Department of Electronics

Cryptography

Fall 2019

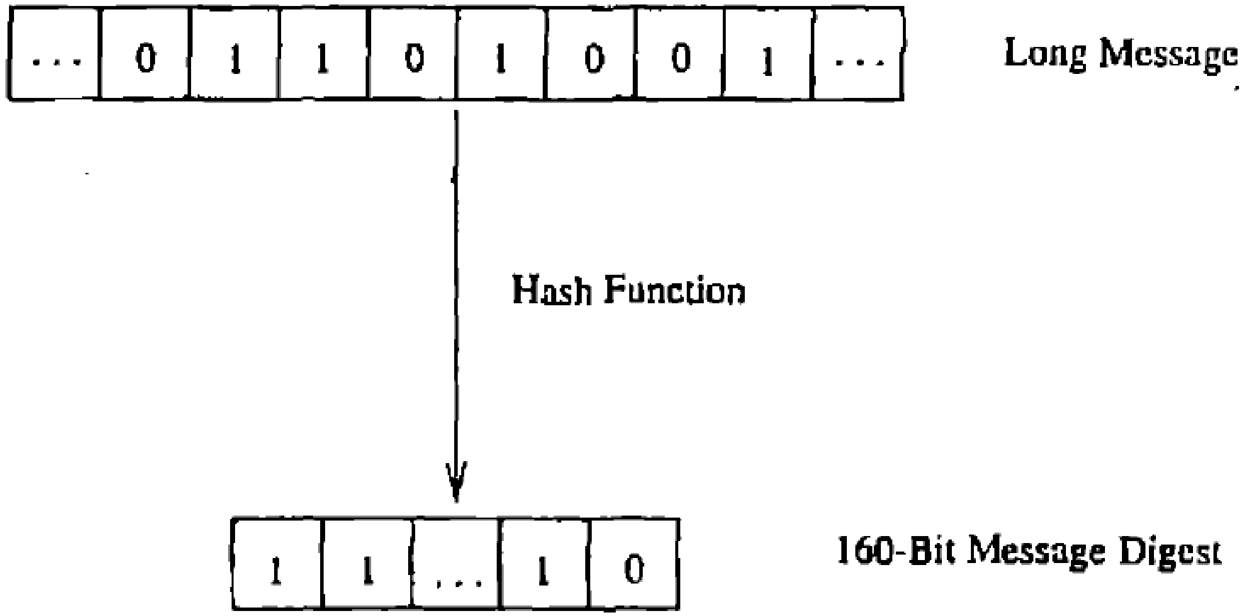
Hasan Mahmood

hasan@qau.edu.pk

Week 8 (23 & 24 October 2019)

The Hash Functions

- Non-invertibility properties
- A cryptographic hash function h takes as input a message of arbitrary length and produces as output a message digest of fixed length
- For example, 160 bits



Hash function properties

1. Given a message m, the message digest $h(m)$ can be calculated very quickly.

2. Given a y, it is computationally infeasible to find an m' with $h(m') = y$ (in other words, h is a **one-way**, or **preimage resistant**, function). Note that if y is the message digest of some message, we are not trying to find this message. We are only looking for some m' with $h(m') = y$.

3. It is computationally infeasible to find messages m_1 and m_2 with $h(m_1) = h(m_2)$ (in this case, the function h is said to be **strongly collision-free**).

Hash function

- Collision free (weakly)
- Preimage resistance
- Requirement 3 is the hardest one to satisfy
- In 2004, Wang, Feng, Lai, and Yu fond many examples of collisions for the popular hash functions MD4, MD5, HAVAL-128 and RIPEMD
- This means that a valid digital signature on one certificate is also valid for the other certificate.
- SHA-1 collision can be determined with around 2^{69} calculations

