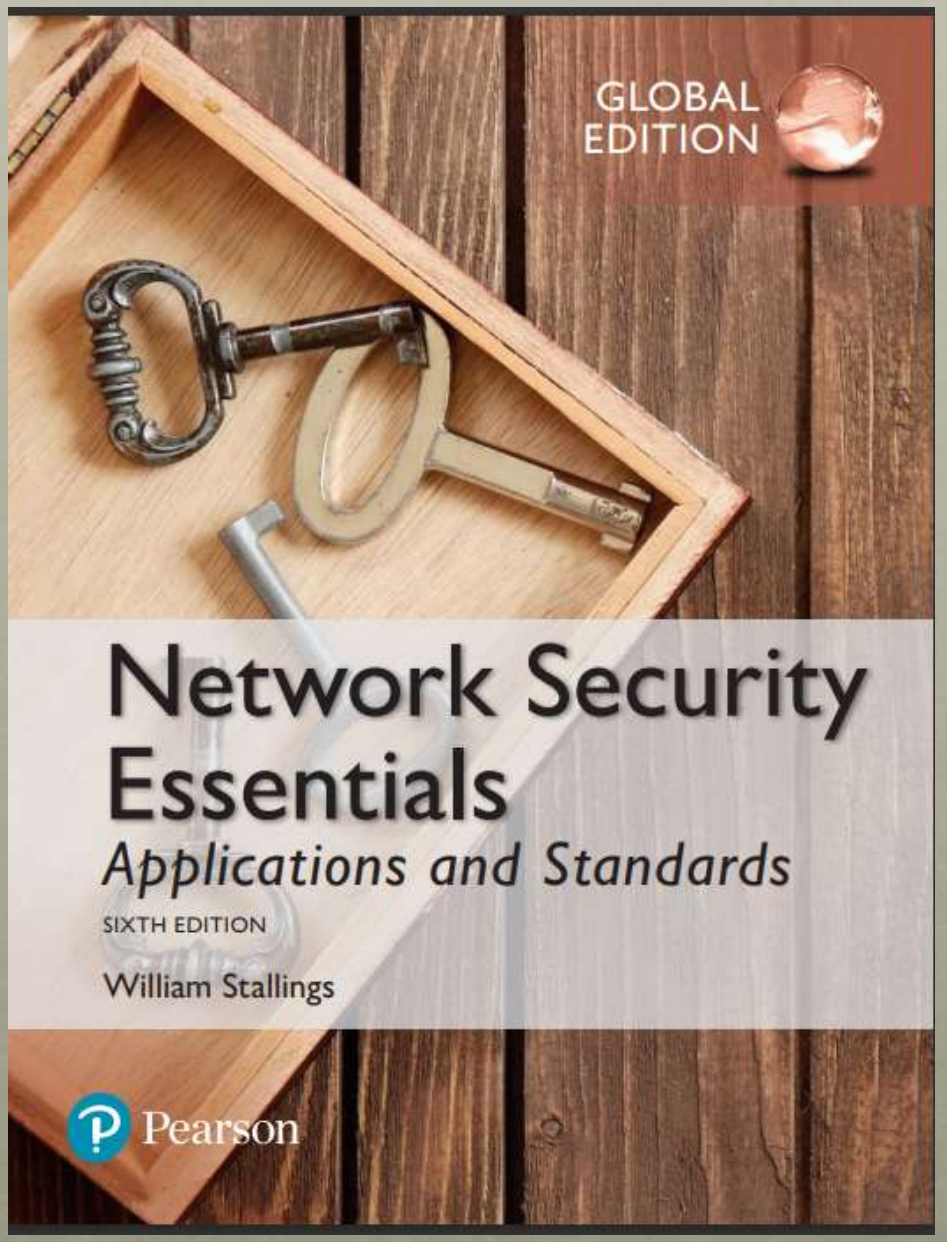
INFORMATION SECURITY

By Dr. Taghinezhad

Mail: a0taghinezhad@gmail.com



NETWORK SECURITY ESSENTIALS CHAPTER 3

by William Stallings

Lecture slides by Lawrie Brown

AUTHENTICATION USING CONVENTIONAL ENCRYPTION

- Is it possible to authenticate by the use of symmetric encryption?
  - If only the sender and receiver share a key
  - Only the genuine sender encrypt a message successfully for the other participant, and receiver can recognize a valid message.
- But, symmetric encryption alone is not a suitable tool for data authentication.
  - Ex, in the ECB mode of encryption, if an attacker reorders the blocks of ciphertext, then each block will still decrypt successfully. However, the reordering may alter the meaning of the overall data sequence.

Message Authentication Without Confidentiality

- When same message is broadcast to a number of destinations.
  - i.e., network unavailable notification
- An exchange in which one side has a heavy load and cannot afford the time to decrypt all incoming messages.
- The Authentication of a computer program in plaintext can be executed without having to decrypt it every time, which would be wasteful of processor resources.

MESSAGE AUTHENTICATION

- Message authentication is concerned with:
  - protecting the integrity of a message
  - validating identity of originator
  - non-repudiation of origin (dispute resolution)
- the three alternative for msg authentication func:
  - message encryption
  - message authentication code (MAC)
  - hash function

MESSAGE AUTHENTICATION CODE

- MAC: the use of **a secret key to generate a small block of data**, known as a message authentication code (MAC), that is **appended to the message**.
- Two communicating parties, say **A and B**, share a common secret key **K_{AB}**
- When A has a message to send to B, it calculates the message authentication code as a function of the message and the key: $MAC_M = F(K_{AB}, M)$. The message plus code are transmitted to the intended recipient.

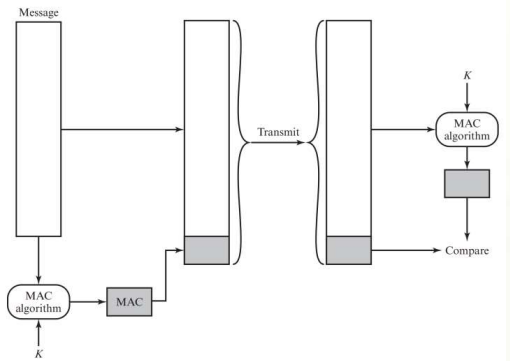


Figure 3.1 Message Authentication Using a Message Authentication Code

MESSAGE AUTHENTICATION CODE

If we assume that only the receiver and the sender know the identity of the secret key, and if the received code matches the calculated code, then the following statements apply:

1. The receiver is **assured** that the **message has not been altered**.

2. The receiver is **assured** that the **message is from the alleged sender**. Because no one else knows the secret key

3. If the message includes **a sequence number** (such as is used with HDLC and TCP), then the receiver can be assured of the proper sequence.

MESSAGE AUTHENTICATION CODE

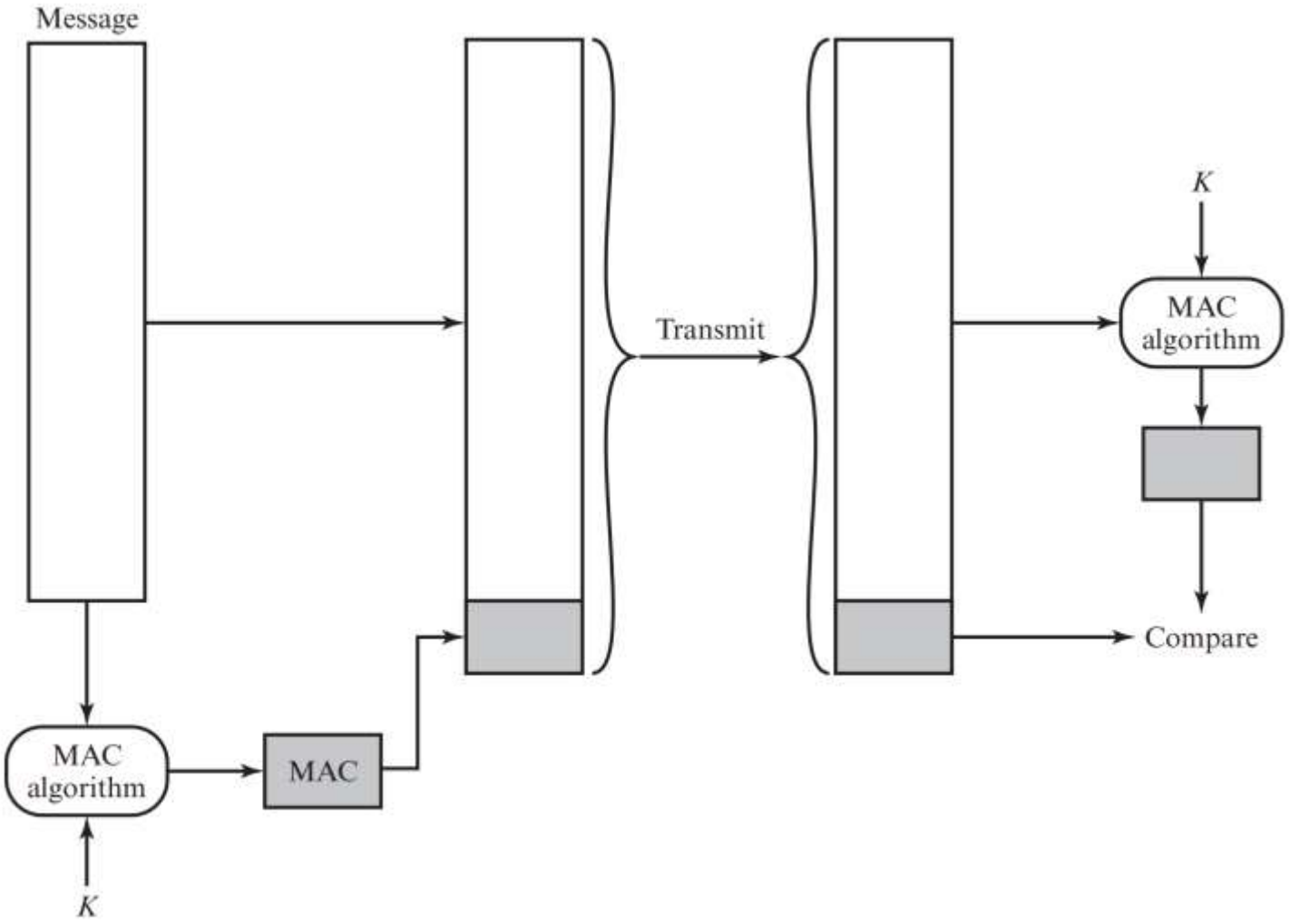


Figure 3.1 Message Authentication Using a Message Authentication Code

HASH FUNCTIONS

- condenses arbitrary message to fixed size

  `h = H(M)`

- usually assume **hash function is public**
- **hash used to detect changes** to message
- want a cryptographic hash function
  - computationally infeasible to find data mapping to specific hash (one-way property)
  - computationally infeasible to find two data to same hash (collision-free property)

