ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

# Cryptography III

Public-key Cryptosystems,

Digital Signatures and Hash functions

# Weaknesses of symmetric cryptosystems

- Managing and distributing shared secret keys is so difficult in a model environment with too many parties and relationships
  - N parties ➔ n(n-1)/2 relationships ➔ each manages (n-1) keys
- No way for digital signatures
  - No non-repudiation service

# Diffie-Hellman new ideas for PKC

- In principle, a PK cryptosystem is designed for a single user, not for a pair of communicating users
  - More uses other than just encryption
- Proposed in Diffie and Hellman (1976) “New Directions in Cryptography”
  - public-key encryption schemes
  - public key distribution systems
    - Diffie-Hellman key agreement protocol
  - digital signature

# Diffie-Hellman's proposal

- Each user creates 2 keys: a secret (private) key and a public key → published for everyone to know

  - The PK is for encryption and the SK for decryption

    X = D(z, E(Z, X))

  - The SK is for creating signatures and the PK for verifying these signatures

    X = E(Z, D(z, X)) → D() for creating signatures, E → verifying

- Also, called asymmetric key cryptosystems
  - Knowing the public-key and the cipher, it is computationally infeasible to compute the private key

# Principles of designing a PK system (trapdoor)

- Using one-way function:
  - Given X, it is easy to compute Y = f(X)
  - Given Y it is hard to compute X = $f^{-1}(Y)$

Example:

  - Given $p_1, p_2, \ldots p_n$ it is easy to compute $N = p_1 * p_2 * \ldots * p_n$ but given N it is hard to find $p_1, p_2, \ldots p_n$

- Such an one-way function can be used as a trapdoor to create a PKC
  - Encryption is easy
  - Decryption is difficult (if not knowing the secret key)

# Merkle – Hellman's encryption scheme using ***Trapdoor Knapsack***

- 1978, Merkle & Hellman proposed an encryption scheme using this Knapsack problem:
  - Given a set of positive numbers $a_i$, $1 \le i \le n$ and $0 < T < \sum_{i=1,n} a_i$; Find a set of indexes $S \subset \{1,2,...,n\}$ such that: $\sum_{i \in S} a_i = T$
  - <u>Example:</u>

    $(a_1, a_2, a_3, a_4) = (2, 3, 5, 7)$ $T = 7$.

    There are 2 solutions: $S = (1, 3)$ as $T = a_1 + a_3$

    and $S = (4)$ as $T = a_4$
- This is a hard problem (NP-hard):
  - No P-time algorithm has been found
  - Exhaustive search: exponential time .

# Merkle – Hellman's encryption scheme

- Consider attempts to create a PK scheme using Knapsack trapdoor; here is a first attempt
  - Select a cargo vector a = $(a_1, a_2, \ldots, a_n)$
  - Encryption: for a binary plaintext block X = $(X_1, X_2, X_3 \ldots, X_n)$ compute: $T = \sum a_i X_i$ (*)
  - Decryption: Given cipher block T, knowing vector a, find $X_i$ that satisfy (*)
- Trapdoor: One way is definitely easy, the other is HARD
- BUT not yet a PK system, we need to make it easy for the owner who knows a secret key

# Merkle – Hellman's encryption scheme

- Merkle added a trick
  - using a super-increasing vector wherein the (i+1)th element is > the sum of all preceding elements (1÷i)
- Using a super-increasing cargo vector, the decryption is so easy

<u>Example</u>

Super-increasing vector: a=(1,2,4,8)

For T=11, we easily compute $X=(X_1,X_2,X_3,X_4)$ such that $T= \sum a_i X_i$ :

Let $T=T_0$

| | | | |
|---|---|---|---|
| $X_4=1$ | $T_0=T_0 - X_4=3$ | ➔ | $(X_1\ X_2\ X_3\ 1)$ |
| $X_3=0$ | $T_2=T_1=3$ | ➔ | $(X_1\ X_2\ 0\ 1)$ |
| $X_2=1$ | $T_3=T_2 - 2=1$ | ➔ | $(X_1\ 1\ 0\ 1)$ |
| $X_1= 1$ | | ➔ | $(1\ 1\ 0\ 1)$ |

# Merkle – Hellman's encryption scheme

- Exercise

  Draw a diagram/pseudo-code to describe an algorithm for the decryption using a super-increasing cargo vector

- To complete the PK scheme however the owner need to disguise his secret key, the super-increasing vector

