ELEC 473 Cryptography and Network Security

7. Data Integrity Algorithms

Instructor: Dr. Jianbing Ni

Fall 2021

Review of C6 Topic 3

Elliptic Curve over R

- An *elliptic curve* E over the field of real numbers R is the set of points (*x*,*y*) with *x* and *y* in R that satisfy the equation $E = \{(x,y) \in \boldsymbol{R} \times \boldsymbol{R} \mid y^2 = x^3 + ax + b\} \cup \{\mathrm{O}\}$, Where $a, b \in \boldsymbol{R}$, $4a^3 + 27b^2 \neq 0$ and O is called the *point at infinity*.
- Group Operation + Over E: Given $P, Q \in E, P = (x_1, y_1), Q = (x_2, y_2)$, to compute $R = P + Q = (x_3, y_3)$

Elliptic Curve Modulo a Prime

- An elliptic curve E over Z_p is the set of points (x, y) $\in Z_p \times Z_p$ to the congruence $y^2 \equiv x^3 + ax + b \mod p$, where a, b $\in Z_p$ are constants such that $4a^3 + 27b^2 \not\equiv 0 \mod p$, together with a special point O called the point at infinity.
- How to generate E(Z_p): *c* in Z_p is a quadratic residue (Q.R.) modulo p if it has a square root in Z_p
- Euler's Theorem: *c* in $(Z_p)^*$ is a Q.R. $\Leftrightarrow$ $c^{(p-1)/2} = 1$ in Z_p
- When p = 3 mod 4, if c $\in Z_p^*$ is Q.R. then $\sqrt{c} = c^{(p+1)/4}$ in Z_p

Elliptic Curve ElGamal: Setup, KeyGen, Encryption, and Decryption

- Security of ElGamal: Elliptic Curve DLP
- Security of ECC vs. RSA/ElGamal

Outline

- Message Authentication Code
- Hash Function
- Digital Signature

Authentication

User authentication: To verify the identity of the user.

Message authentication: To verify the authenticity and integrity of received messages, and also verify the order and timeliness of messages.

- The authentication mechanism needs to generate an authentication tag.
- Authentication tag: The value used to authenticate the message.
- The methods of generating authentication tags are: message authentication code (MAC), and digital signature

Massage Authentication Code (MAC)

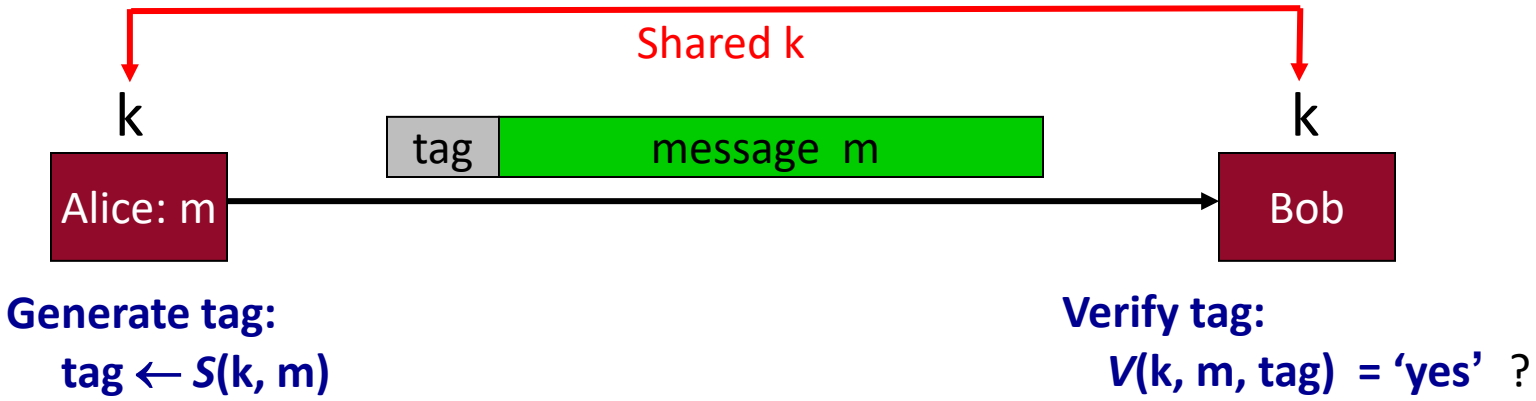
A four-tuple $I=(M, T, K, H)$, where the following conditions are satisfied:

- M is a set of possible messages;
- T is a finite set of possible authentication tags (MAC);
- K, the key space, is a finite set of possible keys;
- For each $k \in K$, there is a function $H_k \in H$, each $H_k: M \rightarrow T$.

- Message Authentication Code (Keyed hash function) $H_k: M \rightarrow T$
- Unkeyed Hash Function $H: M \rightarrow T$

Message Authentication Code (MAC)

A fixed-length value used as an authentication tag/MAC generated from the message by a key-controlled public function (MAC function).



Message Authentication Codes

If only Alice and the Bob know *k*, and the MAC of Bob is consistent with the received MAC.

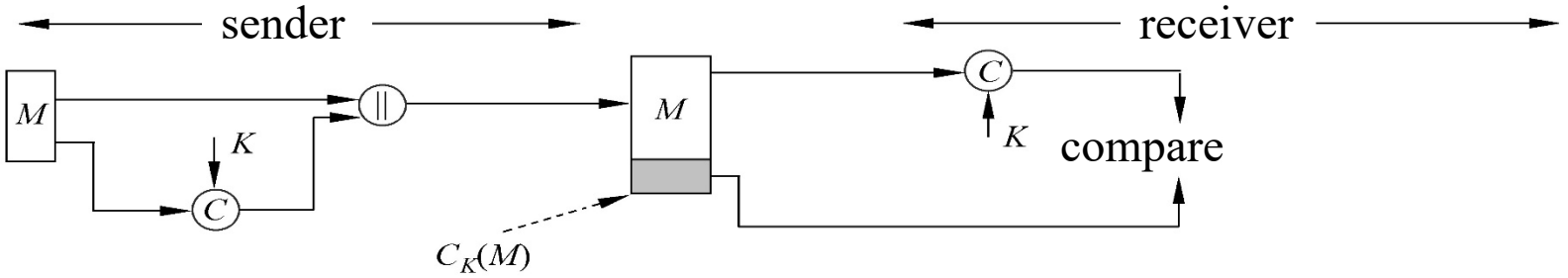
- Integrity: Bob believes that the message sent by Alice has not been tampered.
- Authenticity: Bob believes that Alice is not impersonating.

MAC provides authentication without confidentiality.

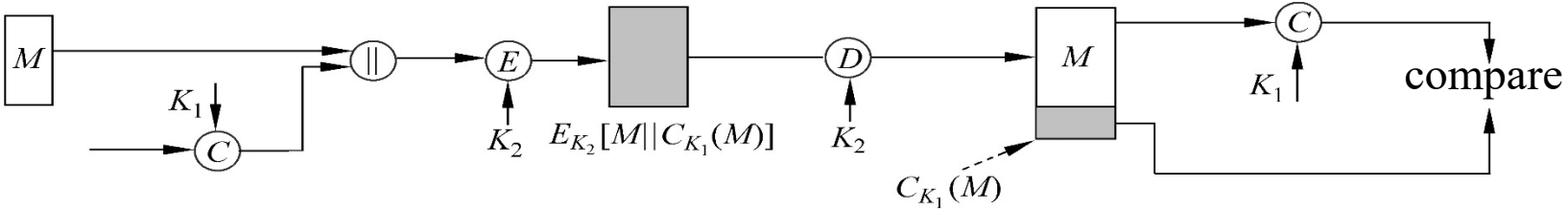
Examples:

Protecting public binaries on disk.

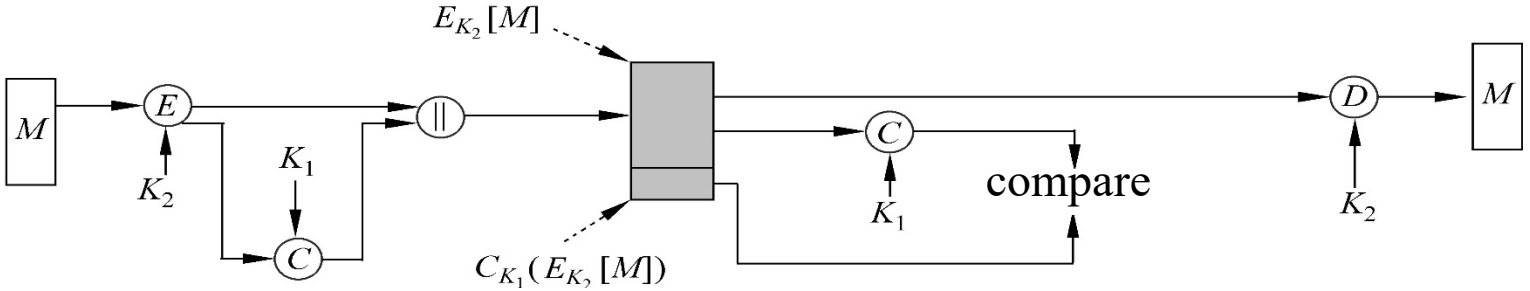
Protecting banner ads on web pages.