TWO SIMPLE INSECURE HASH FUNCTIONS

- consider two simple insecure hash functions
- bit-by-bit exclusive-OR (XOR) of every block
  - $C_i = b_{i1}\ xor\ b_{i2}\ xor\ \ldots\ xor\ b_{im}$
  - a longitudinal redundancy check
  - reasonably effective as data integrity check
- one-bit circular shift on hash value
  - for each successive *n-bit* block
    - rotate current hash value to left by1bit and XOR block
  - good for data integrity but useless for security

SIMPLE XOR HASH FUNC

	bit 1	bit 2	• • •	bit *n*
Block 1	b_{11}	b_{21}		b_{n1}
Block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
Block *m*	b_{1m}	b_{2m}		b_{nm}
Hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR

ONE-WAY HASH FUNCTION

- An alternative to the message authentication code is the one-way hash function.
- A hash function **accepts** a **variable-size message M as input** and produces a **fixed-size message digest H(M)** as output.
- Unlike the MAC, a hash function does not take a secret key as input.
- To authenticate a message, the message digest is sent with the message in such a way that the message digest is authentic

ONE-WAY HASH FUNCTION

- Two approaches for one way hash function:
  - 1) The message digest can be encrypted using conventional encryption (part a); if it is assumed **that only the sender and receiver share the encryption key**, then **authenticity is assured**.
  - 2: The **message digest** can be encrypted using public-key encryption (part b); two advantages: (1) It provides a digital signature as well as message authentication. (2) It does not require the distribution of keys to communicating parties.

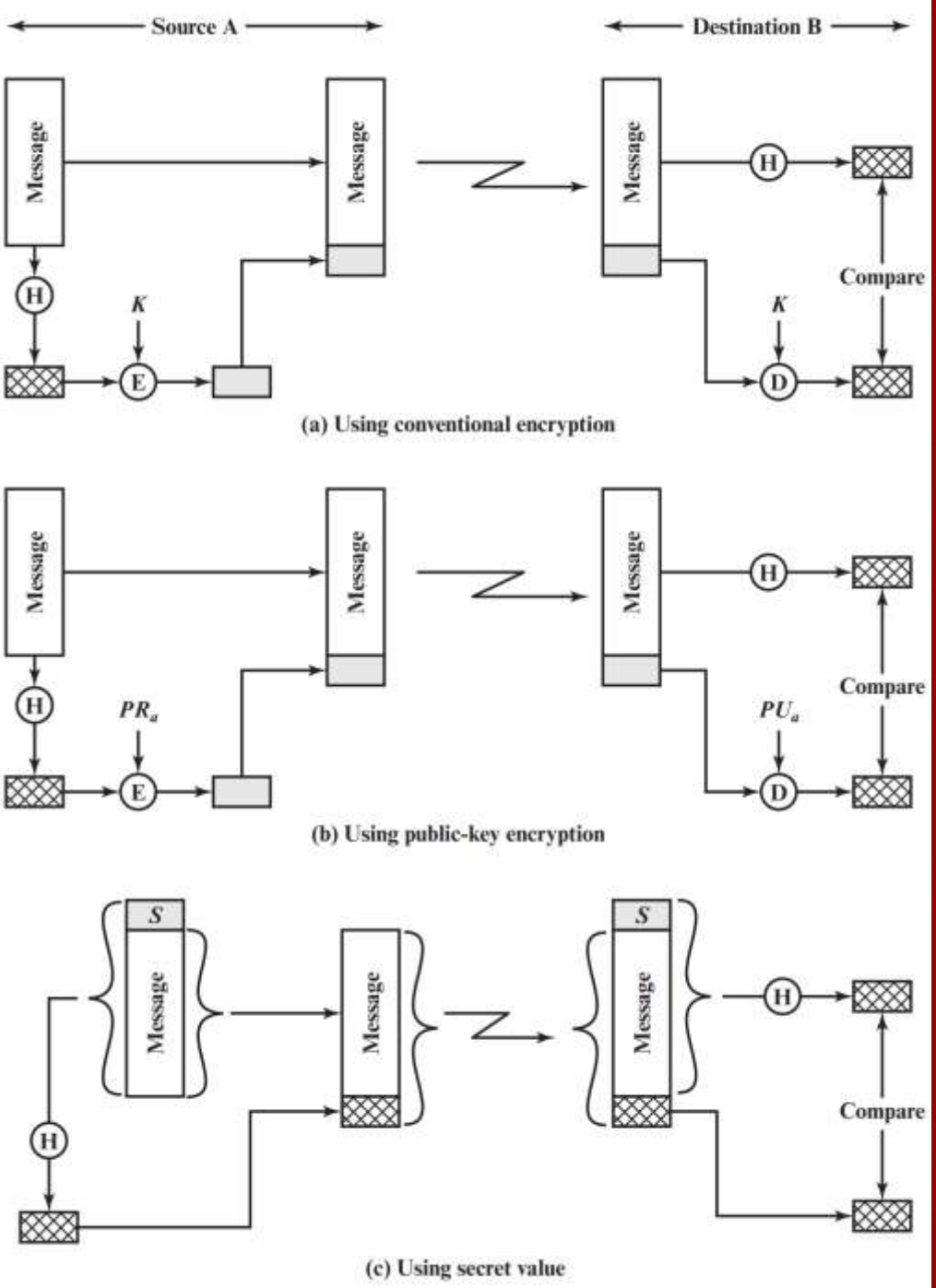


Figure 3.2 Message Authentication Using a One-Way Hash Function

THIRD TECHNIQUE

A and B, share **a common secret** value S_{AB}
When A has a message to send to B, it calculates the hash function over the concatenation of the secret value and the message:
$MD_M = H(S_{AB} || M)$
It then sends $[MD_M || M]$ to B.
Because B possesses S_{AB}, it can recompute $H(S_{AB} || M)$and verify MD_M

ONE-WAY HASH FUNC

Two approaches also have an advantage over approaches that encrypt the entire message in that less computation is required.

However, there are techniques that avoids encryption altogether. Several reasons for this interest are pointed out in:

- **a:Software is quite slow for Encryption,**
- **b:Encryption hardware costs are nonnegligible**
- **c:Encryption hardware** is optimized toward large data sizes while it has large overhead For small blocks of data,
- **d**:An encryption algorithm **may be protected by a patent** (need permission)

HASH FUNCTION REQUIREMENTS

For message, a **hash function (H)** must have the followings:

1. **H can be applied to a block of data** of **any size.**
2. **H produces a fixed-length** output.
3. **H(x) is relatively easy to compute** for any **given x,** making both hardware and software implementations practical.

HASH FUNCTION REQUIREMENTS (CONT.)

4. For any given **code h**, it is computationally **infeasible to find x such that H(x) = h**. (one-way or **preimage resistant**)
5. For any given block **x**, it is computationally infeasible to find **y ≠ x with H(y) = H(x)**. (**second pre-image resistant**) or weak collision resistant.
6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y), referred as: (collision resistant or strong collision resistant).

HASH FUNCTION REQUIREMENTS (CONT.)

- 4-th property (one-way): if the authentication technique involves the use of a secret value. The secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value.
- If **second pre-image resistant** property were not true, Attacker First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code n.
- The sixth property, collision resistant, protects against a sophisticated class of attack known as the **birthday attack**. Details are beyond the scope. The attack reduces the strength of an m-bit hash function from 2^m to $2^{m/2}$

ATTACKS ON HASH FUNCTIONS

- have **brute-force attacks** and cryptanalysis
  - For an *m-bit* hash value, the level of effort is proportional to 2^m, For Average 2^{m-1} values of y to find one that generates a given hash value h
- a preimage or second preimage attack
  - find y s.t. $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$
- hence value $2^{m/2}$ determines strength of hash code against brute-force attacks
  - 128-bits inadequate, 160-bits suspect

ATTACKS ON HASH FUNCTIONS

- brute-force attacks : does not depend on the specific algorithm but depends only on bit length of hash value
- Cryptanalysis: an attack **based on weaknesses in a particular cryptographic algorithm**. For a hash code of length n, the level of effort required is proportional to the following
  - Preimage resistant 2^n
  - Second preimage resistant 2^n
  - Collision resistant (birthday attack) $2^{n/2}$

SECURE HASH ALGORITHM->SHA

- SHA originally designed by NIST & NSA in 1993
- was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
- based on design of MD4 algorithm with key differences
- produces **160-bit hash values**
- recent 2005 results on security of SHA-1 have raised concerns on its use in future applications
- Security researchers have achieved the first real-world collision attack against the SHA-1 hash function at 2017, producing two different PDF files with the same SHA-1 signature1

REVISED SECURE HASH STANDARD

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

SHA APPLICATIONS

- **SHA-2** is used for[1]:
  - Digital certificates validation
  - Authentication
  - Password protection
  - Data integrity checks
  - It is implemented in some widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

SHA VERSIONS

Size are bits	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

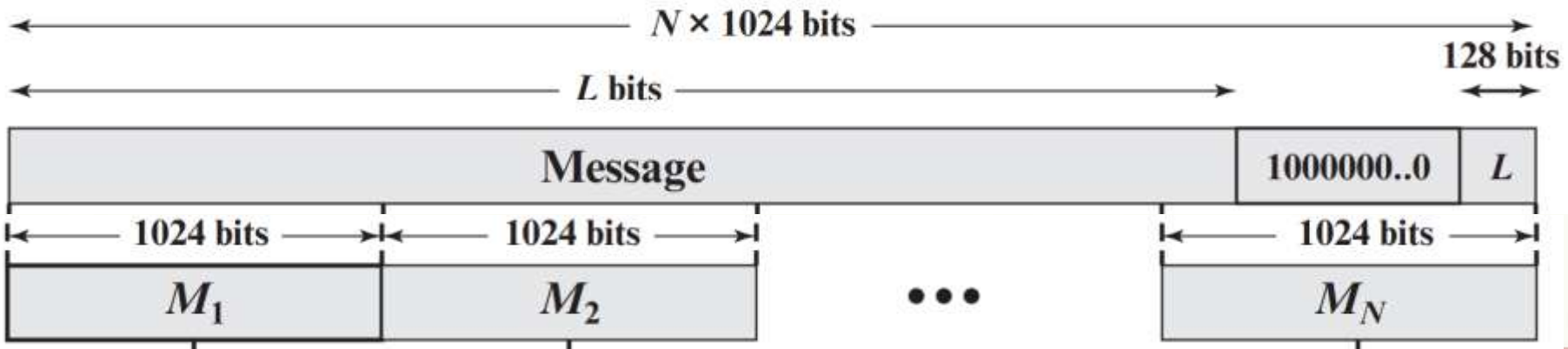
SHA-512

The algorithm takes as input a message with a maximum length of less than 2^{128} bits and produces as output a **512-bit** message digest.

