CS-552

CYBER FORENSICS

Introduction to Cryptography
Part-2

Public-Key Cryptography – General Characteristics - 1

public-key/two-key/asymmetric cryptography

- A concept, there are several such cryptosystems

probably the only revolution in the 3000 years of history of cryptography

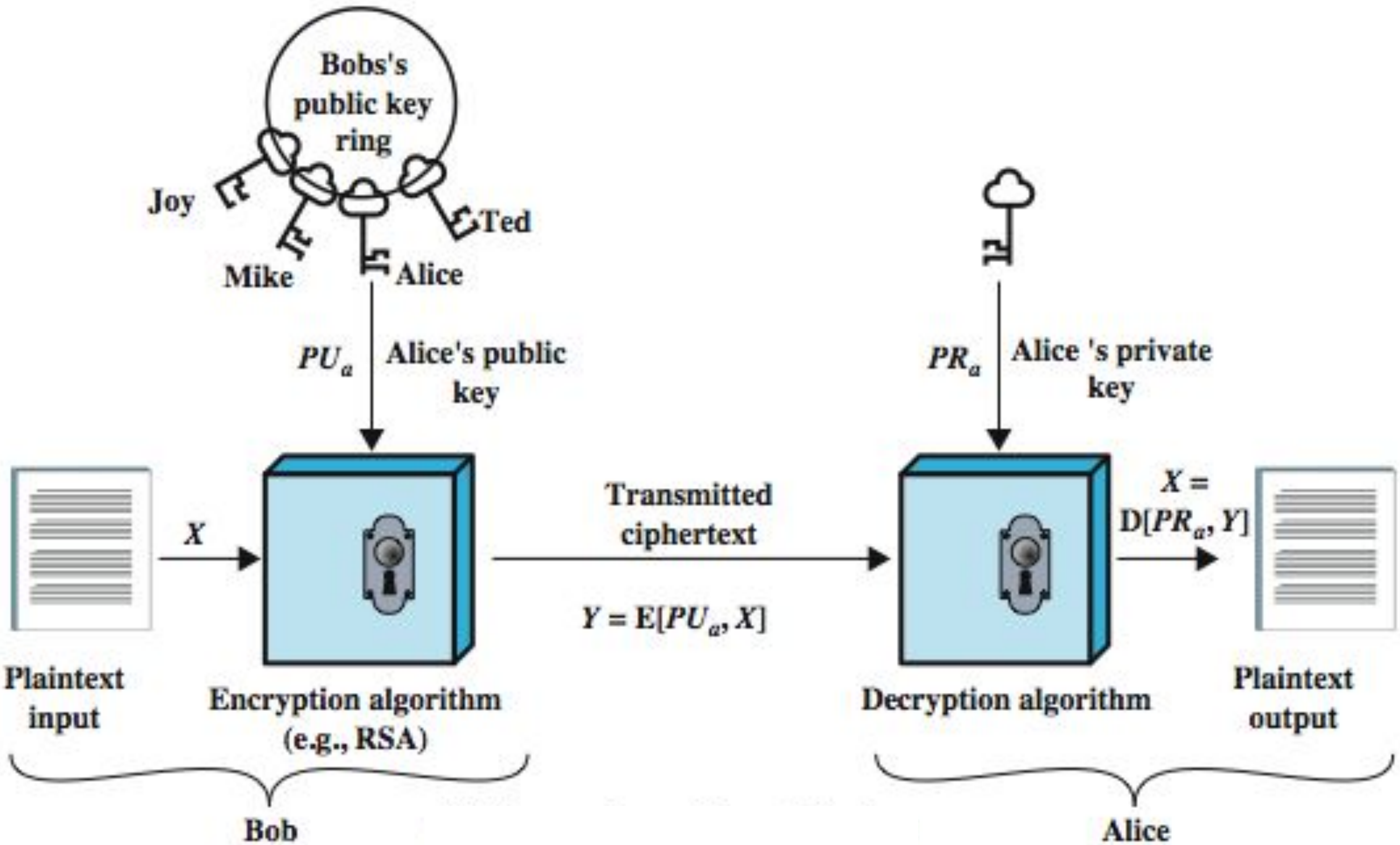
uses 2 keys

- public-key
  - may be known by anybody, and can be used to encrypt messages, and verify signatures
- private-key
  - known only to the owner, used to decrypt messages, and sign (create) signatures

Public-Key Cryptography – General Characteristics - 2

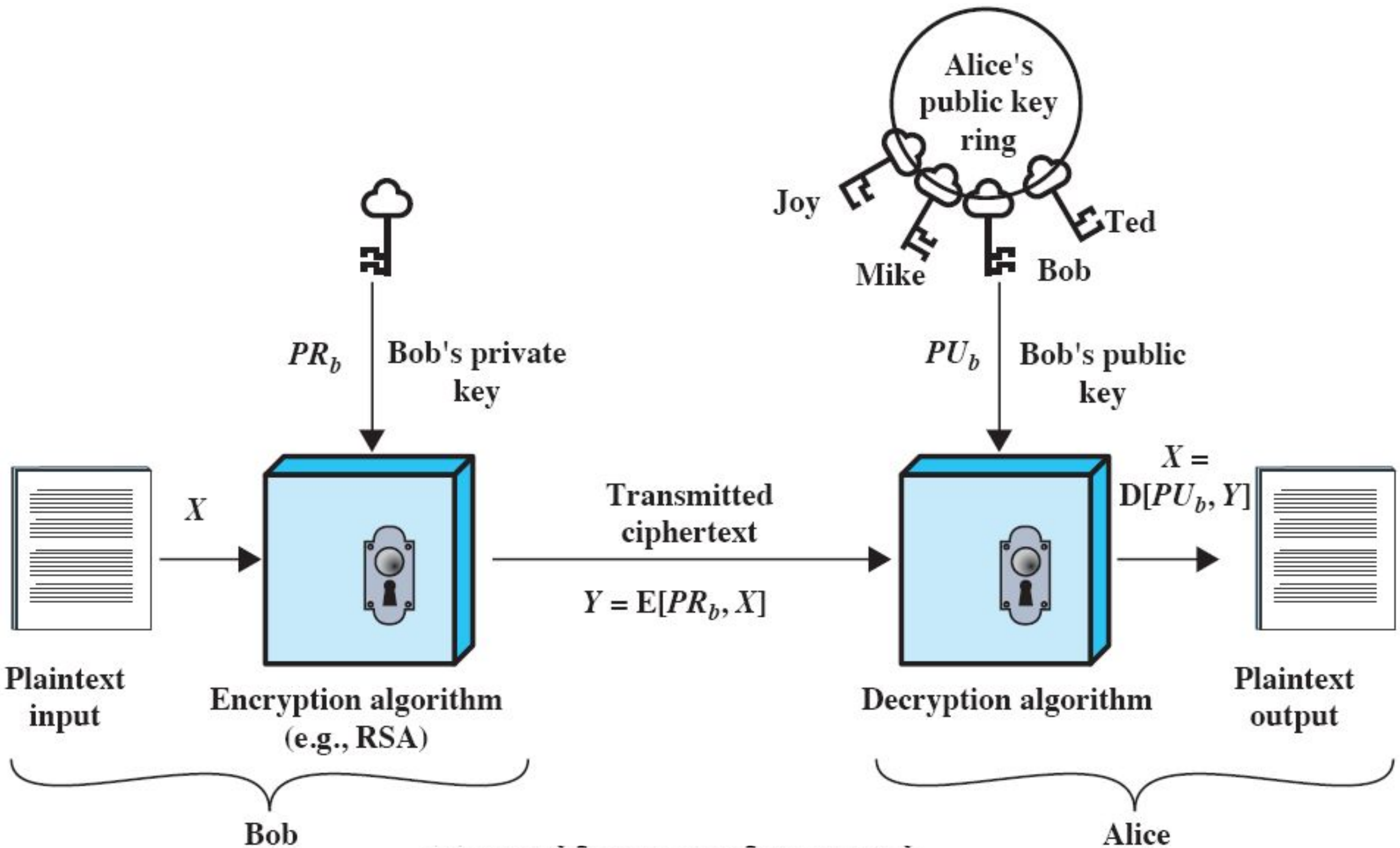
Keys are related to each other but it is not feasible to find out private key from the public one

Public-Key Cryptography - Encryption



Public-Key Cryptography – Authentication –

this is the general idea; actual implementation is via digital signatures and a bit different than this



Public-Key Cryptography – General Characteristics

based on **number theoretic hard problems**

- rather than substitutions and permutations

3 <u>misconceptions</u> about PKC

- it replaces symmetric crypto

  - PKC rather complements private key crypto
- PKC is more secure

  - no evidence for that, security mostly depends on the key size in both schemes
- key distribution is trivial in PKC since public keys are public
  - making something public is not easy. How can you make sure that a public key belongs to the intended person?

  - key distribution is easier, but not trivial

Invention of PKC

PKC is invented by Whitfield Diffie and Martin Hellman in 1976

- PhD student – advisor pair at Stanford Univ.

