UNIT- IV:
Data Integrity, Digital Signature Schemes & Key Management
Message Integrity and Message Authentication, Cryptographic Hash Functions, Digital Signature, Key Management.

Ch-11 – Message Integrity and Message Authentication

1. Message Integrity

The cryptography systems that we have studied so far provide *secrecy* or *confidentiality* but not *integrity*. However there are occasions where we may not even need secrecy but instead must have integrity.

1.1 Message and Message Digest

The electronic equivalent of the document and fingerprint is the message and message digest pair. To preserve the integrity of the message, the message is passed through an algorithm called a **Cryptographic hash function**. The function creates a compressed image of the message that can be used like a fingerprint. Figure 11.1 shows the message, cryptographic hash function, and **message digest**.

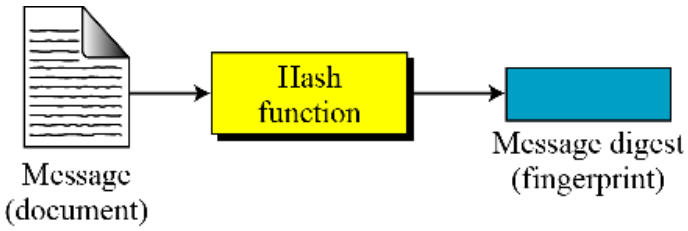


Figure 11.1 Message and Message digest

> The message digest needs to be safe from change.

1.2 Checking Integrity

To check the integrity of a message, or document, we run the cryptographic hash function again and compares the new message digest with the previous one. If both are same, we are sure that the original message has not been changed. Figure 11.2 shows the idea.

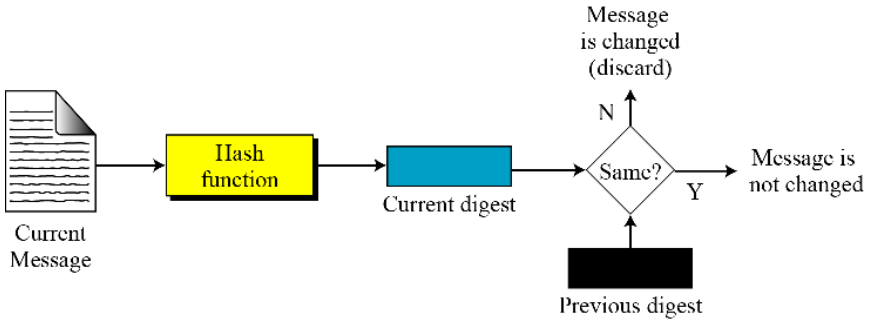


Figure 11.2 Checking integrity

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

1.3 Cryptographic Hash Function Criteria

A cryptographic hash function must satisfy three criteria:

1. Preimage resistance
2. Second preimage resistance
3. Collision resistance.

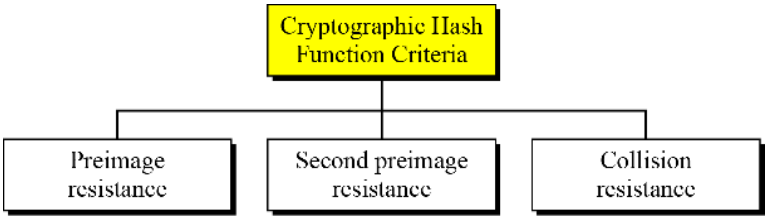
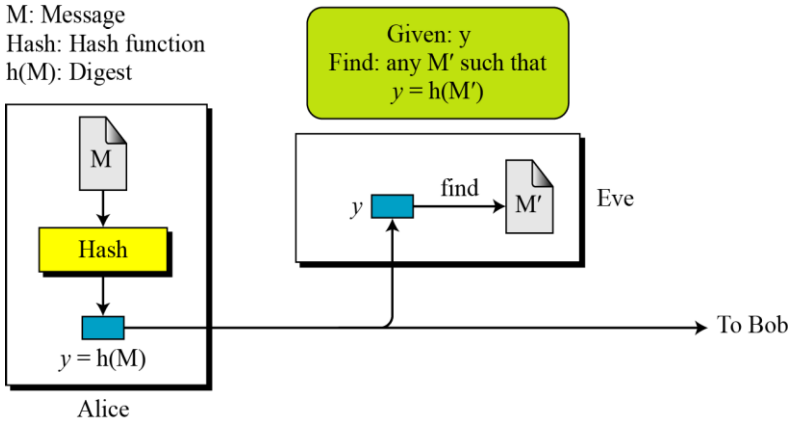


Figure 11.3 Criteria of a cryptographic hash function

Preimage Resistance

- This property of the hash function implies given a hashed value it should difficult for an adversary to compute the preimage of the hashed value.
- Given a hash function h and y=h(M) , it must be extremely difficult for Eve to find any message, M' such that y=h(M').

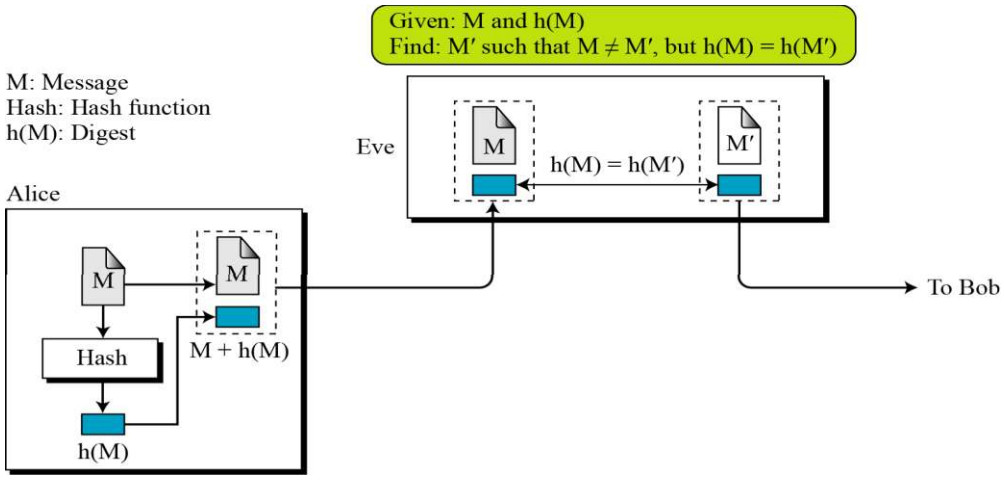


Thus mathematically, a hash function h is said to be preimage resistant, if given the value of y = h(x), for some x it is difficult to compute the value of any x' such that h(x') = y.

Second Preimage Resistance

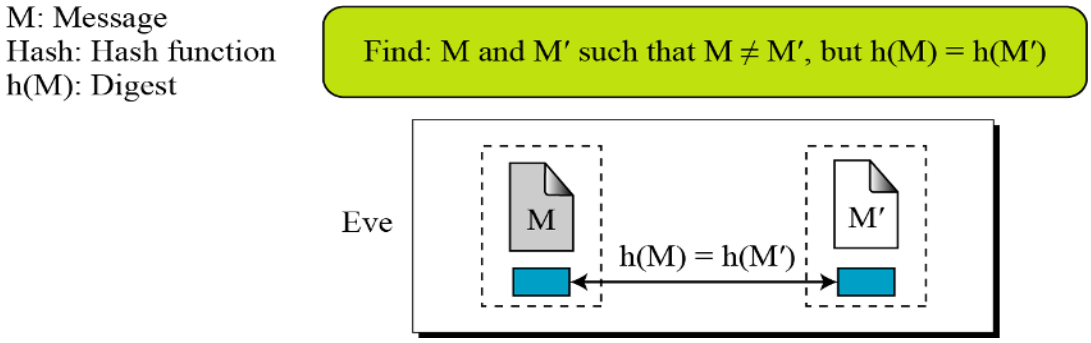
- The second criterion, second Preimage resistance, ensure that a message cannot easily be forged.
- If Alice creates a message and a digest and sends both to Bob, this criterion ensures that Eve cannot easily create another message that hashes to the exact same digest.
- Eve intercepts a message M and its digest h(M).
- She creates another message M' ≠ M, but h(M)=h(M').
- Eve sends the M' and h(M') to Bob. Eve has forged the message.

Downloaded by TONY 562 (tony2802b@gmail.com)



Collision Resistance

- The third criterion, collision resistance, ensures that Eve cannot find two messages that hash to the same digest.
- Here the adversary can create two messages and hashed to the same digest.
- For the moment, suppose two different wills can be created, that hash to the same digest.



2. Random Oracle Model

The **Random oracle model,** which was introduced in 1993 by Bellare and Rogaway, is an ideal mathematical model for a hash function. A function based on this model behaves as follows:

1. When a new message of any length is given, the oracle creates and gives a fixed length message digest that is random string of 0s and 1s. The oracle records the message and the message digest.
2. When a message is given for which a digest exists, the oracle simply gives the digest in the record.
3. The digest for a new message needs to be chosen independently from all previous digest. This implies that the oracle cannot use a formula or an algorithm to calculate the digest.

2.1 Pigeonhole principle

- The first thing we need to be familiar with no understanding the analysis of the Random Oracle Model is the **pigeonhole principle**: if n pigeonholes are occupied by n+1 pigeons, then at least one pigeonhole is occupied by two pigeons.
- The generalized versions of the pigeonhole principle is that n pigeonholes are occupied by kn+1 pigeons, then at least one pigeonhole is occupied by k+1 pigeons.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

- Because the whole idea of hashing dictates that the digest should be shorter than the message, according to the pigeonhole principle there can be collisions.
- In other words, there are some digests that corresponds to more than one message; the relationship between the possible messages and possible digests is many-to-one.

2.2 Birthday Problems

- The second thing we need to know before analyzing the Random Oracle Model is the famous **birthday problems**.
- Four different birthday problems are usually encountered in the probability courses.
- The third problem, sometimes referred to as birthday paradox, is the most common one in the literature.

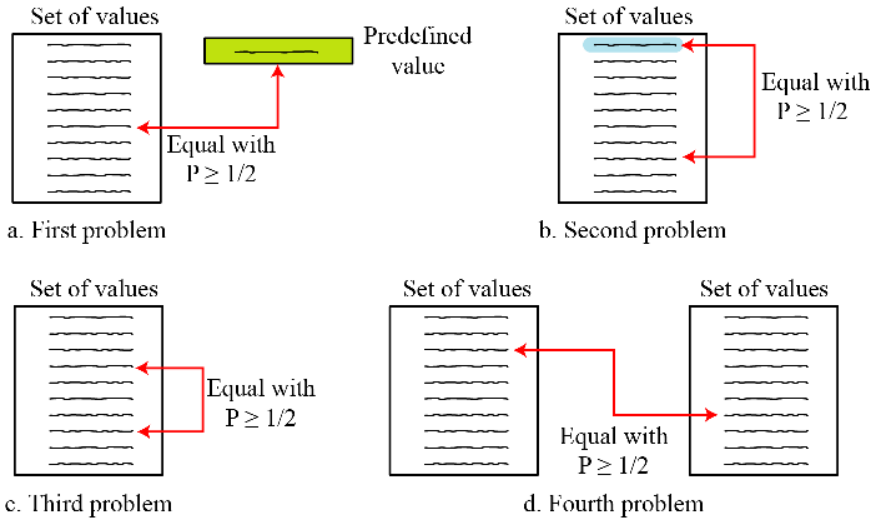


Figure 11.4 Four birthday problems

Description of problems

- **Problem1:** what is the minimum number, k, of students in a classroom such that it is likely that at least one student has a predefined birthday? This problem can be generalized as follows. We have a uniformly distributed random variable with N possible values (between 0 and N-1).What is the minimum number of instances k, such that it is likely that at least one instance is equal to a predefined value.
- **Problem 2:** what is the minimum number, k, of students in a classroom such that it is likely that at least one student has the same birthday as the student selected by the professor? This problem can be generalized as follows. We have a uniformly distributed random variable with N possible values (between 0 and N-1). What is the minimum number of instances, k such that it is likely that at least one instance is equal to the select one?
- **Problem 3:** what is the minimum number, k, of students in a classroom such that it is likely that at least two students have the same birthday? The problems can be generalized as follows. We have a uniformly distributed random variable with N possible values (between 0 and N-1). What is the minimum number of instances, k, such that it is likely that at least two instances are equal?
- **Problem 4:** we have two classes, each with k students. What is the minimum value of k such that it is likely that at least one student from the first classroom has the same birthday as a student from the second classroom? This problem can be generalized as follows. We have a uniformly distributed random variable with N possible values (between 0 and N-1). We generate two sets of random values each with k instances. What is the minimum number of

Downloaded by TONY 562 (tony2802b@gmail.com)

k, such that it is likely that at least one instance from the first set is equal to one instance in the second set.

Comparison

- The value of K in problems 1 or 2 is proportional to N, the value of k in problems 3 or 4 is proportional to $N^{1/2}$.
- The first two problems are related to the preimage and second preimage attacks; the third and fourth problems are related to the collision attack.
- The comparison shows it is much more difficult to launch preimage or second preimage attack than to launch a collision attack.

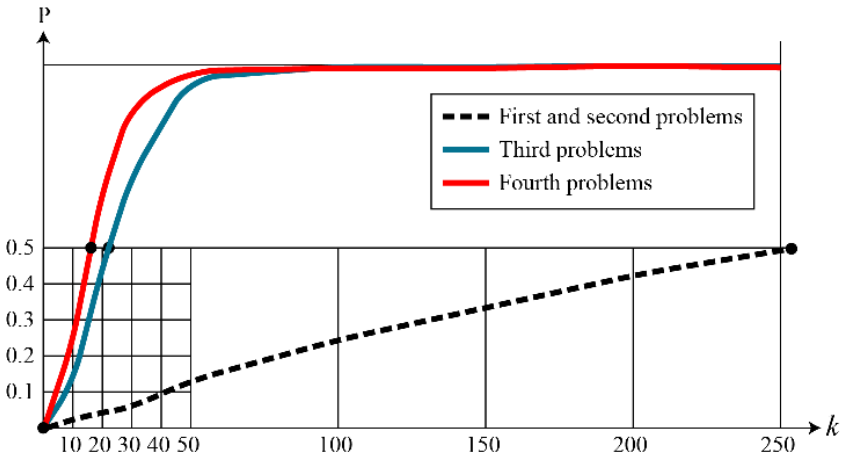


Figure 11.5 Graph of four birthday problems

2.3 Attacks on random oracle model

- To better understand the nature of hash functions and the importance of the Random Oracle Model, consider how Eve can attack a hash function created by the oracle.
- Suppose that the hash function creates a digest of n bits.
- The digest can be thought of as a random variable uniformly distributed between 0 and N-1 in which $N=2^n$.

Preimage attack

- Eve has intercepted a digest D=h(M). She wants to find a message M' such that D=h(M').

```
Preimage_Attack (D)
{
   for (i = 1 to k)
   {
        create (M [i])
        T ← h(M [i])                    // T is a temporary digest
        if (T = D) return M [i]
   }
   return failure
}
```

The difficulty of a preimage attack is proportional to 2^n

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Second Preimage attack

- Eve has intercepted a digest D=h(M) and the corresponding message M. She wants to find another message M' so that h(M')=D.
- Eve can create a list of k-1 messages.

```
Second_Preimage_Attack (D, M)
{
   for (i = 1 to k –1)
   {
        create (M [i])
        T ← h (M [i])                    // T is a temporary digest
        if (T = D) return M [i]
   }
   return failure
}
```

The difficulty of second preimage attack is proportional to 2^n.

Collision attack

- Eve needs to find new messages M and M' such that h(M)=h(M').Eve can create a list of k messages

```
Collision_Attack
{
   for (i = 1 to k )
   {
        create (M[i])
        D[i] ← h (M[i])                  // D [i] is a list of created digests
        for (j = 1 to i – 1)
        {
           if (D[i] = D[j]) return (M[i] and M[j])
        }
   }
   return failure
}
```

The difficulty of a collision attack is proportional to $2^{n/2}$.

Attack	*Value of k with P=1/2*	*Order*
Preimage	$k \approx 0.69 \times 2^n$	2^n
Second preimage	$k \approx 0.69 \times 2^n + 1$	2^n
Collision	$k \approx 1.18 \times 2^{n/2}$	$2^{n/2}$
Alternate collision	$k \approx 0.83 \times 2^{n/2}$	$2^{n/2}$

Table 11.1 Levels of difficulties for each type of attack.

3. Message Authentication

- A message digest guarantees the integrity of the message. It guarantees that the message has not been changed.

Downloaded by TONY 562 (tony2802b@gmail.com)

- A message digest, cannot authenticate the sender of the message.
- When Alice sends a message to Bob, Bob needs to know if the message is coming from Alice.
- To provide a message authentication, Alice needs to provide proof that it is Alice sending message and not an imposter.
- The digest created by a cryptographic hash function is normally called a modification Detection code (MDC).
- The code can detect any modification in the message.

What we need for message authentication is a message authentication code (MAC).

3.1 Modification Detection Code

- A modification detection code (MDC) is a message digest that can prove the integrity of the message: that message has not been changed.
- If Alice needs to send a message to Bob and be sure that the message will not change during transmission, Alice can create a message digest MDC and send both the message and the MDC to Bob.
- Bob can create a new MDC from the message and compare the received MDC and the new MDC.
- If they are same, the message has not been changed. Figure 11.6 shows the idea.

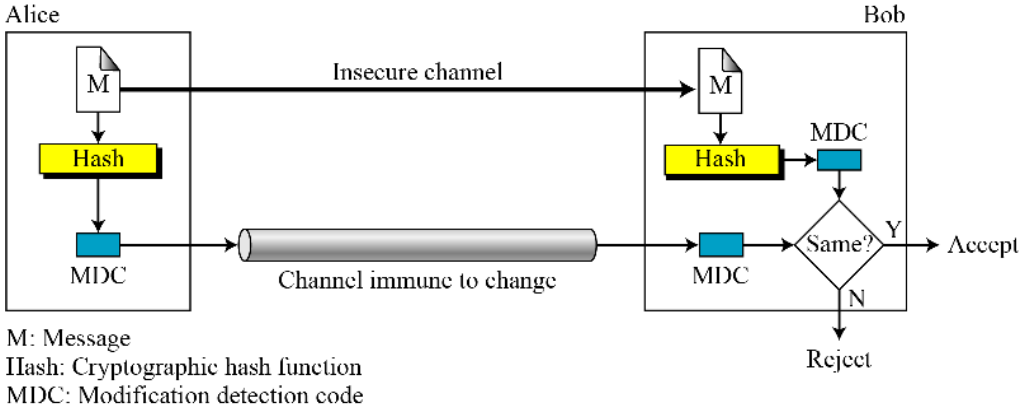


Figure 11.6 Modification detection code (MDC)

3.2 Message Authentication Code (MAC)

To ensure that the integrity of the message and the data origin authentication—that Alice is the originator of the message, not somebody else- we need a change modification detection code (MDC) to message authentication code (MAC). The difference between that a MDC and a MAC is that the second includes a secret between Alice and Bob—for example, a secret key is that Eve does not possess. Figure 11.7 shows the idea.

Alice uses a hash function to create a MAC from the concatenation of the key and the message, h(K | M). She sends a message and the MAC to Bob over the insecure channel. Bob separates the message from MAC. He then makes a new MAC from the concatenation of the message and the secret key. Bob then compares the newly created MAC with the one received. If the two MAC's match, the message is authentic and has not been modified by adversary.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

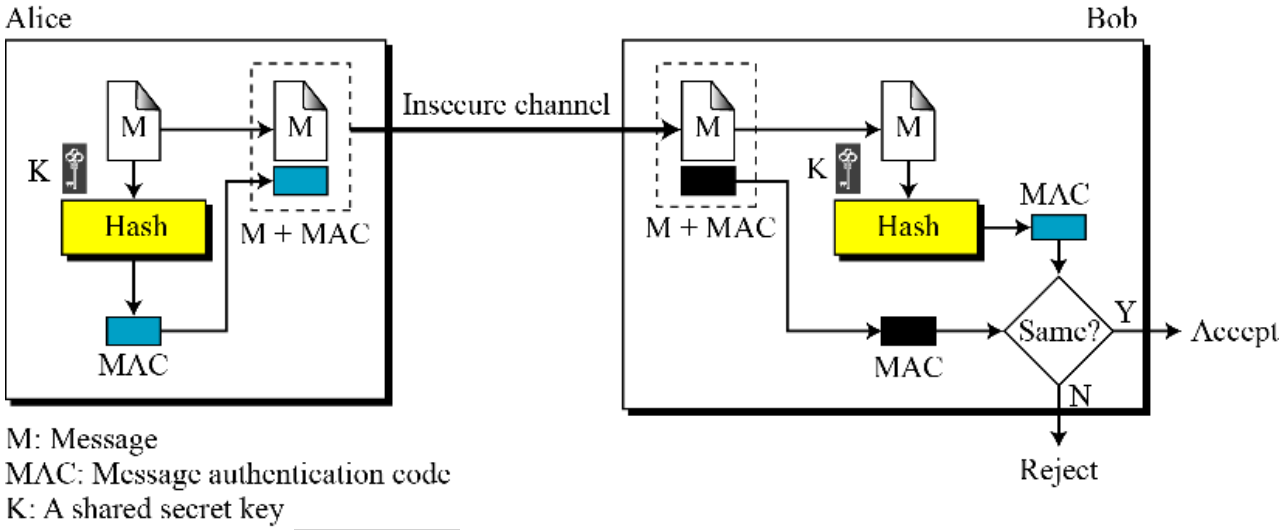


Figure 11.7 Message Authentication Code (MAC)

Security of MAC

Suppose Eve has intercepted the message M and the digest h(K|M). How Eve can forge a message without knowing the secret key? There are three possible cases:

1. If the size of the key allows exhaustive search, Eve may prepend all possible keys at the beginning of the message and make a digest of the (K|M) to find the digest equal to the one intercepted. She then knows the key and can successfully replace the message with a forged message of her choosing.
2. The size of the key is normally very large in MAC.
3. Given some pairs of messages and their MAC's, Eve can manipulate them to come with a new message and its MAC.

The security of the MAC depends on the security of underlying hash algorithm.

Nested MAC

To prove the security of MAC, nested MAC's were designed in which hashing is done in two steps: In the first step, the key is concatenated with the message and is hashed to create an immediate digest. In the second step, the key is concatenated with an intermediate digest to create a final digest.

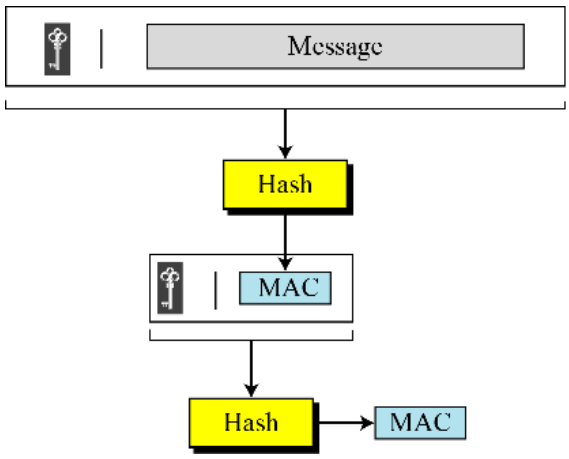


Figure 11.8 Nested MAC

Downloaded by TONY 562 (tony2802b@gmail.com)

HMAC

The implementation of HMAC (Hashed MAC) is more complex than the simplified nested MAC shown in Figure 11.8. There are additional features such as padding. Figure 11.9 shows the details. We go through the steps:

1. The message is divided into N blocks, each of b bits.
2. The secret key is left-padded with 0's to create a b-bit key.
3. The result of step-2 is exclusive-ored with a constant called ipad (input pad) to create a *b*-bit block. The value of ipad is the *b*/8 repetition of the sequence 00110110 (36 in hexadecimal).
4. The resulting block is prepended to the N-block message. The result is N+1 blocks.
5. The result of step-4 is hashed to create an n-bit digest. We call the digest the intermediate HMAC.
6. The intermediate n-bit HMAC is left padded with 0s to make a *b*-bit block.
7. Steps 2 and 3 are repeated by a different constant opad (output pad). The value of opad is the *b*/8 repetition of the sequence 01011100 (5C in hexadecimal).
8. The result of step 7 is prepended to the block of step 6.
9. The result of step 8 is hashed with the same hashing algorithm to create the final n-bit HMAC.

