Public Key Cryptography & Message Authentication

Approaches to Message Authentication

Encryption or confidentiality protects against passive attacks such as eavesdropping, and release of massage contents. A different requirement is to protect against active attacks such as Masquerade, replay, and modification of massage contents. The protection against active attacks known as message authentication. Message authentication is a procedure that allows communicating parties to verify that received messages are authentic. And authentic means when the document or file is genuine and comes from its claimed source. The two important aspects are to (1) verify that the contents of the message have not been altered or modified and (2) that the source is authentic. Message authentication work under integrity in CIA triad.

We have 2 approaches to message authentication as follows:

1. Using conventional encryption: If we use symmetric encryption alone for data authentication, then we assume that:
   a. Only sender and receiver share a key
   b. The message includes error detection code and sequence number, so then the receiver will be sure that the message does not been modified
   c. The message includes a timestamp, so then the receiver is sure that the message does not delay in transit
2. Without message encryption: There are 3 approaches by not using encryption for data authentication as follows:
   a. Append an authentication tag to a message
   b. Message read independent of authentication function

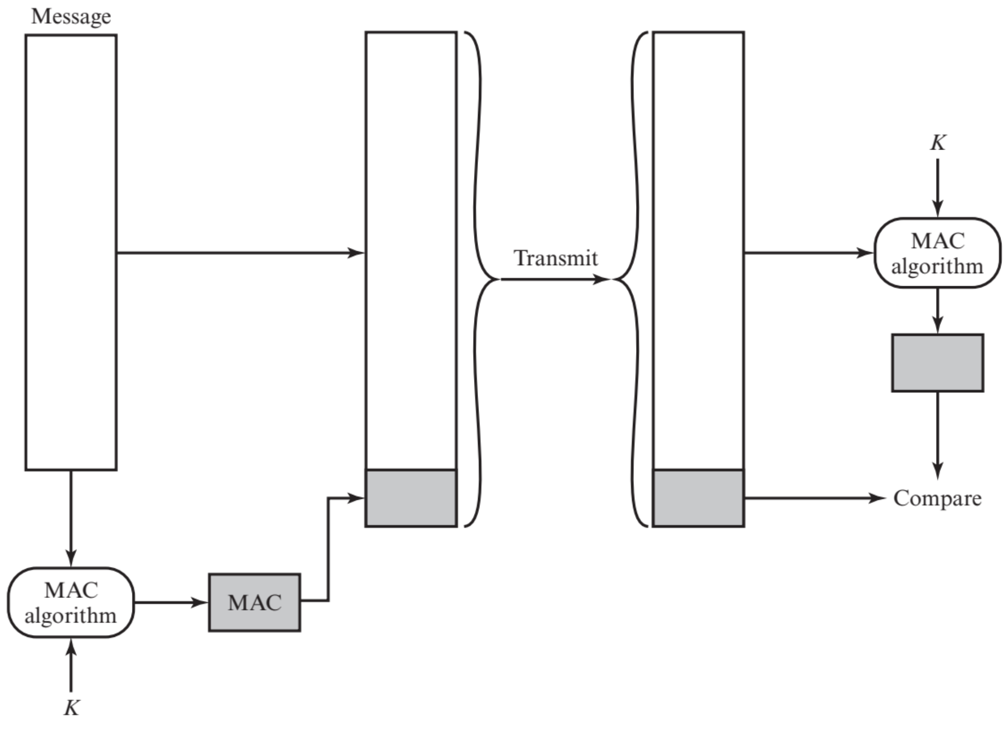
c. No message confidentiality

There are some applications in which message authentication without confidentiality or encryption is preferable:

- Application that broadcasts a message that only one destination needs to monitor for authentication
- Too heavy a load to decrypt then we use random authentication checking for messages
- Computer executables and files that checked when assurance required

Message Authentication Code (MAC)

MAC use a secret key to generate a small block of data that is appended to the message. Assume A and B share a common secret key K_{AB}, then $MAC_M = F(K_{AB}, M)$. The receiver performs the same calculation on the received message using the secret key to generate a new MAC. Then, the received code compared to the calculated code. The figure below shows message authentication using a MAC.



If only the sender and receiver know the secret key, and if the received code matches the calculated code, then:

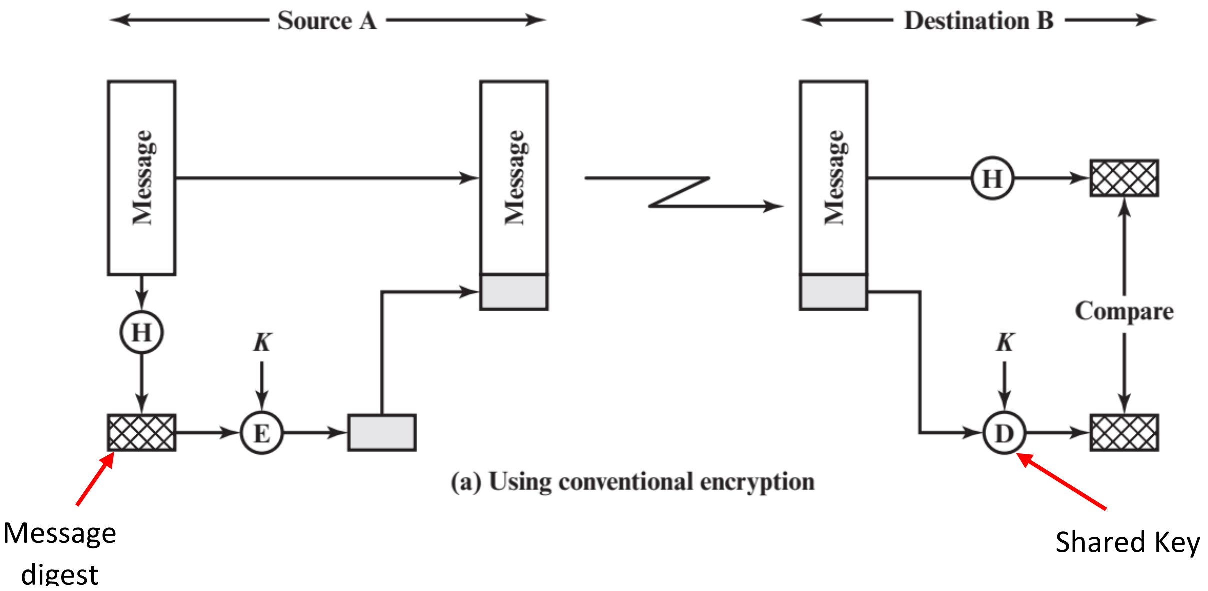
- No modification: Receiver assured that message is not altered
- No masquerading: Receiver assured that the message is from the claimed sender
- No replay: It include a sequence number, assured proper sequence

MAC uses DES algorithm that is recommended by NIST. DES is used to generate an encrypted version of the message and the last number of bits of ciphertext are used as the code. The authentication algorithm need not be reversible. Also, it is less vulnerable to being broken than encryption it means that it stands up to attack.

One-Way Hash Function

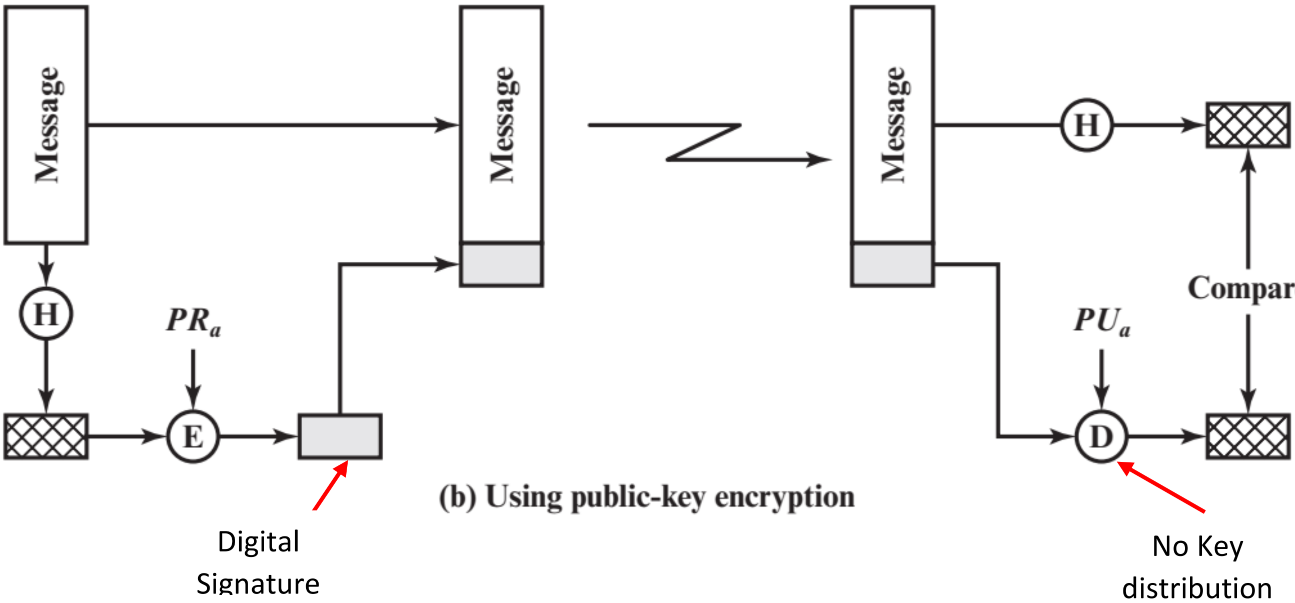
A Hash function accepts a variable size message M as input and produces a fixed-size message digest $H(M)$ as output. A hash function does not use secret key as input. The message digest is sent with the message for authentication. One-way function has 3 ways in which the message can be authenticated as follows:

1. Using conventional encryption: Authenticity is assured, because we assume only sender and receiver knows the secret key.



(a) Using conventional encryption

2. Using public-key encryption: less computation since message does not have to be encrypted. Also, it provides a digital signature as well as message authentication, and it does not require the distribution of keys to communicating parties.

