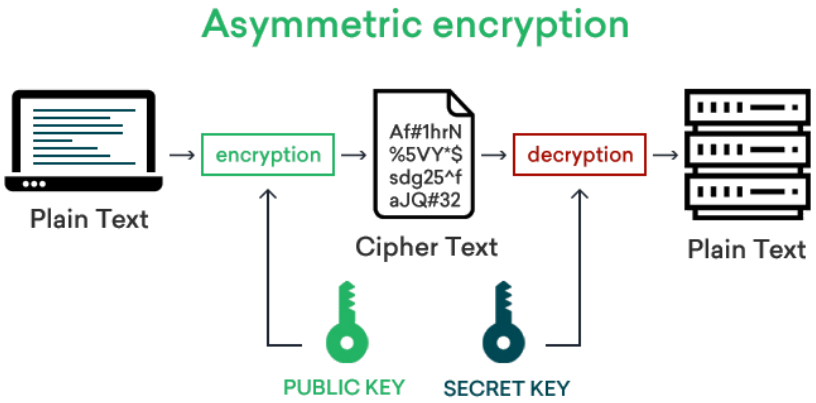


Computer Security

Public-Key Cryptography II

Asymmetric cryptography

I understood the importance in principle of public key cryptography but it's all moved much faster than I expected. I did not expect it to be a mainstay of advanced communications technology.
-Whitfield Diffie

Tamer ABUHMED
Department of Computer Science & Engineering
Sungkyunkwan University

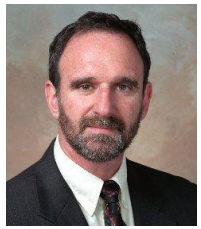
SUNG KYUN KWAN UNIVERSITY

Outline

- Diffie-Hellman Key Exchange
- Public Key Certification
- Hash Functions
  - General idea
  - Requirements and Security
  - Most common hash functions
- Digital Signatures
  - RSA signature
  - DSA signature
- Message Authentication Code
- Keyed Hash Functions
- HMAC: Idea and Security

Diffie-Hellman Key Exchange

- First PKC offered by Diffie and Hellman in 1976
- still in commercial use
- purpose is secure key-exchange
  - actually key "agreement"
  - both parties agree on a session key without releasing this key to a third party
    - to be used for further communication using symmetric crypto
- Security is in the hardness of the discrete logarithm problem
  - given $a^b \bmod n$, a and n, it is computationally infeasible to find out b if n is large enough prime number

Whitfield Diffie Martin Hellman

D-H Key Exchange

Alice **Bob**

Alice	Bob
Alice and Bob share a prime q and α, such that $\alpha < q$ and α is a primitive root of q	Alice and Bob share a prime q and α, such that $\alpha < q$ and α is a primitive root of q
Alice generates a private key X_A such that $X_A < q$	Bob generates a private key X_B such that $X_B < q$
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$	Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$
Alice receives Bob's public key Y_B in plaintext	Bob receives Alice's public key Y_A in plaintext
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$	Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Y_A : A's public key
X_A : A's private key

Y_B : B's public key
X_B : B's private key

Figure 10.1 Diffie-Hellman Key Exchange

D-H Key Exchange Example

Alice

Bob

Choose prime q = **17**, and α = **5**

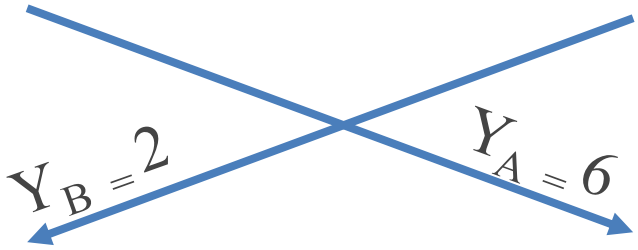
Generate $X_A = 3$

Calculate $Y_A = \alpha^{X_A} \bmod q$
$= 5^3 \bmod 17$
$= 125 \bmod 17$
$= 6$

Choose prime q = **17**, and α = **5**

Generate $X_B = 6$

Calculate $Y_B = \alpha^{X_B} \bmod q$
$= 5^6 \bmod 17$
$= 5618 \bmod 17$
$= 2$



Calculates the shared secret key

$K = Y_B^{X_A} = 2^3 \bmod 17$
$K = 8 \bmod 17 = 8$

Calculates the shared secret key

$K = Y_A^{X_B} = 6^6 \bmod 17$
$K = 46{,}656 \bmod 17 = 8$

D-H Key Exchange – PK Management

- Two issues
  - should we use global parameters (α and q) fixed for all public keys or unique?
  - do we need to make sure that a particular public key Y_i produced by *i*?
- In practice global parameters (α and q) are tied to Y values (public keys). However,
  1. both parties should use the same α and q, and
  2. there is no harm to use fixed α and q for all.
- If the D-H public values are anonymous, then a man-in-the-middle attack is possible

D-H Key Exchange – PK Management

- One PK management method
  - a closed group share common global parameters (α and q)
  - all users pick random secret values (X) and calculate corresponding public values (Y)
  - Y's are published at a trusted database
  - when B wants to create a key for A
    - B gets A's public value Y_A, and calculates the session key
    - A does the same when B sends an encrypted message to it
  - However this method is not practical for distributed applications

D-H Key Exchange – PK Management

- Anonymous public values are problematic
  - causes man-in-the-middle attacks
  - Attacker replaces the Y values with Y' values for which it knows the corresponding X' values
    - at the end A and B generate different sessions keys that are also known by the attacker
    - both A and B presume that other party has the same key, but this is not the case
  - Solution: public values and parameters should be either known or should be endorsed by a trusted entity
    - previous example of trusted database is one solution
    - public key certificates are the most common solution