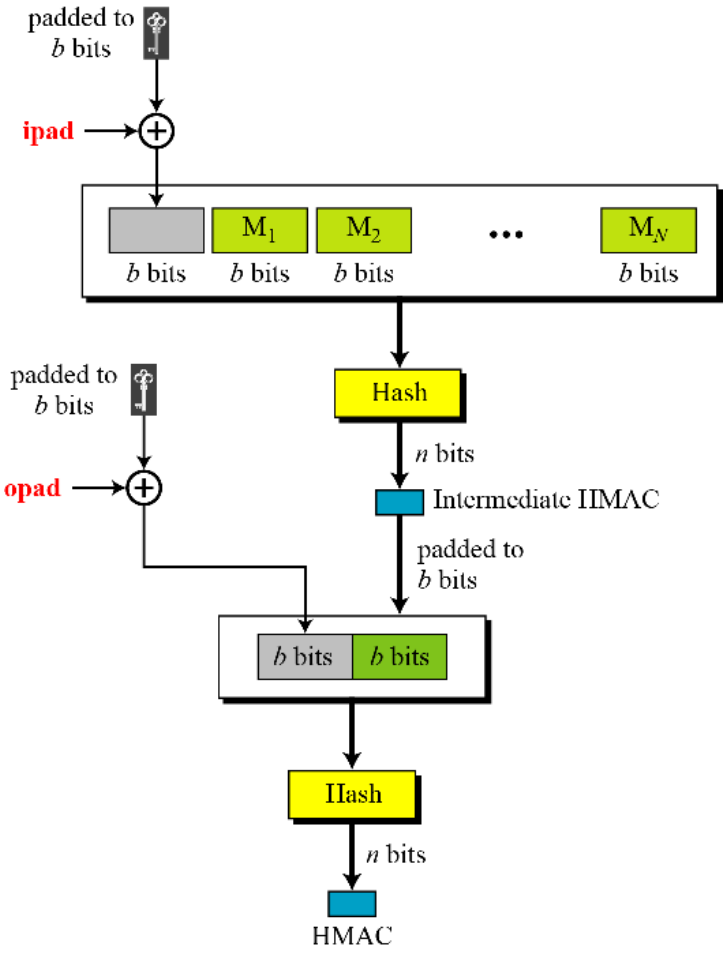


Figure 11.9 Details of HMAC

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

CMAC (Cipher based MAC)

The idea is to create one block of MAC from N blocks of plaintext using a symmetric-key cipher N times. Figure 11.10 shows the idea.

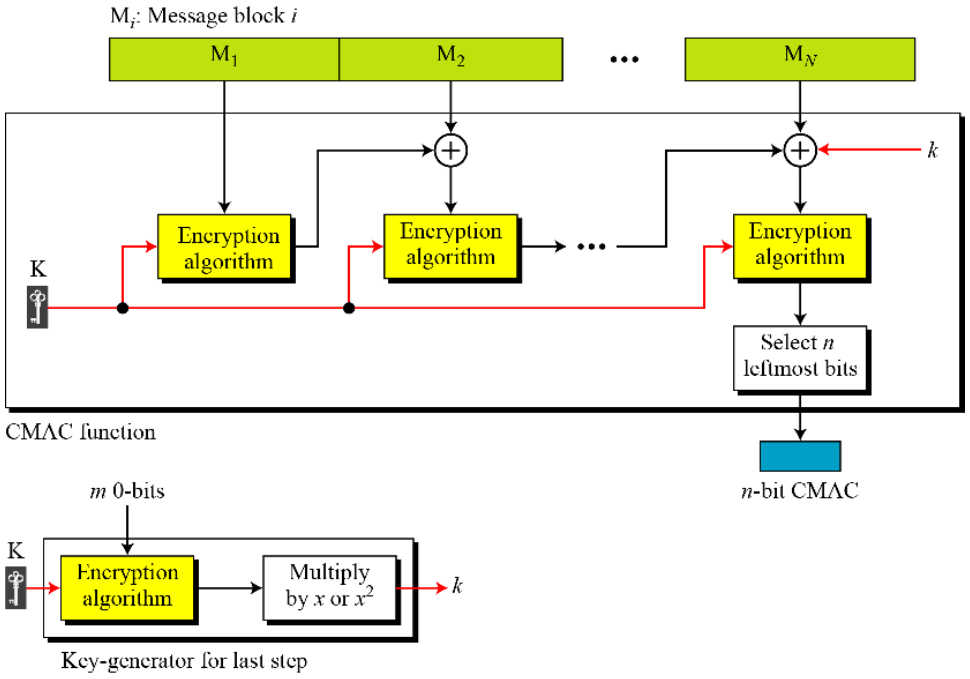


Figure 11.10 CMAC

- The message is divided into *N* blocks, each m bits long.
- The size of the CMAC is n-bits. If the last block is not m-bits, it is padded with a 1- bit followed by enough 0-bits to make it m bits.
- The first block of the message is encrypted with the symmetric key to create an m-bit block of encrypted data.
- This block is XORed with the next block and the result is encrypted again to create a new m-bit block.
- The process continues until the last block of the message is encrypted.
- The n leftmost bit from the last block is the CMAC.

In addition to the symmetric key, K, CMAC also uses another key, k, which is applied only at the last step. This key is derived from the encryption algorithm with plaintext of m 0-bits using the cipher key, K. The result is then multiplied by x if no padding is applied and multiplied by x^2 if padding is applied. The multiplication is in GF(2^m) with the irreducible polynomial of degree m selected By the particular protocol used.

Downloaded by TONY 562 (tony2802b@gmail.com)

Ch-12 – Cryptographic Hash Functions

As discussed in Chapter 11, a cryptographic hash function takes a message of arbitrary length and creates a message digest of fixed length. The ultimate goal of this chapter is to discuss the details of the two most promising cryptographic hash algorithms-Sha-512 and Whirlpool.

1. Iterated Hash Function

All cryptographic hash functions need to create a fixed-size digest out of a variable-size message. Creating such a function is best accomplished using iteration. The fixed size input function is referred to as a *compression function*. It compresses an n-bit string to create an m-bit string where n is normally greater than m. the scheme is referred to as an ***iterated cryptographic hash function***.

Merkle-Damgard Scheme The Merkle-Damgard scheme is an iterated hash function that is collision resistant if the compression function is collision resistant. The scheme is shown in figure 12.1.

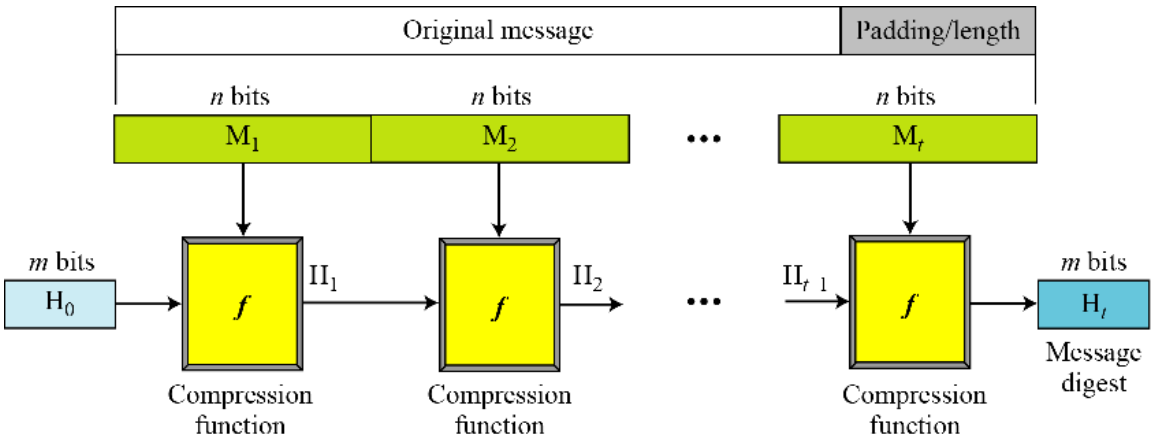


Figure 21.1 Merkle-Damgard scheme

The scheme uses the following steps:

1. The message length and padding are appended to the message to create an augmented message that can be evenly divided into blocks of n bits, where n is the size of the block to be processed by the compression function.
2. The message is then considered as t blocks, each of n bits. We call each block M_1, M_2, …, M_t. We call the digest created at t iterations H_1, H_2, …, H_t.
3. Before starting the iteration, the digest H_0 is set to a fixed value, normally called IV (initial value or initial vector).
4. The compression function at each iteration operates on H_{i-1} and M_i to create a new H_i. In other words, we have $H_i = f(H_{i-1}, M_i)$, where *f* is the compression function.
5. Ht is the cryptographic hash function of the original message, that is h(M).

1.1 Two Groups of Compression Functions

The Merkle-Damgard scheme is the basis for many cryptographic hash functions today. Three is a tendency to use two different approaches in designing a hash function. In the first approach, the compression function is made from scratch: it is particularly designed for this purpose. In the second approach, a symmetric-key block cipher serves as a compression function.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Hash Functions Made from Scratch

A set of cryptographic hash functions uses compression functions that are made from scratch. These compression functions are specifically designed for the purposes they serve.

Message digest (MD) Several hash algorithms were designed by Ron Rivest. These are referred to as MD2, MD4, and MD5, where MD stands for Message Digest. The last version, MD5, is a strengthened version of MD4 that divides the message into blocks of 512 bits and creates a 128-bit digest. It turned out that a message digest of size 128 bits is too small to resist collision attack.

Secure Hash Algorithm (SHA) The Secure Hash Algorithm (SHA) is a standard that was developed by the National Institute of Standard and Technology (NIST). There are four versions: SHA-224, SHA-256, SHA-384, AND SHA-512. Table 12.1 lists some of the characteristics of these versions.

Characteristics	*SHA-1*	*SHA-224*	*SHA-256*	*SHA-384*	*SHA-512*
Maximum Message size	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$	$2^{128} - 1$	$2^{128} - 1$
Block size	512	512	512	1024	1024
Message digest size	160	224	256	384	512
Number of rounds	80	64	64	80	80
Word size	32	32	32	64	64

Table 12.1 Characteristics of Secure Hash Algorithms (SHAs)

Hash Functions Based on Block Ciphers

An iterated cryptographic hash function can use a symmetric-key block cipher as a compression function. The whole idea is that there are several secure symmetric-key block ciphers, such as triple DES or AES, that can be used to make a one-way function instead of creating a new compression function. The block cipher in this case only performs encryption. Several schemes have been proposed. We later describe one of the most promising, Whirlpool.

Rabin Scheme The iterated hash function proposed by Rabin is very simple. The Rabin scheme is based on the Merkle-Damgard scheme. The compression function is replaced by any encrypting cipher. The message block is used as the key; the previously created digest is used as the plaintext. The ciphertext is the new message digest. Note that the size of the digest is the size of data block cipher in the underlying cryptosystem. For example, if DES is used as the block cipher, the size of the digest is only 64 bits. Although the scheme is very simple, it is subject to a meet-in-the-middle attack discussed in chapter 6, because the adversary can use the decryption algorithm of the cryptosystem. Figure 12.2 shows the Rabin scheme.

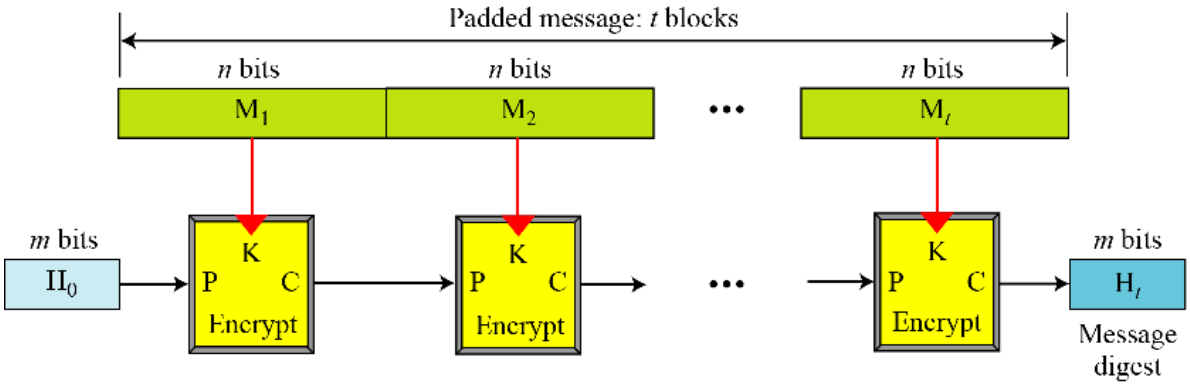


Figure 12.2 Rabin scheme

Downloaded by TONY 562 (tony2802b@gmail.com)

Davies-Meyer Scheme The Davies-Meyer scheme is basically the same as the Rabin scheme except that it uses forward feed to protect against meet-in-the-middle attack. Figure 12.3 shows Davies-Meyer scheme.

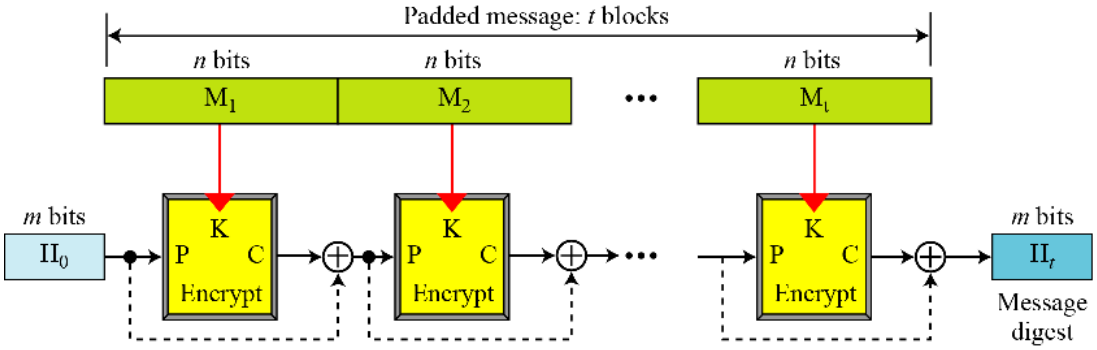


Figure 12.3 Davies-Meyer scheme

Matyas-Meyer-Oseas Scheme The Matyas-Meyer-Oseas scheme is a dual version of the Davies-Meyer scheme: the message block is used as the key to the cryptosystem. The scheme can be used if the data block and the cipher key are the same size. For example, AES is a good candidate for this purpose. Figure 12.4 shows the Matyas-Meyer-Oseas scheme.

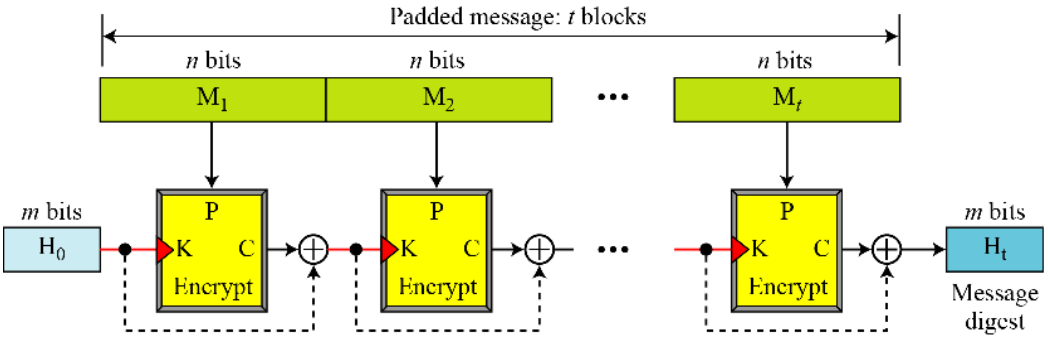


Figure 12.4 Matyas-Meyer-Oseas scheme

Miyaguchi-Preneel Scheme The Miyaguchi-Preneel scheme is an extended version of Matyas-Meyer-Oseas. To make the algorithm stronger against attack, the plaintext, the cipher key, and the ciphertext junction. Figure 12.5 shows the Miyaguchi-Preneel scheme.

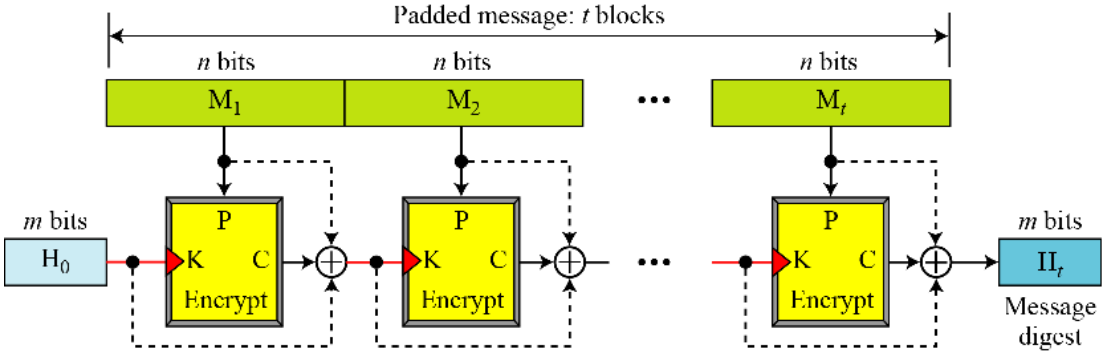


Figure 12.5 Miyaguchi-Preneel scheme

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

2. Description of MD Hash Family

Hash functions of the MD-family are iterated hash functions and follow the MD-design principle. Further, these hash functions share a common structure of the compression function. The compression function consists of two major parts which are the message expansion and the consecutive evaluation of a number of similar operations, called steps. Three steps are usually grouped together into 3-5 rounds. After the last step of the compression function, the input chaining variables are added to the output, which complicates the inversion of the compression function.

3. WHIRLPOOL

Whirlpool is designed by Vincent Rijmen and Paulo S. L. M. Barreto. Whirlpool is an iterated cryptographic hash function, based on the Miyaguchi-Preneel scheme that uses a symmetric-key block cipher in place of the compression function. The block cipher is modified AES cipher that has been tailored for this purpose. Figure 12.6 shows the Whirlpool hash function.

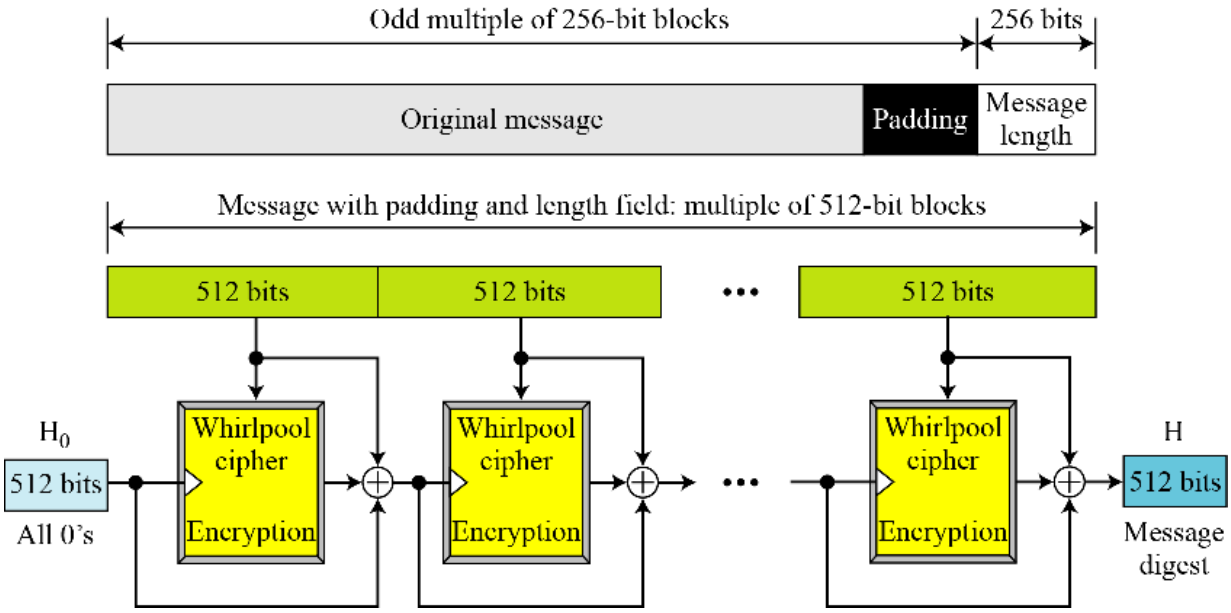


Figure 12.6 Whirlpool hash function

Preparation Before starting the hash algorithm, the message needs to be prepared for processing. Whirlpool required that the length of the original message be less than 2^{256} bits. A message needs to be padded before being processed. The padding is a single 1-bit followed by the necessary numbers of 0-bits. To make the length of the padding an odd multiple of 256 bits. After padding, a block of 256 bits is added to define the length of the original message. This block is treated as an unsigned number.

After padding and adding the length field, the augmented message size is an even multiple of 256 bits or a multiple of 512 bits. Whirlpool creates a digest of 512 bits from a multiple of 512-bit block message. The 512-bit digest, H_0, is initialized to all 0's. This value because the cipher key for encrypting the first block. The ciphertext resulting from encryption each block become the cipher key for the next block after being exclusive-ored with the previous cipher key and the plaintext block. The message digest is the final 512-bit ciphertext after the last exclusive-or operation.

3.1 Whirlpool Cipher

The Whirlpool cipher is a non-Feistel cipher like AES that was mainly designed as a block cipher to be used in a hash algorithm. Instead of giving the whole description of this cipher, we just assume

Downloaded by TONY 562 (tony2802b@gmail.com)

that the reader is familiar with AES from chapter 7. Here the Whirlpool cipher is compared with the AES cipher and their differences are mentioned.

Rounds Whirlpool is a round cipher that uses 10 rounds. The block size and key size are 512 bits. The cipher uses 11 round keys, K_0 to K_{10}, each of 512 bits. Figure 12.7 shows the general design of the Whirlpool cipher.

