

CS355: Cryptography

Lecture 17, 18, 19: Public-
Key Cryptography. RSA.
Attacks against RSA.

Midterm

7:00–9:00 PM

Wed. Feb 28, 2007

ME 261

Monday – review for
Midterm, bring your
questions to class

Public Key Cryptography Overview

- Proposed in Diffie and Hellman (1976) “New Directions in Cryptography”
 - public-key encryption schemes
 - public key distribution systems
 - Diffie-Hellman key agreement protocol
 - digital signature
- Public-key encryption was proposed in 1970 by James Ellis
 - in a classified paper made public in 1997 by the British Governmental Communications Headquarters
- Diffie-Hellman key agreement and concept of digital signature are still due to Diffie & Hellman

Public Key Encryption

- Public-key encryption
 - each party has a PAIR (K, K^{-1}) of keys: K is the **public** key and K^{-1} is the **private** key, such that
$$D_{K^{-1}}[E_K[M]] = M$$
 - Knowing the public-key and the cipher, it is computationally infeasible to compute the private key
 - Public-key crypto systems are thus known to be *asymmetric* crypto systems
 - The public-key K may be made publicly available, e.g., in a publicly available directory
 - Many can encrypt, only one can decrypt

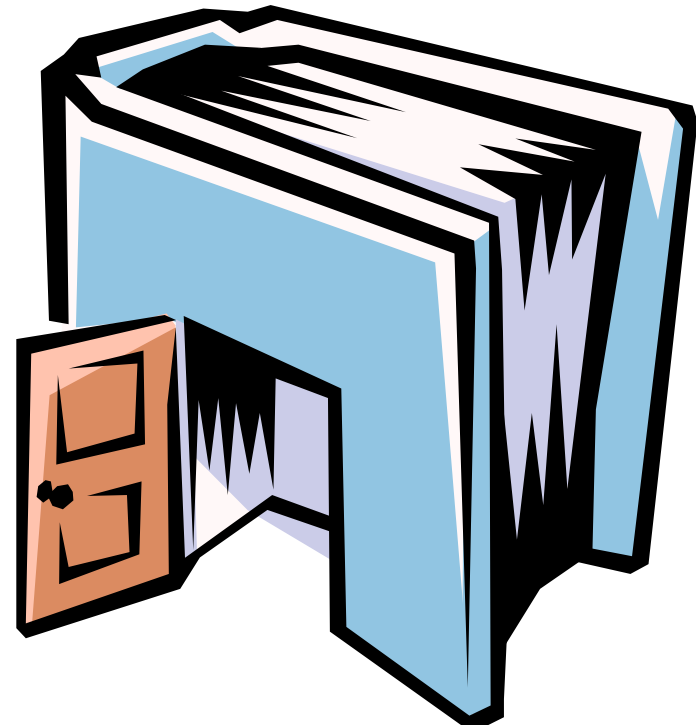
Public-Key Encryption Needs One-way Trapdoor Functions

- Given a public-key crypto system,
 - Alice has public key K
 - E_K must be a one-way function, knowing $y = E_K[x]$, it should be difficult to find x
 - However, E_K must **not** be one-way from Alice's perspective. The function E_K must have a trapdoor such that knowledge of the trapdoor enables one to invert it

Trapdoor One-way Functions

Definition:

A function $f: \{0,1\}^* \rightarrow \{0,1\}^*$ is a trapdoor one-way function iff $f(x)$ is a one-way function; however, given some extra information it becomes feasible to compute f^{-1} : given y , find x s.t. $y = f(x)$



RSA Algorithm

- Invented in **1978** by Ron **R**ivest, Adi **S**hamir and Leonard **A**dleman
 - Published as R L Rivest, A Shamir, L Adleman, "*On Digital Signatures and Public Key Cryptosystems*", Communications of the ACM, vol 21 no 2, pp120-126, Feb 1978
- Security relies on the difficulty of factoring large composite numbers
- Essentially the same algorithm was discovered in 1973 by Clifford Cocks, who works for the British intelligence

$$\mathbb{Z}_{pq}^*$$

- Let p and q be two large primes
- Denote their product $n=pq$.
- $\mathbb{Z}_n^* = \mathbb{Z}_{pq}^*$ contains all integers in the range $[1, pq-1]$ that are relatively prime to both p and q
- The size of \mathbb{Z}_n^* is
 - $\phi(pq) = (p-1)(q-1) = n - (p+q) + 1$
 - For every $x \in \mathbb{Z}_{pq}^*$, $x^{(p-1)(q-1)} \equiv 1 \pmod{n}$