Hash Example

- Efficient in computational requirements
- Start with a message *m* of arbitrary length *L*
- Break the message *m* into *b*-bit blocks, where $n << L$
- *Denote* these n-bit blocks by m_j, $m = [m_1, m_2, \cdots, m_l]$
- The length, l=ceil[L/n], last block is padded with zeroes
- We write the *j*th block m_j as row vector

$$m_j \;\; \doteq \;\; [m_{j1}, m_{j2}, m_{j3}, \cdots, m_{jn}]$$

Example

- Stack these row vectors to form an array

$$h_i = m_{1i} \oplus m_{2i} \oplus \cdots \oplus m_{li}.$$

$$\begin{array}{c}
\begin{bmatrix}
m_{11} & m_{12} & \cdots & m_{1n} \\
m_{21} & m_{22} & \cdots & m_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
m_{l1} & m_{l2} & \cdots & m_{ln}
\end{bmatrix} \\
\begin{array}{cccc}
\Downarrow & \Downarrow & \Downarrow & \Downarrow \\
\oplus & \oplus & \oplus & \oplus \\
\Downarrow & \Downarrow & \Downarrow & \Downarrow
\end{array} \\
\begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} = h(m).
\end{array}$$

Hash example

- Input is arbitrary length message
- Output is *n*-bit message digest
- It is not considered cryptographically secure
- Practical cryptographic hash functions typically make use of several other bit level operations
- Need to avoid collision
- Bit rotation is used, similar to DES

Simple Hash with rotation operation

$$
\begin{bmatrix}
m_{11} & m_{12} & \cdots & m_{1n} \\
m_{22} & m_{23} & \cdots & m_{21} \\
m_{33} & m_{34} & \cdots & m_{32} \\
\vdots & \vdots & \ddots & \vdots \\
m_{ll} & m_{l,l+1} & \cdots & m_{l,l-1}
\end{bmatrix}
$$

$$
\begin{array}{cccc}
\Downarrow & \Downarrow & \Downarrow & \Downarrow \\
\oplus & \oplus & \oplus & \oplus \\
\Downarrow & \Downarrow & \Downarrow & \Downarrow
\end{array}
$$

$$
\begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} = h(m).
$$

The Secure Hash Algorithm (SHA)

- SHA-1 produces 160-bit hash
- The original message is broken into a set of fixed size blocks
- Last block is padded to fill out the block
- Message blocks are processed given sequence of rounds that use a compression function h’
- Current block is combined with the result of previous rounds
- In a good compression function, makes each input bit effect as many output bits as possible.

SHA-1

Take original message and pad it with a 1 bit followed by a sequence of 0 bits

Enough 0 bits are appended to make the mew message 64 bits shout of the next highest multiple of 512 bits in length

We append the 64 bit representation of the length T of the message

For example, if the original message has 2800 bits, we add a 1 and 207 0s to obtain a new message of length 3008=6x512-64

Since 2800=101011110000, we append 52 0s followed by this number

Message length is 3072, broken down into six blocks of length 512

SHA operations

1. $X \wedge Y$ = bitwise "and", which is bitwise multiplication mod 2, or bitwise minimum.

2. $X \vee Y$ = bitwise "or", which is bitwise maximum.

3. $X \oplus Y$ = bitwise addition mod 2.

4. $\neg X$ changes 1s to 0s and 0s to 1s .

5. $X + Y$ = addition of X and Y mod 2^{32}, where X and Y are regarded as integers mod 2^{32}.

6. $X \hookleftarrow r$ = shift of X to the left by r positions (and the beginning wraps around to the end).

SHA operations

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{if } \ 0 \le t \le 19 \\ B \oplus C \oplus D & \text{if } 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \le t \le 59 \\ B \oplus C \oplus D & \text{if } 60 \le t \le 79 \end{cases}$$

Define constants $K_0, \ldots, K_{79}$ as follows:

$$K_t = \begin{cases} \texttt{5A827999} & \text{if } \ 0 \le t \le 19 \\ \texttt{6ED9EBA1} & \text{if } 20 \le t \le 39 \\ \texttt{8F1BBCDC} & \text{if } 40 \le t \le 59 \\ \texttt{CA62C1D6} & \text{if } 60 \le t \le 79 \end{cases}$$