(a) Message authentication



(b) Authenticity and confidentiality: Authentication on plaintext



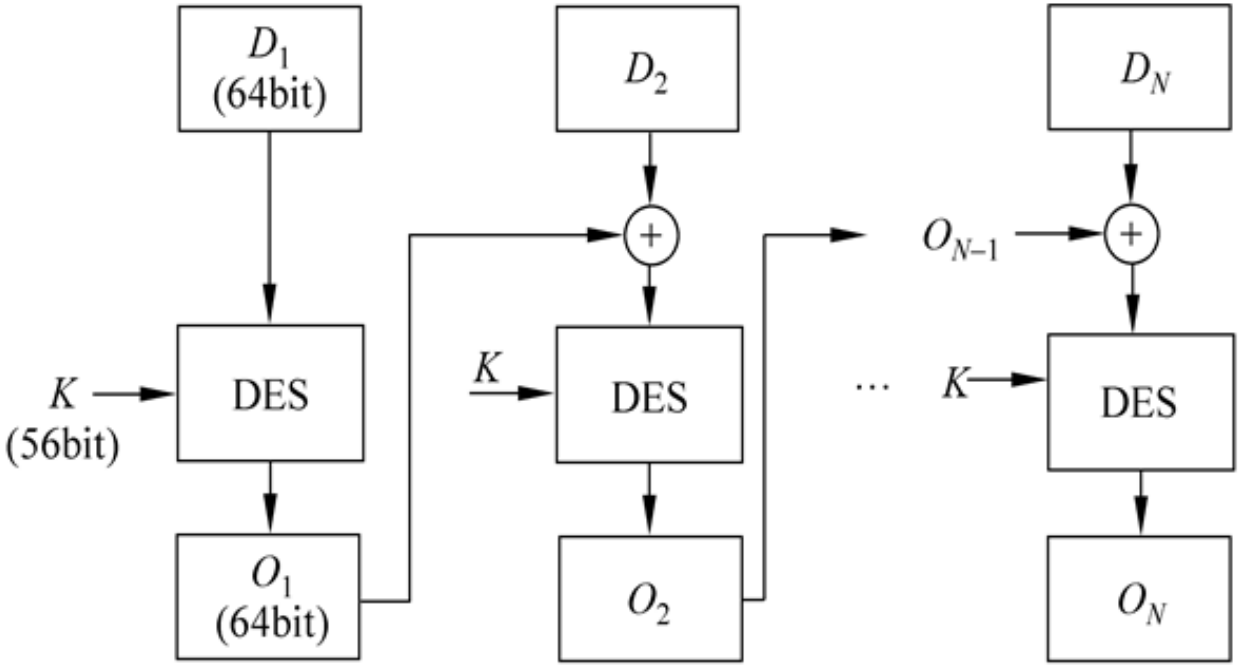
(c) Authenticity and confidentiality: Authentication on ciphertext

MAC Algorithm (One-key MAC)

One of the most widely used message authentication codes.

- The algorithm is based on the DES algorithm with the CBC mode.
- Its initial vector takes a zero vector.
- The data is divided into 64-bit blocks D_1, D_2, ..., D_N.
- If the last block is less than 64 bits, some 0s can be padded on the right.

MAC Algorithm (One-key MAC)



$$O_1 = E_K(D_1)$$
$$O_2 = E_K(D_2 \oplus O_1)$$
$$O_3 = E_K(D_3 \oplus O_2)$$
$$\vdots$$
$$O_N = E_K(D_N \oplus O_{N-1})$$

The MAC is O_N or the leftmost m bits of O_N, where $16 \leq m \leq 64$.

Attacks on MAC

(1) The attackers use the exhaustive search attack to find the secret key.

$$\text{Given } M||C_K(M) = \text{MAC, to find } K$$

(2) Some attacks do not need to find the key used to generate the MAC. An attacker can fake a message for the given authentication tag.

Attack MAC (Find K)

Round 1: Given M_1 and MAC_1, where $\text{MAC}_1 = C_K(M_1)$, calculate $\text{MAC}_i = C_{Ki}(M_1)$ for all 2^k possible keys to yield 2^{k-n} possible keys.

Round 2: Given M_2 and MAC_2, where $\text{MAC}_2 = C_K(M_2)$, calculate $\text{MAC}_i = C_{Ki}(M_2)$ for 2^{k-n} possible keys obtained in round 1, to obtain $2^{k-2\times n}$ possible keys.

In this way, if $k = \alpha n$, the above attack mode requires α rounds on average.

If the key length k is 80 bits and the MAC length n is 32 bits, then about 2^{48} possible keys will be generated in round 1, 2^{16} possible keys will be generated in round 2, and the correct K can be found in round 3.

Attack MAC (Fake Message)

Let $M=(X_1 \| X_2 \| \ldots \| X_m)$, where X_i $(i=1,\ldots,m)$ is 64-bit block,

$$\Delta(M) = X_1 \oplus X_2 \oplus \cdots \oplus X_m$$

$$C_K(M) = E_K\left[\Delta(M)\right]$$

The encryption algorithm is ECB-DES. Thus, the key size is 56 bits and the MAC size is 64 bits. If the adversary gets $M \| C_K(M)$, then the adversary will need to do 2^{56} encryptions using exhaustive search.

However, the adversary can also hack the system in the following way:

$$X_1, \cdots, X_{m-1} \rightarrow Y_1, \cdots, Y_{m-1}$$

$$Y_m = Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{m-1} \oplus \Delta(M)$$

$$X_m \rightarrow Y_m$$

Forge a new message

$$M' = Y_1 \| Y_2 \| \cdots \| Y_{m-1} \| Y_m$$

M and M' has the same MAC

Requirements of MAC

Assuming that the adversary knows the MAC algorithm C but does not know the key K:

- If the adversary gets M and $C_K(M)$, recovering the key K such that MAC(M)= $C_K(M)$ is computationally infeasible.
- If the adversary gets M and $C_K(M)$, constructing a new message M' that satisfies $C_K(M') = C_K(M)$ is computationally infeasible.
- $C_K(M)$ is uniformly distributed: Two messages M and M' are randomly selected, $\Pr[C_K(M)=C_K(M')]=2^{-n}$, where n is the size of MAC.

Summary

- Message Authentication Codes
- One-Key MAC
- Attacks on MAC
- Requirements of MAC

Practice Question of C6T3

1. On the elliptic curve over the real numbers $y^2 = x^3 - 36x$, let P=(-3.5, 9.5) and Q=(-2.5, 8.5). Find $P + Q$ and $2P$.

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8.5 - 9.5}{-2.5 - (-3.5)} = -1$$

$$x_3 = \lambda^2 - x_1 - x_2 = (-1)^2 - (-3.5) - (-2.5) = 7$$

$$y_3 = (x_1 - x_3)\lambda - y_1 = (-3.5 - 7)(-1) - 9.5 = 1$$

$$P + Q = (7, 1)$$

$$\lambda = \frac{3x_1^2 + a}{2y_1} = \frac{3 \times (-3.5)^2 + (-36)}{2 \times 9.5} = \frac{3}{76}$$

$$x_3 = \lambda^2 - 2x_1 = \left(\frac{3}{76}\right)^2 - 2(-3.5) = 7.001$$

$$y_3 = (x_1 - x_3)\lambda - y_1 = (-3.5 - 7.001)\frac{3}{76} - 9.5 = -9.9$$

$$2P = (7.001, 9.9)$$

Practice Question of C6T3

2. Consider the elliptic curve over Z_{23}: $y^2=x^3+x+1$ mod 23. Please list all the non-negative points in the quadrant from (0, 0) through (22, 22) that satisfy the equation mod 23.

x.	x^3+x+1 mod 23.	quad. res?	y.
0	1	yes	1, 22.
1	3	yes	7, 16
2	11	no	
3	8	yes	10, 13
4	0	~~no~~ yes	0, 23.
5	16	yes	4, 19.
6	16	yes	4, 19.
7	6	yes	11, 12.
8	15	no	
9	3	yes	7, 16
10	22	no	
11	9	yes	3, 20
12	16	yes	4, 19
13	3	yes	7, 16
14	22	no	
15	10	no	
16	19	no	
17	9	yes	3, 20
18	9	yes	3, 20
19	2	yes	5, 18
20.	17.	no	
21	14	no	
22	22	no	

All the non-negative points:

(0,1), (0,22), (1,7), (1,16), (3,10), (3,13), (4,0), (5,4), (5,19), (6,4), (6,19), (7,11), (7,12), (9,7), (9,16), (10,3), (10,20), (12,4), (12,19), (13,7), (13,16), (17,3), (17,20), (18,3), (18,20), (19,5), (19,18)

Thanks

ELEC 473 Cryptography and Network Security

7. Data Integrity Algorithms

Instructor: Dr. Jianbing Ni

Fall 2021

Review of C7 Topic 1

Message Authentication

- The authentication tag to verify the authenticity and integrity of received messages, and can verify the order and timeliness of messages
- Methods: message authentication codes (MAC), and digital signatures

Message Authentication Code: A fixed-length value generated from the message by a key-controlled public function (MAC function).

