MAS 433: Cryptography

Lecture 14
Public Key Encryption
Part 1: RSA

Wu Hongjun

Lecture Outline

- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
- **Public key encryption**
  - **RSA**
    - **Specification**
    - **Implementation**
    - **Security**
  - ElGamal
  - Message padding (OAEP)
- Digital signature
- Key establishment and management
- Introduction to other cryptographic topics

Recommended Reading

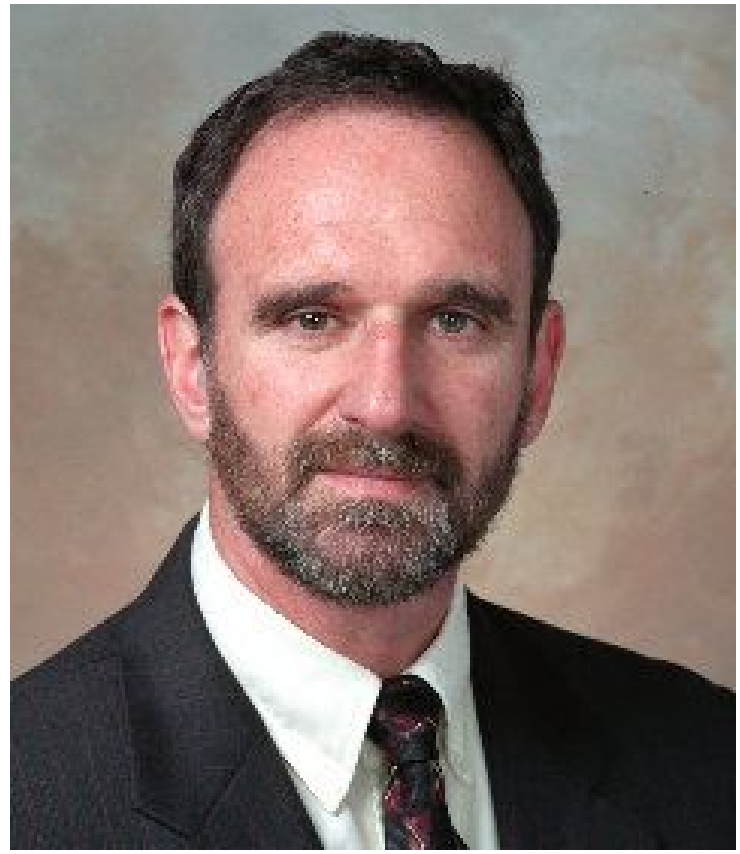
- CTP Section 5.1 to 5.7
- HAC Section 8.1 and 8.2
- Wikipedia
  - Public key cryptosystem
    http://en.wikipedia.org/wiki/Public-key_cryptography
  - RSA
    http://en.wikipedia.org/wiki/RSA
  - Primality testing
    http://en.wikipedia.org/wiki/Primality_test
  - Integer factorization
    http://en.wikipedia.org/wiki/Integer_factorization

Public Key Cryptosystem

- Symmetric key encryption
  - The same secret key is used for encryption and decryption
- How to communicate secretly if sender & receiver do not share a secret key before communication starts?
  - Common problem for large computer network
  - Public key cryptosystems can solve this problem
    - **Diffie-Hellman key exchange** (1976)
      - The first paper on public key cryptosystem
    - **Public key encryption**

Public Key Cryptosystem

Whitfield Diffie



Martin Hellman

Diffie-Hellman Key Exchange

Public Key Encryption

- Each receiver has two keys
  - Encryption key (called public key)
    - Everyone knows the encryption key of a receiver
    - **Everyone can encrypt a message using the public key of a receiver** and send the ciphertext to that receiver
  - Decryption key (called private key)
    - Only the receiver knows its decryption key
      - Difficult to derive the private key from public key
    - **Only the receiver can decrypt the ciphertext encrypted using its public key**

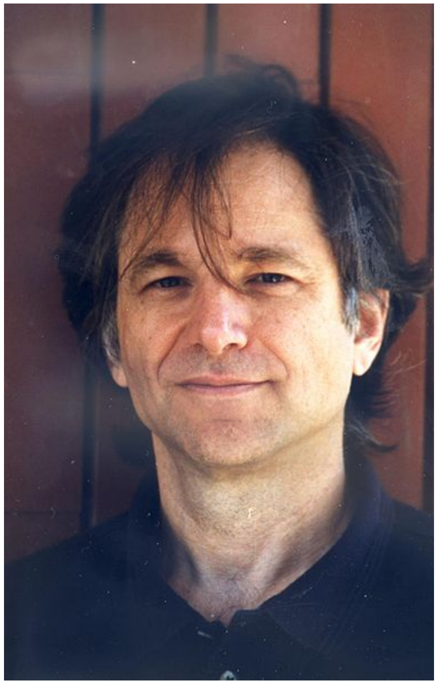
Public Key Encryption

- Many public key encryption schemes
- RSA (1978)
  - The first public key encryption scheme
  - Based on the difficulty of integer factorization & `discrete logarithm’
- ElGamal (1985)
  - Based on the difficulty of discrete logarithm
    - discrete logarithm: $g^x \bmod p = y$ (given y, to find x)

RSA

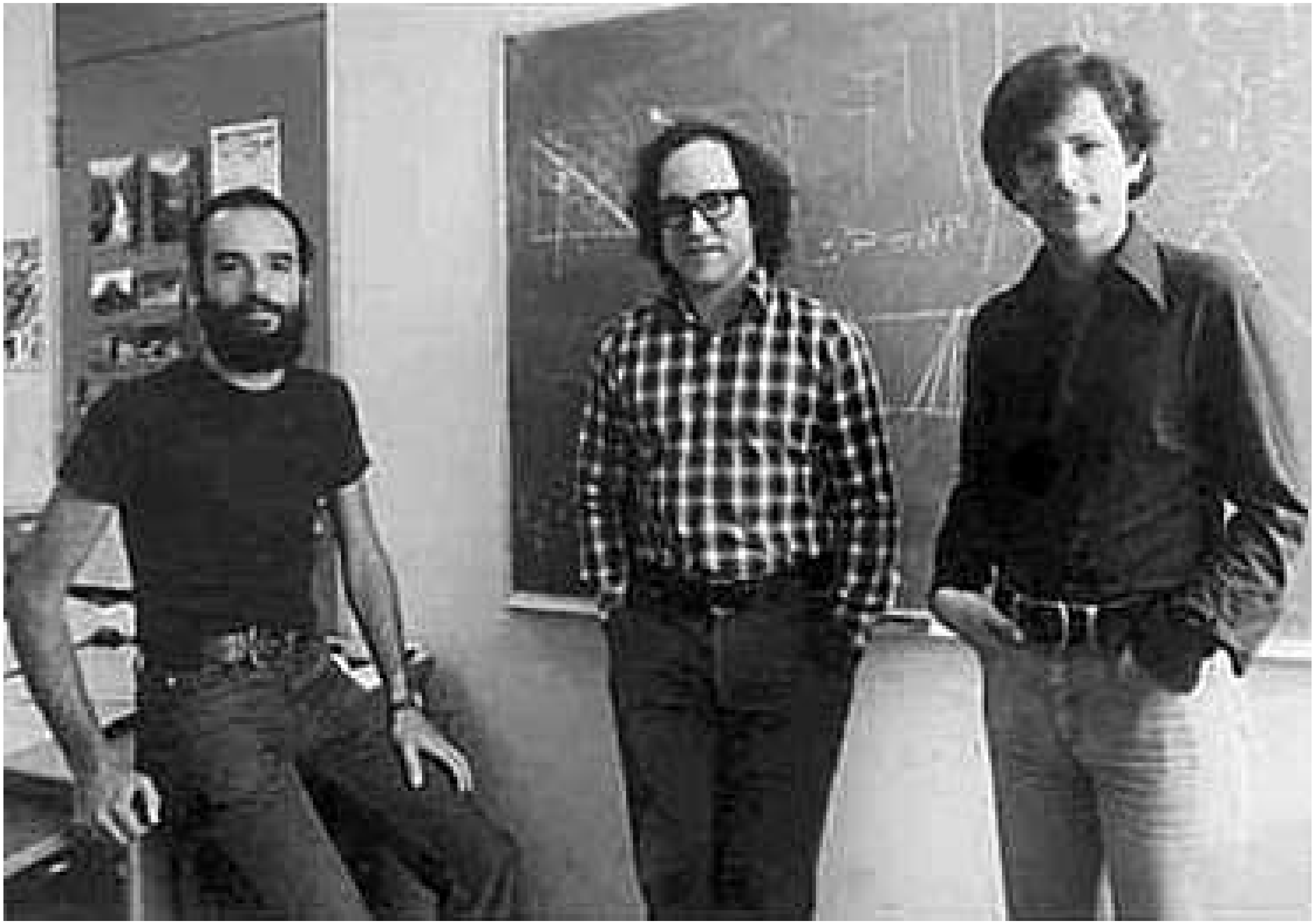
Ron Rivest

Adi Shamir



Leonard Adleman

RSA



RSA

- Key generation of each receiver:
  - Generate two distinct **large** prime numbers p and q
  - Compute $n = p \times q$
  - Compute $\varphi(n) = (p\text{-}1) \times (q\text{-}1)$
    - φ is Euler's totient function
  - Choose an integer e that is coprime to $\varphi(n)$
  - Find d satisfying $e \times d \equiv 1 \bmod \varphi(n)$

public key: *e*, *n*

private key: *d*

RSA

- Encryption

$$\boldsymbol{c = m^e \bmod n}$$ (plaintext m: $0 < m <$ n)

- Decryption

$$\boldsymbol{m = c^d \bmod n}$$

RSA

RSA's decryption recovers the message

Simple but incomplete proof:

- Euler's theorem:

  Let a be a positive integer coprime to n, then

$$a^{\varphi(n)} \equiv 1 \bmod n$$

- RSA decryption:

$$\begin{aligned} c^d \bmod n &= (m^e \bmod n)^d \bmod n \\ &= m^{ed} \bmod n \\ &= m^{\beta\varphi(n)+1} \bmod n \end{aligned}$$

  If m and n are coprime, then $m^{\beta\varphi(n)} \bmod n = 1$

  $\therefore\ c^d \bmod n = m$

RSA

RSA's decryption recovers the message

The complete proof requires the following theorem:

- Fermat's little theorem:

  Let a be a positive integer coprime to a prime number p, then

  $$a^{p-1} \equiv 1 \bmod p$$

- Chinese Remainder Theorem (special case):

  If n_1 and n_2 are coprime, $x < n_1 n_2$, and x satisfies

  $$x \equiv a \bmod n_1$$

  $$x \equiv a \bmod n_2$$

  then there is a unique solution

  $$x = a \bmod n_1 n_2$$

Brief explanation:

$n_1 \mid (x\text{-}a)$

$n_2 \mid (x\text{-}a)$

since n_1 and n_2 are coprime, we get

$n_1 n_2 \mid (x\text{-}a)$

i.e., $x\text{-}a \bmod n_1 n_2 = 0$

RSA

- RSA's decryption recovers the message

  complete proof:

Let $x = c^d \bmod n$,

$$
\begin{aligned}
x \bmod p &= ((m^e)^d \bmod n) \bmod p \\
&= (m^e)^d \bmod p \\
&= m^{\beta(p-1)(q-1)+1} \bmod p
\end{aligned}
$$

If m and p are coprime, according to Fermat's little theorem: $m^{p-1} \bmod p = 1$

$$\therefore \quad x \bmod p = m^{\beta(p-1)(q-1)+1} \bmod p = m \bmod p \qquad (1)$$

If m is the multiple of p, then

$$x \bmod p = m^{\beta(p-1)(q-1)+1} \bmod p = 0 = m \bmod p \qquad (2)$$

From (1) and (2), $\quad x \equiv m \bmod p \qquad (3)$

Similarly: $\quad x \equiv m \bmod q \qquad (4)$

From (3), (4) and Chinese Remainder Theorem:

$$x = m \bmod pq = m$$

RSA

- Example (Toy RSA):

  Key generation:

    - $p = 61,\ \ q = 53$
    - $n = 61 \times 53 = 3233$
    - $\varphi(n) = (61\text{-}1)(53\text{-}1) = 3120$
    - choose public key $e = 17$, e is coprime to $\varphi(n)$
    - find private key $d = 2753$ satisfying $e \times d \equiv 1 \bmod \varphi(n)$

  Encryption:

  If $m = 37$, then $c = 37^{17} \bmod 3233 = 1350$

  Decryption:

  $m = 1350^{2753} \bmod 3233 = 37$

RSA Implementation

- Key generation:
  - Generate two distinct large prime numbers ***p* and *q***
  - Compute $n = p \times q$
  - Compute $\varphi(n) = (p\text{-}1) \times (q\text{-}1)$
    - φ is Euler's totient function
  - Choose an integer e that is coprime to $\varphi(n)$
  - Find ***d*** satisfying $e \times d \equiv 1 \bmod \varphi(n)$

- Encryption

$$c = \boldsymbol{m^e \bmod n}$$

- Decryption

$$m = \boldsymbol{c^d \bmod n}$$

1. **How to find *p* & *q*?**
2. **How to find *d*?**
3. **How to compute (m^e mod *n*) and (c^d mod *n*) efficiently?**

RSA Implementation: How to find p & q?

- How to find a large prime number?
  - Randomly select a large integer
  - Then test whether it is prime or not

- Questions:
  - What is the probability that a large random integer is prime?
  - How to test whether a large random integer is prime?

RSA Implementation: How to find *p* & *q*?

$\pi(x)$: the number of primes less than or equal to a real number x

- Prime Distribution Theorem

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

$$\pi(x) \sim \frac{x}{\ln x}.$$

RSA Implementation: How to find *p* & *q*?

$$\pi(x) \sim \frac{x}{\ln x}$$

- A random 512-bit integer is prime with probability about
$$\frac{1}{\ln 2^{512}} \approx \frac{1}{355}$$
- A random 1024-bit integer is prime with probability about
$$\frac{1}{\ln 2^{1024}} \approx \frac{1}{710}$$

$\Rightarrow$ The probability that a large random integer is sufficiently large for practical applications

RSA Implementation: How to find *p* & *q*?

- Primality testing:
  - Naïve methods
  - **Probabilistic tests**
    - **Low complexity**
    - **Commonly used**
  - Fast deterministic tests

RSA Implementation: How to find *p* & *q*?

- Primality testing
  - Naïve methods
    - The simplest primality testing:
      - To test whether an integer n is prime or not, try all the integers less than or equal to $n^{0.5}$ to check whether n is divisible by any of those integers
      - Complexity: $O(n^{0.5})$

RSA Implementation: How to find *p* & *q*?

- Primality testing
  - Probabilistic tests
    - Fermat primality test
      - Simple, but not useful for detecting some special composite numbers
      - Useful for quick screening, then test the remaining numbers using other primality testing methods
      - Used in Perfect Good Privacy (PGP) for primality testing
    - Miller-Rabin test
      - The commonly used primality testing method
        - » Mathematica, OpenSSL, …

RSA Implementation: How to find p & q?

- Primality testing
  - Probabilistic tests
    - Fermat primality test
      - To test whether an integer n is prime or not, choose some integer a comprime to n $(a>1)$,
        - » If $a^{n-1} \bmod n \neq 1$, then n is composite
        - » If $a^{n-1} \bmod n = 1$, then n may or may not be prime
      - As more different values of a are tested, the accuracy improves
        - » But for some special composite number n (called Carmichael numbers), for all the a coprime to n,

$$a^{n-1} \bmod n = 1$$

RSA Implementation: How to find *p* & *q*?

- Primality testing
  - Probabilistic tests
    - Miller-Rabin test

Given an integer n, write $n-1=2^r s$, where s is odd

Choose a random integer a with $2 \le a \le n$-1

1. If $a^s \not\equiv 1 \pmod{n}$ and $a^{2^j s} \not\equiv -1 \pmod{n}$ for all $0 \le j \le r-1$, then n is a composite;
2. Otherwise, n may or may not be prime

A prime can always pass through the above test (never be identified as composite). A composite number can be identified as probably prime with probability 1/4 for a random integer a . With N random distinct integers a, a composite number is identified as probably prime with probability 2^{-2N}

RSA Implementation: How to find *p* & *q*?

- Primality testing
  - Fast deterministic tests
    - In 2002, Agrawal, Kayal and Saxena found a new deterministic primality test (AKS), with complexity $O((\log n)^{12})$
    - In 2005, the complexity is reduced to $O((\log n)^{6})$. It is still much slower than probabilistic methods

RSA Implementation: How to find d ?

- Use the extended Euclidean algorithm to find d satisfying $ed \equiv 1 \bmod \varphi(n)$
  - Euclidean algorithm
    - to find $\gcd(a, b)$
  - extended Euclidean algorithm
    - to find $ax + by = \gcd(a, b)$
      - If $\gcd(a, b) = 1$, then $ax \equiv 1 \bmod b$; $by \equiv 1 \bmod a$

RSA Implementation: How to compute a^x mod n efficiently?

1. Represent a t-bit exponent x in binary format as

$$x = x_{t-1}x_{t-2}\cdots x_2 x_1 x_0, \text{ i.e., } x = \sum_{i=0}^{t-1} x_i 2^i$$

2. Compute $y_i = a^{2^i} \bmod n$ as

t square-mod operations

$$\underline{y_i = (y_{i-1})^2 \bmod n}, \text{ where } y_0 = a^{2^0} \bmod n = a$$

3. Then $a^x \bmod n$ is computed efficiently as

$$a^x \bmod n = a^{\sum_{i=0}^{t-1} x_i 2^i} \bmod n = \left(\prod_{i=0}^{t-1} a^{x_i 2^i}\right) \bmod n$$

$$= \left(\prod_{i=0}^{t-1} (a^{2^i})^{x_i}\right) \bmod n = \underline{\left(\prod_{i=0}^{t-1} y_i^{x_i}\right) \bmod n}$$

At most t-1 multiply-mod operations

RSA Implementation: How to compute a^x mod n efficiently?

- Implement the method on the previous slide as :

$$
\begin{aligned}
& y = a,\ z = 1 \\
& \text{for } i = 0 \ \text{ to } \ t-1 \ \text{ do} \\
& \{ \\
& \qquad \text{if } \ x_i = 1, \text{ then } z = z \cdot y \bmod n \\
& \qquad y = y^2 \bmod n \\
& \}
\end{aligned}
$$

RSA Implementation: How to compute $a^x \bmod n$ efficiently?

- Square-and-multiply algorithm in the textbook:
  - Compare to the algorithm on the previous slide, the value of i decreases

$$z = 1$$

$$\text{for } i = t-1 \text{ downto } 0 \text{ do}$$

$$\{$$

$$\quad \mathrm{z} = \mathrm{z}^2 \bmod n$$

$$\quad \text{if } x_i = 1, \text{ then } z = z \cdot a \bmod n$$

$$\}$$

RSA Security

Attacks on RSA:

- **To factorize *n***
  - Once n is factorized, d can be computed
  - Difficult for large n
- **Other attacks**

RSA Security: Integer Factorization

- Integer factorization
  - Here we consider only **RSA moduli**
    - Product of two primes (also called semiprimes, biprimes)
- Many integer factorization techniques
  - Trial division
  - …..
  - Dixon’s random squares algorithm
    - Quadratic sieve
    - General number field sieve

RSA Security: Integer Factorization

- Trial division
  - To factorize integer n, try all the integers less than or equal to $n^{0.5}$ to check whether n is divisible
  - Complexity: $O(n^{0.5})$

RSA Security: Integer Factorization

- Dixon's random squares algorithm
  - Basic idea: (Fermat)
    - used in many factorization algorithms

Suppose that we can find $x \not\equiv \pm y \pmod{n}$ such that

$x^2 \equiv y^2 \pmod{n}$, then $n \mid (x-y)(x+y)$.

But neither $(x-y)$ or $(x+y)$ is divisible by n since $x \not\equiv \pm y \pmod{n}$.

Therefore $\gcd(x-y, n)$ is a non-trivial factor of n

$\gcd(x+y, n)$ is another non-trivial factor of n

Example: $10^2 \equiv 32^2 \pmod{77}$

$$\gcd(10+32, 77) = 7, \gcd(10-32, 77) = 11$$

RSA Security: Integer Factorization

- Dixon's random squares algorithm (contd.)

  Smooth number

  – An integer which factors completely into small prime numbers

  – A positive integer is called ***B*-smooth** if none of its prime factors is greater than B.

  – Example:

    $1620 = 2^2 \times 3^4 \times 5$

    1620 is 5-smooth since none of its prime factors is greater than 5.

RSA Security: Integer Factorization

- Dixon's random squares algorithm (contd.)

$Q(x) \approx \alpha\sqrt{n}$

1) Let $m = \lfloor \sqrt{n} \rfloor$, define a function $Q(x) = (m + x)^2 - n$
2) define the value of B (it depends on the size of n, normally it cannot be than 2^{50})

   Denote those primes $\le B$ as $\{p_1, p_2, \cdots, p_t\} = \{-1, 2, 3, 5, \cdots, p_t\}$
3) Randomly select small (positive or negative) integers x.

   Keep those integers satisfying that $Q(x)$ is B-smooth.

   Denote those integers as $x_1, x_2, \cdots, x_\mu$

$$Q(x_i) = p_1^{e_{i,1}} \times p_2^{e_{i,2}} \times p_3^{e_{i,3}} \times \cdots \times p_t^{e_{i,t}}$$

RSA Security: Integer Factorization

- Dixon's random squares algorithm (contd.)

4) With t such x_i, by solving binary linear equations, we can find a subset $A \subset \{1,2,3,4,5,\cdots,t\}$, so that $\sum_{i\in A} e_{i,j}$ is even for all the values of j $(1 \le j \le t)$

5) Then $\prod_{i\in A} Q(x_i) = p_1^{\sum_{i\in A} e_{i,1}} \times p_2^{\sum_{i\in A} e_{i,2}} \times p_3^{\sum_{i\in A} e_{i,3}} \times \cdots \times p_t^{\sum_{i\in A} e_{i,t}} = y^2$

6) Thus $\prod_{i\in A} Q(x_i) \equiv y^2 \bmod n$

$$(\prod_{i\in A} (m + x_i))^2 \equiv y^2 \bmod n$$

$$\mathrm{z}^2 \equiv y^2 \bmod n$$

If $z \not\equiv \pm y \bmod n$, then $\gcd(z - y, n)$ gives a factor of n

Example: Factorize $n = 4841$

1. $m = \lfloor \sqrt{n} \rfloor = 69, Q(x) = (m+x)^2 - n$
2. set $B = 11$, the factor base is $\{-1,2,3,5,7,11\}$
3. $x = -8 \rightarrow Q(x) = -1120 = (-1)\times 2^5 \times 5 \times 7$

   $x = -4 \rightarrow Q(x) = -616 = (-1)\times 2^3 \times 7 \times 11$

   $x = -2 \rightarrow Q(x) = -352 = (-1)\times 2^5 \times 11$

   $x = 0 \rightarrow Q(x) = -80 = (-1)\times 2^4 \times 5$

   $x = 2 \rightarrow Q(x) = 200 = 2^3 \times 5^2$

   $x = 3 \rightarrow Q(x) = 343 = 7^3$

   $x = 6 \rightarrow Q(x) = 784 = 2^4 \times 7^2$
4. Find the set A as $\{x = -4, x = -2, x = 3\}$
5. $y^2 = Q(-4)\times Q(-2)\times Q(3) = (-1)^2 \times 2^8 \times 7^4 \times 11^2$

   $\Rightarrow y = (-1)\times 2^4 \times 7^2 \times 11 \equiv -3783 \pmod{4841}$

   $z = (m-4)\times(m-2)\times(m+3) \equiv 3736 \pmod{4841}$

   $\gcd(z-y, n) = \gcd(3736+3783, 4841) = 103$

   $\gcd(z+y, n) = \gcd(3736-3783, 4841) = 47$

   $n = 4841 = 47 \times 103$

If we use the set A $\{x=6\}$, then
$y=2^2\times 7=28$
$z=(m+6)=75$
Then we get:
$\gcd(z-y,n) = 47$
$\gcd(z+y,n) = 103$

RSA Security: Integer Factorization

- Quadratic sieve
  - Very similar to Dixon's random squares algorithm
  - But with efficient sieving method to generate smooth numbers
- General number field sieve
  - Improve the quadratic sieve
  - Convert the integer factorization problem to factorization over algebraic number field
    - so as to generate "more" smooth number

RSA Security: Integer Factorization

- Complexities of factorization algorithms

Trial division: $O(\sqrt{n}) \to O(e^{0.5\underline{\ln n}})$

Dixon's Random Squares Algorithm: $O(e^{(1+O(1))\underline{(\ln n)}^{1/2}(\ln\ln n)^{1/2}})$

Quadratic sieve: $O(e^{(1+O(1))\underline{(\ln n)}^{1/2}(\ln\ln n)^{1/2}})$

General number field sieve: $O(e^{(1+O(1))\underline{(\ln n)}^{1/3}(\ln\ln n)^{2/3}})$

RSA Security: Integer Factorization

- The size of RSA moduli

  NIST recommendation, 2007:

size of *n*	**security level**
1024-bit	80 bits
2048-bit	112 bits
3072-bit	128 bits
7680-bit	192 bits
15360-bit	256 bits

RSA Security: Integer Factorization

- Some RSA modulus factorization records
  - 129-digit modulus 1994 → Quadratic sieve
  - 155-digit (512-bit) modulus 1999
  - 174-digit (576-bit) modulus 2003
  - 200-digit (663-bit) modulus 2005
  - 232-digit (768-bit) modulus 2009

General number field sieve

Complexity: about 2000 CPU cores (2.2GHz) for 1 year

1024-bit modulus: when? method?

RSA Security: Other Attacks (1)

- Trivial attacks
  - If p or q is known to the attacker → broken
  - If $\varphi(n)$ is known to the attacker → broken

RSA Security: Other Attacks (2)

- Attack on shared modulus
  - Shared modulus
    - each user is given a public key (e_i, n) and private key d_i
    - They share the same modulus n
  - Attack
    - Each user can factorize n easily from e_i and d_i
    - Then each user can find the private keys of other users
    - How to factorize?

RSA Security: Other Attacks (2)

- Attack on shared modulus (contd.)
  - Factorize n from e and d

1) Since $e \cdot d \equiv 1 (\bmod \varphi(n))$,

$$e \cdot d - 1 = \beta(p-1)(q-1),$$

we know that $e \cdot d - 1$ is even

2) Randomly select an integer x, compute

$$y = x^{(e \cdot d - 1)/2} \bmod n$$

3) We know that $x^{e \cdot d - 1} \bmod n = x^{\beta\varphi(n)} \bmod n = 1$ (Euler's theorem)

4) From 2) and 3), we know that

$$y^2 = 1 \bmod n$$

Thus $\gcd(y-1, n)$ gives a factor of n if $y \not\equiv \pm 1 (\bmod n)$

Slide 33

RSA Security: Other Attacks (3)

- The message size is small
  - Attack

  Example: A 64-bit secret m is encrypted as $c = m^e \bmod n$ (n, e, d are huge)

  With probability about 20%, a random 64-bit m can be written as $m = m_1 m_2$, where $m_1, m_2 < 2^{34}$.

  Now an attacker builds two tables:

$$T_1[i] = \frac{c}{i^e} \bmod n \quad \text{for } 1 \le i \le 2^{34}$$

$$T_2[j] = j^e \bmod n \quad \text{for } 1 \le j \le 2^{34}$$

  If $T_1[i] = T_2[j]$ for some i and j, then the message $m = i \times j$

  Complexity: about 2×2^{34}

RSA Security: Other Attacks (4)

- The exponent e is too small
  - Attack 1:

    Example: if $e = 3$, then for small m (say, $m < n^{1/3}$),

    $$c = m^3 \bmod n = m^3$$

    => m can be recovered from c easily
  - Attack 2:

    Example: if $e = 3$, and m is large. The same message m is sent to 3 different receivers

    $$c_1 = m^3 \bmod n_1 \quad (1)$$
    $$c_2 = m^3 \bmod n_2 \quad (2)$$
    $$c_3 = m^3 \bmod n_3 \quad (3)$$

    From Chinese Remainder Theorem and (1), (2), (3),

    $m^3 \bmod n_1 n_2 n_3$ can be obtained, i.e., m^3 becomes known.

    m can thus be recovered easily from m^3

RSA Security: Other Attacks (4)

- Recommended value: $e = 65537 = 2^{16}+1$
  - Encryption takes 17 modular multiplies
  - Fast encryption, but slow decryption
    - Encryption is about 80 times faster than decryption for 1024-bit n
    - But RSA encryption with this e and 1024-bit n is still more than 50 times slower than AES encryption on computer;

RSA Security: Other Attacks (5)

- How about choose small private key *d* to increase decryption speed?
  - In the key generation process, choose *d* first, then compute *e*
  - But **the value of *d* must be large for security reason**
    - Brute force attack: the size of *d* should be more than 128 bits
    - Advanced attack:
      - If $d < n^{0.25}$, *d* can be recovered from *e* and *n* easily (1987)
      - If $d < n^{0.292}$, *d* can be recovered from *e* and *n* easily (1998)
      - It is conjectured that if $d < n^{0.5}$, *d* can be recovered from *e* and *n* easily (open problem)

RSA Security: Other Attacks (6)

- Attack on `public' encryption
  - Attacker can perform encryption of any message
  - If the entropy of the message is not large, the attacker can encrypt all the possible messages, then compare those ciphertexts with the received ciphertext, and recover the message

RSA Security

- How to make RSA strong
  - Large modulus
    - 3072 bits for 128-bit security
    - 15360 bits for 256-bit security
  - Private key larger than $n^{0.5}$
  - Message padding (to learn later)
    - **To introduce randomness to the plaintext *m***
    - To pad message m so that the length of padded message is close to that of n

RSA Applications

- Used in almost all the secure Internet communication applications
  - Public key infrastructure
  - TLS/SSL
  - Secure e-mail: PGP, Microsoft Outlook …

Summary

- Public key encryption
  - Allows two party to communicate secretly without sharing a secret key before communication
- RSA
  - Specifications
  - Implementation
    - Primality testing: Fermat's primality test, Miller-Rabin primality test
    - Extended Euclidean algorithm
    - Fast modular exponentiation
  - Security
    - Integer factorization
      - Dixon's Random Squares algorithm
    - Other attacks
      - Short message
      - Shared public key
      - Small public key
      - Small private key ….