The SHA-1 Algorithm

$$
\begin{aligned}
H_0 &= 67452301 \\
H_1 &= EFCDAB89 \\
H_2 &= 98BADCFE \\
H_3 &= 10325476 \\
H_4 &= C3D2E1F0.
\end{aligned}
$$

The SHA-1 Algorithm

1. Start with a message m. Append bits, as specified in the text, to obtain a message y of the form $y = m_1 \| m_2 \| \cdots \| m_L$, where each m_i has 512 bits.
2. Initialize $H_0 = 67452301$, $H_1 = EFCDAB89$, $H_2 = 98BADCFE$, $H_3 = 10325476$, $H_4 = C3D2E1F0$.
3. For $i = 0$ to $L - 1$, do the following:

(a) Write $m_i = W_0 \| W_1 \| \cdots \| W_{15}$, where each W_j has 32 bits.

(b) For $t = 16$ to 79, let $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \hookleftarrow 1$

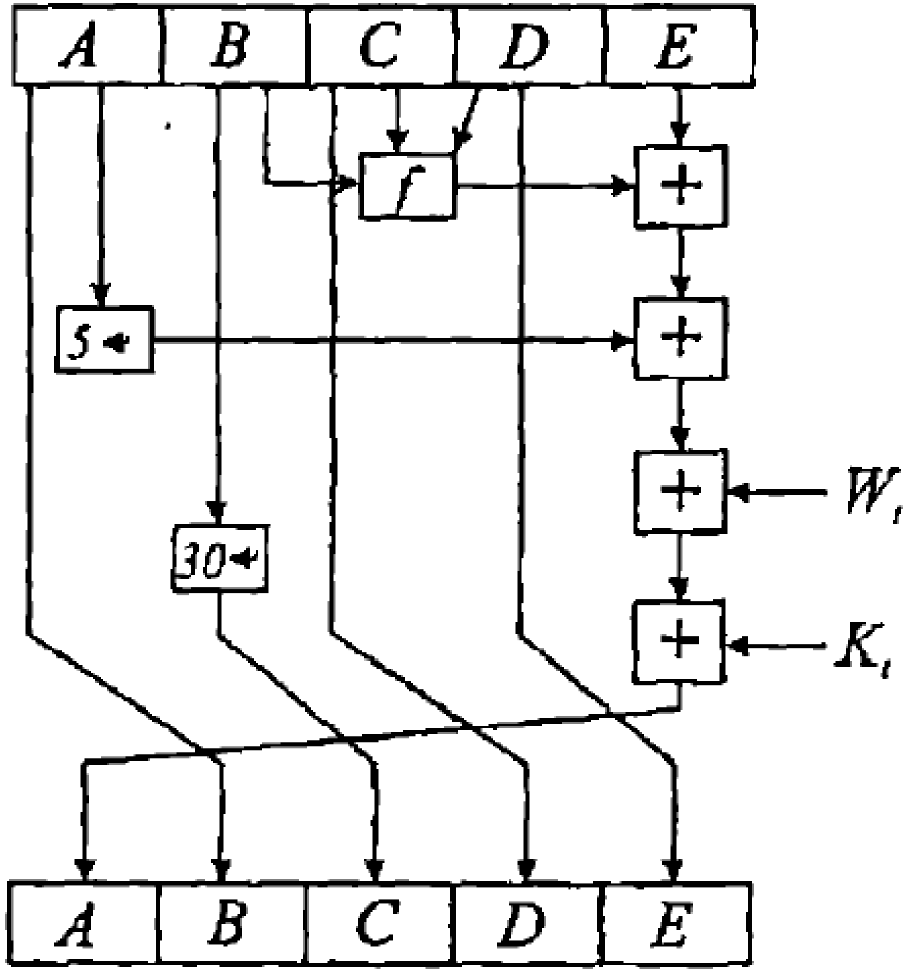
(c) Let $A = H_0$, $B = H_1$, $C = H_2$, $D = H_3$, $E = H_4$.