- Plain text is processed by N blocks which each block is 1024 bits
- Number of rows and steps:80
- Each round=80
- In Each round,
  - Qword or W of 64 bits which is generated from plaintext
  - we use constant eighty k,
  - We use buffer to store intermediate results and output(hashcode)
- Each buffer size: 64bits, so we have (512/64=8) buffer.

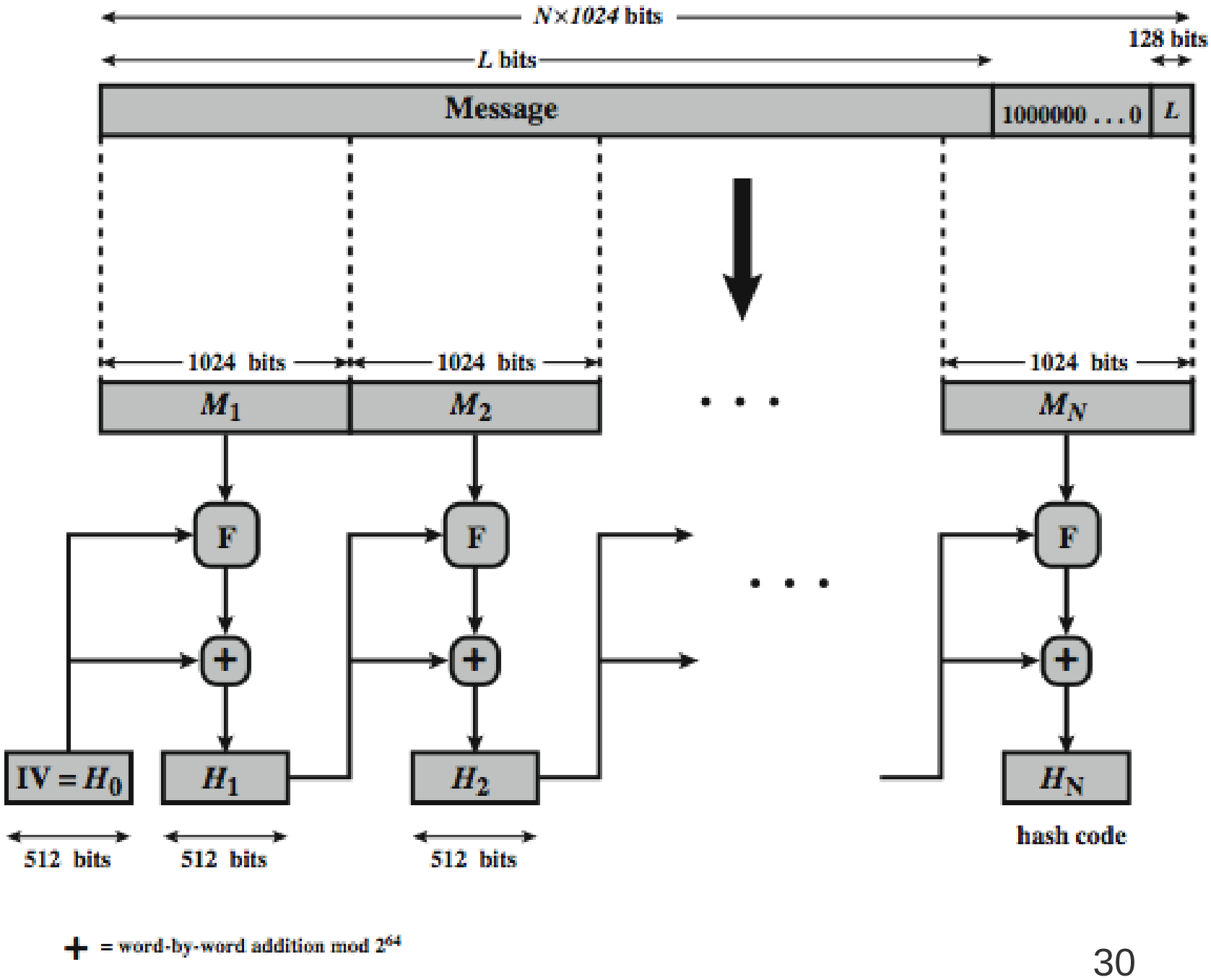
SHA-512

- Step 1: **Append padding bits**: The message is padded so that its length is congruent to 896 modulo 1024 [length = 896 (mod 1024)] or 1024-128=896 bits for the last block.
  - The padding consists of a single 1 bit followed by the necessary number of 0 bit
- Step 2: Append length: A block of 128 bits is appended to the message containing the length of the message



SHA-512

- Step 3: **Initialize hash buffer**: A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h).
  - These registers are initialized to the following 64-bit integers (hexadecimal values):
  - **a** = 6A09E667F3BCC908 **e** = 510E527FADE682D1
  - **b** = BB67AE8584CAA73B **f** = 9B05688C2B3E6C1F
  - **c** = 3C6EF372FE94F82B **g** = 1F83D9ABFB41BD6B
  - **d** = A54FF53A5F1D36F1 **h** = 5BE0CD19137E2179
- Step 4: **Process message (buffers) in 1024-bit (128-word) blocks in 80 rounds;**
  - **this module (algorithm)is labeled F in Figure**
- Step 5 Output of buffer is hash func


N×1024 bits
128 bits
L bits
Message
1000000 . . . 0
L
1024 bits
1024 bits
1024 bits
M1
M2
MN
F
F
F
+
+
+
IV = H0
H1
H2
HN
512 bits
512 bits
512 bits
hash code
+ = word-by-word addition mod 2^64

- Each words is 64 bits, which means 16 words for 1024 bit blocks. Rest of the words are aquired from these 16 words
- K_t are cube root of first 80 prime number.