- Goals: the message sent by Alice has not been tampered; Alice is not impersonating.
- MAC modes: message authentication, authenticity and confidentiality (authentication on plaintext, authentication on ciphertext)
- MAC Algorithm (One-key MAC): based on the DES algorithm with the CBC mode
- Security of MAC: finding MAC key or faking message
- Requirements of MAC: 1. computationally infeasible to recover the key; 2. computationally infeasible to forge a message; 3. MAC is uniformly distributed

Outline

- Message Authentication Code
- Hash Function
- Digital Signature

Hash Function

By a ***hash function***, it means a map

$H: \{0,1\}^* \to \{0,1\}^n, n \in Z.$

hash function maps arbitrarily long strings to strings of fixed length.

Input: the data of an arbitrary size

Output: A fixed-size value, (hash value, message digest, or hash)

A way to ensure data integrity

Example The map that sends

$b_1 \cdots b_k$ in $\{0, 1\}^*$ to

$b_1 \oplus \cdots \oplus b_k$ is a hash function.

Preimage Problem

Preimage Problem

A hash function H: M $\rightarrow$ T and an element $y \in$ T

Find: $m \in$ M such that H(m)=y

Preimage resistance: it is infeasible to find m$\in$M such that H(m)=y

Second Preimage Problem

Second Preimage Problem

A hash function H: M $\rightarrow$ T and an element $m \in$ M

Find: $m' \in$ M such that $m' \neq m$ and H(m')=H(m)

Second preimage resistance: It is infeasible to find $m' \in$ M such that $m' \neq m$ and H(m')=H(m)

Collision Resistance

Let H: M $\rightarrow$T be a hash function (|M| >> |T|)

A **collision** for H is a pair m_0 , $m_1 \in$ M such that:

$$H(m_0) = H(m_1) \quad \text{and} \quad m_0 \neq m_1$$

A function H is **collision resistant** if for all (explicit) "eff" algs. **A**:

$$\mathbf{Adv_{CR}[A, H] = Pr[A \text{ outputs collision for H}]}$$

is "neg".

Example: SHA-256 (outputs 256 bits)

- Stronger than preimage resistance? ✓
- Stronger than second preimage resistance? ✓

Collision Resistance

A ***collision*** of H is a pair $(m, m') \in M^2$ for which $m \neq m'$ and $H(m) = H(m')$.

There are collisions of all hash functions **because they are not injective**.

- The function H(m) is ***weak collision resistant*** if it is infeasible to **compute a collision (m, m') for a given $m \in M$**.
- The function H(m) is ***(strong) collision resistant*** if it is infeasible to **compute any collision (m, m') of H**.

Collision Resistance

Which of the following statements about hash function is incorrect?

A. If a hash function is strong collusion resistant, it must be second pre-image resistant.

B. If an adversary can solve the preimage problem of the hash function, he must be able to find a collusion.

C. If a hash function is weak collusion resistant, it must be second pre-image resistant.

D. If an adversary can find a collusion of the hash function, it must be able to solve the second pre-image problem.

Birthday Attack

We describe a simple attack on hash functions $\mathbf{H}: \Omega^* \rightarrow \Omega^n$ called the **birthday attack**.

It attacks the ***strong collision resistance*** of H. The attack is based on the birthday paradox.

Birthday Paradox

Birthday paradox: Suppose a group of people are in a room. What is the probability that two of them have the same birthday?

Suppose there are n birthdays and that there are k people in the room. An **elementary event** is a tuple $(b_1, \ldots, b_k) \in \{1, 2, \ldots, n\}^k$.

The birthday of the ith person is b_i, $1 \le i \le k$, so we have n^k elementary events.

We assume that those elementary events are equally probable. Then the probability of an elementary event is $1/n^k$.

Birthday Paradox

Let p be the probability that two people in the room have the same birthday. Then with probability $q = 1 - p$, any two people have different birthdays.

Let E be the set of all vectors $(g_1, \ldots, g_k) \in \{1, 2, \ldots, n\}^k$, whose entries are *pairwise different*. Then E models the *Birthday paradox*.

Let $|E|$ denote the number of element in E. Then

$$|E| = (n - 0)\,(n - 1)\,\ldots\,(n - (k - 1)) = \prod_{i=0}^{k-1}(n-i) \quad \text{and}$$

$$q = (n - 0)\,(n - 1)\,\ldots\,(n - (k - 1))\,/\,n^k$$

$$= \frac{1}{n^k}\prod_{i=0}^{k-1}(n-i) \;=\; \prod_{i=1}^{k-1}(1-\frac{i}{n})$$

Since $1 + x \le e^x$ holds for all real numbers, therefore $(e^{-x} = 1\text{-}x+x^2/2!\text{-}x^3/3!+\ldots)$

$$q \le \prod_{i=1}^{k-1} e^{-i/n} \;=\; e^{-\sum_{i=1}^{k-1} i/n} = e^{-k(k-1)/2n}.$$

Birthday Paradox

Assume that $q = 0.5$, then we have

$$q = 0.5 \le e^{-k(k-1)/2n} \Rightarrow \ln(2^{-1}) = \ln e^{-k(k-1)/2n}$$

$$\Rightarrow \ln 2 = k(k-1)/2n$$
$$\Rightarrow k(k-1) = 2n \ln 2 \quad (\ln 2 = 0.693)$$
$$\Rightarrow k^2 - k - 2n \ln 2 = 0$$
$$\Rightarrow k = [1 + (1 + 8n \ln 2)^{0.5})] / 2$$

If $k \ge (1 + (1 + 8n \ln 2)^{0.5}) / 2$, then $q \ge 0.5$.

Birthday Paradox

We describe a simple attack on hash functions $\mathbf{H: \Omega^* \rightarrow \Omega^n}$ (hence the result consists of $|\Omega|^n$ possible strings) called the **birthday attack**.

It attacks the ***strong collision resistance*** of H. The attack is based on the birthday paradox.

Assume that Ω is an alphabet. Then strings from Ω^* can be chosen such that the distribution on the corresponding hash values is the uniform distribution.

If k strings in $x \in \Omega^*$ are chosen, where

$k \geq (1 + (1 + 8\ |\Omega|^n \ln 2)^{0.5}) / 2$, then the probability of two hash values being equal exceeds 0.5.

Birthday Paradox

Assume $\Omega = \{0,1\}$, then $k \geq (1 + (1 + 8 \cdot 2^n \cdot \ln 2)^{0.5}) / 2$ is sufficient to **find two strings having the same hash value with probability greater than 0.5**.

By the table below, if we compute a little more than $2^{n/2}$ hash values, then the birthday attack will succeed with probability > 0.5.

Today, ***n* ≥ 128 or even *n* ≥ 160** is required to prevent the birthday attack.

n (bit length of the hash values)	50	100	150	200
$\log_2 k \approx n/2 \quad (k \approx 2^{n/2})$	25.24	50.24	75.24	100.24

Topic 2 Hash Function

Hash Function Construction

Instructor: Dr. Jianbing Ni

Fall 2021

Hash Function Construction

To design collision resistant (C.R.) hash functions

Step 1: construct C.R, function $h: T \times X \longrightarrow T$

Step 2: given C.R. function h for **short** messages,

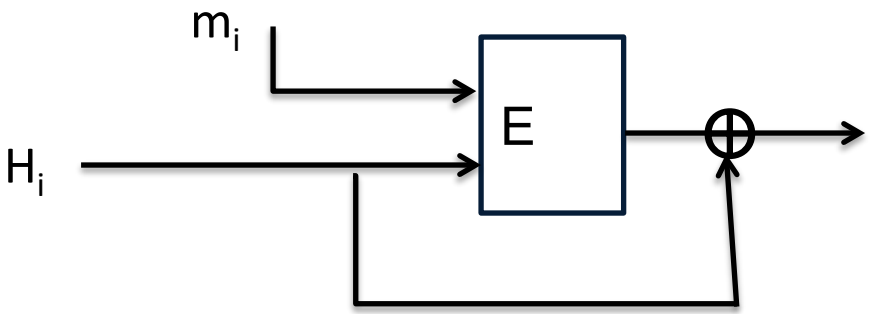
construct C.R. function H for **long** messages

Hash Function from a Block Cipher

Step 1: construct compression function **$h: T \times X \longrightarrow T$**

$E: K \times \{0,1\}^n \longrightarrow \{0,1\}^n$ a block cipher.