(d) For $t = 0$ to 79, do the following steps in succession: $T = (A \hookleftarrow 5) + f_t(B, C, D) + E + W_t + K_t$, $E = D$, $D = C$, $C = (B \hookleftarrow 30)$, $B = A$, $A = T$.

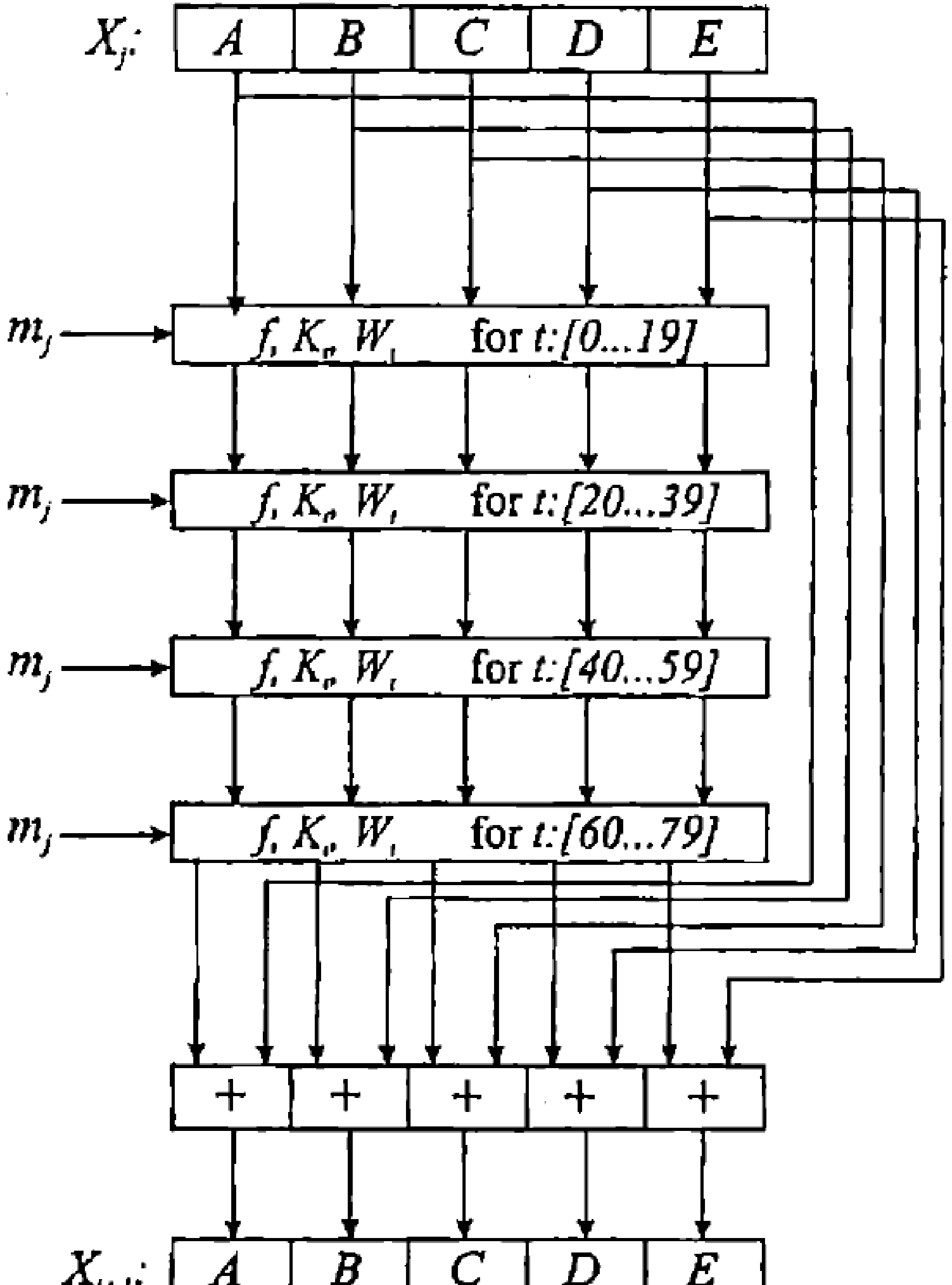
(e) Let $H_0 = H_0 + A$, $H_1 = H_1 + B$, $H_2 = H_2 + C$, $H_3 = H_3 + D$, $H_4 = H_4 + E$.