Some gives credit to Ralph Merkle too

NSA says that they knew PKC back in 60's

First documented introduction of PKC is by James Ellis of UK's CESG (Communications-Electronics Security Group) in 1970

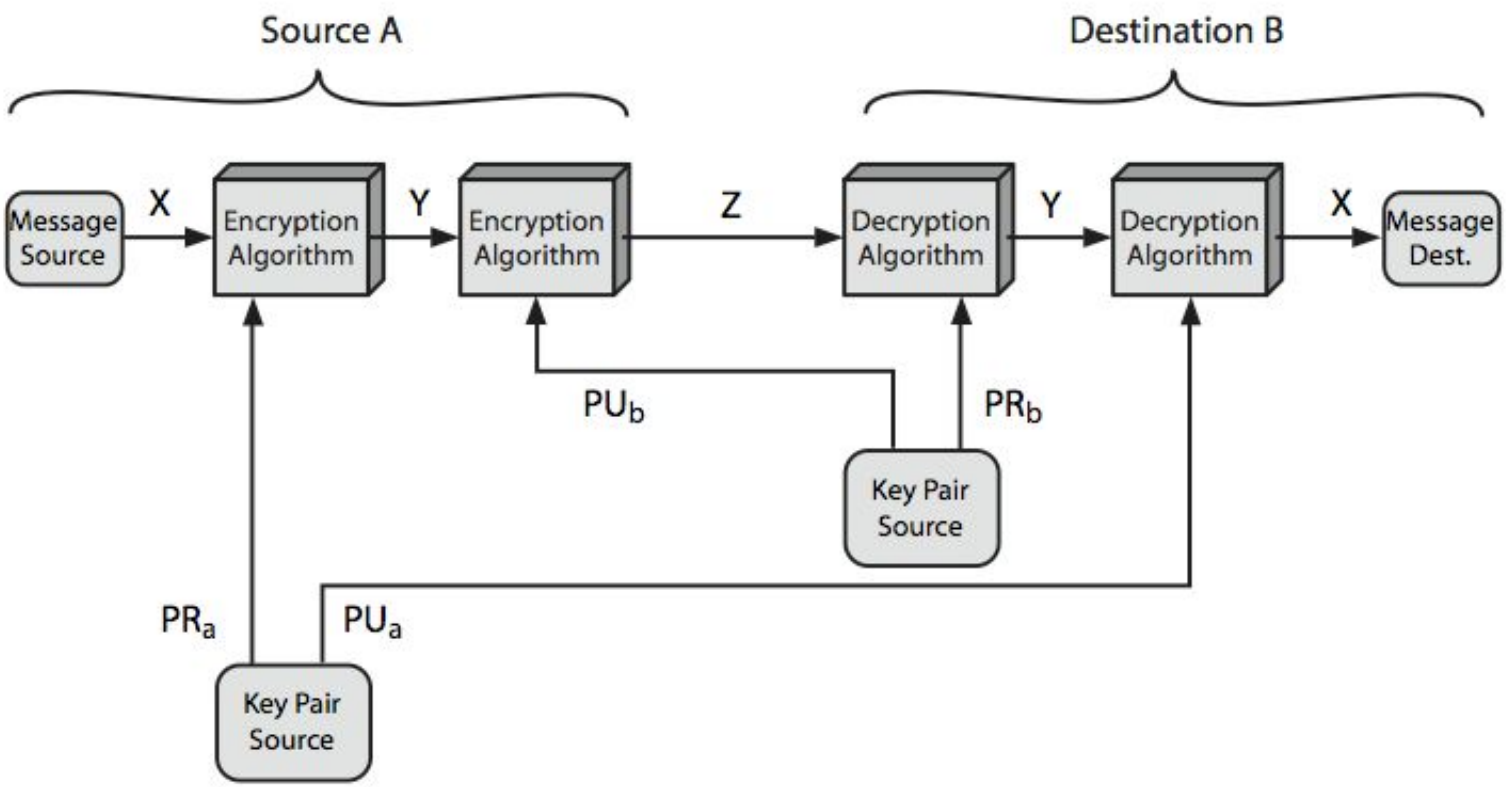
- was a classified report
- declassified in 1987

Why Public-Key Cryptography?

Initially developed to address two key issues:

- key distribution
  - symmetric crypto requires a trusted Key Distribution Center (KDC)
  - in PKC you do not need a KDC to distribute secret keys, but you still need trusted third parties
- digital signatures (non-repudiation)
  - not possible with symmetric crypto

Public-Key Cryptosystems



PU_a	A's Public Key	PU_b	B's Public Key
PR_a	A's Private Key	PR_b	B's Private Key

Applications of Public-Key Cryptosystems

3 categories

- encryption/decryption
  - to provide secrecy
- digital signatures
  - to provide authentication and non-repudiation
- key exchange
  - to agree on a session key

some algorithms are suitable for all uses, others are specific to one

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Some Issues of Public Key Schemes

like ~~private key schemes brute force attack is always theoretically~~ possible

- use large keys
- consider the security vs. performance tradeoff

due to public key / private key relationships, number of bits in the key should be much larger than symmetric crypto keys

- to make the hard problem really hard
- 80-bit symmetric key and 1024-bit RSA key has comparable resistance to cryptanalysis

a consequence of use of large keys is having slower encryption and decryption as compared to private key schemes

- thus, PKC is not a proper method for bulk encryption

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
  - published in 1978
- best known and widely used public-key scheme
- was patented and patent was used by RSA Inc
  - however patent expired in 2000
- uses large integers
  - 1024+ bits
- security depends on the cost of factoring large numbers

RSA Key Setup

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

e is usually a small number

gcd: greatest common divisor

RSA Use

to encrypt a message M < n, the sender:

- obtains public key of recipient `PU={e,n}`
- computes: $C=M^e \bmod n$, where $0 \le M < n$

to decrypt the ciphertext C the owner:

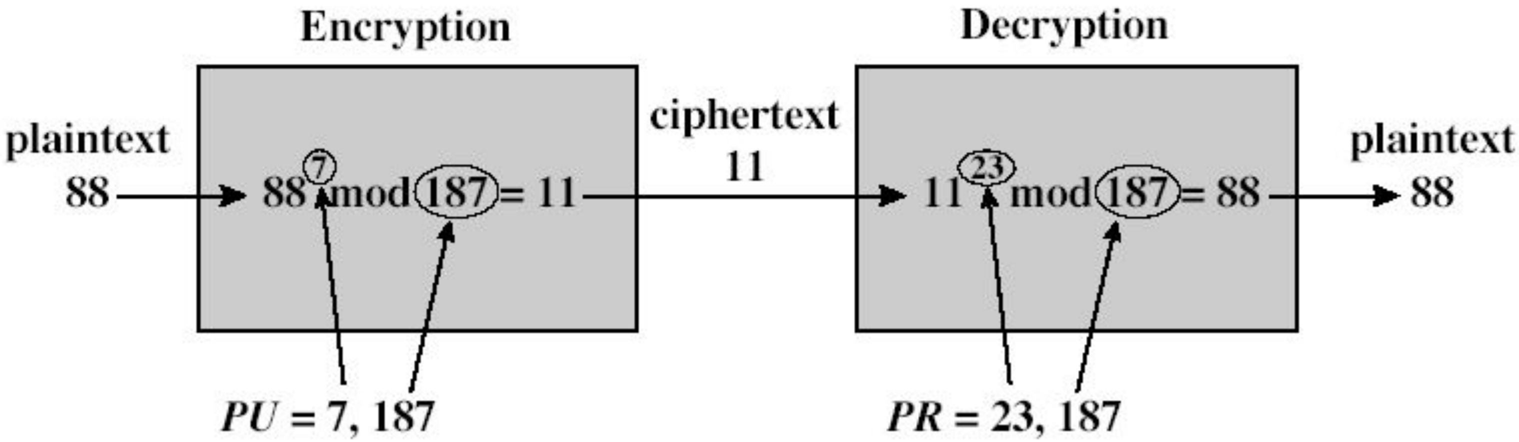
- uses their private key `PR={d,n}`
- computes: $M=C^d \bmod n$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

note that the message M must be smaller than the modulus n: use several blocks if needed

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

RSA Example



p = 17, q = 11, n = p*q= 187

Φ(n) = 16*10 =160, pick e=7, d.e=1 mod Φ(n) d = 23

Why RSA Works

because of Euler's Theorem:

$a^{\varnothing(n)} \bmod n = 1$ where $\gcd(a,n)=1$

in RSA have

- $n=p.q$
- $\varnothing(n)=(p-1)(q-1)$
- carefully chose e & d to be inverses mod $\varnothing(n)$
  - i.e. $e.d = 1 \bmod \varnothing(n)$
- hence $e.d=1+k.\varnothing(n)$ for some k

hence

$$C^d = M^{e.d} = M^{1+k.\varnothing(n)} = M^1.(M^{\varnothing(n)})^k$$

$$= M^1.(1)^k = M^1 = M \bmod n$$

Computational Aspects

An RSA implementation requires complex arithmetic

- modular exponentiation for encryption and encryption
- primality tests
- finding inverse of e mod $\Phi(n)$

There are acceptably fast solutions to those computational problems (see Stallings for details)

RSA Security

4 approaches of attacking on RSA

- brute force key search
  - not feasible for large keys
  - actually nobody attacks on RSA in that way
- mathematical attacks
  - based on difficulty of factorization for large numbers as we shall see in the next slide
- side-channel attacks
  - based on running time and other implementation aspects of decryption
- chosen-ciphertext attack
  - Some algorithmic characteristics of RSA can be exploited to get information for cryptanalysis

