

# INFS 766 Internet Security Protocols

## Public Key Cryptography PKI PGP

ISA 666

Duminda Wijesekera

1

## Outline

- Concepts of Public Key Cryptosystems
- Applications of Public Key Cryptosystems
- Number theory underlies most of public key algorithms.
- Well-known Public Key Cryptosystems
  - Diffie-Hellman: key exchange
  - RSA: encryption and digital signature
  - DSA: digital signature

ISE at George Mason University

ISA 666 Duminda Wijesekera

2

## Motivation of Public Key Cryptography

- Problems of secret key cryptography
  - If the shared secret key is compromised, the adversary could encrypt and decrypt all traffic between the 2 parties
  - Key distribution must be done secretly – difficult for (unknown /far away) parties
  - Need a secret key for each pair of entities – poor scalability
  - Does not provide non-repudiation!

ISE at George Mason University

ISA 666 Duminda Wijesekera

3

## Revolution in Cryptography

- Diffie & Hellman sought to solve 2 problems
  - Find a secure way to distribute keys in the public
  - Provide digital signature for document
- Public key cryptography is based on rigorous mathematical theory, rather than substitutions and permutations.
  - Factorization Problem
  - Discrete Logarithm Problem (DLP)
- It is asymmetric – requires two different keys: private key & public key

ISE at George Mason University

ISA 666 Duminda Wijesekera

4

## Applications of Public-Key Cryptosystems

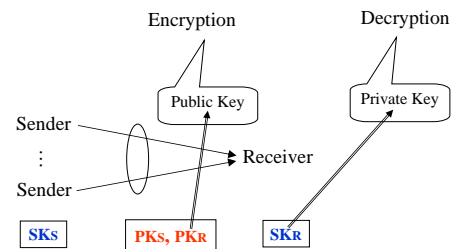
- Encryption/decryption
  - The sender encrypts a message with the receiver's public key
  - Only the receiver can decrypt the message.
- Digital signature
  - The sender signs a message with its private key.
  - Authentication and non-repudiation
- Key exchange
  - Two sides cooperate to exchange a session key.
  - Secret key cryptosystems are often used with the session key.

ISE at George Mason University

ISA 666 Duminda Wijesekera

5

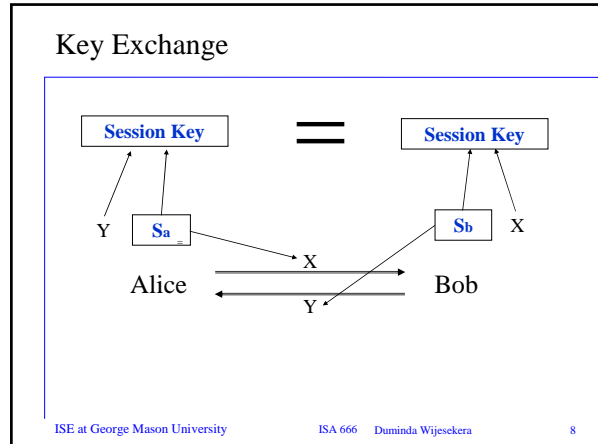
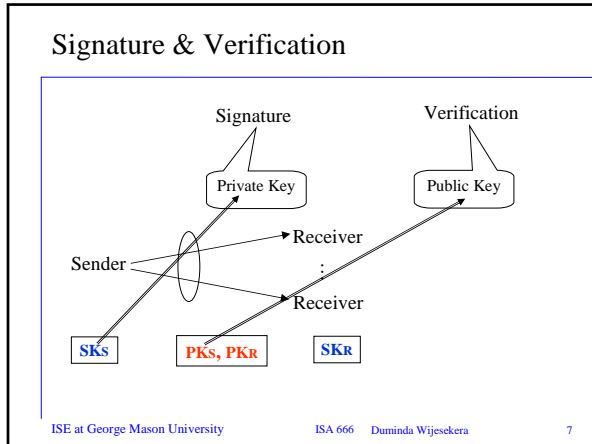
## Encryption & Decryption



