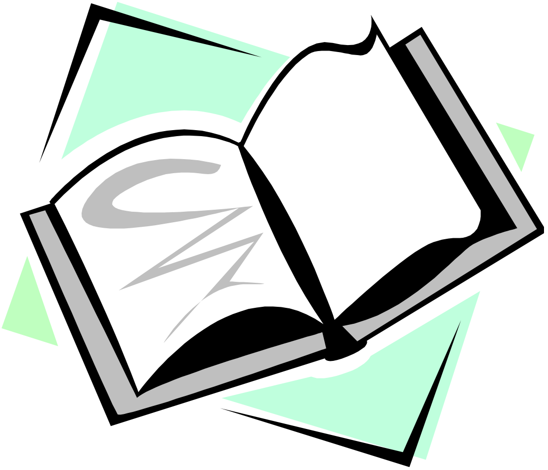
Cryptography
CS 555

Topic 18: RSA Implementation and Security

Outline and Readings

- Outline
  - RSA implementation issues
  - Factoring large numbers
  - Knowing (e,d) enables factoring
  - Prime testing



- Readings:
  - Katz and Lindell: Section 7.2, Appendix B.2

Why does RSA work?

- Need to show that $(M^e)^d \pmod{n} = M$, $n = pq$
- We know that when $M \in Z_{pq}^*$, i.e., when $\gcd(M, n) = 1$, then $M^{ed} \equiv M \quad \pmod{n}$
- What if $\gcd(M, n) \neq 1$?
  - Assume, wlog, that $\gcd(M, n) = p$
  - $ed \equiv 1 \pmod{\Phi(n)}$, 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 Chinese Remainder Theorem that $M^{ed} \equiv M \bmod pq$
- What is the probability that when one chooses $M \in Z_{pq}$, $\gcd(M, n) \neq 1$?

