Introduction to Cryptography CS 355

Lecture 20

Fast Exponentiation & Pohlig-Hellman Exponentiation Cipher

Lecture Outline

- Why public key cryptography?
- Overview of Public Key Cryptography
- RSA
  – square & multiply algorithm
  – RSA implementation
- Pohlig-Hellman

Why does RSA work?

- Need to show that $(M^e)^d \pmod{n} = M$, $n = pq$
- We have shown that when $M \in Z_{pq}^*$, i.e., $\gcd(M, n) = 1$, then $M^{ed} \equiv M \pmod{n}$
- What if $M \in Z_{pq} - \{0\} - Z_{pq}^*$, e.g., $\gcd(M, n) = p$.
  - $ed \equiv 1 \pmod{\Phi(n)}$, so $ed = k\Phi(n) + 1$, for some integer k.
  - $M^{ed} \bmod p = (M \bmod p)^{ed} \bmod p = 0$
    so $M^{ed} \equiv M \bmod p$
  - $M^{ed} \bmod q = (M^{k*\Phi(n)} \bmod q)\,(M \bmod q) = M \bmod q$
    so $M^{ed} \equiv M \bmod q$
  - As p and q are distinct primes, it follows from the CRT that $M^{ed} \equiv M \bmod pq$

Square and Multiply Algorithm for Exponentiation

- Computing $(x)^c \bmod n$
  - Example: suppose that c=53=110101
  - $x^{53}=(x^{13})^2 \cdot x=(((x^3)^2)^2 \cdot x)^2)^2 \cdot x =(((x^2 \cdot x)^2)^2 \cdot x)^2)^2 \cdot x \mod n$

Alg: Square-and-multiply $(x, n, c = c_{k-1}\ c_{k-2} \ldots c_1\ c_0)$

```
z=1
for i ← k-1 downto 0 {
        z ← z² mod n
        if cᵢ = 1 then z ← (z × x) mod n
}
return z
```

Efficiency of computation modulo n

- Suppose that n is a k-bit number, and $0 \le x,y \le n$
  - computing $(x+y)$ mod n takes time $O(k)$
  - computing $(x-y)$ mod n takes time $O(k)$
  - computing (xy) mod n takes time $O(k^2)$
  - computing (x^{-1}) mod n takes time $O(k^3)$
  - computing $(x)^c$ mod n takes time $O((\log c)\ k^2)$

RSA Implementation

n, p, q

- The security of RSA depends on how large n is, which is often measured in the number of bits for n. Current recommendation is 1024 bits for n.
- p and q should have the same bit length, so for 1024 bits RSA, p and q should be about 512 bits.
- p-q should not be small

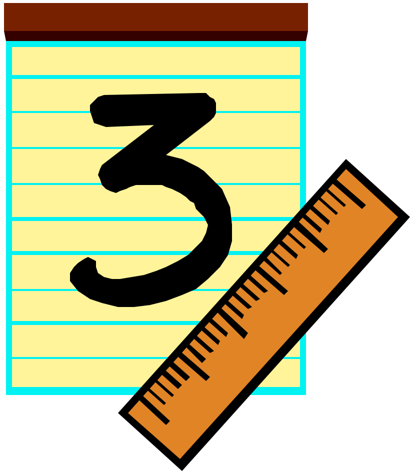
RSA Implementation

- Select p and q prime numbers
- In general, select numbers, then test for primality
- Many implementations use the Rabin-Miller test, (probabilistic test)

RSA Implementation

e

- e is usually chosen to be 3 or $2^{16} + 1 = 65537$
- In order to speed up the encryption
  – the smaller the number of 1 bits, the better
  – why?



Pohlig-Hellman Exponentiation Cipher

- A symmetric key exponentiation cipher
  - encryption key (e,p), where p is a prime
  - decryption key (d,p), where $ed \equiv 1 \pmod{(p-1)}$
  - to encrypt M, compute $M^e \bmod p$
  - to decrypt C, compute $C^d \bmod p$

- Why is this not a public key cipher?
- What makes RSA different?

Distribution of Prime Numbers

Theorem (Gaps between primes)

For every positive integer n, there are n or more consecutive composite numbers.

Proof Idea:

The consective numbers

$$(n+1)! + 2, (n+1)! + 3, \ldots., (n+1)! + n+1$$

are composite.

(Why?)

Distribution of Prime Numbers

Definition

Given real number x, let π(x) be the number of prime numbers = x.

Theorem (prime numbers theorem)

$$\lim_{x \to \infty} \frac{p(x)}{x / \ln x} = 1$$

For a very large number x, the number of prime numbers smaller than x is close to x/ln x.

Generating large prime numbers

- Randomly generate a large odd number and then test whether it is prime.
- How many random integers need to be tested before finding a prime?
  - the number of prime numbers ≤ p is about N / ln p
  - roughly every ln p integers has a prime
    - for a 512 bit p, ln p = 355. on average, need to test about 177=355/2 odd numbers
- Need to solve the Primality testing problem
  - the decision problem to decide whether a number is a prime

Coming Attractions …

- Group
- Quadratic Residues
- Primality Test