Factorization Problem

3 forms of mathematical attacks

- factor n=p.q, hence find ø(n) and then d
- determine ø(n) directly and find d
  - is equivalent of factoring n
- find d directly
  - as difficult as factoring n

so RSA cryptanalysis is focused on factorization of large n

Factorization Problem – RSA Challenges

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve
193	640	November 2005		
232	768	December 2009		
212	704	July 2012		
220	729	May 2016		
230	762	August 2018		

Reasons of improvement in Factorization

increase in computational power

biggest improvement comes from improved algorithm

- “Quadratic Sieve” to “Generalized Number Field Sieve” and “Lattice Sieve”

Side Channel Attacks

For example timing attacks

- based on timing variations in operations
- some operations are slow, some faster depending on the key

In RSA, there are time variations in exponentiation during decryption

countermeasures

- use constant exponentiation time
- add random delays
- blinding (offered by RSA Inc.)
  - multiply the ciphertext by a random value so that attacker cannot know the ciptertext being decrypted
  - let’s see on the board

Key size and Security

Validity period	Minimal RSA key length (bits)	Equivalent symmetric key length (bits)
2003-2010	1024	80
2010-2030	2048	112
2030-	3072	128

Recommendations of RSA Security Inc.
May 6, 2003

Five security levels

NIST SP 800-56

Level	RSA / DH	ECC	Symmetric ciphers
1	1024	160	80
2	2048	224	112
3	3072	256	128
4	8192	384	192
5	15360	512	256

Diffie-Hellman Key Exchange

First PKC offered by Diffie and Hellman in 1976

still in commercial use

purpose is secure key-exchange

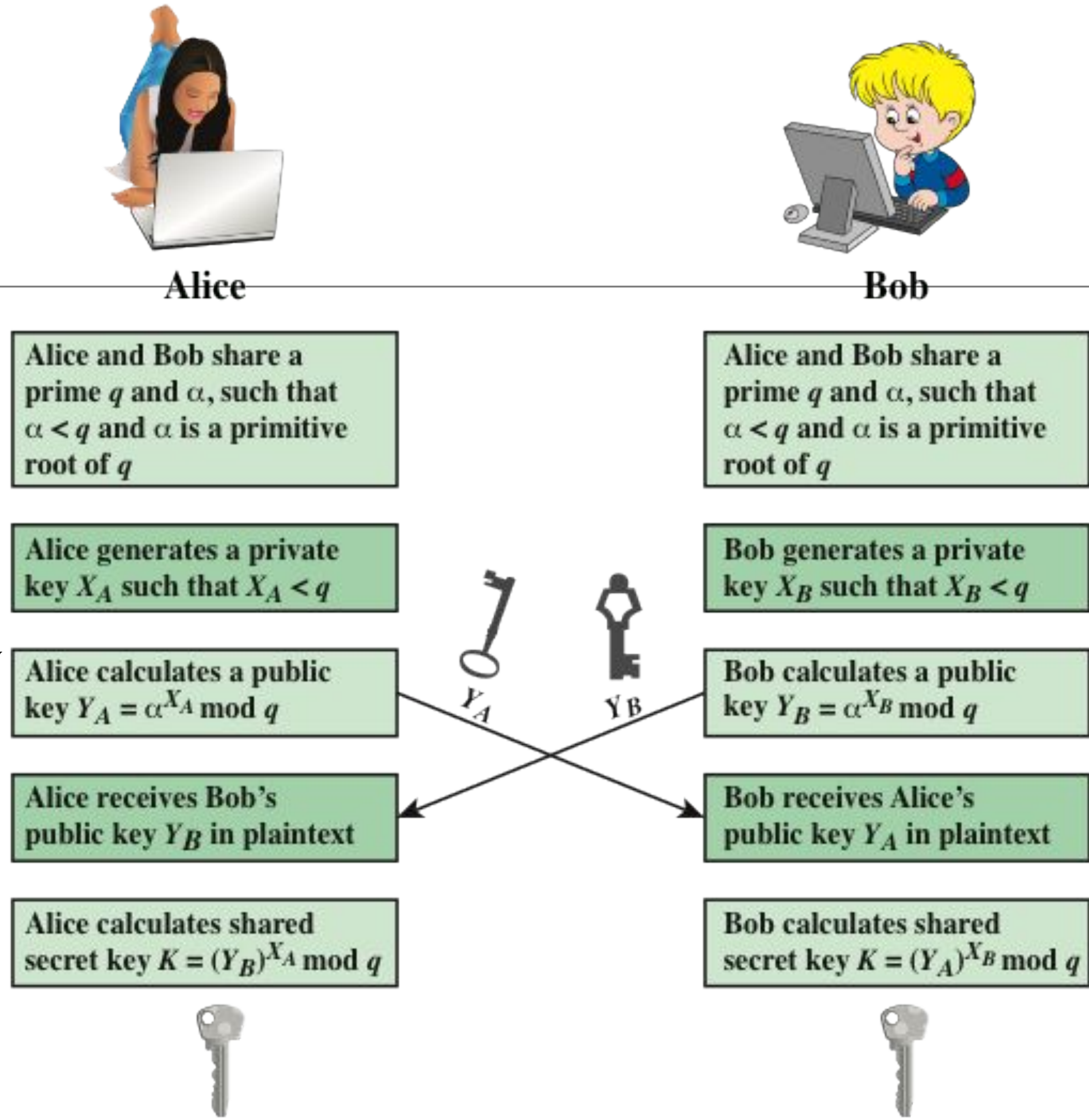
- actually key "agreement"
- both parties agree on a session key without releasing this key to a third party
  - to be used for further communication using symmetric crypto

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

D-H Key Exchange

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 – 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

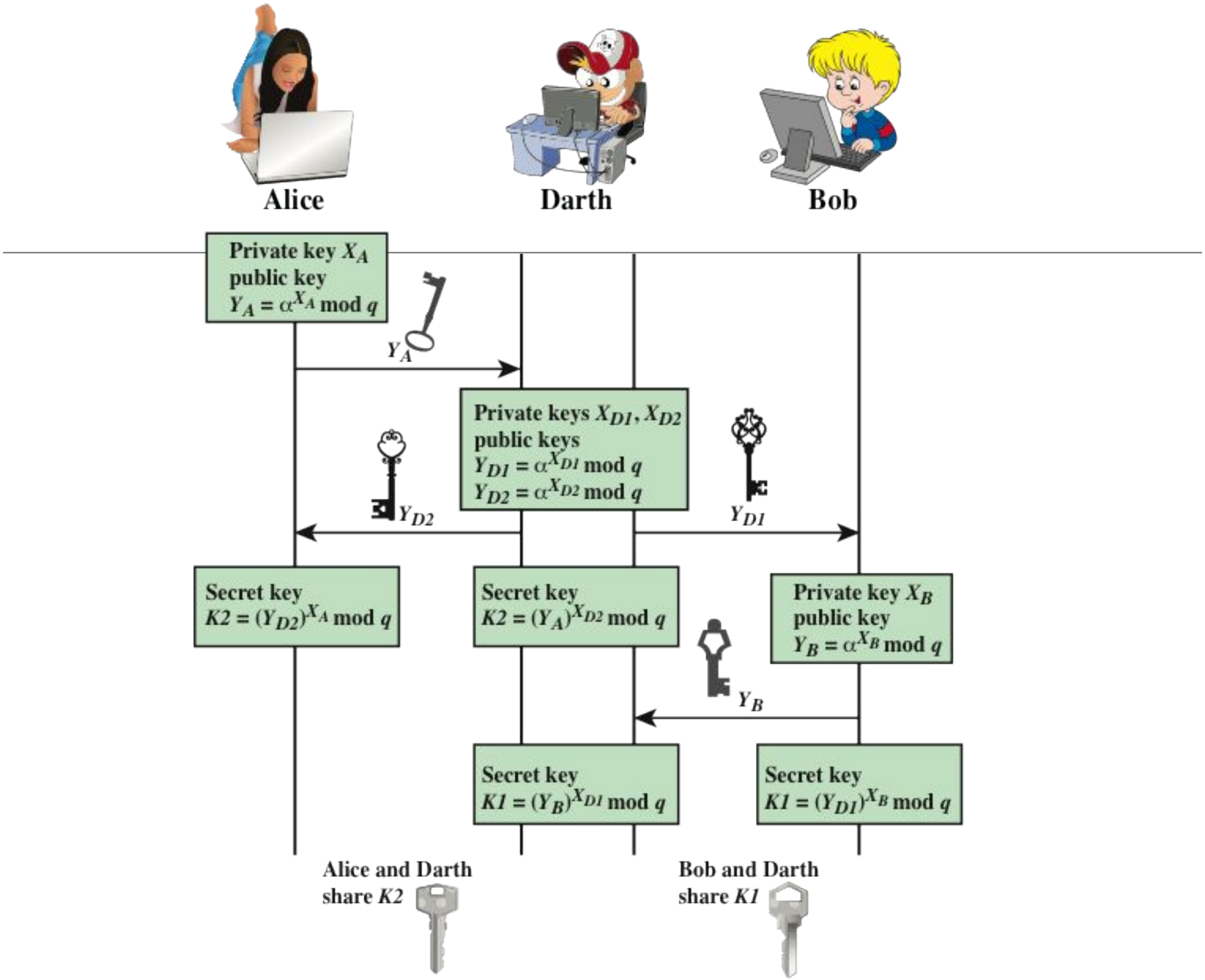


Figure 10.2 Man-in-the-Middle Attack

PKC - Remained

How to use RSA for digital signatures?

