Lecture 12: Public Key Cryptography Part 1

COSC362 Data and Network Security

Book 1: Chapters 9 and 10 – Book 2: Chapters 2 and 21

Spring Semester, 2021

Motivation

- Public key cryptography (PKC) has features that symmetric key cryptography does not have.
- Applied for key management in protocols such as TLS and IPsec.
- RSA is one of the best known public key cryptosystems, widely deployed in practice.
- Alternatives include discrete log based ciphers, also widely deployed and standardised.

Outline

Public Key Cryptography

RSA Algorithms

RSA Implementation

RSA Security

Outline

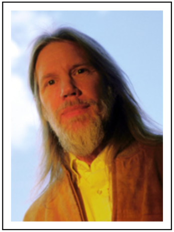
One-Way Functions

- A function f is *one-way* if $f(x) = y$ is easily computed given x, but $f^{-1}(y) = x$ is (computationally) hard to compute given y.
- Open problem: Do one-way functions actually exist?
- Examples of functions believed to be one-way:
  - Multiplication of large primes: the inverse function is integer factorisation.
  - Exponentiation: the inverse function takes discrete logarithms.

Trapdoor One-Way Functions

- A *trapdoor one-way* function f is a one-way function s.t. $f^{-1}(y)$ is easily computed given additional information, called *trapdoor*.
- Example:
  - Modular squaring: given $n = pq$ where p, q are 2 large primes, $f(x) = x^2 \mod n$.
  - If an algorithm takes square roots (i.e. computes f^{-1}) then it can be used to factorise n.
  - The trapdoor is the factorisation of n.
  - If the trapdoor is known then an efficient algorithm finds square roots.

Ciphers Based on Computationally Hard Problems



- Diffie and Hellman published *New Directions in Cryptography* (1976).
- Computational complexity applied in design of encryption algorithms.
- A public key cryptosystem designed by using a trapdoor one-way function.
- Trapdoor is the decryption key.

Also Known as Asymmetric Cryptography

- Asymmetry: encryption and decryption keys are different.
- Encryption key is a *public* key, known to anybody.
- Decryption key is a *private* key, known ONLY to its owner.
- Finding the private key from the knowledge of the public key MUST be a hard computational problem.

Why Public Key Cryptography?

Advantages (in comparison to shared key/symmetric cryptography):

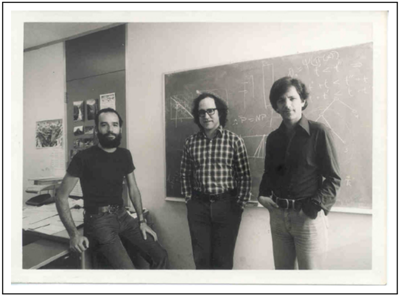
- Key management is simplified:
  - keys do not need to be transported confidentially
- Digital signatures can be obtained.

In Practice

- In a public cipher, encryption keys can be made public.
- Alice stores her public key in a public directory:
  - Anyone can obtain her public key and use it to form an encrypted message to Alice.
  - Since Alice has the private key (associated with her public key), she can decrypt and recover the message.

Outline

Introduction



- Rivest, Shamir and Adleman from MIT in 1977.
- Public key cryptosystem and digital signature scheme.
- Based on integer factorisation problem.
- RSA patent expired in 2000.

Key Generation

Key Generation:

- Randomly choose 2 distinct primes p, q from the set of all primes of a certain size.
- Compute $n = pq$.
- Randomly choose e s.t. $\gcd(e, \phi(n)) = 1$:
  - ϕ is the Euler function.
  - Here, $\phi(n) = \phi(pq) = (p-1)(q-1)$.
- Compute $d = e^{-1} \mod \phi(n)$.
- Set the public key K_E as (n, e).
- Set the private key K_D as (p, q, d).

Encryption and Decryption

Encryption:

- Public encryption key is $K_E = (n, e)$.
- Input is a value M s.t. $0 < M < n$.
- Compute $C = Enc(M, K_E) = M^e \mod n$.

Decryption:

- Private decryption key is $K_D = (p, q, d)$:
  - Note that p, q are not used here.
- Compute $Dec(C, K_D) = C^d \mod n = M$.

Any message requires to be pre-processed to become M:

- Coding it as a number
- Adding randomness

Numerical Example

Key generation:

- Let $p = 43$ and $q = 59$:
  - $n = pq = 2537$
  - $\phi(n) = (p-1)(q-1) = 2436$
- Let $e = 5$:
  - $d = e^{-1} \mod \phi(n) = 5^{-1} \mod 2436 = 1949$
  - Solving $ed + k'\phi(n) = 1$ using the Euclidean algorithm (unknowns are d and the integer k')

Encryption:

- $M = 50$, thus $C = M^e \mod n = 50^5 \mod 2537 = 2488$.

Decryption:

- $C^d \mod n = 2488^{1949} \mod 2537 = 50 = M$.