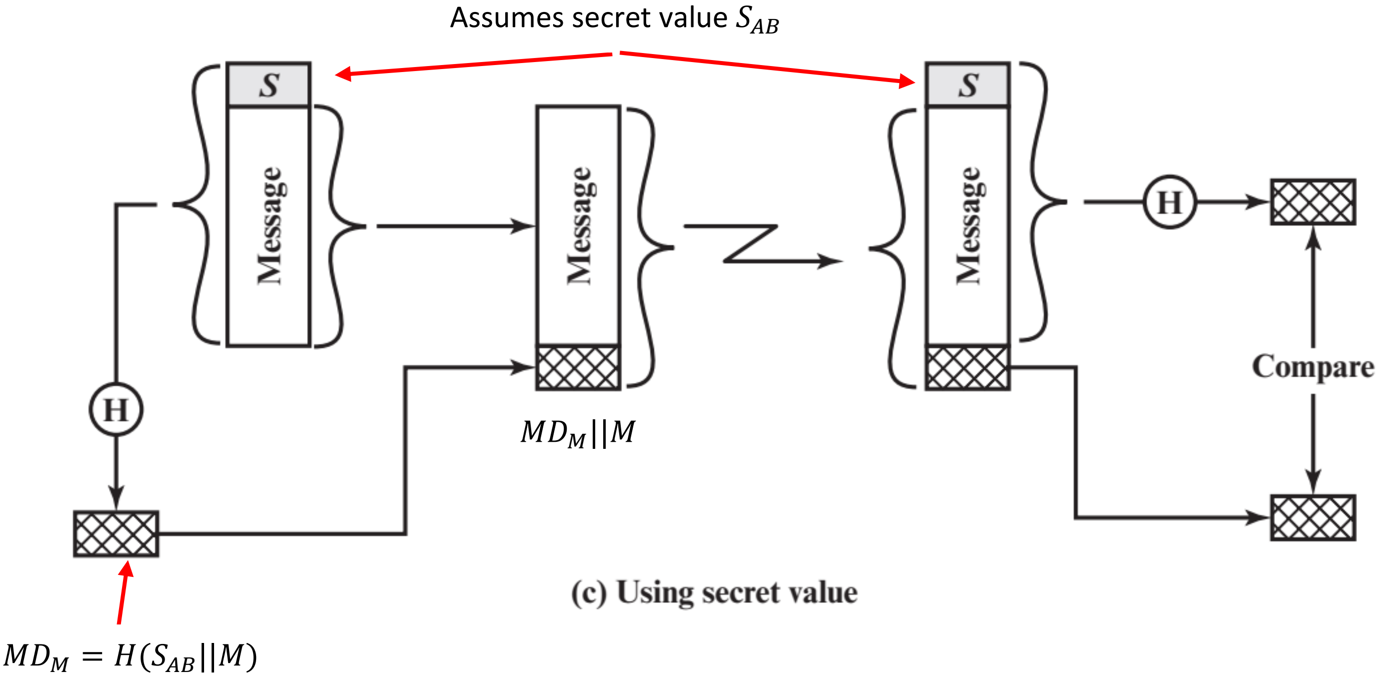


(b) Using public-key encryption

Digital Signature

No Key distribution

3. Using secret value: No encryption for message authentication. Secret value S_{AB} that is shared between communicating parties never sent, so attacker cannot modify the message. It is important technique for digital signatures. The symbol || denotes concatenation.



Assumes secret value S_{AB}

$MD_M||M$

(c) Using secret value

$MD_M = H(S_{AB}||M)$

The reasons to avoid encryption in one-way hash function as follows:

- Encryption software is slow
- Encryption hardware costs are not cheap
- Hardware optimized toward large data sizes
- Algorithms covered by patents
- Algorithms subject to export control

Secure Hash Functions

The one-way hash function or secure hash function is important not only in message authentication but in digital signatures.

Hash Function Requirements

The purpose of it is to produce a fingerprint of a file, message, or other block of data. There are properties for a hash function to be useful for message authentication as follows:

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
4. For any given code h, it is computationally infeasible to find x such that $H(x) = h$. It also known as one-way or preimage resistant
5. For any given block x, it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$. It also referred to as weak collision resistance or second preimage resistant

6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. It also referred to as strong collision resistance

The one-way property means that it is easy to generate a code given message, but impossible to generate a message given a code. This property important if the authentication technique involves the use of secret value because the secret value is not sent. However, if the hash function is not one way, an attacker can easily discover the secret value by the following steps: (1) observe transmission, (2) attacker obtains the message M and the hash code $C = H(S_{AB}||M)$, (3) inverts the hash function to obtain $S_{AB}||M = H^{-1}(C)$. (4) now the attacker can easily recover S_{AB}.

The weak collision resistance property guarantees that it is impossible to find an alternative message with the same hash value as a given message. This will prevent forgery when an encrypted hash code is used. If this property were not true, attacker can do the following: (1) observe a message plus its encrypted hash code, (2) generate an unencrypted hash code from the message, and (3) generate an alternate message with the same hash code.

The hash function that satisfies the first five properties known as a weak hash function. While if the sixth property is also satisfied then it's called a strong hash function. The sixth property strong collision resistance protects against an attack called birthday attack where this attack reduces the strength of an m-bit hash function from 2^m to $2^{m/2}$.

Security of Hash Functions

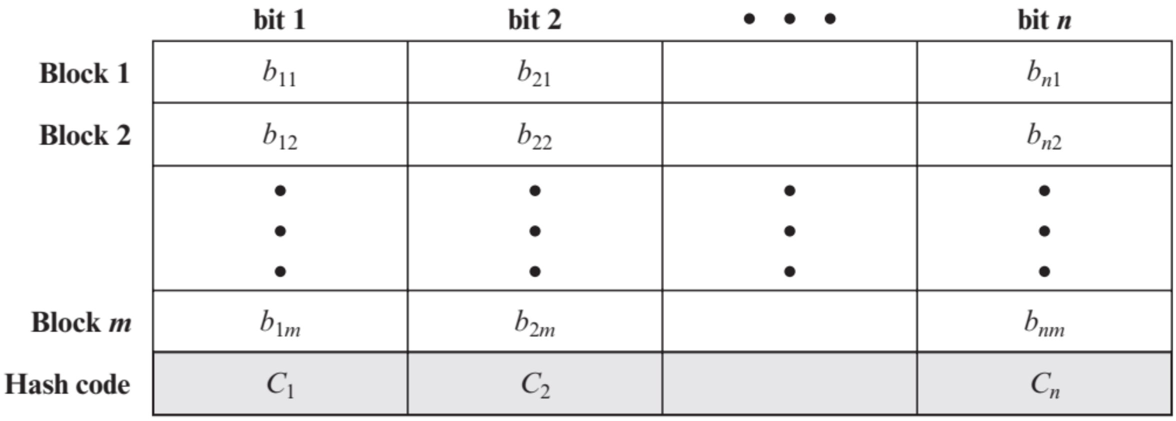
There are two approaches to attacking a secure hash function:

- Cryptanalysis: involves exploiting logical weaknesses in the algorithm
- Brute-force attack: The strength of a hash function against this attack depends solely on the length of the hash code produced by the algorithm. For a hash code of length n the strength of it against brute-force attack shown in the table below.

Preimage resistant or one-way	2^n
Second preimage or weak collision resistant	2^n
Strong collision resistant	$2^{n/2}$

Simple Hash Functions

The simplest hash function is the bit-by-bit exclusive-OR (XOR) that can be expressed as $C_i = b_{i1} \oplus b_{i2} \oplus \ldots \oplus b_{im}$, where C_i is the ith bit of the hash code, m is the number of n-bit blocks in the input, and b_{ij} is the ith bit in jth block.



	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

The bit-by-bit XOR hash function produces a simple parity for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^{-n}. The more predictably formatted data, the function is less effective. We have two ways to improve it as follows:

- Perform 1-bit circular shift on the hash function after each block is processed:
    1. Initially set the n-bit hash value to zero
    2. Process each successive n-bit block of data:
        - Rotate the current hash value to the left by one bit
        - XOR the block into the hash value
- Encrypted hash code is used with a plaintext message, and it will provide good measure of data integrity. A technique proposed by the National Bureau of standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message using the cipher block chaining (CBC) mode. The hash code C will be as follows:

  $$C = X_{N+1} = X_1 \oplus X_2 \dots \oplus X_N$$

  Then, encrypt the entire message plus hash code using CBC mode to produce the encrypted message Y.

  $$X_1 = IV \oplus D(K, Y_1)$$

  $$X_i = Y_{i-1} \oplus D(K, Y_i)$$

  $$X_{N+1} = Y_N \oplus D(K, Y_{N+1})$$

  Finally, the hash code will be as follows

  $$X_{N+1} = X_1 \oplus X_2 \dots \oplus X_N$$

  $$X_{N+1} = [IV \oplus D(K, Y_1)] \oplus [Y_1 \oplus D(K, Y_2)] \oplus \dots \oplus [Y_{N-1} \oplus D(K, Y_N)]$$

The SHA Secure Hash Function

The first SHA or SHA-0 is developed by NIST and published as FIPS 180 in 1993. It is based on MD4 hash function. And because of many weaknesses have been shown in SHA-0 it developed SHA-1 that produces 160-bits hash value. After that, NIST has developed 3 new versions of SHA with 3 hash value length known as SHA-256, SHA-384, and SHA-512 respectively. These three-hash functions known as SHA-2.

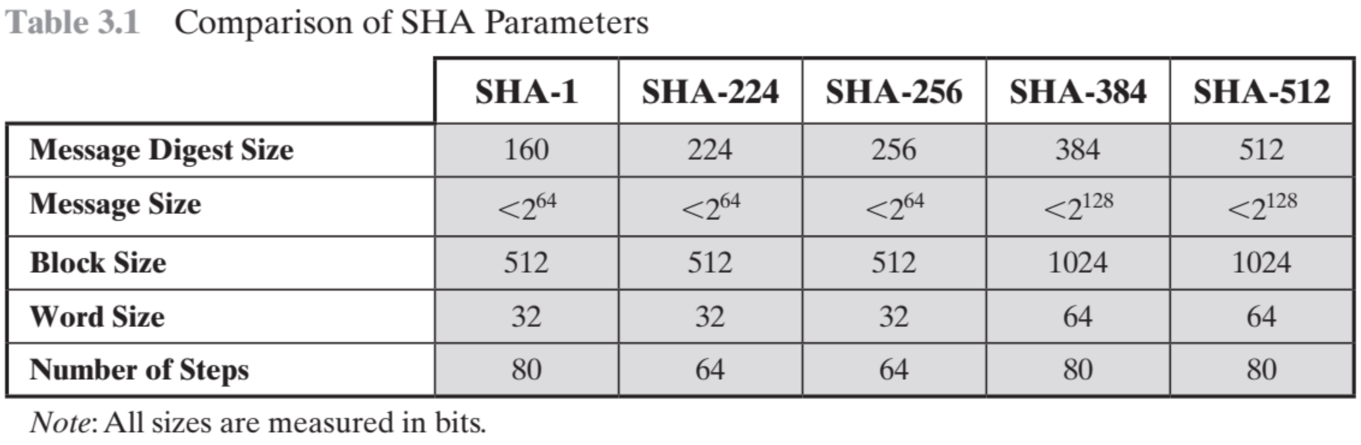


Table 3.1 Comparison of SHA Parameters

	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.

SHA-1 been broken theoretically in 2005 that 3 people (Wang X, Yin Y, and Yu H) published a paper named "Finding Collisions in the Full SHA1". But the successful attack on SHA-1 was in 2017 by a group known as "SHAttered" that they published a paper named "The first collision for full SHA-1" (See shattered.io). After that, some researchers have study "SHAttered" method and develop a theoretically method to break SHA-1 called as Chosen-prefix collisions. They publish a paper named as "From collisions to chosen-prefix collisions applications to full SHA-1". Links as follows:

- Finding Collisions in the Full SHA1:
  https://www.iacr.org/archive/crypto2005/36210017/36210017.pdf
- The first collision for full SHA-1:
  https://shattered.it/static/shattered.pdf

- From collisions to chosen-prefix collisions applications to full SHA-1:

  https://eprint.iacr.org/2019/459.pdf

We will talk about SHA-512 algorithm that takes as input a message with a maximum length of less than 2^{128} bits and produces as output a 512-bit message digest (MD). The input is processed in 1024-bit blocks as shown below in the figure.