# Merkle – Hellman: the disguise mechanism

- *Creating keys*:

  Alice creates a super-increasing vector:

  $$a' = (a_1', a_2', \ldots, a_n')$$

  *a'* will be kept as a part of the secret key

  - Then choose $m > \sum a_i'$ to be used as the modulus and choose $\omega$ that is relatively prime to $m$.
  - Now Alice's public key is the vector *a* as the product of *a'* with $\omega$

    $a = (a_1, a_2, \ldots, a_n)$

    $a_i = \omega \times a_i' \pmod{m};\ i = 1, 2, 3 \ldots n$

  - Alice's secret key is the triple $(a', m, \omega)$

# Merkle-Hellman scheme

- ***Encryption:***
  - When Bob wants to send a message X to Alice, he computes:

$$T=\sum a_i X_i$$

- ***Decryption***:
  - When Alice receives T, she will transform the equation $T = a \times X$ into $T' = a' \times X$ as follows

  She first computes $\omega^{-1}$ i.e. $\omega \times \omega^{-1} = 1$ mod $m$, then compute $T' = T \times \omega^{-1}$ $(mod\ m)$

  - Alice *then solve* $T' = a' \times X$ using the super-increasing vector a'.

- Why?

$$T' = T \times \omega^{-1} = \sum a_i X_i \omega^{-1} = \sum a_i' \omega X_i \omega^{-1}$$
$$= \sum (a_i' \omega \omega^{-1}) X_i = \sum a_i' X_i = a' \times X$$