Exponentiation in Z_{pq}^*

- Motivation: We want to use exponentiation for encryption
- Let e be an integer, $1 < e < (p-1)(q-1)$
- When is the function $f(x) = x^e$, a one-to-one function in Z_{pq}^* ?
- If x^e is one-to-one, then it is a permutation in Z_{pq}^* .

Exponentiation in Z_{pq}^*

- Claim: If e is relatively prime to $(p-1)(q-1)$ then $f(x)=x^e$ is a one-to-one function in Z_{pq}^*
- Proof by constructing the inverse function of f .
As $\gcd(e, (p-1)(q-1))=1$, then there exists d and k s.t. $ed=1+k(p-1)(q-1)$
- Let $y=x^e$, then $y^d=(x^e)^d=x^{1+k(p-1)(q-1)}=x \pmod{pq}$,
i.e., $g(y)=y^d$ is the inverse of $f(x)=x^e$.

RSA Public Key Crypto System

Key generation:

Select 2 large prime numbers of about the same size, p and q

Compute $n = pq$, and $\phi(n) = (q-1)(p-1)$

Select a random integer e , $1 < e < \phi(n)$, s.t.
 $\gcd(e, \phi(n)) = 1$

Compute d , $1 < d < \phi(n)$ s.t. $ed \equiv 1 \pmod{\phi(n)}$

Public key: (e, n)

Private key: d

Note: p and q must remain secret

RSA Description (cont.)

Encryption

Given a message M , $0 < M < n$ $M \in \mathbb{Z}_n \setminus \{0\}$

use public key (e, n)

compute $C = M^e \bmod n$ $C \in \mathbb{Z}_n \setminus \{0\}$

Decryption

Given a ciphertext C , use private key (d)

Compute $C^d \bmod n = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n = M$

RSA Example

- $p = 11, q = 7, n = 77, \phi(n) = 60$
- $d = 13, e = 37$ ($ed = 481; ed \bmod 60 = 1$)
- Let $M = 15$. Then $C \equiv M^e \pmod{n}$
 $C \equiv 15^{37} \pmod{77} = 71$
- $M \equiv C^d \pmod{n}$
 $M \equiv 71^{13} \pmod{77} = 15$

Why does RSA work?

- Need to show that $(M^e)^d \pmod n = M$, $n = pq$
- We have shown that when $M \in \mathbb{Z}_{pq}^*$, i.e., $\gcd(M, n) = 1$, then $M^{ed} \equiv M \pmod n$
- What if $M \in \mathbb{Z}_{pq} \setminus \{0\} \setminus \mathbb{Z}_{pq}^*$, e.g., $\gcd(M, n) = p$.
 - $ed \equiv 1 \pmod{\phi(n)}$, so $ed = k\phi(n) + 1$, for some integer k .

$$M^{ed} \pmod p = (M \pmod p)^{ed} \pmod p = 0$$

so $M^{ed} \equiv M \pmod p$

$$M^{ed} \pmod q = (M^{k\phi(n)} \pmod q) (M \pmod q) = M \pmod q$$

so $M^{ed} \equiv M \pmod q$

As p and q are distinct primes, it follows from the CRT that

$$M^{ed} \equiv M \pmod{pq}$$

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?



