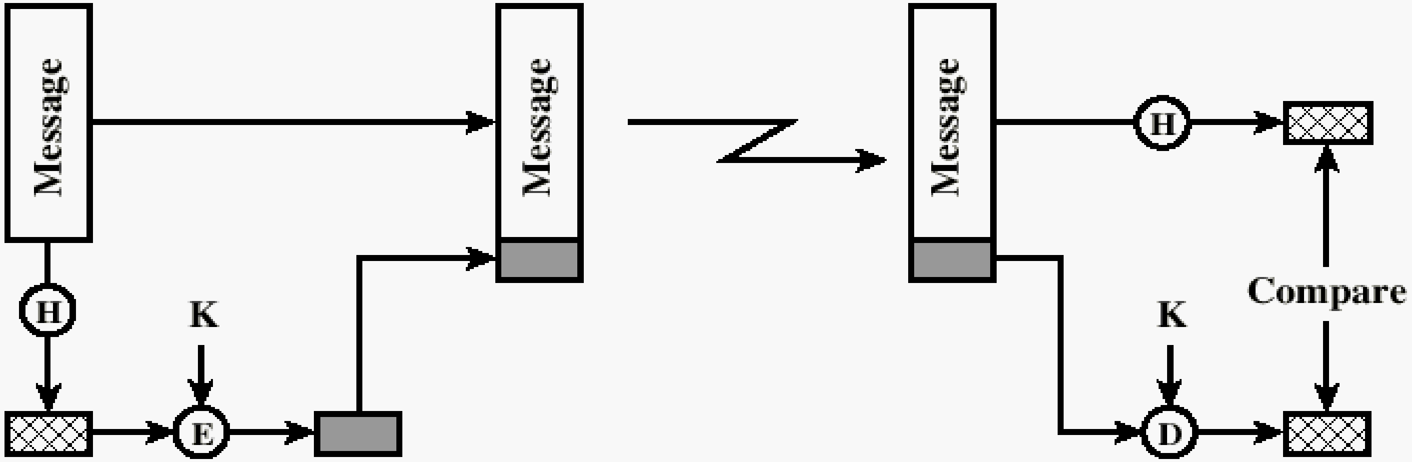
From: http://en.wikipedia.org/wiki/Cryptographic_hash_function

Signature

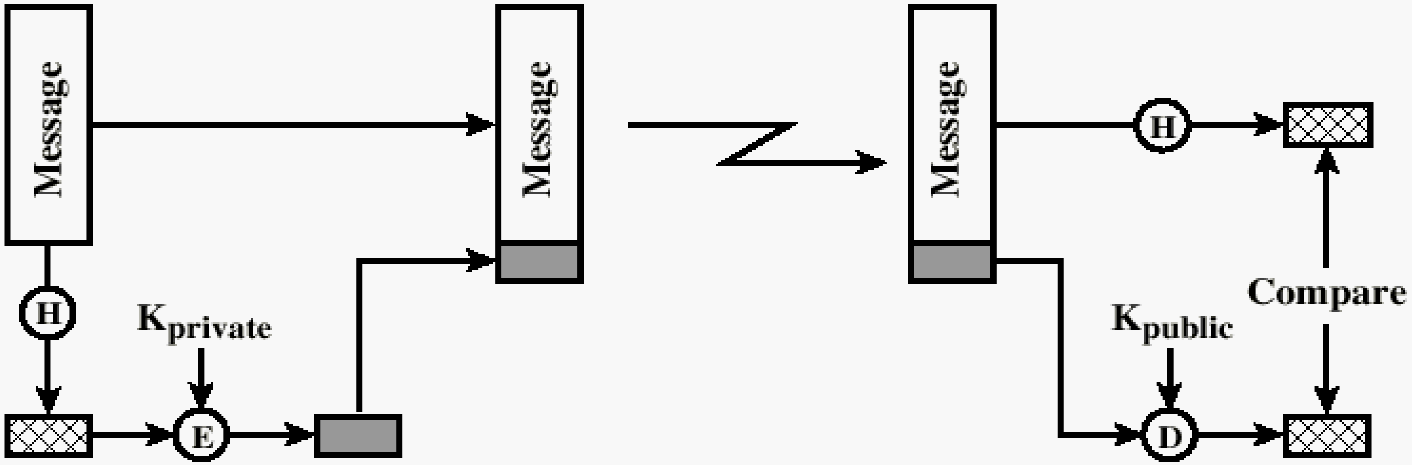
- Derived from the message content
  - May sign part of the message
- Offer
  - Guarantee the message has not been tamper with
  - Non-repudiation