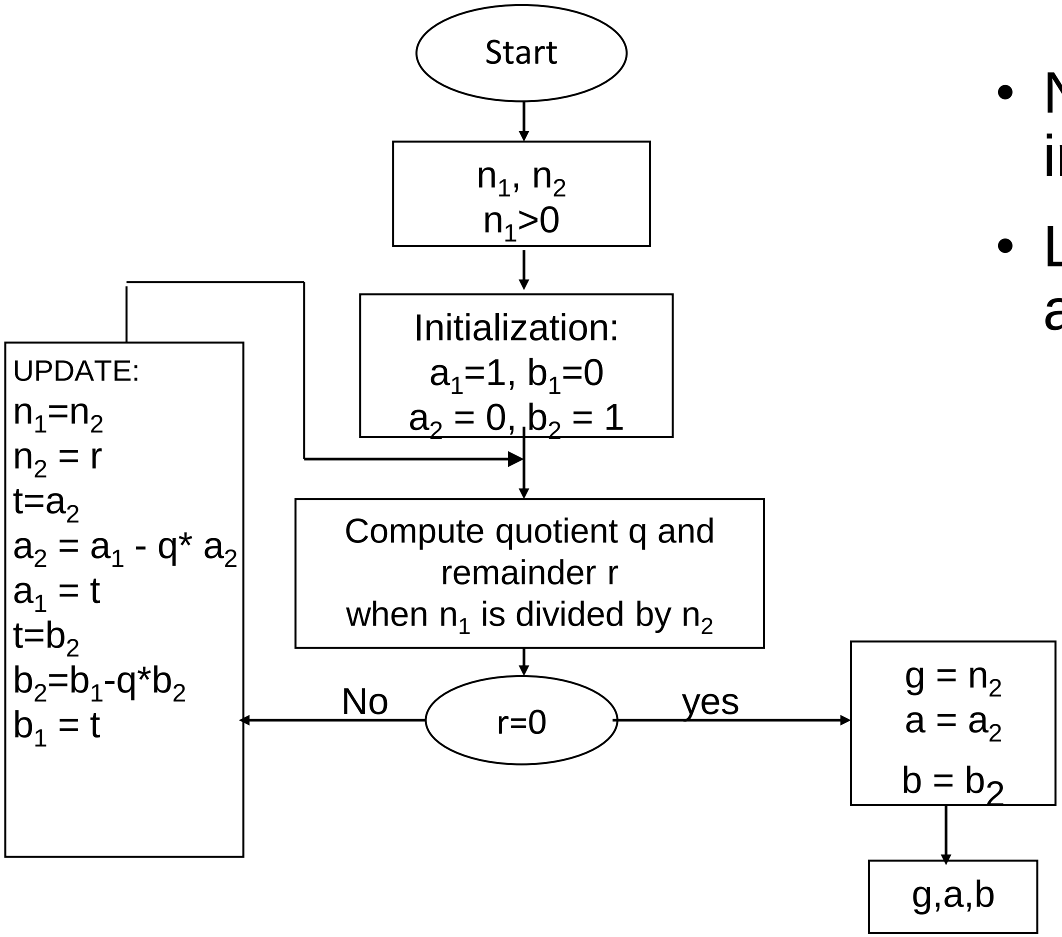
# Failure of Merkle-Hellman PKC

- **Brute Force Attack**
  - For whom not knowing the trapdoor (a', m, $\omega$), decrypting requires the exhaustive search of $2^n$ possible values of X
- **Failure of this Knapsack-based scheme (1982-1984).**
  - Shamir-Adleman showed a weakness by finding a pair ($\omega$',m') to convert a back to a' (finding the private key from the public key)
  - 1984, Brickell announced the collapse of this Knapsack-based system by one hour of computation using Cray -1 for 40 rounds and approx. 100 weights.

# Algorithm for computing modulo inverse

- Computing the inverse of $\omega$ by modulo m
  - Finding $x = \omega^{-1}$ mod m such that $x^*\omega = 1$ (mod m)
  - Many applications such as in the Knapsack trapdoor
- Based on the extended GCD algorithm or the extended Euclidean algorithm (GCD: Greatest common divisor)
  - On finding the GCD of 2 numbers $n_1$ và $n_2$, one will also compute a & b such that $GCD(n_1, n_2) = a \times n_1 + b \times n_2$.
  - If $gcd(n_1,n_2)=1$ then this e-GCD algorithm will find a, b to meet $a \times n_1 + b \times n_2 = 1$, i.e. $n_1$ is the inverse of a by modulo $n_2$

# Homework: prove the correctness of this algorithm




- Numeric example: find the inverse of 11 by modulo 39
- Let $n_1$=39, $n_2$=11 then run the algo as in the following table:

| $n_1$ | $n_2$ | r | q | $a_1$ | $b_1$ | $a_2$ | $b_2$ |
|---|---|---|---|---|---|---|---|
| 39 | 11 | 6 | 3 | 1 | 0 | 0 | 1 |
| 11 | 6 | 5 | 1 | 0 | 1 | 1 | -3 |
| 6 | 5 | 1 | 1 | 1 | -3 | -1 | 4 |

# General remarks on PKC

- Since 1976,many PKC schemes had been proposed many was broken
- A PKC have two main applications
  - Hiding information (including secrete communication)
  - Authentication with digital signatures
- The two algorithms that are most successful are RSA và El-Gamal.
- In general PKC is very slow, not appropriate for on-line encryption
  - Not used for encrypting large volume of date but for special purposes.
  - PKC and SKC are used in combined:
    - Alice and Bob use a PKC system to create a shared secret key between them and then use a SKC system to encrypt the communicated data by using this secret key

# RSA Algorithm

- Invented in **1978** by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman
  - Published as R L Rivest, A Shamir, L Adleman, "*On Digital Signatures and Public Key Cryptosystems*",Communications of the ACM, vol 21 no 2, pp120-126, Feb 1978
  - Security relies on the difficulty of factoring large composite numbers
- Essentially the same algorithm was discovered in 1973 by Clifford Cocks, who works for the British intelligence

# Main idea

- Encryption and decryption functions are modulo exponential in the field $Z_n = \{0,1,2,..n-1\}$
  - Encryption: $Y=X^e$ mod n (or $\pm$ n)
    - a = b $\pm$n ➔ a=b+k*n, $a \in Z_n$, k = 1,2,3,... e.g. 7 = 37 $\pm$10
  - Decryption: $X= Y^d \pm n$
  - The clue is that e & d must be selected such that

$X^{ed} = X \pmod n$

# Main idea

- The way to create such e&d is by using this Euler theorem: $X^{\varphi(n)}=1 \pmod n$

  - $\varphi(n)$: the size of $Z^*_n = \{k:0<k<n| (k,n)=1\}$

  - $\varphi(n)$ can be computed easily if knowing n factoralization
    - n= p*q, where p, q are primes ➔ $\varphi(n) = (p-1)(q-1)$

  - First choose *e* then compute *d* s.t. $e*d= 1 \pm \varphi(n)$
    or $d \equiv e^{-1} \bmod \varphi(n)$, which will assure that
    $X^{ed}= X^{k.\varphi(n)+1} \equiv (X^{\varphi(n)})^k *X \equiv 1^k * X = X \pmod n$