DSA / DSS

- Digital Signature Algorithm / Standard

Elliptic Curve Cryptography (ECC)

- ECDSA – Elliptic Curve DSA
- ECDH – Elliptic Curve D-H

First we will see hash functions

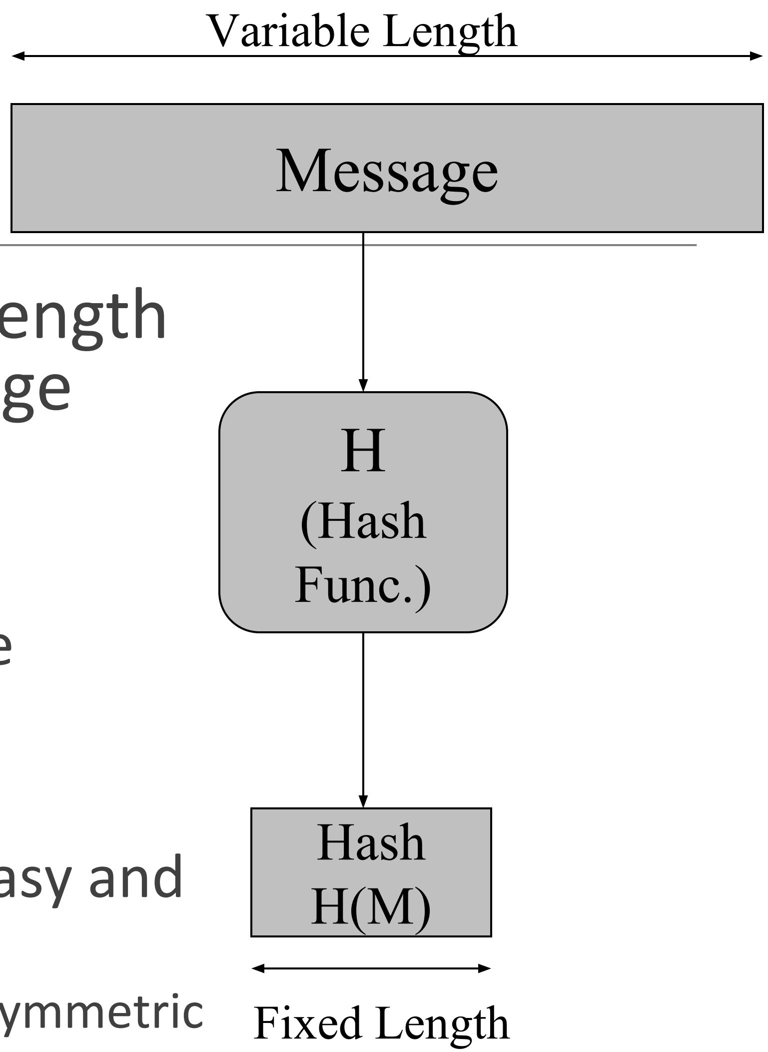
- several application areas

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

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

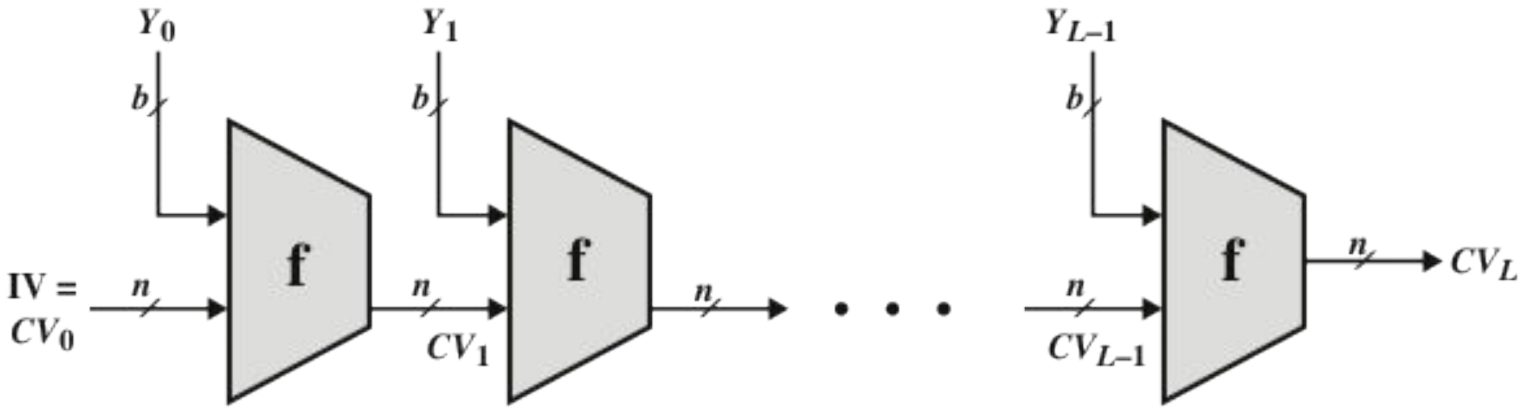
(Strong) collision resistance

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

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-1	SHA-224	SHA-256	SHA-384	SHA-512
		SHA-2			
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
- However, standardization process is delayed (standard has been published on August 5, 2015)
  - There had been controversies (read the wikipedia page of SHA-3)
  - I am not sure if it is going to replace SHA-2

Message Digest Size	224	256	384	512
Message Size	no maximum	no maximum	no maximum	no maximum
Block Size (bitrate r)	1152	1088	832	576
Word Size	64	64	64	64
Number of Rounds	24	24	24	24
Capacity c	448	512	768	1024
Collision resistance	2^{112}	2^{128}	2^{192}	2^{256}
Second preimage resistance	2^{224}	2^{256}	2^{384}	2^{512}