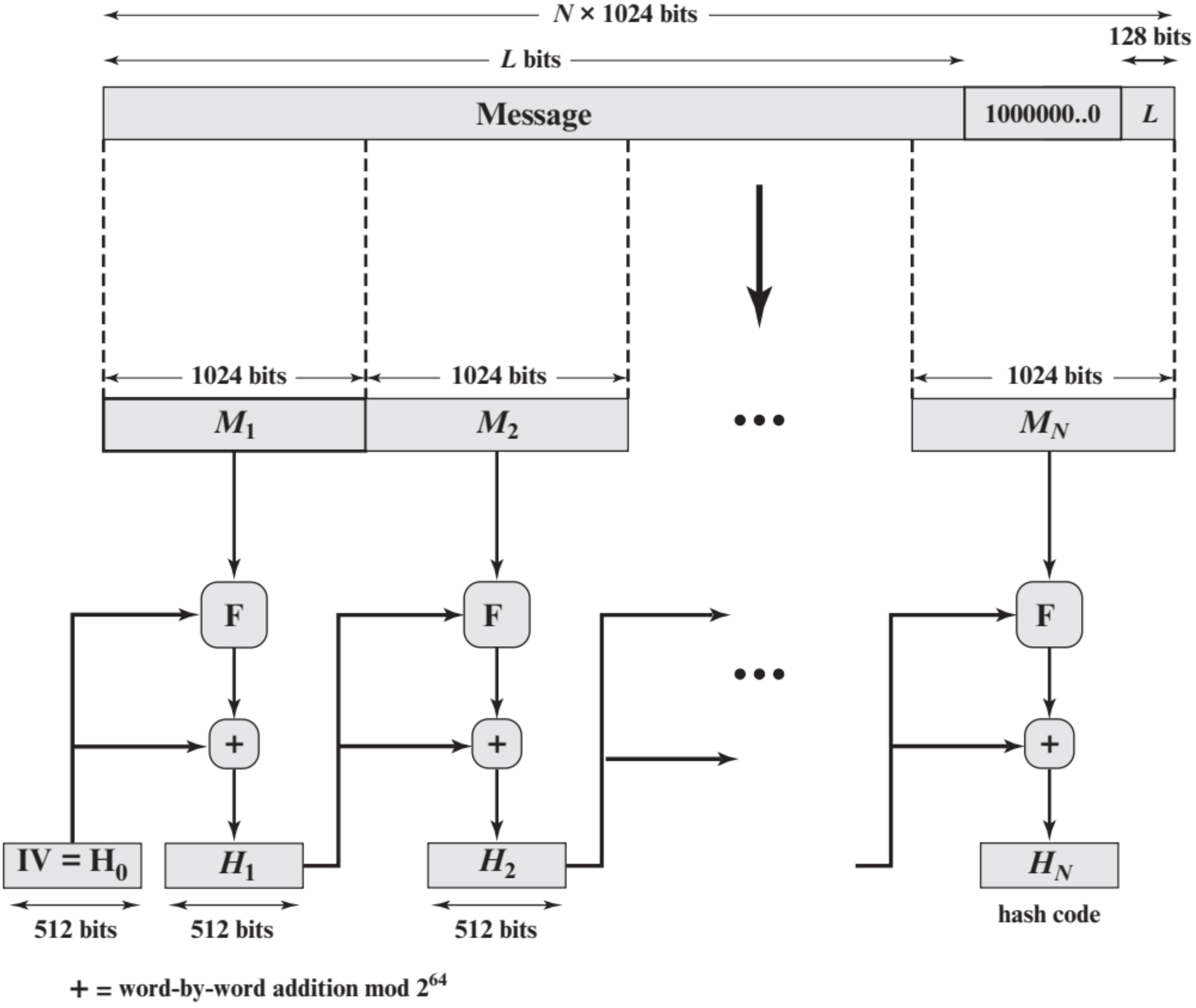


Figure 3.4 Message Digest Generation Using SHA-512

The processing steps as follows:

1. Append padding bits:
    a. The padding is consisting of a single 1 bit followed by 0 bits
    b. The number of padding bits is in range of 1 – 1024
    c. The padding is always added even if the message is already of the desired length
    d. The message is padded so that its length is congruent to 896 modulo 1024
    e. Example: If a message "abc" has a length $8 \times 3 = 24$, so the message is padded with a one bit, then $896 - (24 + 1) = 871$ zero bits.

01100001	01100010	01100011	100..00	00..011000
"a" = 61 in ascii	"b" = 62 in ascii	"c" = 63 in ascii	Padding 872 bits	Length 128 bits
	Message			

2. Append length:
    a. A block of 128 bits is appended to the message, and it contains the length of the original message
    b. Example: as in the previous example shown, we have 24 bits message length, so the 128 bits will be 00...011000. Because 24 in binary is 011000.
3. Initialize hash buffer:
    a. We have in SHA-512 eight 64-bit registers or hash buffers and are initialized to the following hexadecimal values

a = 6A09E667F3BCC908 e = 510E527FADE682D1

b = BB67AE8584CAA73B f = 9B05688C2B3E6C1F

c = 3C6EF372FE94F82B g = 1F83D9ABFB41BD6B

d = A54FF53A5F1D36F1 h = 5BE0CD19137E2179

b. These values are stored in big-endian format, which is the most significant byte of a word in the low-address (leftmost) byte position. These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

c. For example, to calculate the first buffer "a", we will take the first prime number that is 2 and the square root of it equal to 1.4142135623730950488016887242097. Take the decimal part of this whole number and multiply by 2^{64}.

$$2^{64} \times 0.4142135623730950488016887242097$$

It will be equal to 7640891576956012808.6991436125051 as we take the decimal part. Finally, convert the number on the left side of the point (7640891576956012808) to hexadecimal to get 6A09E667F3BCC908 that is the "a" register.

4. Process message in 1024-bit blocks:

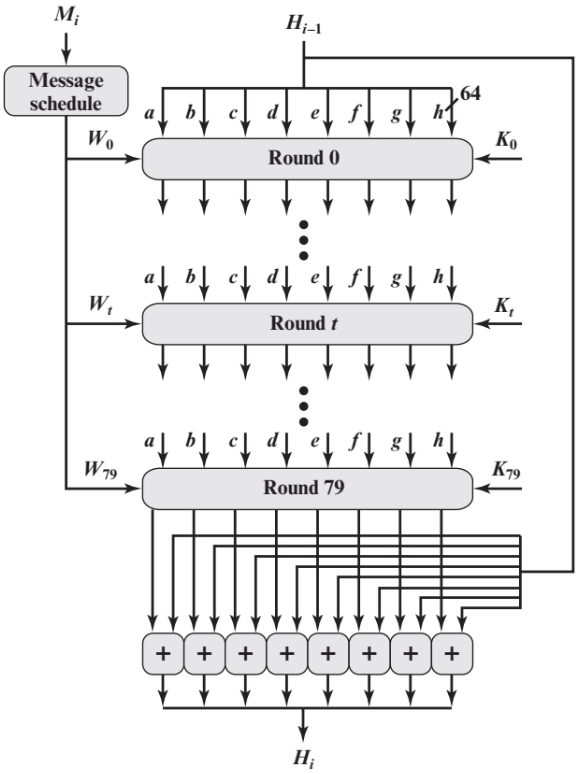


Figure 3.5 SHA-512 Processing of a Single 1024-Bit Block

a. The F module or function in the above diagram is the processing operation for each block. It consists of 80 rounds (0 – 79).

b. Each round takes as input the 512-bit buffer value abcdefgh and updates the contents of the buffer

c. At input to the first round, the buffer has the value of the intermediate hash value, H_{i-1}. Each round t makes use of a 64-bit value W_t derived from the current 1024-bit block being processed (M_i). Each round also makes use of an additive constant K_t

d. The K_t represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers.

e. Finally, the output of the 80th round is added to the input to the first round H_{i-1} to produce H

5. Output: After all N 1024-bit blocks have been processed, the output from the Nth stage is the 512-bit message digest

SHA-512 property is for every bit of the hash code is a function of every bit of the input. The difficulty of coming up with two messages having the same message digest is on the order of 2^{256} operations, while the difficulty of finding a message with a given digest is on the order of 2^{512} operations.

SHA-3

In 2007, NIST announced a competition to produce the next generation NIST hash function which will be known as SHA-3. The basic requirements that must be satisfied by any candidate as follows:

- It must be possible to replace SHA-2 with SHA-3 in any application by a simple drop-in substitution. Therefore, SHA-3 must support hash value lengths of 224, 256, 384, and 512 bits
- The algorithm must process comparatively small blocks at a time instead of requiring that the entire message be buffered in memory before processing it

In 2012, NIST selected a winning submission.

Message Authentication Codes

HMAC

It is MAC using hash code such as SHA-1 instead of symmetric encryption algorithms. HMAC is a specific type of message authentication code involving a cryptographic hash function and a secret cryptographic key. The interest has been increased in developing a MAC derived from a cryptographic hash code are as follows:

- Cryptographic hash functions generally execute faster in software than conventional encryption algorithms such as DES.
  - Why? Because I only encrypt the hash not with the message, so it is faster and that work only if the message is not sensitive nature, so it is faster and easier and simple than to encrypt the entire message.

- A hash function such as SHA-1 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.

Also, there have been several proposals for the incorporation of a secret key into an existing hash algorithm, so how we achieve that?

- By incorporated secret key for encrypt the MAC. And HMAC is when we have a MAC then we use secret key to encrypt it also we called secure message authentication code
- The approach that has received the most support is HMAC

HMAC has been issued as RFC2104. It has been chosen as the mandatory-to-implement MAC for IP security. It is used in other Internet Protocols such as transport layer security (TLS), and secure electronic transaction (SET).

The HMAC design objectives are as follows:

- Availability: To use, without modifications, available hash functions --- in particular, hash functions that perform well in software
- Flexibility: To allow for easy replaceability of the embedded hash function in case faster or more secure hash functions are found or required
- Performance: To preserve the original performance of the hash function without incurring a significant degradation. That we should be sure that we not worse on the performance
- Simplicity: To use and handle keys in a simple way
- Strength: To have a well understood cryptographic analysis of the strength of the authentication

The first two objectives are important to the acceptability of HMAC. And the last objective is the main advantage of HMAC over other proposed hash-based schemes. Furthermore, HMAC treats the hash function as a black box, and the benefit behind that is as follows:

1. An existing implementation of a hash function can be used as a module in implementing HMAC
2. If it is ever desired to replace a given hash function in an HMAC implementation, all that is required is to remove the existing hash function module and drop in the new module

The HMAC algorithm as follows:

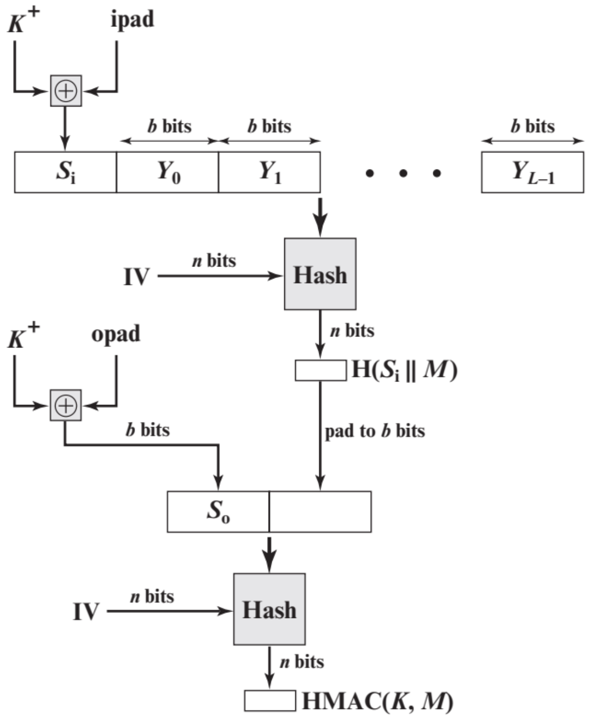


Figure 3.6 HMAC Structure

$$HMAC(K, M) = H[(K^+ \oplus opad) \parallel H[(K^+ \oplus ipad) \parallel M]]$$

Where || means concatenation.