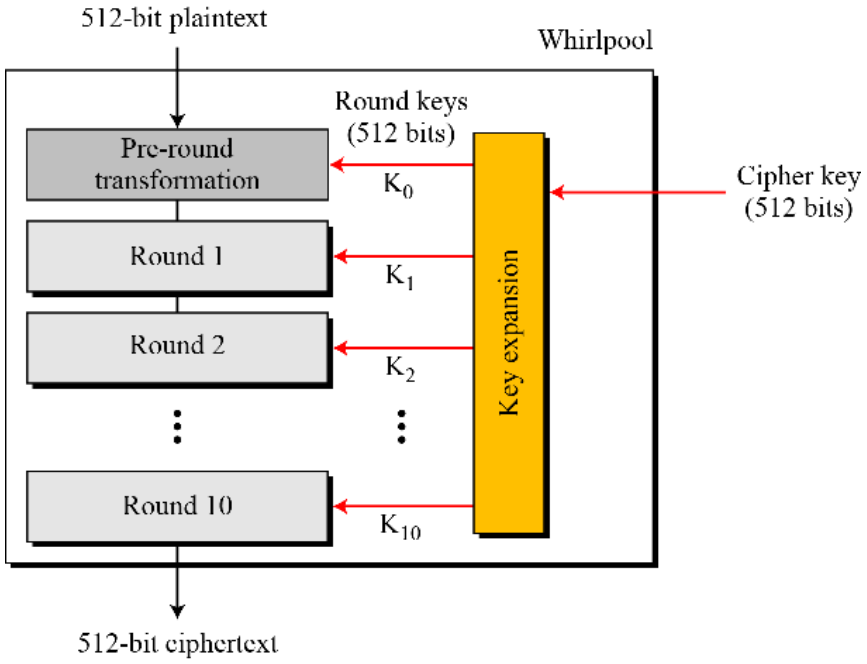


Figure 12.7 General idea of the Whirlpool cipher

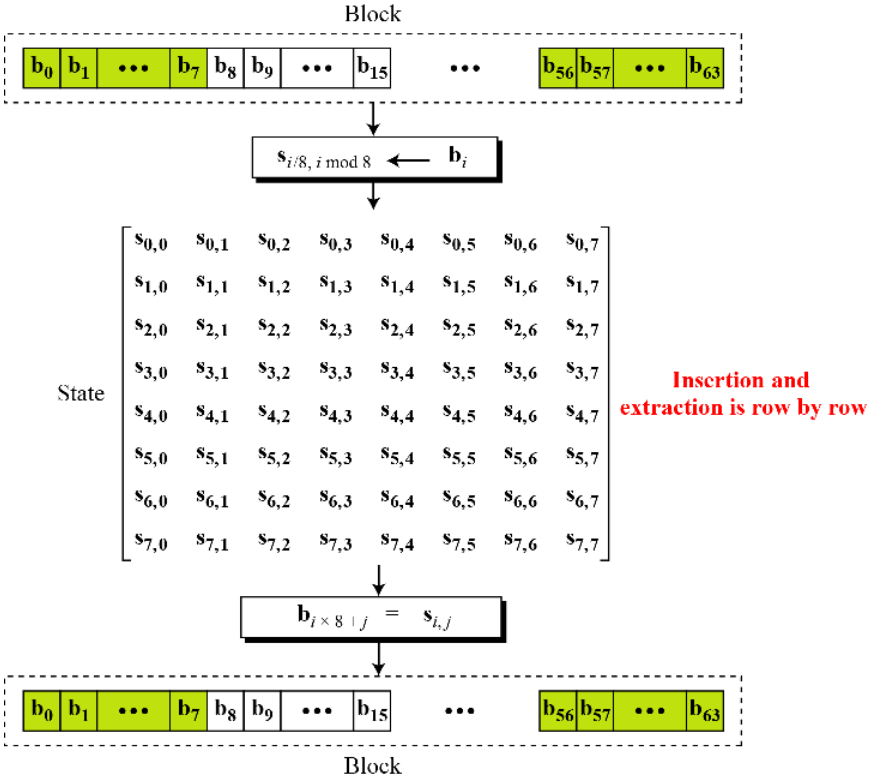


Figure 12.8 Block and state in the Whirlpool cipher

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

States and Blocks Like the AES cipher, the Whirlpool cipher uses states and blocks. However, the size of the block or state is 512 bits. A block is considered as a row matrix of 64 bytes; a state is considered as a square matrix of 8 x 8 bytes. Unlike AES, the block-to-state or state-to-block transformation is done row by tow. Figure 12.8 shows the block, the state, and the transformation in the Whirlpool cipher.

Structure of each round Figure 12.9 shows the structure of each round. Each round uses four transformations.

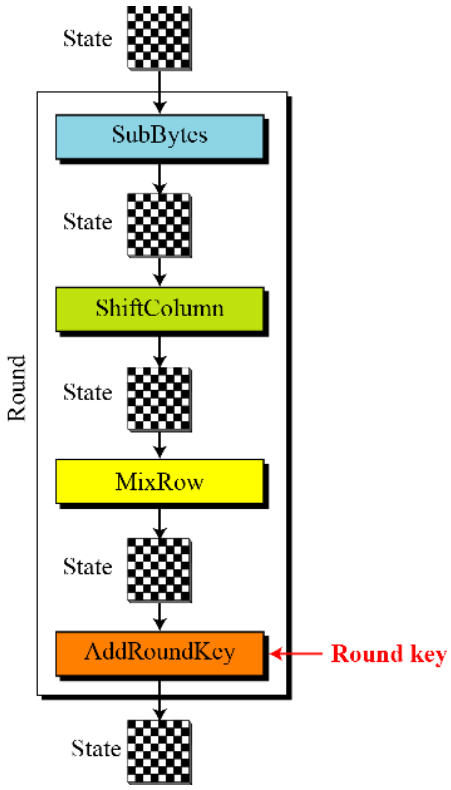


Figure 12.9 Structure of each round in the Whirlpool cipher

SubBytes Like in AES, SubBytes provide a nonlinear transformation. A byte is represented as two hexadecimal digits. The left digit defines the row and the right digit defines the column of the substitution table. The two hexadecimal digits at the junction of the row and the column are the new byte. Figure 12.10 shows the idea.

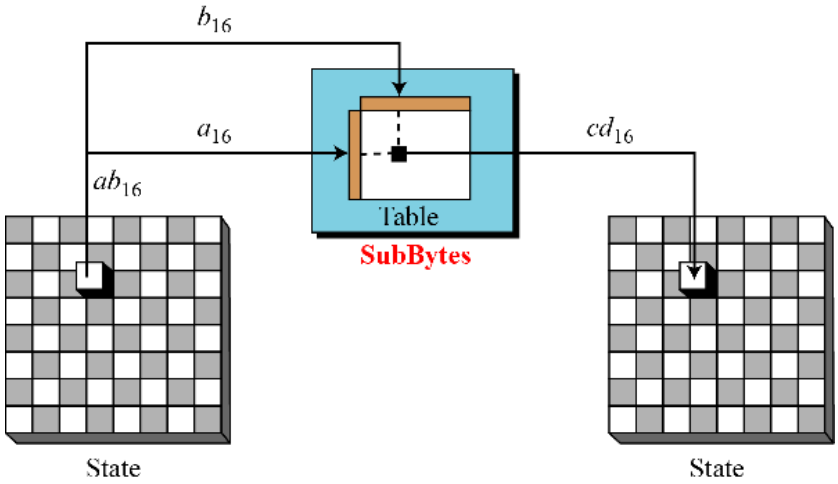


Figure 12.10 SubBytes transformations in the Whirlpool cipher

Downloaded by TONY 562 (tony2802b@gmail.com)

In the SubBytes transformation, the state is treated as an 8 x 8 matrix of bytes. Transformation is done one byte at a time. The contents of each byte are change, but the arrangement of the bytes in the matrix remains the same. In the process, each byte is transformed independently; we have 64 distinct byte-to-byte transformations.

Table 12.2 shows the substitution table (S-Box) for SubBytes transformation. The transformation definitely provides confusion effect. For example, two bytes $5A_{16}$ and $5B_{16}$, which differ only in one bit (the rightmost bit), are transformed to $5B_{16}$ and 88_{16},which differ in five bits.

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	18	23	C6	E8	87	B8	01	4F	36	A6	D2	F5	79	6F	91	52
1	16	BC	9B	8E	A3	0C	7B	35	1D	E0	D7	C2	2E	4B	FE	57
2	15	77	37	E5	9F	F0	4A	CA	58	C9	29	0A	B1	A0	6B	85
3	BD	5D	10	F4	CB	3E	05	67	E4	27	41	8B	A7	7D	95	C8
4	FB	EF	7C	66	DD	17	47	9E	CA	2D	BF	07	AD	5A	83	33
5	63	02	AA	71	C8	19	49	C9	F2	E3	5B	88	9A	26	32	B0
6	E9	0F	D5	80	BE	CD	34	48	FF	7A	90	5F	20	68	1A	AE
7	B4	54	93	22	64	F1	73	12	40	08	C3	EC	DB	A1	8D	3D
8	97	00	CF	2B	76	82	D6	1B	B5	AF	6A	50	45	F3	30	EF
9	3F	55	A2	EA	65	BA	2F	C0	DE	1C	FD	4D	92	75	06	8A
A	B2	E6	0E	1F	62	D4	A8	96	F9	C5	25	59	84	72	39	4C
B	5E	78	38	8C	C1	A5	E2	61	B3	21	9C	1E	43	C7	FC	04
C	51	99	6D	0D	FA	DF	7E	24	3B	AB	CE	11	8F	4E	B7	EB
D	3C	81	94	F7	9B	13	2C	D3	E7	6E	C4	03	56	44	7E	A9
E	2A	BB	C1	53	DC	0B	9D	6C	31	74	F6	46	AC	89	14	E1
F	16	3A	69	09	70	B6	C0	ED	CC	42	98	A4	28	5C	F8	86

Table 12.2 SubBytes transformation table (S-Box)

ShiftColumns To provide permutation, Whirlpool uses the ShiftColumns transformation, which is similar to the ShiftRows transformation in AES, except that the columns instead of rows are shifted. Shifting depends on the position of the column. Column 0 goes through 0-byte shifting (no shifting), while column 7 goes through 7-byte shifting. Figure 12.11 shows the shifting transformation.

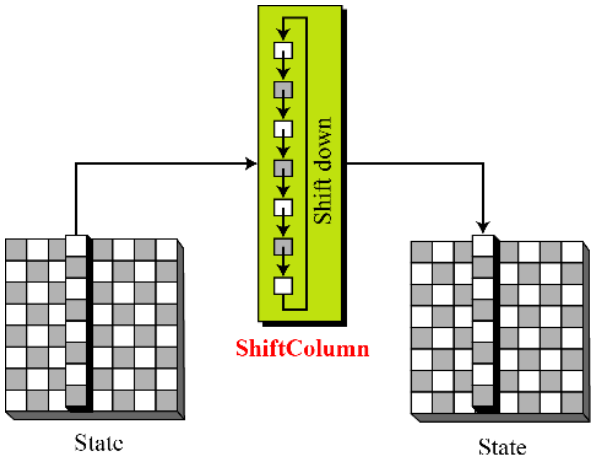


Figure 12.11 ShiftColumns transformation in the Whirlpool cipher

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

MixRows The MixRows transformation has the same effect as the MixColumns transformation in AES; it diffuses the bits. Figure 12.12 shows the MixRows transformation.

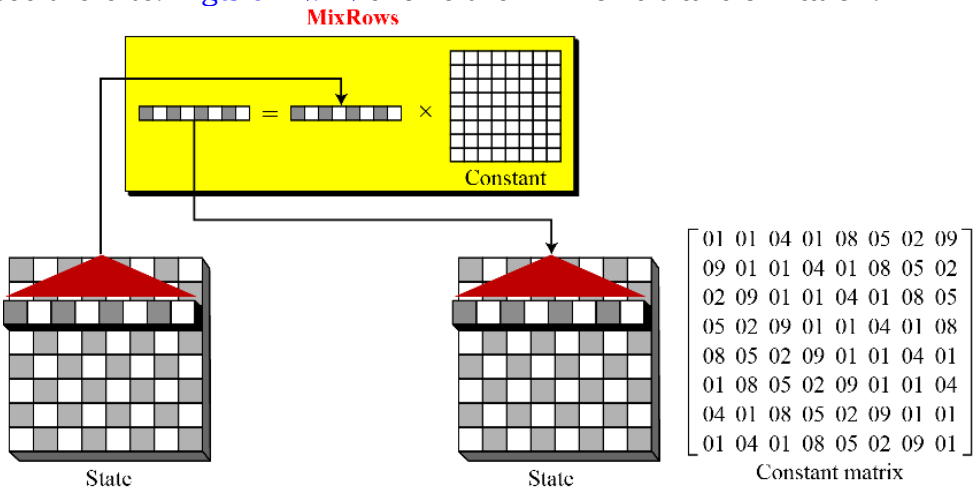


Figure 12.12 MixRows transformation in the Whirlpool cipher

The figure 12.12 shows multiplication of a single row by the constant matrix; the multiplication can actually be done by multiplying the whole state by the constant matrix, each row is the circular right shift of the previous row.

AddRoundKey The Add RoundKey transformation in the Whirlpool cipher is done byte by byte, because each round key is also a state of an 8 x 8 matrix. Figure 12.13 shows the process.

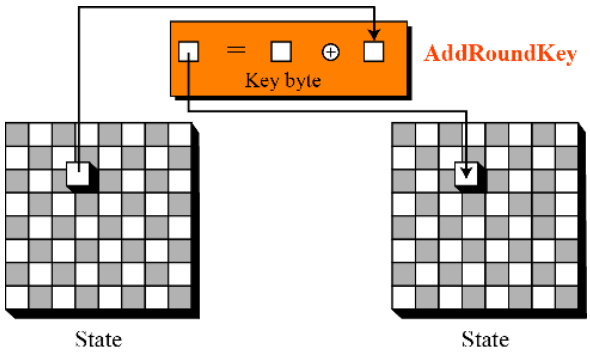


Figure 12.13 AddRoundKey transformation in the Whirlpool cipher

Key Expansion

As figure 12.14 shows, the key-expansion algorithm in Whirlpool is totally different from the algorithm in AES. Instead of using a new algorithm for creating round keys, Whirlpool uses a copy of the encryption algorithm (without the pre-round) to create the round keys. The output of each round in the encryption algorithm is the round key for that round. The key-expansion algorithm uses 10 round constants (RCs) as the round keys and the encryption algorithm uses the output of each round of the key-expansion algorithm as the round keys. The key-generation algorithm treats the cipher key as the plaintext and encrypts it. Note that the cipher key is also K_0 for the encryption algorithm.

Downloaded by TONY 562 (tony2802b@gmail.com)

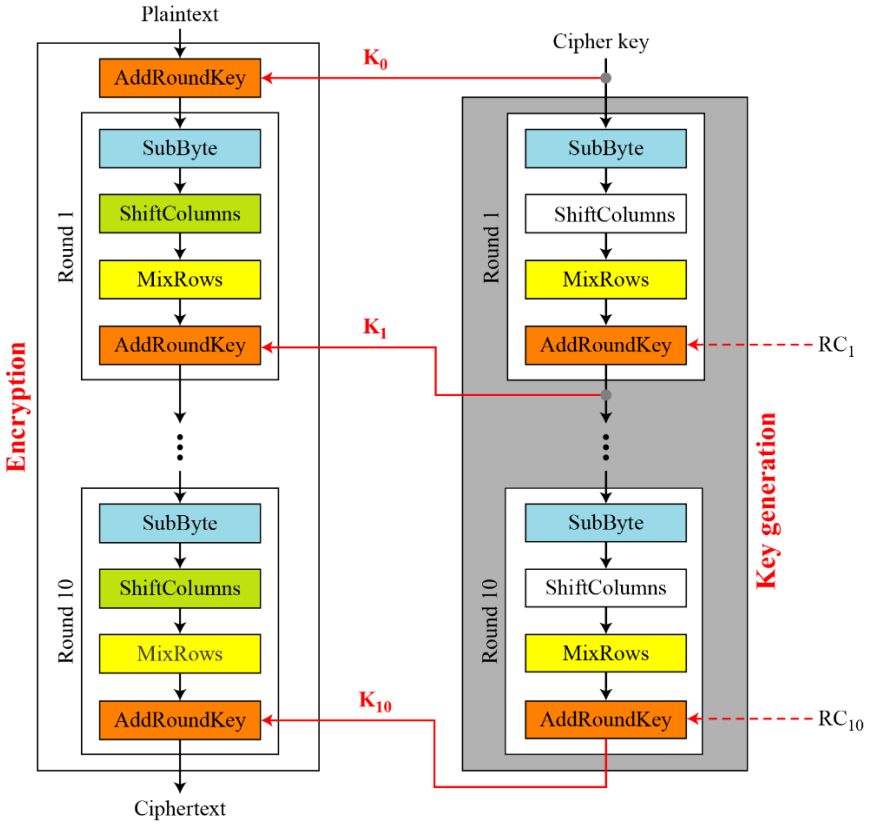


Figure 12.14 Key expansion in the Whirlpool cipher

Round Constants Each round constant, RC_r is an 8 x 8 matrix where only the first row has non-zero values. The rest of the entries are all 0's. The values for the first row of RC1 uses the first eight entries in the SubBytes transformation table 12.2. In other words, RC1 uses the first eight entries in the SubBytes transformation table [Table 12.2]; RC2 uses the second eight entries, and so on. For example, figure 12.15 shows RC3, where the first row is the third eight entries in the SubBytes table.

$$RC_3 = \begin{bmatrix} \mathbf{1D} & \mathbf{E0} & \mathbf{D7} & \mathbf{C2} & \mathbf{2E} & \mathbf{4B} & \mathbf{FE} & \mathbf{57} \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{bmatrix}$$

Figure 12.15 Round constant for the third round

Table 12.3 shows the main characteristics of the Whirlpool cipher.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Block size: 512 bits
Cipher key size: 512 bits
Number of rounds: 10
Key expansion: using the cipher itself with round constants as round keys
Substitution: SubBytes transformation
Permutation: ShiftColumns transformation
Mixing: MixRows transformation
Round Constant: cubic roots of the first eighty prime numbers

Table 12.3 The main characteristics of the Whirlpool cipher.

Analysis Although Whirlpool has not been extensively studied or tested, it is based on a robust scheme (Miyaguchi-Preneel), and for a compression function uses a cipher that is based on AES, a cryptosystem that has been proved very resistant to attacks In addition, the size of the message digest is the same as for SHA-512. Therefore it is expected to be a very strong cryptographic hash function. However, more testing and researches are needed to confirm this. The only concern is that Whirlpool, which is based on a cipher as the compression function, may not be as efficient as SHA-512, particularly when it is implemented in hardware.

4. SHA-512

SHA-512 is the version of SH with a 512-bit message digest. This is designed based on the Merkle-Damgard scheme. It is the latest version, it has a more complex structure than the others, and its message digest is the longest.

Characteristics	SHA-512
Maximum Message size	$2^{128} - 1$
Block size	1024
Message digest size	512
Number of rounds	80
Word size	64

Table 12.4 Characteristic of SHA-512

4.1 Introduction

SHA creates a digest of 512 bits from a multiple-block message. Each block is 1024 bits length, as shown in figure 12.16.

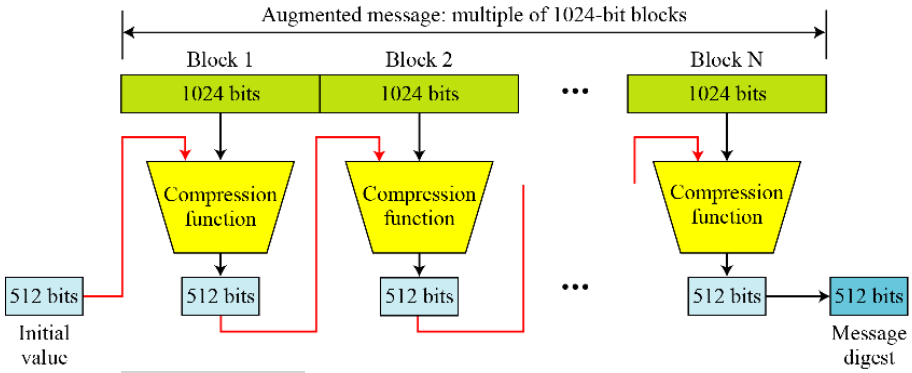


Figure 12.16 Message digest creation SHA-512

Downloaded by TONY 562 (tony2802b@gmail.com)

The digest is initialized to a predetermined value of 512-bits. The algorithm mixes this initial value with the first block of the message to create the first intermediate message digest of 512 bits. The digest is then mixed with the second block to create the second intermediate digest. Finally, the (N-1)th digest is mixed with the Nth block to create the Nth digest. When the last block is processed, the resulting digest is the message digest for the entire message.

Message Preparation
SHA-512 insists that the length of the original message be less than 2^{128} bits. This is not usually a problem because 2^{128} bits is probably larger than the total storage capacity of any system.

Length Field and Padding Before the message digest can be created, SHA-512 requires the addition of a 128-bit unsigned-integer length field to the message that defines the length of the message in bits. This is the length of the original message before padding. The length field defines the length of the original message before adding the length field or the padding (figure 12.17).

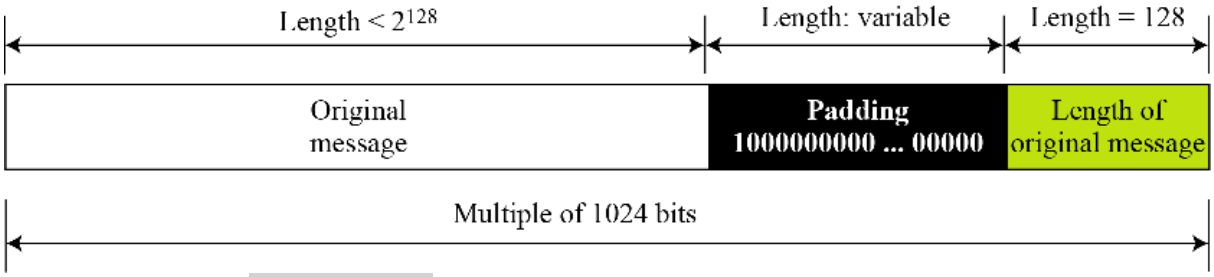


Figure 12.17 Padding and length field in SHA-512

Before the addition of the length field, we need to pad the original message to make the length a multiple of 1024. We reserve 128 bits for the length field, as shown in Figure. The length of the padding field can be calculated as follows.

Let |M| be the length of the original message and |P| be the length of the padding field

$$(|M| + |P| + 128) = 0 \bmod 1024 \quad \rightarrow \quad |P| = (-|M| - 128) \bmod 1024$$

The format of the padding is one 1 followed by number of 0s. (Ex: 1000000000.....)

Example:
What is the number of padding bits if the length of the original message is 2590 bits in SHA-512?

Solution:
We can calculate the number of padding bits as follows:

$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

The padding consists of one 1 followed by 353 0's.

Words SHA-512 operates on words; it is word oriented. A word is defined as 64 bits. This means that, after the padding and the length field are added to the message, each block of the message consists of sixteen 64-bit words. The message digest is also made of 64-bit words, but the message digest is only eight words and the words are named A, B, C, D, E, F, G, and H, as shown in figure 12.18.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

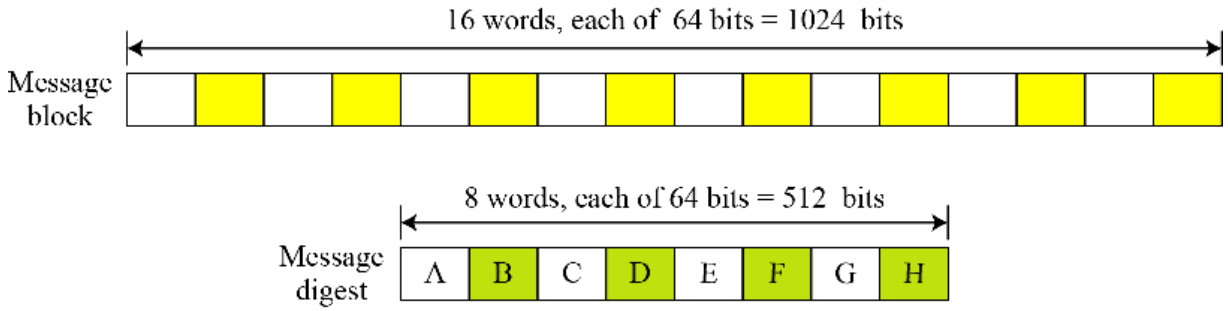


Figure 12.18 A message block and the digest as words

SHA-512 is word-oriented. Each block is 16 words; the digest is only 8 words.

Word Expansion Before processing, each message block must be expanded. A block is made of 1024 bits, or sixteen 64-bit words. As we will see later, we need 80 words in the processing phase. So the 16-word block needs to be expanded to 80 words, from W_0 to W_{79}. Figure 12.19 shows the word-expansion process. The 1024-bit block becomes the first 16 words; the rest of the words come from already-made words according to the operation shown in the figure.