ISE at George Mason University

ISA 666 Duminda Wijesekera

6



- ### Requirements for Public-Key Algorithms
- It is computationally easy to
    - generate a (public, private) key pair.
    - to generate a ciphertext using the public key.
    - to decrypt the ciphertext using the private key.
    - to sign with the private key.
    - to verify the signature with the public key.
  - It is computationally infeasible to
    - determine the private key from the public key.
    - recover the message from the ciphertext and the public key.
    - forge a signature.
- ISE at George Mason University    ISA 666    Duminda Wijesekera    9

- ### Trapdoor One-Way Function
- Essential requirement: **Trapdoor one-way function.**
  - One-way function  $f$ 
    - One-to-one mapping
    - $Y=f(X)$ : easy
    - $X=f^{-1}(Y)$ : infeasible
  - Trapdoor one-way function
    - One-to-one mapping
    - $Y=f_k(X)$ : easy if  $k$  and  $X$  are known
    - $X=f^{-1}_k(Y)$ : easy if  $k$  and  $Y$  are known
    - $X=f^{-1}_k(Y)$ : infeasible if  $Y$  is known but  $k$  is unknown.
  - **Designing public-key algorithm is to find appropriate trapdoor one-way function.**
- ISE at George Mason University    ISA 666    Duminda Wijesekera    10

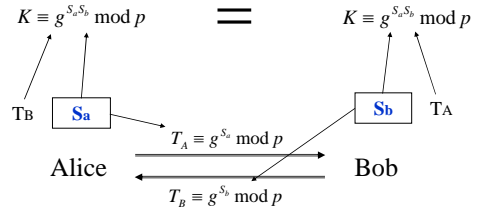
- ### Goals of Public-Key Cryptanalysis
- Given the public key, cipher text, signature, to
    - find out the private key
    - find out the message encrypted
    - forge the signature
- ISE at George Mason University    ISA 666    Duminda Wijesekera    11

- ### Public-Key Cryptanalysis
- Brute-force attack
    - Try all possible keys
  - Probable-message attack
    - The public key is known.
    - Encrypt all possible messages
    - Try to find a match between the ciphertext and one of the above encrypted messages.
  - Derivation of private key from public key
    - Try to find the relationship between the public key and the private key and compute the private key from the public one.
- ISE at George Mason University    ISA 666    Duminda Wijesekera    12

## Diffie-Hellman Key Exchange

- Published in
  - W. Diffie and ME Hellman, "New Directions in Cryptography", in IEEE Transactions on Information Theory, IT-22 no 6 (November 1976) p. 644-654
- The first public key algorithm
- Allows two users to agree on a secret key over public channel
- No encryption, decryption, nor authentication
- What's involved?
  - $p$  is a large prime number (about 1024 bits),  $g < p$  and  $g$  is a primitive root of  $p$ .
  - $p$  and  $g$  are publicly known

## Diffie-Hellman Key Exchange



## Diffie-Hellman Key Exchange

- Procedure

<u>Alice</u>	<u>Bob</u>
pick secret $S_a$ randomly	pick secret $S_b$ randomly
compute $T_A = g^{S_a} \pmod{p}$	compute $T_B = g^{S_b} \pmod{p}$
send $T_A$ to Bob	send $T_B$ to Alice
compute $T_B^{S_a} \pmod{p}$	compute $T_A^{S_b} \pmod{p}$

Alice and Bob reached the same secret  $g^{S_a S_b} \pmod{p}$ , which is then used as the shared key.