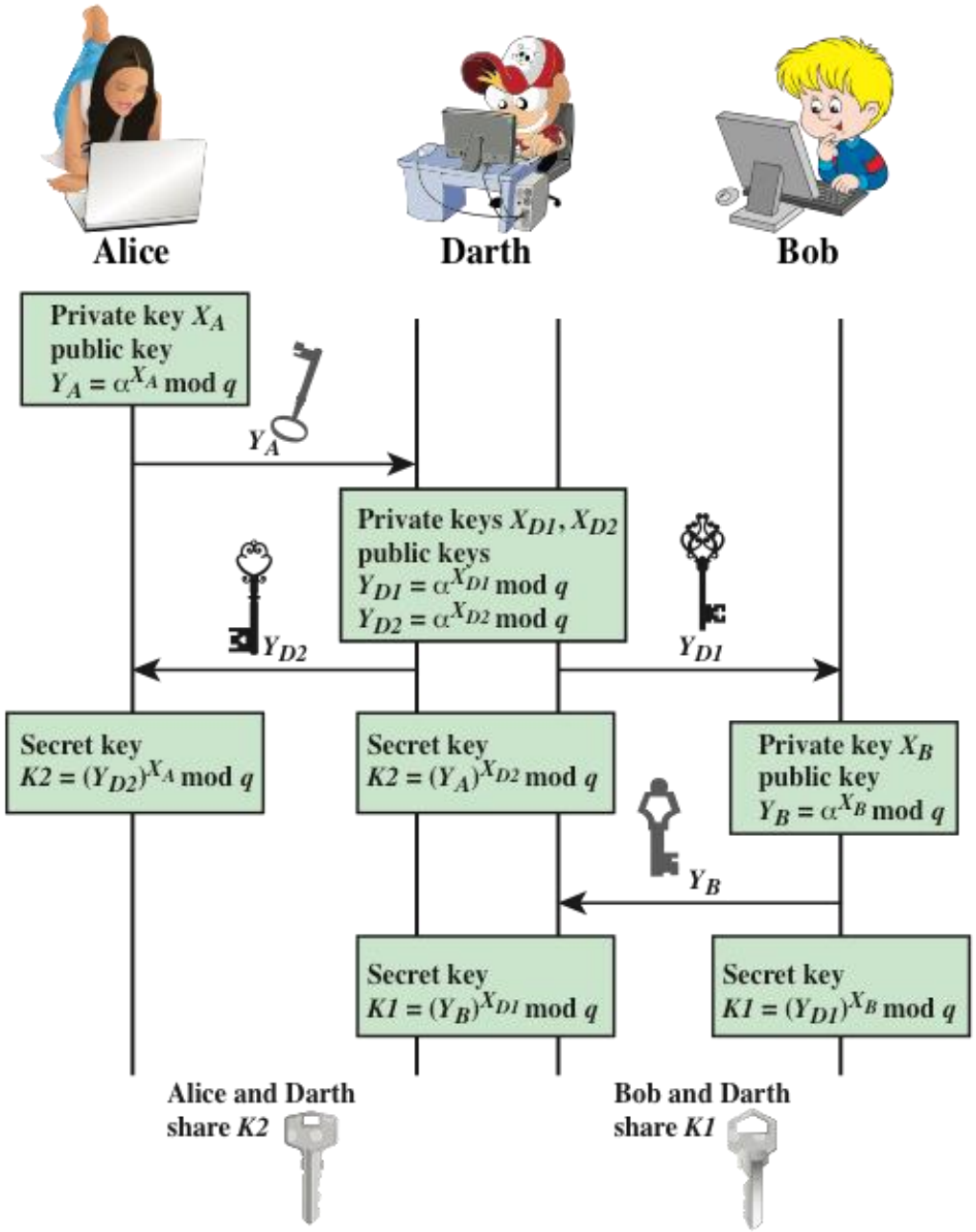


Figure 10.2 Man-in-the-Middle Attack

Public Key Certification

public key problem:

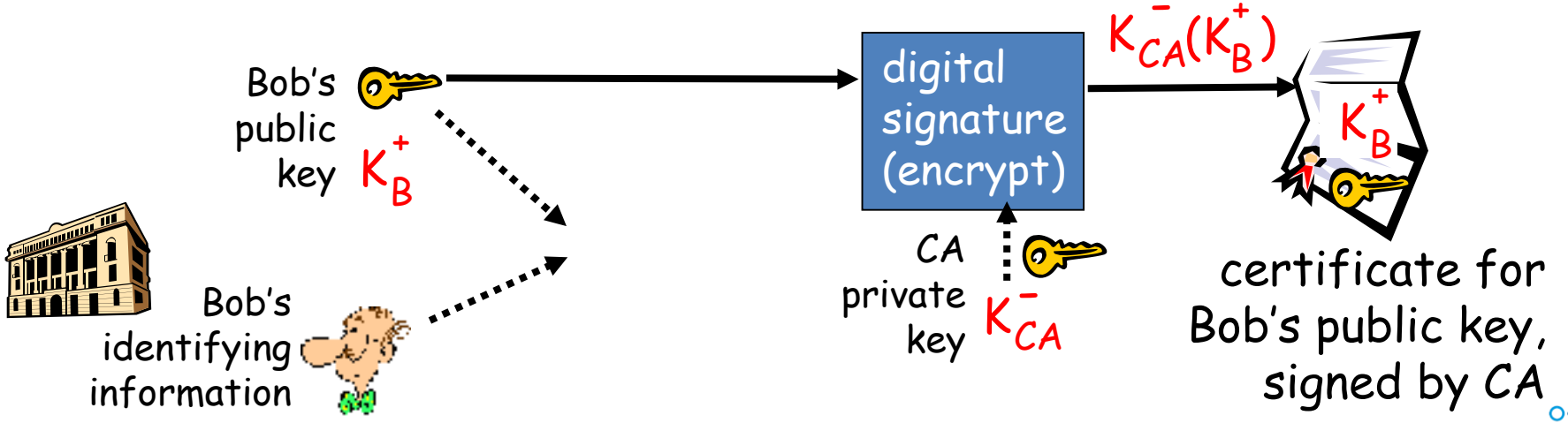
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she *know* it is Bob's public key, not Trudy's?

solution:

- trusted certification authority (CA)

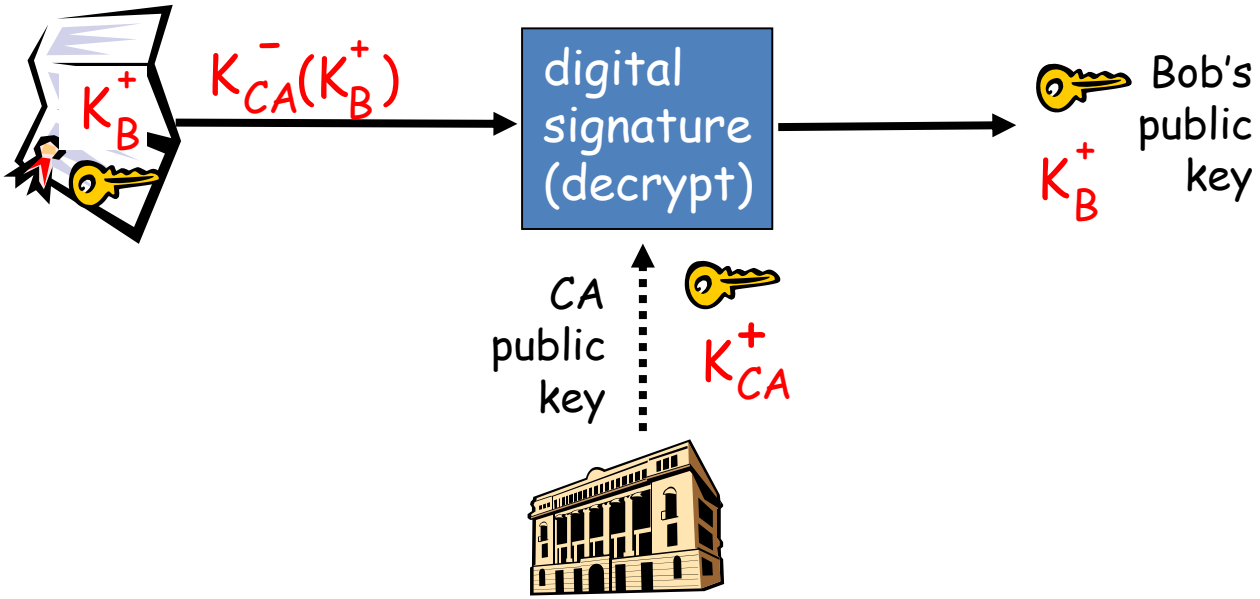
Certification Authorities

- Certification Authority (CA): binds public key to particular entity, E.
- E registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA: CA says "This is E's public key."

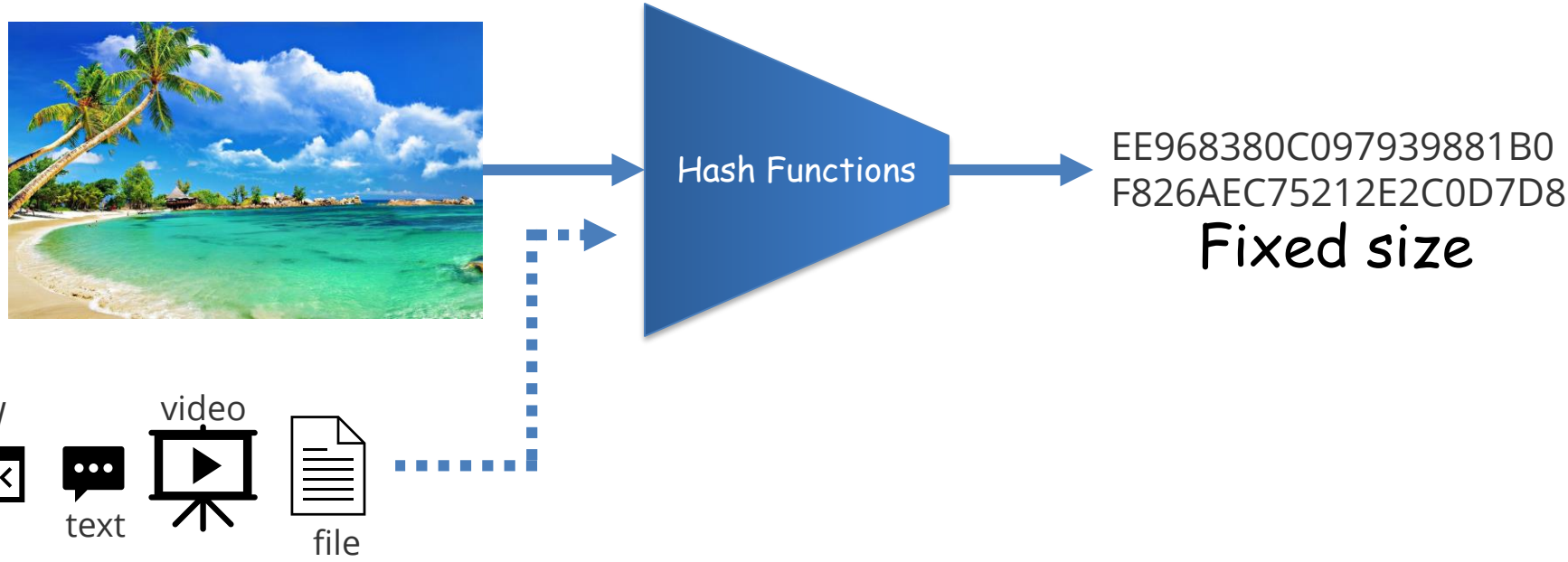


Certification Authorities

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere).
  - apply CA's public key to Bob's certificate, get Bob's public key