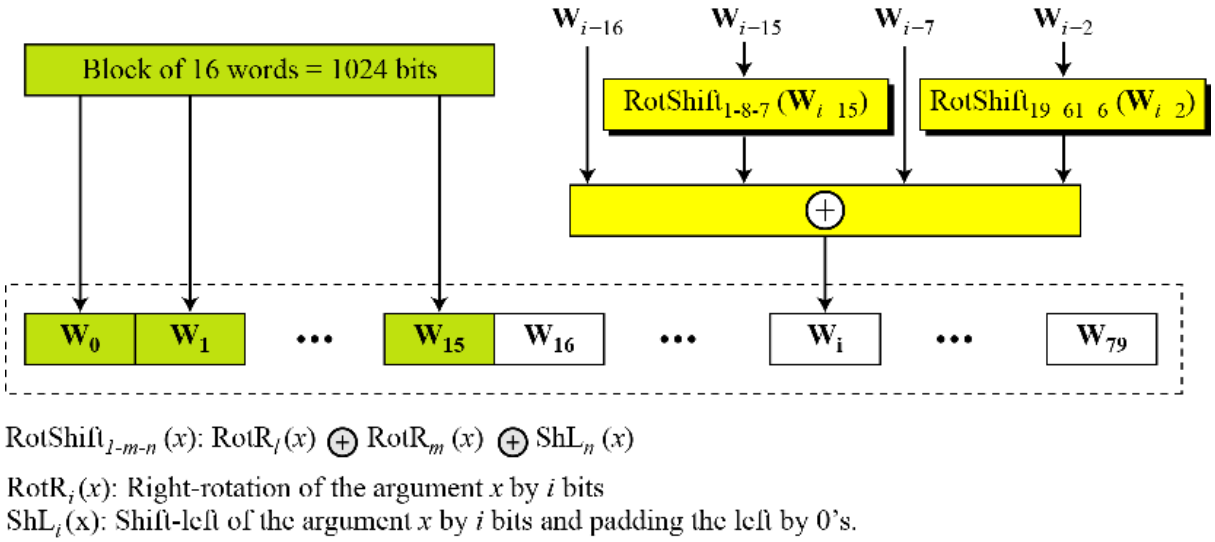


Figure 12.19 Word expansion in SHA-512

Example:
Show how W60 is made.

Solution:
Each word in the range W16 to W79 is made from four previously-made words. W60 is made as

$$W_{60} = W_{44} \oplus \text{RotShift}_{1\text{-}8\text{-}7}(W_{45}) \oplus W_{53} \oplus \text{RotShift}_{19\text{-}61\text{-}6}(W_{58})$$

Message Digest Initialization The algorithm uses eight constants for message digest initialization. We call these constants A0 to H0 to match with the word naming used for the digest. Table 12.5 shows the value of these constants.

Downloaded by TONY 562 (tony2802b@gmail.com)

Buffer	*Value (in hexadecimal)*	*Buffer*	*Value (in hexadecimal)*
A_0	**6A09E667F3BCC908**	E_0	**510E527FADE682D1**
B_0	**BB67AE8584CAA73B**	F_0	**9B05688C2B3E6C1F**
C_0	**3C6EF372EF94F828**	G_0	**1F83D9ABFB41BD6B**
D_0	**A54FE53A5F1D36F1**	H_0	**5BE0CD19137E2179**

Table 12.5 Values of constants in message digest initialization of SHA-512

The values are calculated from the first eight prime numbers (2, 3, 5, 7, 11, 13, 17, and 19). Each value is the fraction part of the square root of the corresponding prime number after converting to binary and keeping only the first 64-bits. For example, the eighth prime is 19, with the square root $(19)^{1/2} = 4.35889894354$. Converting the number to binary with only 64 bits in the fraction part, we get

$(100.0101\ 1011\ 1110 \ldots 1001)_2 = (4.5BE0CD19137E2179)_{16}$

SHA-512 keeps the fraction part, $(5BE0CD19137E2179)_{16}$, an unsigned integer.

4.2 Compression Function

SHA-512 creates a 512-bit (eight 64-bit words) message digest from a multiple-block message where each block is 1024 bits. The processing of each block of data in SHA-512 involves 80 rounds. Figure 12.20 shows the general outline for the compression function. In each round, the contents of eight previous buffers, one word from the expanded block (W_i), and one 64-bit constant (K_i) are mixed together and then operated on to create a new set of eight buffers. At the beginning of processing, the values of the eight buffers are saved into eight temporary variables. At the end of the processing (after step 79), these values are added to the values created from step 79. We call this last operation the final adding, as shown in figure 12.20.

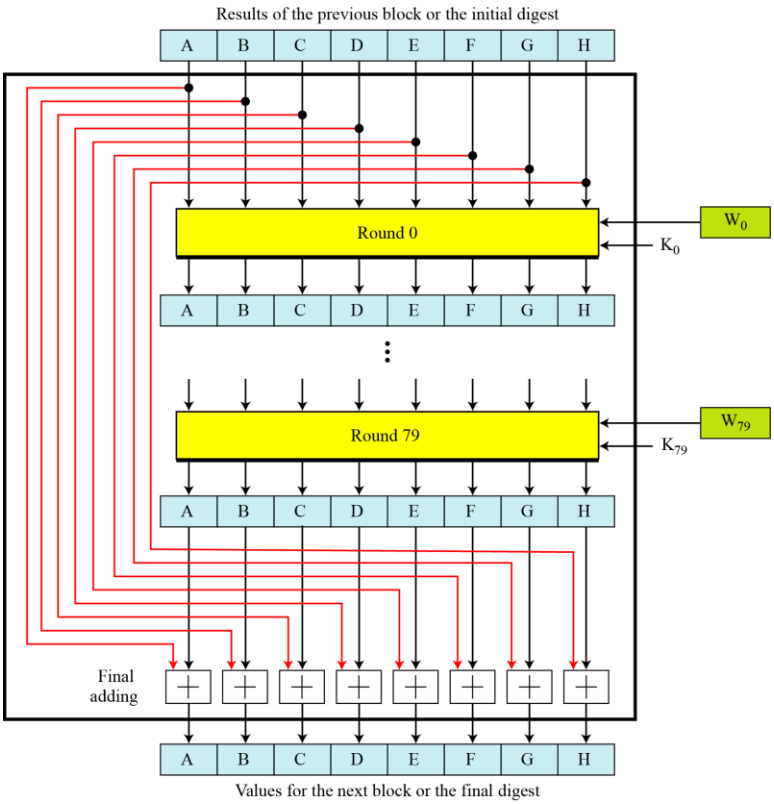


Figure 12.20 Compression function in SHA-512

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Structure of each Round

In each round, eight new values for the 64-bit buffers are created from the values of the buffers in the previous round. As figure 12.21 shows, six buffers are the exact copies of one of the buffers in the previous round as shown below:

$$A \rightarrow B \quad B \rightarrow C \quad C \rightarrow D \quad E \rightarrow F \quad F \rightarrow G \quad G \rightarrow H$$

Two of the new buffers, A and E, receive their inputs from some complex functions that involve some of the previous buffers, the corresponding word for this round (W_i), and the corresponding constant for this round (K_i). Figure shows the structure of each round.

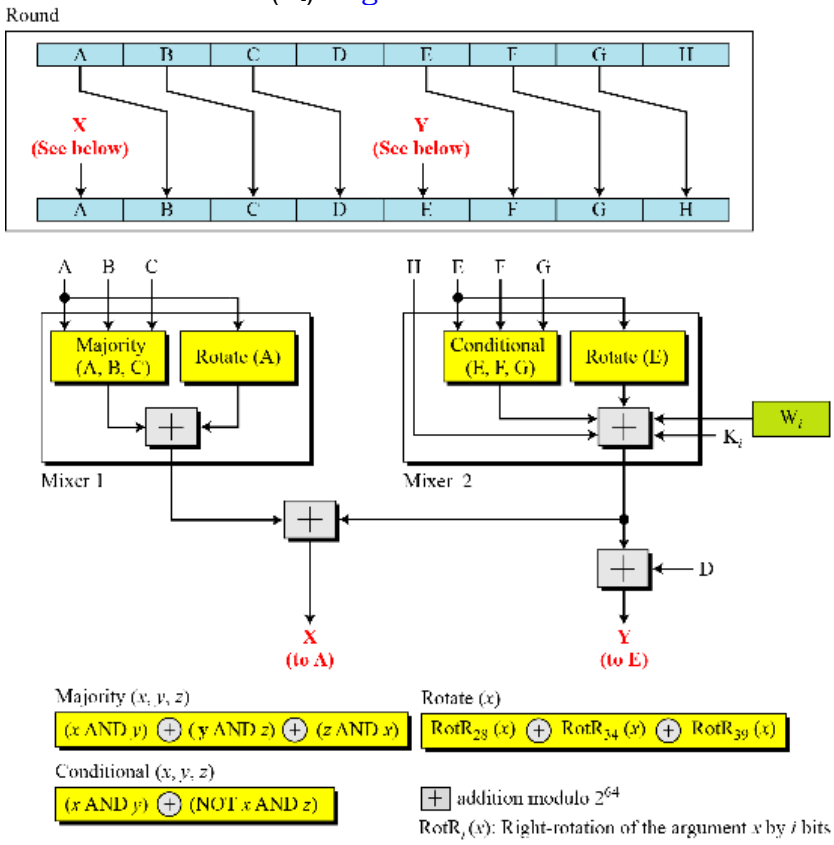


Figure 12.21 Structure of each round in SHA-512

There are two mixers, three functions, and several operators. Each mixer combines two functions. The description of the functions and operators follows:

1. The Majority function, as we call it, ***if two or three bits are 1's (in A_i, B_i, and C_i), then the resulting bit is 1; otherwise it is 0.***

   We can write it in bitwise function as

   $$(A_j \text{ and } B_j) \oplus (B_j \text{ and } C_j) \oplus (C_j \text{ and } A_j)$$

2. The Conditional function, as we call it, is also a bitwise function. It takes three corresponding bits in three buffers (E, F, and G) and calculates

   $$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ and } G_i)$$

   The resulting bit is the logic "**if E_j then F_j; else G_j**".

3. The rotate function, as we all it, right-rotates the three instances of the same buffer (A or E) and applies the exclusive-or operation on the results.

   $$\text{Rotate (A): } \text{RotR}_{28}(A) \oplus \text{RotR}_{34}(A) \oplus \text{RotR}_{29}(A)$$

Downloaded by TONY 562 (tony2802b@gmail.com)

Rotate (E): $\text{RotT}_{28}(E) \oplus \text{RotR}_{34}(E) \oplus \text{RotR}_{29}(E)$

4. The right-rotation function, $\text{RotR}_i(x)$, is the same as the one we used in the word-expansion process. It right-rotates its argument i bits; it is actually a circular shift-right operation.
5. The addition operator used in the process is addition modulo 2^{64}. This means that the result of adding two or more buffers is always a 64-bit word.
6. There are 80 constants, K_0 to K_{79}, each of 64 bits as shown in table 12.6 in hexadecimal format. Similar to the initial values for the eight digest buffers, these values are calculated from the first 80 prime numbers (2, 3, … , 409).

Each value is the fraction part of the cubic root of the corresponding prime number after converting it to binary and keeping only the first 64 bits. For example, the 80th prime is 409, with the cubic root $(409)^{1/3} = 7.42291412044$. Converting this number to binary with only 64 bits in the fraction part, we get

$$(111.0110\ 1100\ 0100\ 0100 \ldots 0111)_2 \rightarrow (7.6C44198C4A475817)_{16}$$

SHA-512 keeps the fraction part, $(6C44198C4A475817)_{16}$, as an unsigned integer.

Analysis With a message digest of 512 bits, SHA-512 expected to be resistant to all attacks, including collision attacks. It has been claimed that this version's improved design makes it more efficient and more secure that the previous versions.

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

Table 12.6 Eighty constants used for eighty rounds in SHA-512

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Ch-13 – Digital Signature

1. Comparison

Let us begin by looking at the differences between conventional signatures and digital signatures.

1.1 Inclusion

- A conventional signature is included in the document; it is part of a document.
- When we write a check, the signature is on the check; it is not a separate document. But when we sign a document digitally, we send the signatures as a separate document.
- The sender sends two documents; the message and the signature.
- The recipient receives both document and verifies that the signature belongs to the supposed sender.
- If this is proven, the message is kept; otherwise, it is rejected.

1.2 Verification method

- The second difference between the two types of signatures is the method of verifying the signature.
- For a conventional signature, when the recipient receives a document, she compares the signature on the document with the signature on file.
- If they are the same, the document is authentic.
- The recipient needs to have a copy of this signature on file for comparison.
- For a digital signature, the recipient receives the message and the signature.
- A copy of the signature is not stored anywhere.
- The recipient needs to apply a verification technique to combination of the message and the signature to verify the authenticity.

1.3 Relationship

- For a conventional signature, there is normally a one- to-many relationship between a signature and documents.
- A person uses the same signature to sign many documents.
- For a digital signature, there is a one-to-one relationship between a signature and a message.
- Each message has its own signature.
- The signature of one message cannot be used in another message.
- If Bob receives two messages, one after another, from Alice, he cannot use the signature of the first message to verify the second.
- Each message needs a new signature.

1.4 Duplicity

- Another differences between the two types of signatures is a quality called duplicity.
- In conventional signature, a copy of the signed document can be distinguished from the original one on file.

Downloaded by TONY 562 (tony2802b@gmail.com)

- In digital signature, there is no such distinction unless there is a factor of time (such as a timestamp) on the document.
- For example, suppose Alice sends a document instructing Bob to pay Eve.
- If Eve intercepts the document and the signature, she can replay it later to get money again from Bob.

2. Process

- Figure 13.1 shows the digital signature process. The sender uses a signing algorithm to sign the message.
- The message and the signature are sent to the receiver. The receiver receives the message and the signature and applies the verifying algorithm to the combination.
- If the result is true, the message is accepted; otherwise, it is rejected.

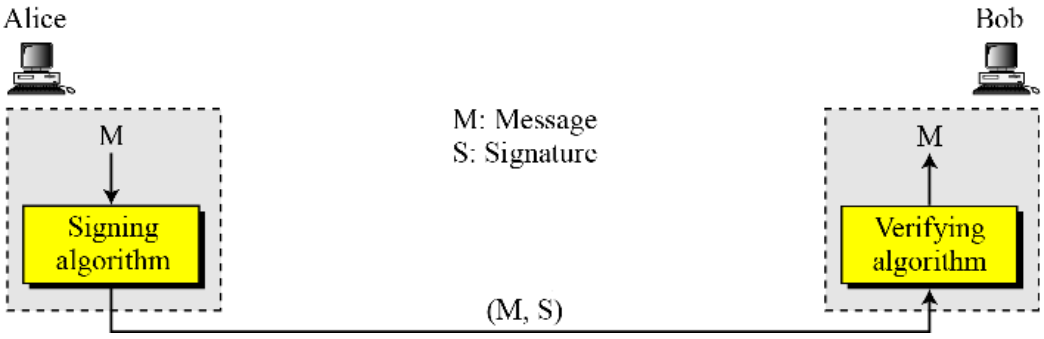


Figure 13.1 Digital signature process

2.1 Need for Keys

- In a digital signature, the signer uses her private key, applied to a signing algorithm to sign the document.
- The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document. See Figure 13.2.

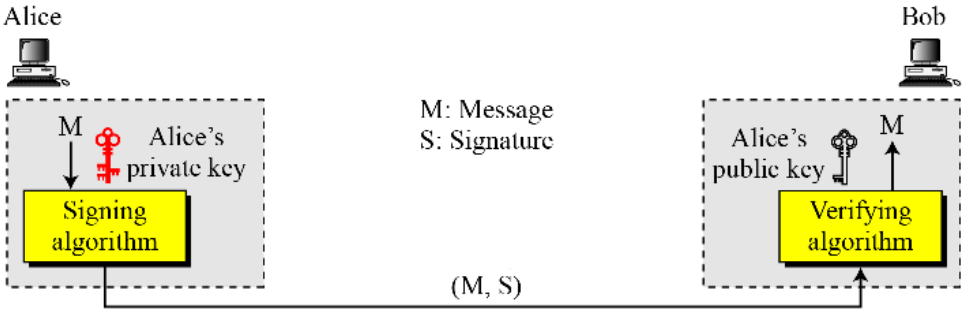


Figure 13.2 Adding key to the digital signature process

A digital signature needs a public-key system. The signer signs with her private key; the verifier verifies with the signer's public key.

A cryptosystem uses the public key and private of receiver; a digital signature uses the private and public keys of the sender.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

2.2 Signing the digest

Figure 13.3 shows signing a digest in a digital signature system.

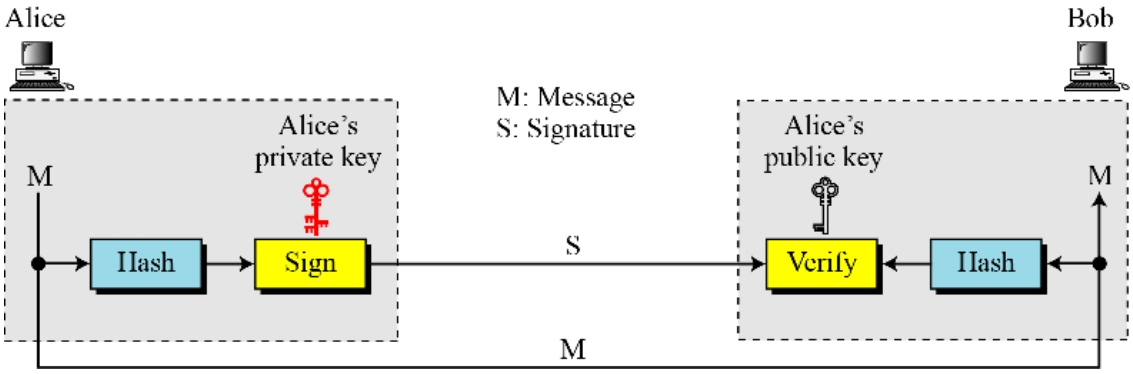


Figure 13.3 Signing the digest

- A digest is made out of the message at Alice's site. The digest then goes through the signing process using Alice's private key. Alice then sends the message and the signature to Bob.
- At Bob's site, using the same public hash function, a digest is first created out of the received message. Calculations are done on the signature and the digest. The verifying process also applies criteria on the result of the calculation to determine the authenticity of the signature. If authentic, the message is accepted; otherwise it is rejected.

3. Services provided by digital signature

A digital signature can directly provide the message authentication, message integrity and non-repudiation; for message confidentiality we still need encryption/decryption.

3.1 Message Authentication

A secure digital signature scheme, like a secure conventional signature (one that cannot be copied) can provide message authentication (also referred to as data-origin authentication). Bob can verify that the message is sent by Alice because Alice's public key is used in verification. Alice's public key cannot verify the signature signed by the Eve's private key.

3.2 Message Integrity

The integrity of the message is preserved even if we sign the whole message because we cannot get the same signature if the message is changed. The digital signature schemes today use a hash function in the signing and verifying algorithms that preserve the integrity of the message.

3.3 Nonrepudiation

If Alice signs a message and then denies it, can Bob later prove that Alice actually signed it? For example, If Alice sends a message to a bank (Bob) and asks to transfer $10,000 from her account to Ted's account, can Alice later deny that she sent this message? Bob might have a problem. Bob must keep the signature on file and later use Alice's public key to create the original message to prove the message in the file and later use Alice's public key to create the original message to prove the message in the file and the newly created message are the same. This is not feasible because

Downloaded by TONY 562 (tony2802b@gmail.com)

Alice may have changed her private or public key during this time; she may also claim that the file containing the signature is not authentic.

One solution is a trusted third party. People can create an established trusted party among themselves. Figure 13.4 shows how a trusted party can present Alice from denying that she sent the message.

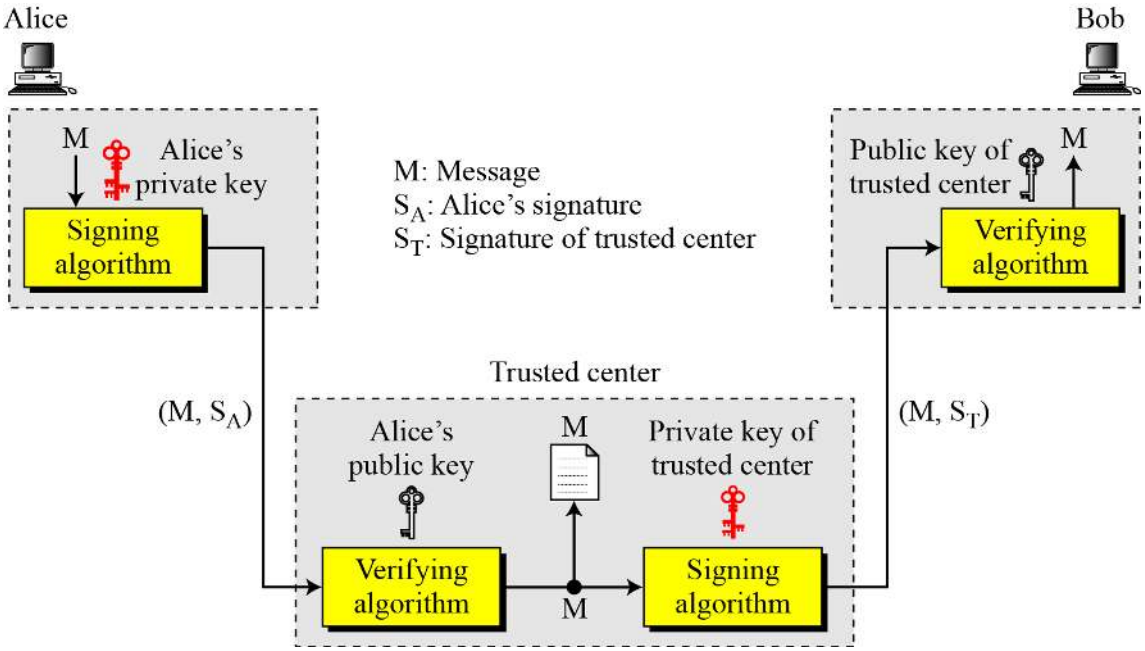


Figure 13.4 Using a trusted center for nonrepudiation

3.4 Confidentiality

- A digital signature does not provide confidential communication
- If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key cryptosystem. Figure 13.5 shows how this extra level can be added to a simple digital signature scheme.

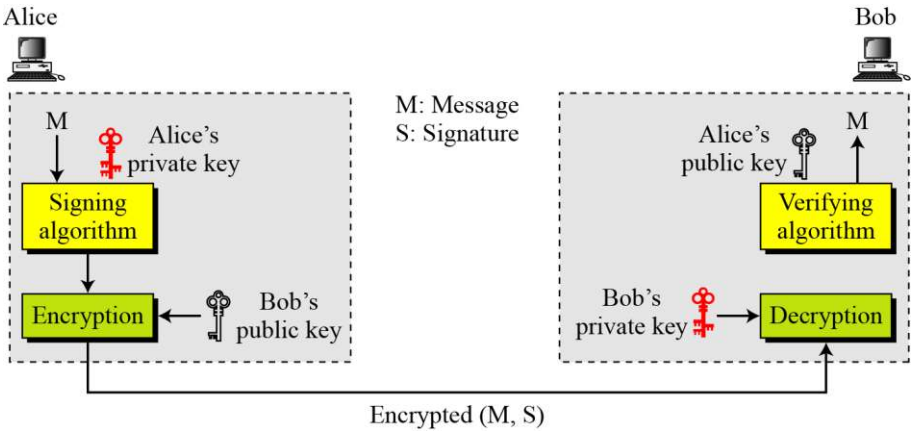


Figure 13.5 Adding confidentiality to a digital signature scheme

- We have shown asymmetric key encryption/decryption just to emphasize the type of keys used at each end. Encryption/decryption can also be done with as symmetric key.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

A digital signature does not provide privacy. If there is a need for privacy, another layer of encryption/decryption must be applied.

4. ATTACKS ON DIGITAL SIGNATURE

4.1 Attack types

We will look on three kinds of attacks on digital signatures; key-only, known message and chosen-message.

Key-Only Attack

- In this, Eve has access only to the public information released by Alice. To forge a message, she has access to some documents previously signed by Alice.
- Eve tries to create another message and forge Alice's signature on it.
- This is similar to the known-plaintext attack we discussed for encipherment.

