T-79. 4501
Cryptography and Data Security

Lecture 6: Number Theory
- -Prime numbers
- -Chinese remainder theorem
- -Euler's totient function
- -Euler's theorem

Stallings: Ch 8

Lecture 7: RSA
- -Public Key Principle
- -Setting up RSA
- -Prime number generation
- -RSA encryption and decryption
- -Square and multiply
- -Security of RSA

Stallings: Ch 9.1-2

Prime Numbers

Definition: An integer $p > 1$ is a prime if and only if its only positive integer divisors are 1 and p.

Fact: Any integer $a > 1$ has a unique representation as a product of its prime divisors

$$a = \prod_{i=1}^{t} p_i^{e_i} = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}$$

where $p_1 < p_2 < \ldots < p_t$ and each e_i is a positive integer.

Some first primes: 2,3,5,7,11,13,17,… For more primes, see:

www.utm.edu/research/**primes**/

Example: Composite (non-prime) numbers and their factorisations:
18 = $2 \cdot 3^2$, 27 = 3^3, 42 = $2 \cdot 3 \cdot 7$, 84773093 = $8887 \cdot 9539$

Euclidean Algorithm

Given two positive integers and their representations as products of prime powers, it would be easy to extract from them the maximum set of common prime powers.

For example gcd(18, 42) = gcd($2 \cdot 3^2$, $2 \cdot 3 \cdot 7$) = $2 \cdot 3 = 6$.

On the other hand, given just one (composite) integer, its factorization is hard to compute (in general).

Euclidean Algorithm is an efficient algorithm for finding the gcd of two integers. It is based on the following fact:

Let $a > b$. Then gcd(a,b) = gcd(a mod b, b).

Example: gcd(42, 18) = gcd(6, 18) = 6.

Example: gcd(595,408) = gcd(187,408) = gcd(187,34) = gcd(17,34) = 17.

Slowest case: Fibonacci sequence 1, 2, 3, 5, 8,13, 21,…,$F_n = F_{n-1} + F_{n-2}$. For example it takes 5 iterations to compute gcd(21,13); in general it takes n-2 iterations to compute gcd(F_n,F_{n-1})

Chinese Remainder Theorem (two moduli)

Problem: Assume m_1 and m_2 are coprime. Given x_1 and x_2, how to find $0 \le x < m_1 m_2$ such that

$$x = x_1 \bmod m_1$$

$$x = x_2 \bmod m_2$$

Solution: Use the Extended Euclidean Algorithm to find u and v such that $u \cdot m_1 + v \cdot m_2 = 1$. Then

$$x = x \cdot (u \cdot m_1 + v \cdot m_2) = x \cdot u \cdot m_1 + x \cdot v \cdot m_2$$

$$= (x_2 + r \cdot m_2) \cdot u \cdot m_1 + (x_1 + s \cdot m_1) \cdot v \cdot m_2.$$

It follows that

$$x = x \bmod (m_1 m_2) = (x_2 \cdot u \cdot m_1 + x_1 \cdot v \cdot m_2) \bmod (m_1 m_2)$$

Chinese Remainder Theorem (general case)

Theorem: Assume $m_1, m_2, \ldots, m_t$ are mutually coprime. Denote $M = m_1 \cdot m_2 \cdot \ldots \cdot m_t$. Given $x_1, x_2, \ldots, x_t$ there exists a unique x, $0 \leq x < M$, such that

$$x = x_1 \bmod m_1$$

$$x = x_2 \bmod m_2$$

$$\ldots$$

$$x = x_t \bmod m_t$$

x can be computed as

$$x = (x_1 \cdot u_1 \cdot M_1 + x_2 \cdot u_2 \cdot M_2 + \ldots + x_t \cdot u_t \cdot M_t) \bmod M,$$

where $M_i = (m_1 \cdot m_2 \cdot \ldots \cdot m_t)/m_i$ and $u_i = M_i^{-1} \pmod{m_i}$.

Proof: For the case $t = 2$ see previous page. The general case is handled in T-79.5501.

Chinese Remainder Theorem: Example

Let $m_1 = 7$, $m_2 = 11$, $m_3 = 13$. Then $M = 1001$.