## Diffie-Hellman Example

Alice and Bob want to establish a shared secret key

- Agree on values  $p=353$  (prime) and  $g=3$
- Select the random secret values:
  - Alice chooses  $X_a=97$ , Bob chooses  $X_b=233$
- Derive the public keys:
  - $Y_a = g^{X_a} \pmod{p} = 3^{97} \pmod{353} = 40$  (Alice's)
  - $Y_b = g^{X_b} \pmod{p} = 3^{233} \pmod{353} = 248$  (Bob's)
- Derive the shared secret key
  - $K = Y_b^{X_a} \pmod{p} = 248^{97} \pmod{353} = 160$  (Alice's)
  - $K = Y_a^{X_b} \pmod{p} = 40^{233} \pmod{353} = 160$  (Bob's)

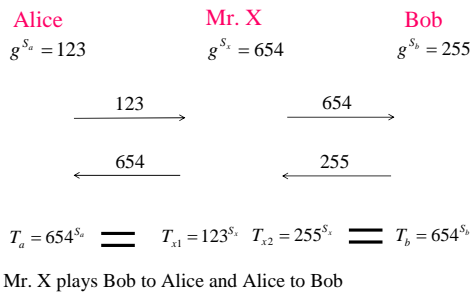
## DH Security - Discrete Logarithm Is Hard

- $T = g^s \pmod{p}$
- Given  $T, g, p$ , it is computationally infeasible to compute the value of  $s$  (discrete logarithm)
- Is DH key exchange 100% secure?

## Diffie-Hellman Scheme

- Security factors
  - Discrete logarithm very difficult.
  - Shared key (the secret) itself never transmitted.
- Disadvantages:
  - Expensive exponential operation
  - Cannot be used to encrypt anything.
  - No authentication, so you can not sign anything.

## Man-In-The-Middle Attack



## Diffie-Hellman in Phone Book Mode

- DH is subject to active man-in-the-middle attack because their public key-component may be intercepted and substituted
- Phone book mode allows everyone to generate the public key-component in advance and publish them through other reliable means
- All communicating parties agree on their common  $\langle g, p \rangle$
- Essential requirement: authenticity of the public key.

## Encryption With Diffie-Hellman

- Everyone computes and publishes  $\langle p, g, T \rangle$   
Where  $T = g^S \pmod p$
- Alice communicates with Bob:
  - Alice
    - Picks a random secret  $S_a$
    - Computes  $g^{S_a} \pmod p$
    - Use  $K_{ab} = T_b^{S_a} \pmod p$  to encrypt message
    - Send encrypted message along with  $g^{S_a} \pmod p$
  - Bob
    - $(g^{S_a})^{S_b} \pmod p = (g^{S_b})^{S_a} \pmod p = T_b^{S_a} \pmod p = K_{ab}$
    - Use  $K_{ab}$  to decrypt
- Essentially key distribution + encryption

## RSA (Rivest, Shamir, Adleman)

- Published in
  - R. Rivest, A. Shamir, and L. Adleman, "[A Method for Obtaining Digital Signatures and Public-Key Cryptosystems](#)", CACM 21, pp. 120--126, Feb. 1978
  - The first public key encryption and signature system
- Support both public key encryption and digital signature.
- Assumption/theoretical basis:
  - Factorization of large primes is hard.
- Variable key length (usually 1024 bits).
- Variable plaintext block size.
  - Plaintext must be "smaller" than the key.
  - Ciphertext block size is the same as the key length.

## The RSA Algorithm

- To generate key pair:
  - Pick large primes  $p$  and  $q$
  - Let  $n = p * q$ , keep  $p$  and  $q$  to yourself!
  - For public key, choose  $e$  that is relatively prime to  $\phi(n) = (p-1)(q-1)$ .  
public key =  $\langle e, n \rangle$
  - For private key, find  $d$  that is the multiplicative inverse of  $e \pmod{\phi(n)}$ , i.e.,  $e * d = 1 \pmod{\phi(n)}$
  - Private key =  $\langle d, n \rangle$ .