Square and Multiply Algorithm for Exponentiation

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

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 c_i = 1 then z ← (z × x) mod n
}
return z
```

Efficiency of computation modulo n

- Suppose that n is a k-bit number, and $0 \leq x,y \leq n$
  - computing $(x+y) \bmod n$ takes time $O(k)$
  - computing $(x-y) \bmod n$ takes time $O(k)$
  - computing $(xy) \bmod n$ takes time $O(k^2)$
  - computing $(x^{-1}) \bmod n$ takes time $O(k^3)$
  - computing $(x)^c \bmod n$ 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.
- Currently, 1024 bits for n is considered similar to 80-bit security, and is not recommended for serious security
- p and q should have the same bit length, so for 2048 bits RSA, p and q should be about 1024 bits.
- p – q should not be small
  - Otherwise, factoring pq is easy

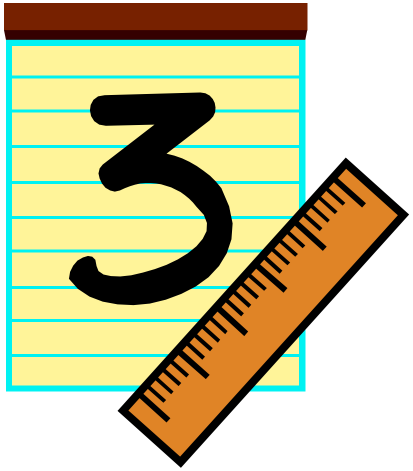
RSA Implementation

- 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?

Factoring Large Numbers

- One idea many factoring algorithms use:
  - Suppose one find $x^2 \equiv y^2 \pmod{n}$ such that $x \neq y \pmod{n}$ and $x \neq -y \pmod{n}$.
  - Then $n \mid (x-y)(x+y)$.
  - As neither $(x-y)$ or $(x+y)$ is divisible by n; $\gcd(x-y,n)$ is a non-trivial factor of n
  - Given one factor, easily compute the other

More Details on Factoring

- Fact: if $n=pq$, then $x^2 \equiv 1 \pmod n$ has four solutions that are $<n$.
  - $x^2 \equiv 1 \pmod n$ if and only if
    both $x^2 \equiv 1 \pmod p$ and $x^2 \equiv 1 \pmod q$
  - Two trivial solutions: 1 and n-1
    - 1 is solution to $x \equiv 1 \pmod p$ and $x \equiv 1 \pmod q$
    - n-1 is solution to $x \equiv -1 \pmod p$ and $x \equiv -1 \pmod q$
  - Two other solutions
    - solution to $x \equiv 1 \pmod p$ and $x \equiv -1 \pmod q$
    - solution to $x \equiv -1 \pmod p$ and $x \equiv 1 \pmod q$
  - E.g., $n=3\times5=15$, then $x^2 \equiv 1 \pmod{15}$ has the following solutions: 1, 4, 11, 14

An Example

- Knowing a nontrivial solution to $x^2 \equiv 1 \pmod{n}$
  - compute gcd(x+1,n) and gcd(x-1,n)
- E.g., 4 and 11 are solution to $x^2 \equiv 1 \pmod{15}$
  - gcd(4+1,15) = 5
  - gcd(4-1,15) = 3
  - gcd(11+1,15) = 3
  - gcd(11-1, 15) = 5

Time complexity of factoring

- quadratic sieve:
  - $O(e^{(1+o(1))\mathrm{sqrt}(\ln n \ln\ln n)})$ for n around 2^{1024}, $O(e^{68})$
- elliptic curve factoring algorithm
  - $O(e^{(1+o(1))\mathrm{sqrt}(2 \ln p \ln\ln p)})$, where p is the smallest prime factor
  - for n=pq and p,q around 2^{512}, for n around 2^{1024} $O(e^{65})$
- number field sieve
  - $O(e^{(1.92+o(1)) (\ln n)^{\wedge}1/3 (\ln\ln n)^{\wedge}2/3})$, for n around 2^{1024} $O(e^{60})$
- 768-bit modulus was factored in 2009
- Extrapolating trends of factoring suggests that
  - 1024-bit moduli will be factored by 2018

RSA Security

- RSA security depends on hardness of factoring $n=pq$
  - Knowing $\Phi(n)$ enables factoring n
  - Knowing (e,d) such that ed mod $\Phi(n)=1$ enables factoring n

$\Phi(n)$ implies factorization

- Knowing both n and $\Phi(n)$, one knows

$$n = pq$$
$$\Phi(n) = (p-1)(q-1) = pq - p - q + 1$$
$$= n - p - n/p + 1$$
$$p\Phi(n) = np - p^2 - n + p$$
$$p^2 - np + \Phi(n)p - p + n = 0$$
$$p^2 - (n - \Phi(n) + 1)\, p + n = 0$$

- There are two solutions of p in the above equation.
- Both p and q are solutions.

Factoring when knowing e and d

- Knowing ed such that $ed \equiv 1 \pmod{\Phi(n)}$

  write $ed - 1 = 2^s r$ (r odd)

  choose w at random such that $1<w<n-1$

  if w not relative prime to n then return gcd(w,n)

  (if gcd(w,n)=1, what value is ($w^{2^s r}$ mod n)?)