Encryption Correctness

Does encryption followed by decryption get back where we started from?

$$(M^e)^d \mod n = M \ ?$$

- $d = e^{-1} \mod \phi(n)$, thus $ed \mod \phi(n) = 1$:
  - there is some integer k s.t. $ed = 1 + k\phi(n)$
- $(M^e)^d \mod n = M^{ed} \mod n = M^{1+k\phi(n)} \mod n$.

To complete the proof, we need to show:

$$M^{1+k\phi(n)} \mod n = M \ (1)$$

Proving Equation (1)

Case 1: assuming $\gcd(M, n) = 1$.
Applying Euler's theorem directly to get:

- $M^{\phi(n)} \mod n = 1$

$$\begin{aligned} M^{1+k\phi(n)} \mod n &= M \times (M^{\phi(n)})^k \mod n \\ &= M \times (1)^k \mod n \\ &= M \end{aligned}$$

Proving Equation (1)

Case 2: assuming $\gcd(M, n) \neq 1$.
Remember that $n = pq$ where p, q are primes, and $M < n$:

- Thus either $\gcd(M, p) = 1$ or $\gcd(M, q) = 1$.

Supposing $\gcd(M, p) = 1$ (and the other case is similar):

- $\gcd(M, q) = q$, thus there exists some integer l s.t. $M = lq$

Applying Fermat's theorem to get:

- $M^{\phi(p)} \mod p = M^{p-1} \mod p = 1$

$$
\begin{aligned}
M^{1+k\phi(n)} \mod p &= M \times (M^{\phi(n)})^k \mod p \\
&= M \times (M^{p-1})^{(q-1)k} \mod p \\
&= M \times (1)^{(q-1)k} \mod p \\
&= M \mod p \quad (2)
\end{aligned}
$$

Proving Equation (1)

Case 2 (continued):

Since $M = lq$, it follows that $M^{1+k\phi(n)} \mod q = 0$ (3).

Applying the Chinese Remainder Theorem (CRT):

- It is possible since $n = pq$ for p, q primes.
- There is a unique solution $x = M^{1+k\phi(n)} \mod n$ to equations (2) and (3).
- The solution $x = M$ satisfies (2) and (3), and it is the unique solution for $M^{1+k\phi(n)} \mod n$:
  - $M = M^{1+k\phi(n)} \mod p$
  - $M = M^{1+k\phi(n)} \mod q \ (= 0)$
- Equation (1) is satisfied too.

Applications

- Message encryption
- Digital signature
- Distribution of a shared key for symmetric key encryption (hybrid encryption)
- User authentication by proving knowledge of the private key corresponding to an authenticated public key

Outline

Public Key Cryptography

RSA Algorithms

RSA Implementation

RSA Security

Evolution

Optimisations in RSA implementation have been widely studied:

- Key generation:
  - Generating large primes p, q
  - Choice of e
- Encryption and decryption:
  - Fast exponentiation
  - Faster decryption using CRT
- Data formatting:
  - Padding

Generating Large Primes

- Primes p, q should be random of a chosen length:
  - Today, the recommended one is at least 1024 bits.
- Simple algorithm:
  1. Select a random odd number r of the required length.
  2. Check whether r is prime:
     - If so, then output r and halt.
     - Otherwise, increment r by 2 and go to Step 2.
- Fast way to check for primality (e.g. Miller-Rabin test).

Choice of e

- Public exponent e should be chosen at random for best security.
- A small value is often used in practice:
    - It has a large effect on efficiency.
    - $e = 3$ is the smallest possible value and sometimes used (but security problems!).
    - $e = 2^{16} + 1$ is a popular choice.
- A smaller than average value for private exponent d is also possible:
    - But at least $\sqrt{n}$ to avoid known attacks.

Fast Exponentiation

- Using *square-and-multiply* modular exponentiation algorithm for encryption and decryption.
- e in binary representation:
  - $e = e_0 2^0 + e_1 2^1 + \cdots + e_k 2^k$, where e_i are bits
- Let M be the message to encrypt:
  - $M^e = M^{e_0} \times (M^2)^{e_1} \times \cdots \times (M^{2^k})^{e_k}$

Square-and-multiply Algorithm

Data: $M, n, e = e_k \dots e_1 e_0$
Result: $M^e \bmod n$
$z \leftarrow 1$;
for $i = 0$ to k **do**
  if $e_i = 1$ **then**
    $z \leftarrow z * M \bmod n$;
  end
  if $i < k$ **then**
    $M \leftarrow M^2 \bmod n$;
  end
end
return z

Cost

- If $2^k \leq e < 2^{k+1}$, then the algorithm uses k squarings:
  - If b of e_i bits are '1', then the algorithm uses $b - 1$ multiplications.
  - 1st computation $z \leftarrow z * M$ is not counted because $z = 1$.
- n is a 2048-bit modulus and so e is of at most 2048 bits.
- Computing $M^e \mod n$ requires at most:
  - 2048 modular squarings
  - 2048 modular multiplications
- On average, only half of bits e_i are '1':
  - Only 1024 multiplications
- Reducing modulo n after every operation!

