Cryptography III

Public-key systems, digital signatures, 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

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: $Y=X^e \bmod n$ (or $\pm n$)
    - $a = b \pm n \rightarrow a=b+k*n$, $a \in Z_n$, $k = 1,2,3,...$ 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 $Y= X^e \pm 143$

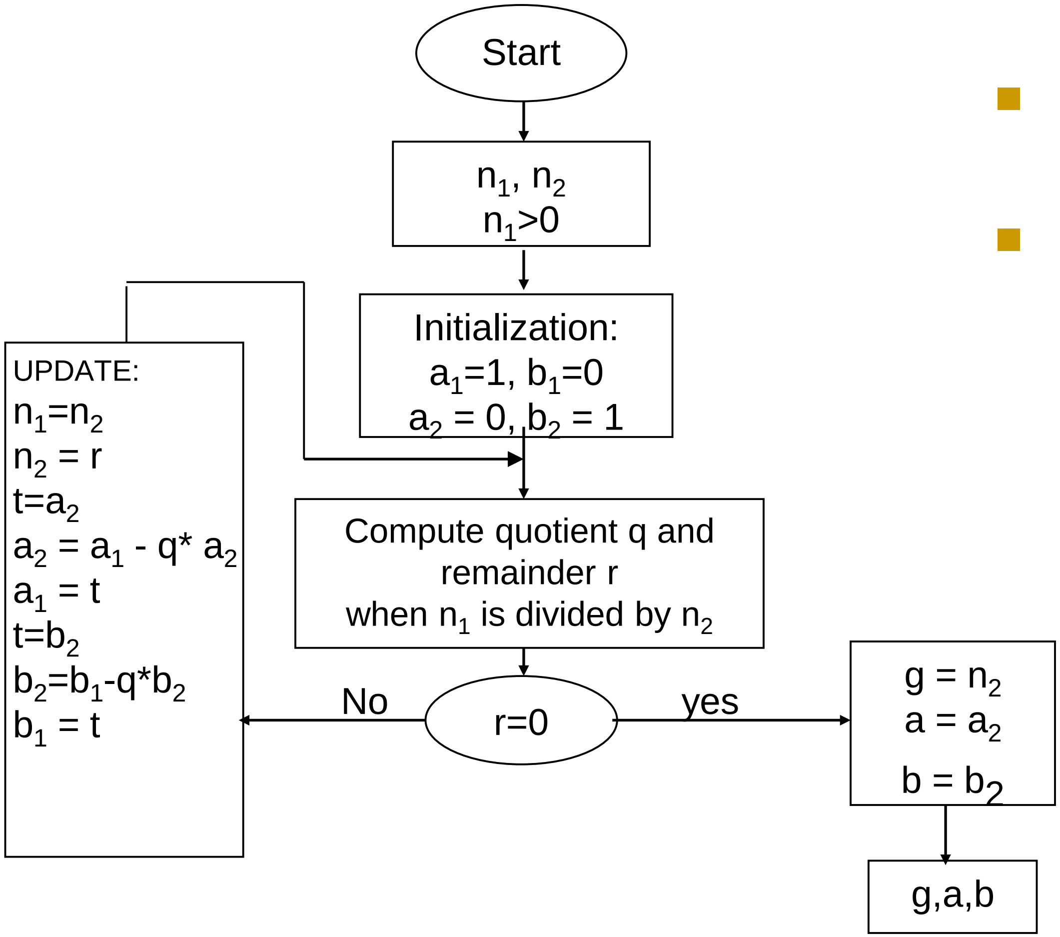
  E.g. For X = (0000010) = 2, we have

  $Y= E_Z(X) = X^{37}= 12 \pm 143$ ➔ Y=(00001100)
- Decryption: $X= D_Z(Y)=12^{13}=2 \pm 143$

Algorithm for computing modulo inverse

- Computing the inverse of ω 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 implementation

- 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) ≈ $10^{9.7+\frac{1}{50}\log_2 n}$

  - $\log_2 n$: 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^α (modul n)

- Determine coefficients α_i in the binary representation of α :
  $\alpha = \alpha_0 2^0 + \alpha_1 2^1 + \alpha_2 2^2 + \ldots + \alpha_k 2^k$

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

$$X^2 = X \times X$$
$$X^4 = X^2 \times X^2$$
$$\ldots$$
$$X^{2^k} = X^{2^{k-1}} \times X^{2^{k-1}}$$