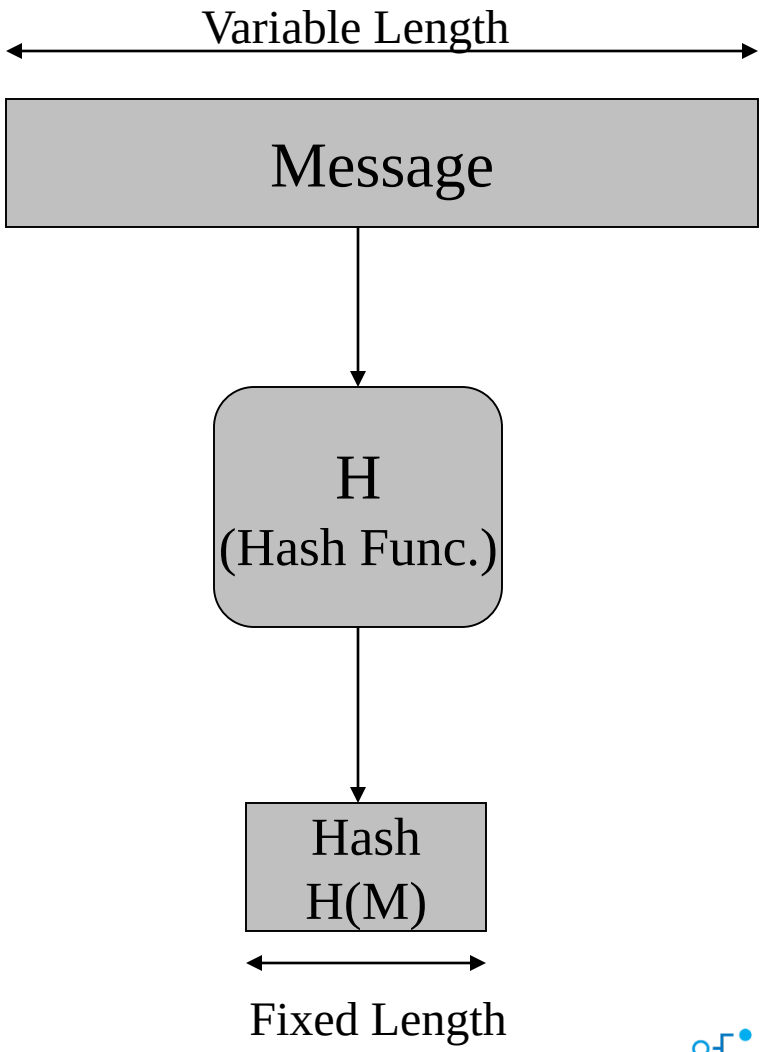


Hash Functions

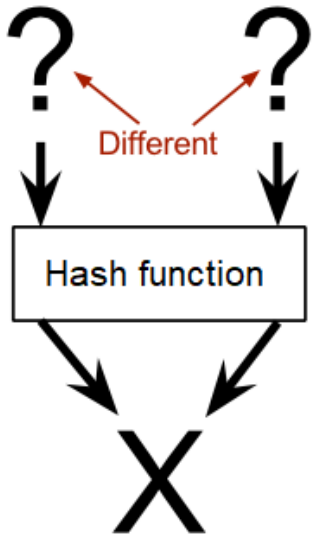


Hash Functions

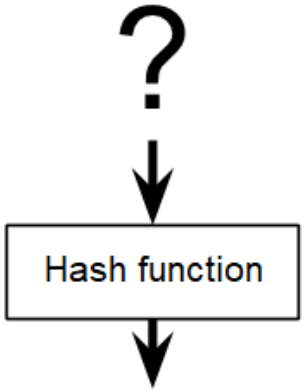
- are used to generate fixed-length fingerprints of arbitrarily large messages
- denoted as H(M)
  - M is a variable length message
  - H is the hash function
  - H(M) is of fixed length
  - H(M) calculations should be easy and fast
    - indeed they are even faster than symmetric ciphers



Hash functions – Requirements and Security

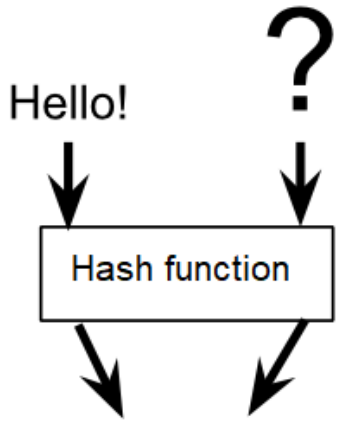


Collision resistance



334d016f755cd6dc58c53a86e1
83882f8ec14f52fb05345887c8
a5edd42c87b7

Preimage resistance



334d016f755cd6dc58c53a86e1
83882f8ec14f52fb05345887c8
a5edd42c87b7

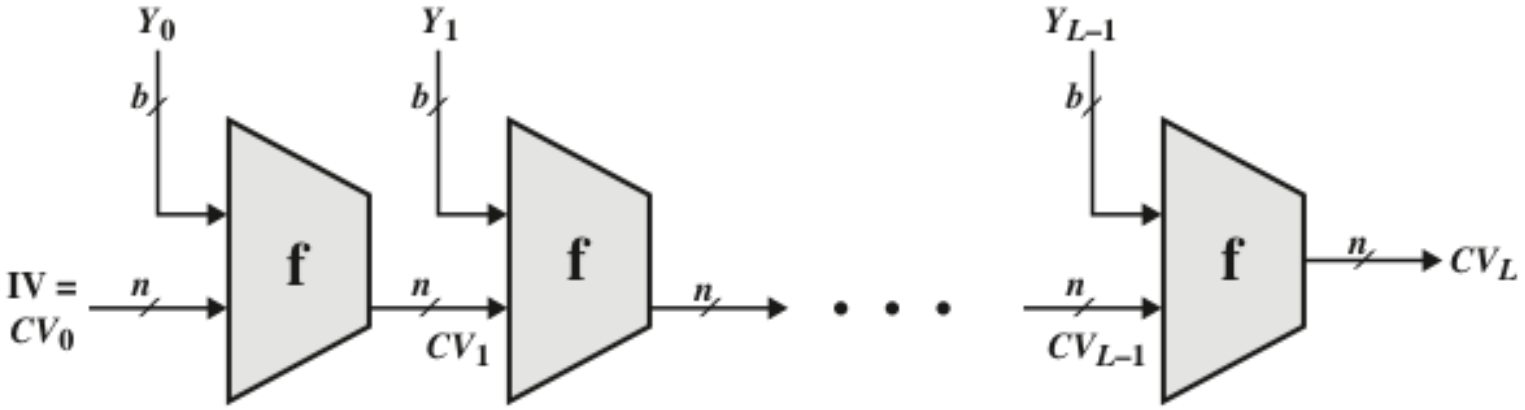
Second-preimage resistance

Hash functions – Requirements and Security

- Hash function should be a **one-way** function
  - given h, it is computationally infeasible to find x such that $h = H(x)$
  - complexity of finding x out of h is 2^n, where n is the number of bits in the hash output
  - Called one-way property (a.k.a. preimage resistance)
- **Weak collision resistance (a.k.a. second preimage resistance)**
  - given x, it is computationally infeasible to find y with $H(x) = H(y)$
  - complexity of attack is 2^n
- **(Strong) collision resistance**
  - It is computationally infeasible to find any pair x, y such that $H(x) = H(y)$
  - complexity is $2^{n/2}$

