Asymmetric Ciphers: RSA

2019. 3. 19

Contents

- Introduction to crypto
- Symmetric-key cryptography
  - Stream ciphers
  - Block ciphers
  - Block cypher operation modes
- Asymmetric-key cryptography
  - RSA
  - Diffie-Hellman
  - ECC
  - Digital signature
  - Public key Infrastructure
- Cryptographic hash function
  - Attack complexity
  - Hash Function algorithm
- Integrity and Authentication
  - Message authentication code
  - GCM
  - Digital signature
- Key establishment
  - server-based
  - Public-key based
  - Key agreement (Diffie-Hellman)

Limitation of symmetric key

- Key distribution problem
  - How can keys be exchanged secretly?
- Too many symmetric keys
  - For n users, each user should keep n-1 keys and in total n(n+1)/2 keys are required.
- Alice and Bob may cheat each other.
  - They may deny the use of their keys.
  - Need a way for non-repudiation

Asymmetric key cryptography

- Two keys
  - Each user generates two keys: **public key** and **private key**
  - Each user keeps its private key in secret, but lets others know its own **public key**.
  - At key generation time, two keys are computed.

K^+ (public key) K^- (private key)

Uses of Public Key Crypto

- Encryption
  - Suppose we encrypt M with Bob's public key
  - Bob's private key can decrypt to recover M
- Digital Signature
  - Sign by encrypting with sender's private key
  - Anyone can **verify** signature by decrypting with the sender's public key
  - Like a handwritten signature, but much better than that.
- Key exchange
  - We will talk about it later.

How to build public key crypto

- Based on "trap door one-way function"
  - "One-way" means easy to compute in one direction, but hard to compute in other direction
  - One-way function f(x)
    - Computing $y=f(x)$ is computationally easy.
    - Computing $x=f^{-1}(y)$ is computationally infeasible.
  - "Trap door" used to create key pairs

3 kinds of public key crypto

- There are 3 kinds of mathematically hard one-way functions on which the public key crypto are based.
  - Factoring integers
    - RSA
  - Discrete Logarithm
    - Diffie-Hellman, Elgamal, DSA
  - Elliptic curve: generalized discrete log
    - ECDH, ECDSA

RSA

History

- Diffie and Hellman published the idea of the public key crypto in 1976.
- The RSA crypto was published by **R**ivest, **S**hamir, and **A**dleman (MIT) in 1977, and Clifford Cocks (GCHQ), independently,
- So far, RSA is the most widely used the public key cypto although ECC is gaining attention recently.

Factoring Integers

- Let p and q be two large prime numbers
- Compute $N = pq$, which is easy.
- but, to find p and q from N such that $N=pq$ for large enough p and q is computationally very hard problem.

Encryption and Decryption

- **Public key** $K^{+}=(N,e)$
- **Private key** $K^{-}= d$
- Encryption $y=E_{K+}(x) = x^{e} \bmod N$
- Decryption $x=D_{K-}(y) = y^{d} \bmod N$

keys generation algorithm

At the setup time, the public and private keys are computed as follows:

1. Choose two large prime numbers (p, q<2^{512})
2. Compute N=p·q (N<2^{1024})
3. Compute φ(N)=(p-1)(q-1)
4. Choose e ∈{1,2,3,..., φ(N)-1} such that
   gcd(e, φ(N)) = 1
5. Compute d such that
   d·e=1 mod φ(N) (by Extended Euclidean Alg.)
6. Return K^+=(N,e) and K^-= d

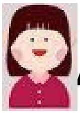
RSA encrypt/decrypt

- Message M is treated as a number ($M \in \{0,1,2,\ldots,N-1\}$)
- To encrypt M, compute
  $C = M^e \bmod N$
- To decrypt C, compute
  $M = C^d \bmod N$
- Recall that N and e are public
- If an attacker can factor N=pq, he can use e to easily find d since $ed = 1 \bmod (p-1)(q-1)$
- Factoring the modulus breaks RSA
  - Is the factoring the only way to break RSA?

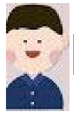
Does RSA really work?