Known-Message Attack

- In this, Eve has access to one or more message signature pairs.
- In-other words, she access to some documents previously signed by Alice.
- Eve tries to create another message and forge Alice's signature on it.
- This is similar to the known-plaintext attack we discussed for encipherment.

Chosen-Message Attack

- In the chosen-message attack, Eve somehow makes Alice sign one or more messages of her.
- Eve now has a chosen-message/signature pair.
- Eve later creates another message, with the content she wants, and forge Alice's signature on it.
- This is similar to the chosen-plaintext attack we discussed for encipherment.

4.2 Forgery Types

If the attack is successful, the result is a forgery. We can have two types of forgery: existential and selective.

Existential Forgery

- In an existential forgery, Eve may be able to create a valid message-signature pair, but not one that she can really use.
- In other words, a document has been forged, but the content is randomly calculated.
- This type of forgery is probable, but fortunately Eve cannot benefit from it very much.
- Her message could be syntactically or semantically unintelligible.

Selective Forgery

- In selective forgery, Eve may be able to forge Alice's signature on a message with the content selectively chosen by Eve.
- Although this is beneficial to Eve, and may be very detrimental to Alice, the probability of such forgery is low, but not negligible.

Downloaded by TONY 562 (tony2802b@gmail.com)

5. Digital Signature Schemes

5.1 RSA Digital Signature Scheme

In chapter 10 we discussed how to use RSA cryptosystem to provide privacy. The RSA idea can also be used for signing and verifying a message. In this case, it is called the RSA digital signature scheme. The sender uses her own private key to sign the document; the receiver uses the sender's public key to verify it. Figure 13.6 gives the general idea behind the RSA digital signature scheme.

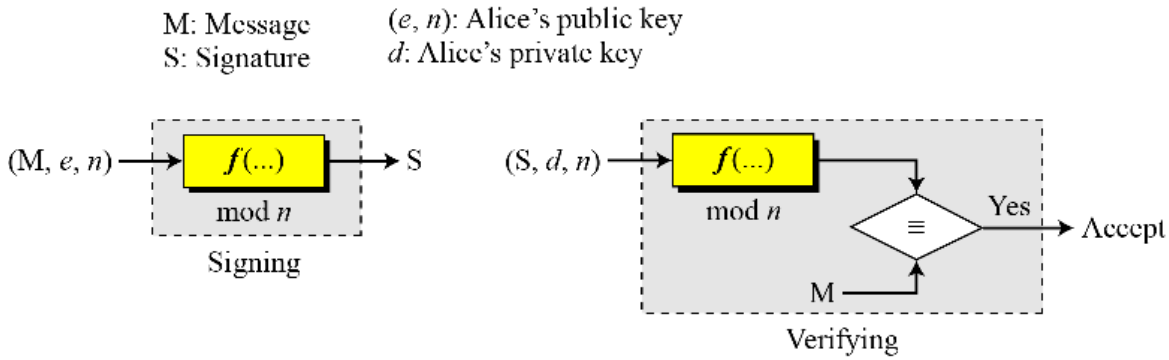


Figure 13.6 General idea behind the RSA digital signature scheme

- The signing and verifying sites uses the same function, but with different parameters.
- The verifier compares the message and the output of the function for congruence.
- If the result is true, the message is accepted.

Key Generation

RSA_Key_Generation

{

Select two large primes p and q such that $p \neq q$.

$n \leftarrow p \times q$

$\phi(n) \leftarrow (p-1) \times (q-1)$

Select e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$

$d \leftarrow e^{-1} \bmod \phi(n)$ // d is inverse of e modulo $\phi(n)$

Public_key $\leftarrow$ (e, n) // To be announced publicly

Private_key $\leftarrow d$ // To be kept secret

return Public_key and Private_key

}

Alice keeps d; she publicly announces n and e.

Signing and Verifying

Figure 13.7 shows the RSA digital signature scheme.

Signing Alice creates a signature out of the message using her private exponent, $S = M^d \bmod n$ and sends the message and the signature to Bob.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

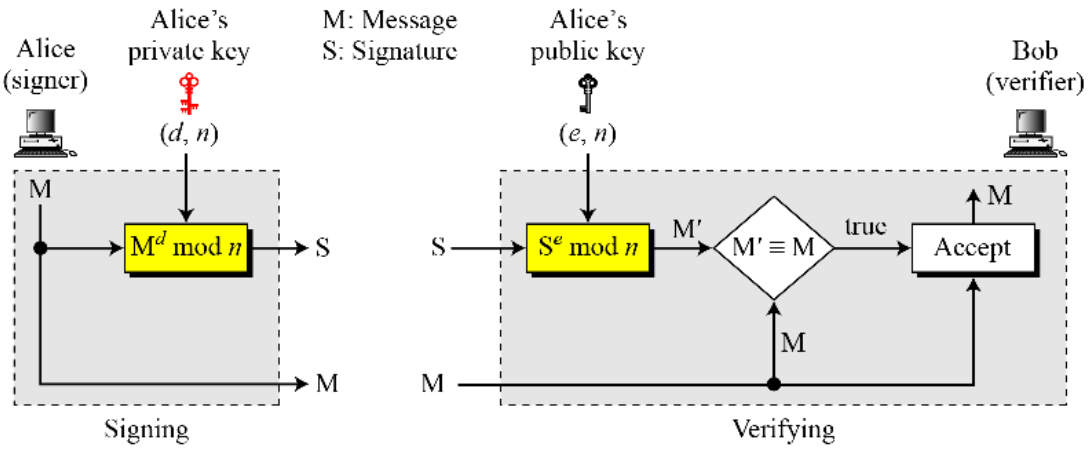


Figure 13.7 RSA digital signature scheme

Verifying

- Bob receives M and S, Bob applies Alice's public exponent to the signature to create a copy of the message M' = S^e mod n.
- Bob compares the value of M' with the value of M.
- If the two values are congruent, Bob accepts the message.
- To prove this, we start with the verification criteria:

M' = M mod n. → $S^e \equiv M \bmod n$ → $M^{d \times e} \equiv M \pmod n$

The last congruent holds because d x e = 1 mod Φ(n) (See Euler's theorem in chapter 9)

5.2 Attacks on RSA Signature

There are some attacks that Eve can apply to the RSA digital signature scheme to forge Alice's signature.

Key-only attack

- Eve has access only to Alice's public key.
- Eve intercepts the pair (M, S) and tries to create another message M' = S^e (mod n).

Known-Message attack

- Here Eve uses the multiplicative property of RSA.
- Assume that Eve has intercepted two message-signature pair (M_1, S1) and (M_2, S2) that have been created using the same private key.
- If M = ($M_1 \times M_2$) mod n, then S = ($S_1 \times S_2$) mod n

$$S = (S_1 \times S_2) \bmod n = (M_1^d \times M_2^d) \bmod n = (M_1 \times M_2)^d \bmod n = M^d \bmod n$$

Chosen Message Attack

This attack also uses the multiplicative property of RSA. Eve can somehow ask Alice to sign two legitimate messages, M_1 and M_2, for her and later creates a new message M = M_1 x M_2. Eve can later claim that Alice has signed M. The attack is also referred to as multiplicative attack. This is a very

Downloaded by TONY 562 (tony2802b@gmail.com)

serious attack on the RSA digital signature scheme because it is a selective forgery (Eve can manipulate M_1 and M_2 to get a useful M).

RSA Signature on the Message Digest

As we discussed before, signing a message digest using a strong hash algorithm has several advantages. In the case of RSA, it can make the signing and verifying processes much faster because the RSA digital signature scheme is nothing other than encryption with the private key and decryption with the public key. The use of a strong cryptographic hashing function also makes the attack on the signature much more difficult as we will explain shortly. Figure 13.8 shows the scheme.

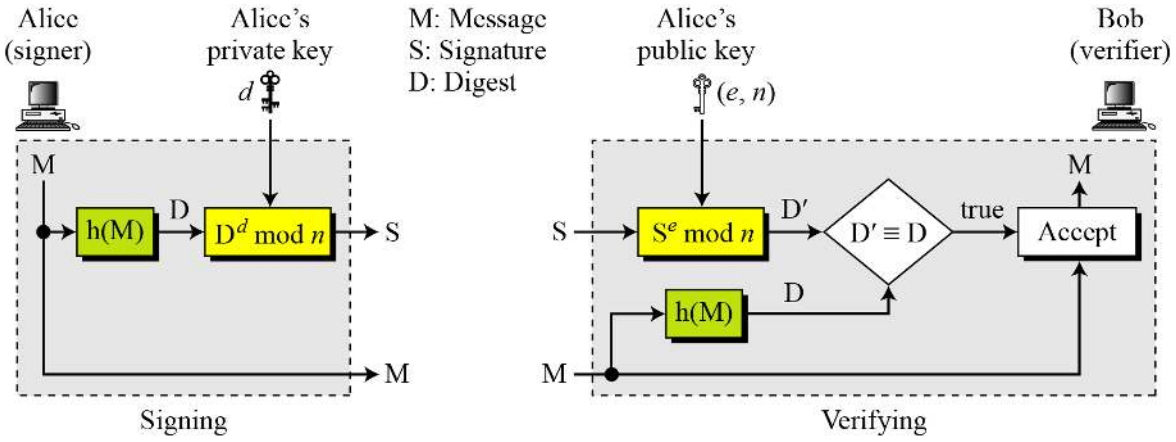


Figure 13.8 The RSA signature on the message digest

Alice, the signer, first uses an agreed-upon hash function to create a digest from the message, D = h(M). She then signs the digest, $S = D^d \bmod n$. The message and the signature are sent to Bob. Bob, the verifier, receives the message and the signature. He first uses Alice's public exponent to retrieve the digest, $D' = S^e \bmod n$. He then applies the hash algorithm to the message received to obtain D = h(M). Bob now compares the two digests, D and D'. If they are congruent to modulo n, he accepts the message.

Attacks on RSA Signed Digests

Key Only Attack We can have three cases of this attack:

a. Eve intercepts the pair (S, M) and tries to find another message M' that creates the same digest, h(M) = h(M'). As we learned in chapter 11, if the hash algorithm is second preimage resistant, this attack is very difficult.

b. Eve finds two messages M and M' such that h(M) = h(M'). She lures Alice to sign h(M) to find S. Now Eve has a pair (M', S) which passes the verifying test, but it is the forgery. We learned in chapter 11 that if the hash algorithm is collision resistant, this attack is very difficult.

c. Eve may randomly find message digest D. which may match with a random signature S. She then finds a message M such that D = h(M). As we learned in Chapter 11, if the hash function is preimage resistant, this attack is very difficult to launch.

Known Message Attack Let us assume Eve has two message-signature pairs (M_1, S_1) and (M_2, S_2) which have been created using the same private key. Eve calculates $S \equiv S_1 \times S_2$. If she can find a message M such that $h(M) \equiv h(M_1) \times h(M_2)$, she has forged a new message. However, finding M given h(M) is very difficult if the hash algorithm is preimage resistant.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Chosen-Message Attack Eve can ask Alice to sign two legitimate messages M_1 and M_2 for her. Eve then creates a new signature $S \equiv S_1 \times S_2$. Since Eve can calculate $h(M) \equiv h(M_1) \times h(M_2)$, if she can find a message M given h(M), the new message is a forgery, However, finding M given h(M), is very difficult if the hash algorithm is preimage resistant.

5.3 ElGamal Digital Signature Scheme

The ElGamal cryptosystem was discussed in chapter 10. The ElGamal digital signature scheme uses the same keys, but the algorithm, as expected, is different. Figure 13.9 gives the general idea behind the ElGamal digital signature scheme.

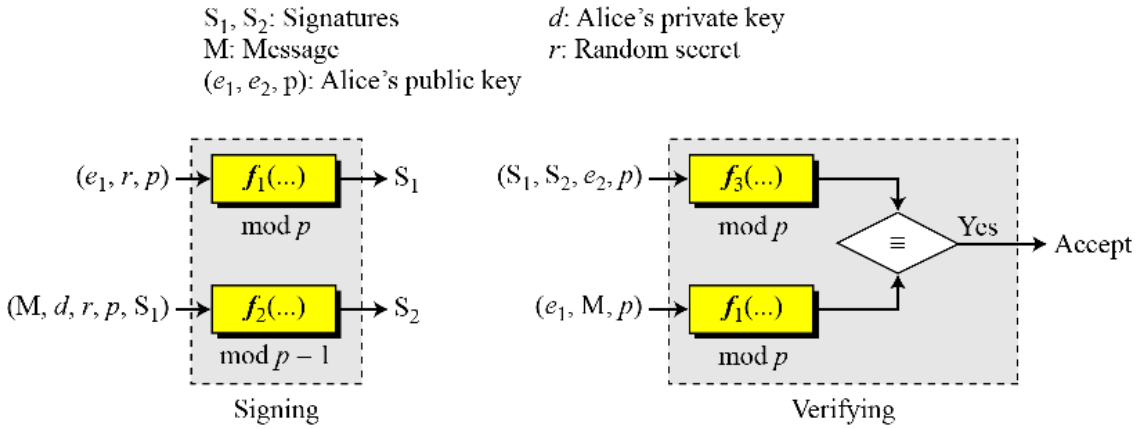


Figure 13.9 General idea behind the ElGamal digital signature scheme

- In the signing process, two functions create two signatures: in the verifying process the outputs of two functions are compared for verification.
- Note that one function is used both for signing and verifying but the function uses different inputs.

Key Generation

- The key generation procedure here is exactly the same as the one used in the cryptosystem.
- Let p be a prime number large enough that the discrete log problem is intractable in Zp*.
- Let e1 be a primitive element in Zp*, Alice selects her private key d to be less than p-1.
- She calculates $e_2=e_1^d$.
- Alice's public key is the tuple (e1, e2, p); Alice's private key is d.

Verifying and Signing

Figure 13.10 shows the ElGamal digital signature scheme.

Signing:

- Alice can sign the digest of a message to any entity, including Bob:

1. Alice chooses a secret random number r, public and private keys can be used repeatedly, Alice needs a new r each time the signs a new message.

Downloaded by TONY 562 (tony2802b@gmail.com)

M: Message — r: Random secret
S_1, S_2: Signatures — d: Alice's private key
V_1, V_2: Verifications — (e_1, e_2, p): Alice's public key

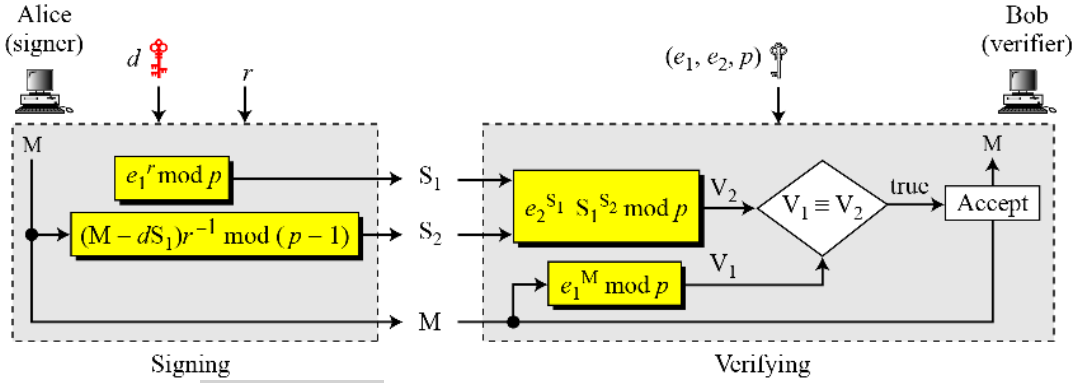


Figure 13.10 ElGamal digital signature scheme

- **Signing** Alice can sign the digest of a message to any entity, including Bob:
  1. Alice chooses a secret random number r. Note that although public and private keys can be used repeatedly, Alice needs a new r each time she signs a new message.
  2. Alice calculates the first signature $S_1 = e_1^{\,r} \bmod p$.
  3. Alice calculates the second signature $S_2 = (M - d \times S_1) \times r^{-1} \bmod (p-1)$, where r^{-1} is the multiplicative inverse of r modulo p.
  4. Alice sends M, S_1, and S_2 to Bob.

- **Verifying** An entity, such as Bob, receives M, S_1, and S_2, which can be verified as follows:
  1. Bob checks to see if $0 < S_1 < p$
  2. Bob checks to see if $0 < S_2 < p - 1$
  3. Bob calculates $V_1 = e_1^{\,M} \bmod p$
  4. Bob calculates $V_2 = e_2^{\,S_1} \times S_1^{\,S_2} \bmod p$
  5. If V_1 is congruent to V_2, the message is accepted; otherwise, it is rejected. We can prove the verification criterion using $e_2 = e_1^{\,d}$ and $S_1 = e_1^{\,r}$

$$V_1 \equiv V_2 \pmod p \rightarrow e_1^{\,M} \equiv e_2^{\,S_1} \times S_1^{\,S_2} \pmod p \equiv (e_1^{\,d})^{S_1} (e_1^{\,r})^{S_2} \pmod p \equiv e_1^{\,d S_1 + r S_2} \pmod p$$

We get: $e_1^{\,M} \equiv e_1^{\,d S_1 + r S_2} \pmod p$

8.39 x 11.00 in

Forgery in the ElGamal Digital Signature Scheme

The ElGamal scheme is vulnerable to existential forgery, but it is very hard to do a selective forgery on this scheme.

Key-Only Forgery In this type of forgery, Eve has access only to the public key. Two kinds of forgery are possible:

1. Eve has a predefined message M. She needs to forge Alice's signature on it. Eve must find two valid signatures S_1 and S_2 for this message. This is a selective forgery.
   a. Eve can choose S_1 and calculate S_2. She needs to have $d^{S_1} S_1^{S_2} \equiv e_1^{\,M} \pmod p$. In other words, $S_1^{S_2} \equiv e_1^{\,M} d^{-S_1} \pmod p$ or $S_2 \equiv \log_{S_1} (e_1^{\,M} d^{-S_1}) \pmod p$. This means computing the discrete logarithm, which is very difficult.
   b. Eve can choose S_2 and calculate S_1. This is much harder than part *a*.
2. Eve may be able to find three random values, M, S_1, and S_2 such that the last two are the signature of the first one. If Eve can find two new parameters x and y such that $M = xS_2 \bmod (p-1)$ and $S_1 = -yS_2 \bmod (p-1)$, she can forge the message, but it might not be very useful for her. This is an existential forgery.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Known-Message Forgery If Eve has intercepted a message M and its two signatures S_1 and S_2, she can find another message M′, with the same pair of signatures S_1 and S_2. However, note that this is also an existential forgery that does not help Eve very much.

5.4 Schnorr Digital Signature Scheme

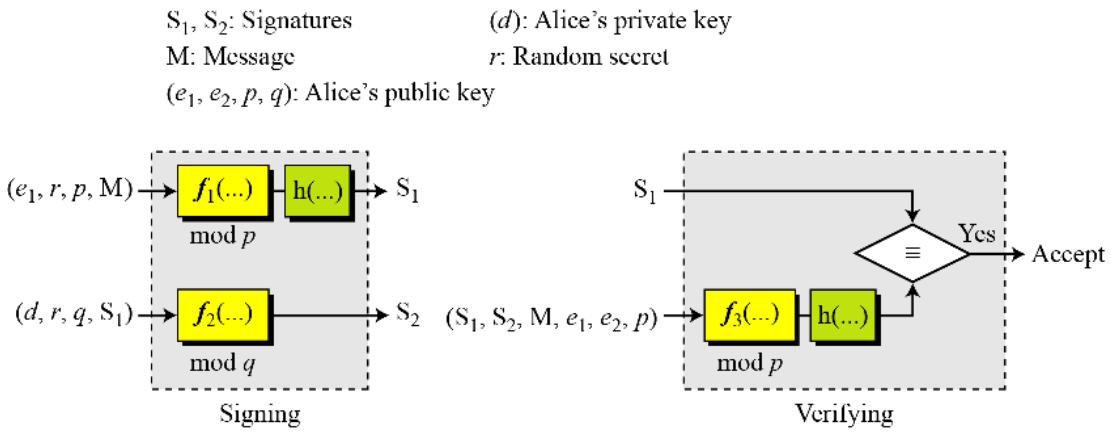


Figure 13.11 General idea behind the Schnorr digital signature scheme

Key Generation

1) Alice selects a prime p, which is usually 1024 bits in length.
2) Alice selects another prime q.
3) Alice chooses e_1 to be the qth root of 1 modulo p.
4) Alice chooses an integer, d, as her private key.
5) Alice calculates $e_2 = e_1^d \bmod p$.
6) Alice's public key is (e_1, e_2, p, q); her private key is (d).

Signing and Verifying

Figure 13.12 shows the Schnorr digital signature scheme.

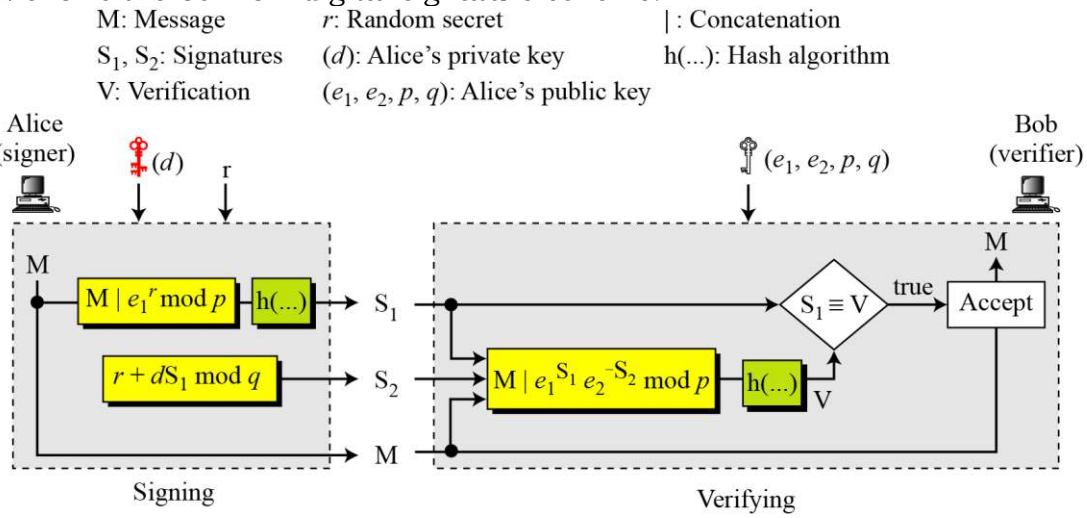


Figure 13.12 Schnorr digital signature scheme

Downloaded by TONY 562 (tony2802b@gmail.com)

Signing

1. Alice chooses a random number r.
2. Alice calculates $S_1 = h(M | e_1^r \bmod p)$.
3. Alice calculates $S_2 = r + d \times S_1 \bmod q$.
4. Alice sends M, S_1, and S_2.

Verifying Message

1. Bob calculates $V = h(M \mid e_1^{S2} e_2^{-S1} \bmod p)$.
2. If S_1 is congruent to V modulo p, the message is accepted;

5.5 Digital Signature Standard

- The Digital Signature Standard (DSS) was adopted by the NIST in 1994.
- DSS uses a DSA based on the ElGamal scheme with some ideas from the Schnorr scheme.
- Figure 13.13 gives the general idea behind the DSS scheme.