Digital Signatures

Mechanism for non-repudiation

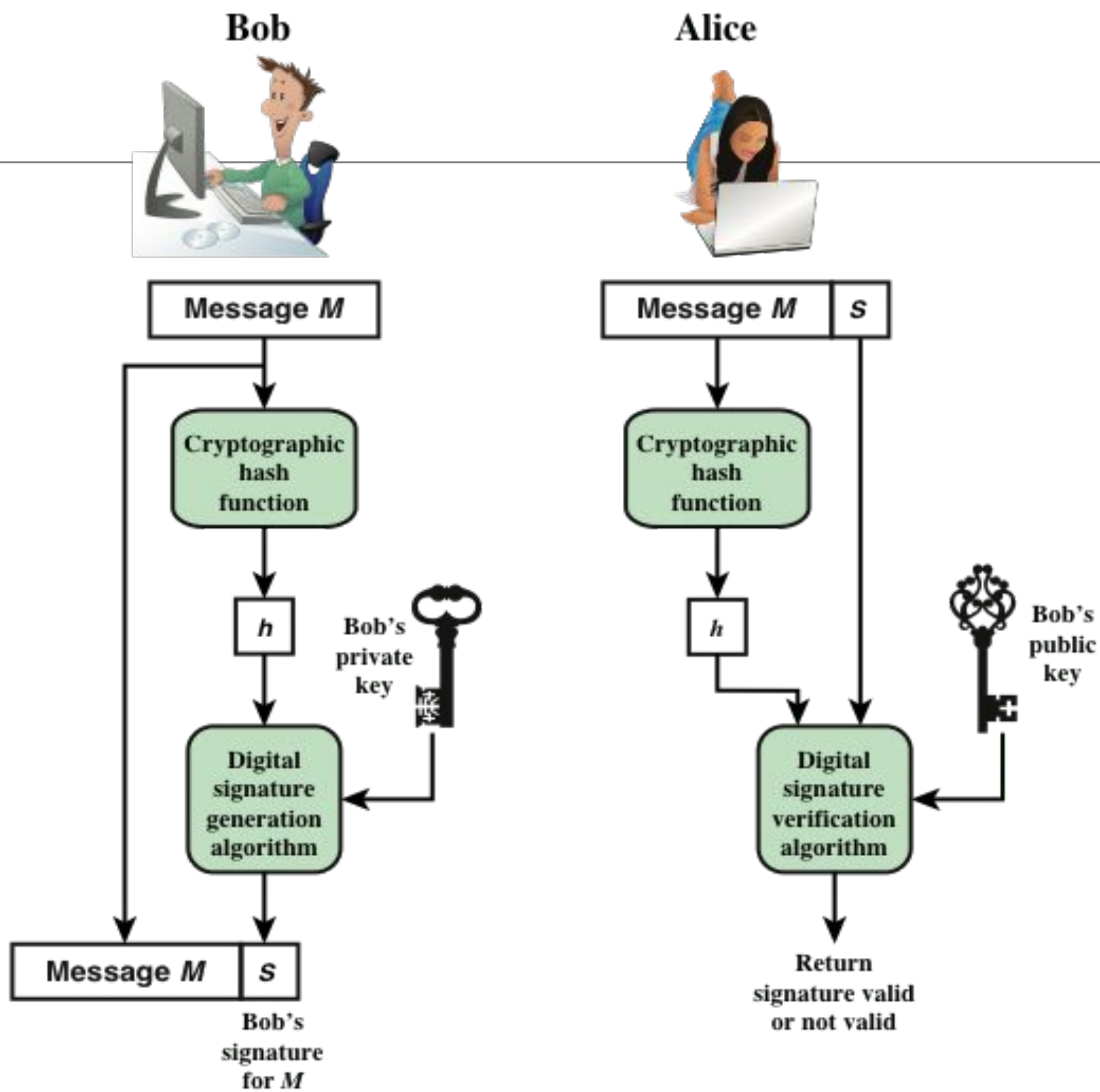
Basic idea

- 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

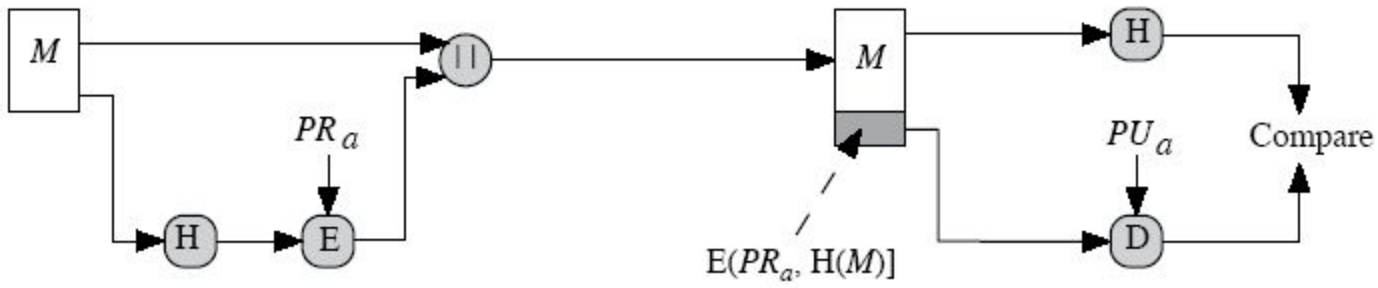
- Why?

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

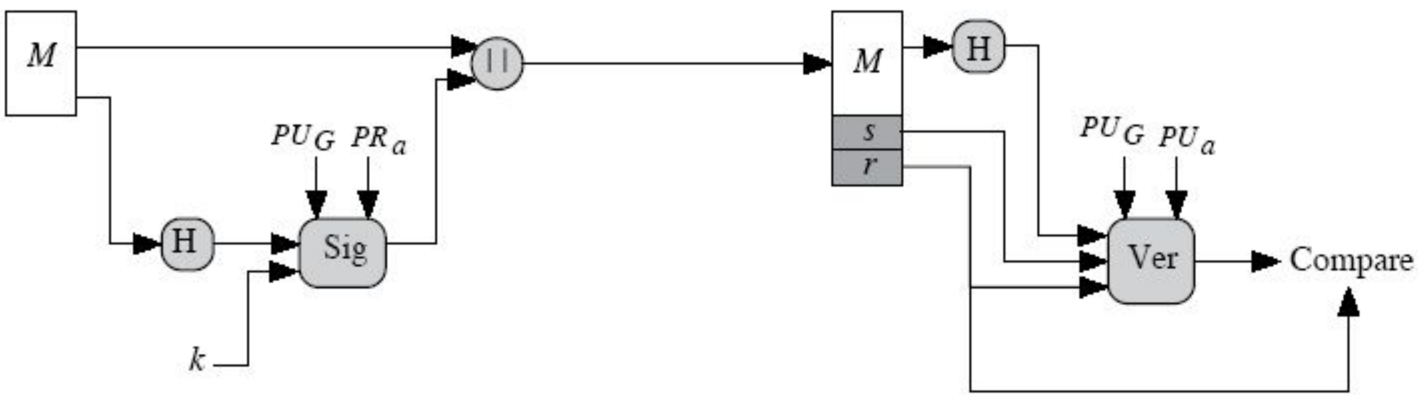
D: RSA Public Key Operation PU_a: Sender's Public Key

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

Elliptic Curve Cryptography

Based on the difficulty of Elliptic Curve Discrete Logarithm problem

- details are not in the scope of this course
- a concise description is in Sections 10.3 and 10.4 of Stallings

Actually a set of cryptosystems

- each elliptic curve is one cryptosystem
  - 160-bit, 163-bit, 233-bit, … defined in IEEE P1363 standard

Key size is smaller than RSA

- 160-bit ECC is almost has the security as 1024 bit RSA

Private Key operation is faster than RSA, public key operation is almost equal

Elliptic Curve Cryptography

Key exchange

- ECDH
  - Elliptic Curve Diffie-Hellman

Digital Signatures

- ECDSA
  - Elliptic Curve Digital Signature Algorithm

ECDH and ECDSA are standard methods

Encryption/Decryption with ECC is possible, but not common

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

Message Encryption

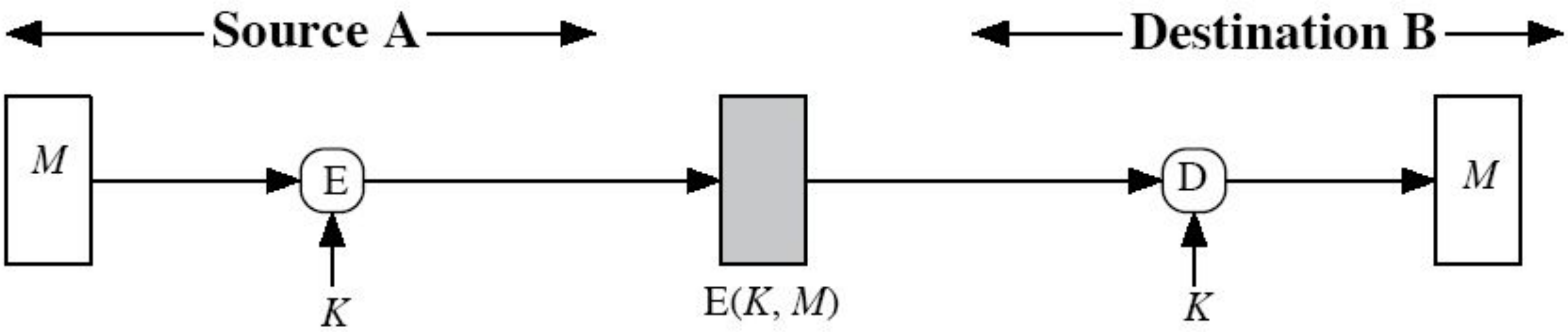
- provides message authentication, but …

Message Authentication Code Functions

- similar to encryption functions, but not necessarily reversible
- Generally Hash based MAC is used (will see)

Actually hash functions are used for message authentication in several ways (will see)

Using Message Encryption for Authentication

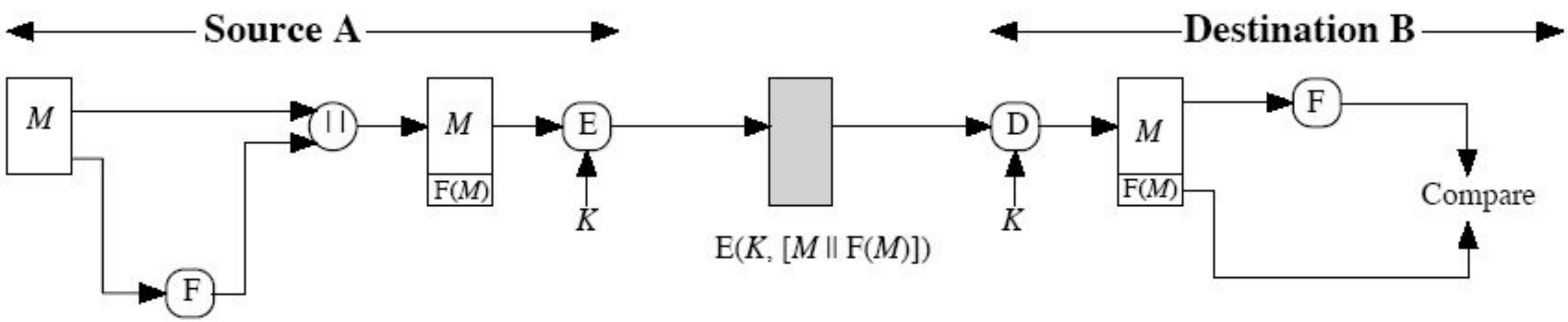


Provides encryption. What about authentication?