Faster Decryption Using CRT

Using CRT to decrypt C w.r.t. p, q separately:

- Compute $M_p = C^{d \bmod (p-1)} \bmod p$ and $M_q = C^{d \bmod (q-1)} \bmod q$.
- Solve $M \bmod n$ using CRT:
  - $d = \big(d \bmod (p-1)\big) + k(p-1)$ for some k:

    $$\begin{aligned} M \bmod p &= C^{d \bmod n} \bmod p = C^d \bmod p \\ &= C^{d \bmod (p-1)} C^{k(p-1)} \bmod p = C^{d \bmod (p-1)} \\ &= M_p \end{aligned}$$

    Thus $M \equiv M_p \bmod p$.
  - Similarly, $M \equiv M_q \bmod q$.
- Then, output $M = q \times (q^{-1} \bmod p) \times M_p + p \times (p^{-1} \bmod q) \times M_q \bmod n$ (see slide 5 of Lecture 10).

Example

See previous example:

- $p = 43$, $q = 59$, and so modulus $n = 43 \times 59 = 2537$
- Ciphertext $C = 2488$ and private exponent $d = 1949$
- $d \mod (p-1) = 1949 \mod 42 = 17$
- $d \mod (q-1) = 1949 \mod 58 = 35$
- $M_p = 2488^{17} \mod 43 = 37^{17} \mod 43 = 7$
- $M_q = 2488^{35} \mod 59 = 16^{35} \mod 59 = 50$
- Using CRT:

$$
\begin{aligned}
M &= q \times (q^{-1} \mod p) \times M_p + \\
&\quad p \times (p^{-1} \mod q) \times M_q \mod n \\
&= 59 \times (59^{-1} \mod 43) \times 7 + \\
&\quad 43 \times (43^{-1} \mod 59) \times 50 \mod 2537 \\
&= 50 \mod 2537
\end{aligned}
$$

How faster is Decryption with CRT?

- Exponents $d \mod (p-1)$ and $d \mod (q-1)$ are about half the length of d.
- Complexity of exponentiation (with square-and-multiply) increases with the cube of the input length:
  - Computing M_p and M_q each uses $1/2^3 = 1/8$ of computation for $M = C^d \mod n$.
- About 4 times less computation:
  - If M_p and M_q can be computed in parallel, then the time is up to 8 times faster.
- Good reason to store p, q with d.

Padding

- Encryption directly on message encoded as a number is a weak cryptosystem, vulnerable to attacks such as:
  - Building up a dictionary of known plaintexts.
  - Guessing the plaintext and checking if it encrypts to the ciphertext.
  - Håstad's attack.
- Padding mechanism must be used to prepare message for encryption:
  - It must include redundancy and randomness.

Håstad's Attack

- The SAME message M is encrypted without padding to 3 different ciphertexts C_1, C_2, C_3.
- Public exponent $e = 3$ used by ALL recipients.
- Cryptanalysis:

$$\begin{aligned} C_1 &= M^3 \mod n_1 \\ C_2 &= M^3 \mod n_2 \\ C_3 &= M^3 \mod n_3 \end{aligned}$$

  Equations solved using CRT to obtain M^3 in the ordinary (non-modular) integers.
- M found by taking a cube root.

Padding Types

- PKCS #1: simple, ad-hoc design for encryption and digital signature.
- Optimal asymmetric encryption padding (OAEP):
  - Designed by Bellare and Rogaway (1994).
  - Security proof in a suitable model.
  - Standard: IEEE P1363 Standard specifications for public key cryptography.

Outline

Attacks

Most of existing attacks avoided by using standardised padding mechanisms.

- Factorisation of the modulus n:
  - Factorisation is believed to be a hard problem.
  - Factorisation can be prevented by choosing n large enough.
- Finding d from n and e:
  - Finding d is as hard for the adversary as factorising the modulus n.

Equivalence with Factorisation Problem

- An attacker factorises n into its prime factors p, q, and thus recover d:
  - Breaking RSA is not harder than the factorisation problem.
- Breaking RSA is shown to be as hard as the RSA problem:
  - It is unknown if RSA problem is as hard as the factorisation problem.
  - It is also unknown if factorisation is really computationally hard!

Finding d without factorising the modulus n? No!
Miller's theorem: determining d from e, n is as hard as factorising n.

Other Attacks

- Quantum computers: not existing yet (at least commercially):
  - Shor's theoretical algorithm can factorise n in polynomial time.
- Timing analysis: using timing of decryption process to obtain information about d:
  - Demonstrated in practice for RSA in smart cards.
  - Avoided by randomising decryption process.

Practical Problems with Key Generation

- Implementation of OpenSSL in Debian-based Linux system used massively reduced randomness for RSA key generation (2008).
- Lenstra and others published a study of over 6 million RSA keys deployed on the Internet (2012):
  - 270,000 keys (4%) were identical.
  - 12,934 keys (0.2%) provide no security because sharing one prime factor with each other.
  - Certainly due to poor random number generation.