Public Key Cryptography

CSS441: Security and Cryptography

Sirindhorn International Institute of Technology
Thammasat University

Prepared by Steven Gordon on 20 December 2015
css441y15s2l07, Steve/Courses/2015/s2/css441/lectures/public-key-cryptography.tex, r4295

Contents

Birth of Public-Key Cryptosystems

- Beginning to 1960's: permutations and substitutions (Caesar, rotor machines, DES, . . .)
- 1960's: NSA secretly discovered public-key cryptography
- 1970: first known (secret) report on public-key cryptography by CESG, UK
- 1976: Diffie and Hellman public introduction to public-key cryptography
  - Avoid reliance on third-parties for key distribution
  - Allow digital signatures

Principles of Public-Key Cryptosystems

- Symmetric algorithms used same secret key for encryption and decryption
- Asymmetric algorithms in public-key cryptography use one key for encryption and different but related key for decryption
- Characteristics of asymmetric algorithms:
  - Require: Computationally infeasible to determine decryption key given only algorithm and encryption key
  - Optional: Either of two related keys can be used for encryption, with other used for decryption

Public and Private Keys

Public-Private Key Pair

- User A has pair of related keys, public and private: (PU_A, PR_A); similar for other users

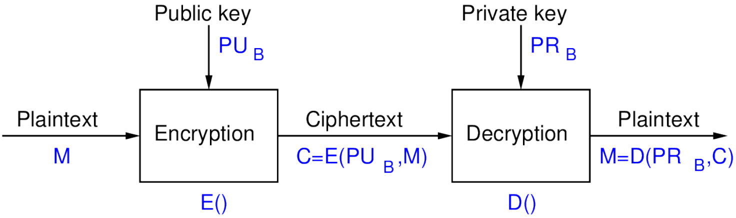
Public Key

- Public, Available to anyone
- For secrecy: used in encryption
- For authentication: used in decryption

Private Key

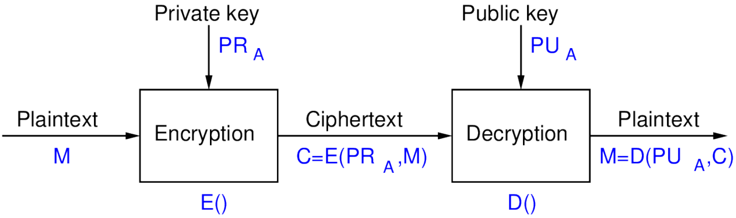
- Secret, known only by owner
- For secrecy: used in decryption
- For authentication: used in decryption

Confidentiality with Public Key Crypto



- Encrypt using receivers public key
- Decrypt using receivers private key
- Only the person with private key can successful decrypt

Authentication with Public Key Crypto



- Encrypt using senders private key
- Decrypt using senders public key
- Only the person with private key could have encrypted

Conventional vs Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

Credit: Table 9.2 in Stallings, *Cryptography and Network Security*, 5th Ed., Pearson 2011

Applications of Public Key Cryptosystems

- Secrecy, encryption/decryption of messages
- Digital signature, *sign* message with private key
- Key exchange, share secret session keys

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Credit: Table 9.3 in Stallings, *Cryptography and Network Security*, 5th Ed., Pearson 2011

Requirements of Public-Key Cryptography

1. Computationally easy for B to generate pair (PU_b, PR_b)
2. Computationally easy for A, knowing PU_b and message M, to generate ciphertext:

$$C = \mathrm{E}(PU_b, M)$$

3. Computationally easy for B to decrypt ciphertext using PR_b:

$$M = \mathrm{D}(PR_b, C) = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

4. Computationally infeasible for attacker, knowing PU_b and C, to determine PR_b
5. Computationally infeasible for attacker, knowing PU_b and C, to determine M
6. (Optional) Two keys can be applied in either order:

$$M = \mathrm{D}[PU_b, \mathrm{E}(PR_b, M)] = \mathrm{D}[PR_b, \mathrm{E}(PU_b, M)]$$

Requirements of Public-Key Cryptography

6 requirements lead to need for trap-door one-way function

- Every function value has unique inverse
- Calculation of function is easy
- Calculation of inverse is infeasible, unless certain information is known

$$
\begin{array}{ll}
Y = f_k(X) & \text{easy, if } k \text{ and } Y \text{ are known} \\
X = f_k^{-1}(Y) & \text{easy, if } k \text{ and } Y \text{ are known} \\
X = f_k^{-1}(Y) & \text{infeasible, if } Y \text{ is known but } k \text{ is not}
\end{array}
$$

- What is easy? What is infeasible?
  - Computational complexity of algorithm gives an indication
  - Easy if can be solved in polynomial time as function of input

Public-Key Cryptanalysis

Brute Force Attacks

- Use large key to avoid brute force attacks
- Public key algorithms less efficient with larger keys
- Public-key cryptography mainly used for key management and signatures

Compute Private Key from Public Key

- No known feasible methods using standard computing

Probable-Message Attack

- Encrypt all possible M' using PU_b—for the C' that matches C, attacker knows M
- Only feasible of M is short
- Solution for short messages: append random bits to make it longer