- Note this works because we know n's factorization
  - From e we compute $d \equiv e^{-1} \bmod \varphi(n)$ since we know $\varphi(n)$, otherwise it is computational infeasible to compute d s.t. $X^{ed} \equiv 1 \bmod n$

# RSA PKC

- **Key generation:**
  - Select 2 large prime numbers of about the same size, p and q
  - Compute $n = pq$, and $\Phi(n) = (q-1)(p-1)$
  - Select a random integer e, $1 < e < \Phi(n)$, s.t. $\gcd(e, \Phi(n)) = 1$
  - Compute d, $1< d< \Phi(n)$ s.t. $ed \equiv 1 \bmod \Phi(n)$
  - **Public key: (e, n) and Private key: d**
    - **Note: p and q must remain secret**

# RSA PKC (cont)

- **Encryption**
  - Given a message M, $0 < M < n$: $M \in Z_n - \{0\}$
  - use public key (e, n) compute

    $C = M^e \bmod n$, i.e. $C \in Z_n - \{0\}$

- **Decryption**
  - Given a ciphertext C, use private key (d) compute $M = C^d \bmod n$

- Why work?
  - $(M^e \bmod n)^d \bmod n = M^{ed} \bmod n = M$

# Example

- Parameters:
  - Select p = 11 vàq = 13
  - n=11*13=143; m= (p-1)(q-1) =10 *12=120
  - Choose e=37 ➔ gcd(37,120=1
  - Using  the algo gcd: e*d =1 ±120 ➔ d= 13 (e*d=481)

- To encrypt a binary string
  - Split it into segments of u bit s.t. $2^u \leq 142$ ➔ u = 7. That is each segment present a number from 0 to 127
  - Compute Y= $X^e$±143

  E.g. For X = (0000010) = 2, we have

  Y= $E_Z(X) = X^{37}$= 12±143 ➔ Y=(00001100)

- Decryption:  X= $D_Z(Y)=12^{13}$=2±143

# RSA implementation

- Execution of RSA is about thousand times slower than DES
  - Even using the fast exponential algorithm and specifically designed hardwares
- n, p, q
  - The security of RSA depends on how large n is, which is often measured in the number of bits for n. Current recommendation is 1024 bits for n.
  - p and q should have the same bit length, so for 1024 bits RSA, p and q should be about 512 bits.
  - p-q should not be small
  - Way to select p and q
    - In general, select large numbers (some special forms), then test for primality
    - Many implementations use the Rabin-Mille test, (probabilistic test)

# *Factorization Prolem*

- Estimated time using the sieve algorithm

$$L(n) \approx 10^{9.7+\frac{1}{50}\log_2 n}$$

- $\log_2 n$: the number of bits in representing n

- By 1996, for n=200, L(n) ≈ 55,000 years.

- Using parallel computing, one can factorize a 129–digit number in 3 months by distributing the workload to the computers throught out the Internet at 1996-7

- Today, for applications requiring high security levels one should values of in 1024-bit or even 2048-bit.

# *Modulo Exponential*

- Fast algorithm to compute exponential in $Z_n$ (modulo n):
  Computing $X^{\alpha}$ (modul n)

- Determine coefficients $\alpha_i$ in the binary representation of $\alpha$ :

$$\alpha = \alpha_0 2^0 + \alpha_1 2^1 + \alpha_2 2^2 + ... + \alpha_k 2^k$$

- Loop in k rounds to compute these *k* modulo exponential, với i=1,k :

$$X^2 = X \times X$$

$$X^4 = X^2 \times X^2$$

$$...$$

$$X^{2^k} = X^{2^{k-1}} \times X^{2^{k-1}}$$

- Now compute $X^{\alpha}$ mod n by multiplying theses $X^{2i}$ computed in the previous steps but only with corresponding coefficients $\alpha_i$ =1:

$$(X^{2^i})^{\alpha_i} = \begin{cases} 1, \alpha_i = 0 \\ X^{2^i}, \alpha_i = 1 \end{cases}$$

# Suggested topics for Reports

- The implementation and correctness of the extended GCD algorithm
- The probabilistic primality test
- Exponential algorithms and implementation
- The correctness of RSA algorithms
- Common Attacks to RSA

# Digital Signatures

- Motivation
  - Diffie-Hellman proposed the idea (1976)
  - Simulation of the real-world into digital worlds
    - Paper contracts need signed to be valid so do electronic versions

- The proofs conveyed in signatures
  - Data integrity: information is original, not modified
  - Authentication: The source of the info is correct, not impersonated

# DS: how they work

- Digital Signature: a data string which associates a message with some originating entity.