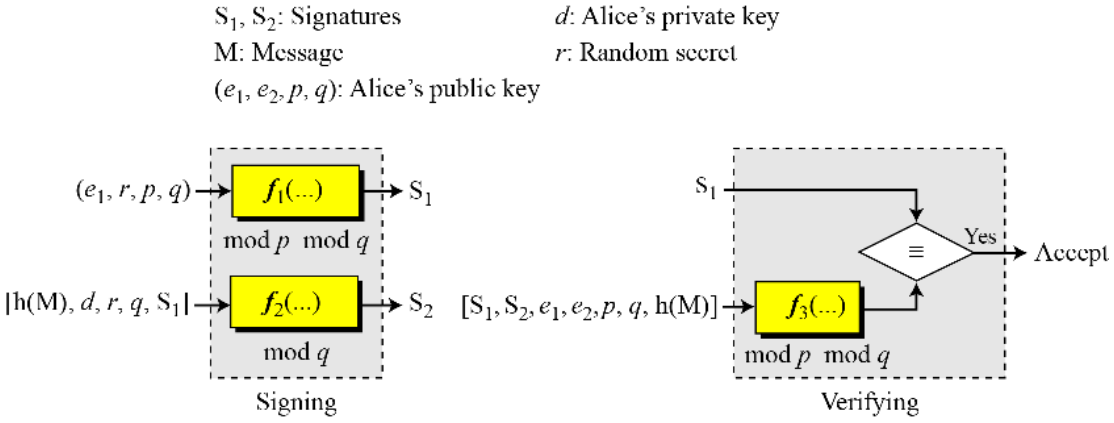


Figure 13.13 General idea behind DSS scheme

Key Generation

1) Alice chooses primes p and q.
2) Alice uses $<Z_p^*, \times>$ and $<Z_q^*, \times>$.
3) Alice creates e_1 to be the q^{th} root of 1 modulo p.
4) Alice chooses d and calculates $e_2 = e_1^d$.
5) Alice's public key is (e_1, e_2, p, q); her private key is (d).

Verifying and Signing

Figure 13.14 shows the DSS Scheme.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

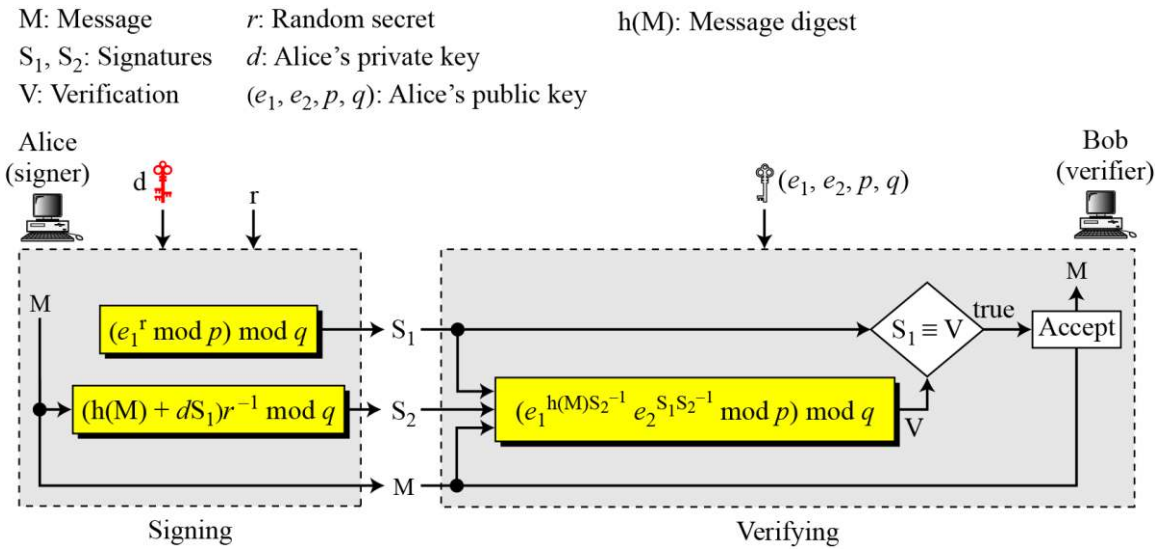


Figure 13.14 DSS scheme

- **Signing** The following shows the steps to sign the message:
  1. Alice chooses a random number r ($1 \le r \le q$). Note that although public and private keys can be chosen once and used to sign many messages, Alice needs to select a new r each time she needs to sign a new message.
  2. Alice calculates the first signature $S_1 = (e_1^{\,r} \bmod p) \bmod q$. Note that the value of the first signature does not depend on M, the message.
  3. Alice creates a digest of message h(M).
  4. Alice calculates the second signature $S_2 = (h(M) + d\,S_1)r^{-1} \bmod q$. Note that the calculation of S_2 is done in modulo q arithmetic.
  5. Alice sends M, S_1, and S_2 to Bob.

- **Verifying** Following are the steps used to verify the message when M, S_1, and S_2 are received:
  1. Bob checks to see if $0 < S_1 < q$.
  2. Bob checks to see if $0 < S_2 < q$.
  3. Bob calculates a digest of M using the same hash algorithm used by Alice.
  4. Bob calculates $V = [(e_1^{\,h(M)S_2^{-1}}\, e_2^{\,S_1S_2^{-1}}) \bmod p] \bmod q$.
  5. If S_1 is congruent to V, the message is accepted; otherwise, it is rejected.

5.6 Elliptic Curve Digital Signature Scheme

Figure 13.15 gives the general idea behind ECDSS.

Key Generation

Key generation follows these steps:

1) Alice chooses an elliptic curve $E_p(a, b)$.
2) Alice chooses another prime q the private key d.
3) Alice chooses $e_1(\ldots, \ldots)$, a point on the curve.
4) Alice calculates $e_2(\ldots, \ldots) = d \times e_1(\ldots, \ldots)$.
5) Alice's public key is (a, b, p, q, e1, e2); her private key is d.

Downloaded by TONY 562 (tony2802b@gmail.com)

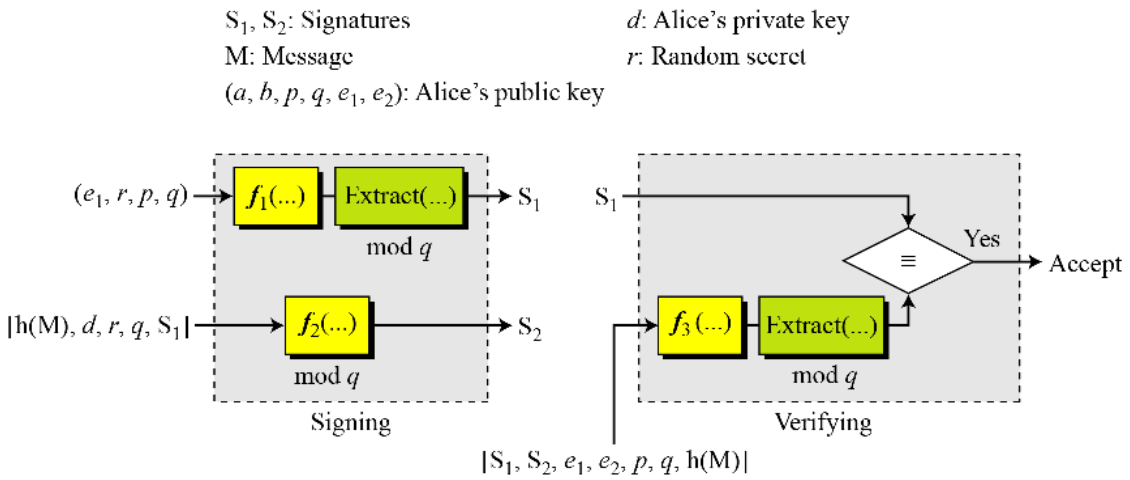


Figure 13.15 General idea behind the ECDSS scheme

Signing and Verifying

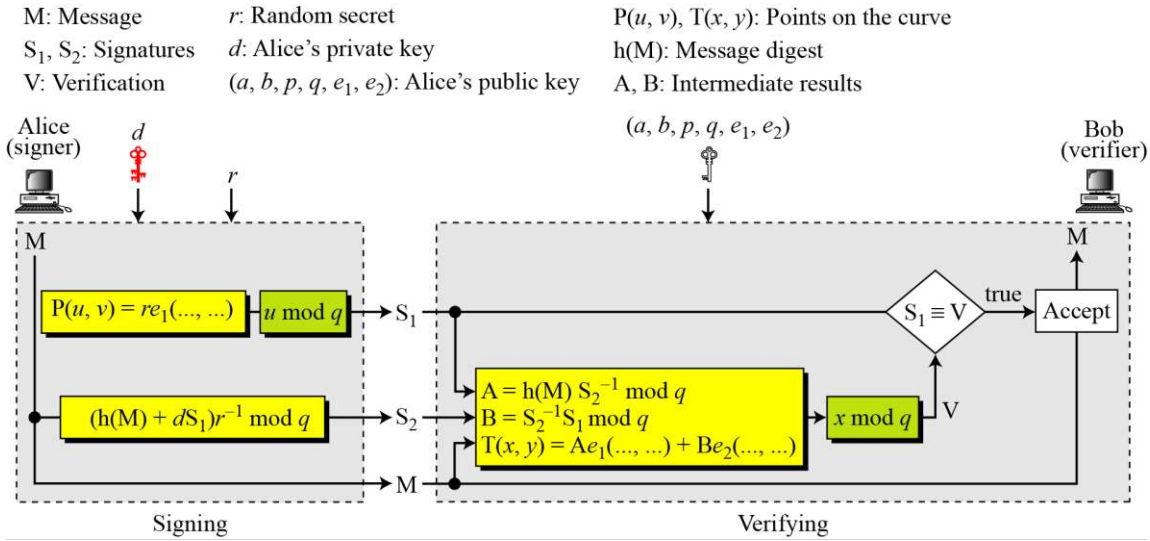


Figure 13.16 The ECDSS scheme

■ **Signing** The signing process consists mainly of choosing a secret random number, creating a third point on the curve, calculating two signatures, and sending the message and signatures.

1. Alice chooses a secret random number r, between 1 and $q - 1$.
2. Alice selects a third point on the curve, $P(u, v) = r \times e_1 (\ldots, \ldots)$.
3. Alice uses the first coordinates of $P(u, v)$ to calculate the first signature S_1. This means $S_1 = u \bmod q$.
4. Alice uses the digest of the message, her private key, and the secret random number r, and the S_1 to calculate the second signature $S_2 = (h(M) + d \times S_1)\, r^{-1} \bmod q$.
5. Alice sends M, S_1, and S_2.

■ **Verifying** The verification process consists mainly of reconstructing the third point and verifying that the first coordinate is equivalent to S_1 in modulo q. Note that the third point was created by the signer using

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

the secret random number r. The verifier does not have this value. He needs to make the third point from the message digest, S_1 and S_2:

1. Bob uses M, S_1, and S_2 to create two intermediate results, A and B:

$$A = h(M)\, S_2^{-1} \bmod q \quad \text{and} \quad B = S_2^{-1}\, S_1 \bmod q$$

Bob then reconstructs the third point $T(x, y) = A \times e_1(\ldots, \ldots) + B \times e_2(\ldots, \ldots)$.

2. Bob uses the first coordinate of $T(x, y)$ to verify the message. If $x = S_1 \bmod q$, the signature is verified; otherwise, it is rejected.

DSS Versus RSA

Computation of DSS signatures is faster than computation of RSA signatures when using the same p.

DSS Versus ElGamal

DSS signatures are smaller than ElGamal signatures because q is smaller than p.

6. Variations and Applications

6.1 Variations

Time Stamped Signatures

- Sometimes a signed document needs to be timestamped to prevent it from being replayed by an adversary.
- This is called timestamped digital signature scheme.
- For example, if Alice signs a request to her Bank, Bob, to transfer some money to Eve, the document can be intercepted and replayed by Eve if there is no timestamp on the document.
- Including the actual date and time on the documents may create a problem if the clocks are not synchronized and a universal time is not used.
- One solution is to use a nonce.
- A nonce is a number that can be used only once, when the receiver receives a document with a nonce, be makes a note that the number is now used by the sender and cannot be used again.
- In other words, a new nonce defines the “present time”; a used nonce defines “past time”.

Blind Signatures:

- Sometimes we have a document that we want to get signed without revealing the contents of the document to the signer.
- David Chaum has developed some patented blind digital signature schemes for this purpose.
- The main idea is as follows:
    a. Bob creates a message and blinds it. Bob sends the blinded message to Alice
    b. Alice signs the blinded message and returns the signature on the blinded message.

Downloaded by TONY 562 (tony2802b@gmail.com)

c. Bob unbinds the signature to obtain a signature on the original message.

Undeniable Digital Signatures

- Undeniable digital schemes are elegant inventions of chaum and van Antwerpen.
- This scheme has three components: a signing algorithm, a verification protocol, and a disavowal protocol.
- The signing algorithm allows Alice to sign a message.
- The verification protocol uses the challenge-response mechanism to involve Alice for verifying the signature. This prevents the duplication and distribution of the signed message without Alice's approval.
- The disavowal protocol helps Alice deny a forged signature. To prove that signature is forgery, Alice needs to take part in the disavowal protocol.

Applications

- Most of these applications directly or indirectly require the use of public keys.
- To use a public key, a person should prove that she actually owns the public key.

For this reason, certificates and certificate authorities (CAs) has been developed.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

1. Symmetric-key distribution

- Symmetric key cryptography is more efficient than asymmetric key cryptography for enciphering large messages.
- In this, however needs a shared secret key between two parties.
- If Alice needs to exchange confidential messages with N people, she needs N different keys.
- If N people need to communicate with each other, N (N-1)/2 keys are needed.
- The number of keys is not the only problem; the distribution of keys is another. If Alice and Bob want to communicate, they need a way to exchange a secret key.

1.1 Key-Distribution Center: KDC

- A practical solution is the use of trusted third party, referred to as a Key-Distribution center (KDC).
- To reduce the number of keys, each person establishes a shared secret key with the KDC as shown in figure 15.1.

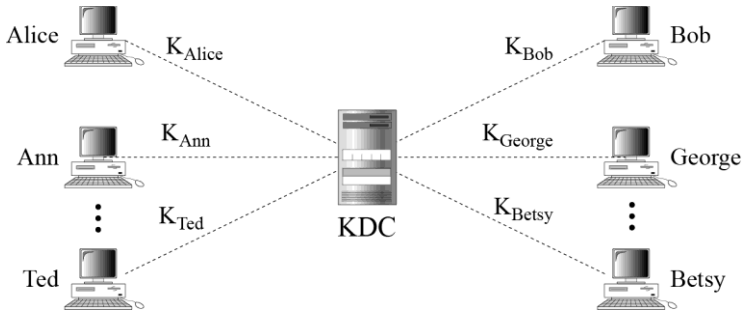


Figure 15.1 Key-distribution center (KDC)

- A secret key is established between the KDC and each member. Alice has a secret key with the KDC, which we refer to as K_{Alice}; Bob has a secret key with the KDC, which we refer to as K_{Bob}. And so on.
- How Alice can send a confidential message to Bob. This process is as follows:
    1. Alice sends a request to the KDC stating that she needs a session (temporary) secret key between herself and Bob.
    2. The KDC informs Bob about Alice's request.
    3. If, Bob agrees, a session key is created between the two.

The secret key between Alice and Bob that is established with the KDC is used to authenticate Alice and Bob to prevent Eve from impersonating either of them.

Flat Multiple KDCs

- When the number of people using a KDC increases, the system becomes unmanageable and a bottleneck can result.
- To solve this problem, we need to have multiple KDCs. We can divide the worlds into domains.

Downloaded by TONY 562 (tony2802b@gmail.com)

- Each domain can have one or more KDCs (for redundancy in case of failure).
- Now if Alice wants to send a confidential message to Bob, who belongs to another domain, Alice contacts her KDC, which in turn contacts the KDC in Bob's domain.
- The two KDC's can create a secret key between Alice and Bob.
- Figure 15.2 shows KDCs all at the same level. We call this flat multiple KDCs.

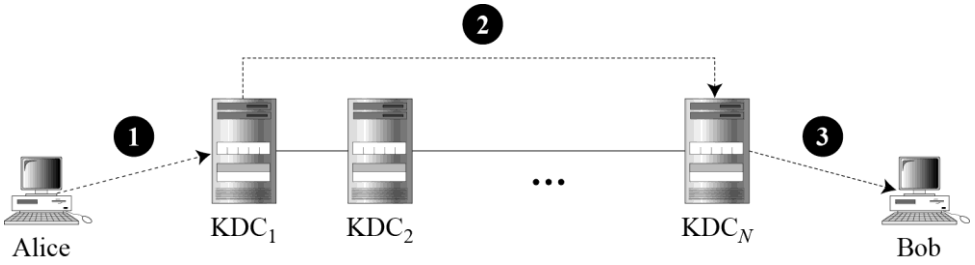


Figure 15.2 Flat multiple KDCs

Hierarchical Multiple KDCs

- The concepts of flat multiple KDCs can be extended to a hierarchical system of KDCs, with one or more KDCs at the top of the hierarchy.
- For example, there can be local KDCs, national KDCs, and international KDCs.
- When Alice needs to communicate with Bob, who lives in another country , she sends her request to a local KDC; the local KDC, relays the request to the national KDC, the national KDC relays the request to an international KDC, the request is then relayed all the way down to the local KDC where Bob lives.
- Figure 15.3 shows a configuration of hierarchical multiple KDCs.

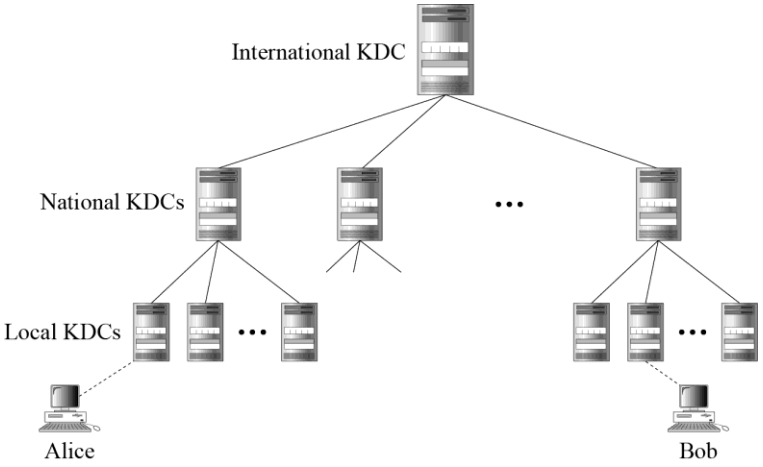


Figure 15.3 Hierarchical multiple KDCs

Session keys:

- A KDC creates a secret key for each member. The secret key can be used only between the member and the KDC, not between two members.
- If Alice needs to communicate secretly with Bob, she needs a secret key between herself and Bob.
- A KDC can create a session key between Alice and Bob, using their keys with the center.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

- The keys of Alice and Bob are used to authenticate Alice and Bob to the center and to each other before the session key is established.
- After communication is terminated, the session key is no longer useful.

A simple protocol using a KDC

Let us see how a KDC can create a session key K_{AB} between Alice and Bob. Figure 15.4 shows the steps.

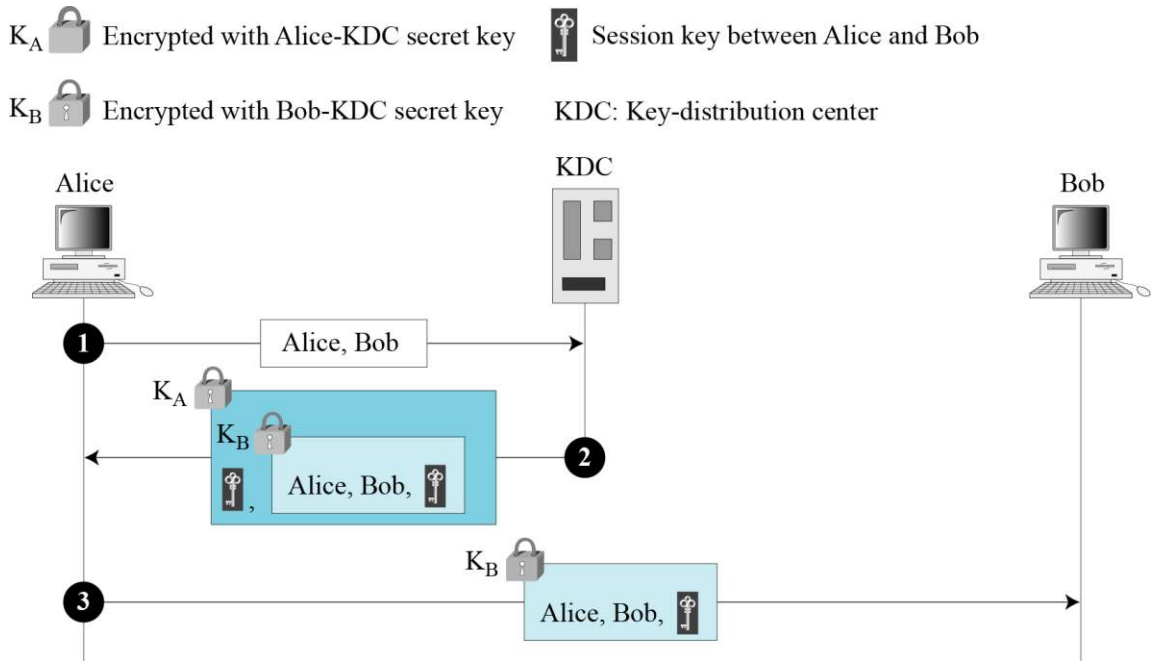


Figure 15.4 First approach using KDC

1. Alice sends a plaintext message to the KDC to obtain a symmetric session key between Bob and herself. The message contains her registered identity and the identity of Bob. This message is not encrypted, it is public. The KDC does not care.
2. The KDC receives the message and creates what is called ticket. The ticket is encrypted using Bob's key (K_B). The ticket contains the identities of Alice and Bob and the session key (K_{AB}). The ticket with a copy of the session key is sent to Alice. Alice receives the message, decrypts it, and extracts the session key.
3. Alice sends the ticket to Bob. Bob opens the ticket and knows that Alice needs to send messages to him using K_{AB} as the session key.

Needham – Schroeder Protocol

- It is foundation for many other protocols.
- This protocol uses multiple challenge-response interactions between parties to achieve a flawless protocol. It uses two nonce: R_A and R_B. Figure 15.5 shows the fine steps used in this protocol.

1. Alice sends a message to the KDC that includes her nonce, R_A, her identity and Bob's identity.
2. The KDC sends an encrypted message to Alice that includes Alice's nonce, Bob's identity, the session key, and an encrypted ticket for Bob. The whole message is encrypted with Alice's key.
3. Alice sends Bob's ticket to him.

Downloaded by TONY 562 (tony2802b@gmail.com)

4. Bob sends his challenge to Alice (RB), encrypted with the session key.
5. Alice responds to Bob's challenge. Note that the response carries RB-1 instead of RB.