Hash function – General idea

- Iterated hash function idea by Ralph Merkle
  - a sequence of compressions
  - if the compression function is collision-free, so is the hash function
  - MD5, SHA-1, SHA-2 and some others are based on that idea



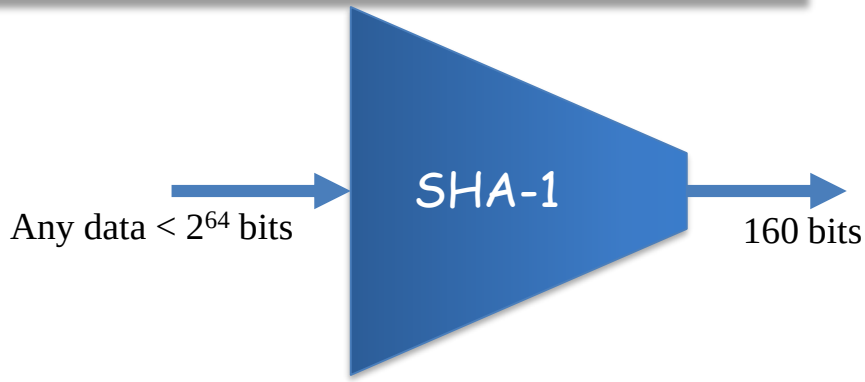
IV = Initial value
CV_i = chaining variable
Y_i = ith input block
f = compression algorithm
L = number of input blocks
n = length of hash code
b = length of input block

Important Hash Functions

- MD5
  - Message Digest 5
  - another Ron Rivest contribution
  - arbitrarily long input message
    - block size is 512 bits
  - 128-bit hash value
- has been used extensively, but its importance is diminishing
  - brute force attacks
    - 2^{64} is not considered secure complexity any more
  - cryptanalytic attacks are reported

Important Hash Functions

- SHA-1
  - Secure Hash Algorithm – 1
  - NIST standard
    - FIPS PUB 180-1
  - input size < 2^{64} bits
  - hash value size 160 bits
    - brute force attacks are not so probable
      - 2^{80} is not-a-bad complexity
  - A Crypto 2005 paper explains an attack against strong collision with 2^69 complexity
    - have raised concerns on its use in future applications
  - Later several other attacks are reported (some of them are partial attaks)
  - Eventually a practical attack is reported by the team at CWI Amsterdam and Google (approx. 2^63 complexity)
    - Paper at https://marc-stevens.nl/research/papers/SBKAM17-SHAttered.pdf
    - Link https://shattered.io/



Important Hash Functions

- However, NIST had already (in 2002) published FIPS 180-2 to standardize (SHA-2 family)
  - SHA-256, SHA-384 and SHA-512
  - for compatible security with AES
  - structure & detail is similar to SHA-1
  - but security levels are rather higher
  - 224 bit (SHA-224) is later added in 2008 as FIPS 180-3

SHA-2

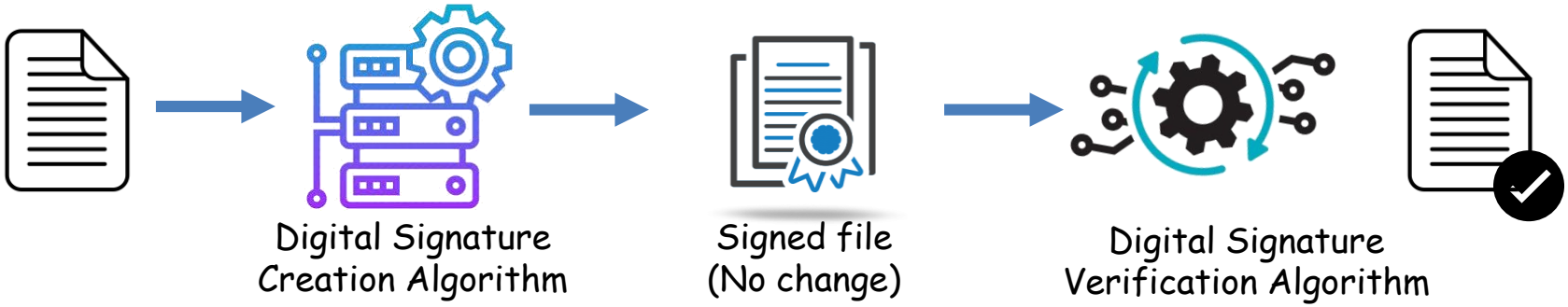
	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Message Digest Size	160	224	256	384	512
Message Size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block Size	512	512	512	1024	1024
Word Size	32	32	32	64	64
Number of Steps	80	64	64	80	80

Note: All sizes are measured in bits.

Important Hash Functions

- SHA-3
  - In 2007, NIST announced a competition for the SHA-3, next generation NIST hash function
  - Winning design was announced by NIST in October 2, 2012
  - The winner is *Keccak* by by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche
  - Different design principles than other SHAs
    - Called *Sponge* construction
  - On August 5, 2015 NIST announced that SHA-3 had become a hashing standard

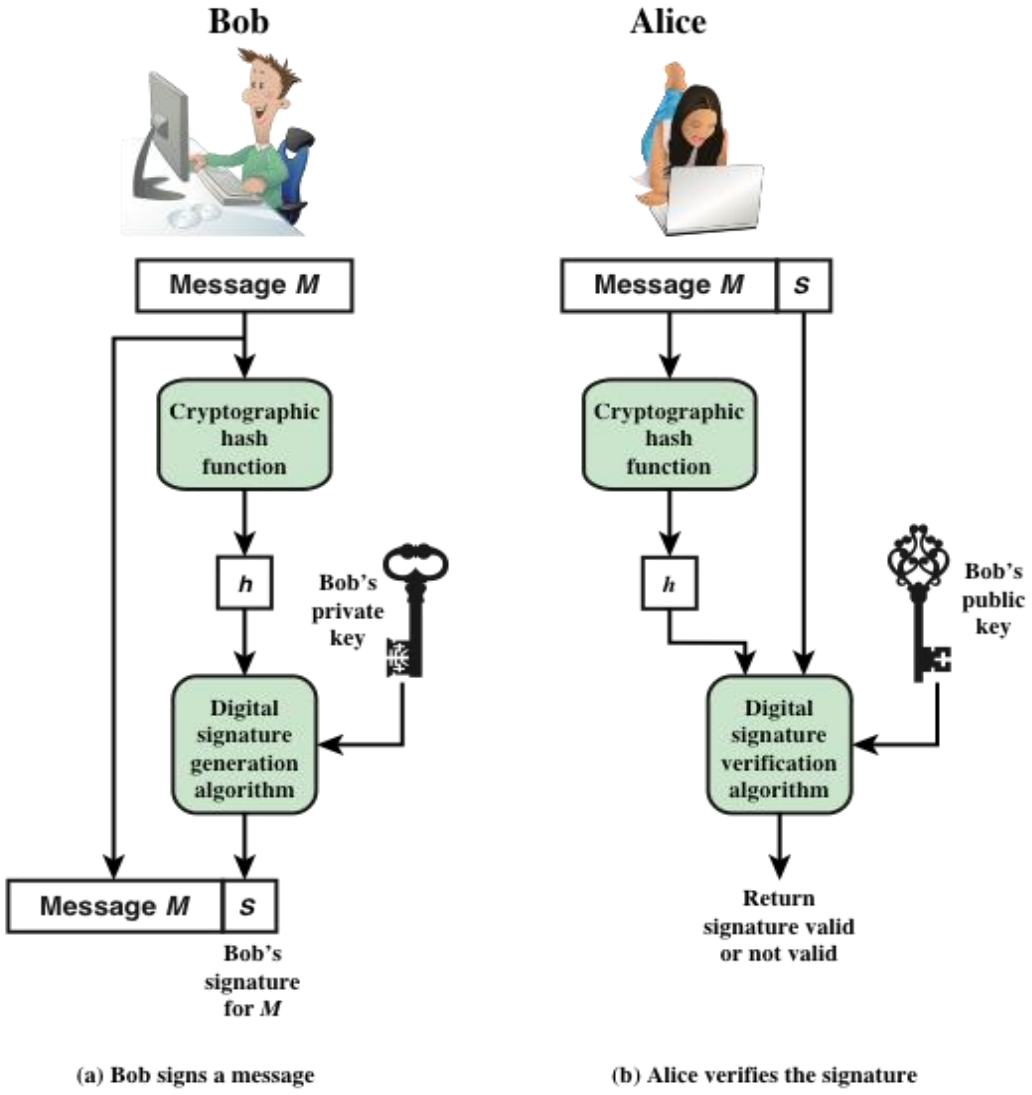
Digital Signatures



Digital Signatures

- Mechanism for non-repudiation
- Basic idea
  - use private key on the message to generate a piece of information that can be generated only by yourself
    - because you are the only person 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
  - Why?