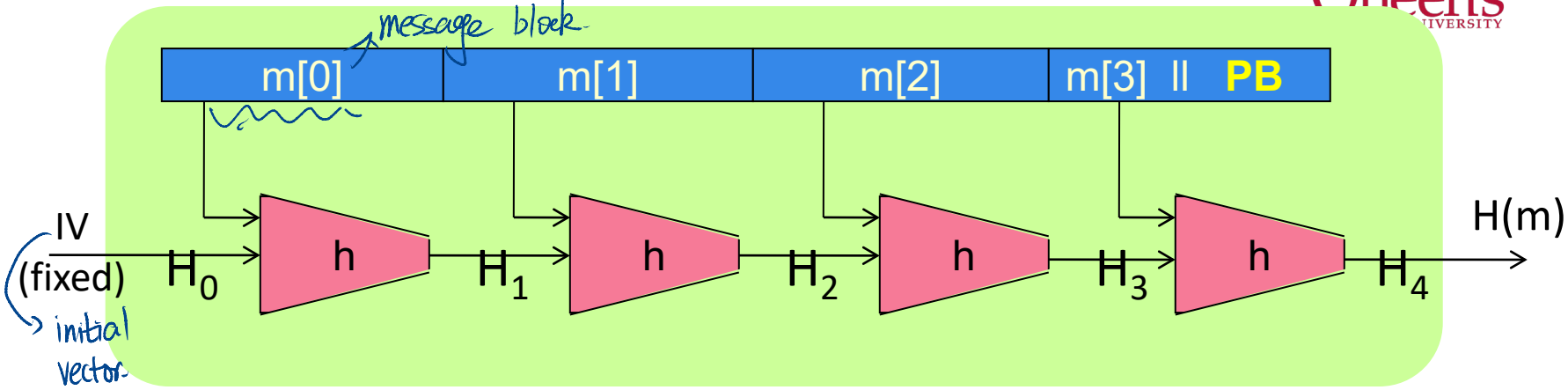
The **Davies-Meyer** compression function**: $h(H, m) = E(m, H) \oplus H$**



Thm: Suppose E is an ideal cipher

Finding a collision *h(H,m)=h(H',m')* takes $O(2^{n/2})$ evaluations of (E,D).

Merkle-Damgård Iterated Construction



Given $h: T \times X \longrightarrow T$ (compression function)

we obtain $H: X^{\leq L} \longrightarrow T$ H_i - chaining variables

PB: padding block

1000…0 || msg len

64 bits

If no space for PB add another block?

MD Collision Resistance

Thm: If h is collision resistant then so is H.

Proof: collision on H ⇒ collision on h

Suppose $M \neq M'$, $H(M) = H(M')$. We build a collision for h.

$IV = H_0, \quad H_1, \ \ldots, \ H_t, \quad H_{t+1} = H(M)$

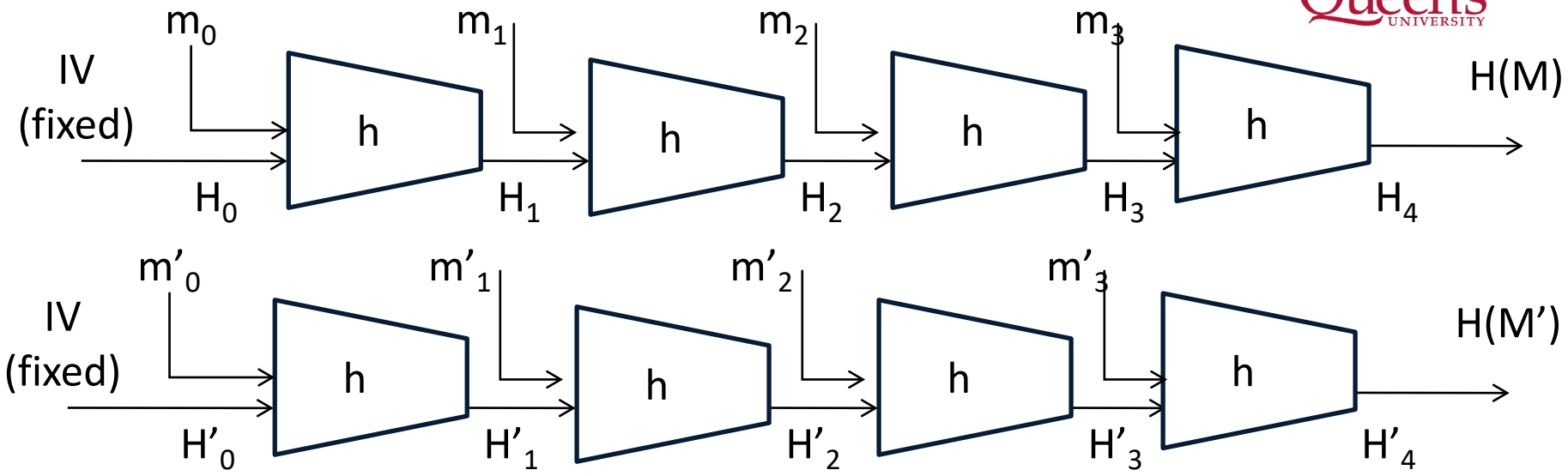
$IV = H_0', \quad H_1', \ \ldots, \ H'_r, \quad H'_{r+1} = H(M')$

$r = t$?

$h(H_t, m_t \,\|\, PB) = H_{t+1} = H'_{r+1} = h(H'_r, m'_r \,\|\, PB')$

$r \stackrel{?}{=} t$ $\quad r \neq t \Rightarrow PB \neq PB'$ ✓

Collision on H ⇒ Collision on h



|M| = |M'|, M≠M', $H_0 = H'_0$ = IV, Suppose H(M) = H(M') . We build a collision on h.

$r=t \Rightarrow |M|=|M'|$, $M \neq M'$, $H(M)=H(M') \Rightarrow$

$h(H_t, m_t \| PB) = h(H'_t, m'_t \| PB) \Rightarrow H_t \overset{?}{\neq} H'_t$, or $m_t \overset{?}{\neq} m'_t \Rightarrow$ Collision on h

Collision on H ⇒ Collision on h

Case 1: $m_i \neq m'_i$ *or* $H_i \neq H'_i$. But since $h(m_i, H_i) = h(m'_i, H'_i)$, there is a collision in h and we are done.

Case 2: $m_i = m'_i$ *and* $H_i = H'_i$ for all i. But then M = M', violating our assumption.

$M = M'$ ×

$m_t = m'_t$. $H_t = H'_t \Rightarrow h(H_{t-1}, m_{t-1}) = h(H'_{t-1}, m'_{t-1})$

$H_{t-1} \stackrel{?}{\neq} H'_{t-1}$ or $m_{t-1} \stackrel{?}{\neq} m'_{t-1} \Rightarrow$ Collision on h.

$m_{t-1} = m'_{t-1}$. and $H_{t-1} = H'_{t-1} \Rightarrow h(H_{t-2}, m_{t-2}) = h(H'_{t-2}, m'_{t-2})$

$H_{t-2} \stackrel{?}{\neq} H'_{t-2}$ or $m_{t-2} \stackrel{?}{\neq} m'_{t-2} \Rightarrow$ Collision on h.

$m_{t-2} = m'_{t-2}$, $H_{t-2} = H'_{t-2} \Rightarrow h(H_{t-3}, m_{t-3}) = h(H'_{t-3}, m'_{t-3})$

$\vdots$

$H_1 = H'_1$, $m_1 = m'_1 \Rightarrow h(H_0, m_0) = h(H'_0, m'_0) \Rightarrow m_0 \stackrel{?}{\neq} m'_0 \Rightarrow$ Collision on h

$m_0 = m'_0$

MD5

Rivest, 1991

Based on Merkle–Damgård construction.

A one-way function: the Davies–Meyer structure

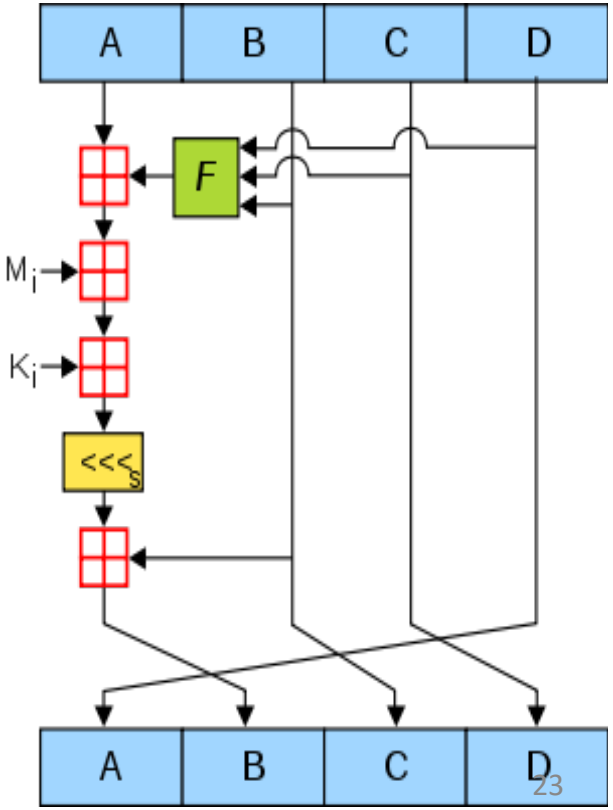
Very popular until 2010.

