F4: public key cryptography (PKC)

tkkwon@snu.ac.kr

PKC – General Characteristics

- public-key/two-key/asymmetric cryptography
- uses 2 keys
  - public-key
    - may be known by anybody, and can be used to encrypt messages, and verify signatures
  - private-key (secret key)
    - known only to the recipient, used to decrypt messages, and sign (create) signatures
- keys are related to each other but it is not feasible to find out private key from the public one
  - Modular arithmetic

PKC – General Characteristics

- It is computationally easy to encrypt/decrypt messages when the relevant keys are known
  - RSA, ElGamal
- Trapdoor one-way function
  - k**u** (K^+ or k_p): p**u**blic-key,
  - k**r** (K^- or k_s): p**r**ivate key or secret key

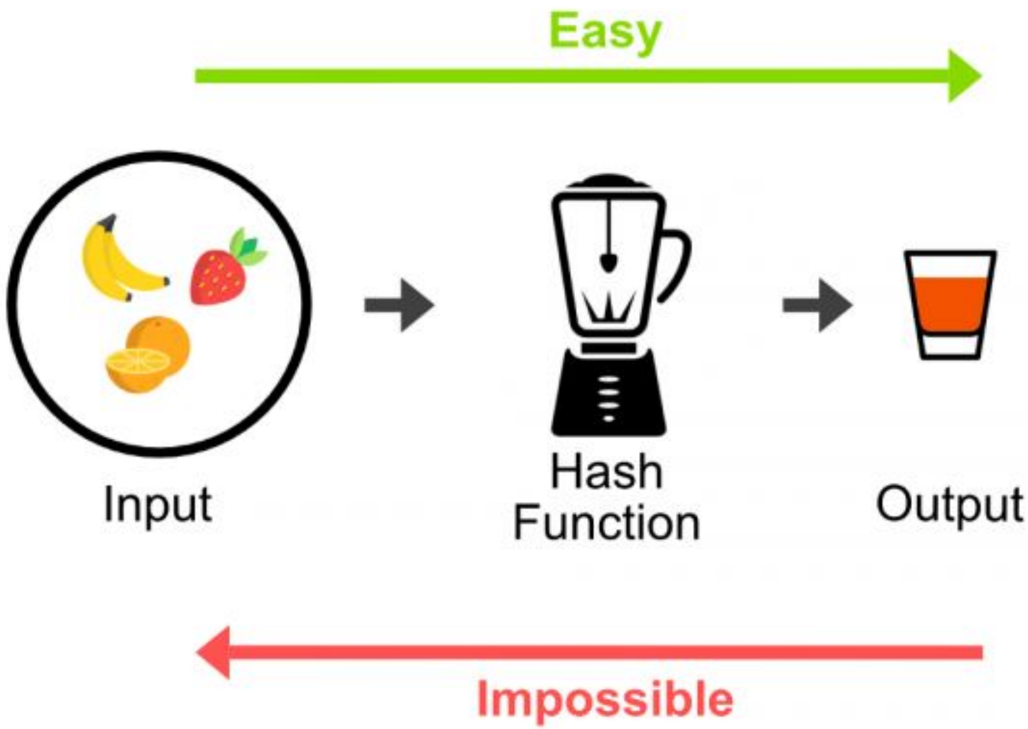
```
Y=f_ku(X) easy, if ku and X are known
X=f_kr^-1(Y)  easy, if kr and Y are known,
     but infeasible if kr is not known
```

$Y=f_{ku}(X)$ easy, if ku and X are known

$X=f_{kr}^{-1}(Y)$ easy, if kr and Y are known, but infeasible if kr is not known

a one-way function

- before discussing trapdoor one-way function, let’s talk about a one way function
- a one-way function is a function that is easy to compute on every input, but hard to invert given the image of a random input.