- yes, but there must be a mechanism to detect the restored M is the same as the sent M
  - intelligible restored plaintext (may be difficult)
  - error control codes (checksum), see next slide

Using Message Encryption for Authentication

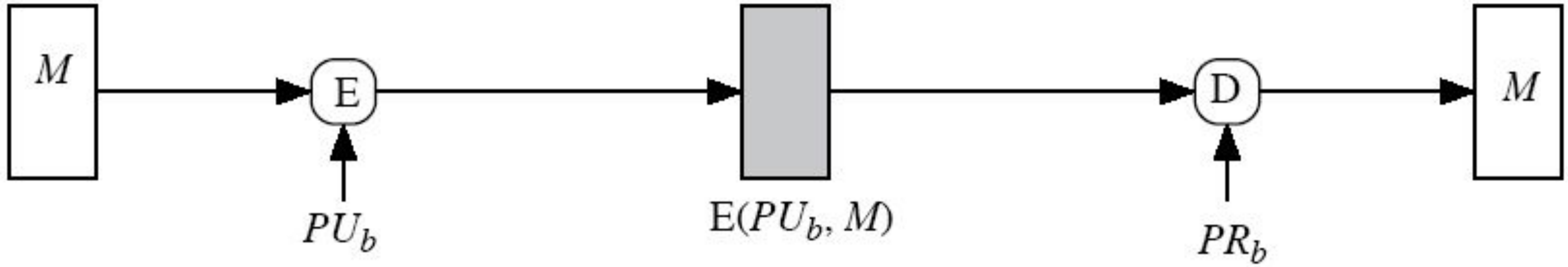
Addition of FCS (frame check sequence) helps to detect if both M's are the same or not



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) and MAC Functions

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*)

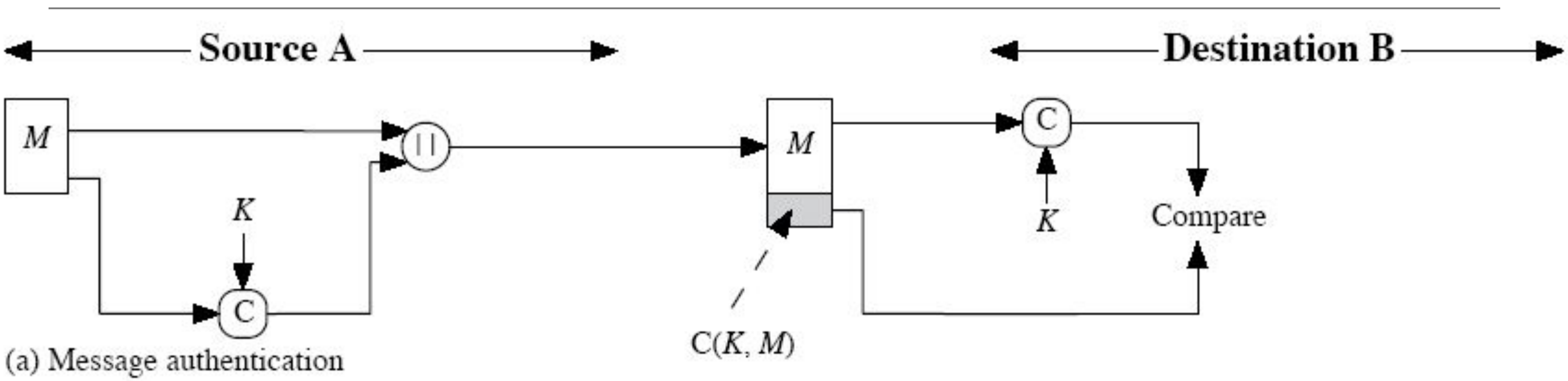
appended to message

receiver performs same computation on message and checks if matches the received MAC

provides assurance that message is unaltered and comes from sender

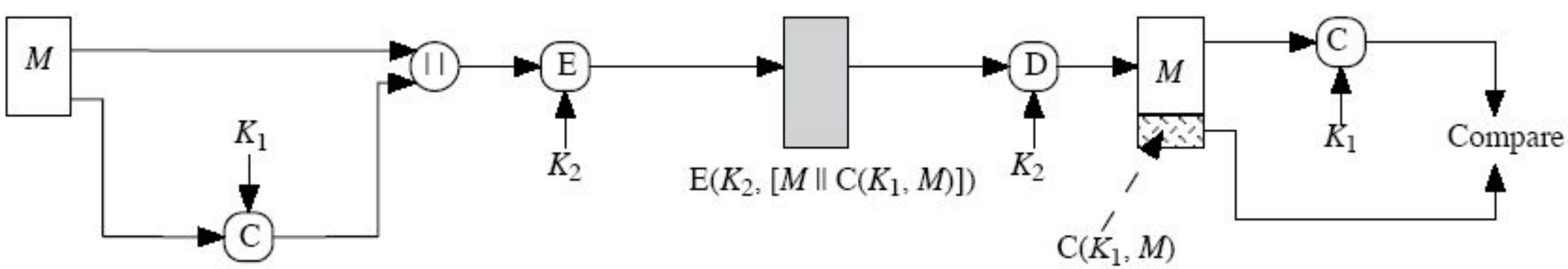
MAC – Basic Idea and Model

Only 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?

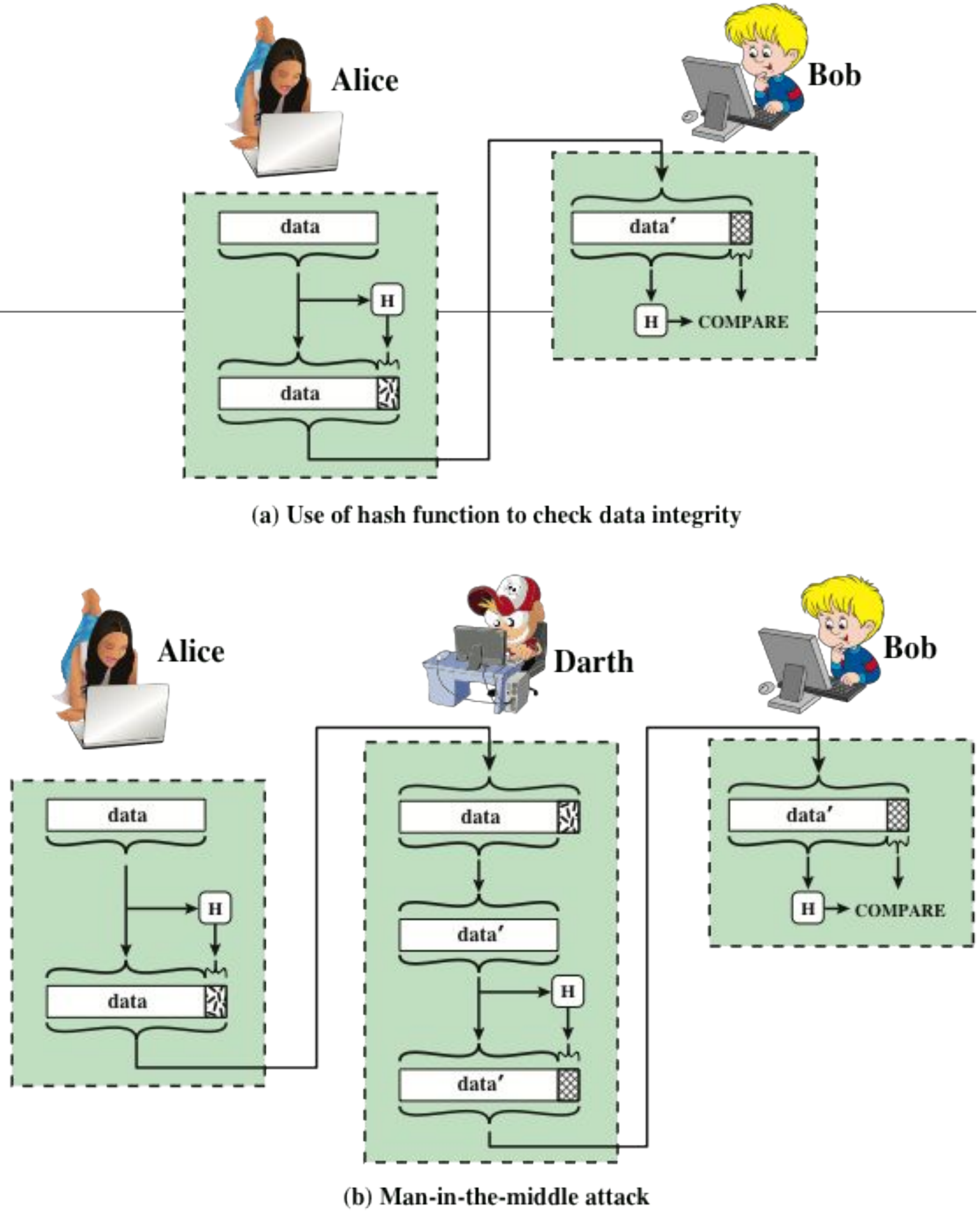
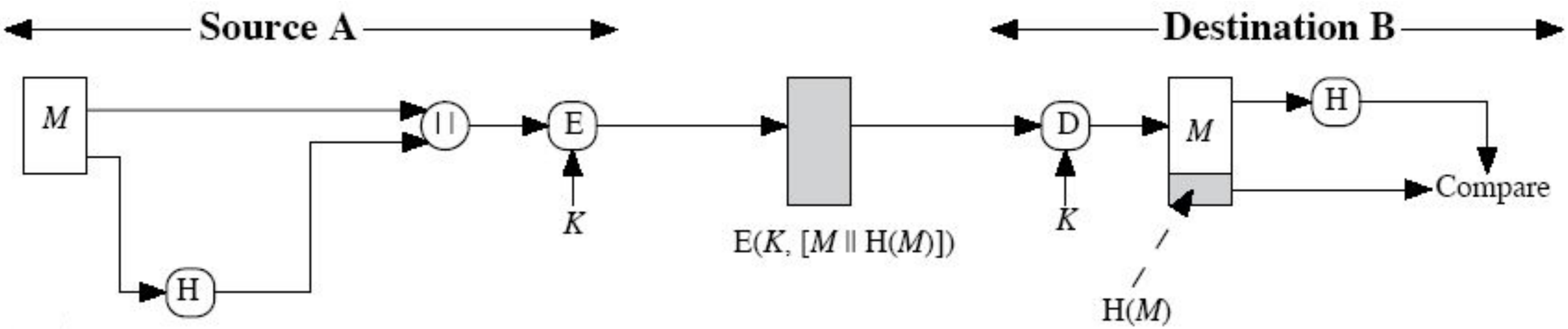
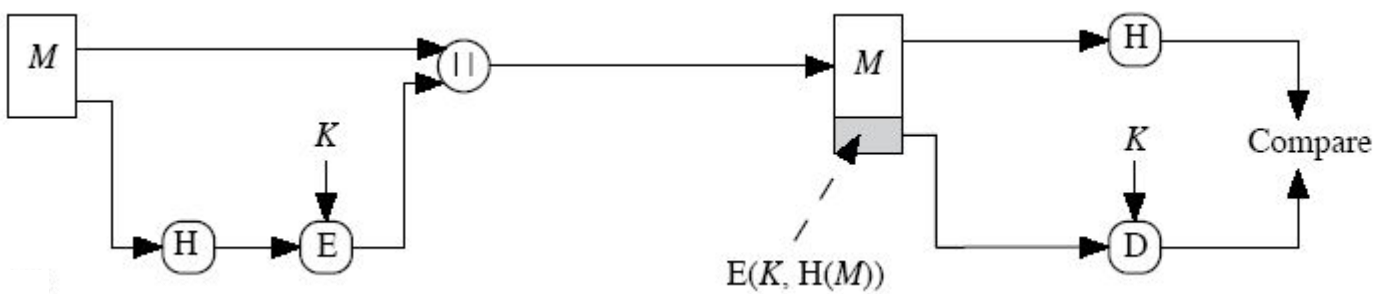