- Digital Signature Scheme:
  - a signing algorithm: takes a message and a (private) signing key, outputs a signature
  - a verification algorithm: takes a (public) key verification key, a message, and a signature

- A DS is created based on a PK system
  - Alice signs message X by creating $Y=D_{z_A}(X)$, so the signed document now is $(X, Y=D_{z_A}(X))$.
  - Bob who receives (X,Y), computes $X'=E_{Z_A}(Y)$ then compare if X=X' to confirm the document's validity

# Non-repudiation

- We mention more on applications of DS

- Non-repudiation

    - The signer can't deny that his/her created the document
        - Only Alice knows $z_A$ to create $(X, Y=D_{z_A}(X))$ but everyone else can verify

    - So we say the DS scheme provides non-repudiation

# Public notary

- Motivation
  - Alice may lost her secret key or someone stole it ➔ that bad guy can impersonate Alice to create documents with Alice signatures out of Alice's control
  - Alice can also deny a document truly signed by her in the past: Alice claims the document was impersonated by someone stealing her SK

- Solution: Public notary service
  - A third party – a public notary – can be hired for important documents
  - The trusted notary also signs on the same document, that is to create his signature on the concatenation of the document and Alice's signature

# Proof of delivery (receipts)

- Motivation
  - The sender need proof that the receiver has already got his message
  - The receiver can't deny that once the sender got a receipt

- Solution: An adjudicated protocol
  - A→B: $Y=E_{z_B}(D_{z_A}(X))$
  - B computes: $X'=E_{z_A}(D_{z_B}(Y))$
    - When receiving Y, B computes and checks if X'=X then signs on X' and pass to A as a receipt .
  - B→A: $Y=E_{z_A}(D_{z_B}(X'))$
    - By computing $D_{z_A}(Y)$, A now gets $D_{z_B}(X')$, a B's signature on X
  - Only when A has Y she can consider that B has receive her doc
  - Later, B can not deny receiving X since A can prove otherwise by showing $D_{z_B}(Y)$)

# Weakness of the signature scheme mentioned so far

- When using a PKC to sign X, X can be long ➔ splitting into blocks and signs

  $X = (X_1, X_2, X_3, \ldots X_t) \Rightarrow (SA(X_1), SA(X_2), SA(X_3), \ldots SA(X_t))$

- This creates vulnerability to attack on manipulating blocks
  - The attacker can change order of blocks, remove/ add in a few

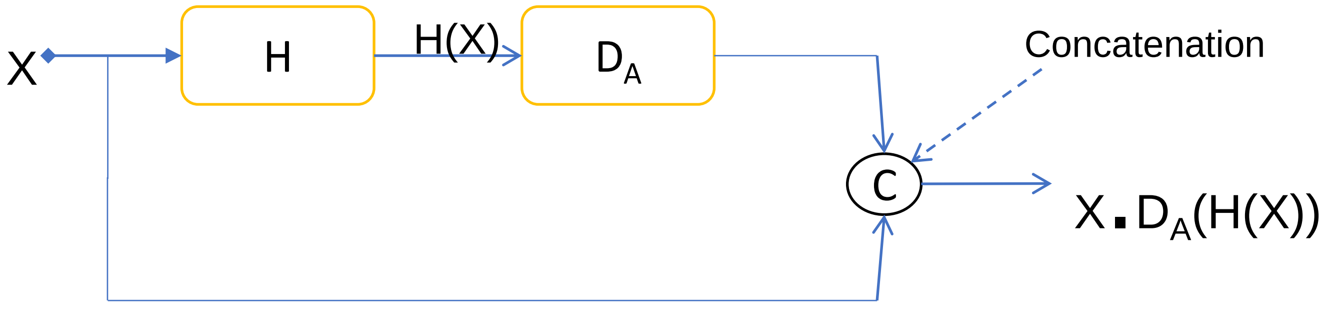
- Slow: PKC is already slow, now is run multiple times

- Signature is long, as long as the message itself.