Source: https://computersciencewiki.org/index.php/One-way_function

Some examples of one way functions

- Cryptographic hash function:
  - Converts an arbitrary size message x into a tag of fixed length y
  - f: $x \rightarrow y$, $|y|$ = constant
- multiplying two prime numbers vs Factoring:
  - $f(p,q) \rightarrow p*q$
  - If p and q are prime it is hard to recover them from p*q
- exponentiation vs Discrete Log:
  - f: $x \rightarrow g^x \bmod p$

  where p is prime and g is a "generator" (*i.e.,* g^1, g^2, g^3, … generates all values < p)

One-way functions in PKC

- y = ciphertext x = plaintext k = <u>public</u> key

- Consider: $y = f_k(x)$ or $f(k,x)$
- Everyone knows k and f
  - computing $f(x)$ should be easy
  - $f^{-1}(y)$ should be hard
- Otherwise an eavesdropper could decrypt y
- But what about the intended recipient, who should be able to decrypt y?

Trapdoor One-Way Functions

Easy: $x \xrightarrow{f} y$

Hard: $x \xleftarrow{f^{-1}} y$

Easy: $x \xleftarrow[\text{trapdoor}]{f^{-1}} y$

- *A **one-way** function with a "trapdoor"*
- *The **trapdoor** is a private key that makes it easy to invert the function $y = f(x)$*
- Example: **RSA**

  $y = x^e \bmod n$

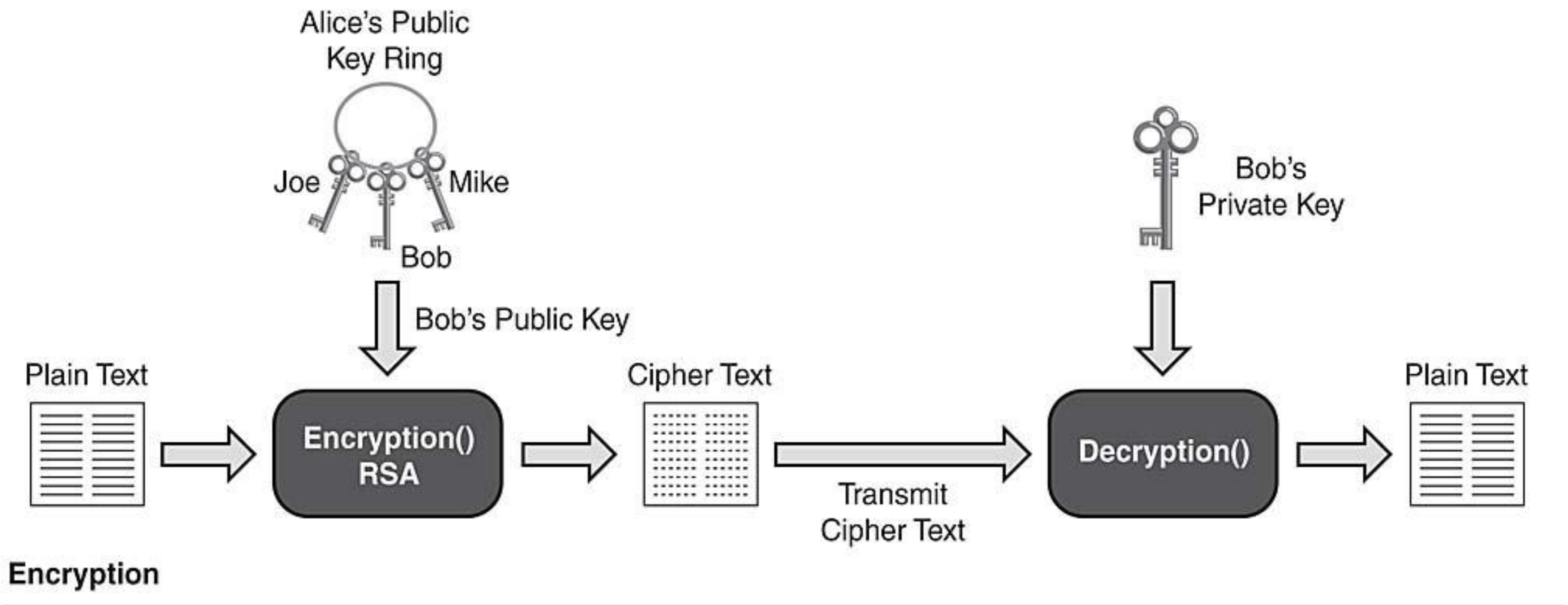
  where $n = pq$ (p, q: prime, and p, q, e: random)