Figure 11.2 Attack Against Hash Function

Hash based message authentication using symmetric encryption

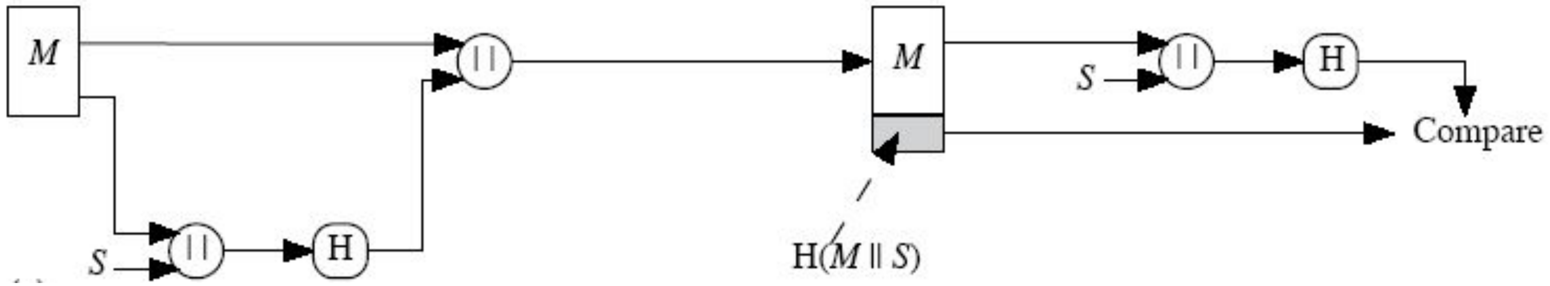
with confidentiality





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:

```
HMACK = Hash[(K+ XOR opad) ||
         Hash[(K+ XOR ipad)|| M)]]
```

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

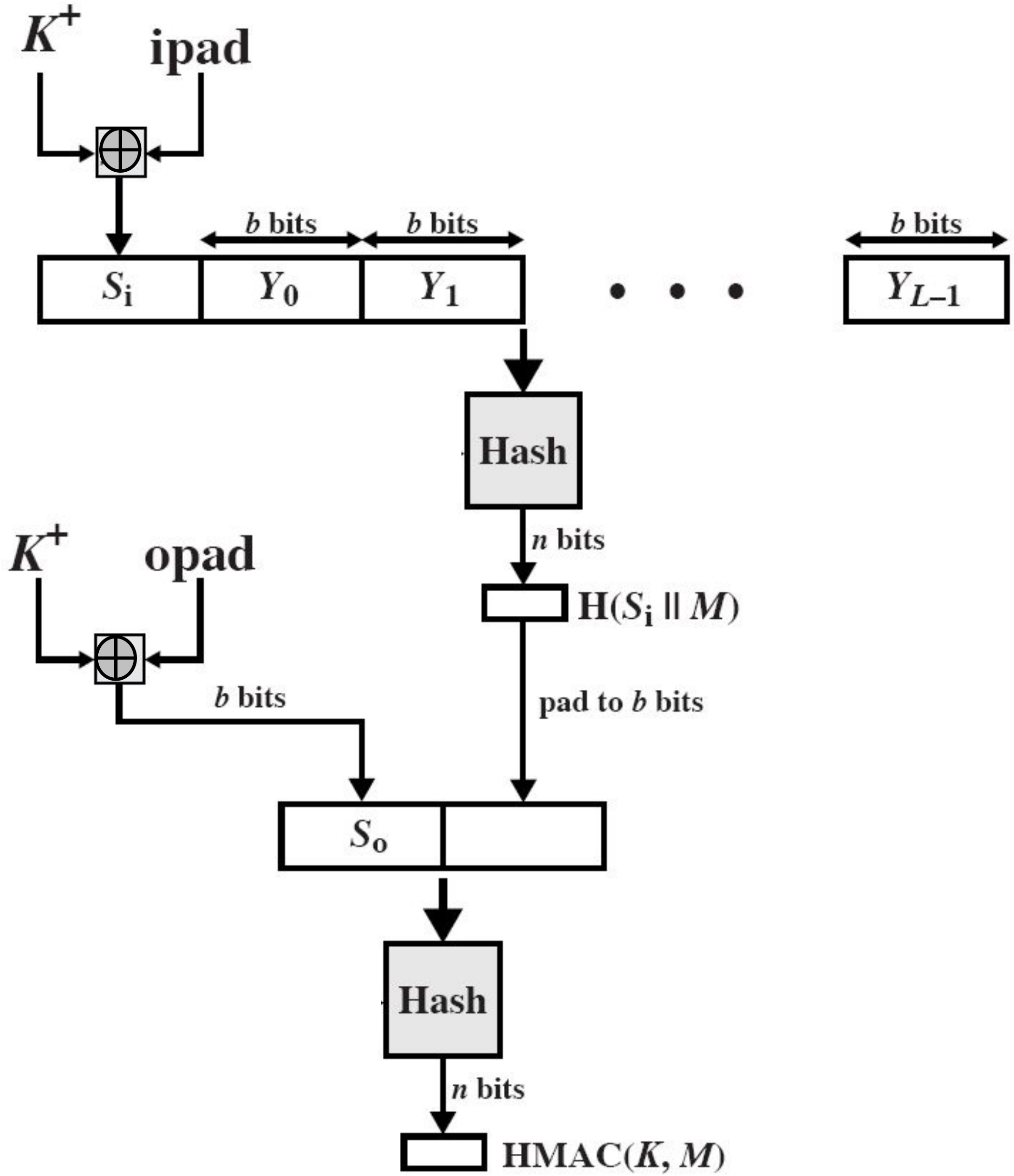
where K^+ is the key padded out to block size of the hash function

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.

CMAC - Cipher based MAC

We said Hash-based MAC is preferable

But sometimes people use Cipher based MAC due to the facts that:

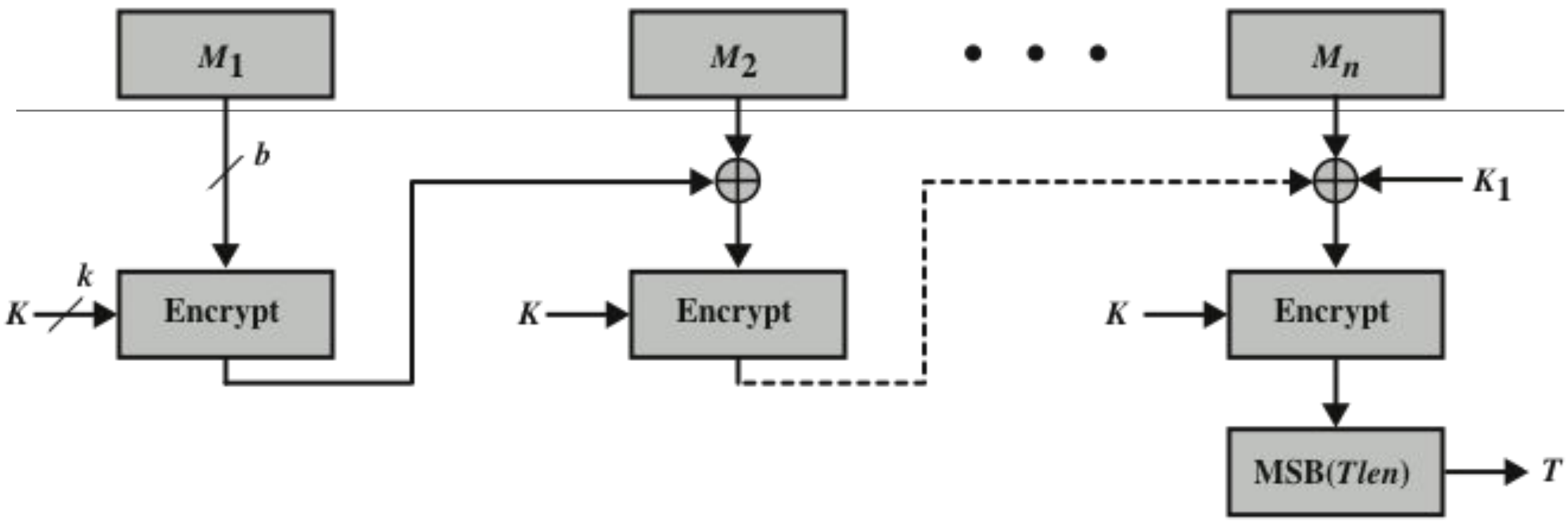
- Standard ciphers are secure enough; no need to bother with security of Hash functions
- Ciphers are used for confidentiality, so it is implemented in the system; no need to have extra implementation for hash function

NIST had and old standard based on DES but totally insecure.