Generic Digital Signature Model



(a) Bob signs a message

(b) Alice verifies the signature

Digital Signature – RSA approach

M: message to be signed H: Hash function

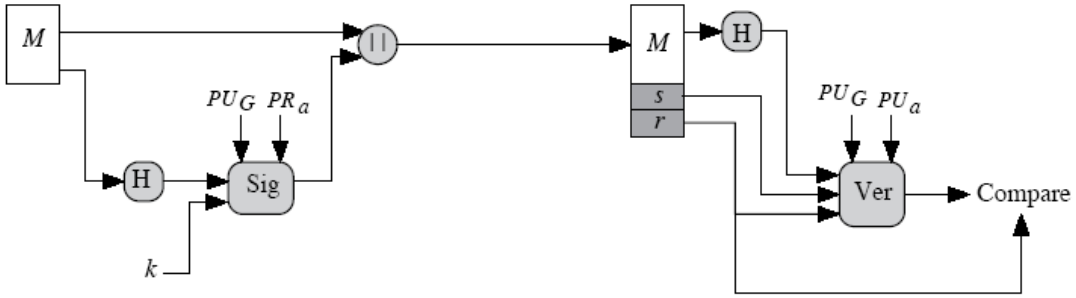
E: RSA Private Key Operation PR_a: Sender's Private Key

D: RSA Public Key Operation PU_a: Sender's Public Key

E [PR_a,H(M)] Signature of A over M

Digital Signature – DSA approach

- DSA: Digital Signature Algorithm
  - NIST standard - FIPS 186 - current revision is 186-4 (2013)
  - Key limit 512 – 1024 bits, only for signature, no encryption
    - Starting186-3, increased up to 3072
  - based on discrete logarithm problem
  - Message hash is not restored for verification (difference from RSA)



M: message to be signed H: Hash function

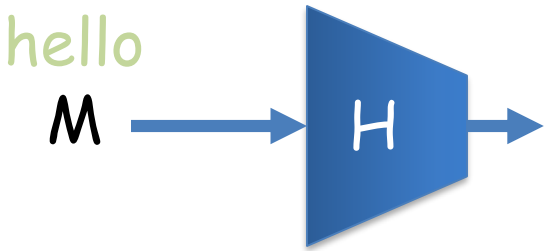
Sig: DSA Signing Operation PR_a: Sender's Private Key

Ver: DSA Verification Operation PU_a: Sender's Public Key

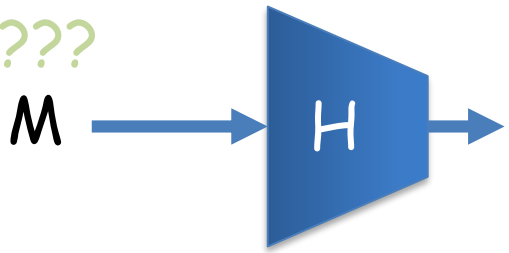
s, r Sender's signature over M PU_G: Global Public Key components

Collision resistant hash functions and digital signatures

- Have you seen the reason why hash functions should be collision resistant?
  - because otherwise messages would be changed without changing the hash value used in signature and verification



AAF4C61D
DCC5E8A2
DABEDE0F
3B482CD9
AEA9434D



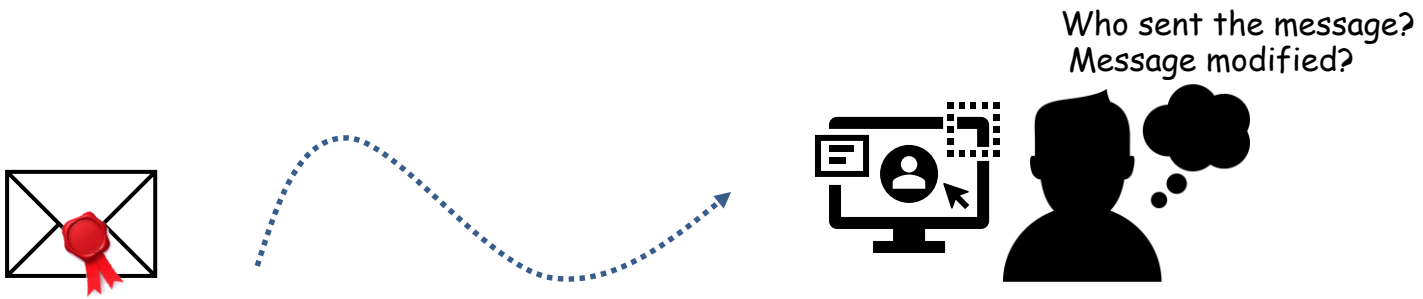
AAF4C61D
DCC5E8A2
DABEDE0F
3B482CD9
AEA9434D

Collision resistant hash functions and digital signatures

- Birthday attack
  - generate two messages
    - one with legitimate meaning
    - one fraudulent
  - create a set of messages from each of them that carries the same meaning
    - play with blanks, synonyms, punctuations
  - calculate the hashes of those two sets
  - you should have $2^{n/2}$ messages (and hashes) in each set for 0.63 probability of a match, where n is the hash size
  - if a match is found, then the fraudulent hash could be replaced with the legitimate one without affecting the signature

Message Authentication

Using asymmetric or symmetric cryptography



Message Authentication

- Making sure of
  - message has been sent by the alleged sender
  - message has been received intact
    - no modification
    - no insertion
    - no deletion
  - i.e., Message Authentication also covers integrity
- Digital Signatures
  - provides integrity + authentication + nonrepudiation
- We will see mechanisms that provide authentication, but not non-repudiation

Mechanisms for Message Authentication

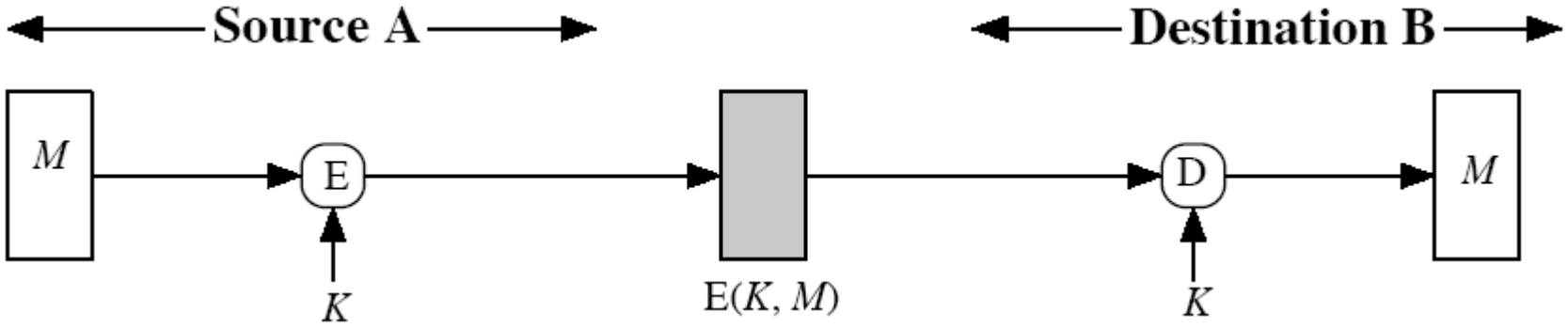
- General idea
  - receiver makes sure that the sender knows a secret shared between them
  - in other words, sender demonstrates knowledge of that shared secret
  - without revealing the shared secret to unauthorized parties of course
- We will see some mechanisms for this purpose