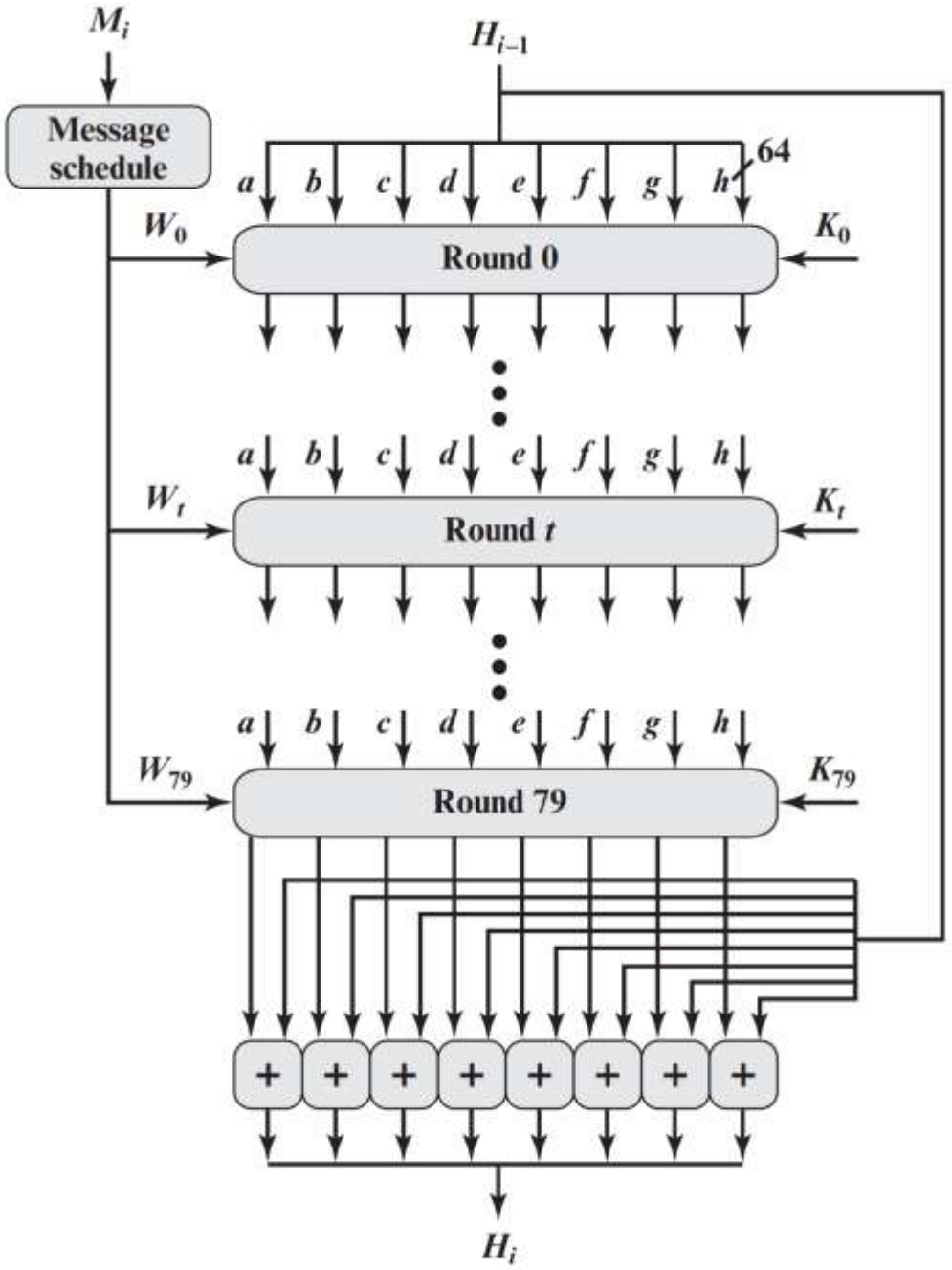
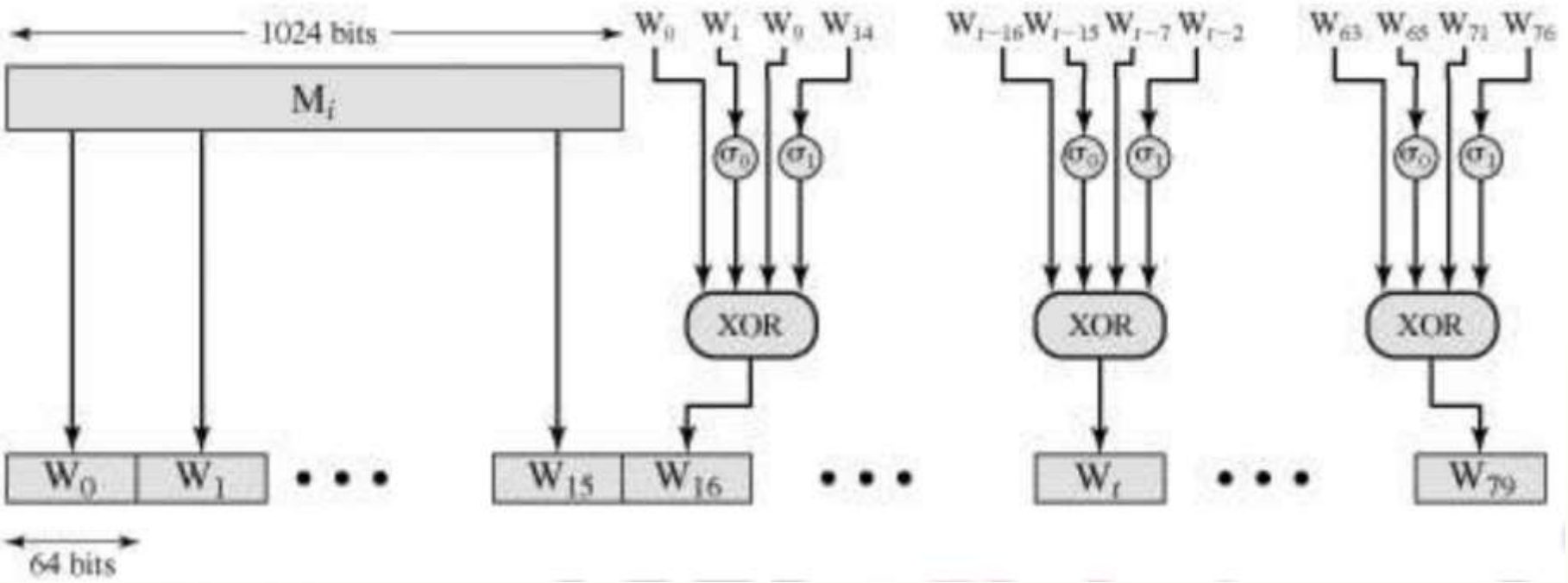


Figure 3.5 SHA-512 Processing of a Single 1024-Bit Block

SHA, WORD-SELECTION



KEYED HASH FUNCTIONS AS MACS

- want a MAC based on a hash function?
  - because hash functions are generally faster
  - crypto hash function code(library) is widely available
- hash includes a key along with message
- original proposal:

  `KeyedHash = Hash(Key|Message)`
  - some weaknesses were found with this , however eventually HMAC is proposed
- A hash function such as SHA was not designed for use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key.
- **HMAC** has been issued as RFC 2104, has been chosen as the mandatory to-implement MAC for IP Securiy.

PROBLEM OF CRYPTOGRAPY WITH MSG AUTHENTICATION

send Ali 200$	Encipher-	h15jxFKqraHIUeq4iEo5nA==

Now attacker receive the message and changes 5 to 4 in decipher

h15jxFKqraHIUeq4iEo4nA== ->Decipher-> send Ali 300$

However, there is still a chance that attacker change message and regenerate hash and send it.

PROBLEM OF CRYPTOGRAPY WITH MSG AUTHENTICATION

What if we use hash?

If we use hash as a checksum, we can understand it,

We better append a key between two party to the message inorder to prevent tempering, because if attacker alter the M, he can not recompute Hash. This is MAC

The problem of MAC is that, attacker can append to M, and recalculate hash if he can guess the length of the shared key.

HMAC Design Objectives

- use, without modifications, hash functions
- allow for easy replaceability of embedded hash function
- preserve original performance of hash function without significant degradation
- use and handle keys in a simple way.
- have well understood cryptographic analysis of authentication mechanism strength

HMAC

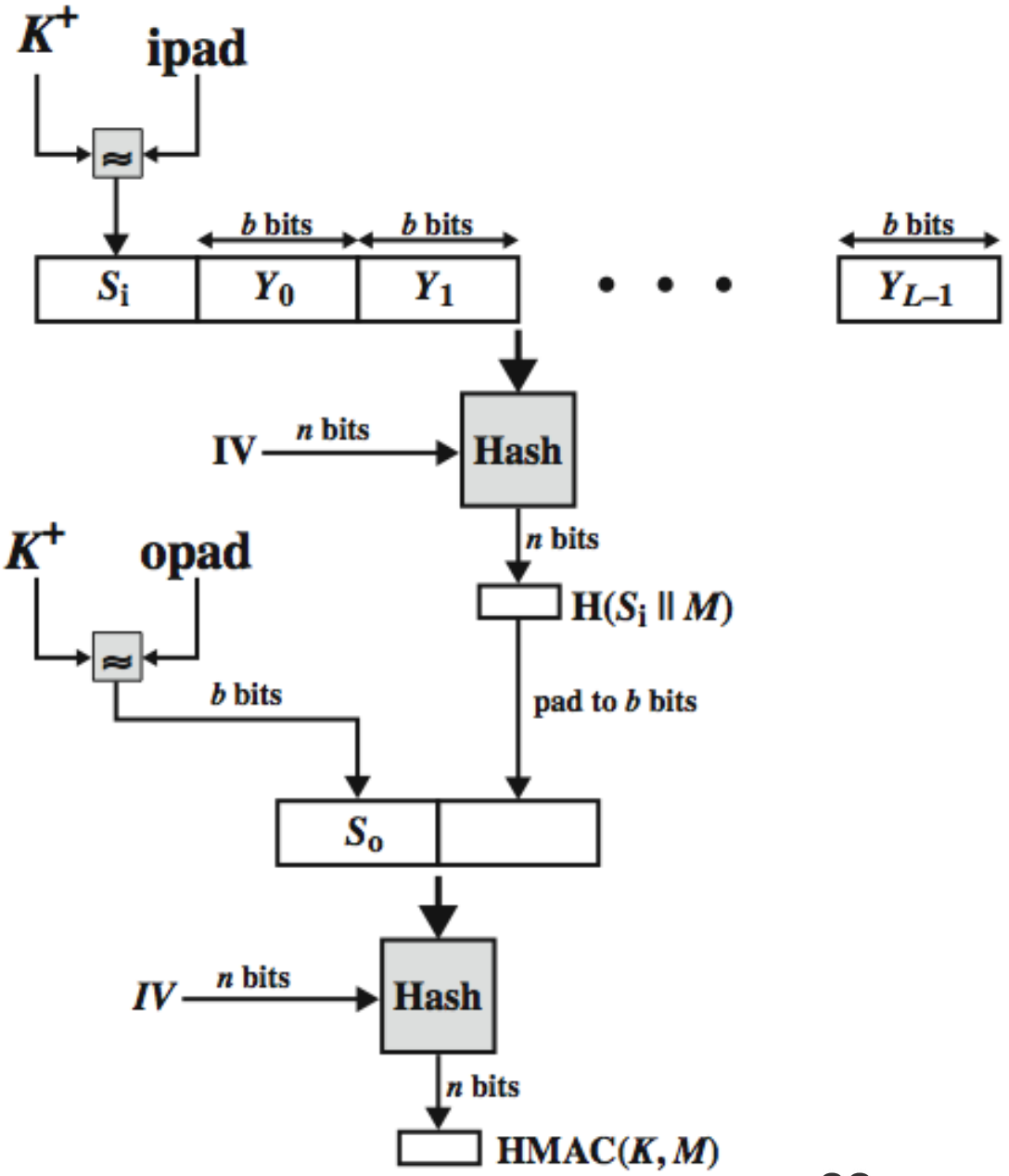
- specified as Internet standard RFC2104
- uses hash function on the message:

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

  - **where K^+ is the key padded out to size of msg, e(x.1024bit for SHA-256)**
  - **opad, ipad are specified padding constants**
- overhead is just 3 more hash calculations than the message needs alone
- any hash function can be used
  - eg. MD5, SHA-1, RIPEMD-160, Whirlpool

HMAC OVERVIEW

Message M consist of b bits, based on hash input, for SHA-3 it is 1024bit block.
K+ should be b bits, which expended from key K by appending zeros to the end of K.
ipad = 00110110 (36 in hexadecimal) repeated b/8 times
opad = 01011100 (5C in hexadecimal) repeated b/8 times



HMAC SECURITY

- proved security of HMAC relates to that of the underlying hash algorithm
- attacking HMAC requires either:
  - brute force attack on key used
  - birthday attack (but since keyed would need to observe a very large number of messages)
- choose hash function used based on speed verses security constraints
- HMAC can be formulated by:
- $HMAC(K, M) = H[(K^+ \oplus opad) || H[(K^+ \oplus ipad) ||M]]$