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

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

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

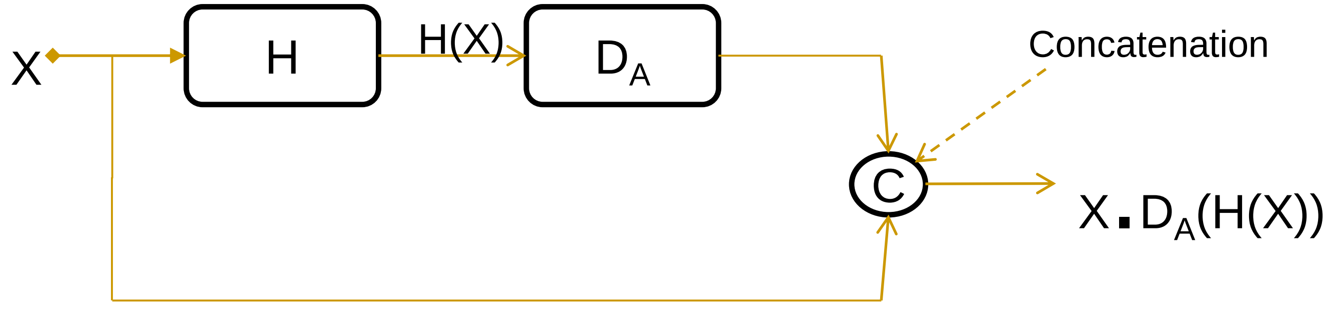
- When using a PKC to sign X, X can be long ➔ splitting into blocks and signs

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

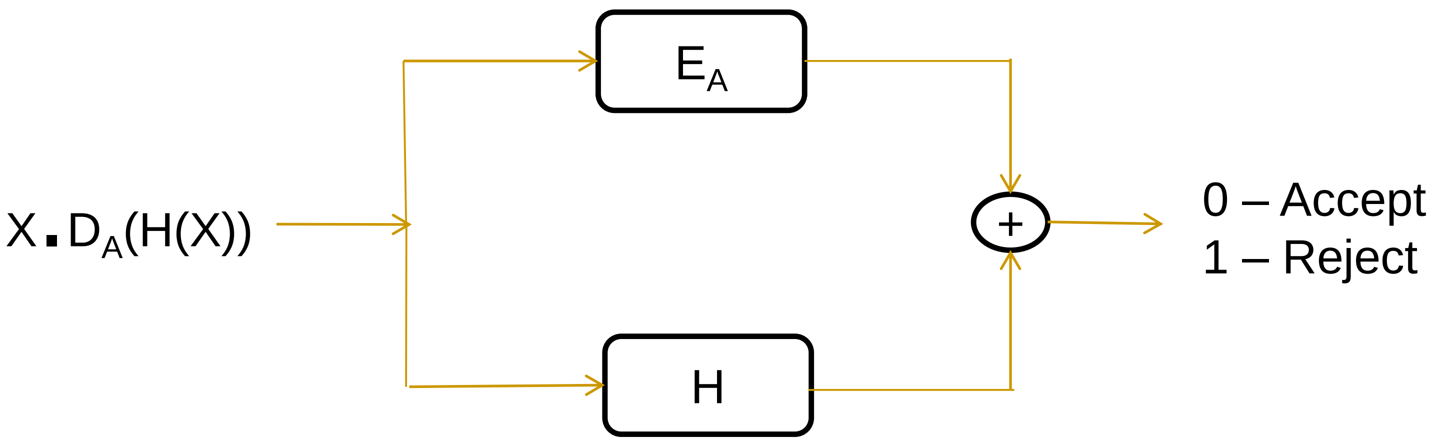
Hash Functions

- 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

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

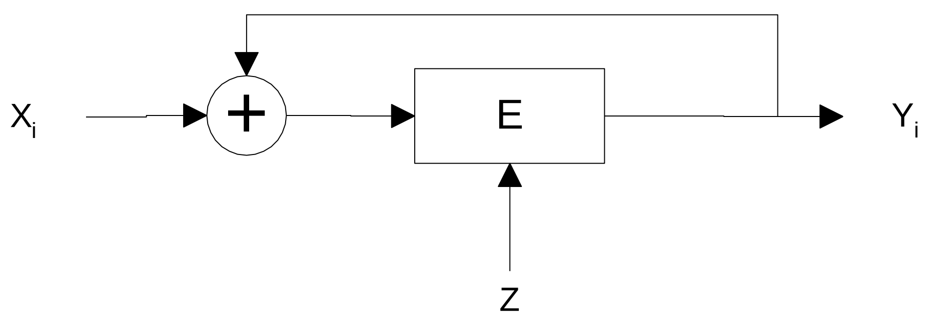
Birthday paradox

- 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***