## How Does RSA Work?

- Given  $\text{pubKey} = \langle e, n \rangle$  and  $\text{privKey} = \langle d, n \rangle$
- Message =  $m$ 
  - encryption:  $c = m^e \pmod n, m < n$
  - decryption:  $m = c^d \pmod n$
  - signature:  $s = m^d \pmod n, m < n$
  - verification:  $m = s^e \pmod n$

## An Example

- Choose  $p = 7$  and  $q = 17$ .
- Compute  $n = p * q = 119$ .
- Compute  $\phi(n) = (p-1)(q-1) = 96$ .
- Select  $e = 5$ , which is relatively prime to  $\phi(n)$ .
- Compute  $d = 77$  satisfying  $e * d = 1 \pmod{\phi(n)}$ .
- **Public key:**  $\langle 5, 119 \rangle$
- **Private key:**  $\langle 77, 119 \rangle$
- **Message** = 19
- **Encryption:**  $19^5 \pmod{119} = 66$
- **Decryption:**  $66^{77} \pmod{119} = 19$ .

## Why Does RSA Work?

- Given pub =  $\langle e, n \rangle$  and private =  $\langle d, n \rangle$ 
  - $n = p * q, \phi(n) = (p-1)(q-1)$
  - $e * d = 1 \pmod{\phi(n)}$
  - $x^{e * d} = x \pmod{n}$
  - encryption:  $c = m^e \pmod{n}$
  - decryption:  $m = c^d \pmod{n} = m^{e * d} \pmod{n} = m \pmod{n}$   
=  $m$  (since  $m < n$ )
  - digital signature (similar)

## Practical Considerations of RSA

- How to find large prime numbers?
- How to do modular exponentiation with large numbers?
  - $123456789^{987654321} \pmod{234567891} = ?$
- How to do RSA encryption/decryption, signature efficiently?

## Finding Large Prime Numbers

- Good news
  - Infinite number of prime numbers ☺
- Bad news
  - The prime number ratio decreases as the prime number gets big ☹
- Brute-force
  - Try to divide  $n$  by  $2 \dots n^{1/2}$
  - Impractical for large number!!!
- No known practical method to determine if a given large number is prime ☹
- However fast probabilistic primality test exists.  
That is, determine if a larger number is likely to be a prime.

## Finding Large Prime Numbers (Cont'd)

- Primality test
  - Randomly pick  $0 < a < n$ , see if  $a^{n-1} \pmod{n} = 1$ ?
  - If  $a^{n-1} \pmod{n} \neq 1$ ,  $n$  is not prime for sure
  - If  $a^{n-1} \pmod{n} = 1$ ,  $n$  is very likely to be prime.
  - The false positive rate is  $10^{-13}$  for 100 digit number
    - Exist  $n > 0$  such that  $a^{n-1} \pmod{n} = 1$  for all  $0 < a < n$
- Implication
  - We may (with small probability) choose some non-prime numbers for  $p$  &  $q$ , which would fail RSA operations (encryption/decryption, signature/verification)

## Modular Exponentiation with Big Number

- To calculate  $66^{77} \pmod{119}$ 
    - multiply 66 by itself for 76 times, then mod 119
    - take mod 119 every time multiply 66
      - take advantage of property of modulo operation:
- $$x \times y \pmod{n} = [(x \pmod{n}) \times (y \pmod{n})] \pmod{n}$$
- “Squaring and Multiplication”
  - The number of square and multiplication operations is linear to the number of bits of the exponent 77

## Fast Modular Exponentiation

- To calculate  $x^y \bmod n$ 
  - Assume  $y = b_0 b_1 \dots b_{k-1}$  has  $k$  bits
  - $z = 1$
  - for ( $i = 0; i < k; i++$ )
  - {  $z = (z \times z) \bmod n;$
  - if ( $b_i = 1$ )
  - $z = (z \times x) \bmod n;$
  - }
  - /\* now  $z = x^y \bmod n$  \*/