PRIVATE-KEY CRYPTOGRAPHY

- traditional **private/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed communications are compromised
- also is **symmetric**, parties are equal
- does not protect sender from receiver forging a message & claiming is sent by sender

PUBLIC-KEY CRYPTOGRAPHY

- probably most significant advance in the 3000 year history of cryptography
- uses **two** keys – a public & a private key
- **asymmetric** since parties are **not** equal
- uses clever application of number theoretic concepts to function
- complements **rather than** replaces private key crypto

MISCONCEPTION ABOUT PUBLIC-KEY

- public-key encryption ~~is~~ more secure from cryptanalysis than conventional encryption:
  - Not true: security of any encryption scheme depends on
    - (1) the length of the key
    - (2) the computational work involved in breaking a cipher.
- public-key encryption ~~is~~ a general-purpose technique that has made conventional encryption obsolete.
  - Not true: **due to the computational overhead of current public-key** encryption schemes, no likelihood for abandoning conventional encryption

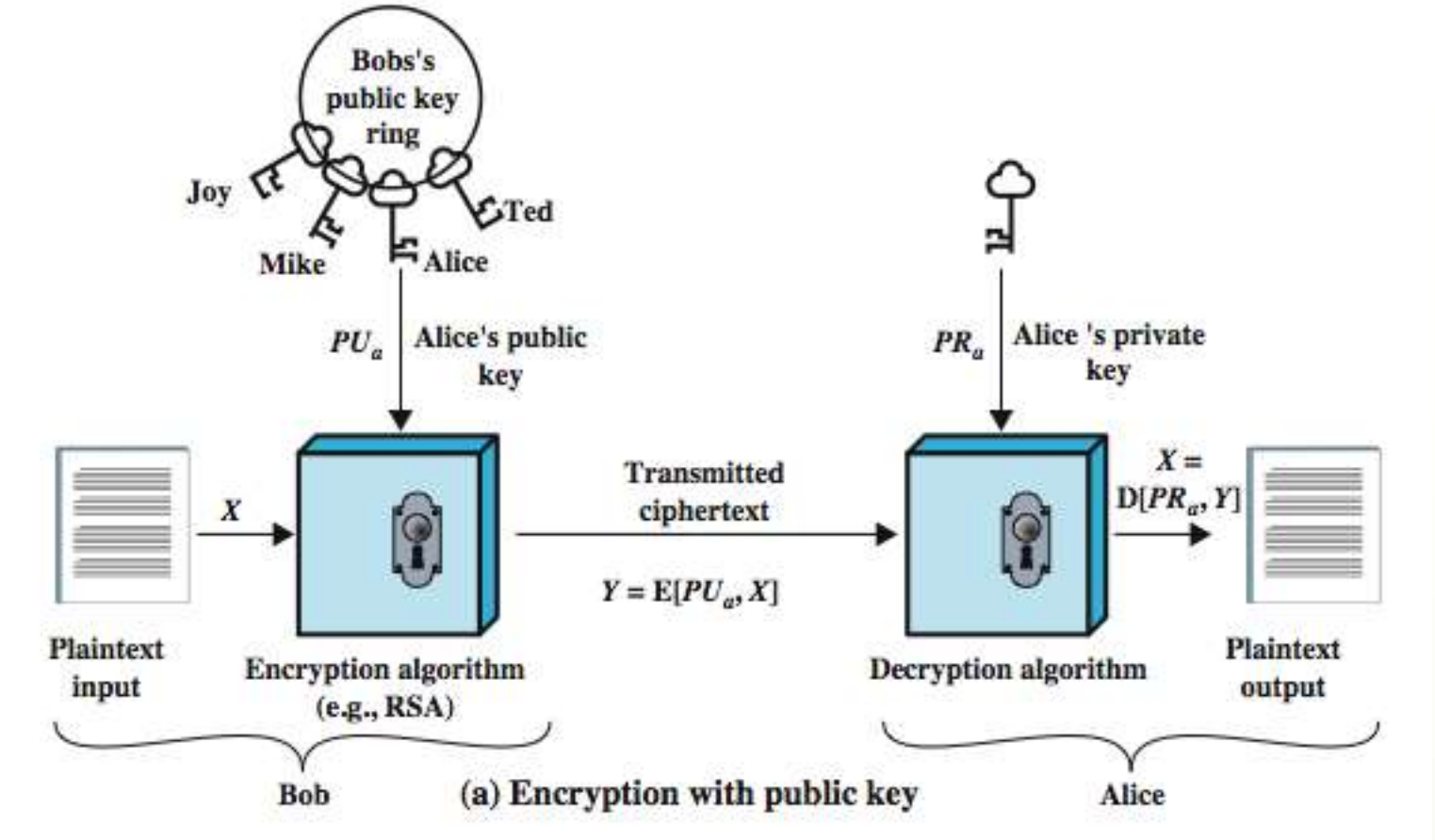
WHY PUBLIC-KEY CRYPTOGRAPHY?

- developed to **address two key issues**:
  - **key distribution** – how to have secure communications in general without having to trust a KDC(key distribution center) with your key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
  - known earlier in classified community
  - NSA know it in 60's

PUBLIC-KEY CRYPTOGRAPHY

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **infeasible to determine private key from public**
- is **asymmetric** because
  - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

PUBLIC-KEY CRYPTOGRAPHY



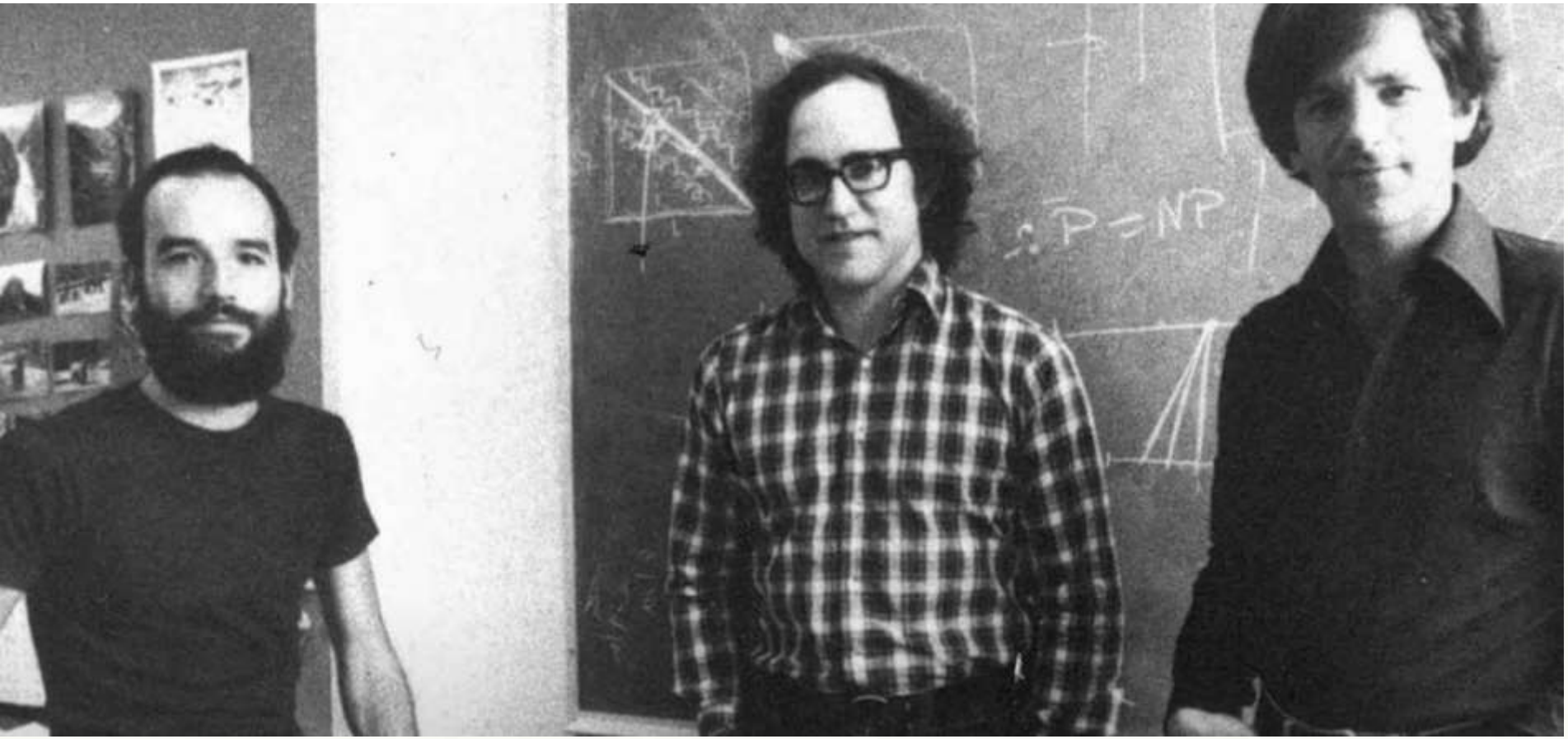
SYMMETRIC VS PUBLIC-KEY

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- based on exponentiation in a finite (Galois) field over integers modulo a prime
  - nb. exponentiation takes $O((\log n)^3)$ operations (easy)
- RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and n - 1 for some n
- security due to cost of factoring large numbers
  - nb. factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA



Rivest (middle), Shamir (left) & Adleman (Right)

RSA EN/DECRYPTION

- to encrypt a message M the sender:
  - obtains **public key** of recipient **PU**={**e**,**n**}
  - computes: $C = M^e \bmod n$, where $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses their private key PR={**d**,**n**}
  - computes: $M = C^d \bmod n$
- note that the message M must be smaller than the modulus n (block if needed)

RSA EN/DECRYPTION

- PU={**e,n**} = (5,14)
- PlainText: B -> ConvertToNumber-> 2
- $C = M^e \bmod n$ -> $C= 2^5 \bmod 14$
- Cipher=4 ->ConvertText-> D
- PR={**d,n**}=(11,14)
- Decipher= $M = C^d \bmod n$ -> $4^{11} \bmod 14=2$
- So 2 is the Same as B the message.