- Given $C = M^e \bmod N$ we must show
  $M = C^d \bmod N = M^{ed} \bmod N$
- We'll use **Euler's Theorem:**
  If x is relatively prime to n, then $x^{\varphi(n)} = 1 \bmod n$
- Facts:
  1) $ed = 1 \bmod (p-1)(q-1)$
  2) By definition of "mod", $ed = k(p-1)(q-1) + 1$
  3) $\varphi(N) = (p-1)(q-1)$
- Then $ed - 1 = k(p-1)(q-1) = k\varphi(N)$
- Finally, $\mathbf{M^{ed}} = M^{(ed-1)+1} = M \cdot M^{ed-1} = M \cdot M^{k\varphi(N)} = M \cdot (M^{\varphi(N)})^k \bmod N = M \cdot 1^k \bmod N =$ **$\mathbf{M} \bmod \mathbf{N}$**

Simple RSA Example


Alice

Message $M=8$


Bob

1. Select large primes $p=11$, $q=3$
2. $N=pq=33$
3. $\varphi(n)=(p-1)(q-1)=20$
4. Choose $e=3$ (relatively prime to 20)
5. Find $d=7$ such that $ed=1 \bmod 20$

$K+ = (33,3)$

$K- = 7$

$Y = M^e \bmod 33$
$= 8^3 = 512 = 17 \bmod 33$

$C=17$

$M = C^d \bmod N = 17^7 = 410{,}338{,}673$
$= 12{,}434{,}505 * 33 + 8 = 8 \bmod 33$

RSA computation time

- RSA computations are involved in only one arithmetic operation, i.e., modular exponential computation.
- For encryption,

$$C = M^e \bmod N$$

- For decryption,

$$M = C^d \bmod N$$

Fast way of doing modular exponentiation

- Suppose that we compute 5^{20}
  - $5^{20} = 95367431640625 = 25 \bmod 35$
- A better way: **repeated squaring(Square-and-multiply)**
  - 20 = 10100 base 2
  - (1, 10, 101, 1010, 10100) = (1, 2, 5, 10, 20)
  - Note that $2 = 1 \cdot 2$, $5 = 2 \cdot 2 + 1$, $10 = 2 \cdot 5$, $20 = 2 \cdot 10$
  - $5^1 = 5 \bmod 35$
  - $5^2 = (5^1)^2 = 5^2 = 25 \bmod 35$
  - $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \bmod 35$
  - $5^{10} = (5^5)^2 = 10^2 = 100 = 30 \bmod 35$
  - $5^{20} = (5^{10})^2 = 30^2 = 900 = 25 \bmod 35$

Square-and-Multiply

- Compute X^{26}.
- Binary representation of 26 = (1,1,0,1,0)

Step		Binary exponent	Op	Comment
1	$x = x^1$	$(1)_2$		Initial setting, h_4 processed
1a	$(x^1)^2 = x^2$	$(10)_2$	SQ	Processing h_3
1b	$x^2 * x = x^3$	$(11)_2$	MUL	$h_3 = 1$
2a	$(x^3)^2 = x^6$	$(110)_2$	SQ	Processing h_2
2b	-	$(110)_2$	-	$h_0 = 0$
3a	$(x^6)^2 = x^{12}$	$(1100)_2$	SQ	Processing h_1
3b	$x^{12} * x = x^{13}$	$(1101)_2$	MUL	$h_1=1$
4a	$(x^{13})^2 = x^{26}$	$(11010)_2$	SQ	Processing h_0
4b	-	$(11010)_2$	-	$h_0 = 0$

(source: Understanding of Cryptography)

Fast encryption

- Surprisingly, the very small integers can be chosen for the public key *e*.
- In practice, three values have particular importance.
  - 3(2^1+1), 17(2^4+1), 65537(2^{16}+1)

Fast decryption

- However, small private key *d* cause security weakness.
  - d must have at least 0.3t bits, where t is the bit length of the modulus n.
- The Chinese Remainder Theorem (CRT) can be used to accelerate exponentiation a little.

Protocol Attacks

- RSA encryption is deterministic.
- Plaintexts M=0, 1, -1 produce ciphertext C=0, 1, -1.
- RSA is malleable, which means attacker can manipulate the plaintext in a predictable way.
- Countermeasure : padding
  - Embeds (concatenate) a random structure into a plaintext before encryption.