1. Append zeros to the left end of K to create a b-bit string K^+. For example, if K is of length 160 bits and b = 512, then K will be appended with 44 zero bytes

2. XOR K^+ with ipad to produce the b-bit block S_i
3. Append M to S_i
4. Apply H to the stream generated in step 3
5. XOR K^+ with opad to produce the b-bit block S_o
6. Append the hash result from step 4 to S_o
7. Apply H to the stream generated in step 6 and output the result

Terminology for HMAC:

- H: embedded hash function, such as SHA-1
- M: Message input including padding
- Y_i: ith block of M
- L: Number of blocks in M
- b: number of bits in a block
- n: length of hash code produced
- K: secret key
  - If K > b, hash K to reduce its length to b through message digest
  - If K < b, append 0 bits to the left of K so that its length = b
  - If K = b, do nothing
- K^+: K padded with zeros on the left so that the result is b bits in length
- Ipad: 00110110 or 36 in hexadecimal repeated b/8 times
- Opad: 01011100 or 5C in hexadecimal repeated b/8 times

Note:

The XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, but a different set of bits. And by-passing

S_i and S_o through the hash algorithm, we have pseudorandomly generated two keys from K.

MACs Based on Block Ciphers

Cipher-based message authentication code (CMAC)

The Cipher-Based Message Authentication Code (CMAC) mode of operation is NIST standard SP 800-38B. It is for use with AES and triple DES. First, let us define the operation of CMAC when the message is an integer multiple n of the cipher block length b. For AES, b = 128 and for 3DES, b = 64. The message is divided into n blocks (M_1, M_2,…,M_n). For AES, the key size is 128, 192, or 256 bits. For 3DES, the key size is 112 or 168 bits. CMAC is calculated as follows:

$$C_1 = E(K, M_1)$$

$$C_2 = E(K, [M_2 \oplus C_1])$$

$$C_3 = E(K, [M_3 \oplus C_2])$$

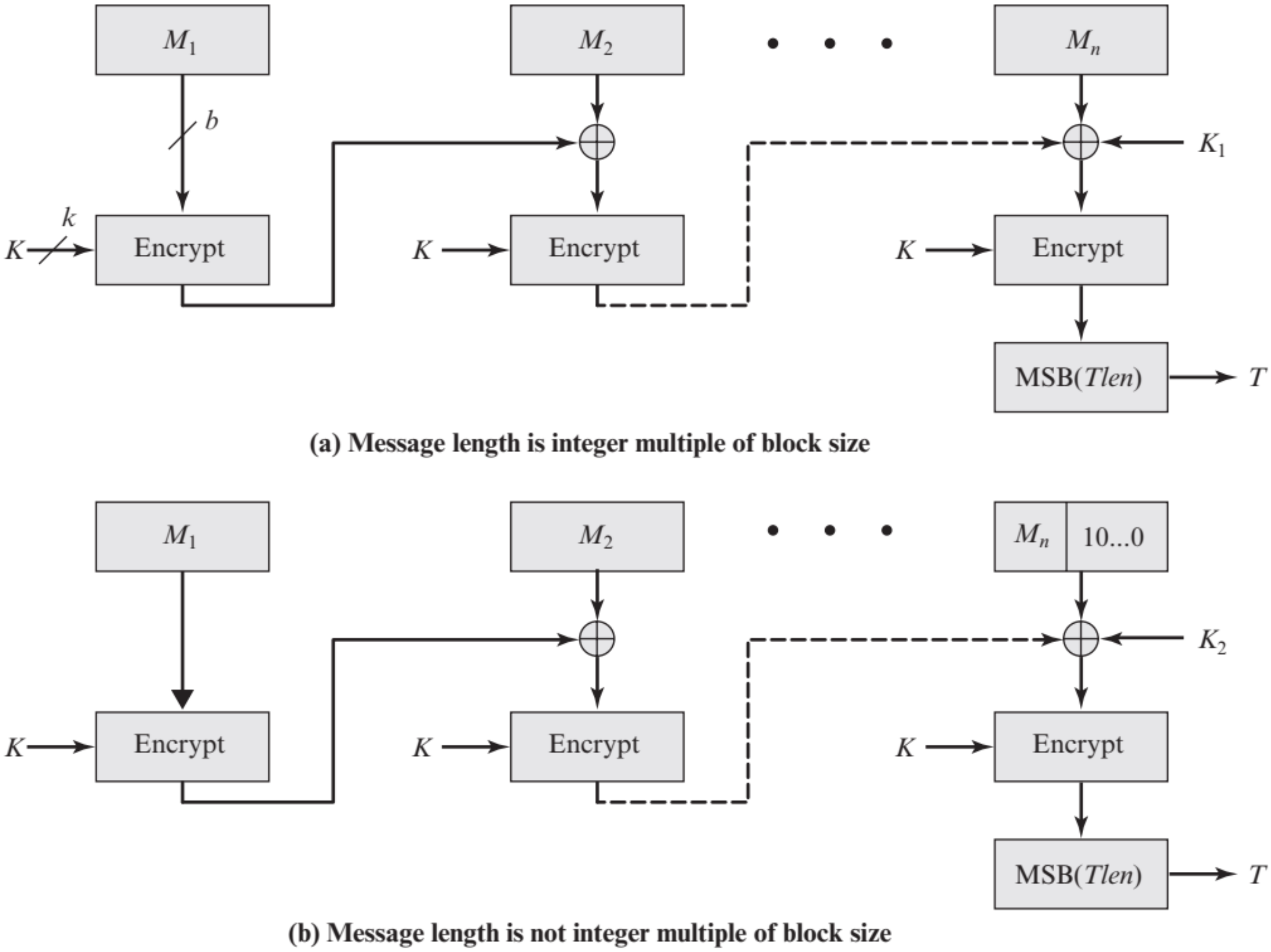
$$\vdots$$

$$C_n = E(K, [M_N \oplus C_{n-1} \oplus K_1])$$

$$T = MSB_{Tlen}(C_n)$$

Where T: message authentication code or tag, $Tlen$: bit length of T, and $MSB_s(X)$: the s leftmost bits of the bit string X.

If the message is not an integer multiple of the cipher block length, then the final block is padded to the right (least significant bits) with a 1 and as many 0s as necessary so that the final block is also of length b. To generate the two n-bit keys, the block cipher is applied to the block that consists entirely of 0 bits. The first subkey is derived from the resulting ciphertext by a left shift of one bit and, conditionally, by XORing a constant that depends on the block size. The second subkey is derived in the same manner from the first subkey.



(a) Message length is integer multiple of block size

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

Figure 3.7 Cipher-Based Message Authentication Code (CMAC)

Counter with cipher block chaining-message authentication code (CCM)

It is defined in SP 800-38C by NIST and referred to as an authenticated encryption that is a term used to describe encryption systems that simultaneously protect confidentiality and authenticity (integrity) of communications. The key algorithmic ingredients of CCM as follows:

- AES encryption algorithm
- CTR mode of operation

- CMAC authentication algorithm

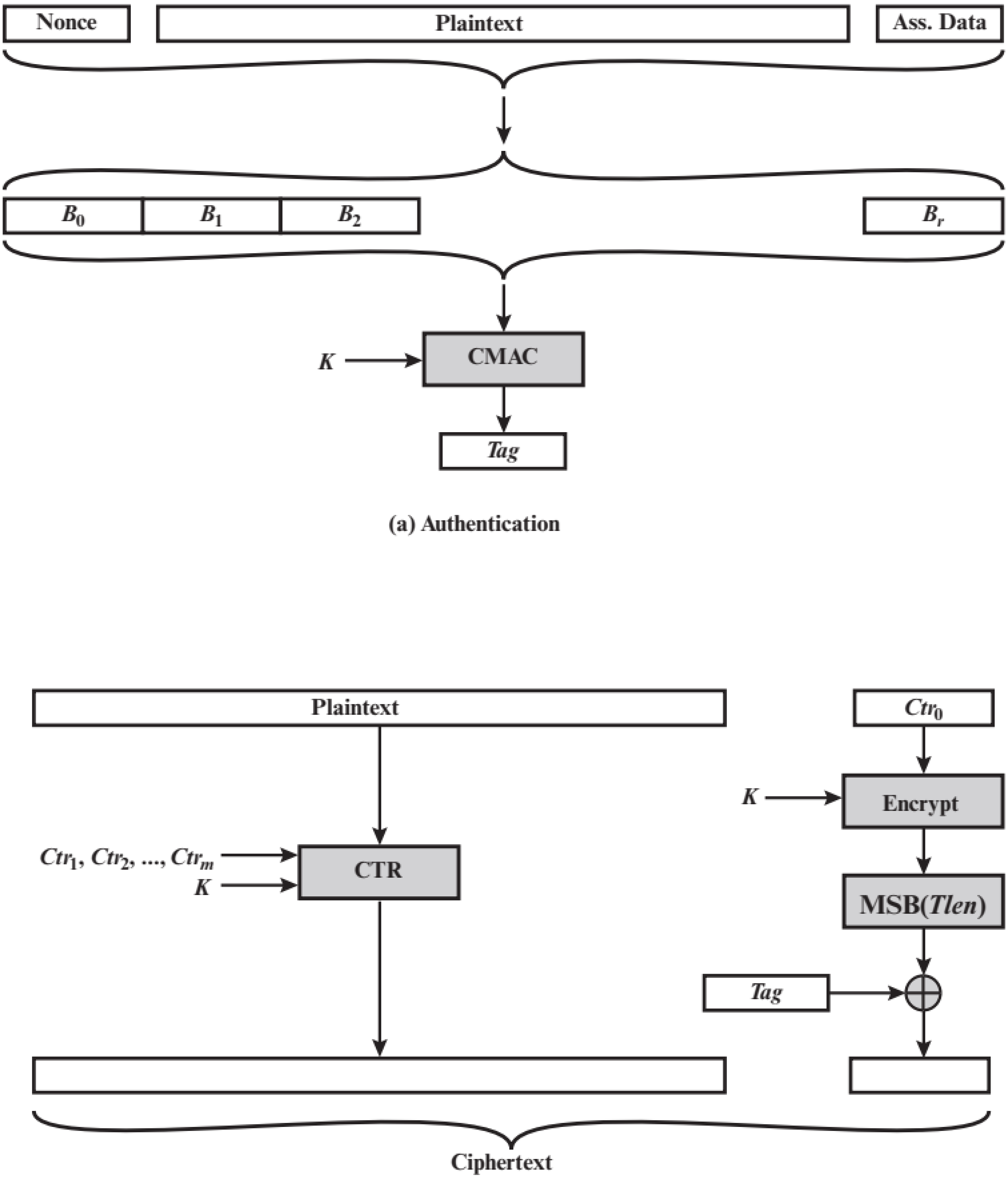
A single key K is used for both encryption and MAC algorithms. The input to the CCM encryption process consists of 3 elements as follows:

1. Data that will be both authenticated and encrypted. This the plaintext message P of data block
2. Associated data A that will be authenticated but not encrypted. For example, a protocol header that must be transmitted in the clear for proper protocol operation, but which needs to be authenticated
3. A nonce N that is assigned to the payload and the associated data. It is a unique value that is different for every instance during the lifetime of protocol association, and it is intended to prevent certain types of attacks such as replay attacks.

The structure of CCM as follows:

- For authentication, the input includes the nonce, the associated data, and the plaintext. That is formatted as a sequence of blocks B_0 through B_r. The first block contains the nonce plus some formatting bits that indicate the lengths of the N, A, and P elements.
- The resulting sequence of blocks serves as input to the CMAC algorithm, which produces a MAC value with length $Tlen$.
- For encryption, a sequence of counters is generated that must be independent of the nonce. The authentication tag is encrypted in CTR mode using the single counter Ctr_0. The $Tlen$ most significant bits of the output are XORed with the tag to produce an encrypted tag. The remaining counters are used for the CTR mode

encryption of the plaintext. The encrypted plaintext is concatenated with the encrypted tag to form the ciphertext output.