- 2004: First collision attacks
- 2005: Practical collision attack; SSL cert. with same MD5 hash.
- ~2010: Forged Microsoft MD5 certificates used in Flame malware

Preimage resistance: Mostly ok.

Input: arbitrary message size
Block: 512 bits
Output: 128 bits.

64 rounds of Compression:



SHA-1

Designed by NSA; based on Rivest's MD4 & MD5 designs

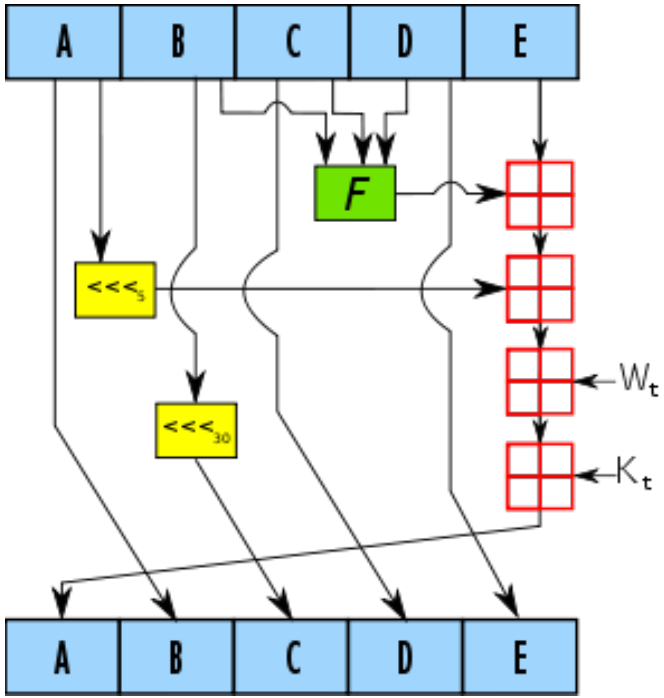
SHA 1993; SHA-1 1995

160-bit output size

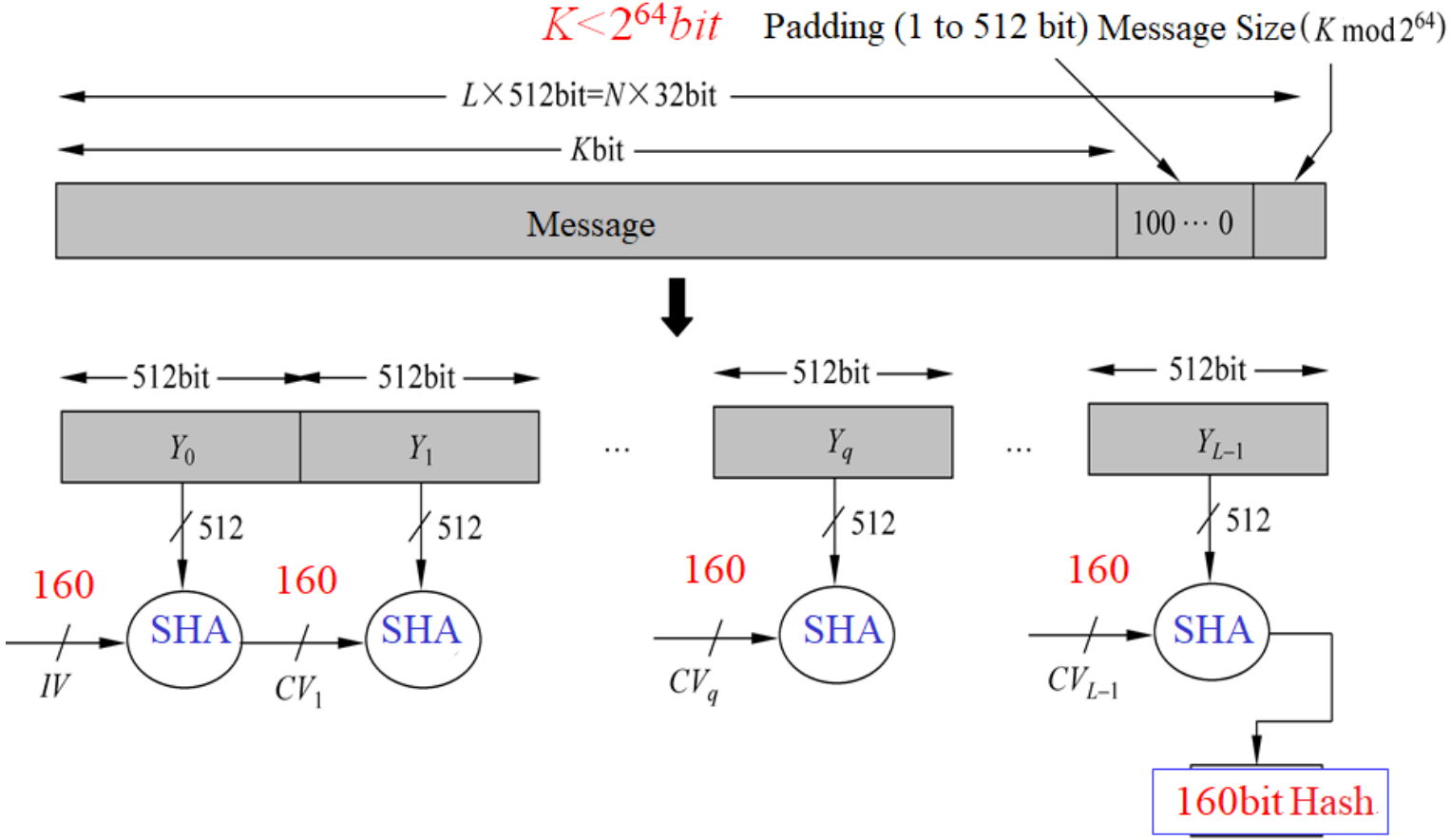
Input: Any message less than 2^{64} bits, divided into 512-bit blocks.

Output: 160-bit message digest.

80 rounds of:



SHA-1 Algorithm



1. Message Padding

Bits Padding: to pad the bits so that the message is 448 bits modulo 512, i.e., the message size after padding is a multiple of 512 minus 64, and the remaining 64 bits are used in step 2.

Step 1 is necessary, even if the message size has been met.
For example, if the message length is 448 bits, the padding size is 512 bits, and the full size becomes 960 bits.
The number of padding bits is larger than or equal to 1 and less than or equal to 512.
The padding method is: the first bit is 1, and the others are 0.

