



CSC 474/574

Information Systems Security

Topic 2.4 Public Key Algorithms

Public Key Algorithms

- Public key algorithms covered in this class
 - RSA: encryption and digital signature
 - Diffie-Hellman: key exchange
 - DSA: digital signature
- Number theory underlies most of public key algorithms.

Use 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.

Requirements for Public-Key Algorithms

- It is computationally easy to generate a pair of public key and private key.
- It is computationally easy to generate a ciphertext using the public key.
- It is computationally easy to decrypt the ciphertext using the private key.
- It is computationally infeasible to determine the private key from the public key.
- It is computationally infeasible to recover the message from the ciphertext and the ciphertext.

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
 - $X=f_k^{-1}(Y)$: easy if k and Y are known
 - $X=f_k^{-1}(Y)$: infeasible if Y is known but k is unknown.
- Designing public-key algorithm is to find appropriate trapdoor one-way function.

Public-Key Cryptanalysis

- Brute-force attack
 - Try all possible keys
- 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.
- 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.

RSA (Rivest, Shamir, Adleman)

- The most popular one.
- 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.

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 $\phi(n) = (p-1)(q-1)$, let $\text{pub} = \langle e, n \rangle$
 - For private key, find d that is the multiplicative inverse of $e \bmod \phi(n)$, i.e., $e * d = 1 \bmod \phi(n)$, let $\text{pri} = \langle d, n \rangle$.

How Does RSA Work?

- Given $\text{pub} = \langle e, n \rangle$ and $\text{priv} = \langle d, n \rangle$
 - encryption: $c = m^e \bmod n, m < n$
 - decryption: $m = c^d \bmod n$
 - signature: $s = m^d \bmod n, m < n$
 - verification: $m = s^e \bmod n$

An Example

- Choose $p = 7$ and $q = 17$.
- Compute $n = p * q = \underline{\hspace{1cm}}$.
- Compute $\varphi(n) = (p-1)(q-1) = \underline{\hspace{1cm}}$.
- Select $e = 5$, which is relatively prime to $\varphi(n)$.
- Compute $d = \underline{77}$ such that $e * d = 1 \bmod \varphi(n)$.
- Public key: $\langle \underline{\hspace{1cm}}, \underline{\hspace{1cm}} \rangle$
- Private key: $\langle \underline{\hspace{1cm}}, \underline{\hspace{1cm}} \rangle$
- Encryption: $19^5 \bmod 119 = 66$
- Decryption: $66^{77} \bmod 119 = 19$.

Why Does RSA Work?

- Given $\text{pub} = \langle e, n \rangle$ and $\text{priv} = \langle d, n \rangle$
 - $n = p * q, \phi(n) = (p-1)(q-1)$
 - $E * d = 1 \pmod{\phi(n)}$
 - $x^{e*d} = x \pmod n$
 - 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)

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 you can factor big number n then given public key $\langle e, n \rangle$, you 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 by observing how long a computer takes to decipher messages.
 - The attack proceeds 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

$a^b \pmod n$.

$d \leftarrow 1$

For $i \leftarrow k$ downto 0

$d \leftarrow d * a \pmod n$

→ If $b_i = 1$

Then $d \leftarrow d * a \pmod n$

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 $C' = C * r^e \bmod n$
 - Compute $M' = (C')^d \bmod n$
 - Compute $M = M' * r^{-1} \bmod n$.
 - Performance penalty: 2 – 10%.

Diffie-Hellman Key Exchange

- Shared key, public communication
- No authentication of partners
- What's involved?
 - p is a large prime number (about 512 bits), $g < p$ and g is a primitive root of p .
 - p and g are publicly known

Diffie-Hellman Key Exchange

- Procedure

Alice

pick secret S_a randomly
randomly

compute $T_A = g^{S_a} \bmod p$

send T_A to Bob

compute $T_B^{S_a} \bmod p$

Bob

pick secret S_b

compute $T_B = g^{S_b}$

send T_B to Alice

compute $T_A^{S_b} \bmod p$

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

DH Security - Discrete Logarithm Is Hard

- $T = g^s \text{ mod } p$
- Given T, g, p , it is computationally infeasible to compute the value of s (discrete logarithm)

Diffie-Hellman Scheme

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

Man-In-The-Middle Attack

Alice

$$g^{Sa}=123$$

$$123 \square$$

$$654^{Sa}=123^{Sx}$$

Mr. X

$$g^{Sx}=654$$

$$654 \square$$

$$\square 654$$

Bob

$$g^{Sb}=255$$

$$\square 255$$

$$255^{Sx}=654^{Sb}$$

- Mr. X plays Bob to Alice and Alice to Bob

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

Digital Signature Standard (DSS)

- By NIST
- Related to El Gamal
- Use SHA (SHA-1) to generate the hash value and Digital Signature Algorithm (DSA) to generate the digital signature.
- Speeded up for signer rather than verifier: smart cards

Digital Signature Algorithm (DSA)

- 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 $g > 1$.
 - g is of order $q \bmod p$.
- User's private key x
 - Random integer with $0 < x < q$
- User's public key y
 - $y = g^x \bmod p$
- User's per message secret number
 - $k =$ random integer with $0 < k < q$.

DSA (Cont'd)

- Signing
 - $r = (g^k \bmod p) \bmod q$
 - $s = [k^{-1}(H(M) + xr)] \bmod q$
 - Signature = (r, s)
- Verifying
 - $M', r', s' =$ received versions of M, r, s .
 - $w = (s')^{-1} \bmod q$
 - $u_1 = [H(M')w] \bmod q$
 - $u_2 = (r')w \bmod q$
 - $v = [(g^{u_1}y^{u_2}) \bmod p] \bmod q$
 - if $v = r'$ then the signature is verified

Why Is DSA Secure

- No revealing of private key x
- Can't forge a signature without x
- 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