Problem: Compute x, $0 \le x \le 1000$ such that

$x = 5 \bmod 7$

$x = 3 \bmod 11$

$x = 10 \bmod 13$

Solution:

$M_1 = m_2 m_3 = 143$; $M_2 = m_1 m_3 = 91$; $M_3 = m_1 m_2 = 77$

$u_1 = M_1^{-1} \bmod m_1 = 143^{-1} \bmod 7 = 3^{-1} \bmod 7 = 5$; similarly

$u_2 = M_2^{-1} \bmod m_2 = 3^{-1} \bmod 11 = 4$; $u_3 = (-1)^{-1} \bmod 13 = -1$.

Then

$x = (5 \cdot 5 \cdot 143 + 3 \cdot 4 \cdot 91 + 10 \cdot (-1) \cdot 77) \bmod 1001 = 894$

Euler's Totient Function $\phi(n)$

Definition: Let $n > 1$ be integer. Then we set

$$\phi(n) = \#\{\, a \mid 0 < a < n, \gcd(a,n) = 1\},$$

that is, $\phi(n)$ is the number of positive integers less than n which are coprime with n.

For prime p, $\phi(p) = p - 1$. We define $\phi(1) = 1$.

For a prime power p^e, we have $\phi(p^e) = p^{e-1}(p-1)$.

The multiplicative property holds:

$$\phi(m \times n) = \phi(m) \times \phi(n), \text{ for all } m, n, \gcd(m,n) = 1 .$$

Now Euler's totient function can be computed for any integer using its prime factorisation.

Example: $\phi(18) = \phi(2 \times 3^2) = \phi(2) \times \phi(3^2) = (2-1) \times (3-1)3^1 = 6$, that is, the number of invertible (coprime with 18) numbers modulo 18 is equal to 6. They are: $1, 5, 7, 11, 13, 17$.

Euler's Theorem

$$Z_n^* = \{a \mid 0 < a < n, \gcd(a,n) = 1\}, \text{and } \#Z_n^* = \phi(n)$$

Euler's Theorem: For any integers n and a such that $a \neq 0$ and gcd(a,n) = 1 the following holds:

$$a^{\phi(n)} \equiv 1 (\mathrm{mod}\, n)$$

Fermat's Theorem: For a prime p and any integer a such that $a \neq 0$ and a is not a multiple of p the following holds:

$$a^{p-1} \equiv 1 (\mathrm{mod}\, p)$$

Setting up the RSA

- Generate two different odd primes p and q
- Compute $n = pq$ and compute $\phi(n) = (p-1)(q-1)$
- Select a public exponent e such that $\gcd(e, \phi(n)) = 1$
- Using Extended Euclidean Algorithm compute the multiplicative inverse of e modulo $\phi(n)$. Denote $d = e^{-1} \bmod \phi(n)$.

Public key: $K_{\text{pub}} = (n, e)$

Private key: $K_{\text{priv}} = (n, d)$

(or $K_{\text{priv}} = (p, q, d)$. This is needed if private computations make use of the CRT.)

n is called the RSA modulus; e is the public encryption exponent; d is the private decryption exponent.

RSA encryption and decryption

Let M be a message, $0 \le M < n$. Then

encryption of M is $C = M^e \bmod n$

decryption of C is $M = C^d \bmod n$

This works, because $(M^e)^d \bmod n = M$.

Proof. (For $M \in Z_n^*$): By Euler's theorem, $M^{\phi(n)} \equiv 1 (\bmod n)$. On the other hand, $ed \equiv 1 (\bmod \phi(n))$

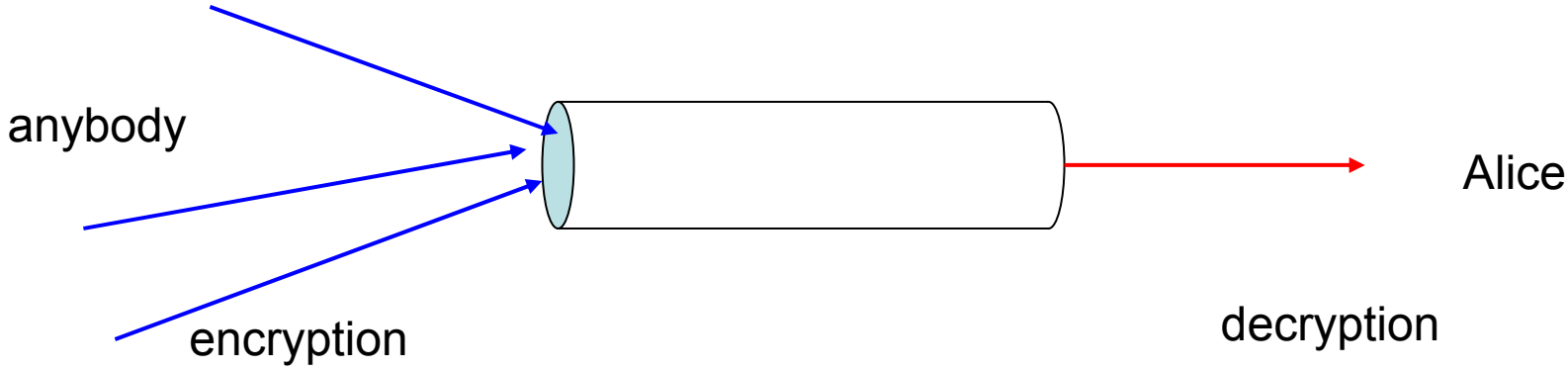
It follows:

$$(M^e)^d = M^{ed} = M^{1+k\phi(n)} = M \cdot (M^{\phi(n)})^k = M (\bmod n)$$

The Principle of Public Key Cryptosystems

Encryption operation is public

Decryption operation is private



Alice's key for a public key cryptosystem is a pair: $(K_{\text{pub}}, K_{\text{priv}})$ where K_{pub} is public and K_{priv} cannot be used by anybody else than Alice.

Generating primes

Prime Number Theorem: Let

$$\pi(n) = \#\{a \mid 1 < a \le n, a \text{ is prime}\}.$$

Then

$$\pi(n) \approx \frac{n}{\ln n}.$$

Hence the probability that a randomly picked m-bit number is a prime is

$$2^{1-m}\left(\pi\left(2^m\right) - \pi\left(2^{m-1}\right)\right),$$

and twice as large if only odd numbers are picked. The number is reasonably large (see exercise for an example).

The primality of a random prime is tested using some primality test: Solovay-Strassen, Miller-Rabin,…, for a deterministic test (not yet practical) see: http://www.cse.iitk.ac.in/news/primality.html

Miller-Rabin Primality test

1. Let $n \geq 3$ be odd, consider the even number n -1, and write it as

$$n-1 = 2^k q, \text{ with } q \text{ odd}$$

2. Select a random integer a, 1< a < n -1.

3. If $a^q \bmod n = 1$ then return: n maybe a prime.

4. For j = 0 to k -1 do

5. if $a^{2^j q} \bmod n = n-1$ then return: n may be a prime

6. Return: n is composite

For a given a the probability that a composite integer n passes the test in at most ¼. By repeating the test for a number of different base numbers a the probability of accepting a composite number as a prime can be made sufficiently small.

Square and multiply

Fast exponentiation algorithms exist. The simplest one is the Square and Multiply Algorithm. It has two versions: left-to-right or right-to-left. Below we show the right-to-left (from the lsb to msb) version.

To compute $a^d \bmod n$

use the binary representation of the exponent d :

$$d = d_0 + d_1 \cdot 2 + d_2 \cdot 2^2 + \cdots + d_{k-1} 2^{k-1}$$

where $d_0, d_1, d_2, \ldots, d_{k-1}$ are bits, i.e. equal to 0 or 1. Now

$$a^d = a^{d_0 + d_1 \cdot 2 + d_2 \cdot 2^2 + \cdots + d_{k-1} 2^{k-1}} = a^{d_0} (a^2)^{d_1} (a^{2^2})^{d_2} \cdots (a^{2^{k-1}})^{d_{k-1}} (\bmod n)$$

Compute the k powers:

$$a = a^{2^0}; a^{2^1}; a^{2^2}; a^{2^3}; \cdots; a^{2^{k-1}} (\bmod n)$$

and from of product (modulo n) of those powers $a^{2^i} (\bmod n)$, for which the corresponding $d_i = 1$.

Using CRT

- The private operation in RSA can be facilitated (for example, to fit small processors) by computing modulo p and modulo q and then combining the results to get the result modulo pq.
- Then the private key must contain the information about p and q , that is, $K_{priv} = (p,q,d)$

Security of RSA

- If factoring of n is easy, then $\phi(n)$ is easy to compute. Given $\phi(n)$ and e, it is easy to compute the private exponent d .

- But even if factoring is hard (as it is believed to be) there may be some other ways to break RSA, without factoring the modulus. But no such break is known. All known breaks can be handled by proper selection of parameters, and message formatting.