RSA LIMITATION

- RSA limit: Message size must be less or equal to our key size which is only 256 bytes
- So for bigger data,
  - we encrypt AES which has 256bits or 512bits key size using RSA.
  - We send AES key using RSA.
  - We send our message body Decrypted by AES.

RSA KEY SETUP

- each user generates a public/private key pair by:
- 1) selecting two large primes at random: `p, q` (i..e:17,11)
- computing their system modulus `n=p.q`(i.e.,:187)
  - note `ø(n)=(p-1)(q-1)` selecting at random the encryption key `e`
  - where `1<e<ø(n), gcd(e,ø(n))=1 (i.e., e=7)`
- solve following equation to find decryption key `d`
  - `e.d mod 160 =1 and 0≤d≤n (i.e., d=23->bcz 7*23=161)`
- publish their public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

RSA EXAMPLE - KEY SETUP

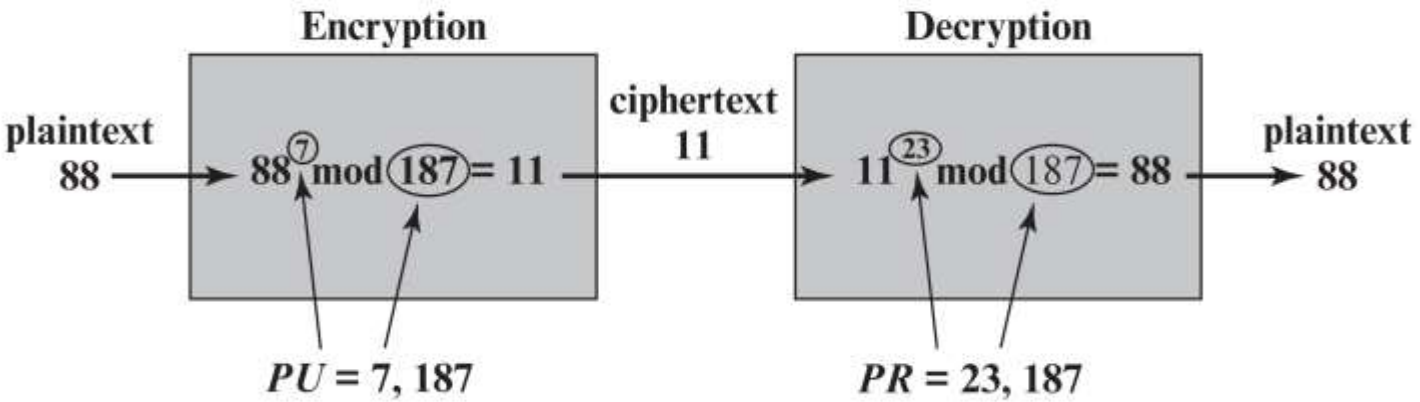
1. Select primes: $p=17$ & $q=11$
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $d.e \bmod 160 = 1$ and $d < 160$
   Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $PU=\{7,187\}$
7. Keep secret private key $PR=\{23,187\}$

RSA EXAMPLE - EN/DECRYPTION

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $M<n \Rightarrow 88<187$)
- encryption:

  $C = 88^7 \bmod 187 = 11$
- decryption:

  $M = 11^{23} \bmod 187 = 88$



DIFFIE-HELLMAN KEY EXCHANGE

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

STEPS

- all users agree on global parameters (a and q):
  - large prime integer or polynomial `q`
  - `a` being a primitive root mod `q`

- عدد a باید ریشه اولیه به هنگ q باشد.
- هم نهشتی اعداد (هم باقی‌مانده بودن)

12 mod 5=2
27 mod 5=2

$27 \equiv 12 \pmod 5$

۲۷ با ۱۲ به هنگ ۶ هم‌نهشت است

پیدا کردن ریشه اولیه

- ریشه اولیه a به روی qیعنی
- $a^i \ mod \ q, \ (1 \leq i < q)$ باید تمامی خروجی ها برای این عبارت متفاوت باشند
- $a^1, a^2, a^3 \ldots \ a^{q-1}$

$p=7$

$a^i \bmod 7$

a^1	a^2	a^3	a^4	a^5	a^6	...
1	1	1	1	1	1	×
2	4	1	2	4	1	×
3	2	6	4	5	1	✓
4	2	1	4	2	1	×
5	4	6	2	3	1	✓
6	1	6	1	6	1	×

- نحوه پیدا کردن ریشه اول a به هنگ ۷

DIFFIE-HELLMAN SETUP

- all users agree on global parameters (a and q):
  - large prime integer or polynomial q
  - a being a primitive root mod q
- each user (eg. A) generates their key: x
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $\mathbf{y_A} = a^{x_A} \bmod q$
- each user makes public that key y_A

مثال عددی پروتکل

1. دو طرف روی مقدار عدد اول ۲۳ =qو مقدار اولیهٔ ۵ =aتوافق می‌کنند.
2. طرف اول مقدار پنهانی ۶ =Xaرا انتخاب و ($a^{Xa} \bmod q$) را برای طرف دوم ارسال می‌کند که آن را A فرض می‌کنیم.
   - $۵^{۶} \bmod ۲۳ = ۸$
3. طرف دوم مقدار پنهانی ۱۵ =Xb را انتخاب و ($a^{Xb} \bmod q$) را برای طرف اول ارسال می‌کند که آن را B می‌نامیم
   - $۵^{۱۵} \bmod ۲۳ = ۱۹$
4. طرف اول مقدار $(B)^{x_A} \bmod q$ را محاسبه کرده و به عنوان کلید رمز مشترک در نظر می‌گیرد.
   - $۱۹^{۶} \bmod ۲۳ = ۲$
5. طرف دوم مقدار $(A)^{x_B} \bmod q$ را محاسبه کرده و به عنوان کلید رمز مشترک در نظر می‌گیرد.
   - $۸^{۱۵} \bmod ۲۳ = ۲$

- میبینیم هر دو طرف رمز مشترک را یک عدد به دست آورده اند. در حقیقت رمز مشترک از فرمول زیر به دست می‌آید. $a^{x_A.x_B} \bmod q$

DIFFIE-HELLMAN KEY EXCHANGE

- shared session key for users A & B is K_{AB}:

  $K_{AB} = a^{x_A . x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (which **B** can compute)

  $= y_B^{x_A} \bmod q$ (which **A** can compute)

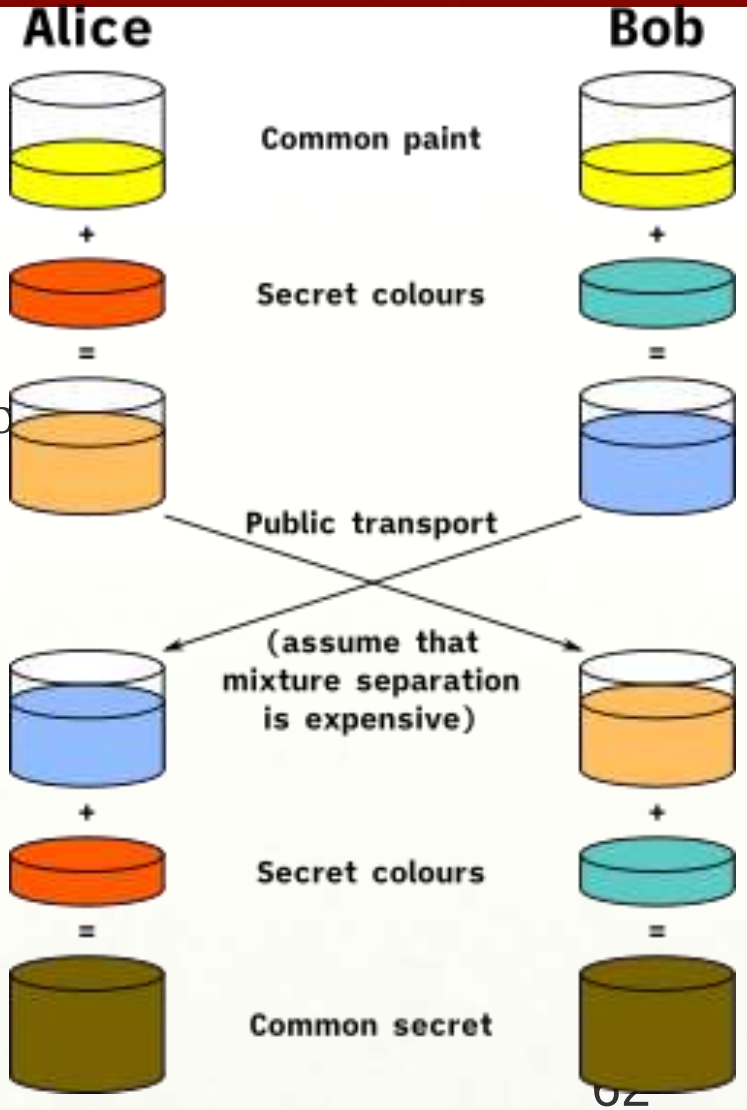
- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x, must solve discrete log

DIFFIE-HELLMAN EXAMPLE

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $a=3$
- select random secret keys:
  - A chooses $x_A=97$, B chooses $x_B=233$
- compute respective public keys:
  - $y_A=3^{97} \bmod 353 = 40$ (Alice)
  - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:
  - $K_{AB}= y_B^{x_A} \bmod 353 = 248^{97} = 160$ (Alice)
  - $K_{AB}= y_A^{x_B} \bmod 353 = 40^{233} = 160$ (Bob)

DIFEI-HELLMAN EXAMPLE ON COLOR

- Alic and Bob agree on a common paint
- Each add its secret color
- They exchange their output color
- They add each secret color to the received co
- The result is the shared color.