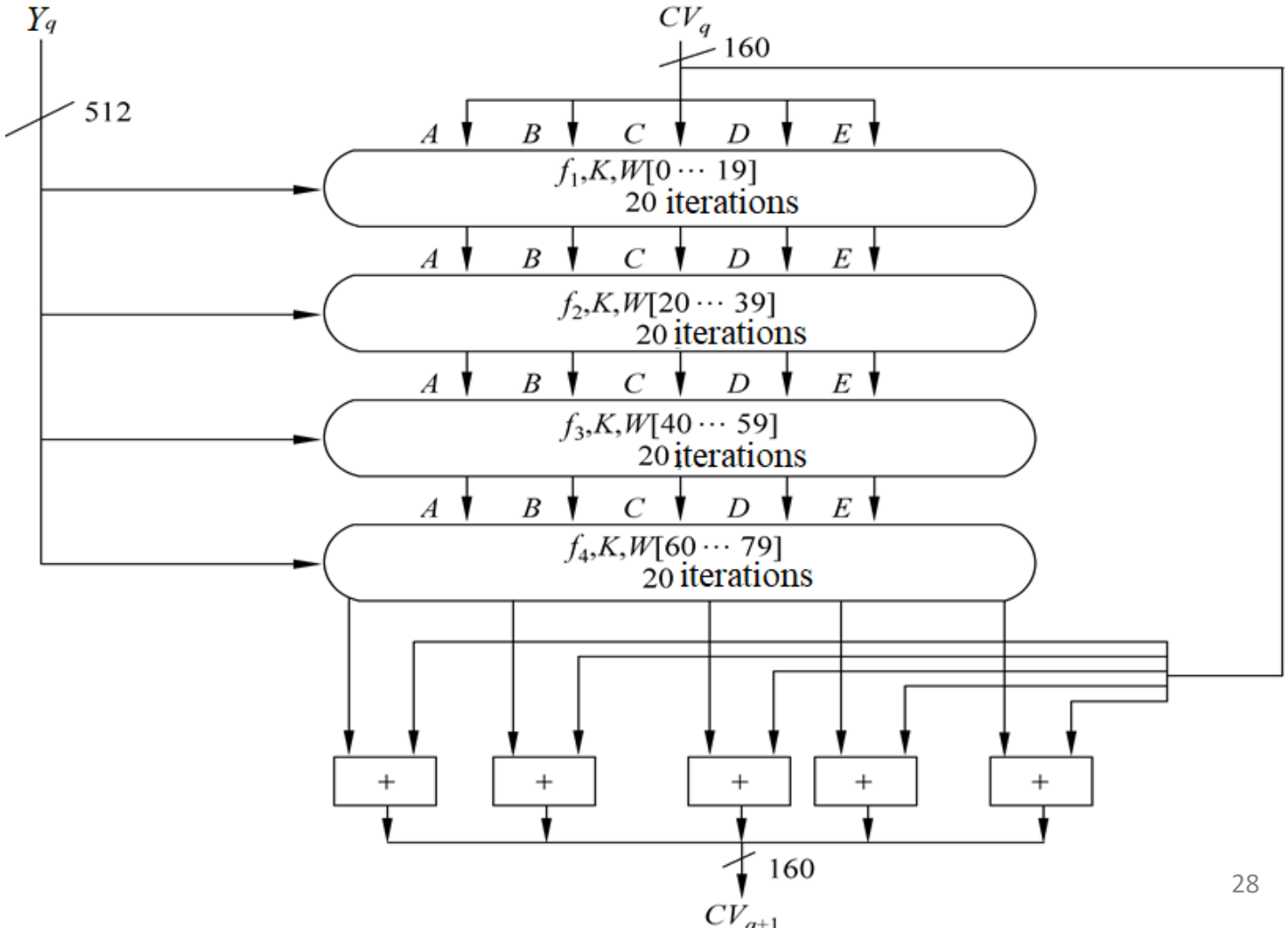
2. Append Length

Append Length: the left 64 bits is used to denote the message size before padding in a big-endian manner.

Big-endian stores the most significant byte in the low address byte.
The opposite storage method is called the little-endian method.

- The message size is a multiple of 512 (set to L times)
- The message is divided into Y_0, Y_1,..., Y_{L-1}
- Each block is 16×32-bit words
- The total number of words in the message is N=L×16
- The message is represented in words as M[0], M[1],...,M[N-1].

SHA


Y_q
512
CV_q
160
A B C D E
$f_1, K, W[0 \cdots 19]$
20 iterations
A B C D E
$f_2, K, W[20 \cdots 39]$
20 iterations
A B C D E
$f_3, K, W[40 \cdots 59]$
20 iterations
A B C D E
$f_4, K, W[60 \cdots 79]$
20 iterations
+
+
+
+
+
160
CV_{q+1}

3. Buffer Initialization

Use a 160-bit buffer to store intermediate results and output hash values.

The buffer can be represented as five 32-bit long registers (A, B, C, D, E), each storing data in big-endian mode.

The initial values are A=67452301, B=EFCDAB89, C=98BADCFB, D=10325476, E=C3D2E1F0.

4. 80 Rounds

The output of the fourth round is added to the input CV_q of the first round to produce CV_{q+1}, where the addition modulo 2^{32} is to add the word in each register and each word of the CV_q.

After the L blocks are compressed, the output of the last block is the 160-bit message digest.

The processing from step 3 to step 5 can be denoted as follows:

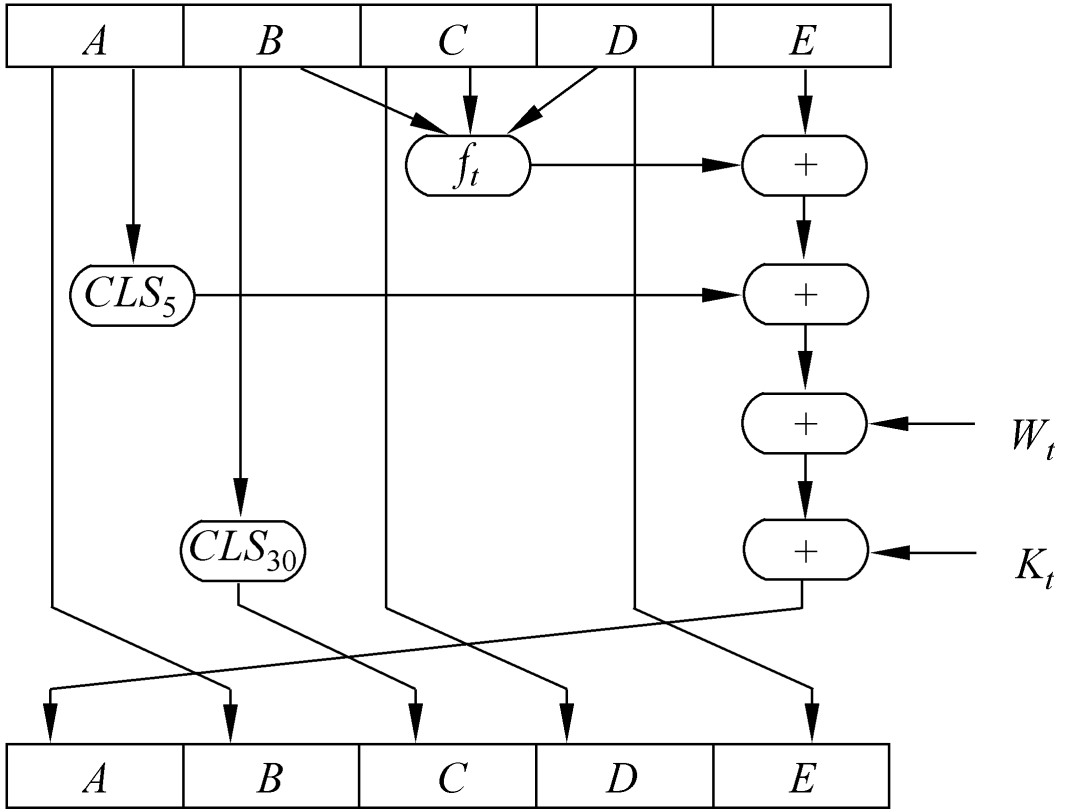
CV_0=IV;

CV_{q+1}=SUM_{32}(CV_q, $ABCDE_q$);

MD=CV_L

- IV: Initial value of ABCDE.
- $ABCDE_q$: output of the q^{th} block after the last round of processing
- L: the number of block for the message (including padding and length fields)
- SUM32: is the addition modulo 2^{32} of the corresponding word
- MD: The final digest value.

One Iteration



$$A, B, C, D, E \leftarrow (E + f_t(B, C, D) + CLS_5(A) + W_t + K_t),\ A,\ CLS_{30}(B),\ C,\ D$$

One Iteration

The compression function of SHA consists of 4 rounds of processing. Each round of processing consists of 20 iterations. The form of each iteration is

A, B, C, D, E: 5 words of the buffer
t: number of iterations ($0 \le t \le 79$)
$f_t(B,C,D)$: basic logic functions used in the t-th iteration
CLS_s: left loop shift *s* bit
W_t: a 32-bit word derived from the current 512-bit block
K_t: constant for addition
+: addition modulo 2^{32}.

Basic Logic function f

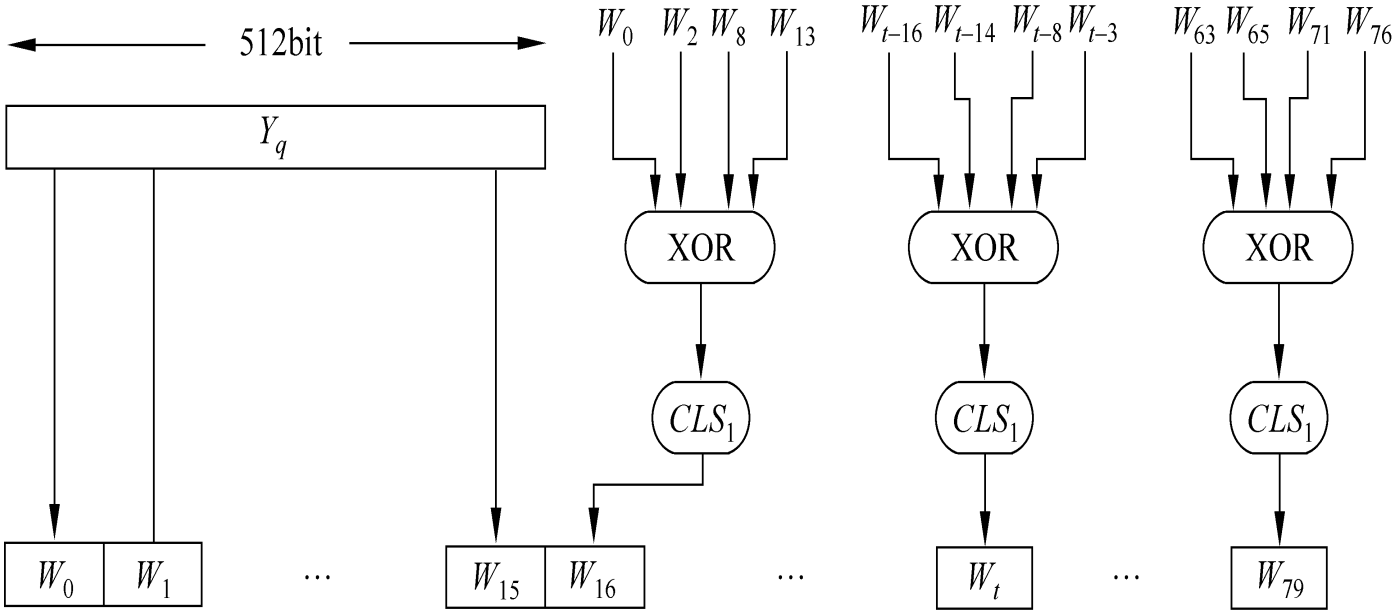
Input: 3×32-bit words
Output: 1×32-bit word