(a) Authentication

(b) Encryption

Figure 3.8 Counter with Cipher Block Chaining-Message Authentication Code

Public-Key Cryptography Principles

Public-Key Encryption Structure

Public-Key first proposed by Diffie and Hellman in 1976. It was based on mathematical functions rather than on simple operations on bit patterns. It is asymmetric that means it uses two separate keys, one for encryption and the other for decryption. There are several common misconceptions about public-key as follows:

- Public-Key encryption is more secure from cryptanalysis than conventional encryption
  - The security of any encryption scheme depends on: (1) the length of the key and (2) the computational work involved in breaking a cipher
- Public-Key encryption is a general-purpose technique that has made conventional encryption obsolete
- There is a feeling that key distribution is trivial when using public-key encryption, compared to the rather cumbersome handshaking involved with key distribution centers for conventional encryption

Public-Key encryption scheme has 6 ingredients as follows:

1. Plaintext: data
2. Encryption algorithm
3. Public key: Used for encryption
4. Private key: Used for decryption
5. Ciphertext: For a given message, two different keys will produce two different ciphertexts
6. Decryption algorithm

The steps for public key as follow:

1. Each user generates a pair of keys

2. Each user places one of the two keys in a public register, this is the public key. The private key kept private. Each user maintains a collection of public keys obtained from others
3. If Bob wants to send a private message to Alice, Bob encrypts the message using Alice's public key
4. When Alice receives the message, she decrypts it using her private key

The private key referred to it as private not secret to avoid confusion with conventional encryption.

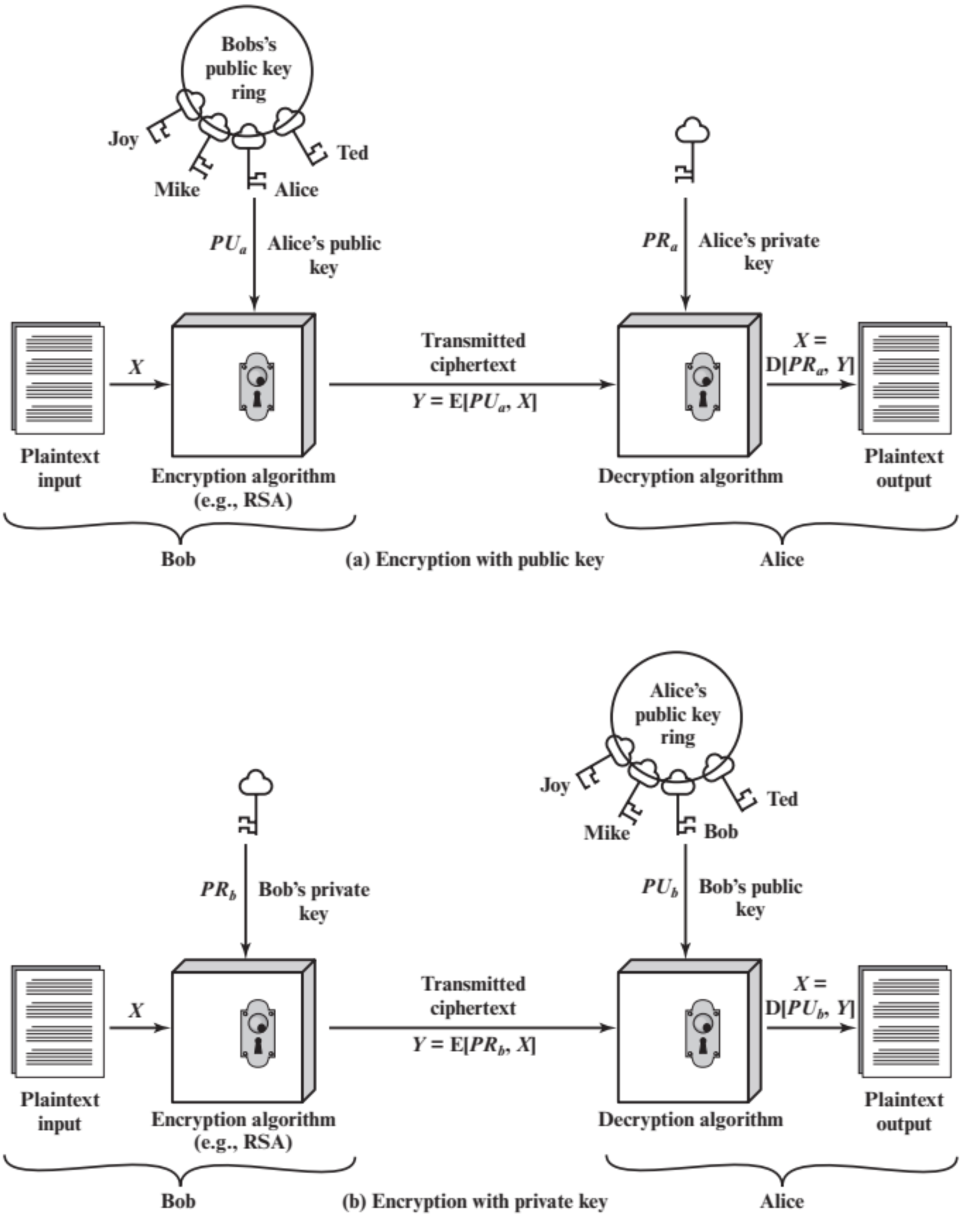


Figure 3.9 Public-Key Cryptography

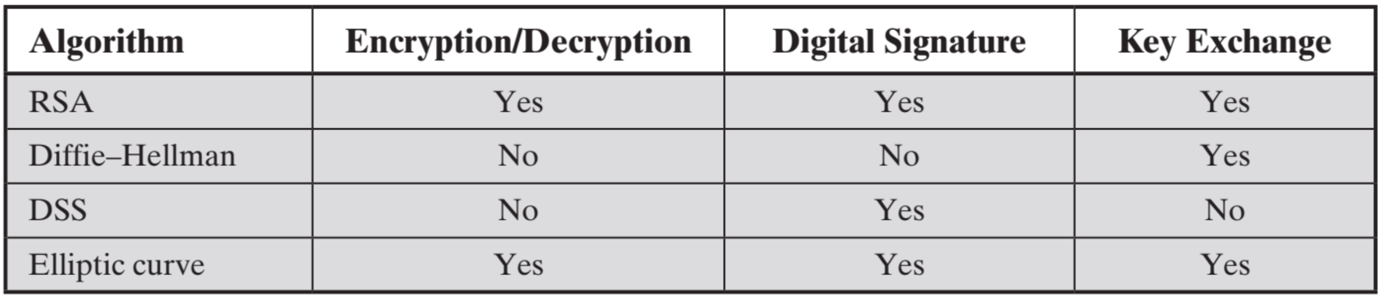
Applications for Public-Key Cryptosystems

Public-key systems are characterized using a cryptographic type of algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key, the receiver's public key, or both to perform some type of cryptographic function. The use of public-key cryptosystems classified into three categories as follows:

1. Encryption/decryption: The sender encrypts a message with the recipient's public key
2. Digital signature: The sender signs a message with its private key
3. Key exchange: Two sides cooperate to exchange a session key

Public-key algorithms require more computation than symmetric algorithms for comparable security and a comparable plaintext length. So for that, public-key algorithms are used only for short messages or data blocks such as to encrypt a secret key or PIN. The table shows public-key algorithms and suitable application for it.

Table 3.2 Applications for Public-Key Cryptosystems



Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Diffie–Hellman	No	No	Yes
DSS	No	Yes	No
Elliptic curve	Yes	Yes	Yes

Requirements for Public-Key Cryptography

1. Easy for a party B to generate a pair (public key PU_b, private key PR_b)
2. Easy for sender A, knowing the public key and the message to be encrypted M to generate the corresponding ciphertext

$$C = E(PU_b, M)$$

3. Easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message

$$M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$$

4. Infeasible for an opponent knowing the public key to determine the private key
5. Infeasible for an opponent knowing the public key and a ciphertext to recover the original message
6. Either of the two related keys can be used for encryption with the other used for decryption

$$M = D[PU_b, E(PR_b, M)] = D[PR_b, E(PU_b, M)]$$

Public-Key Cryptography Algorithms

I will talk about RSA and Diffie-Hellman algorithms that are the most used public-key algorithms. As well as I will talk about two other algorithms.

The RSA Public-Key Encryption Algorithm

It was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT. It was first published in 1978. RSA is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n.

Basic RSA Encryption and Decryption

Encryption and decryption are as follows for converting message M to ciphertext C and vice versa.

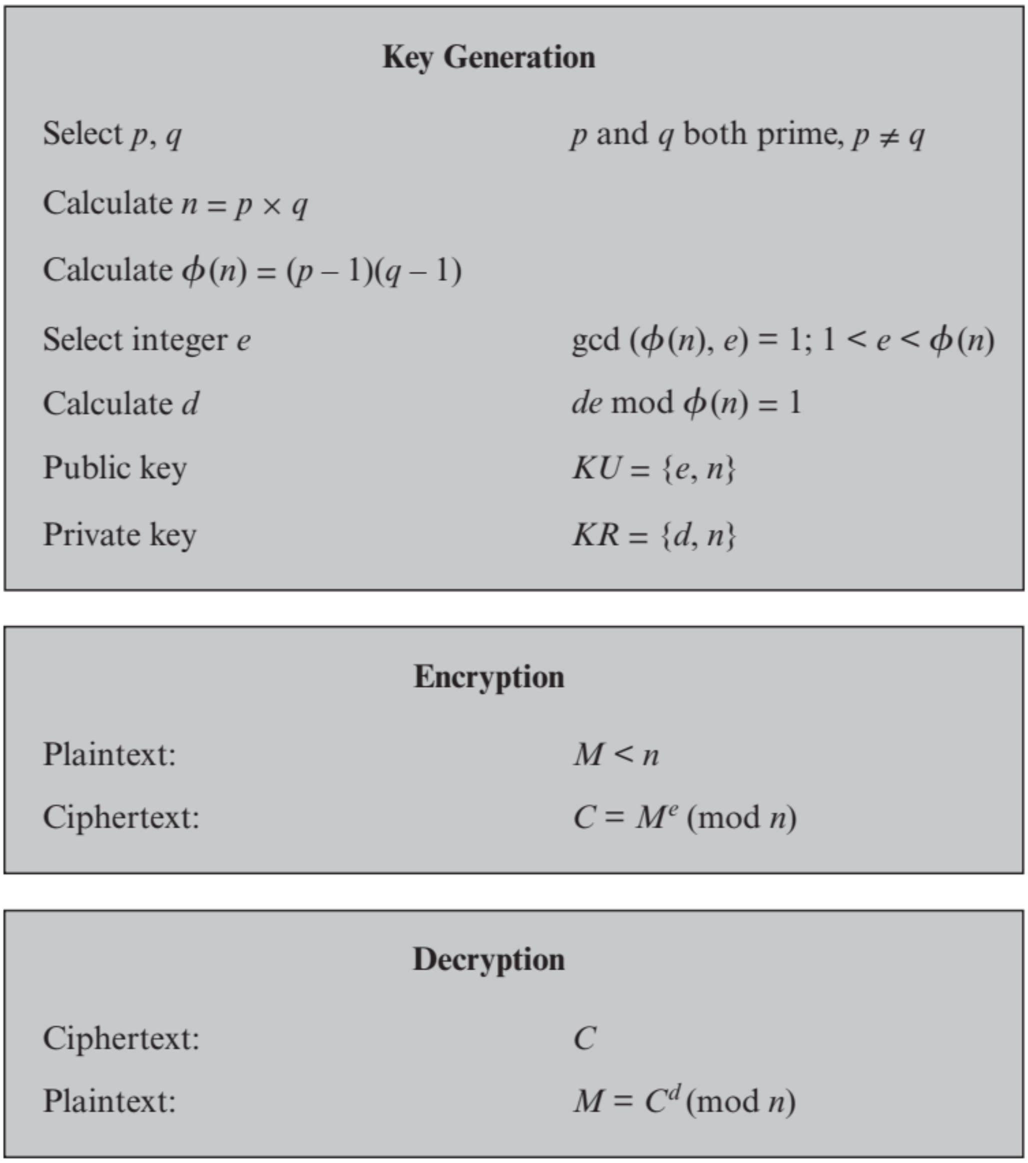
$$C = M^e \ mod \ n$$

$$M = C^d \; mod \; n = (M^e)^d \; mod \; n = M^{ed} \; mod \; n$$

Both sender and receiver must know the values of n and e, and only the receiver knows the value of d. RSA requirements to satisfy public-key encryption as follows:

- It is possible to find values of e, d, n such that $M^{ed} \; mod \; n = M$ for all $M < n$
- It is relatively easy to calculate M^e and C^d for all values of $M < n$
- It is infeasible to determine d given e and n

The first requirement met easily, but the third or last requirement met for large e and n values.