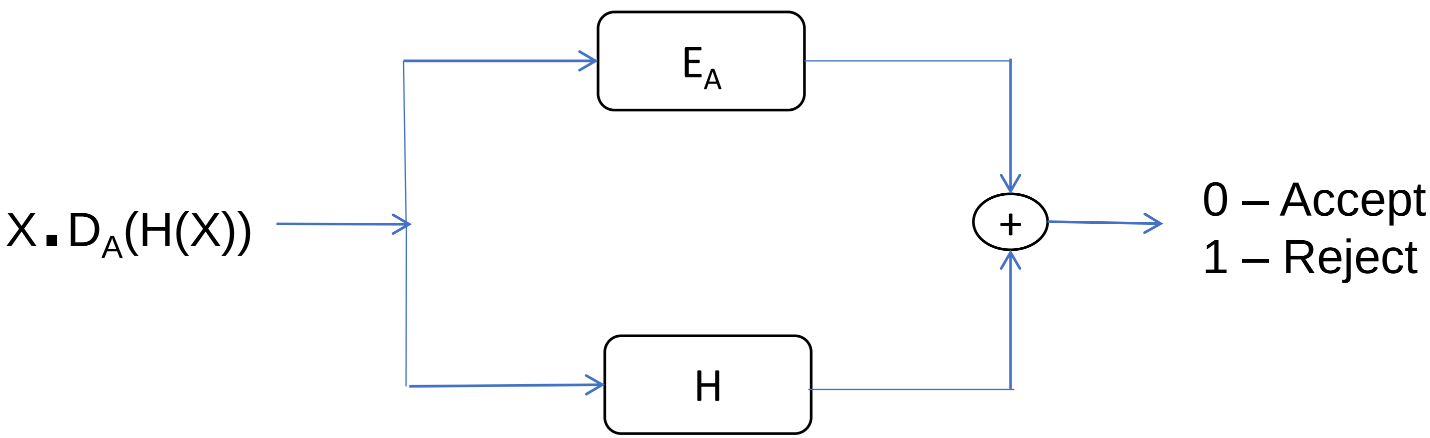
# Hash Functions

- A hash function H maps a message of variable length n bits to a fingerprint of fixed length m bits, with m < n.
  - This hash value is also called a digest (of the original message).
  - Since n>m, there exist many X which are map to the same digest ➔ collision.
- Applications
  - Digital signatures
  - Message authentication

# DS schemes with hash functions




Signature Generator




Signature Verifier

# Main properties

Given a hash function H: $X \rightarrow Y$

- Long message → short, fixed-length hash
- One-way property: given $y \in Y$

  it is computationally infeasible to find a value $x \in X$ s.t. $H(x) = y$
- Collision resistance (collision-free)

  it is computationally infeasible to find any two distinct values $x', x \in X$ s.t. $H(x') = H(x)$
  - This property prevent against signature forgery

# Collisions

- Avoiding collisions is theoretically impossible
  - Dirichlet principle: n+1 rabbits into n cages → at least 2 rabbits go to the same cage
  - This suggest exhaustive search: try |Y|+1 messages then must find a collision (H:X→Y)
- In practice
  - Choose |Y| large enough so exhaustive search is computational infeasible.
    - |Y| not too large or long signature and slow process
  - However, collision-freeness is still hard

# Birthday attack

- Can hash values be of 64 bits?
  - Look good, initially, since a space of size $2^{64}$ is too large to do exhaustive search or compute that many hash values
  - However a birthday attack can easily break a DS with a 64-bit hash function
    - In fact, the attacker only need to create a bunch of $2^{32}$ messages and then launch the attack with reasonably high probability for success.

# How is the attack

- Goal: given H, find x, x' such that H(x)=H(x')
- Algorithm:
  - pick a random set S of q values in X
  - for each $x \in S$, computes $h_x = H(x)$
  - if $h_x = h_{x'}$ for some $x' \neq x$ then collision found: (x,x'), else fail
- The average success probability is

  $\varepsilon = 1\text{-}\exp({}^{q(q-1)}/_{2|Y|})$
  - Suppose Y has size $2^m$, choose $\mathbf{q \approx 2^{m/2}}$ then $\varepsilon$ is almost 0.5!