  (p & q) or d (where $ed = 1 \bmod (p-1)(q-1)$) can be used as trapdoors
- In public-key algorithms

  $f(x)$ is easy with public key (*e.g.,* e and n in RSA)

  $f^{-1}(y)$ is easy only with trapdoor (*e.g.,* d in RSA)

Public-Key Cryptography: Encryption



Alice

Bob

Source: NetworkWorld

Math Expression of PKC

- Bob has a public key, k_p, and a secret key, k_s
- Bob's public key is known to Alice
- Everybody knows encryption and decryption fns.
- Asymmetric Cipher: $f^{-1}(k_s, f(k_p, m)) = m$

Alice

Bob

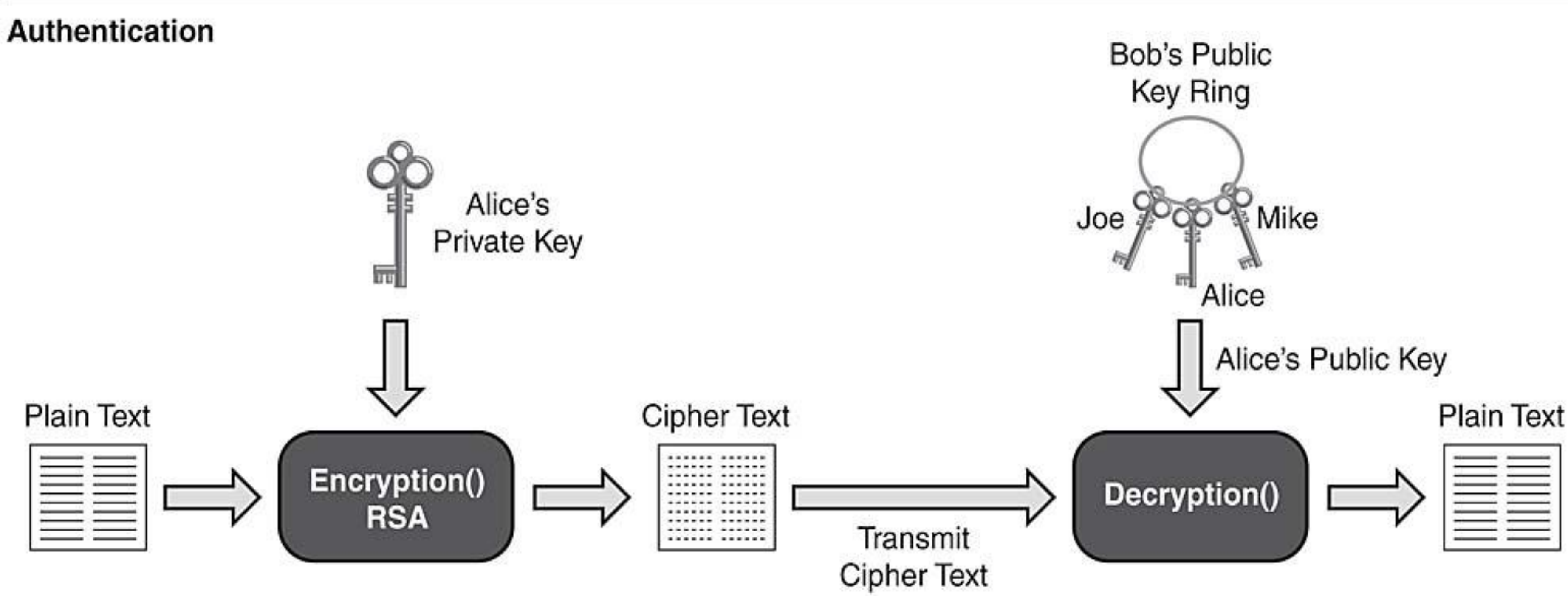
1. Construct m
2. Compute $c = f(k_p, m)$
3. Send c to Bob