Key Generation

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	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 3.10 The RSA Algorithm

Let us take an example on RSA algorithm:

1. Select two prime numbers, $p = 17$ and $q = 11$
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\varphi(n) = 16 \times 10 = 160$
4. Select e such that e is relatively prime to $\varphi(n) = 160$ and less than $\varphi(n)$; we choose $e = 7$
5. Determine d such that $de \; mod \; 160 = 1$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$

So, the public key is $PU = \{7, 187\}$ and the private key $PR = \{23, 187\}$. If we have a plaintext or message $M = 88$ to encrypt it, we need to calculate $C = 88^7 mod \; 187 = 11$. And assume we are the receiver, and we need to decrypt the ciphertext $C = 11$, we will calculate $M = 11^{23} \; mod \; 187 = 88$.

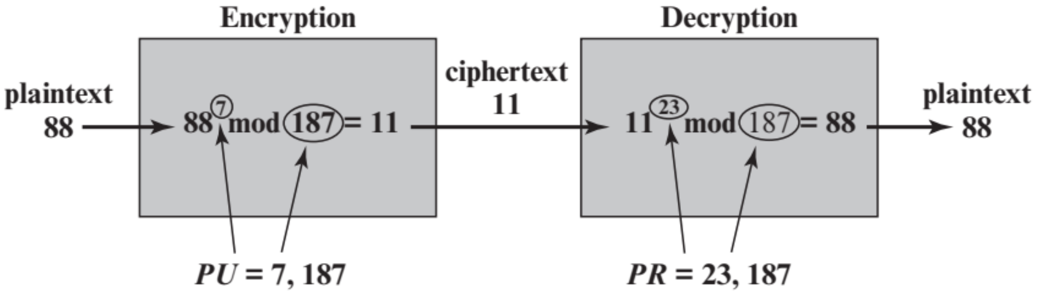


Figure 3.11 Example of RSA Algorithm

As we said before, RSA need to show that $M^{ed} \; mod \; n = M$, and since $ed \equiv 1(mod \; \varphi(n))$ we have that $ed = k\varphi(n) + 1$ for some integer k. So:

$$M^{ed} mod \; n = M^{k\varphi(n)+1} mod \; n =$$

$$M^{t\varphi(n)} M^1 \; (mod \; n) = 1^t M \; (mod \; n) = M \; (mod \; n)$$

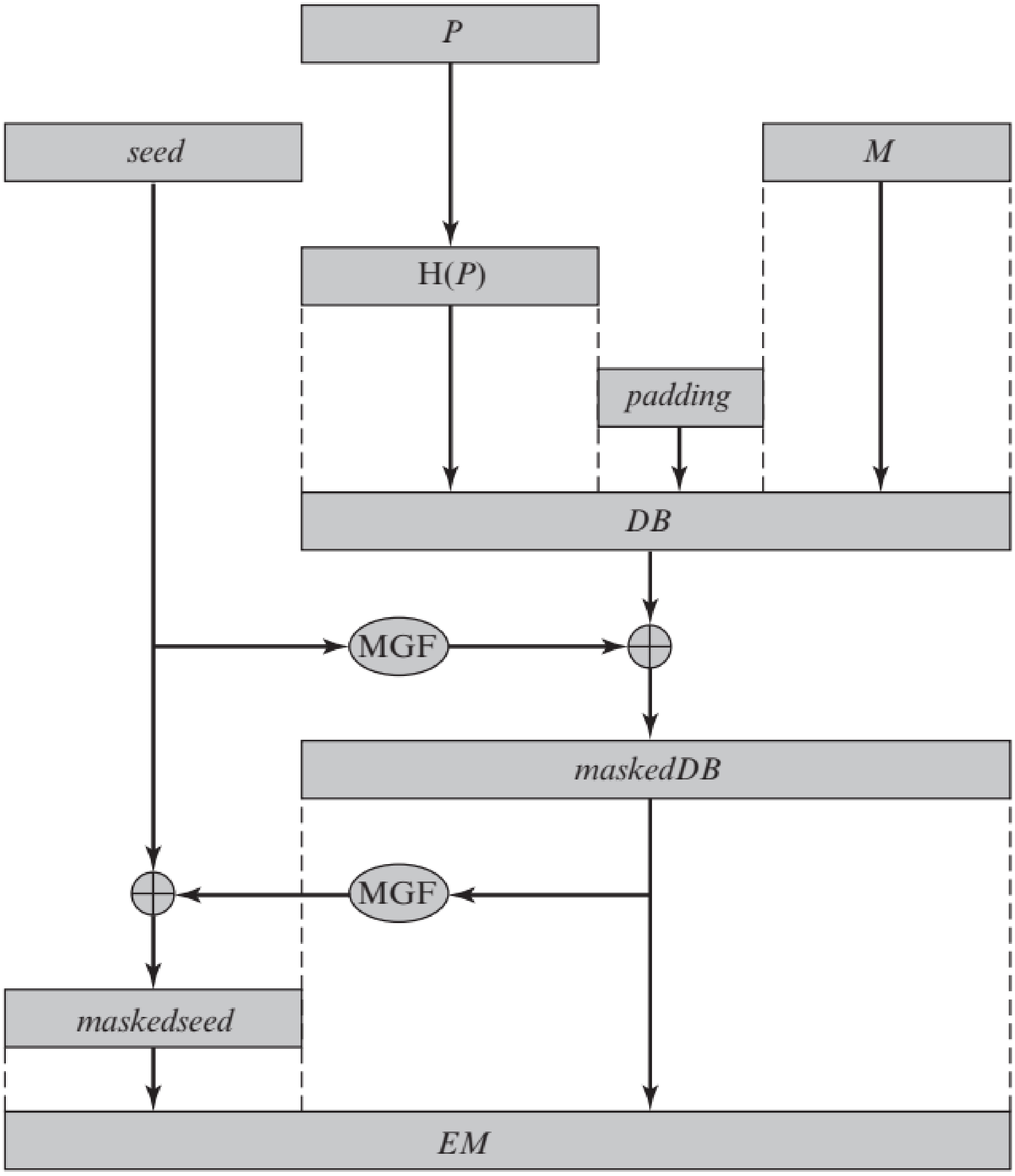
As desired.

Security Considerations

There are possible attack approaches which can be used on RSA as follows:

- Mathematical attacks: Many approaches using mathematics to try to factor the product of two primes. So, the defense against these kinds of attacks is to use large key size. NIST in SP 800-131A recommends the use of a 2048-bit key size, and a report 2014 from the European Union Agency for Network and information security recommends a 3072-bit key length.
- Timing attacks: They are dependent on the running time of the decryption algorithm. In which the attacker analyze the time taken to execute the algorithm. Several approaches to mask the time required so as to prevent attempts to deduce key size have been suggested such as introducing a random delay.
- Chosen ciphertext attacks: It happens by selecting blocks of data that when processed using the target's private key. These attacks can be reduced by using padding to hide the actual encrypted plaintext. RSA security inc. recommends preventing chosen ciphertext attacks by modifying the plaintext using a procedure known as optimal asymmetric encryption padding (OAEP). OAEP steps are as follows:
    1. Message M to be encrypted is padded
    2. A set of optional parameters P is passed through a hash function H
    3. The output is then padded with zeros to get the desired length in the overall data block (DB)
    4. A random seed is generated and passed through another hash function called the mask generating function (MGF)

5. The resulting hash value is bit-by-bit XORed with DB to produce maskedDB
6. The maskedDB is passed through the MGF to form a hash that is XORed with the seed to produce the masked seed
7. The concatenation of the maskedseed and the maskedDB forms the encoded message (EM)
8. Finally, the EM is then encrypted using RSA



P = encoding parameters	DB = data block
M = message to be encoded	MGF = mask generating function
H = hash function	EM = encoded message

Figure 3.12 Encryption Using Optimal Asymmetric Encryption Padding (OAEP)

Diffie-Hellman Key Exchange

It is the first published public-key algorithm, and the first paper that defined public-key cryptography. There are several commercial products employ this key exchange technique. The purpose of the algorithm is to enable two users to securely exchange a secret key that then can be used for subsequent encryption of messages and it is limited on this job that is to exchange keys. It depends for its effectiveness on the difficulty of computing discrete logarithms. Discrete logarithm is as follows:

- Define a primitive root of a prime number p as one whose powers generate all the integers from 1 to $p - 1$. So, if a is a primitive root of the prime number p, then the numbers $a\ mod\ p, a^2\ mod\ p, \dots, ap^{-1}\ mod\ p$ are distinct and consist of the integers from 1 through $p - 1$ in some permutation.
- For any integer b less than p and a primitive root a of prime number p, one can find a unique exponent i such that $b = a^i\ mod\ p$ where $0 \leq i \leq (p - 1)$.
- The i is referred to as the discrete logarithm or index of b for the base a, $mod\ p$. And we denote it as $dlog_{a,p}(b)$.

The Algorithm

The steps for Diffie-Hellman algorithm as follows:

1. There are 2 publicly known numbers:
   a. A prime number q
   b. An integer α that is a primitive root of q
2. Assume A and B want to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A}\ mod\ q$
3. Similarly, B will select random integer $X_B < q$ and computes $Y_B = \alpha^{X_B}\ mod\ q$

4. The X value kept private, and the Y will be shared publicly between A and B

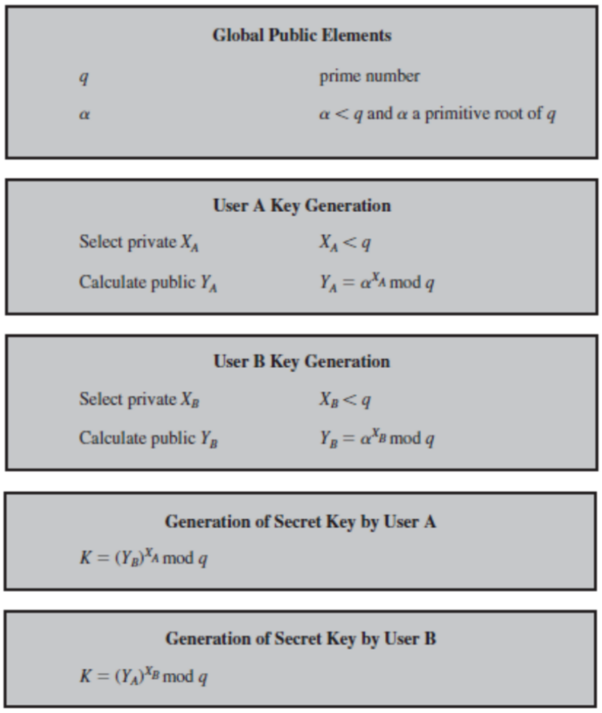
5. User A computes the key $K = (Y_B)^{X_A}\ mod\ q$

6. User B computes the key $K = (Y_A)^{X_B}\ mod\ q$

7. The two K calculations will produce the same result:

$$
\begin{aligned}
K &= (Y_B)^{X_A}\ mod\ q \\
&= (\alpha^{X_B}\ mod\ q)^{X_A}\ mod\ q \\
&= (\alpha^{X_B})^{X_A}\ mod\ q \\
&= \alpha^{X_B X_A}\ mod\ q = (\alpha^{X_A})^{X_B}\ mod\ q \\
&= (\alpha^{X_A}\ mod\ q)^{X_B}\ mod\ q \\
&= (Y_A)^{X_B}\ mod\ q
\end{aligned}
$$

An adversary only has $q, a, Y_A, and\ Y_B$. So, he forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute $X_B = dlog_{\alpha,q}(Y_B)$. The security of the Diffie–Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms.