  compute $w^r, w^{2r}, w^{4r}, \ldots$, by successive squaring until find $w^{2^t r} \equiv 1 \pmod n$

  Fails when $w^r \equiv 1 \pmod n$ or $w^{2^t r} \equiv -1 \pmod n$

  Failure probability is less than ½ (Proof is complicated)

Example: Factoring n given (e,d)

- Input: $n=2773$, $e=17$, $d=157$
- $ed-1=2668=2^2\bullet 667$ $(r=667)$
- Pick random w, compute $w^r \bmod n$
  - $w=7$, $7^{667}=1$ no good
  - $w=8$, $8^{667}=471$, and $471^2=1$, so 471 is a nontrivial square root of 1 mod 2773
  - compute $\gcd(471+1, 2773)=59$
  - $\gcd(471-1, 2773)=47$.
  - $2773=59\bullet 47$

Summary of Math-based Attacks on RSA

- Three possible approaches:
  1. Factor n = pq
  2. Determine $\Phi(n)$
  3. Find the private key d directly
- All are equivalent
  - finding out d implies factoring n
  - if factoring is hard, so is finding out d
- Should never have different users share one common modulus

The RSA Problem

- The RSA Problem: Given a positive integer n that is a product of two distinct large primes p and q, a positive integer e such that gcd(e, (p-1)(q-1))=1, and an integer c, find an integer m such that $m^e \equiv c \pmod{n}$
  - widely believed that the RSA problem is computationally equivalent to integer factorization; however, no proof is known
- The security of RSA encryption's scheme depends on the hardness of the RSA problem.

Other Decryption Attacks on RSA

Small encryption exponent e

- When e=3, Alice sends the encryption of message m to three people (public keys (e, n_1), (e, n_2), (e,n_3))
  - $C_1 = M^3 \bmod n_1$, $C_2 = M^3 \bmod n_2$, $C_3 = M^3 \bmod n_3$,
- An attacker can compute a solution to the following system

$$x \equiv c_1 \bmod n_1$$

$$x \equiv c_2 \bmod n_2$$

$$x \equiv c_3 \bmod n_3$$

- The solution x modulo $n_1 n_2 n_3$ must be M^3
  - (No modulus!), one can compute integer cubit root
- Countermeasure: padding required

Other Attacks on RSA

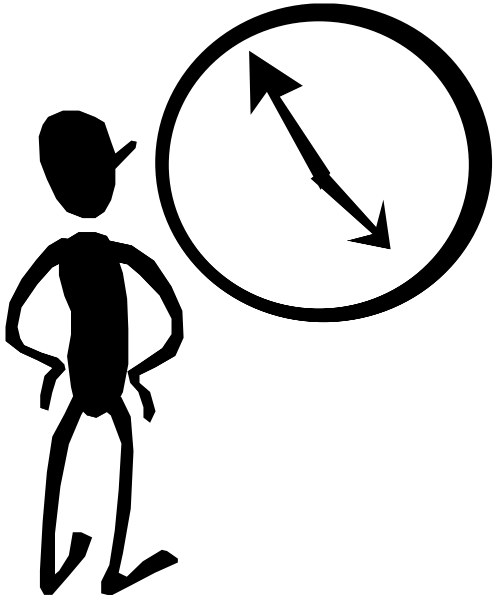
Forward Search Attack

- If the message space is small, the attacker can create a dictionary of encrypted messages (public key known, encrypt all possible messages and store them)
- When the attacker 'sees' a message on the network, compares the encrypted messages, so he finds out what particular message was encrypted

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Timing Attacks

- *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems (1996), Paul C. Kocher*
- By measuring the time required to perform decryption (exponentiation with the private key as exponent), an attacker can figure out the private key
- Possible countermeasures:
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations



Timing Attacks (cont.)

- Is it possible in practice? YES.

OpenSSL Security Advisory [17 March 2003]

Timing-based attacks on RSA keys

==================================

OpenSSL v0.9.7a and 0.9.6i vulnerability

Researchers have discovered a timing attack on RSA keys, to which OpenSSL is generally vulnerable, unless RSA blinding has been turned on.

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 $\pi(x)$ be the number of prime numbers $\leq$ x.

Theorem (prime numbers theorem)

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

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 $\leq$ 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

Naïve Method for Primality Testing

Theorem

Composite numbers have a divisor below their square root.

Proof idea:

n composite, so $n = ab$, $0 < a \le b < n$, then $a \le sqrt(n)$, otherwise we obtain $ab > n$ (contradiction).

Algorithm 1

