

ENEE 459-C

Computer Security

**Public key Encryption:
RSA and ElGamal**



UNIVERSITY OF
MARYLAND

Review of Secret Key (Symmetric) Cryptography

- Confidentiality
 - block ciphers with encryption modes
- Integrity
 - Message authentication code (keyed hash functions)
- Limitation: sender and receiver must share the same key
 - Needs secure channel for key distribution
 - Impossible for two parties having no prior relationship
 - Needs many keys for n parties to communicate

Concept of Public Key Encryption

- Each party has a pair (K, K^{-1}) of keys:
 - K is the **public** key, and used for encryption
 - K^{-1} is the **private** key, and used for decryption
 - Satisfies $D_{K^{-1}}[E_K[M]] = M$
- Knowing the public-key K , it is computationally infeasible to compute the private key K^{-1}
 - **Easy to check K, K^{-1} is a pair**
- The public-key K may be made publicly available, e.g., in a publicly available directory
 - Many can encrypt, only one can decrypt
- Public-key systems aka *asymmetric* crypto systems

Public Key Cryptography Early History

- Proposed by Diffie and Hellman, documented in “New Directions in Cryptography” (1976)
 1. Public-key encryption schemes
 2. Key distribution systems
 - Diffie-Hellman key agreement protocol
 3. Digital signature
- Public-key encryption was proposed in 1970 in a classified paper by James Ellis
 - paper made public in 1997 by the British Governmental Communications Headquarters
- Concept of digital signature is still originally due to Diffie & Hellman

Public Key Encryption Algorithms

- Almost all public-key encryption algorithms use either number theory and modular arithmetic, or elliptic curves
- RSA
 - based on the hardness of factoring large numbers
- El Gamal
 - Based on the hardness of solving discrete logarithm
 - Use the same idea as Diffie-Hellman key agreement

RSA Cryptosystem

■ Setup:

- $n = pq$, with p and q primes
- e relatively prime to $\phi(n) = (p - 1)(q - 1)$
- d inverse of e in $Z_{\phi(n)}$

■ Keys:

- Public key: $K_E = (n, e)$
- Private key: $K_D = d$

■ Encryption:

- Plaintext M in Z_n
- $C = M^e \bmod n$

■ Decryption:

- $M = C^d \bmod n$

■ Example

■ Setup:

- ♦ $p = 7, q = 17$
- ♦ $n = 7 \cdot 17 = 119$
- ♦ $\phi(n) = 6 \cdot 16 = 96$
- ♦ $e = 5$
- ♦ $d = 77$

■ Keys:

- ♦ public key: (119, 5)
- ♦ private key: 77

■ Encryption:

- ♦ $M = 19$
- ♦ $C = 19^5 \bmod 119 = 66$

■ Decryption:

- ♦ $C = 66^{77} \bmod 119 = 19$

Complete RSA Example

- Setup:

- $p = 5, q = 11$
- $n = 5 \cdot 11 = 55$
- $\phi(n) = 4 \cdot 10 = 40$
- $e = 3$
- $d = 27$ ($3 \cdot 27 = 81 = 2 \cdot 40 + 1$)

- Encryption

- $C = M^3 \pmod{55}$

- Decryption

- $M = C^{27} \pmod{55}$

| | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| M | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| C | 1 | 8 | 27 | 9 | 15 | 51 | 13 | 17 | 14 | 10 | 11 | 23 | 52 | 49 | 20 | 26 | 18 | 2 |
| M | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| C | 39 | 25 | 21 | 33 | 12 | 19 | 5 | 31 | 48 | 7 | 24 | 50 | 36 | 43 | 22 | 34 | 30 | 16 |
| M | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 |
| C | 53 | 37 | 29 | 35 | 6 | 3 | 32 | 44 | 45 | 41 | 38 | 42 | 4 | 40 | 46 | 28 | 47 | 54 |

Questions 1

- In the previous example, why encrypting small messages, e.g., $M=2,3,4$ is tricky?
 - Recall $\text{Enc}(M) = M^3 \pmod{55}$
- Let $N=20434394384355534343545428943483434356091$. Assume it is the product of two primes
 - Can e be the number 4343253453434536?