Mechanisms for Message Authentication

1) Message Encryption
   - provides message authentication, but …
2) Message Authentication Code Functions
   - similar to encryption functions, but not necessarily reversible
   - Generally Hash based MAC is used (will see)
3) Actually hash functions are used for message authentication in several ways (will see)

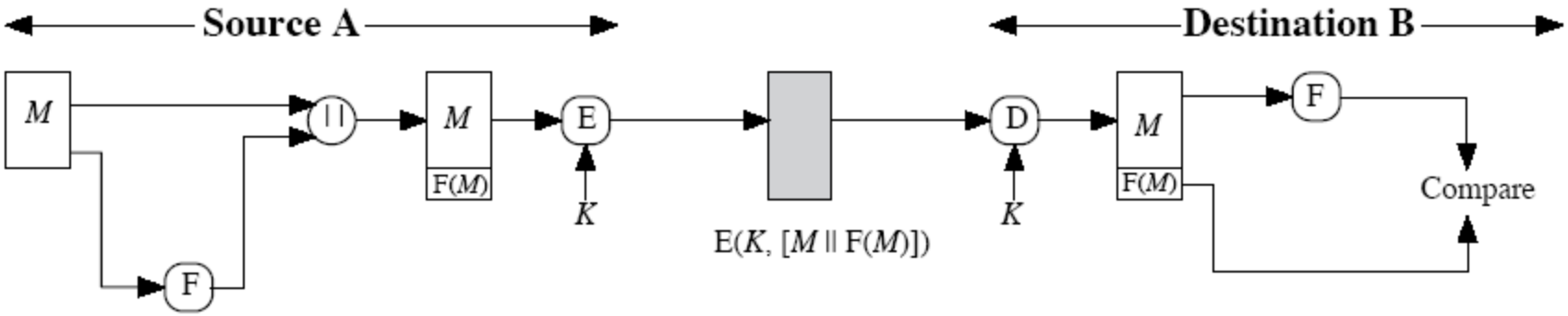
Using Message Encryption for Authentication

- Provides encryption. What about authentication?



Using Message Encryption for Authentication

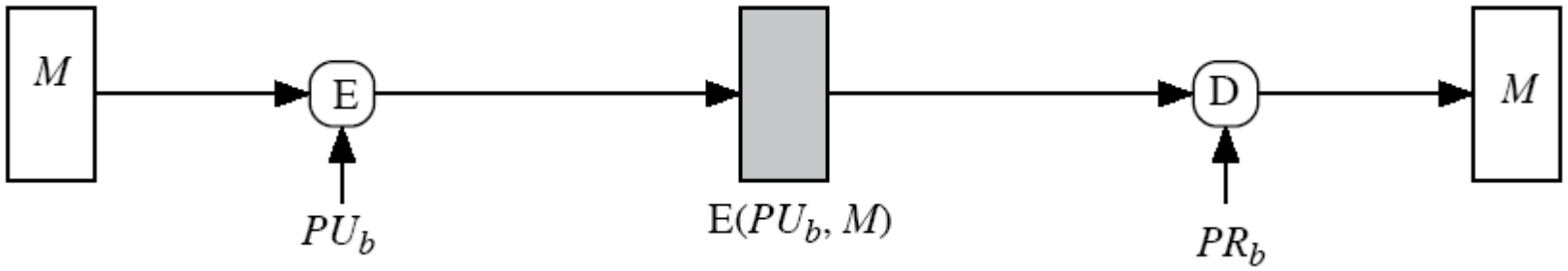
- Addition of FCS (frame check sequence) helps to detect if both M's are the same or not



F: FCS function

Using Message Encryption for Authentication

- What about public-key encryption?



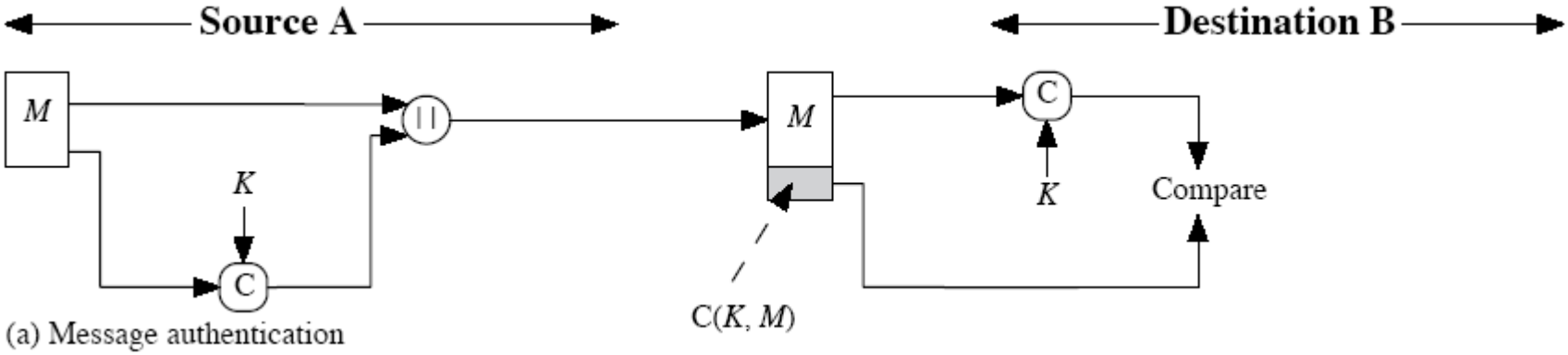
- Provides confidentiality, but not authentication
  - Why?
  - What should be done for authentication using public-key crypto?
  - we have seen the answer before.

Message Authentication Code (MAC)

- An alternative technique that uses a secret key to generate a small fixed-size block of data
  - based on the message
  - not necessarily reversible
  - secret key is shared between sender and receiver
  - called *cryptographic checksum* or *MAC* (*message authentication code*)
- appended to message
- receiver performs same computation on message and checks if matches the received MAC
- provides assurance that message is unaltered and comes from sender