$$\xrightarrow{c}$$

4. Receive c from Alice
5. Compute $m = f^{-1}(k_s, c)$
6. Deliver m

Public-Key Cryptography - Authentication

Using public and private keys is Commutative!



Alice

Bob

Source: NetworkWorld

Why PKC?

- Initially developed to address two challenging issues:
  - key distribution
    - symmetric crypto requires how to securely share the key
    - With PKC, the public key can be known to everyone
      - Public Key Infrastructure (PKI) distributes public keys
      - But we still need trusted third parties in PKI
  - digital signatures (non-repudiation)
    - not possible with symmetric crypto

PKC applications

- encryption and decryption
  - confidentiality
- digital signatures
  - authentication and non-repudiation
- key exchange
  - to agree on a session key (for symmetric cipher)
  - why not use PKC for encryption and decryption?

PKC ciphers and issues

- two PKC ciphers
  - RSA
  - ElGamal

- Performance issues
  - too slow compared to symmetric cryptography
  - when two parties communicate, it is better to derive a symmetric key by PKC
    - then the derived key is used for encryption/decryption

how to make a shared key btw. two parties

- two remote points
  - send a message?
  - ask a trusted third party?
- Diffie-Hellman (DH) algorithm
  - based on discrete logarithm

background of DH algorithm

- modulo operation
- By Fermat's little theorem: $a^{(p-1)} = 1 \pmod p$
- Example of Z_7 (actually Z_7^*)

Generators

x	x^2	x^3	x^4	x^5	x^6
1	1	1	1	1	1
2	4	1	2	4	1
<u>3</u>	**2**	**6**	**4**	**5**	**1**
4	2	1	4	2	1
<u>5</u>	**4**	**6**	**2**	**3**	**1**
6	1	6	1	6	1

DH Algorithm

- DH model's primary contribution:
  - Take a prime p and a primitive element g
    - Cyclic group in finite field
  - Publicize both g and p
  - Alice chooses some $x \in Z_p^*$ and sends (g^x mod p) to Bob
  - Bob chooses some $y \in Z_p^*$ and sends (g^y mod p) to Alice
  - Eve can see both (g^x mod p) and (g^y mod p) but she cannot calculate x or y
    - Discrete logarithm problem (DLP)

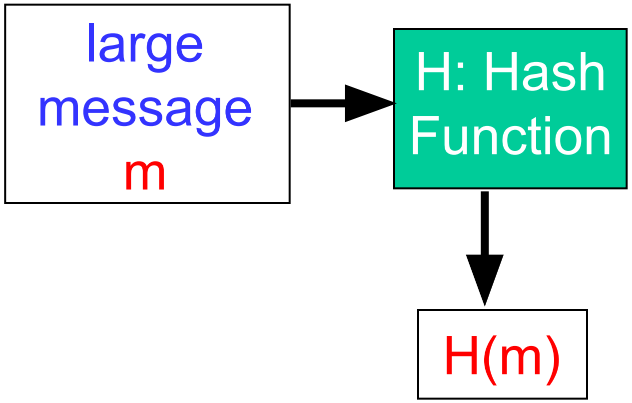
D-H Algorithm

$$\text{Alice} \quad \underset{g^y \bmod p}{\overset{g^x \bmod p}{\rightleftarrows}} \quad \text{Bob}$$