Questions 2

- Problem with RSA?
- Does it satisfy semantic security?

Security of RSA

- Security of RSA based on difficulty of factoring
 - Widely believed
 - Best known algorithm takes exponential time
- How can you break RSA if you can factor?
- In 1999, 512-bit challenge factored in 4 months using 35.7 CPU-years
 - 160 175-400 MHz SGI and Sun
 - 8 250 MHz SGI Origin
 - 120 300-450 MHz Pentium II
 - 4 500 MHz Digital/Compaq
- In 2005, a team of researchers factored the RSA-640 challenge number using 30 2.2GHz CPU years
- In 2004, the prize for factoring RSA-2048 was \$200,000
- Current practice is 2,048-bit keys
- Estimated resources needed to factor a number within one year
- Note that for RSA to be secure, both p and q must be large primes

| Length (bits) | PCs | Memory |
|---------------|----------------------|--------|
| 430 | 1 | 128MB |
| 760 | 215,000 | 4GB |
| 1,020 | 342×10^6 | 170GB |
| 1,620 | 1.6×10^{15} | 120TB |

Correctness

- We show the correctness of the RSA cryptosystem for the case when the plaintext M does not divide n

- Namely, we show that

$$(M^e)^d \bmod n = M$$

- Since $ed \bmod \phi(n) = 1$, there is an integer k such that

$$ed = k\phi(n) + 1$$

- Since M does not divide n , by Euler's theorem we have

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

- Thus, we obtain

$$(M^e)^d \bmod n =$$

$$M^{ed} \bmod n =$$

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

$$MM^{k\phi(n)} \bmod n =$$

$$M (M^{\phi(n)})^k \bmod n =$$

$$M (M^{\phi(n)} \bmod n)^k \bmod n =$$

$$M (1)^k \bmod n =$$

$$M \bmod n =$$

$$M$$

- Proof of correctness can be extended to the case when the plaintext M divides n

Algorithmic Issues

- The implementation of the RSA cryptosystem requires various algorithms
- Overall
 - Representation of integers of arbitrarily large size and arithmetic operations on them
- Encryption
 - Modular power
- Decryption
 - Modular power
- Setup
 - Generation of random numbers with a given number of bits (to generate candidates p and q)
 - Primality testing (to check that candidates p and q are prime)
 - Computation of the GCD (to verify that e and $\phi(n)$ are relatively prime)
 - Computation of the multiplicative inverse (to compute d from e)

Modular Power

- The repeated squaring algorithm speeds up the computation of a modular power $a^p \bmod n$
- Write the exponent p in binary

$$p = p_{b-1}p_{b-2} \cdots p_1p_0$$
- Start with

$$Q_1 = a^{p_{b-1}} \bmod n$$
- Repeatedly compute

$$Q_i = ((Q_{i-1})^2 \bmod n) a^{p_{b-i}} \bmod n$$
- We obtain

$$Q_b = a^p \bmod n$$
- The repeated squaring algorithm performs $O(\log p)$ arithmetic operations

Example

- $3^{18} \bmod 19$ ($18 = 10010$)
- $Q_1 = 3^1 \bmod 19 = 3$
- $Q_2 = (3^2 \bmod 19)3^0 \bmod 19 = 9$
- $Q_3 = (9^2 \bmod 19)3^0 \bmod 19 = 81 \bmod 19 = 5$
- $Q_4 = (5^2 \bmod 19)3^1 \bmod 19 = (25 \bmod 19)3 \bmod 19 = 18 \bmod 19 = 18$
- $Q_5 = (18^2 \bmod 19)3^0 \bmod 19 = (324 \bmod 19) \bmod 19 = 17 \cdot 19 + 1 \bmod 19 = 1$

| | | | | | |
|---------------|---|---|---|----|---|
| p_{5-i} | 1 | 0 | 0 | 1 | 0 |
| $2^{p_{5-i}}$ | 3 | 1 | 1 | 3 | 1 |
| Q_i | 3 | 9 | 5 | 18 | 1 |