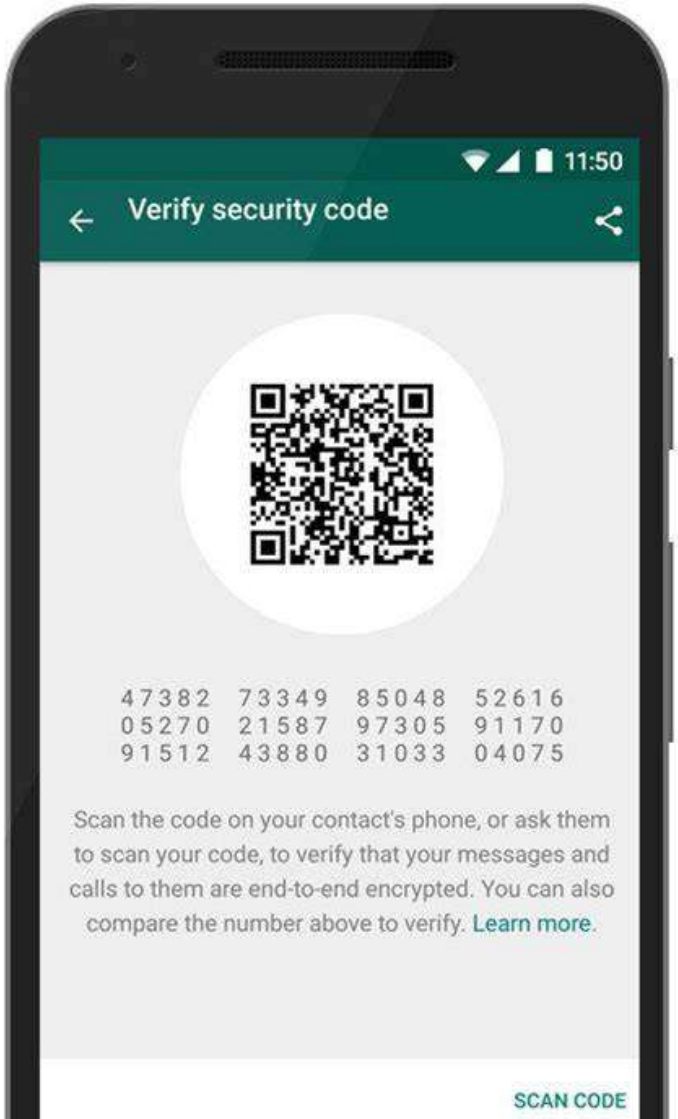


KEY EXCHANGE PROTOCOLS

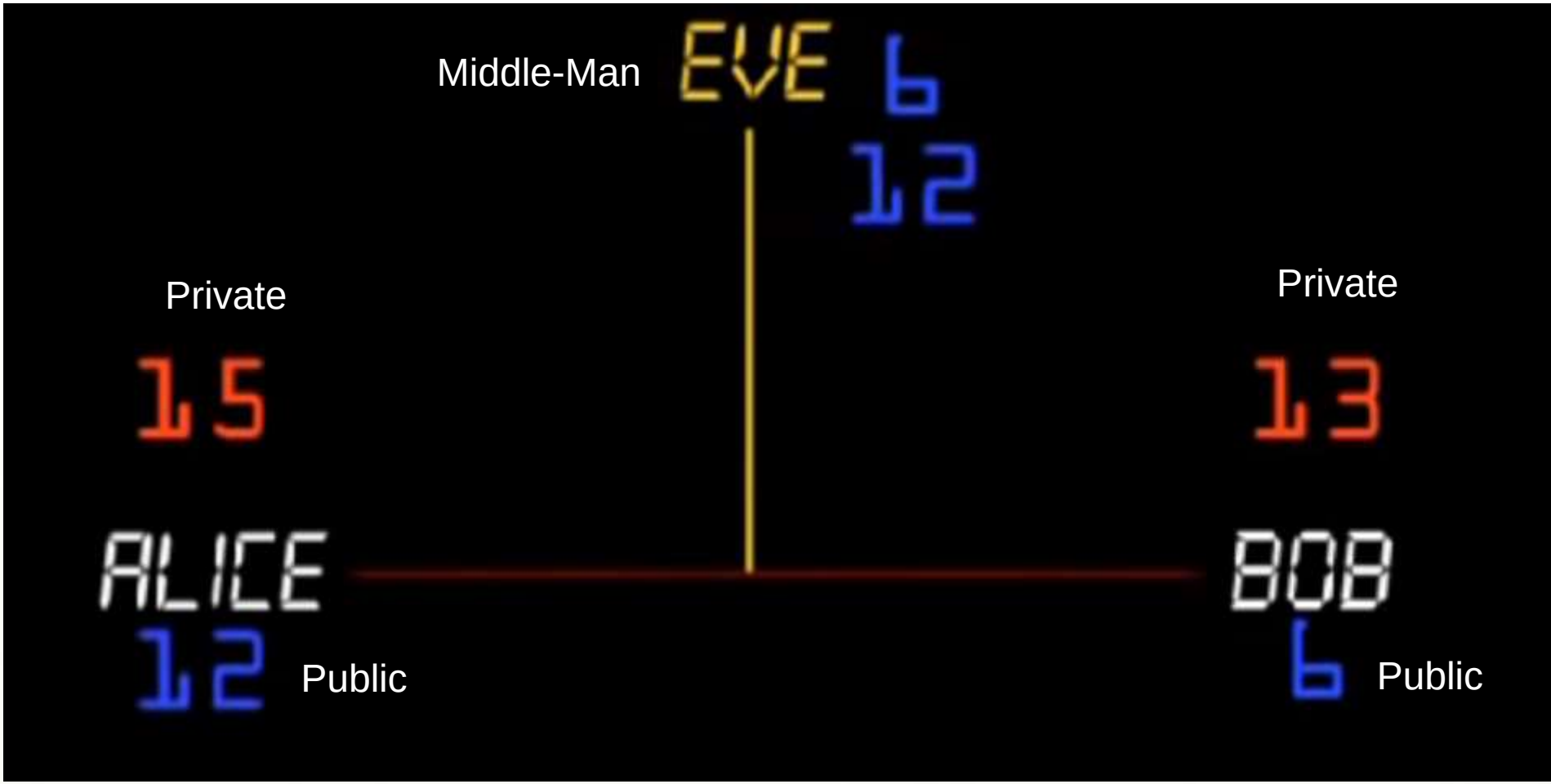
- users could create random private/public D-H keys each time they communicate
- users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
- both of these are vulnerable to a meet-in-the-Middle Attack
- authentication of the keys is needed

DIFEI-HELLMAN

- Applications:
  - TLS protocol
  - Whatsapp in every chat
  - Telegram on secret chat
  - Signal



ATTACK IS NOT POSSIBLE



Eve can not calculate any of the parties private key.

EFFECTIVE:MAN-IN-THE-MIDDLE ATTACK

1. Eve (Middle man) prepares two private / public keys random private keys X_{D1} and X_{D2}, and then computing the public keys Y_{D1} and Y_{D2}.

2. Alice transmits her public key Y_A to Bob

3. But, Eve intercepts Y_A and instead transmits his first public key Y_{D1} to Bob. Eve also calculates a shared key with Alice $K2 = (Y_A)^{X\_B} \bmod q$

4. Bob receives the public key Y_{D1} and calculates the shared key (with Eve instead of Alice) $K1 = (Y_A)^{X\_B} \bmod q$

5. Bob transmits his public key Y_B to Alice

6. Eve intercepts Y_B and transmits his second public key Y_{D2} to Alice. Eve calculates a shared key with Bob $K1 = (Y_B)^{X\_B} \bmod q$

7. Alice receives the key Y_{D2} and calculates the shared key (with Eve instead of Bob) $K1 = (Y_A)^{X\_A} \bmod q$

- Eve can then intercept, decrypt, re-encrypt, forward all messages between Alice & Bob