Steps	Function	Operation
$0 \le t \le 19$	$f_1 = f_t(B,C,D)$	$(B \wedge C) \vee (\overline{B} \wedge D)$
$20 \le t \le 39$	$f_2 = f_t(B,C,D)$	$B \oplus C \oplus D$
$40 \le t \le 59$	$f_3 = f_t(B,C,D)$	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$
$60 \le t \le 79$	$f_4 = f_t(B,C,D)$	$B \oplus C \oplus D$

Wt Generation

Generate $W_0, W_1, \ldots, W_{15}$, $W_{16}, W_{17}, \ldots, W_{79}$ from the current 512-bit block

$$W_t = CLS_1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3})$$



Constants for Addition

K_t = 0x5A827999 if $0 \leq t \leq 19$

K_t = 0x6ED9EBA1 if $20 \leq t \leq 39$

K_t= 0x8F1BBCDC if $40 \leq t \leq 59$

K_t= 0xCA62C1D6 if $60 \leq t \leq 79$

Security of SHA-1

- 2004, Rijmen published an attack which finds collisions with a computational effort of fewer than 2^{80} operations
- 2004, Xiaoyun Wang reduce the complexity for finding a collision in SHA-1 to 2^{63}
- 2008, Stéphane Manuel reported hash collisions with an estimated theoretical complexity of 2^{51} to 2^{57}
- 2015, Marc Stevens published a freestart collision attack on SHA-1's compression function that requires only 2^{57} SHA-1 evaluations
- February 23, 2017, CWI and Google announced the *SHAttered* attack, in which they generated two different PDF files with the same SHA-1 hash in roughly $2^{63.1}$ SHA-1 evaluations

SHA-2

A set of cryptographic hash functions designed by NSA, first published in 2001

Based on Merkle–Damgård construction

A one-way compression function: the Davies–Meyer structure

SHA-2 family consists of six hash functions with digests that are 224, 256, 384 or 512 bits:

SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.

- SHA-256 computed with 32-bit words, 512-bit block size, and 64 rounds
- SHA-512 computed with 64-bit words, 1024-bit block size, and 80 rounds
- SHA-224 and SHA-384 are truncated versions of SHA-256 and SHA-512 respectively, computed with different initial values.
- SHA-512/224 and SHA-512/256 are also truncated versions of SHA-512.

SHA-3

Public competition by NIST, similar to AES:

NIST's request for proposals (2007)

64 submissions (2008)

14 semi-finalists (2009)

5 finalists (2010)

Winner: Keccak (2012)

Designed by Bertoni, Daemen, Peeters, Van Assche.

"Sponge Construction", a completely different structure.

Summary

- Hash Function
- Collision Resistance
- Birthday Attack
- Hash Function Construction
- MD5, SHA1, SHA2, SHA3

Practice Question

We consider a simplified version of the Merkle-Damgård construction. Suppose

$$\textbf{compress}: \{0,1\}^{m+t} \rightarrow \{0,1\}^{m},$$

Where $t \geq 1$, and suppose that $x = x_1||x_2||\cdots||x_k$, where $|x_1|=|x_2|=\cdots=|x_k|=t$.

We study the following iterated hash function:

Algorithm: SIMPLIFIED MERKLE-DAMGARD (x, k, t)
external compress
$z_1 \leftarrow 0^m||x_1$
$g_1 \leftarrow$ **compress** (z_1)
for $i \leftarrow 1$ to k-1
    do $z_{i+1} = g_i \,||\, x_{i+1}$
        $g_{i+1} \leftarrow$ **compress**(z_{i+1})
$h(x) \leftarrow g_k$
return ($h(x)$)

Practice Question

Suppose that **compress** is collision resistant, and suppose further that **compress** is **zero-preimage resistant**, which means that it is hard to find $z \in \{0,1\}^{m+t}$ such that **compress**$(z)=0^m$. Under these assumptions, prove that h is collision resistant.

Thanks

ELEC 473 Cryptography and Network Security

7. Data Integrity Algorithms

Instructor: Dr. Jianbing Ni

Fall 2021

Review of C7 Topic 2

Hash Function

- maps arbitrarily long strings to strings of fixed length.
- Input: the data of arbitrary size
- Output: A fixed-size value (hash value, message digest, or hash)

Properties of hash function

- Preimage resistance: It is infeasible to find $m \in M$ such that $H(m)=y$.
- Second Preimage resistance: It is infeasible to find $m' \in M$ such that $m' \neq m$ and $H(m')=H(m)$,.
- Collision resistance: It is infeasible to find a pair $m_0, m_1 \in M$ such that $m_0 \neq m_1$ and $H(m_0)=H(m_1)$.

Birthday Attack and Birthday Paradox

- If the length of hash value is n bits, birthday attack can compute more than $2^{n/2}$ values to find a collision for the hash function with the probability greater than ½.

Hash Function Construction

- Step 1: construct C.R function $h: T \times X \longrightarrow T$ -- The Davies-Meyer compression function
- Step 2: given C.R. function h for short messages, to construct C.R. function H for long messages -- Merkle-Damgård Iterated Construction