Contents

RSA

- Ron Rivest, Adi Shamir and Len Adleman
- Created in 1978; RSA Security sells related products
- Most widely used public-key algorithm
- Block cipher: plaintext and ciphertext are integers

The RSA Algorithm

Key Generation

1. Choose primes p and q, and calculate $n = pq$
2. Select e: $gcd(\phi(n), e) = 1, 1 < e < \phi(n)$
3. Find $d \equiv e^{-1} \pmod{\phi(n)}$

$PU = \{e, n\}$, $PR = \{d, n\}$, p and q also private

Encryption

Encryption of plaintext M, where $M < n$:

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

Decryption

Decryption of ciphertext C:

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

Requirements of the RSA Algorithm

1. Possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$
2. Easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$
3. Infeasible to determine d given e and n

- Requirement 1 met if e and d are relatively prime
- Choose primes p and q, and calculate:

$$n = pq$$
$$1 < e < \phi(n)$$
$$ed \equiv 1 \pmod{\phi(n)} \text{ or } d \equiv e^{-1} \pmod{\phi(n)}$$

- n and e are public; p, q and d are private

Example of RSA Algorithm

RSA Implementation Example

- Encryption:

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

- Decryption:

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

- Modulus, n of length b bits
- Public exponent, e
- Private exponent, d
- Prime1, p, and Prime2, q
- Exponent1, $d_p = d \pmod{p-1}$
- Exponent2, $d_q = d \pmod{q-1}$
- Coefficient, $q_{inv} = q^{-1} \pmod{p}$
- Private values: $\{n, e, d, p, q, d_p, d_q, q_{inv}\}$
- Public values: $\{n, e\}$

Computational Efficiency of RSA

- Encryption and decryption require exponentiation
  - Very large numbers; using properties of modular arithmetic makes it easier:

$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

- Choosing e
  - Values such as 3, 17 and 65537 are popular: make exponentiation faster
  - Small e vulnerable to attack: add random padding to each M
- Choosing d
  - Small d vulnerable to attack
  - Decryption using large d made faster using Chinese Remainder Theorem and Fermat's Theorem
- Choosing p and q
  - p and q must be very large primes
  - Choose random odd number and test if its prime (probabilistic test)

Security of RSA

- Brute-Force attack: choose large d (but makes algorithm slower)
- Mathematical attacks:
  1. Factor n into its two prime factors
  2. Determine $\phi(n)$ directly, without determining p or q
  3. Determine d directly, without determining $\phi(n)$

  - Factoring n is considered fastest approach; hence used as measure of RSA security
- Timing attacks: practical, but countermeasures easy to add (e.g. random delay). 2 to 10% performance penalty
- Chosen ciphertext attack: countermeasure is to use padding (Optimal Asymmetric Encryption Padding)

Progress in Factorisation

- Factoring is considered the easiest attack
- Some records by length of n:
  - 1991: 330 bits (100 digits)
  - 2003: 576 bits (174 digits)
  - 2005: 640 bits (193 digits)
  - 2009: 768 bit (232 digits), 10^{20} operations, 2000 years on single core 2.2 GHz computer
- Typical length of n: 1024 bits, 2048 bits, 4096 bits

Contents

Diffie-Hellman Key Exchange

- Diffie and Hellman proposed public key crypto-system in 1976
- Algorithm for exchanging secret key (not for secrecy of data)
- Based on discrete logarithms
- Easy to calculate exponential modulo a prime
- Infeasible to calculate inverse, i.e. discrete logarithm

Diffie-Hellman Key Exchange Algorithm

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

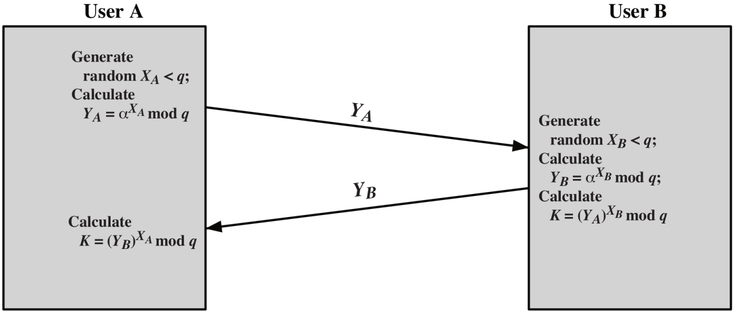
$K = (Y_B)^{X_A} \bmod q$

Calculation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Credit: Figure 10.1 in Stallings, *Cryptography and Network Security*, 5th Ed., Pearson 2011

Diffie-Hellman Key Exchange



Credit: Figure 10.2.2 in Stallings, *Cryptography and Network Security*, 5th Ed., Pearson 2011

Diffie-Hellman Key Exchange Example

Security of Diffie-Hellman Key Exchange

- Insecure against man-in-the-middle-attack
- Countermeasure is to use digital signatures and public-key certificates

Contents

Other Public-Key Cryptosystems

ElGamal Crypto-system

- Similar concepts to Diffie-Hellman
- Used in Digital Signature Standard and secure email

Elliptic Curve Cryptography

- Uses elliptic curve arithmetic (instead of modular arithmetic in RSA)
- Equivalent security to RSA with smaller keys (better performance)
- Used for key exchange and digital signatures