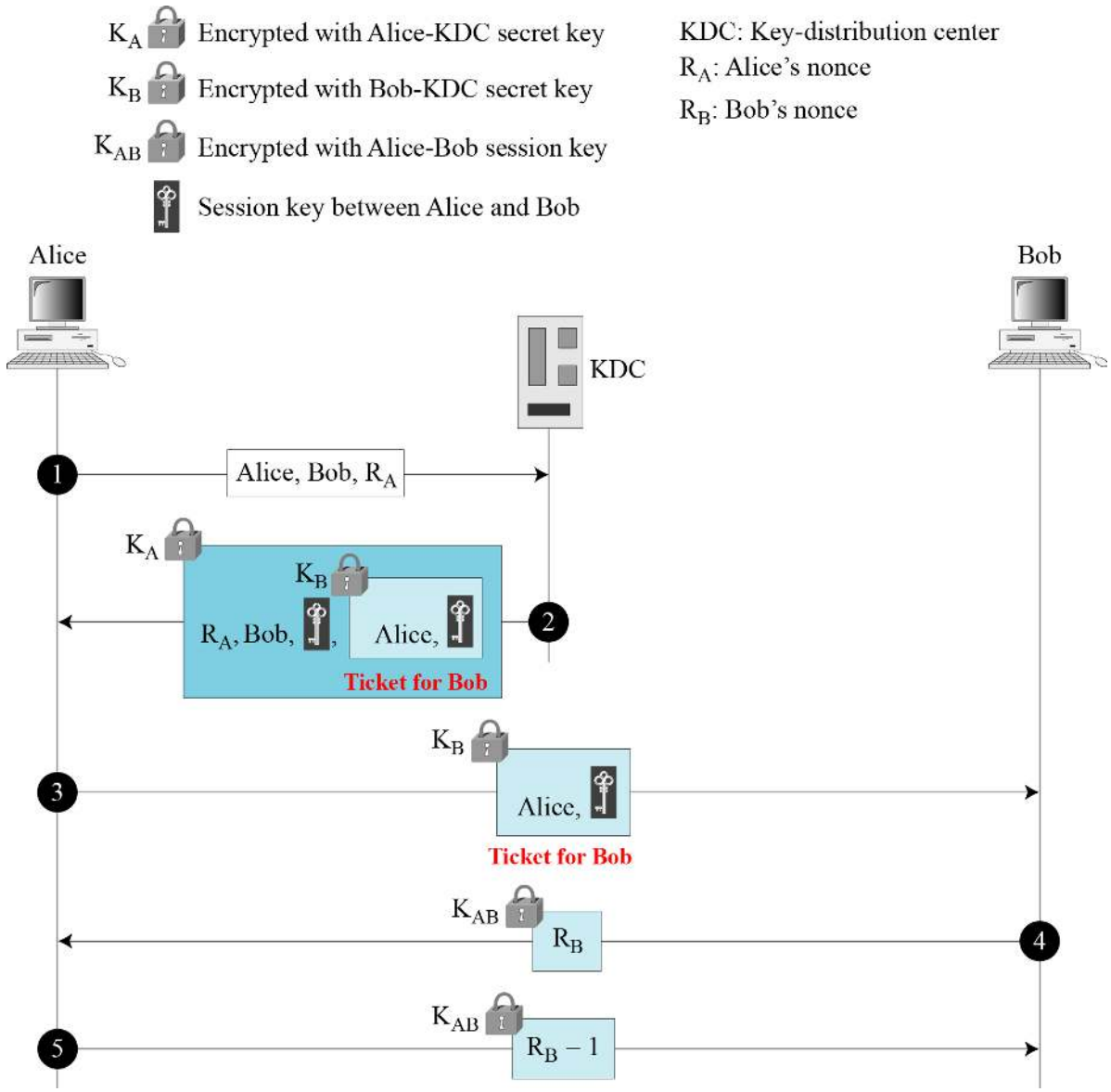


Figure 15.5 Needham-Schroeder protocol

Otway –Rees Protocol

A third approach is the Otway-Rees protocol, another elegant protocol. Figure 15.6 shows this five-step protocol.

1. Alice sends a message to Bob that includes a common nonce, R , the identities of Alice and Bob, and a ticket for KDC that includes Alice's nonce RA, a copy of the common nonce R, and the identities of Alice and Bob.
2. Bob creates the same type of ticket, but with his own nonce RB. Both tickets are sent to the KDC.
3. The KDC creates a message that contains R, the common nonce, a ticket for Alice and a ticket for Bob; the message is sent to Bob. The tickets contain the corresponding nonce, RA or RB, and the session key, KAB.
4. Bob sends Alice her ticket.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

5. Alice sends a short message encrypted with her session key KAB to show that she has the session key.

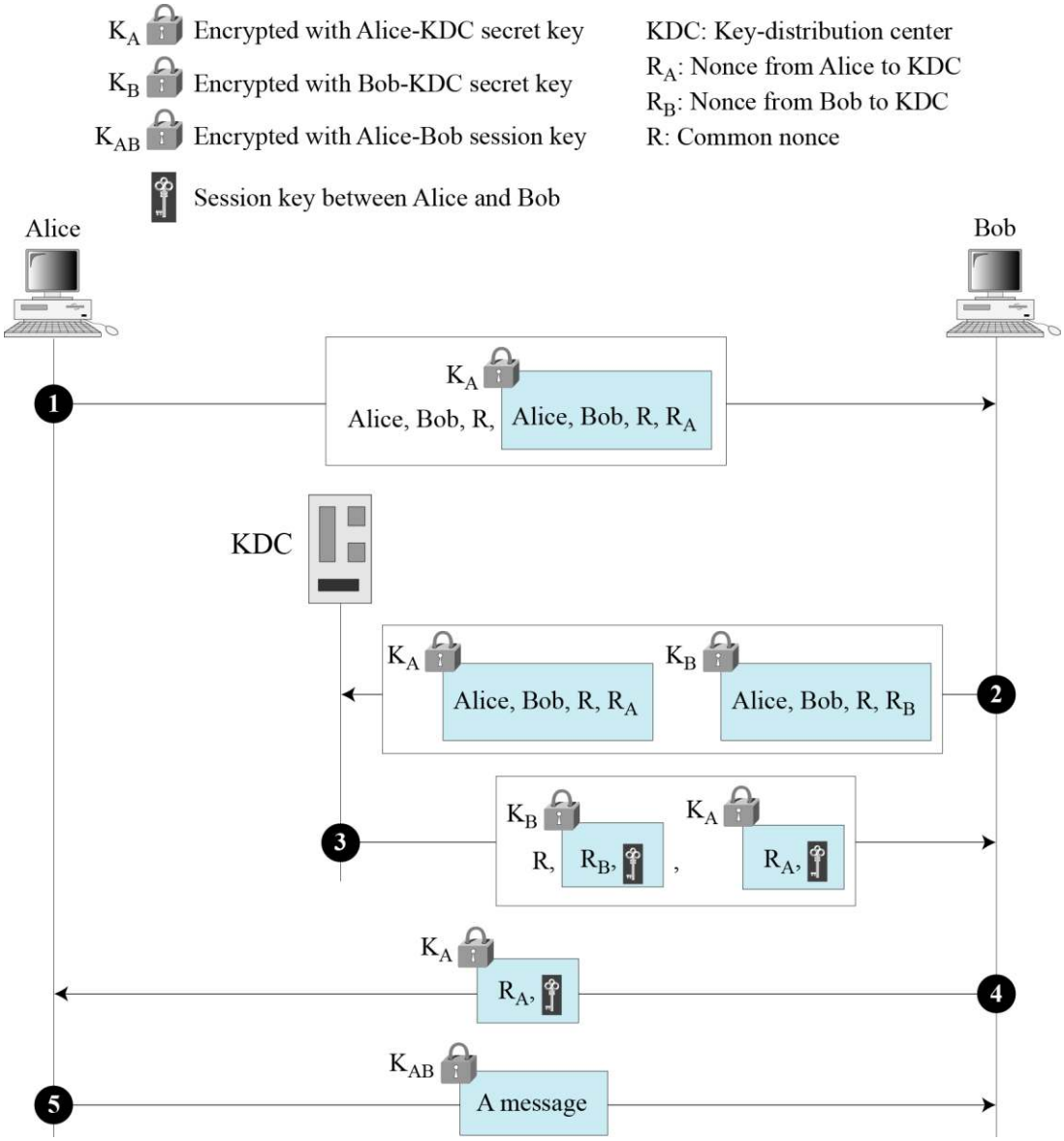


Figure 15.6 Otway-Rees protocol

2. KERBEROS

- Kerberos is an authentication protocol, and at the same time a KDC, that has become very popular.
- Several systems, including Windows 2000, use Kerberos.
- It is named after the three-headed dog in Greek mythology that guards the gates of Hades.

2.1 Servers

- Three servers are involved in the Kerberos protocol: an authentication server (AS), ticket-granting server (TGS), and a real (data) server that provides services to others.
- In our examples and figures, Bob is the real server and Alice is the user requesting service.
- Figure 15.7 shows the relationship between these three servers.

Downloaded by TONY 562 (tony2802b@gmail.com)

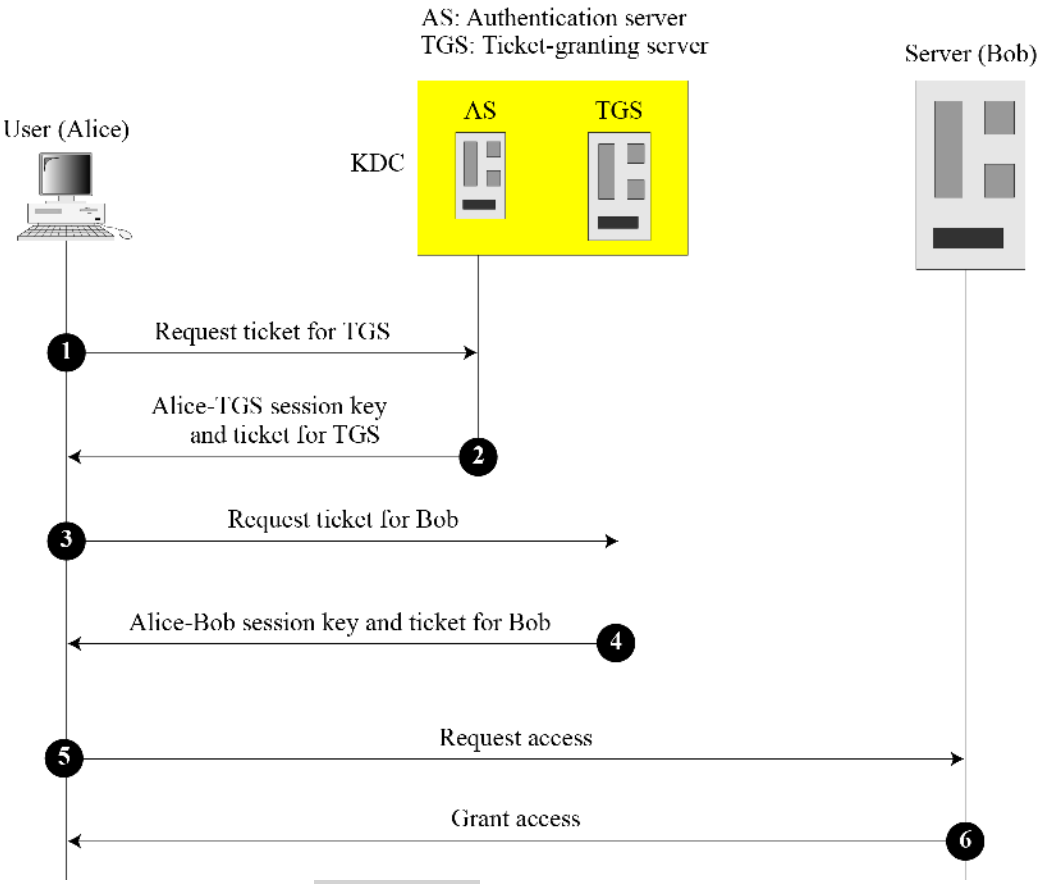


Figure 15.7 Kerberos servers

Authentication Server:

- The authentication server (AS) is the KDC in the Kerberos protocol. Each user registers with the AS and is granted a user identity and a password.
- The AS has a database with these identities and the corresponding passwords.
- The AS verifies the user, issues a session key to be used between Alice and the TGS, and sends a ticket for the TGS.

Ticket-Granting Server:

- TS issues a ticket for the real server (Bob).
- It also provides the session key (K_{AB}) between Alice and Bob.
- Kerberos has separated user verification from the issuing of tickets.
- In this way, though Alice verifies her ID just once with the AS, she can contact the TGS multiple times to obtain tickets for different real servers.

Real server

- The real server (Bob) provides services for the user (Alice).
- Kerberos is designed for a client –server program, such as FTP, in which a user uses the client process to access the server process.
- Kerberos is not used for person-to –person authentication.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Operation

A client process can access a process running on the real server (Bob) in six steps, as shown in figure 15.8.

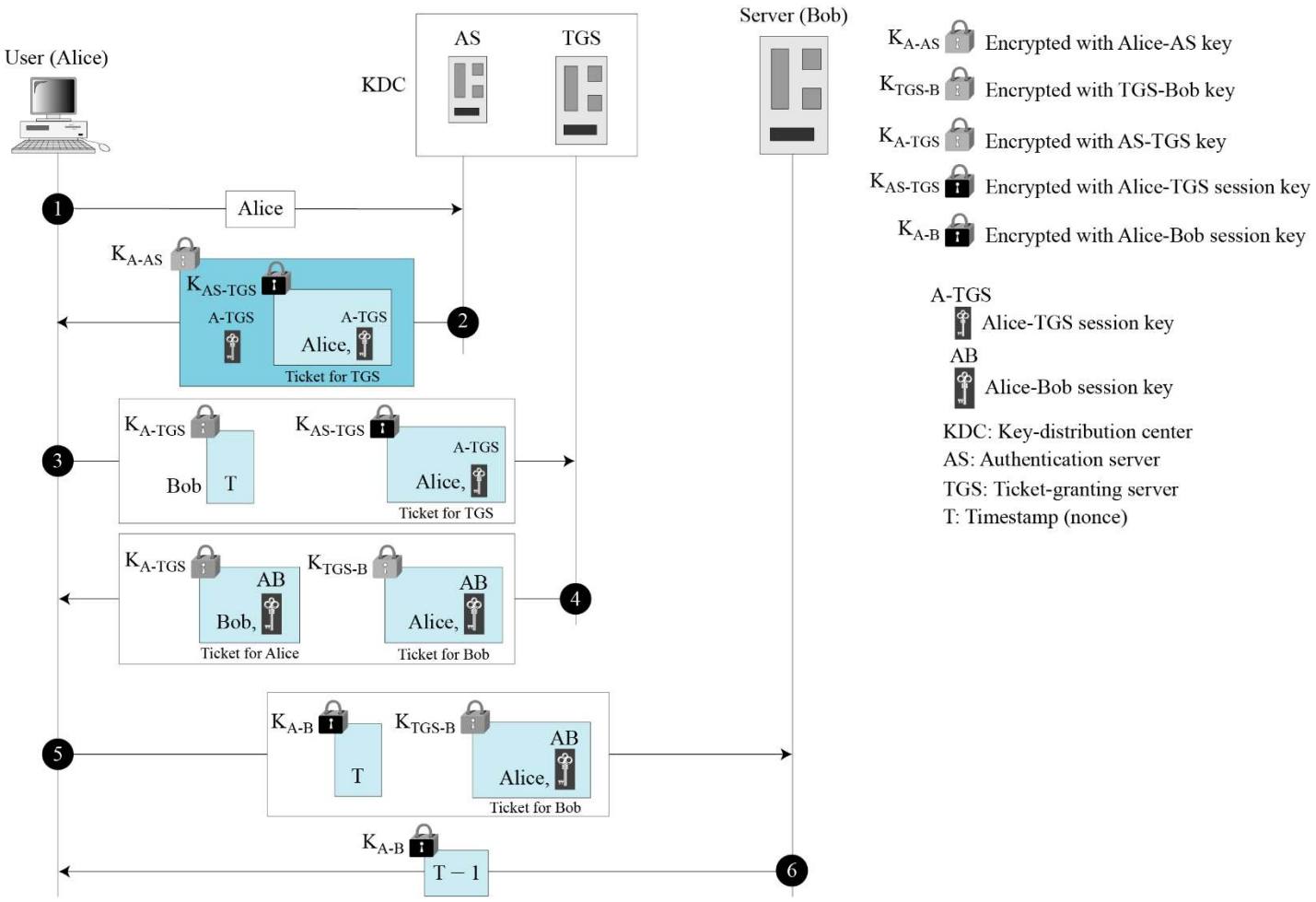


Figure 15.8 *Kerberos example*

1. Alice sends her request to the AS in plaintext using her register identity.
2. The AS sends a message encrypted with Alice's permanent symmetric key, KA-AS. The message contains two items: a session key, KA-TGS that is used by Alice to contact the TGS, and a ticket for the TGS that is encrypted with the TGS symmetric key, KA-TGS.
3. Alice now sends three items to the TGS. The first is the ticket received from the AS. The second is the name of the real server (Bob), the third is a timestamp that is encrypted by KA-TGS and the ticket are extracted.
4. Now, the TGS sends two tickets, each containing the session key between Alice and Bob

   KA-B. The ticket for Alice is encrypted with KA-TGS; the ticket for Bob is encrypted with Bob's key, KTGS-B.
5. Alice sends Bob's ticket with the timestamp encrypted by KA-B.
6. Bob confirms the receipt by adding 1 to the timestamp. The message is encrypted with KA-B and sent to Alice.

Downloaded by TONY 562 (tony2802b@gmail.com)

Using Different Servers

- Note that if Alice needs to receive services from different servers, she need repeat only the last four steps. The first two steps have verified Alice's identity and need not be repeated.

Kerberos Version 5:

The minor differences between version 4 and version 5 are briefly listed below:

1. Version 5 has a longer ticket lifetime.
2. Version 5 allows tickets to be renewed.
3. Version 5 can accept any symmetric-key algorithm.
4. Version 5 uses a different protocol for describing data types.
5. Version 5 has more overhead than version 4

3. Symmetric Key Agreement

Alice and Bob can create a session key between themselves without using KDC. This method of session-key creation is referred to as the symmetric – key agreement.

Here we have two common methods, Diffie Hellman and station – to-station are discussed here.

Diffie and Hellman key-exchange Algorithm

This algorithm only used for key exchange

1. Select a prim number q.
2. Find primitive roots of q.
3. Select a primitive root Ω among them.
4. Select A's private key $X_A < q$.
5. Calculate A's public key $Y_A = \Omega^{XA} \bmod q$.
6. Select B's private key $X_B < q$.
7. Calculate B's public key $Y_B = \Omega^{XB} \bmod q$.
8. A's shared key $k = (Y_B)^{XA} \bmod q$.
9. B's shared key $k = (Y_A)^{XB} \bmod q$.

Here A's shared key equals to B's shared key.

Proof:

A's shared key $K = (Y_B)^{XA} \bmod q$.

$= (\Omega^{XB} \bmod q)^{XA} \bmod q$

$= (\Omega^{XBXA} \bmod q) \bmod q$

$= (\Omega^{XA} \bmod q)^{XB} \bmod q$

$= (Y_A)^{XB} \bmod q$

= B's shared key.

In the above, p, is a large prime number on the order of 300 decimal digits (1024 bits).

Ω, is a generator of order q-1 in the group <Zq*, x>.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

Man-in-the-Attack

The algorithm depicted above is insecure against a man-in-the-middle attack. Suppose Alice and Bob wish to exchange keys, and Darth is the adversary. The attack proceeds as follows (Figure 15.9).

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. Darth also calculates $K2 = (Y_A)^{XD2} \bmod q$.
4. Bob receives Y_{D1} and calculates $K1 = (Y_{D1})^{XB} \bmod q$.
5. Bob transmits Y_B to Alice.
6. Darth intercepts Y_B and transmits Y_{D2} to Alice. Darth calculates $K1 = (Y_B)^{XD1} \bmod q$.
7. Alice receives Y_{D2} and calculates $K2 = (Y_{D2})^{XA} \bmod q$.

At this point, 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. All future communication between Bob and Alice is compromised in the following way.

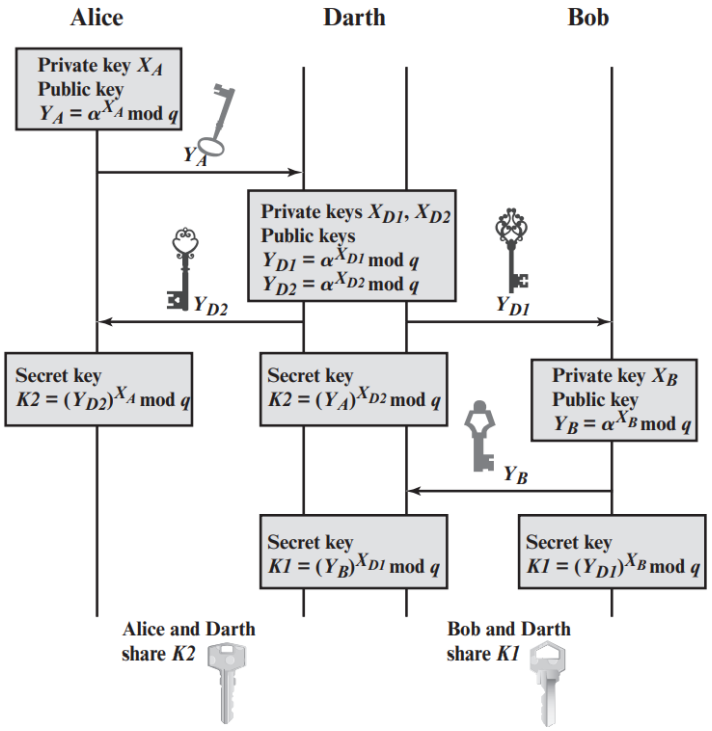


Figure 15.9: Man-in-the-Middle Attack

Example 1:

$Q = 353$

$\Omega = 3$

$X_A = 97 < 353$

$Y_A = \Omega^{XA} \bmod q$

$= 3^{97} \bmod 353$

$= 40.$

Downloaded by TONY 562 (tony2802b@gmail.com)

$X_B = 233 < 353$

$Y_B = \Omega^{XB} \bmod q$

$= 3^{233} \bmod 353$

$= 248.$

A's shared key $= (Y_B)^{XA} \bmod q$

$= (248)^{97} \bmod 353$

$= 160.$

B's shared key $= (Y_A)^{XB} \bmod q$

$= (40)^{233} \bmod 353$

$= 160.$

Example 2:

User A and B use the Diffie-Hellman key exchange technique with a common prime q = 71 and a primitive root $\Omega = 7$.

a) If user A has private key $X_A = 5$, what is A's public key Y_A?
b) If user B has private key $X_B = 12$, what is B's public key Y_B?
c) What is the shared secret key?

a) A's public key

$Y_A = \Omega^{XA} \bmod q$

$= 7^5 \bmod 71$

$= 16807 \bmod 71$

$= 51$

b) B's public key Y_B

$Y_B = \Omega^{XB} \bmod q$

$= 7^{12} \bmod 71$

$= 4$

c) Shared secret key

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

$= 4^5 \bmod 71$

$= 1024 \bmod 71$

$= 30.$

c) Shared secret key

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

$= 51^{12} \bmod 71$

$= 30.$

Example 3:

Consider a Diffie-Hellman Scheme with a common prime q = 11 and a primitive root $\Omega = 2$.

a) Show that 2 is a primitive root of 11.
b) If user A has public key $Y_A = 9$, what is A's private key X_A?

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

c) If user B has public key $Y_B = 3$, what is the shared secret key K?

a)

	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}
a^i	2	4	8	16	32	64	128	256	512	1024
$a^i \bmod 11$ $1 \le i \le 10$	2	4	8	5	10	9	7	3	6	1

$a^i \bmod 11$ where $1 \le i \le 10$ have ten different elements from 1 to 10. So, 2 is a primitive root of 11.

b)

$$Y_A = \Omega^{XA} \bmod q$$
$$9 = 2^{XA} \bmod 11$$
$$9 = 2^6 \bmod 11$$
$$9 = 64 \bmod 11$$
$$\Rightarrow X_A = 6$$

c)

$$\text{Shared secret key } K = (Y_B)^{XA} \bmod q$$
$$= 3^6 \bmod 11$$
$$= 729 \bmod 11$$
$$= 3$$

Example 4:

Alice and Bob use the Diffie–Hellman key exchange technique with a common prime q = 157 and a primitive root a = 5.

a. If Alice has a private key $X_A = 15$, find her public key Y_A.
b. If Bob has a private key $X_B = 27$, find his public key Y_B.
c. What is the shared secret key between Alice and Bob?

a) A's public key

$$Y_A = \Omega^{XA} \bmod q$$
$$= 5^{15} \bmod 71$$
$$= 16807 \bmod 157$$
$$= 79$$

b) B's public key Y_B

$$Y_B = \Omega^{XB} \bmod q$$
$$= 5^{27} \bmod 157$$
$$= 65$$

c) Shared secret key

$$K = (Y_B)^{XA} \bmod q$$
$$= 65^{15} \bmod 157$$
$$= 78$$

c) Shared secret key

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

Downloaded by TONY 562 (tony2802b@gmail.com)

$= 79^{27} \bmod 157$

$= 78$

Example 5:

Alice and Bob use the Diffie-Hellman key exchange technique with a common prime q = 23 and a primitive root a = 5.

a. If Bob has a public key $Y_B = 10$, what is Bob's private key Y_B?
b. If Alice has a public key $Y_A = 8$, what is the shared key K with Bob?
c. Show that 5 is a primitive root of 23.

a)

$Y_A = \Omega^{XA} \bmod q$

$10 = 5^{XB} \bmod 23$

$10 = 5^3 \bmod 23$

⇨ $X_B = 3$

b)

Shared secret key $K = (Y_B)^{XA} \bmod q$

$= 8^3 \bmod 23$

$= 6$

c)

$5^0 \bmod 23 = 1$	$5^1 \bmod 23 = 5$	$5^2 \bmod 23 = 2$	$5^3 \bmod 23 = 10$	$5^4 \bmod 23 = 4$	$5^5 \bmod 23 = 20$	$5^6 \bmod 23 = 8$	$5^7 \bmod 23 = 17$	$5^8 \bmod 23 = 16$	$5^9 \bmod 23 = 11$	$5^{10} \bmod 23 = 9$	$5^{11} \bmod 23 = 22$	$5^{12} \bmod 23 = 18$	$5^{13} \bmod 23 = 21$	$5^{14} \bmod 23 = 13$	$5^{15} \bmod 23 = 19$	$5^{16} \bmod 23 = 3$

From the above table we can say 5 is a primitive root of 23.