- Alice calculates the key; k = $(g^y)^x$ mod p
- Bob calculates the same key; k = $(g^x)^y$ mod p
- Since Eve does not know x or y, she cannot calculate the key k
- Diffie and Hellman developed this method to share a key using some publicly available information

MESSAGE INTEGRITY

Message Digest

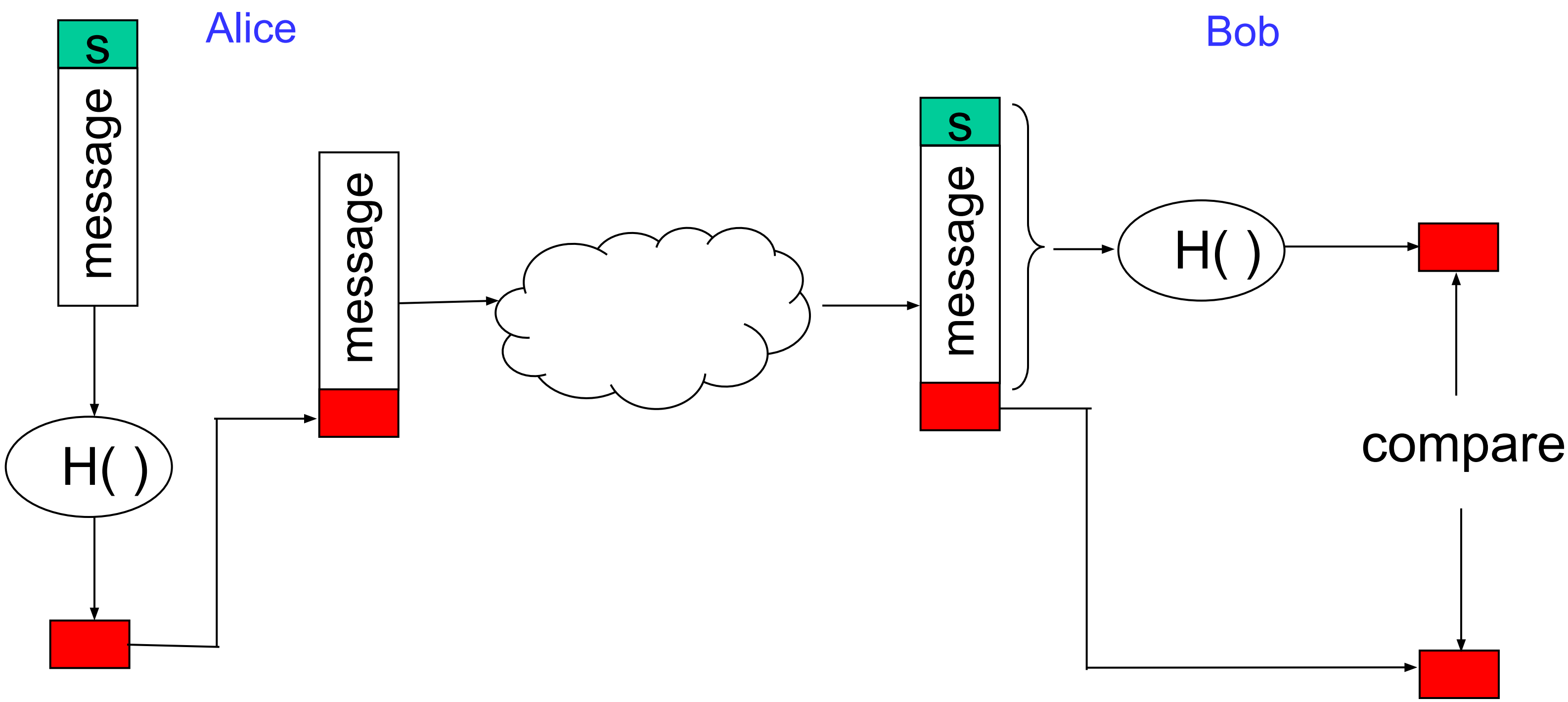


- Function H( ) that takes as input an arbitrary length message and outputs a fixed-length string: message digest, tag, fingerprint, hash
- Note that H( ) can be a many-to-1 function
- H( ) is called a "hash function"
  - MD5, SHA-1, SHA-2, SHA-3