Square and Multiply Algorithm for Exponentiation

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

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

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

RSA Implementation: Decryption

- CRT is used in RSA by creating two equations for decryption:

The goal is to compute M , from $M = C^d \pmod n$

$$M1 = M \pmod p = C^d \pmod p$$

$$M2 = M \pmod q = C^d \pmod q$$

Fermat theorem on the exponents

$$M1 \equiv C^{d \pmod{(p-1)}} \pmod p$$

$$M2 \equiv C^{d \pmod{(q-1)}} \pmod q$$

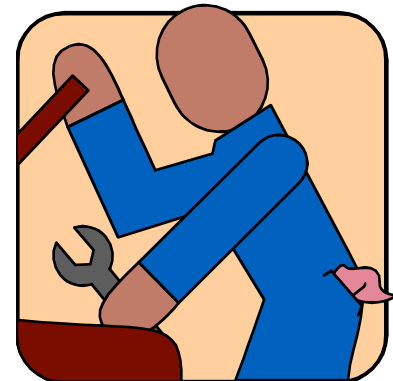
- then the pair of equations

$$M \equiv M1 \pmod p,$$

$$M \equiv M2 \pmod q$$

has a unique solution M .

$$M \equiv M1(q^{-1} \pmod p)q + M2(p^{-1} \pmod q)p \pmod n$$



Efficiency of computation modulo n

- Suppose that n is a k -bit number, and $0 < x, y < n$
 - 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 $O((\log c) k^2)$

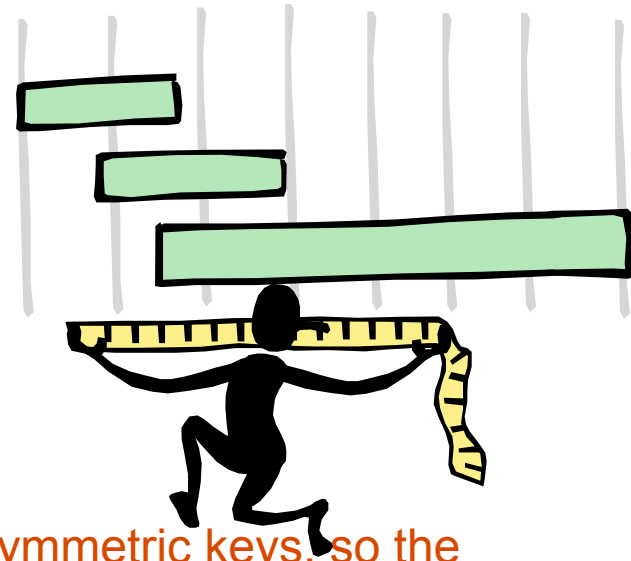
RSA on Long Messages

- RSA requires that the message M is at most $n-1$ where n is the size of the modulus.
- What about longer messages?

They are broken into blocks.

Smaller messages are padded.

CBC is used to prevent attacks regarding the blocks.



- **NOTE:** In practice RSA is used to encrypt symmetric keys, so the message is not very long.

Pohlig-Hellman Exponentiation Cipher

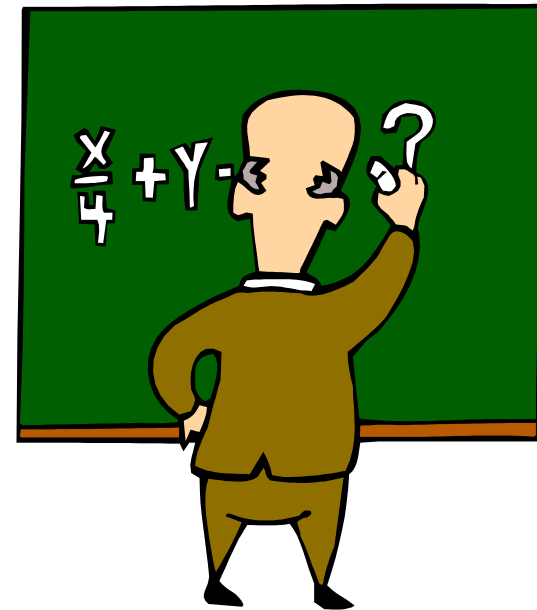
- A symmetric key exponentiation cipher
 - encryption key (e,p) , where p is a prime
 - decryption key (d,p) , where $ed \equiv 1 \pmod{p-1}$
 - to encrypt M , compute $M^e \pmod p$
 - to decrypt C , compute $C^d \pmod p$

Attacks Against RSA

Math-Based Key Recovery Attacks

- Three possible approaches:
 1. Factor $n = pq$
 2. Determine $\phi(n)$
 3. Find the private key d directly

- All the above are equivalent to factoring n



Knowing $\phi(n)$ 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 Large Numbers

- **RSA-640 bits, Factored Nov. 2 2005**
- **RSA-200 (663 bits) is the largest RSA Challenge Number factored to date, May 2005**
- Three most effective algorithms are
 - quadratic sieve
 - elliptic curve factoring algorithm
 - number field sieve

Fermat Factorization

- Compute $n + 1, n + 2^2, n + 3^2 \dots$ till we find a square
- $n + x^2 = y^2$
- So $n = (x-y)(x+y)$, $p = x-y$ and $q = x+y$
- Example: $n = 295927, 295927 + 3^2 = 544^2$
- So $n = (544+3)(544-3)$
- Works well when p and q are close to each other

Factoring Large Numbers

- One idea many factoring algorithms use:
 - Suppose one finds $x^2 \equiv y^2 \pmod{n}$ such that $x \neq y \pmod{n}$ and $x \neq -y \pmod{n}$.
Then $n \mid (x-y)(x+y)$. Neither $(x-y)$ or $(x+y)$ is divisible by n ; thus, $\gcd(x-y, n)$ has a non-trivial factor of n

Factoring when knowing e and d

- **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., $n=3 * 5=15$, then $x^2 \equiv 1 \pmod{15}$ has the following solutions: 1, 4, 11, 14

Factoring when knowing e and d

- 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 $x^2 \equiv 1 \pmod{15}$
 - $\gcd(4+1, 15) = 5$
 - $\gcd(4-1, 15) = 3$
 - $\gcd(11+1, 15) = 3$
 - $\gcd(11-1, 15) = 5$

Factoring when knowing e and d (page 186)

- 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 $\gcd(w, n)$
compute $w^r, w^{2r}, w^{4r}, \dots$, until find $w^{2^t r} \equiv 1 \pmod{n}$
Fails when $w^r \equiv 1 \pmod{n}$ or $w^{2^t r} \equiv -1 \pmod{n}$

Input: $n=2773, e=17, d=157$

$$ed-1=2668=2^2 \cdot 667 \quad (r=667)$$

Pick random w , compute $w^r \pmod{n}$

$$w=7, 7^{667} \pmod{2773} = 1 \text{ no good}$$

$$w=8, 8^{667} \pmod{2773} = 471, \text{ and } 471^2 = 1, \text{ so } 471 \text{ is a nontrivial square root of } 1 \pmod{2773}$$

$$\text{compute } \gcd(471+1, 2773) = 59$$

$$\gcd(471-1, 2773) = 47.$$

$$2773 = 59 \cdot 47$$

Example: Factoring n given d

- Input: $n=2773$, $e=17$, $d=157$
- $ed-1=2668=2^2 \cdot 667$ (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 $\gcd(471+1, 2773)=59$

$\gcd(471-1, 2773)=47$.

$2773=59 \cdot 47$

Decryption attacks on RSA

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

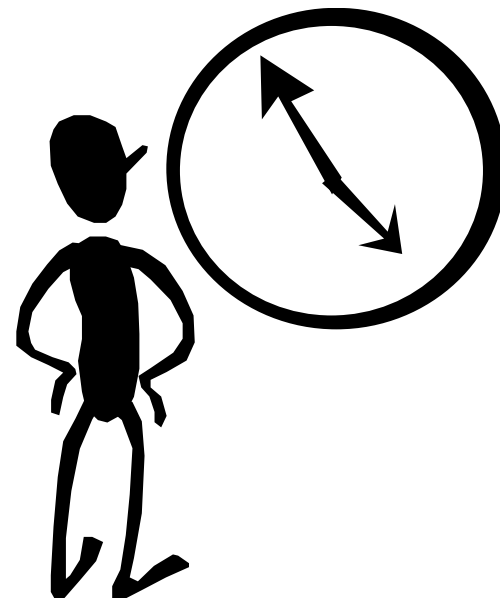
The security of RSA encryption's scheme depends on the hardness of the RSA problem.

Summary of Key Recovery 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

Finding d: 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.

RSA blinding: the decryption time is no longer correlated to the value of the input ciphertext

Instead of computing $c^d \bmod n$, choose a secret random value r and compute $(r^e c)^d \bmod n$.

A new value of r is chosen for each ciphertext.