3.2 Station–to–Station key Agreement

The station-to–station protocol is a method based on Diffie-Hellman. It uses digital signatures with public key certificates to establish a session key between Alice and Bob as shown in figure 15.12.

The following shows the steps:

- After calculating R1, Alice sends R1 to Bob (steps 1 and 2 in Fig 15.12).
- After calculating R2 and the session key, Bob concatenates Alice's ID, R1 and he signs the result with his private key (steps 3, 4 and 5 in Fig 15.12).
- After calculating the session key, if Bob's signature is verified, Alice concatenate Bob's ID,R1 and R2.she then signs the result with her own private key and sends Bob. The signature is encrypted with the session key (steps 6, 7 and 8 in Fig 15.12).
- If Alice's signature is verified, Bob keeps the session key (steps 9 in Fig 15.12).

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

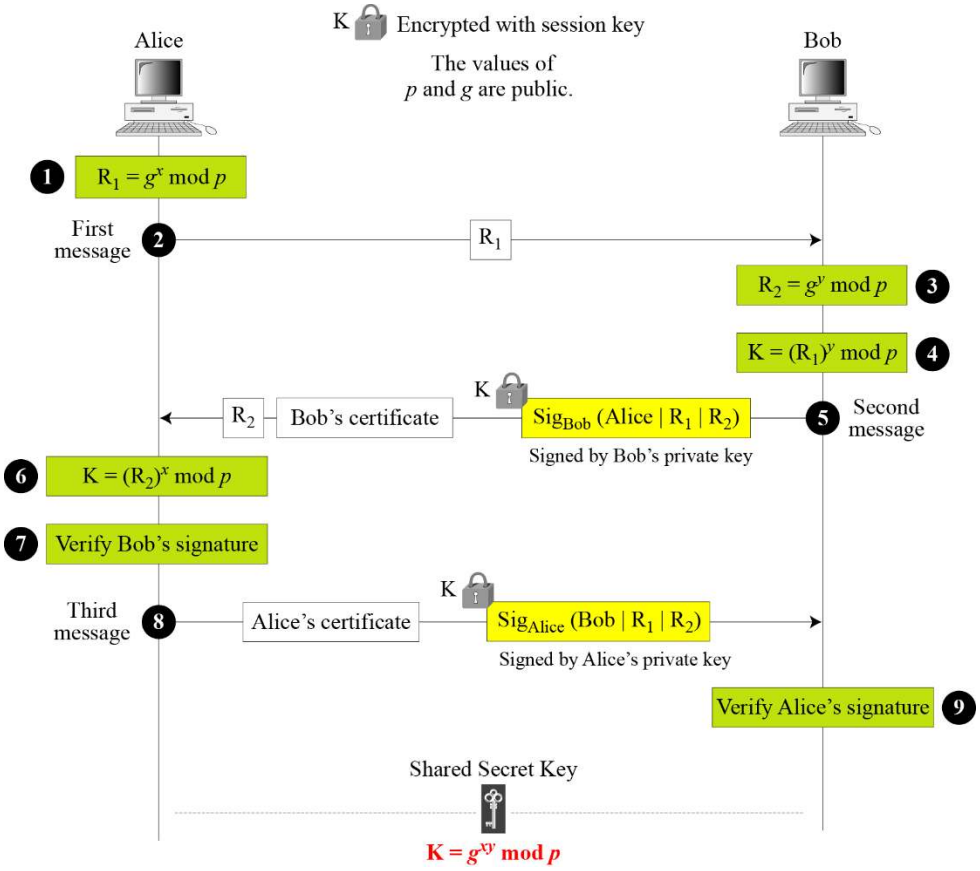


Figure 15.12 Station-to-station key agreement method

4. Public- Key Distribution

- In Asymmetric key cryptography, people do not need to know a symmetric shared key.
- If Alice wants to send a message to Bob, she only needs to know Bob's public key, which is open to the public and available to everyone.
- If Bob needs to send a message to Alice, he only needs to know Alice's public key, which is also known to everyone.
- In public key cryptography, everyone shields a private key and advertises a public key.

4.1 Public Announcement

The native approach to announce public keys publicly. Bob can put his public key on his website or announce it in a local or national newspaper.

- When Alice needs confidential message to Bob, she can obtain Bob's public key from his site or from the newspaper, or even send a message to ask for it.
- This approach, however, is not secure, it is subject to forgery.
- For example, Eve could make such a public announcement.
- Before Bob can react, damage could be done. Eve can fool Alice into sending her a message that is intended for Bob.
- Eve could also sign a document with a corresponding forged private key and make everyone

Downloaded by TONY 562 (tony2802b@gmail.com)

believe it was signed by Bob.

- Eve can intercept Bob's response and substitute her own forged public key for Bob's public key.

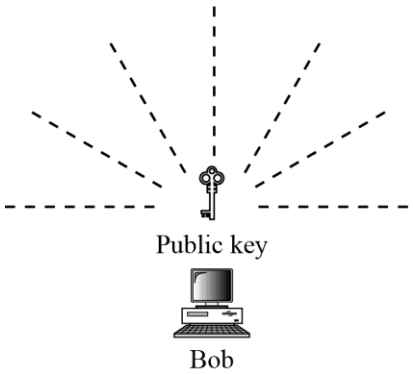


Figure 15.13 Announcing a public key

4.2 Trusted Center

- A more secure approach is to have a trusted center retain a directory of public keys.
- The, directory, like the one used in a telephone system, is dynamically updated. Each user can select a private and public key, keep the private key, and deliver the public key for insertion into the directory.
- The center requires that each user register in the center and prove his or her identity.
- The directory can be publicly advertised by the trusted center.
- The center can also respond to any inquiry about a public key.

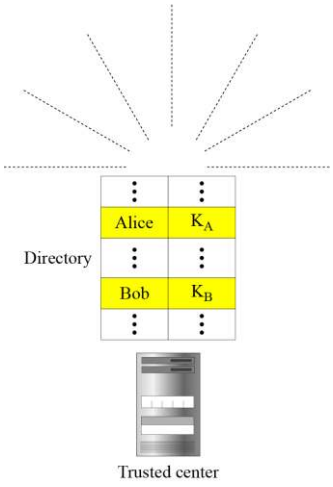


Figure 15.14 Trusted center

4.3 Controlled Trusted Center:

- A higher level of security can be achieved if there are added controls on the distribution of the public key.
- The public key announcements can include a timestamp and be signed by an authority to prevent interception and modification of the response.
- If Alice, needs to know Bob's public key, she can send a request to the center including Bob's name and timestamp.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

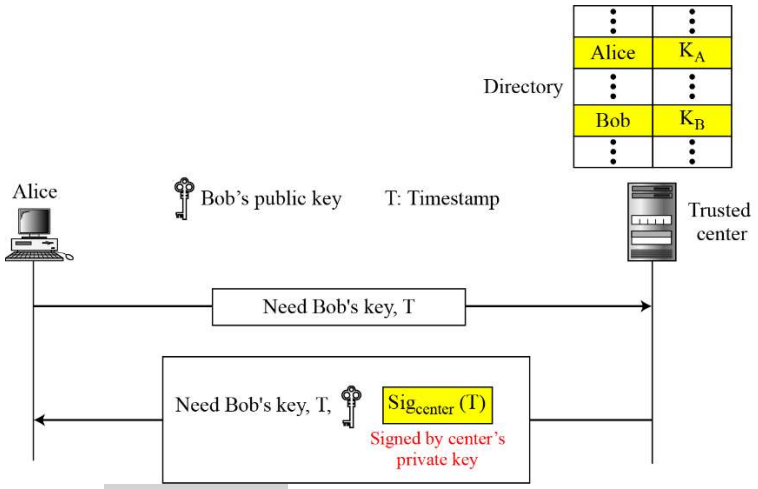


Figure 15.15 Controlled trusted center

4.4 Certification Authority

- The previous approach can create a heavy load on the center if the number of requests is large.
- The alternative is to create public-key certificates.
- Bob wants two things; he wants people to know his public key, and he wants no one to accept a forged public key as his.
- Bob can go to a certification authority (CA), a federal or state organization that binds a public key to an entity and issues a certificate.
- The CA check Bob's identification. If then asks for Bob's public key and a writes it on the certificate.
- To prevent the certificate itself from being forged, the CA signs the certificate with its private key.
- Now Bob can upload the signed certificate, anyone who wants Bob's public key downloads the signed certificate and uses the center's public key to extract Bob's public key. Figure 15.16 shows the concept.

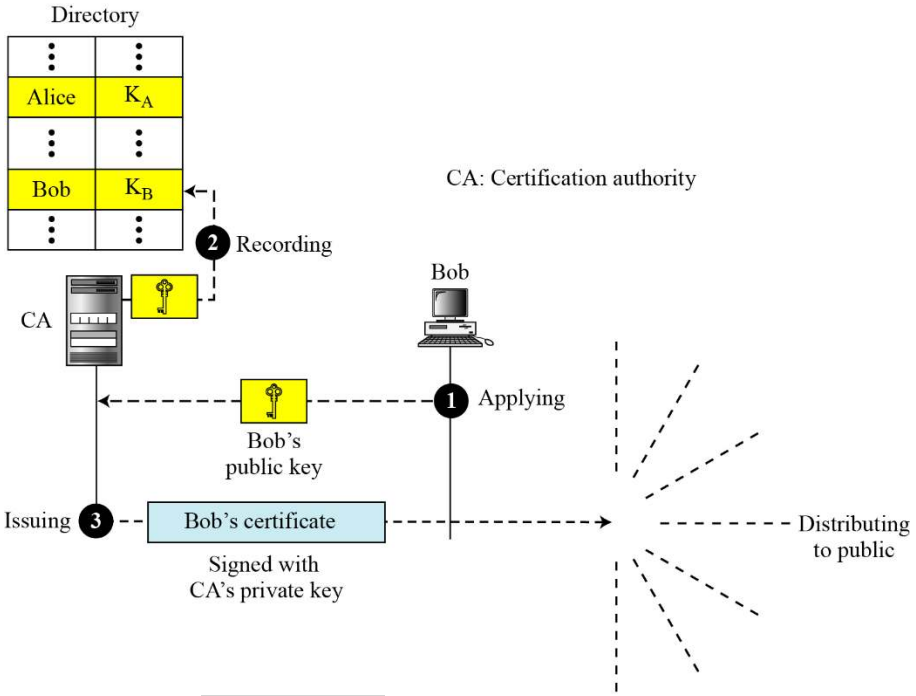


Figure 15.16 Certification authority

Downloaded by TONY 562 (tony2802b@gmail.com)

4.5 X.509

- Although the use of a CA has solved the problem of public key fraud, it has created a side-effect. Each certificate may have a different format.
- If, Alice wants to use a program to automatically download different certificates and digests belonging to different people, the program may not be able to do this.
- One certificate may have the public key in one format and another in a different format.
- The public key may be on the first line in one certificate, and on the third line in another, anything that needs to be used universally must have a universal format.
- The ITU has designed X.509.

Certificate

Figure 15.17 shows the format of a certificate.

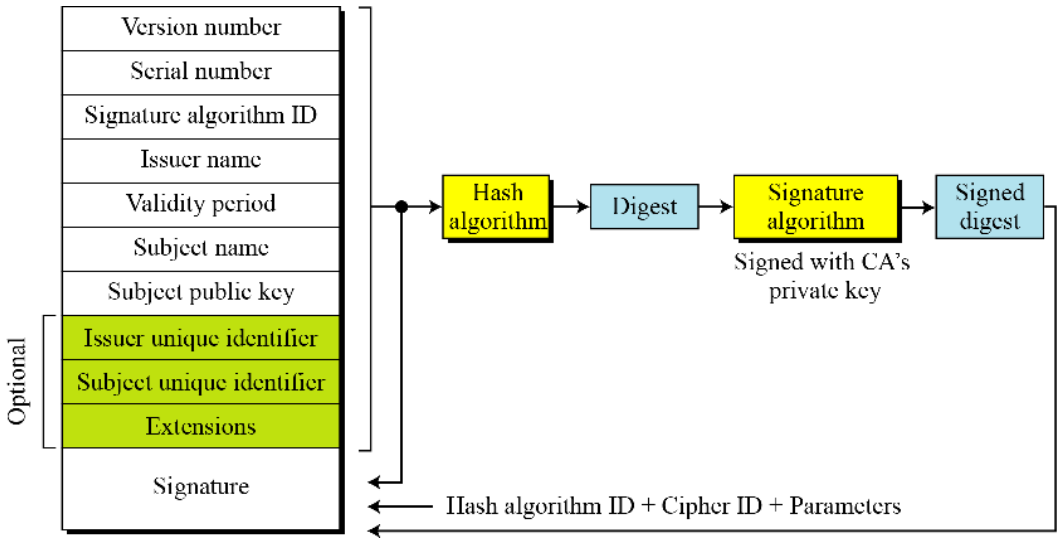


Figure 15.17 X.509 certificate format

A certificate has the following fields:

- **Version number:** This field defines the version of X.509 of the certificate. The version number started at 0; the current version is 2.
- **Serial number:** This field defines a number assigned to each certificate. The value is unique for each certificate issuer.
- **Signature algorithm ID:** This field identifies the algorithm used to sign the certificate. Any parameter that is needed for the signature is also defined in this field.
- **Issuer name:** This field identifies the certification authority that issued the certificate. The name is normally a hierarchy of strings that defines a country, a state, organization, department, and so on.
- **Validity Period:** This field defines the earliest time and the latest time the certificate is valid.
- **Subject name:** This field defines the entity to which the public key belongs. It is also a hierarchy of strings. Part of the field defines what is called the common name, which is the actual name of the beholder of the key.
- **Subject public key:** This field defines the Owner's public key, the heart of the certificate. This field also defines the corresponding public-key algorithm and its parameters.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

- **Issuer unique identifier:** This optional field allows two issuers to have the same issuer field value, if the issuer unique identifiers are different.
- **Subject unique identifier:** This optional field allows two different subjects to have the same subject field value, if the subject unique identifiers are different.
- **Extensions:** This optional field allows issuers to add more private information to the certificate.
- **Signature:** This field is made of three sections. The first section contains all other fields in the certificate. The second section contains the digest of the first section encrypted with the CA's public key. The third section contains the algorithm identifier used to create the second section.

Certificate Renewal

- Each certificate has a period of validity. If there is no problem with the certificate, the CA issues a new certificate before the old one expires.
- The process is like the renewal of credit cards by a credit card company; the credit card holder normally receives a renewed credit card before the one expires.

Certificate Revocation

In some cases a certificate must be revoked before its expiration. Here are some examples:

a) The user's private key might have been comprised.

b) The CA is no longer willing to certify the user. For example, the user's certificate relates to an organization that she no longer works for.

c) The CA's private key, which can verify certificates, may have been compromised. In this case, the CA needs to revoke all unexpired certificates.

The revocation is done by periodically issuing a certificate revocation list (CRL). Figure 15.18 shows the certificate revocation list.

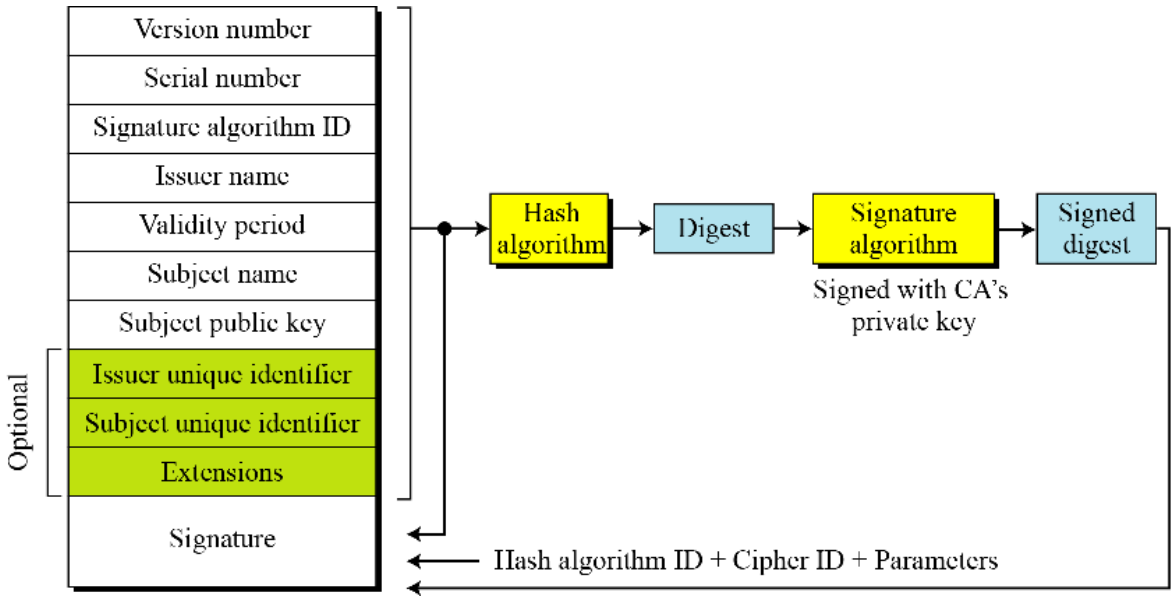


Figure 15.17 Certificate revocation format

Downloaded by TONY 562 (tony2802b@gmail.com)

A certificate revocation list has the following fields:

- **Signature algorithm ID:** This field is the same as the one in the certificate.
- **Issuer name:** this field is the same as the one in the certificate.
- **This update date:** This field defines when the list is released.
- **Next update date:** This field defines the next date when the new list will be released.
- **Revoked certificate:** This is a repeated list of all unexpired certificates that have been revoked. Each list contains two sections: user certificate serial number and revocation date.
- **Signature:** This field is the same as the one in the certificate list.

Delta Revocation

- To make revocation more efficient, the delta certificate revocation list, has been introduced.
- A delta CRL, is created and posted on the directory if there are changes after this update date and next update date.

4.6 Public-Key Infrastructures (PKI)

- It is a model for creating, distributing, and revoking certificates based on the X.509.
- The internet Engineering Task Force has created the Public-Key Infrastructure X.509 (PKIX).

Duties

Several duties have been defined for a PKI. The most important ones are shown in figure 15.19.

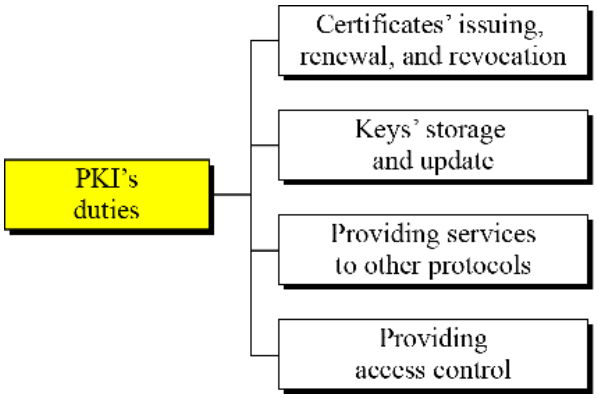


Figure 15.19 Some duties of a PKI

Trust Model

It is not possible to have just one CA issuing all certificates for all users in the world.

There should be many CAs, each responsible for creating, sorting, issuing and revoking a limited number of certificates. The trust model defines rules that specify how a user can verify a certificate received from a CA.

Hierarchical Model

- In this model there is a tree type structure with a root CA. The root CA has a self- signed, self-issued certificate; it need to be trusted by other CAs and users for the system to work.

This document is available free of charge on studocu

Downloaded by TONY 562 (tony2802b@gmail.com)

- Figure 15.20 shows a trust model of this kink with three hierarchical levels. The number of levels can be more than three in a real situation.

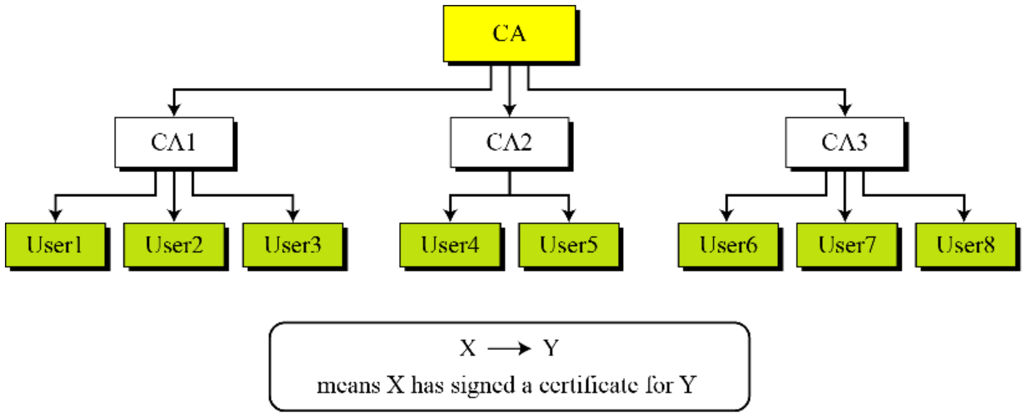


Figure 15.20 PKI hierarchical model

The figure shows that the CA (the root) has signed certificates for CA1, CA2, and CA3; CA1 has signed certificates for User1, User2, and User3; and so on. PKI uses the following notation to mean the certificate issued by authority X for entity Y.

X<<Y>>

Mesh model

- The hierarchical model may work for an organization or a small community.
- A larger community may need several hierarchical structures connected together. One method is to use a mesh model to connect the roots together.
- In this model, each root is connected to every other root as shown in figure 15.21.

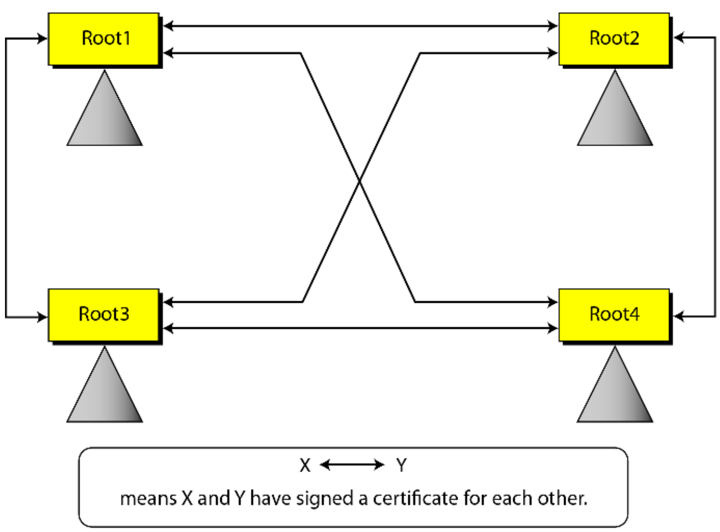


Figure 15.21 Mesh model

= =

Downloaded by TONY 562 (tony2802b@gmail.com)