Global Public Elements

q prime number

α $\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A $X_A < q$

Calculate public Y_A $Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B $X_B < q$

Calculate public Y_B $Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 3.12 The Diffie-Hellman Key Exchange Algorithm

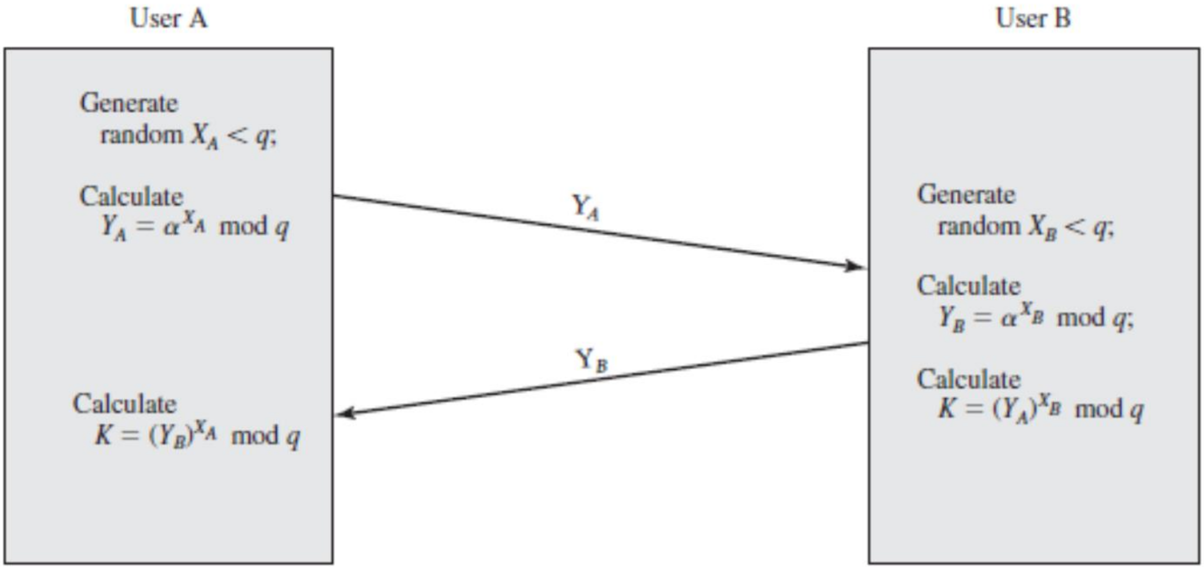

User A
User B
Generate random $X_A < q$;
Calculate $Y_A = \alpha^{X_A} \bmod q$
Y_A
Generate random $X_B < q$;
Calculate $Y_B = \alpha^{X_B} \bmod q$;
Y_B
Calculate $K = (Y_A)^{X_B} \bmod q$
Calculate $K = (Y_B)^{X_A} \bmod q$

Figure 3.13 Diffie-Hellman Key Exchange

Let us take an example if we have $q = 353$ and $\alpha = 3$. Users A and B select a secret key $X_A = 97$ and $X_B = 233$. Then, each computes its public key:

- A computes $Y_A = 3^{97}\ mod\ 353 = 40$. And B computes $Y_B = 3^{233}\ mod\ 353 = 248$.

After that. They will exchange public keys, and each will compute the common secret key:

- A computes $K = (248)^{97}\ mod\ 353 = 160$. And B computes $K = (40)^{233}\ mod\ 353 = 160$.

Now, if an attacker knows $q = 353, a = 3, Y_A = 40, and\ Y_B = 248$. It would be possible to determine the secret key 160 by using Brute force. The attacker wants to solve $3^a\ mod\ 353 = 40$ and $3^b\ mod\ 353 = 248$. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponent value of 97, which provide $3^{97}\ mod\ 353 = 40$. So, we need to use larger numbers to make the problem impossible or impractical.

Key Exchange Protocols

Another example on Diffie-Hellman algorithm, suppose that a group of users each generate a long-lasting private value X_A and calculate a public value Y_A. These public values, together with global public values for q and α, are stored in some central directory. So, any time any user can access another user public key, calculate a secret key, and use that to send an encrypted message to the user. If the central directory is trusted, then this form of communication provides both confidentiality because no other user can read the message, and it provides authentication that the recipient knows that only the sender could have created a message using this key. However, this technique does not protect against replay attacks.

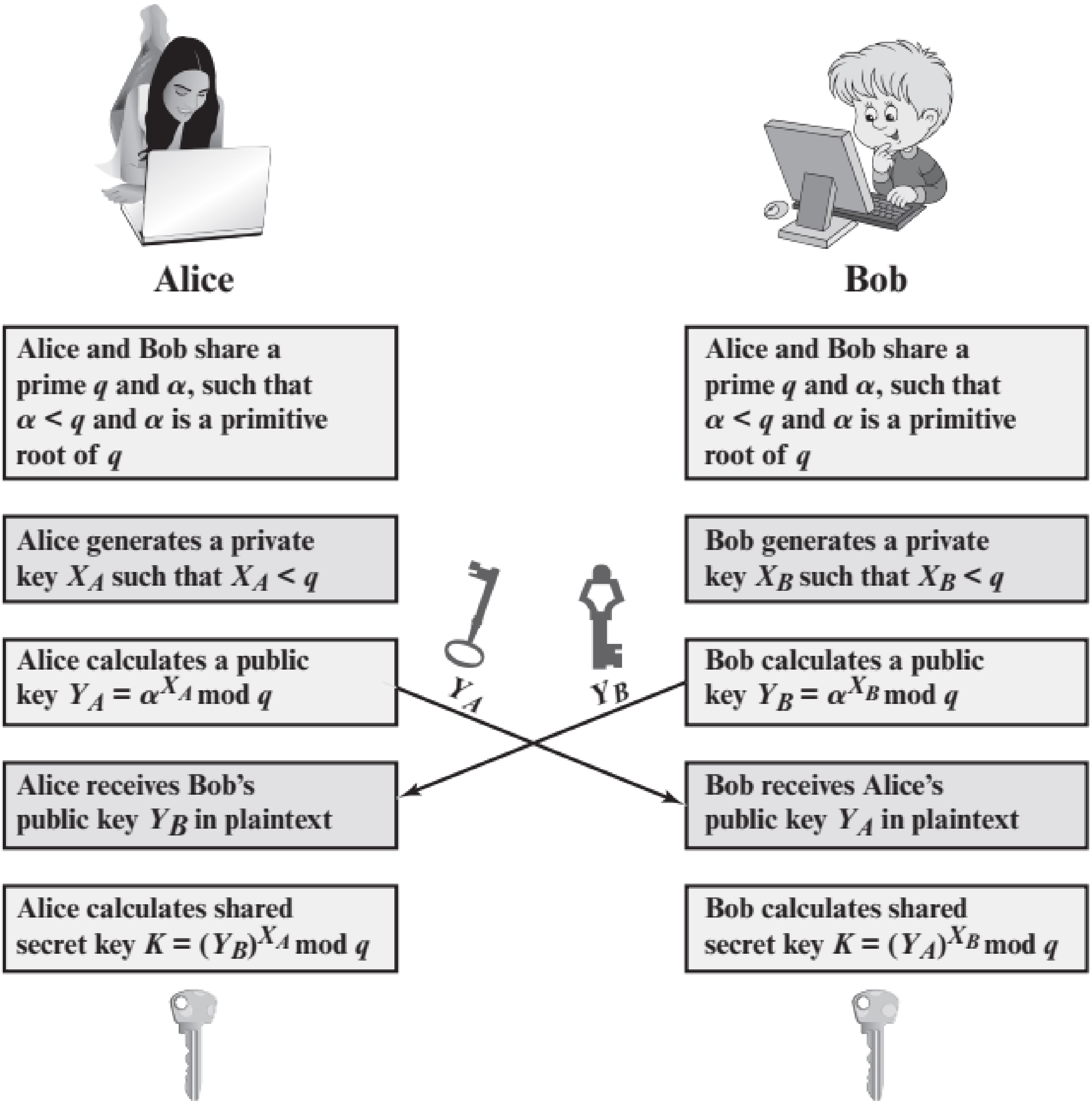

Alice
Bob
Alice and Bob share a prime q and α, such that $\alpha < q$ and α is a primitive root of q
Alice and Bob share a prime q and α, such that $\alpha < q$ and α is a primitive root of q
Alice generates a private key X_A such that $X_A < q$
Bob generates a private key X_B such that $X_B < q$
Alice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

Figure 3.13 The Diffie–Hellman Key Exchange

Man-In-The-Middle Attack

Diffie-Hellman key exchange is insecure against a man-in-the-middle attack that is an attack lets somebody eavesdrop to a communication between two users. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows:

1. Darth prepares for the attack by generating two random private keys X_{D1} and X_{D2}, and then computing the corresponding public keys Y_{D1} and Y_{D2}
2. Alice transmits Y_A to Bob
3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Then, calculates $K2 = (Y_A)^{X_{D2}}\ mod\ q$
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{X_B}\ mod\ q$
5. Bob transmits Y_B to Alice
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Then, calculates $K1 = (Y_B)^{X_{D1}}\ mod\ q$
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A}\ mod\ q$

Now, Bob and Alice think that they share a secret key. But instead, Bob and Darth share secret key $K1$, and Alice and Darth share secret key $K2$. So, all future communication between them will be compromised as follows:

1. Alice sends an encrypted message M: $E(K2, M)$
2. Darth intercepts the encrypted message and decrypts it to recover M
3. Darth sends Bob $E(K1, M)$ or $E(K1, M')$, where M' is any message. Because Darth has two cases:
    a. Simply eavesdrop on the communications without altering it
    b. Modify the message going to Bob

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. And this vulnerability can be overcome by using digital signatures and public-key certificates as we will discuss it later.