MAC

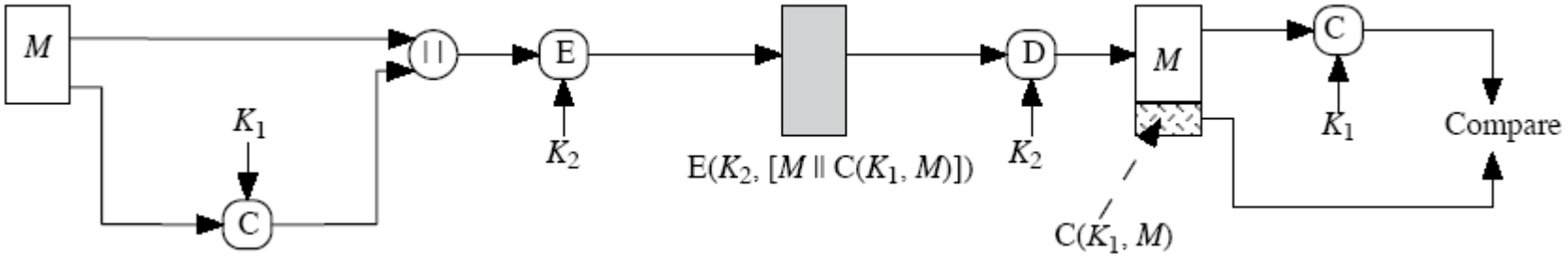
- Only authentication



(a) Message authentication

C: MAC function

- Authentication and confidentiality



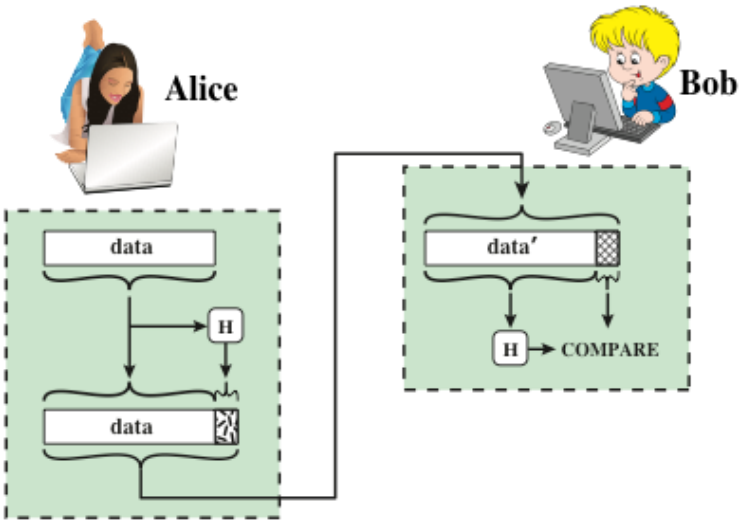
MAC – The Basic Question

- Is MAC a signature?
  - No, because the receiver can also generate it

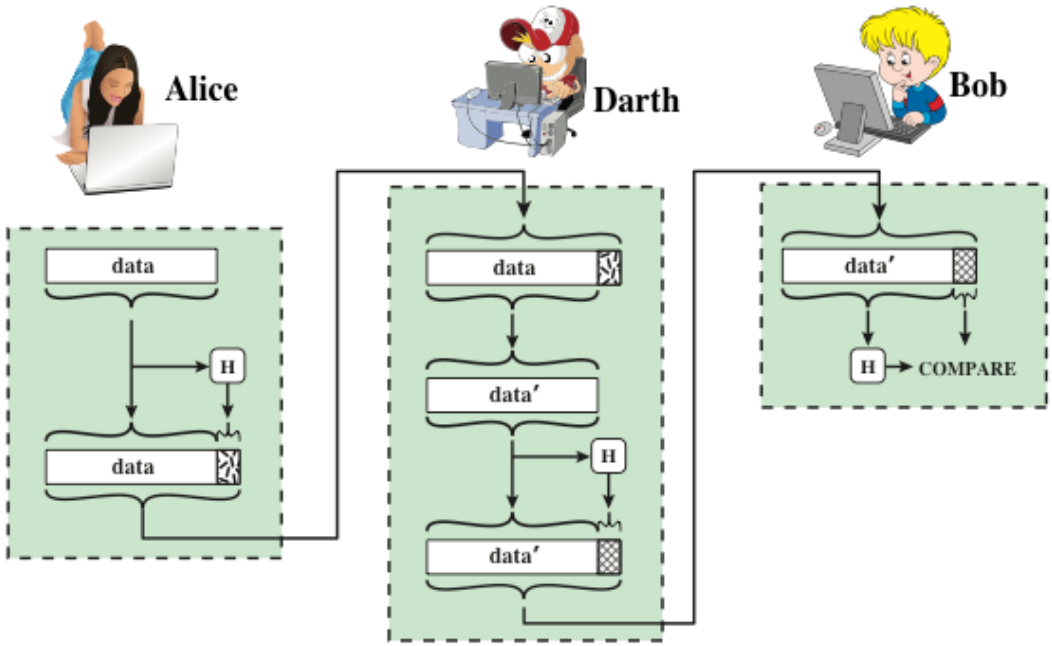
Hash based Message Authentication

- Hash Functions
  - condenses arbitrary messages into fixed size
- We can use hash functions in authentication and digital signatures
  - with or without confidentiality

Can we just use hash function for integrity?



(a) Use of hash function to check data integrity

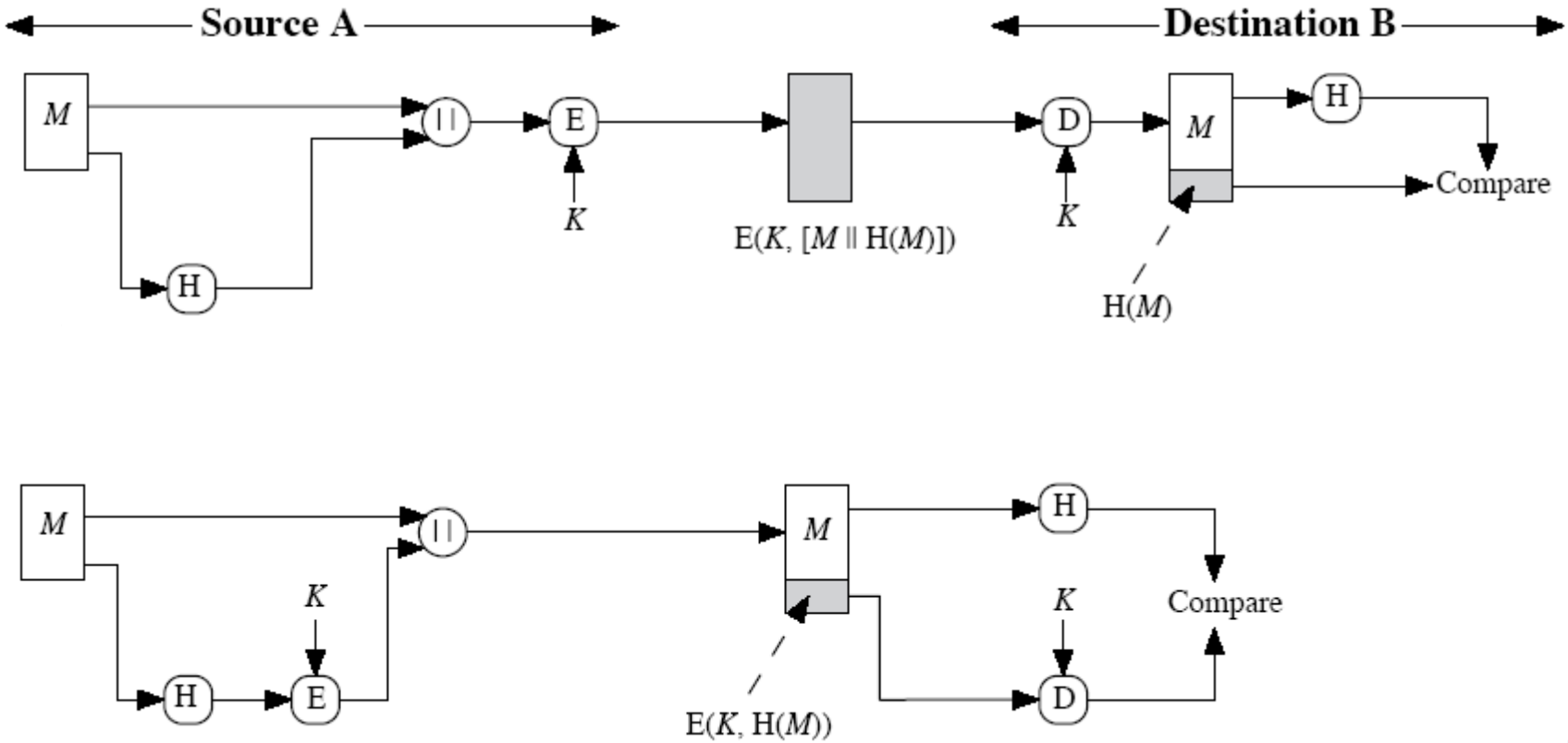


(b) Man-in-the-middle attack

Figure 11.2 Attack Against Hash Function

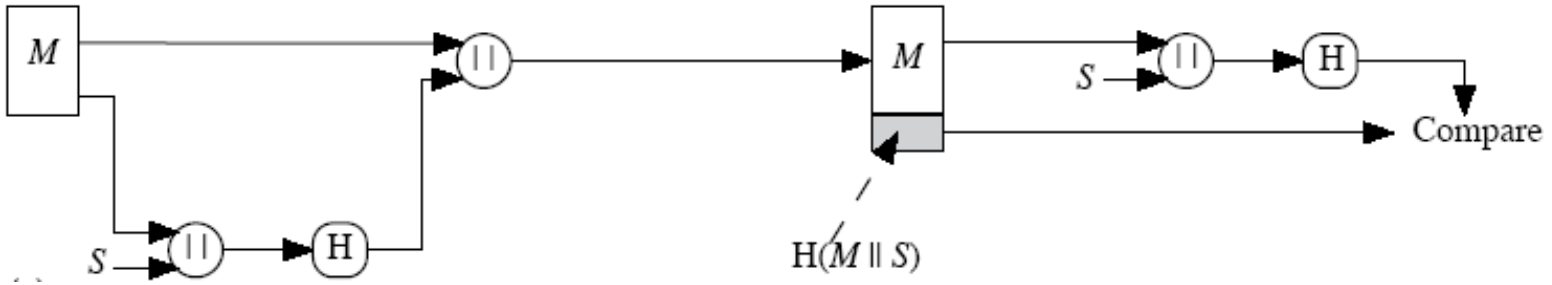
Hash based message authentication using symmetric encryption