Current standard is CMAC (NIST Pub. 800-38B)

- For AES (but any block cipher can be used)

CMAC - Cipher based MAC



(a) Message length is integer multiple of block size

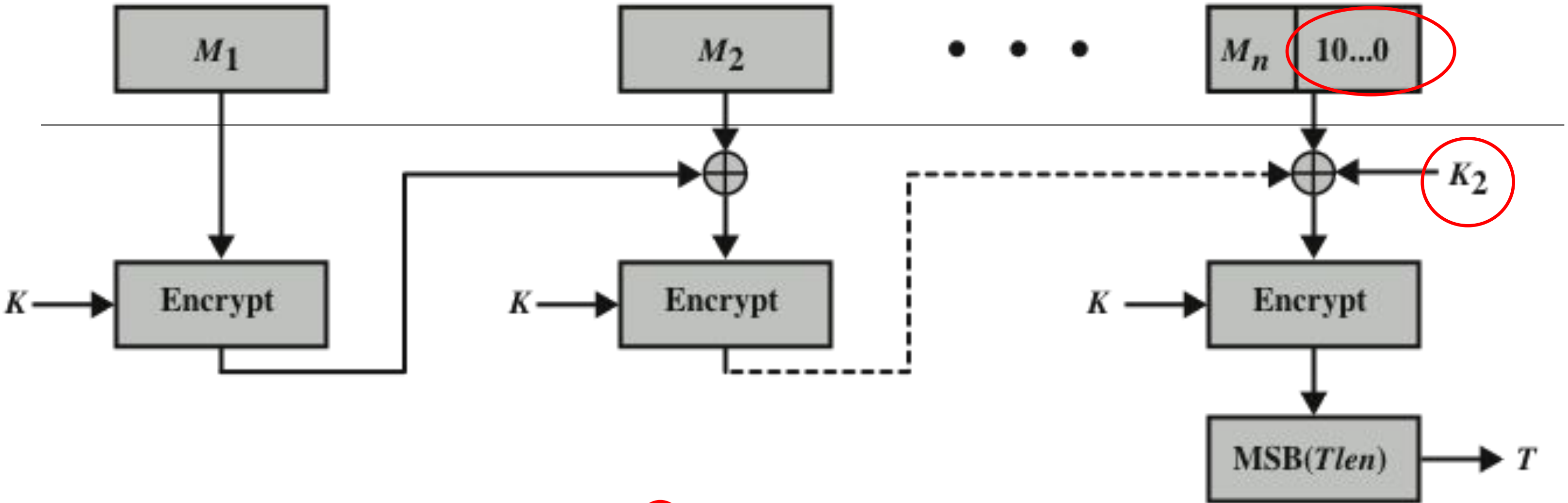
b: block size of block cipher

K: shared key

T: tag, actually it is the Message Authentication Code; most significant Tlen bits of the last block are used as message authentication code

K_1: a special conditioner for the last block; calculated as $E(K, 0^b)$ shifted one bit left

CMAC - Cipher based MAC



(b) Message length is not integer multiple of block size

b: block size of block cipher
K: shared key
T: tag, actually it is the Message Authentication Code; most significant Tlen bits of the last block are used as message authentication code
K_2: a special conditioner for the last block; calculated as $E(K, 0^b)$ shifted two bits left

Authenticated Encryption (AE)

A term used to describe encryption systems that simultaneously protect confidentiality and authenticity of communications

Approaches:

- Hashing followed by encryption: $h = H(M)$, $E(K, (M \| h))$
- Authentication followed by encryption (SSL/TLS idea)
  - $T = MAC(K_1, M)$, $E(K_2, [M \| T])$
- Encryption followed by authentication (IPSec idea)
  - $C = E(K_2, M)$, $T = MAC(K_1, C)$ pair (C, T) is sent
- Independently encrypt and authenticate (SSH idea)
  - $C = E(K_2, M)$, $T = MAC(K_1, M)$ pair (C, T) is sent

Straightforward mechanisms but need to be used very carefully by knowing the underlying characteristics of the algorithms WEP Syndrome

Chaining-Message Authentication Code (CCM)

Standardized by NIST specifically to support the security requirements of IEEE 802.11 WiFi

- NIST SP 800-38C

Variation of the encrypt-and-MAC approach

- AES encryption algorithm
- CTR mode of operation
- CMAC authentication algorithm

Single key *K* is used for both encryption and MAC algorithms

Other main inputs

- Plaintext (*P*)
- Associated data (*A*) - authenticated but not encrypted
  - Typically “protocol header”
- nonce (*N*) - different value for each payload; to avoid replay attacks

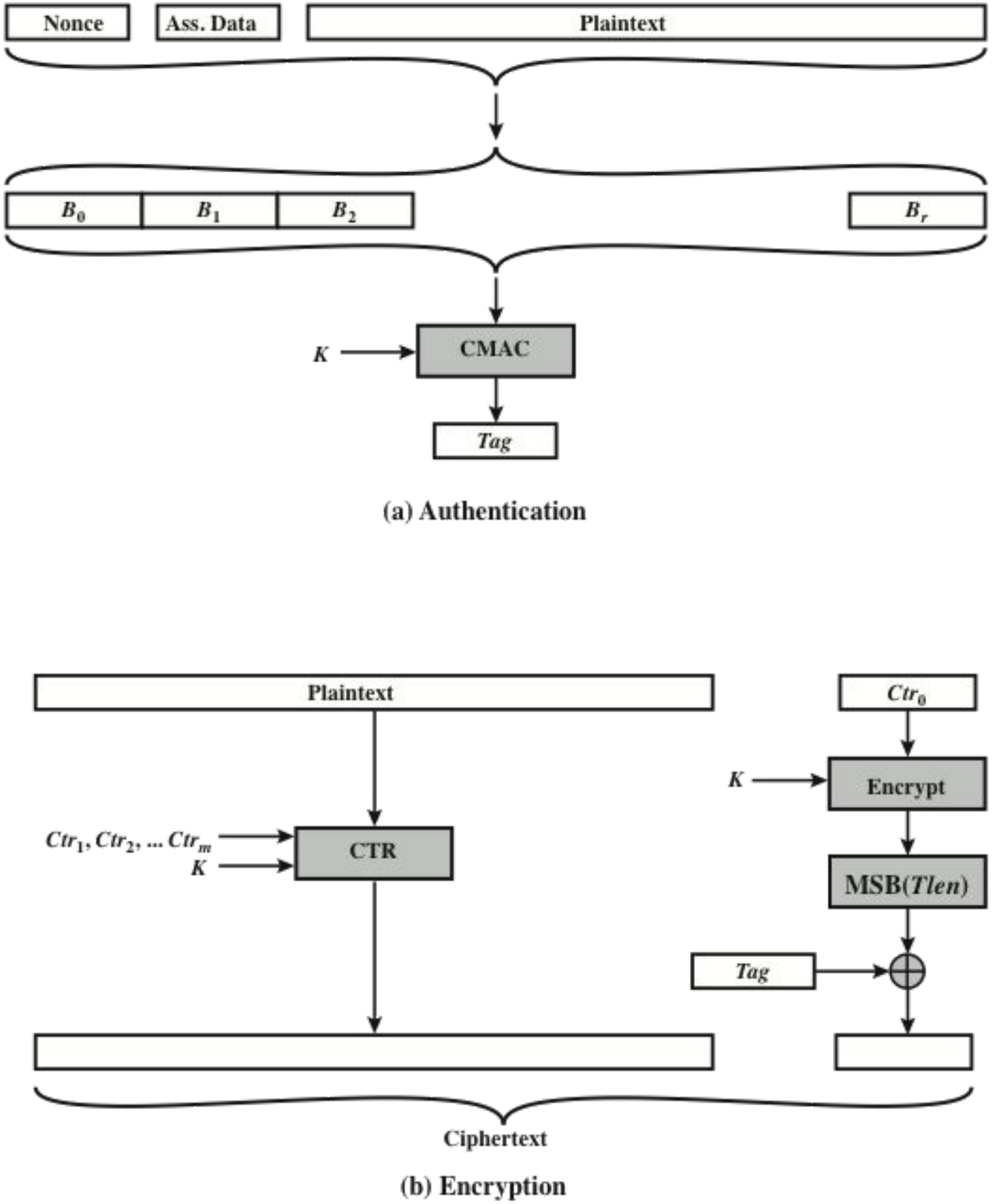


Figure 12.9 Counter with Cipher Block Chaining-Message Authentication Code (CCM)

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