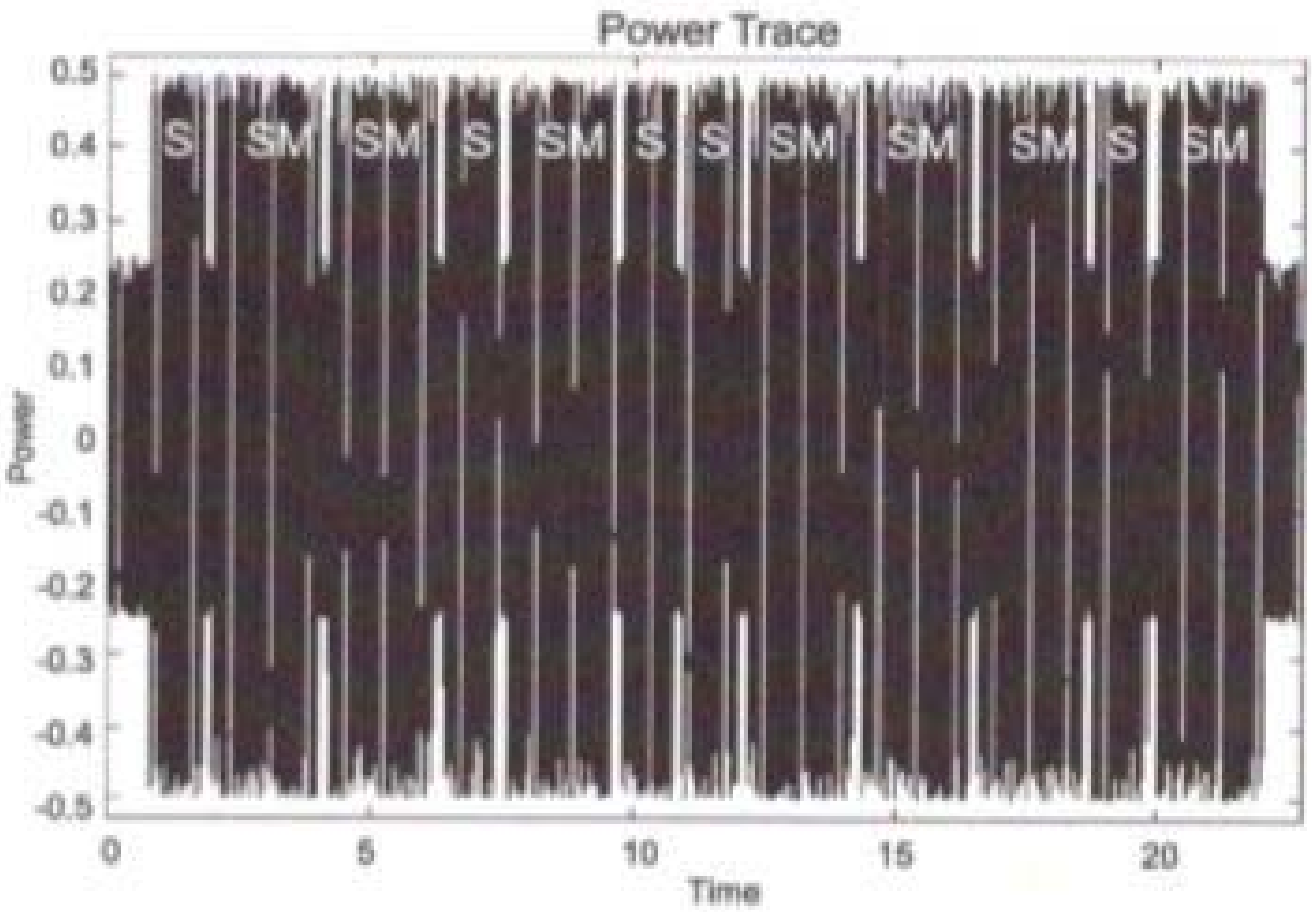
Mathematical Attacks

- Factor $\varphi(n)=(p-1)(q-1)$
- It can be prevented by using a sufficiently large n.
  - Currently factoring the number with n=664 bits are recorded. So, n should be at least 1024 bits, and recommended more than that (2048-4096 bits) for long term safety.

Implementation Attacks

- Side-channel analysis
  - When an attacker can have direct access to the RSA implementation, he can obtain some information about a private key which can be leaked though physical channels such as power consumption or EM emanation.
  - One example is that power rate shows the bit information of private key d during computation of square-and-multiply algorithm.

Ex, power trace of an RSA implimentation



(source: Understanding of Cryptography)

Looking back RSA

- Currently RSA is the most widely used public crypto.
- Main uses are digital signature and key exchange.
- Currently 1024bits cannot be factored, but 2048 to 3076 bits are highly recommended for long-term security.
- Ingenuous implementation exposes several attacks. Meticulous implementation is required.

Encrypting Large File with RSA?

- time of 1024-bit RSA encryption
  - ~1 ms on 1 GHz Pentium
- time of 1024-bit RSA decryption
  - ~10 ms on 1 GHz Pentium
- time to encrypt 1 Mbyte file?
  - Encrypt 1024 bits / RSA operation = 128 bytes
  - 1 Mbyte = 2^{20} bytes
  - Time: $2^{20} / 2^{7} * 1$ms = 2^{13} ms = 8 seconds!
  - Compare with the time by the symmetric key?

Symmetric-key vs. public-key

- Symmetric crypto
  - 80 bit key for high security (year 2010)
  - ~1,000,000 ops/s on 1GHz processor
  - 10x speedup in HW
- Public-key crypto
  - 2048 bit key (RSA) for high security (year 2010)
  - ~100 signatures/s
    ~1000 verify/s (RSA) on 1GHz processor
  - Limited speedup in HW