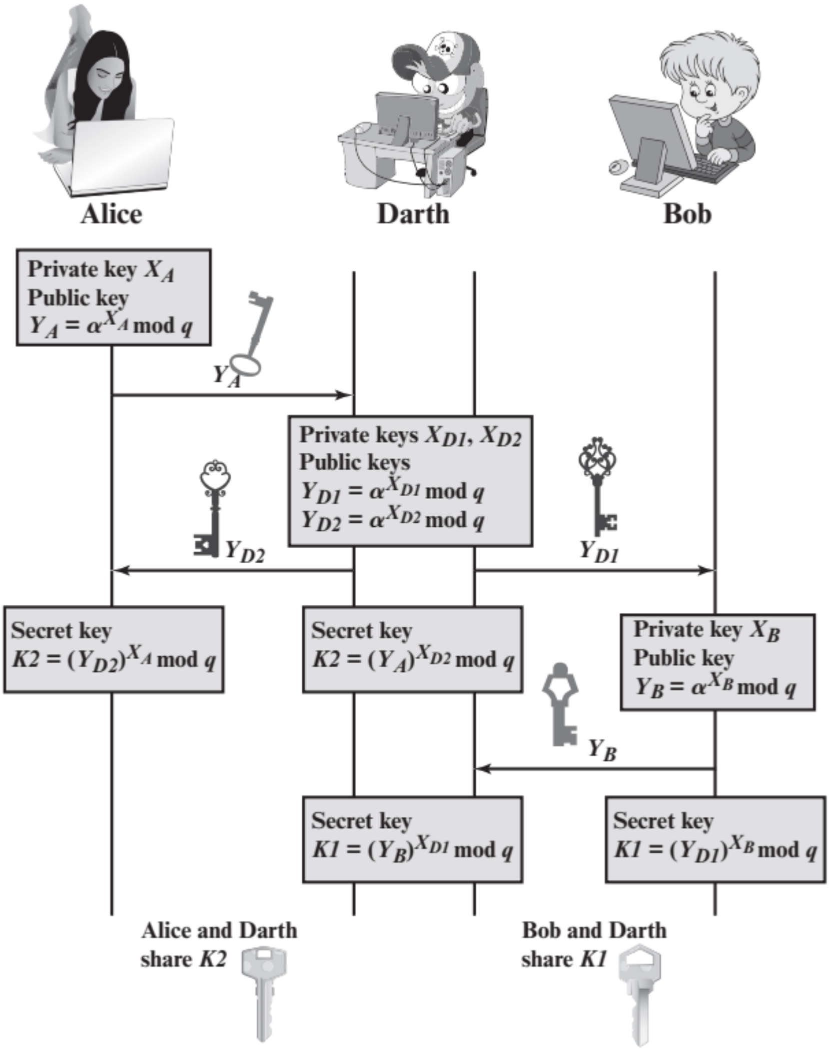

Alice
Darth
Bob
Private key X_A
Public key
$Y_A = \alpha^{X_A} \bmod q$
Y_A
Private keys X_{D1}, X_{D2}
Public keys
$Y_{D1} = \alpha^{X_{D1}} \bmod q$
$Y_{D2} = \alpha^{X_{D2}} \bmod q$
Y_{D2}
Y_{D1}
Secret key
$K2 = (Y_{D2})^{X_A} \bmod q$
Secret key
$K2 = (Y_A)^{X_{D2}} \bmod q$
Private key X_B
Public key
$Y_B = \alpha^{X_B} \bmod q$
Y_B
Secret key
$K1 = (Y_B)^{X_{D1}} \bmod q$
Secret key
$K1 = (Y_{D1})^{X_B} \bmod q$
Alice and Darth share $K2$
Bob and Darth share $K1$

Figure 3.14 Man-in-the-Middle Attack

Other Public-Key Cryptography Algorithms

Now I will talk about two public-key algorithms that have found commercial acceptance, DSS and ECC.

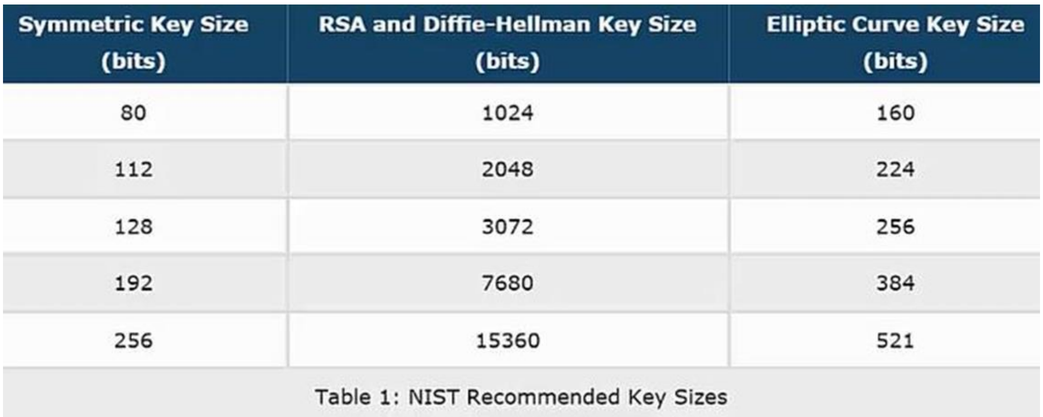
Digital Signature Standard

Digital Signature Standard (DSS) has been published by NIST as FIPS PUB 186. It makes use of the SHA-1 and presents a new digital signature technique, the digital signature algorithm (DSA). DSS originally proposed in 1991 and revised in 1993 and again in 1996.

DSS uses an algorithm that is designed to provide only the digital signature function. It cannot be used for encryption or key exchange.

Elliptic-Curve Cryptography

It is based on the use of a mathematical construct known as the elliptic curve. The principal attraction of ECC compared to RSA is that it appears to offer equal security for a far smaller bit size, thereby reducing processing overhead. Because the bit length for RSA has increased that put a heavier processing on applications especially for electronic commerce sites that conduct large numbers of secure transactions. As ECC overcome RSA from this view, but still the confidence level of ECC is not yet as high as that in RSA. Also, ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman.



Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Table 1: NIST Recommended Key Sizes

Digital Signatures

It is defined in NIST FIPS PUB 186-4 as follows:

> “The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity, and signatory non-repudiation.”

The definition above defines digital signature as:

- Data-dependent bit pattern

- Generated by an agent as a function of a file, message, or other form of data block
- Another agent can access the data block and its associated signature and verify that:
    1. The data block has been signed by the alleged signer
    2. The data block has not been altered since the signing
- The signer cannot repudiate the signature

There are three digital signatures algorithms:

- Digital Signature Algorithm (DSA): The original NIST-approved algorithm that is based on the difficulty of computing discrete logarithms
- RSA Digital Signature Algorithm: Based on the RSA public-key algorithm
- Elliptic Curve Digital Signature Algorithm (ECDSA): Based on elliptic-curve cryptography

Digital Signature Generation and Verification

Suppose Bob wants to send a message to Alice. He wants Alice to be certain that the message is indeed from him. So, Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message. The hash value will be added with Bob's private key, and it will serve as input to a digital signature generation algorithm that produces a short block that functions as a digital signature. Bob sends the message with the signature attached. When Alice receives the message plus signature, she:

1. Calculates a hash value for the message
2. Provides the hash value and Bob's public key as inputs to a digital signature verification algorithm

- If the algorithm returns the result that the signature is valid, Alice assured that the message must have been signed by Bob

It is impossible to alter the message without access to Bob's private key that is Bob only the one who have it. So, the message is authenticated both in terms of source and in terms of data integrity. But wait that process does not provide confidentiality because the message being sent is safe from alteration, but not safe from eavesdropping that is transferred in cleartext. Also, any observer can decrypt the message by using the sender's or Bob's public key.

RSA Digital Signature Algorithm

The core purpose of RSA digital signature algorithm is to encrypt the hash of the message to be signed using RSA. It will first modify the hash value to enhance security. There are several approaches, one of them is RSA Probabilistic Signature Scheme (RSA-PSS) that is the latest of the RSA schemes and recommended by the RSA Laboratories. The steps of RSA-PSS as follows:

1. Generate a hash value, or message digest, $mHash$ from the message M to be signed
2. Pad $mHash$ with a constant value padding1 and pseudorandom value salt to form M'
3. Generate hash value H from M'
4. Generate a block DB consisting of a constant value padding 2 and salt
5. Use the mask generating function MGF, which produces a randomized output from input H of the same length as DB

6. Create the encoded message (EM) block by padding H with the hexadecimal constant BC and the XOR of H and DB
7. Encrypt EM with RSA using the signer's private key

The objective with this algorithm is to make it more difficult for an adversary to find another message that maps to the same message digest as a given message or to find two messages that map to the same message digest. Because the salt changes with every use, signing the same message twice using the same private key will yield two different signatures.

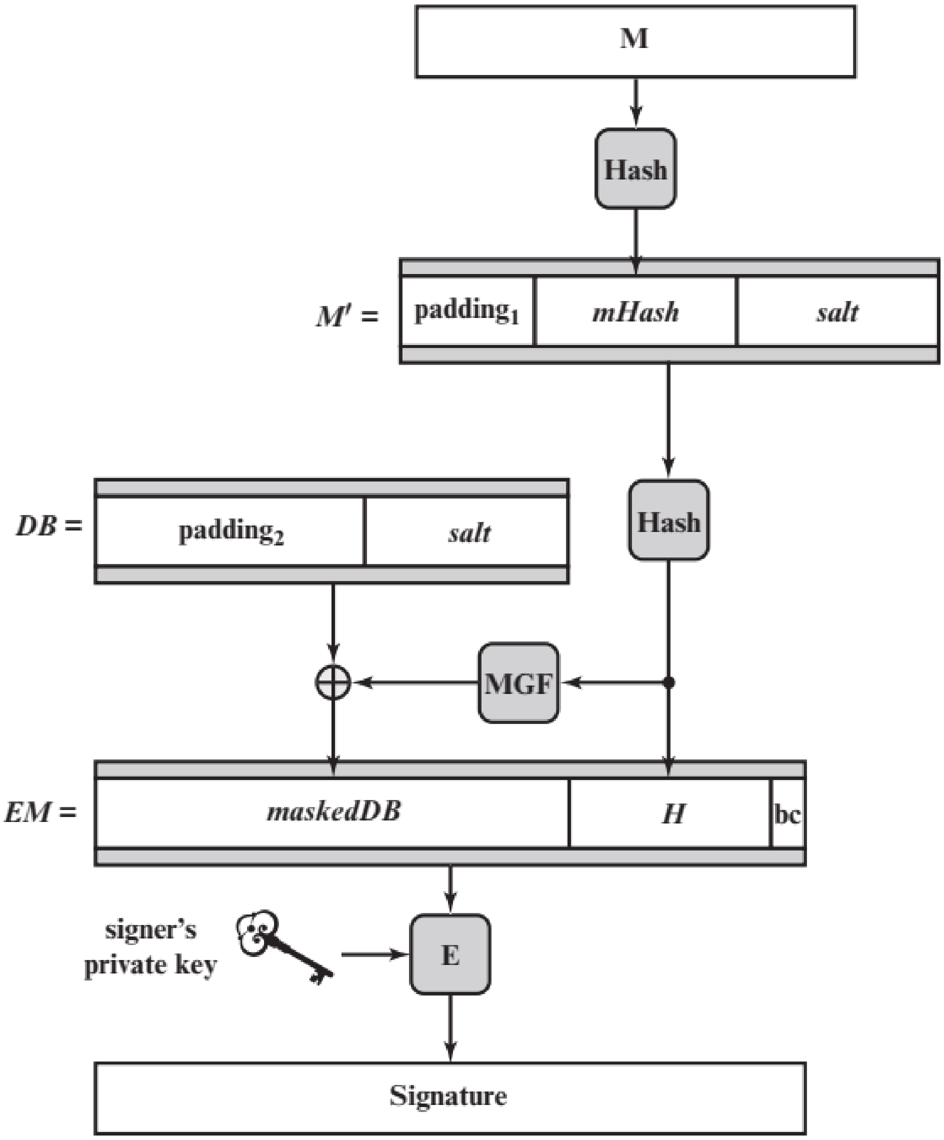


Figure 3.16 RSA-PSS Encoding and Signature Generation

References & Useful Resources

- Design and analysis of security protocols course, UAE University, Dr. Khalid Shuaib slides
- William Stallings, Network Security Essentials Applications and Standards, Sixth Edition, 2014, Pearson. ISBN-13: 9780133370430