Chinese remainder theorem light

- Let $N=pq$. Let
 - $x \bmod p = a_1$
 - $x \bmod q = a_2$
- Then
 - $x \bmod N = a_1 * q * \text{inverse}(q \text{ in } Z_p) + a_2 * p * \text{inverse}(p \text{ in } Z_q) \bmod N$
 - **Let's prove it**
 - This can be used to compute $W^x \bmod N$, for big W^x , more efficiently
 - How?
- Use of theorem
 - Say you want to compute $18^{25} \bmod 35$ ($35 = 5*7$)
 - Compute $18^{25} \bmod 5 = 18^{(25 \bmod 4)} \bmod 5 = 18^1 \bmod 5 = 3 = a_1$
 - Compute $18^{25} \bmod 7 = 18^{(25 \bmod 6)} \bmod 7 = 18^1 \bmod 7 = 4 = a_2$
 - Note that $\text{inverse}(5 \text{ in } Z_7)=3$ and $\text{inverse}(7 \text{ in } Z_5)=3$
 - Therefore the solution we are looking for is $3*7*3+4*5*3 \bmod 35= 18$
- Used in the decryption procedure of RSA: Why cannot it be used in the encryption?
- Also we can prove correctness of RSA for general message M

Pseudoprimality Testing

- Testing whether a number is prime (primality testing) is a difficult problem, though polynomial-time algorithms exist
- An integer $n \geq 2$ is said to be a base- x pseudoprime if
 - $x^{n-1} \bmod n = 1$ (Fermat's little theorem)
- Composite base- x pseudoprimes are rare:
 - A random 100-bit integer is a composite base-2 pseudoprime with probability less than 10^{-13}
 - The smallest composite base-2 pseudoprime is 341
- Base- x pseudoprimality testing for an integer n :
 - Check whether $x^{n-1} \bmod n = 1$
 - Can be performed efficiently with the repeated squaring algorithm

RSA security and properties

- Plain RSA is deterministic.
- Why is this a problem?
- Plain RSA is also homomorphic. What does this mean?
 - Multiply ciphertexts to get ciphertext of multiplication!
 - $[(m_1)^e \bmod N][(m_2)^e \bmod N] = (m_1 m_2)^e \bmod N$
- However, not additively homomorphic
- Both additively + multiplicative homomorphic (aka fully-homomorphic) encryption open problem till 2009
- A breakthrough result from IBM (Craig Gentry) answered this open problem, constructing such an encryption scheme

Real World Usage of RSA

- Randomized RSA
 - To encrypt a message M under an RSA public key (n,e) , generate a new random session AES key K , compute the cipher text as
 - $[K^e \bmod N, \text{AES}_K(M)]$
- This prevents an adversary distinguishing two encryptions of the same message since K is chosen at random every time the encryption takes place
- Could the following encryption work for arbitrary messages M ?
 - $(M||r)^e \bmod N$, for random r

ElGamal Encryption

- Encrypts messages $m \in \mathbb{Z}_p$
- **Secret key:** random number $a \in \mathbb{Z}_p$
- **Public key:** $A = g^a$
- **Encryption:** Pick a random $r \in \mathbb{Z}_p$ and set
 - $R = A^r = g^{ar}$
 - $c_1 = g^r$
 - Then **Enc(m) = (c₁, c₂)** where **c₂ = mR mod p**
 - **Dec(c₁, c₂) = c₂ * (1/c₁^a) mod p** where **c₁^a = g^{ar}**
- Security depends on Computational Diffie-Hellman (CDH) assumption: given (g, g^a, g^b) it is hard to compute g^{ab}
- Do not use same r twice