## Fast Modular Exponentiation Example

- To calculate  $66^{77} \bmod 119$  ( $77 = 1001101$ )
  - 1  $66 \bmod 119 = 66$
  - 10  $66^2 \bmod 119 = 72$
  - 100  $66^4 \bmod 119 = (66^2 \bmod 119)^2 \bmod 119 = 67$
  - 1000  $66^8 \bmod 119 = (66^4 \bmod 119)^2 \bmod 119 = 86$
  - 1001  $66^9 \bmod 119 = [66(66^8 \bmod 119)] \bmod 119 = 83$
  - 10010  $66^{18} \bmod 119 = (66^9 \bmod 119)^2 \bmod 119 = 106$
  - 10011  $66^{19} \bmod 119 = [66(66^{18} \bmod 119)] \bmod 119 = 94$
  - 100110  $66^{38} \bmod 119 = (66^{19} \bmod 119)^2 \bmod 119 = 30$
  - 1001100  $66^{76} \bmod 119 = (66^{38} \bmod 119)^2 \bmod 119 = 67$
  - 1001101  $66^{77} \bmod 119 = [66(66^{76} \bmod 119)] \bmod 119 = 19$
- How many modular multiplications are needed for exponentiation with a 512 bit exponent?
  - on average 768
  - why?

## Further Optimization of RSA Calculation

- Choose small public number  $e$ 
  - $e = 3$ 
    - Faster encryption with public key
    - Faster signature verification with public key
  - Make sure
    - $m > n^{1/3}$ , otherwise,  $m$  could be determined easily from  $c$ .
    - Not the same message to multiple recipients  $\langle 3, n_1 \rangle, \langle 3, n_2 \rangle, \dots, \langle 3, n_k \rangle$ , otherwise,  $m$  can be derived by Chinese remainder theorem
    - 3 is relative prime to  $\phi(n)$
- $e = (2^{16} + 1) = 65537$

## RSA Signing/Decryption Optimization

- Want to calculate  $m = c^d \bmod n$  where  $n = p \cdot q$
  - Easy to compute  $[c^d \bmod p]$  and  $[c^d \bmod q]$  and combine
- Do
- Compute:  $c_p = c \bmod p, c_q = c \bmod q$
  - $m_p = c_p^d \bmod p$  and  $m_q = c_q^d \bmod q$
  - Use Chinese remainder theorem to get  $m = c^d \bmod pq$
- Note: Don't need to compute  $c^d \bmod p$  etc, because for  $d \gg p, c^d \bmod p = c^{d \bmod (p-1)} \bmod p$
- Example Applications: Compute  $\langle e, d \rangle$  where  $p=3, q=7$ .
- Then  $n = 3 \cdot 7 = 21$  and  $\phi(n) = (3-1)(7-1) = 12$ .
  - Choose  $e=5$ , then  $d=17$  so  $\langle e, d \rangle = \langle 5, 17 \rangle, m=4$
  - $c = m^e \bmod 21 = 4^5 \bmod 21 = 1024 \bmod 21 = 16$
  - $c_p = 16 \bmod 3 = 1$
  - $c_q = 16 \bmod 7 = 2$
  - $m = c^d \bmod 21 = 16^{17} \bmod 21$ .
  - To Solve this try finding  $16^3 \bmod 3$  and  $16^3 \bmod 7$ .
    - $16^3 \bmod 3 = 16^{3 \bmod (3-1)} \bmod 3 = 16^1 \bmod 3 = 1$
    - Similarly,  $16^3 \bmod 7 = 16^{3 \bmod (7-1)} \bmod 7 = 16^1 \bmod 7 = 2$
  - According to Chinese remainder theorem, we have  $c^d \bmod 21 = 4 = m$

## RSA Optimization Example

- If  $d$  is 512-bits and  $p, q$  are 256-bits each
  - Thus can break one 512-bit exponentiation into two 256-bit exponentiations
  - 768 modular multiplications  $\Rightarrow$  transformed into  $\Rightarrow 384 + 384$  modular multiplications
  - Makes RSA about twice as fast