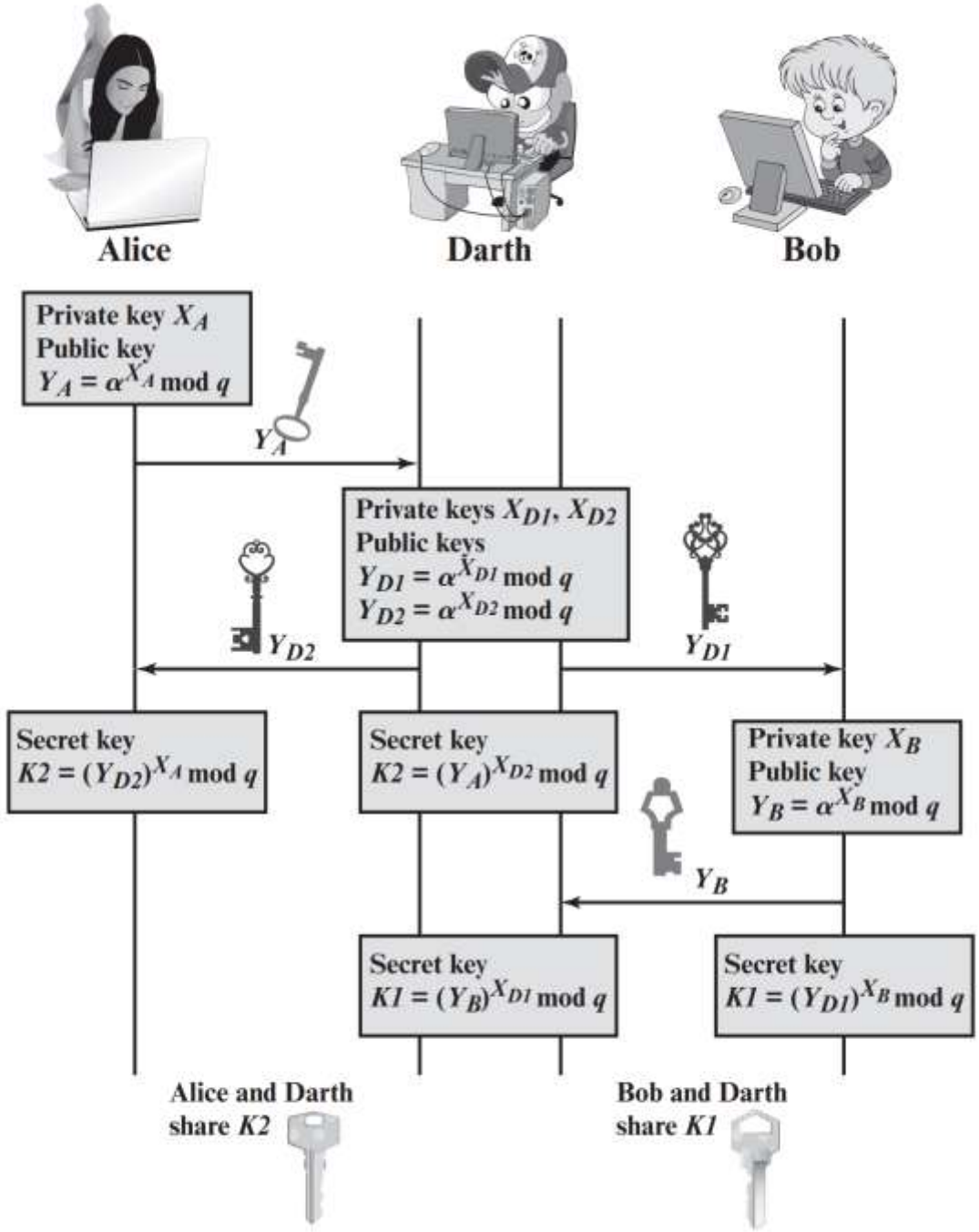


Figure 3.14 Man-in-the-Middle Attack

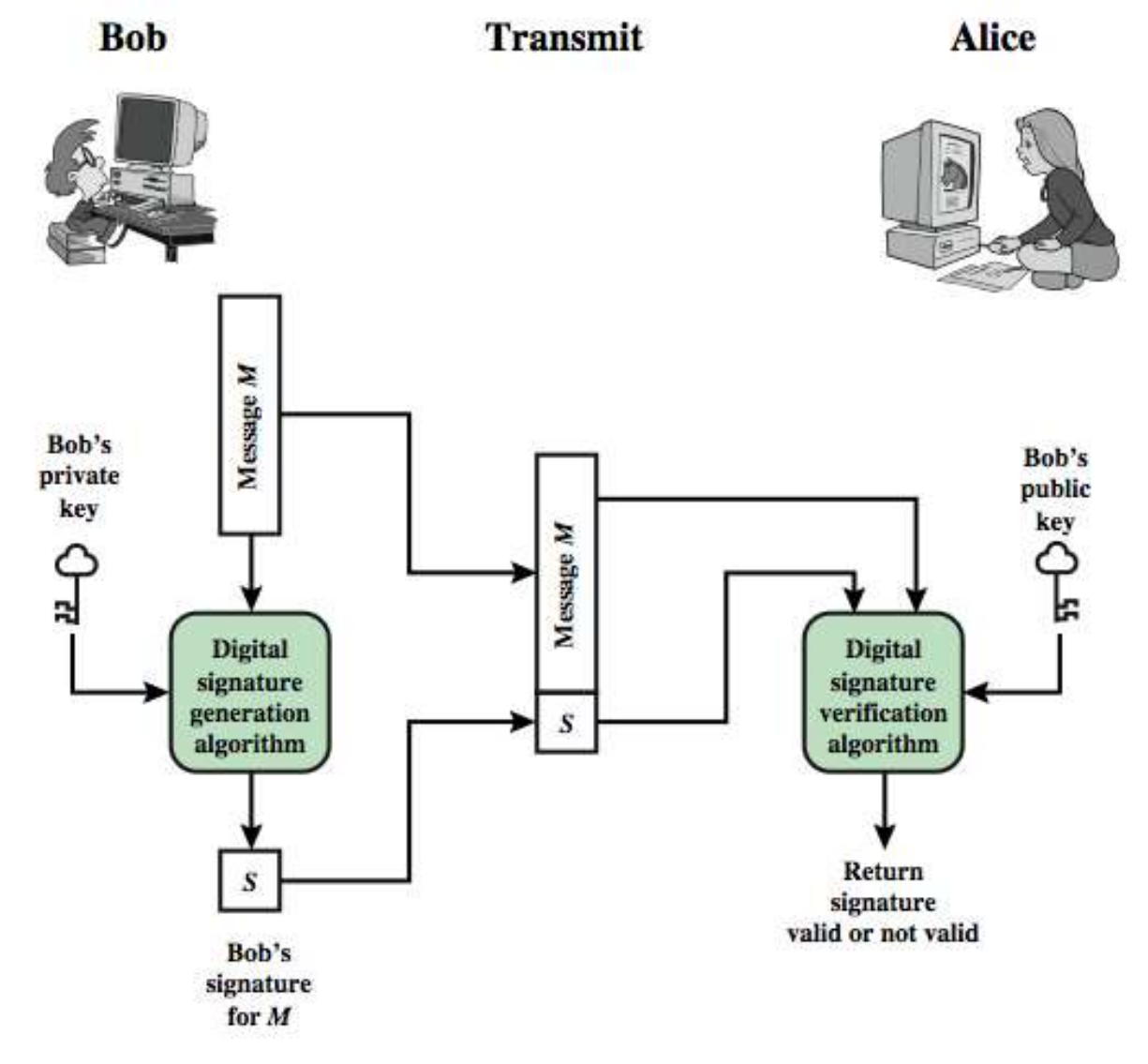
DIGITAL SIGNATURES

- have looked at message authentication
  - but does not address issues of lack of trust
- digital signatures provide the ability to:
  - verify author, date & time of signature
  - authenticate message contents
  - be verified by third parties to resolve disputes
- hence include authentication function with additional capabilities

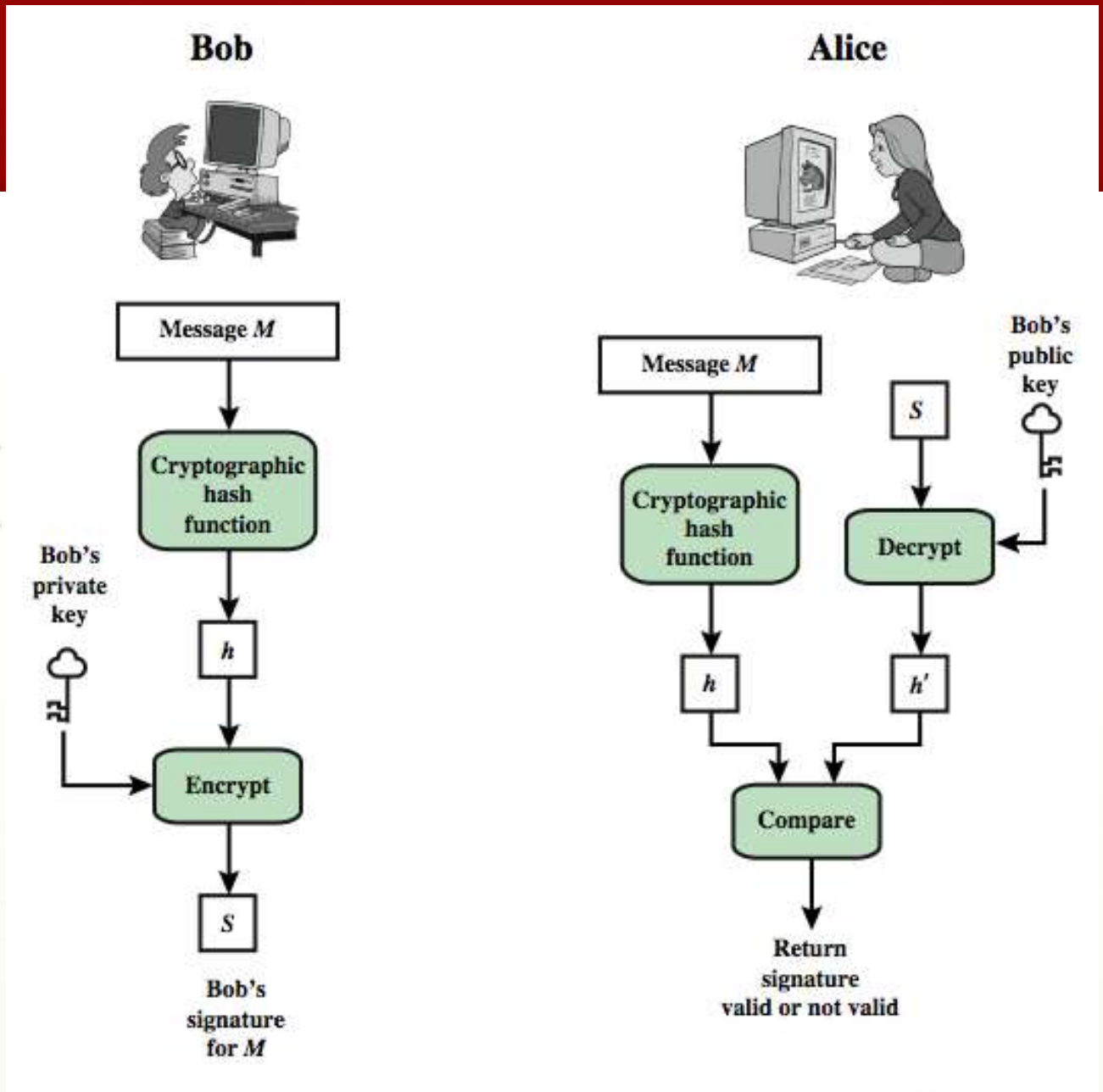
DIGITAL SIGNATURES

- Suppose that Bob wants to send a message to Alice.
- the message can or not be kept as a secret,
- Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message. S=digital_signature_algorithm(hash value+ Bob's private key)
- Bob sends => M (message) + S (signature).
- When Alice receives M (message) + S (signature).
- she (1) calculates a hash value for the message and (2) provides the hash value and Bob's public key as inputs to a digital signature verification algorithm.
- If the algorithm verifies that it is communing from the valid auther.

DIGITAL SIGNATURE MODEL



DIGITAL SIGNATURE MODEL



Chapter 9 – Public Key Cryptography and RSA

Every Egyptian received two names, which were known respectively as the true name and the good name, or the great name and the little name; and while the good or little name was made public, the true or great name appears to have been carefully concealed.

—***The Golden Bough*, Sir James George Frazer**