Digital signature using one-way hash functions

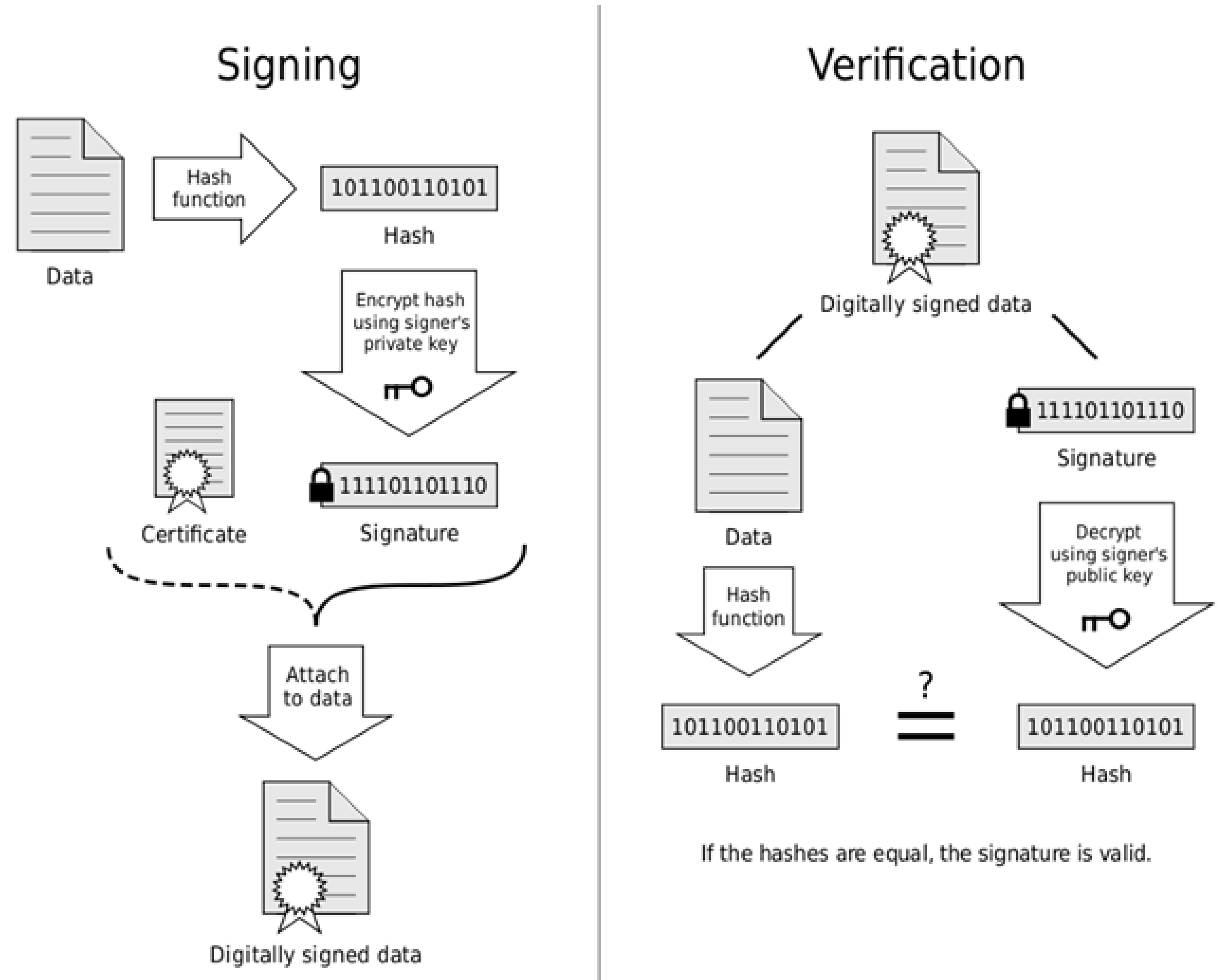


(a) Using conventional encryption



(b) Using public-key encryption

Signature



Source: http://en.wikipedia.org/wiki/File:Digital_Signature_diagram.svg

Diffie-Hellman Key Agreement

- Discovered by Whitfield Diffie and Martin Hellman
  - "New Directions in Cryptography", 1976
- Diffie-Hellman key agreement protocol
  - Allows two users to exchange a secret key
  - Requires no prior secrets
  - Real-time over an untrusted network
- Based on the difficulty of computing discrete logarithms of large numbers
- Requires two large numbers, one prime (P), and (G), a primitive root of P