## RSA Performance vs DES

- Fast RSA implementations can encrypt KB/Sec
  - (like a analog modem)
- Fast DES implementations can encrypt MB/Sec
  - (like a LAN)
- Two to three orders difference in speed is likely to remain
  - Determined by inherent computation complexity
- <http://www.eskimo.com/~weidai/benchmarks.html>

## The Security of RSA

- Attacks against RSA
  - Brute force: Try all possible private keys
    - Can be defeated by using a large key space
  - Mathematical attacks
    - Factor  $n$  into  $n=p*q$ .
    - Determine  $\phi(n)$  directly: equivalent to factoring  $n$ .
    - Determine  $d$  directly: at least as difficult as factoring  $n$ .
  - Timing attacks
    - Recover the private key according to the running time of the decryption algorithm.

## The Security of RSA (Cont'd)

- Factoring large integer is very hard!
- But if a big number  $n$  can be factored, then given public key  $\langle e, n \rangle$ , can find  $d$ , and hence the private key by:
  - Knowing factors  $p, q$ , such that,  $n = p*q$
  - Then  $\phi(n) = (p-1)(q-1)$
  - Then  $d$  such that  $e*d = 1 \pmod{\phi(n)}$
- Ways to make  $n$  difficult to factor
  - $p$  and  $q$  should differ in length by only a few digits
  - Both  $(p-1)$  and  $(q-1)$  should contain a large prime factor
  - $\gcd(p-1, q-1)$  should be small.
  - $d > n^{1/4}$ .

## The Security of RSA (Cont'd)

- Timing attacks
  - Determine a private key by observing how long it takes to decipher messages.
  - The attack proceeds computing bit by bit.
  - The attacker is able to determine bit  $j$  because for some  $d$  and  $a$ , the marked step is extremely slow

Algorithm for computing  $a^b \pmod n$ .

```

d ← 1
For i ← 0 till to k-1
    d ← d*d mod n
    If  $b_i = 1$ 
        Then d ← d*a mod n
Return d.
```

## The Security of RSA (Cont'd)

- Countermeasures against the timing attack
  - Constant exponentiation time
    - Don't return the result if the computation is too fast.
    - Hurt the performance.
  - Random delay
    - Confuse the timing attack by adding a random delay.
    - The attacker may be able to defeat random delay if the delay is not added carefully.
  - Blinding
    - Multiply the ciphertext by a random number before performing exponentiation.

## The Security of RSA (Cont'd)

- RSA Data Security's blinding algorithm
  - Generate a random number  $r$  between 0 and  $n-1$  such that  $\gcd(r, n) = 1$ .
  - Compute  $C' = C * r^e \pmod n$
  - Compute  $M' = (C')^d \pmod n$
  - Compute  $M = M' * r^{-1} \pmod n$ .
  - Performance penalty: 2 – 10%.

## Digital Signature Standard (DSS)

- Developed by NIST
- Related to El Gamal - a variation
- Use SHA (SHA-1) to generate the hash value and Digital Signature Algorithm (DSA) to generate the digital signature.
- Faster for the signer, but not for the verifier:
- Potential application: smart cards

## Digital Signature Algorithm (DSA)

- **Pre Processing Stage: generate public parameters**
  - $p$  (512 to 1024 bit prime)
  - $q$  (160 bit prime):  $q|p-1$
  - $g = h^{(p-1)/q} \bmod p$ , where  $1 < h < (p-1)$  such that  $g > 1$ .
  - **Note:**  $g$  is of order  $q \bmod p$  and  $g^q = h^{p-1} = 1 \bmod p$
- **Choose a long term public/private key pair (T,S)**
  - User's private key  $S$ 
    - Random integer with  $0 < S < q$
  - User's public key  $y$ 
    - $T = (g^S \bmod p)$