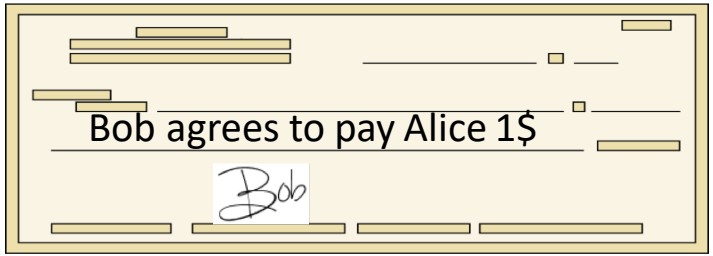
MD5, SHA1, SHA2, SHA3

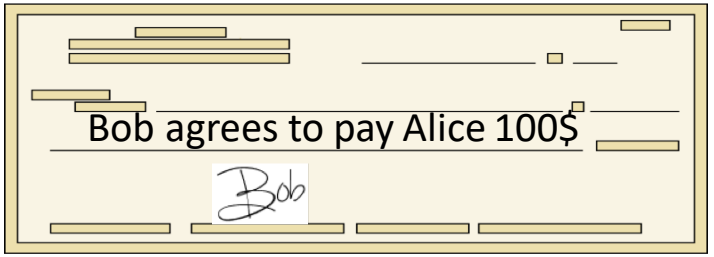
Outline

- Message Authentication Code
- Hash Function
- Digital Signature

Physical Signature

Goal: bind document to author



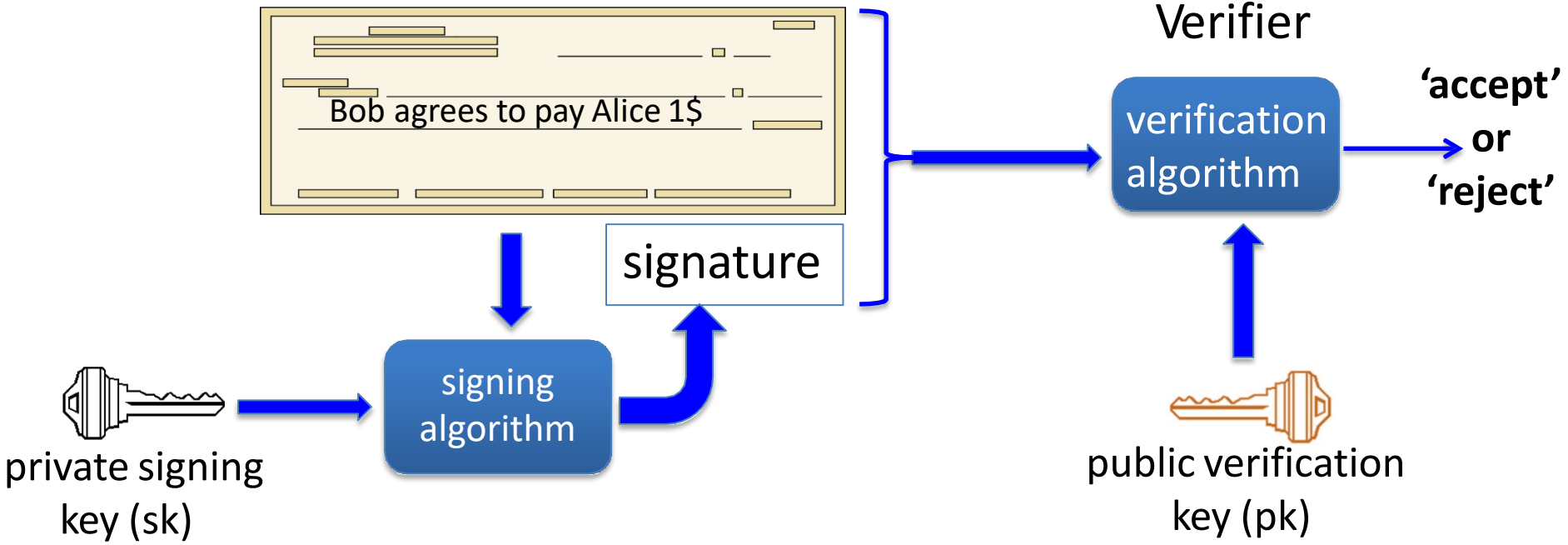


Problem in the digital world:

anyone can copy Bob's signature from one doc to another

Digital Signature

Solution: make signature depend on document signer



Digital Signature

- The electronic analog of handwritten signature.
  - Message Integrity: The message has not been modified.
  - Sender Authenticity: The sender is truly Bob.
  - Non-repudiation: The sender cannot deny his signature.

- Properties:
  - Easy to generate.
  - Easy to verify by a verifier.
  - Hard to forge.

- Represented as a **string of bits** attached to a message.

Digital Signatures: Syntax

A signature scheme (KeyGen, Sign, Verif) is a triple of algorithms:

- KeyGen(λ): randomized alg. outputs a key pair (pk, sk)
- Sign(sk, m∈M): outputs sig. σ
- Verif(pk, m, σ): outputs 'accept' or 'reject'

- Consistency: for all (pk, sk) output by KeyGen :

$$\forall m \in M: \quad \text{Verif}(pk, m, \text{Sign}(sk, m)) = \text{'accept'}$$

Requirements

- **Sign** and **Verif** can be publically known; only its private signing key is secret.
- **Verif(pk,m,σ) = "accpet"** if and only if **σ =Sign(sk,m)**.
  - Without the private signing key, it is computationally infeasible to construct a valid pair **(m, σ)** such that **Verif(pk,m,σ) = "accept"**.
  - **If** (m, **σ, Verif(pk,m,σ)**) is known, it is computationally infeasible to find m' such that **Verif(pk,m',σ)**=**Verif(pk,m,σ)**.
- It is possible to have two messages with the same digital signature. That is, there exists **m** and **m'** such that, **Sign(sk,m) = Sign(sk,m')**.
- It is computationally infeasible to find such an **m'** if **(m, σ, Verif(pk,m,σ))** is known.

Adversarial Goals

Total Break:

- Eve is able to determine Alice's private signing key and generate a valid signature on any message

Selective Forgery:

- Eve is able to create a valid signature on a given message m, where *m* has been *chosen* by Eve prior to the attack.

Existential Forgery:

- Eve can create a valid signature for at least one message. In other words, Eve is able to create a message/signature pair (m,σ), where m is a message and V(pk,m,σ)=accept.

Possible Attacks

Queen's UNIVERSITY

Key-Only Attack:

- The attacker is only given the public verification key.

Known-Message Attack:

- The attacker is given valid signatures for a variety of messages known by the attacker but not chosen by the attacker.

Chosen-Message Attack: more powerful

- The attacker first learns signatures on arbitrary messages of the attacker's choice.

Secure Digital Signature

Attacker's power: **chosen message attack**

- for $m_1, m_2, \ldots, m_q$ attacker is given $\sigma_i = \text{Sign}(sk, m_i)$

Attacker's goal: **existential forgery** (可伪造)

- produce some **new** valid message/sig pair (m, σ).

$$m \notin \{ m_1, \ldots, m_q \}$$

$\Rightarrow$ attacker cannot produce a valid sig. for a new message

RSA Signature

Developed in 1978 by Rivest, Shamir, and Adleman (RSA)
Security is based on the hard problem of "integer factorization"

KeyGen(λ): ⇒ if $\lambda = 512$, $|p| = |q| = 512$ bits.

1. Generate two large random distinct primes p and q, each roughly the same size,
2. Compute $n = pq$ and $\varphi(n) = (p-1)(q-1)$ (与n互质的数目)
3. Select random integer e: $1 < e < \varphi(n)$, such that $\gcd(e, \varphi(n)) = 1$
4. Compute unique integer d: $1 < d < \varphi(n)$, such that $ed = 1, \bmod \varphi(n)$
5. Public verification key is *(n, e)* and the private signing key is *d*.

RSA Signature - Sign and Verify

Sign(sk,m): To sign a message $m \in M$, Alice should:

- Compute: $\sigma = H(m)^d \mod n$
- Alice's signature for m is σ

Verif(pk, m, σ): To verify Alice's signature and recover m, Bob should:

- Obtain Alice's public verification key (n, e)
- Compute: $h = \sigma^e \mod n$
- Verify that $H(m) = h$; if not, reject the signature

$h = \sigma^e = [H(m)^d]^e = H(m)^{de} = H(m)^{1+k\varphi(n)} = H(m) \cdot H(m)^{\varphi(n) \cdot k}$

$= H(m) \cdot 1 = H(m) \mod n$

Correctness of RSA

- Euler's theorem: $a^{\varphi(n)} \equiv 1 \mod n, gcd(a, n) = 1$
- If σ is a signature for m, then: $\sigma = H(m)^d \mod n$
- Since $n = pq, ed \equiv 1 \mod \varphi(n)$, then:

$$\begin{aligned} \sigma^e &\equiv H(m)^{ed} \equiv H(m)^{1 \mod \varphi(n)} \equiv H(m)^{\varphi(n)k+1} \\ &\equiv H(m)^{\varphi(n)k} \cdot H(m) \equiv H(m) \mod n \end{aligned}$$

RSA Signature Example

Alice

- p=5 q=7 n = 35 $\varphi(n)$ = 4·6=24
- e = 5;
- ed = 5d=1 mod 24 => d = 5
  - Public verification key: (n=35, e=5) Private signing key: d=5
- $M \in [0, n\text{-}1]$
- For all $m \in M$, H(m) = 26, $\sigma = 26^5 \bmod 35 = 31 \bmod 35$

Bob:

- h=$31^5 \bmod 35$ = H(m)=26 $\in [0, n\text{-}1]$

Digital Signature Standard (DSS) and DSA

- designed by NIST & NSA in early 90's
- published as FIPS-186 in 1991
- DSS is the standard, DSA is the digital signature algorithm
- DSA creates a 320 bit signature and has 512-1024 bit security
- DSA has smaller signature size and faster computation than RSA
- DSA's security depends on difficulty of discrete logarithm problem

DSA Key Generation

- **Setup**: have shared global public key values (p, q, g):
  - choose 160-bit prime number q
  - choose a large prime p with $2^{L-1} < p < 2^L$
    - where L= 512 to 1024 bits and is a multiple of 64
    - such that q is a 160 bit prime divisor of (p-1)
  - choose $g = h^{(p-1)/q}$
    - where $1<h<p-1$ and $h^{(p-1)/q} \bmod p > 1$
- **KeyGen**: A user, Alice, chooses the secret signing key and compute the public verification key:
  - choose a random private signing key: $x < q$
  - compute the public verification key: $y = g^x \bmod p$

DSA Signature Generation

Sign: to sign a message m, Alice:

- generates a random value k, $k < q$, k must be random, be destroyed after use, and never be reused
- computes the signature pair:

  $r = (g^k \bmod p) \bmod q$

  $s = [k^{-1}(H(m) + x \cdot r)] \bmod q$
- sends the signature $\sigma=(r, s)$ with message m

DSA Signature Verification

Verify: Having received m and the signature $\sigma=(r, s)$, to verify a signature, the user Bob

- computes:

  $w = s^{-1} \bmod q$

  $u_1 = (H(m) \cdot w) \bmod q$

  $u_2 = (r \cdot w) \bmod q$

  $v = ((g^{u_1} y^{u_2}) \bmod p) \bmod q$

- if $v=r$, then signature is valid.

Summary

- Digital Signatures
- Security of Digital Signatures
- RSA, DSS

Thanks