```
for (i=2, i < sqrt(n) + 1); i++) {
    If i a divisor of n {
            n is composite
    }
}
n is prime
```

Running time is O(sqrt(n)), which is exponential in the size of the binary representation of n

More Efficient Algorithms for Primality Testing

- Primality testing is easier than integer factorization, and has a polynomial-time algorithm.
  - The Agrawal–Kayal–Saxena primality test was discovered in 2002
  - Improved version of the algorithm runs in $O((\ln x)^6)$, less efficient than randomized algorithms

How can we tell if a number is prime or not without factoring the number?

- The most efficient algorithms are randomized.
  - Solovay-Strassen
  - Rabin-Miler

Quadratic Residues Modulo A Prime

Definition

- a is a quadratic residue modulo p if $\exists$ b $\in Z_p^*$ such that $b^2 \equiv a \bmod p$,
- otherwise when $a \neq 0$, a is a quadratic nonresidue
- $\mathcal{Q}_p$ is the set of all quadratic residues
- $\overline{\mathcal{Q}}_p$ is the set of all quadratic nonresidues
- If p is prime there are (p-1)/2 quadratic residues in Z_p^*, that is $|Q_p| = (p-1)/2$
  – let g be generator of Z_p^*, then $a=g^j$ is a quadratic residue iff. j is even.

How Many Square Roots Does an Element in Q_p have?

- A element a in Q_p has exactly two square roots
  - a has at least two square roots
    - if $b^2 \equiv a \bmod p$, then $(p-b)^2 \equiv a \bmod p$
  - a has at most two square roots in Z_p^*
    - if $b^2 \equiv a \bmod p$ and $c^2 \equiv a \bmod p$, then $b^2 - c^2 \equiv 0 \bmod p$
    - then $p \mid (b+c)(b-c)$, either $b=c$, or $b+c=p$

Legendre Symbol

- Let p be an odd prime and a an integer. The Legendre symbol is defined

$$\left(\frac{a}{p}\right)=\begin{cases} 0, \text{ if } p \mid a \\ 1, \text{ if } a \in Q_p \\ -1, \text{ if } a \in \overline{Q}_p \end{cases}$$

Euler's Criterion

Theorem: If $a^{(p-1)/2} \equiv 1 \bmod p$, then a is a quadratic residue (if $\equiv -1$ then a is a quadratic nonresidue)

I.e., the Legendre symbol $\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p$

Proof. If $a = y^2$, then $a^{(p-1)/2} = y^{(p-1)} = 1 \pmod p$

If $a^{(p-1)/2}=1$, let $a = g^j$, where g is a generator of the group Z_p^*. Then $g^{j(p-1)/2} = 1 \pmod p$. Since g is a generator, $(p-1) \mid j(p-1)/2$, thus j must be even. Therefore, $a=g^j$ is QR.

Jacobi Symbol

- Let n ≥ 3 be odd with prime factorization

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$$

- The Jacobi symbol is defined to be

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{e_1} \left(\frac{a}{p_2}\right)^{e_2} \dots \left(\frac{a}{p_k}\right)^{e_k}$$

- The Jacobi symbol is in {0,-1,1}, and can be computed without factoring n or knowing whether n is prime or not

Euler Pseudo-prime

- For any prime p, the Legendre symbol $\left(\frac{a}{p}\right)$ = $a^{(p-1)/2}$ mod p
- For a composite n, if the Jacobi symbol $\left(\frac{a}{n}\right)$ = $a^{(n-1)/2}$ mod n then n is called an Euler pseudo-prime to the base a,
  - i.e., a is a "pseudo" evidence that n is prime
- For any composite n, the number of "pseudo" evidences that n is prime for at most half of the integers in Z_n^*

The Solovay-Strassen Algorithm for Primality Testing

Solovay-Strassen(n)

choose a random integer a s.t. $1 \leq a \leq n-1$

$x \leftarrow \left(\frac{a}{n}\right)$

if x=0 then return ("n is composite") // $\gcd(x,n) \neq 1$

$y \leftarrow a^{(n-1)/2} \bmod n$

if (x=y) then return ("n is prime")

// either n is a prime, or a pseudo-prime

else return ("n is composite")

// violates Euler's criterion

If n is composite, it passes the test with at most ½ prob. Use multiple tests before accepting n as prime.

Rabin-Miller Test

- Another efficient probabilistic algorithm for determining if a given number n is prime.
  - Write n-1 as $2^k m$, with m odd.
  - Choose a random integer a, $1 \leq a \leq n-1$.
  - $b \leftarrow a^m \bmod n$
  - if b=1 then return "n is prime"
  - compute $b, b^2, b^4, \ldots, b^{2^{(k-1)}}$, if we find -1, return "n is prime"
  - return "n is composite"
- A composite number pass the test with ¼ prob.
- When t tests are used with independent a, a composite passes with $(1/4)^t$ prob.
- The test is fast, used very often in practice.

Why Rabin-Miller Test Work

Claim: If the algorithm returns "n is composite", then n is not a prime.

Proof: if we choose a and returns composite on n, then

- $a^m \neq 1$, $a^m \neq -1$, $a^{2m} \neq -1$, $a^{4m} \neq -1$, …, $a^{2^{k-1}m} \neq -1$ (mod n)
- suppose, for the sake of contradiction, that n is prime,
- then $a^{n-1} = a^{2^{k}m} = 1$ (mod n)
- then there are two square roots modulo n, 1 and -1
- then $a^{2^{k-1}m} = a^{2^{k-2}m} = a^{2m} = a^m = 1$ (contradiction!)
- so if n is prime, the algorithm will not return "composite"

Coming Attractions …

- Public Key Encryption
- Reading: Katz & Lindell: Chapter 10