## During message transmission

- **Transmission stage: Choose a per message private/public key pair ( $T_m, S_m$ )**
  - Message =  $m$
  - Choose a random integer  $S_m$  as private key satisfying  $0 < S_m < q$ .
  - Choose  $T_m = ((g^{S_m} \bmod p) \bmod q)$  as the public key
  - Calculate  $S_m^{-1} \bmod q$
  - Compute the message digest  $d_m$  using SHS
  - Signature  $X = S_m^{-1}(d_m + ST_m) \bmod q$
- **Transmitted information**
  - The message  $m$
  - Per message public key  $T_m$
  - Computed signature  $X = S_m^{-1}(d_m + ST_m) \bmod q$

## On Retrieval

- Compute  $X^{-1} \bmod q$
- Compute  $d_m$  from the message by using SHA
- Compute  $x = d_m(X^{-1} \bmod q)$
- Compute  $y = T_m(X^{-1} \bmod q)$
- Compute  $z = (g^x (T^y \bmod p)) \bmod q$
- If  $z = T_m$  then the signature is verified

## DSS: Correctness of the Verification

- Let  $v = (d_m + ST_m)^{-1} \bmod q$ . Then
  - $X^{-1} = (S_m^{-1}(d_m + ST_m))^{-1} \bmod q$   
 $= S_m(d_m + ST_m)^{-1} = S_m v \bmod q$
- Hence
- $x = d_m \cdot X^{-1} = d_m S_m v \bmod q$
  - $y = T_m \cdot X^{-1} = T_m S_m v \bmod q$
  - $Z = g^x T^y = g^{d_m S_m v} g^{S T_m S_m v} = g^{(d_m + ST_m) S_m v} = g^{S_m v} = S_m \bmod p \bmod q$

## Why Is DSA Secure?

- No revealing of private key  $S$
- Can't forge a signature without  $S$
- No duplicate messages with matched signature
- Need a per-message secret number  $k$ 
  - If  $k$  is known, the private key  $x$  can be computed
  - Two messages sharing the same  $k$  can reveal the private key  $x$

## Digital Signature Brainstorming

- What's the difference between a physical signature on paper and a digital signature?
- Ownership proof
  - Physical signature on a book?
  - Digital signature on the pdf file of the book?

## Public Key Cryptography Brainstorming

- What if Alice's public key is forged?
  - Everyone (but Alice) somehow get a fake public key (generate by Bob) and they all believe it is Alice's public key
- Alice can not communicate with everyone through public key cryptosystem
  - Alice can not decrypt anything encrypted by others using the fake Alice's public key
  - When Alice signs anything with her private key, everyone else will think it is fake
- **It is critically important to have everyone have authentic public keys of everyone!**
- But how?

## Public Key Infrastructure (PKI)

- What is PKI for?
  - Facilitate secure distribution of every one's public key
  - Provide some "authenticity" of the public keys distributed
- PKI consists of
  - Certificates
  - Certification authorities (CA)
  - Certificate repository
  - Means to revoke certificate
  - Certificate chaining

## Certificate

- A signed message claiming somebody's public key is such such
  - [Alice's public key is 12345]<sub>Bob</sub>
- If Carol does not know Bob or Bob's key, then the certificate signed by Bob doesn't mean much to Carol
- If Carol knows and trust David, who can certify Bob's key, then Carol may have some trust on Bob's certificate
- There could be a chain of certificate that indirectly certify the authenticity of the public key

## PKI Trust Models

- Trust anchor
  - The public key that the verifier trusts in advance (through some other means, ie manually handed by a long-time friend)
- Where to get the trust anchor?
- How to derive trust from the trust anchor?

## Certificate Revocation

- There is a need to revoke a certificate before it is expired
  - Lost credit card
  - Termination of affiliation with some organization
- Revocation Mechanisms
  - CRL (certification revocation list)
  - Delta CRLs
  - First valid certificate
  - OLRs (online revocation server)
  - Good-list, bad-list

## PGP

- Read Chapter 22

## Question?

- Given that Alice's public/private key pair is  $(PK_A, SK_A)$ , Bob's public/private key pair is  $(PK_B, SK_B)$ .
- Alice wants to transfer a message  $m$  to Bob
- Requirement: confidentiality, authentication, non-repudiation