Diffie-Hellman Key Agreement Protocol 1/2

Public parameters:

g, p: two large primes, $g < p$

p at least 512 bits

private parameters:

a: random number, selected by Alice

b: random number, selected by Bob

Compute public values:

$x = g^a \bmod p$, calculated by Alice

$y = g^b \bmod p$, calculated by Bob

Diffie-Hellman Key Agreement Protocol 2/2

Compute shared private key:

$k_a = y^a \bmod p$, calculated by Alice

$k_b = x^b \bmod p$, calculated by Bob

They can now communicate using symmetric keys

Because $K_a = K_b$

Alice calculated $K_A = ((g^b \bmod p)^a \bmod p)$,
result is $K_A = (g^{ab} \bmod p)$

Bob calculated $K_B = ((g^a \bmod p)^b \bmod p)$,
result is $K_B = (g^{ab} \bmod p)$

Session key $K_A = K_B = g^{ab} \bmod p$

Diffie-Hellman Key Agreement Protocol

1: B -> A: g,p // $g < p$

2: A -> B : $x=(g^a \bmod p)$ //a=Alice secret

3: B -> A : $y=(g^b \bmod p)$ //b=Bob secret

=====Now Alice & Bob can start communicating=====

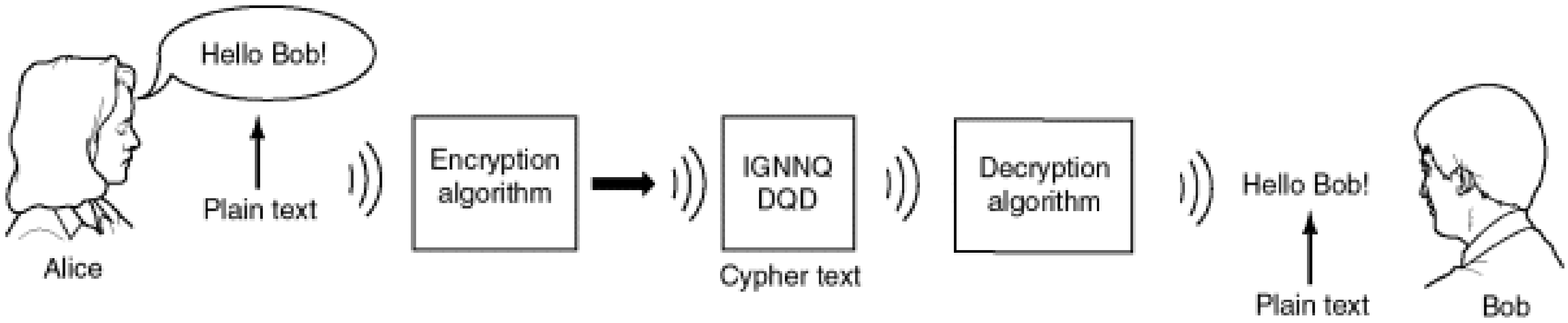
4: A -> B : $\{ M_A \}K_A$

5: B -> A : $\{ M_B \}K_B$

Session key $K_A=K_B= g^{ab} \bmod p$

Example

- Alice and Bob wish to have a secure conversation.
  - They decide to use symmetric encryption to communicate
  - and Diffie-Hellman protocol to calculate the session key



Example

- Alice and Bob interchange public numbers
  - $p = 23, \; g = 9$
- Alice and Bob select private secret
  - $a = 4$
  - $b = 3$
- Alice and Bob compute public values
  - $X = g^a \bmod p = 9^4 \bmod 23 = 6561 \bmod 23 = 6$
  - $Y = g^b \bmod p = 9^3 \bmod 23 = 729 \bmod 23 = 16$
- Alice and Bob exchange public numbers (6 & 16)

Example

- Alice and Bob compute symmetric keys
  - $k_a = y^a \bmod p = 16^4 \bmod 23 = 9$
  - $k_b = x^b \bmod p = 6^3 \bmod 23 = 9$
- Alice and Bob now can talk securely using K=9

The Computational Diffie-Hellman Assumption

- Eve, an eavesdropper
  - Knows: g, p, $x=(g^a \bmod p)$ and $y=(g^b \bmod p)$
  - But, does not know a or b
- Assumption: it is very hard to calculate $(g^{ab} \bmod p)$

Applications

- Diffie-Hellman is currently used in many protocols, namely:
  - Secure Sockets Layer (SSL / https)
  - Transport Layer Security (TLS)
  - Secure Shell (SSH)
  - Internet Protocol Security (IPSec)
  - Public Key Infrastructure (PKI)

The End