4. Output $H_0 \| H_1 \| H_2 \| H_3 \| H_4$. This is the 160-bit hash value.

SHA-1 Step 3



A B C D E
f
5 ←
30 ←
W_t
K_t
A B C D E



X_j: A B C D E
m_j
f, K_t, W_t for t:[0...19]
f, K_t, W_t for t:[20...39]
f, K_t, W_t for t:[40...59]
f, K_t, W_t for t:[60...79]
A B C D E

Discrete Logarithm

- Based on difficulty of factoring, discrete logarithm has similar benefits from complexity
- Fix a prime *p*, let α and β be nonzero integers mod *p*, suppose,

$$\beta \equiv \alpha^x \pmod{p}.$$

- The problem of finding *x* is called the discrete logarithm problem
- If *n* is the smallest positive integer such that

$$\alpha^n \equiv 1 \pmod{p}$$

- We may assume that, (discrete log of b with respect to a, $0 \leq x < n$)

$$x = L_\alpha(\beta)$$

Discrete Logarithms, example

- P=11, α=2
- Since 2^6=9 (mod 11)
- We have $L_2(9)$=6
- Also, $2^6=2^{16}=2^{20}$=9 (mod 11), so we consider taking any one of 6, 16, 26 as the discrete logarithm
- Smallest nonnegative value, namely 6
- α is taken to be a primitive root of mod *p, e*very β is a power of α (mod p)
- If α is not a primitive root, then discrete logarithm will not be defined for certain values of β

Discrete log properties

- For small *p*, it is easy to calculate discrete logs by exhaustive search, when *p* is large, this is not feasible
- Discrete logs are hard to compute in general, basis of several ciphers
- Size of the largest primes for which discrete logs can be computed is approximately the same size as the size of the largest integers that could be factored
- A function f(x) is called one-way function if f(x) is easy to compute, given y, it is computationally infeasible to find x with f(x)=y
- It is easy to compute $\alpha^x \pmod{p}$, but solving $\alpha^x = \beta$ for x is hard
- Multiplication of large primes can also be regarded as (probable) one-way function
- It is easy to multiply large primes but difficult to factor the result to recover the primes

Bit Commitment

- Alice claims that she has a method to predict the outcome of games
- She wans to sell her method to Bob
- Bob needs proof, for the game to be played coming weekend
- Alice does not want to disclose the result because Bob may bet
- Alice offers to prove her method by previous week games

Here's the setup. Alice wants to send a bit b, which is either 0 or 1, to Bob. There are two requirements.

1. Bob cannot determine the value of the bit without Alice's help.
2. Alice cannot change the bit once she sends it.

Bit Commitment, solution

- Alice puts the bit in a box, put her lock on, and send it to Bob
- When Bob wants the value of the bit, Alice removes the lock and Bob opens the box
- How do you implement this mathematically?
- (Alice and Bob do not have to be in the same room when the bit is revealed)

Bit Commitment, solution

- Alice and Bob agree on a large prime p=3 (mod 4) & primitive root α
- Alice chooses a random number x < p-1, whose 2nd bit x_1 is b
- She sends

$$\beta \equiv \alpha^x \pmod p$$

- We assume that Bob cannot compute discrete log of p
- Therefore, he cannot determine the value of b=x_1
- When Bob wants to know the value of b_1, Alice sends him the full value of x, and by looking at x mod 4, he find b
- Alice cannot send a value of x different than the one already used
- Bob checks, for unique solution at x < p-1, can be done for 100 bits for ex.

$$\beta \equiv \alpha^x \pmod p$$

Diffie-Hellman Key Exchange

- Establish keys for use in cryptographic protocols (DES or AES)
- Two parties are widely separated, communication over public channel
- Public key methods (RSA) is one of the solution
- Discrete log logarithms
- Alice and Bob establish a private key K

Diffie-Hellman Key Exchange

1. Either Alice or Bob selects a large, secure prime number p and a primitive root $\alpha \pmod{p}$. Both p and α can be made public.