- Desirable properties:
  - Easy to calculate
  - Irreversibility: Can't determine m from H(m)
  - Collision resistance: Computationally difficult to find m and m' such that H(m) = H(m')
  - Seemingly random output

Source: Kurose at UMass

Message Authentication Code (MAC)



- Authenticates sender
- Verifies message integrity
- No encryption!
- Also called “keyed hash”
- Notation: t = H(s||m) ; send m||t

s = shared secret

||: concatenation

Source: Kurose at UMass

MAC properties

- Symmetric
  - MACs are based on secret symmetric keys
  - The generating and verifying parties must share a secret key.
- Arbitrary message size
  - MACs accept messages of arbitrary length.
- Fixed output length
  - MACs generate fixed-size authentication tags.
- Message integrity
  - MACs provide message integrity: Any manipulations of a message in transit will be detected by the receiver since its MAC does not match with the modified message.
- Message authentication
  - The receiving party is assured of the origin of the message.
- No non-repudiation
  - Since MACs are based on symmetric principles, they do not provide non-repudiation.

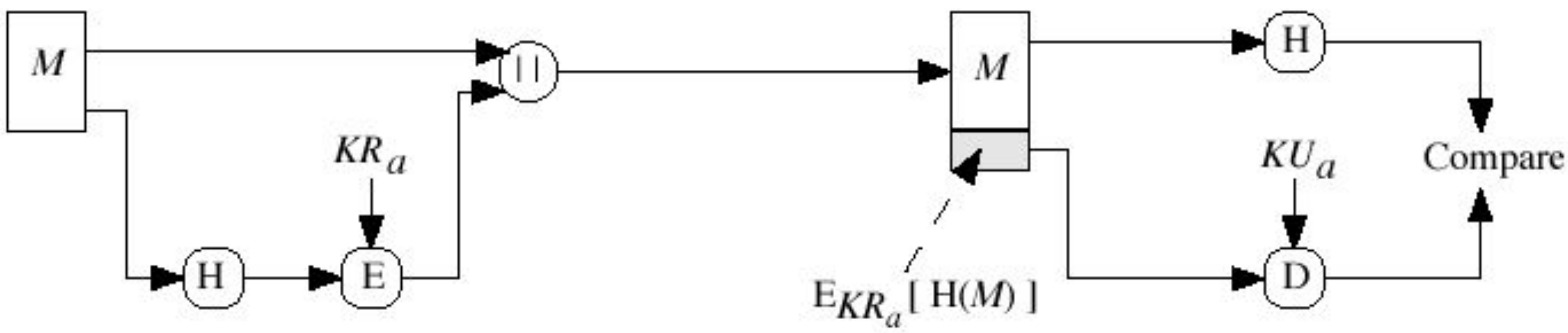
Digital Signatures

- data integrity, non-repudiation, authentication
- Basic idea: leveraging PKC
  – use your private key on the message to generate a piece of information that can be generated only by yourself
    - because you are the only one who knows your private key
  – public key can be used to verify the signature
    - so everybody can verify
- Generally signatures are created and verified over the hash of the message
  – Not over the original message. Why?

Digital Signature – PKC approach

Sender Alice

Receiver



Source: W. Stallings "Cryptography and Network Security"

$\boxed{M}$: message to be signed

$\boxed{H}$: Hash function

$\boxed{E}$: RSA Private Key Operation

KR_a: Sender's Private Key

$\boxed{D}$: RSA Public Key Operation

KU_a: Sender's Public Key

E_{KR_a}[H(M)]: Signature of sender Alice over hash of M