- with confidentiality



Other Hash based message authentication techniques

- Authentication is based on a shared-secret *s*, but no encryption function is employed



Keyed Hash Functions

- it is better to have a MAC using a hash function rather than a block cipher
  - because hash functions are generally faster
  - not limited by export controls unlike block ciphers
- hash functions are not designed to work with a key
- Solution: hash includes a key along with the message
- original proposal:

```
KeyedHash = Hash(Key || Message)
```

  - by Gene Tsudik (1992)
- eventually led to development of HMAC
  - by Bellare, Kanetti and Krawczyk

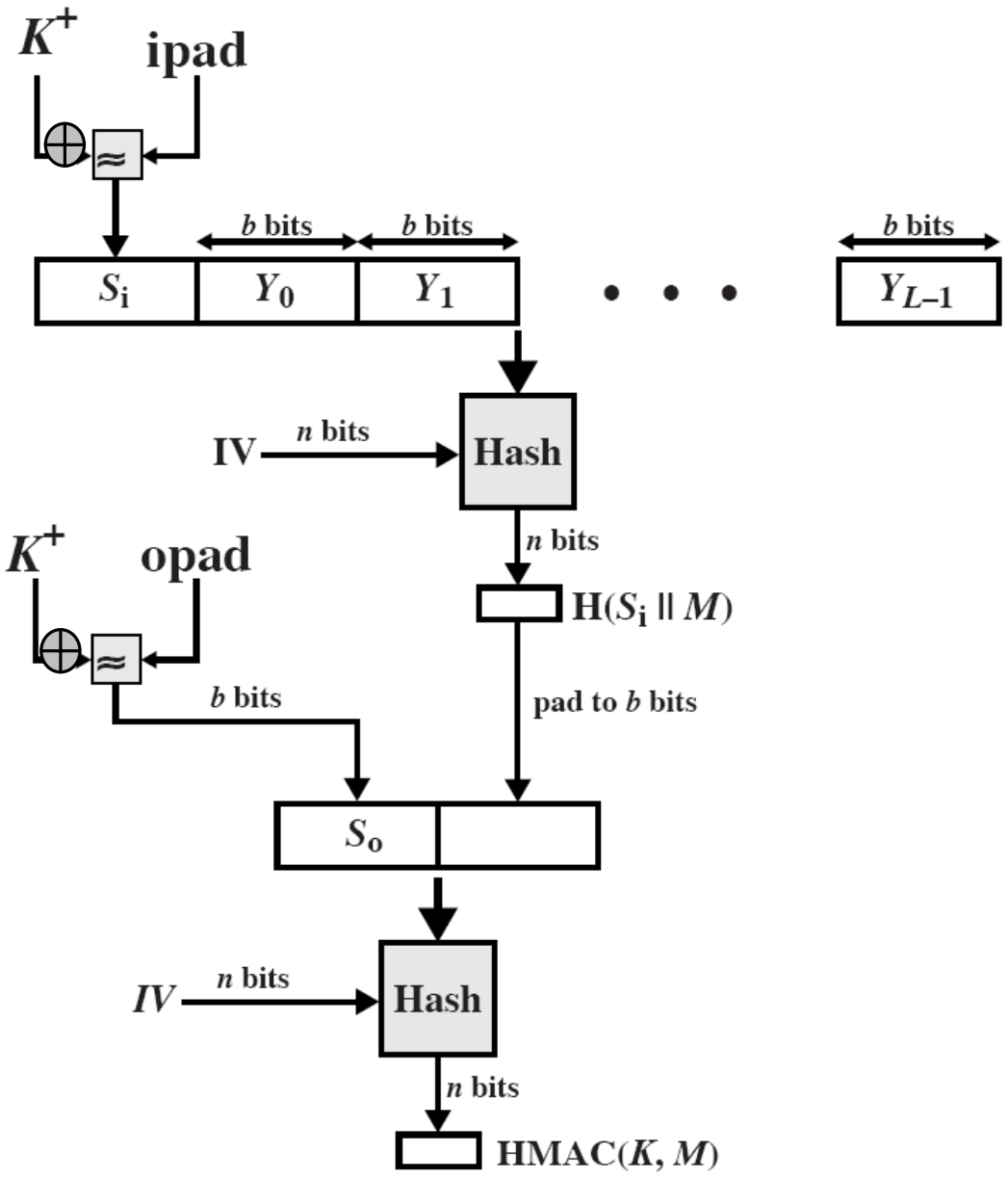
HMAC

- specified as Internet standard RFC2104
  - used in several products and standards including IPSec and SSL
- uses hash function on the message:

$$HMAC_K = Hash[(K^+ \text{ XOR opad}) \,||\, Hash[(K^+ \text{ XOR ipad}) \,||\, M)]]$$

- where K^+ is the key padded out to block size of the hash function
- and opad, ipad are some padding constants
- overhead is just 3 more blocks of hash calculations than the message needs alone
- any hash function (MD5, SHA-1, …) can be used

HMAC structure



HMAC Security

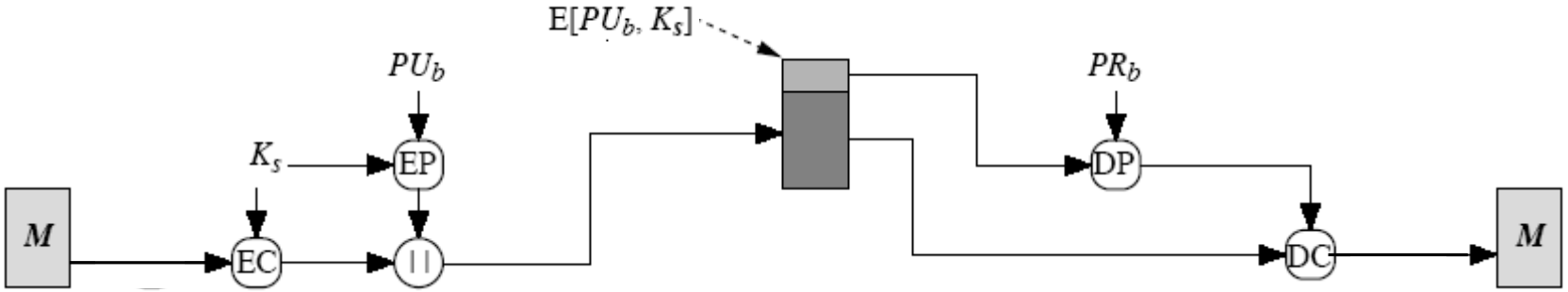
- HMAC assumes a secure hash function
  - as their creators said
    - "you cannot produce good wine using bad grapes"
- it has been proved that attacking HMAC is equivalent the following attacks on the underlying hash function
  - brute force attack on key used
  - birthday attack
    - find M and M' such that their hashes are the same
    - since keyed, attacker would need to observe a very large ($2^{n/2}$ messages) number of messages that makes the attacks infeasible
    - Let's discuss if MD5-based HMAC is secure.

Message Encryption

- Public key encryption for the bulk message is too costly
  - bulk encryption should be done using symmetric (conventional) crypto
- If a key is mutually known (e.g. if D-H is used)
  - use it to encrypt data
  - this method is useful for connection oriented data transfers where the same key is used for several data blocks
- If no key is established before
  - mostly for connectionless services (such as e-mail transfer)
  - best method is enveloping mechanism

Digital Envelopes

- A randomly chosen one-time symmetric encryption key is encrypted with public key of the recipient
- fast en/decryption without pre-establishment of keys



EC: Conventional Encryption

EP: Public-key Encryption

Ks: Session key (one-time)

DC: Conventional Decryption

DP: Public-key Decryption

Summary

- Diffie-Hellman Key Exchange
- Hash Functions
  - General idea
  - Requirements and Security
  - Most common hash functions
- Digital Signatures
  - RSA signature
  - DSA signature
- Message Authentication Code
- Keyed Hash Functions
- HMAC: Idea and Security