2. Alice chooses a secret random x with $1 \leq x \leq p-2$, and Bob selects a secret random y with $1 \leq y \leq p-2$.

3. Alice sends $\alpha^x \pmod{p}$ to Bob, and Bob sends $\alpha^y \pmod{p}$ to Alice.

4. Using the messages that they each have received, they can each calculate the session key K. Alice calculates K by $K \equiv (\alpha^y)^x \pmod{p}$, and Bob calculates K by $K \equiv (\alpha^x)^y \pmod{p}$.

The ElGamal Public Key Cryptosystem

- Difficulty based on computing discrete logarithms
- Alice wants to send message m to Bob
- Bob chooses a large prime *p* and a primitive root α
- Assume m is an integer between 0 and *p*
- Bob chooses a secret integer a and computes: $\beta \equiv \alpha^a \pmod{p}$
- The information (p, α, β) is made public

Alice actions

1. Downloads (p, α, β)
2. Chooses a secret random integer k and computes $r \equiv \alpha^k \pmod{p}$
3. Computes $t \equiv \beta^k m \pmod{p}$
4. Sends the pair (r, t) to Bob

Bob decrypts by computing

$$tr^{-a} \equiv m \pmod{p}.$$

This works because

$$tr^{-a} \equiv \beta^k m(\alpha^k)^{-a} \equiv (\alpha^a)^k m\alpha^{-ak} \equiv m \pmod{p}.$$

Digital Signatures

- With the development of electronics commerce and electronics documents, the traditional methods of signature no longer suffice
- Electronics forgery, changing the digitized signatures
- We require digital signature not to be separated from the document
- Cannot be attached to other message(s), the signature is tied only to the signer and message
- Digital signature needs to be easily verified by other parties
- Two distinct steps: the signing process and the verification process
- We are not trying to encrypt the message!

RSA Signature

Bob has a document m that Alice agrees to sign. They do the following:

1. Alice generates two large primes p, q, and computes $n = pq$. She chooses e_A such that $1 < e_A < \phi(n)$ with $\gcd(e_A, \phi(n)) = 1$, and calculates d_A such that $e_A d_A \equiv 1 \pmod{\phi(n)}$. Alice publishes (e_A, n) and keeps private d_A, p, q.

2. Alice's signature is
$$y \equiv m^{d_A} \pmod{n}.$$

3. The pair (m, y) is then made public.

RSA Signature Verification

Bob can then verify that Alice really signed the message by doing the following:

1. Download Alice's (e_A, n).
2. Calculate $z \equiv y^{e_A} \pmod{n}$. If $z = m$, then Bob accepts the signature as valid; otherwise the signature is not valid.

RSA Variant

- Signing of document without knowing its contents
- Suppose Bob has made an important discovery
- He wants to record publicly what he has done

1. Alice chooses an RSA modulus n ($n = pq$, the product of two large primes), an encryption exponent e, and decryption exponent d. She makes n and e public while keeping p, q, d private. In fact, she can erase p, q, d from her computer's memory at the end of the signing procedure.

2. Bob chooses a random integer $k \pmod{n}$ with $\gcd(k, n) = 1$ and computes $t \equiv k^e m \pmod{n}$. He sends t to Alice.

3. Alice signs t by computing $s \equiv t^d \pmod{n}$. She returns s to Bob.

4. Bob computes $s/k \pmod{n}$. This is the signed message m^d.

$$s/k \equiv t^d/k \equiv k^{ed}m^d/k \equiv m^d \pmod{n},$$