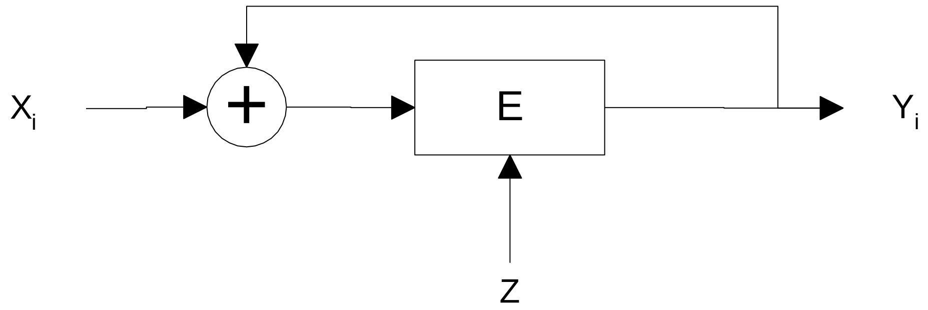
# Birthday paradox

- Given a group of people, the minimum number of people
  - such that two will share the same birthday with probability at least 50%

  is only 23 ➔ why "paradox"
  - Computing the chance
    - 1 - (1- 1/365)(1-2/365)…(1-22/365) = 1-0.493 = 0.507

# Common techniques to build hash functions

- Using SKC
  - E.g. using SKC in CBC mode
- Using modulo arithmetic operations
- Specific designs
  - MD4, MD5, SHA




$X = X_1\ X_2\ X_3\ \ldots\ X_n$

$Y_i = E_Z(X_i \oplus Y_{i-1})$

$H(X) = Y_n$

# MAC: message authentication code

- Hash function is public and the key shared between the sender and the receiver is secret
  - Sender computes mac1 = MAC(M, H, K) and sends it along with the message M
  - Receiver computes mac2 = MAC(M, H, K) and checks if mac1 = mac2 ? Yes ➔ the message is authentic; no => reject it
- The output of MAC can not be produced without knowing the secret key
  - So, this mechanism provides data integrity and ***source authentication***

# More on the Birthday Paradox

What is the probability that two persons in a room of 23 have the same birthday?

# Birthday Paradox

- Ways to assign $k$ different birthdays without duplicates:

$N = 365 * 364 * \ldots * (365 - k + 1)$

$= 365! / (365 - k)!$

- Ways to assign $k$ different birthdays with possible duplicates:

$D = 365 * 365 * \ldots * 365 = 365^k$

# Birthday “Paradox”

Assuming real birthdays assigned randomly:

N/D = probability there are no duplicates

1 - N/D = probability there is a duplicate

$$= 1 - 365! / ((365 - k)!(365)^k)$$

# Generalizing Birthdays



$$P(n, k) = 1 - n!/(n-k)!n^k$$

Given $k$ random selections from $n$ possible values, $P(n, k)$ gives the probability that there is at least 1 duplicate.

# Birthday Probabilities

P(∃ two match) = 1 – *P*(all are different)

*P*(2 chosen from *N* are different)

$= 1 - 1/N$

*P*(3 are all different)

$= (1 - 1/N)(1 - 2/N)$

*P*(*k* trials are all different)

$= (1 - 1/N)(1 - 2/N) \dots (1 - (k - 1)/N)$

ln (*P*)

$= \ln(1 - 1/N) + \ln(1 - 2/N) + \dots \ln(1 - (k - 1)/N)$

# Happy Birthday Bob!

$\ln(P) = \ln(1 - 1/N) + \ldots + \ln(1 - (k-1)/N)$

For $0 < x < 1$: $\ln(1 - x) \leq x$

$$\ln(P) \leq -(1/N + 2/N + \ldots + (k-1)/N)$$

Gauss says:

$$1 + 2 + 3 + 4 + \ldots + (k-1) + \mathrm{k} = ½\,\mathrm{k}\,(k+1)$$

So,

$$\ln(P) \leq -½\,(k\text{-}1)\,k/N$$

$$P \leq e^{½\,(k\text{-}1)k\,/\,N}$$

Probability of match $\geq 1 - e^{-½\,(k\text{-}1)k\,/\,N}$

# Applying Birthdays

$$P(n, k) > 1 - e^{-k*(k-1)/2n}$$

For $n = 365$, $k = 20$:

$$P(365, 20) > 1 - e^{-20*(19)/2*365}$$

$$P(365, 20) > .4058$$

For $n = 2^{64}$, $k = 2^{32}$: $P\,(2^{64}, 2^{32}) > .39$

For $n = 2^{64}$, $k = 2^{33}$: $P\,(2^{64}, 2^{33}) > .86$

For $n = 2^{64}$, $k = 2^{34}$: $P\,(2^{64}, 2^{34}) > .9996$

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

# Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict