Asymmetric Key Cryptography

Haipeng Dai

haipengdai@nju.edu.cn
313 CS Building
Department of Computer Science and Technology
Nanjing University

Problems of Symmetric key Cryptosystems

- In symmetric key cryptosystems, before any ciphertext can be transmitted between two parties, a prior secure transmission of the key k is required.
  - In practice, this is often very difficult to achieve.
- Can we design an asymmetric key cryptosystem such that:
  - An entity has two key: a public key PU_a and a private key PR_a
  - $X = D(PR_a, E(PU_a, X))$ for confidentiality
  - $X = D(PU_a, E(PR_a, X))$ for authentication (non-repudiation)
- The idea of a public-key cryptosystem was proposed by Diffie and Hellman in 1976.
- RSA Cryptosystem was first invented in 1977 by Rivest, Shamir, and Adleman.

Misconceptions Concerning Public-Key Encryption

- Public-key encryption is more secure from cryptanalysis than symmetric encryption
- Public-key encryption is a general-purpose technique that has made symmetric encryption obsolete
- There is a feeling that key distribution is trivial when using public-key encryption, compared to the cumbersome handshaking involved with key distribution centers for symmetric encryption

Principles of Public-Key Cryptosystems

- The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption:

Key distribution

- **How to have secure communications in general without having to trust a KDC with your key**

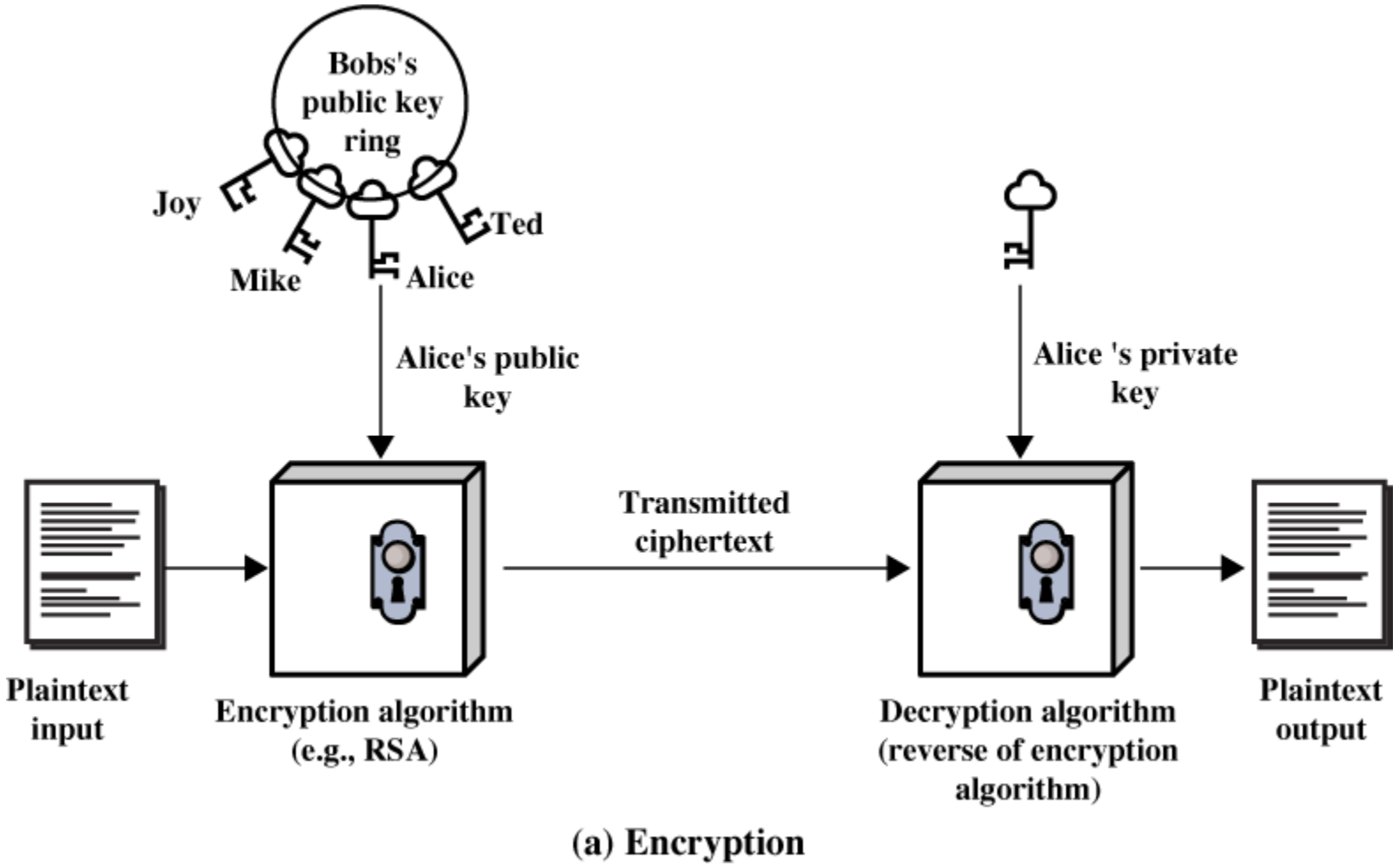
Digital signatures

- **How to verify that a message comes intact from the claimed sender**

- Whitfield Diffie and Martin Hellman from Stanford University achieved a breakthrough in 1976 by coming up with a method that addressed both problems and was radically different from all previous approaches to cryptography

Public-key Cryptography (1/2)

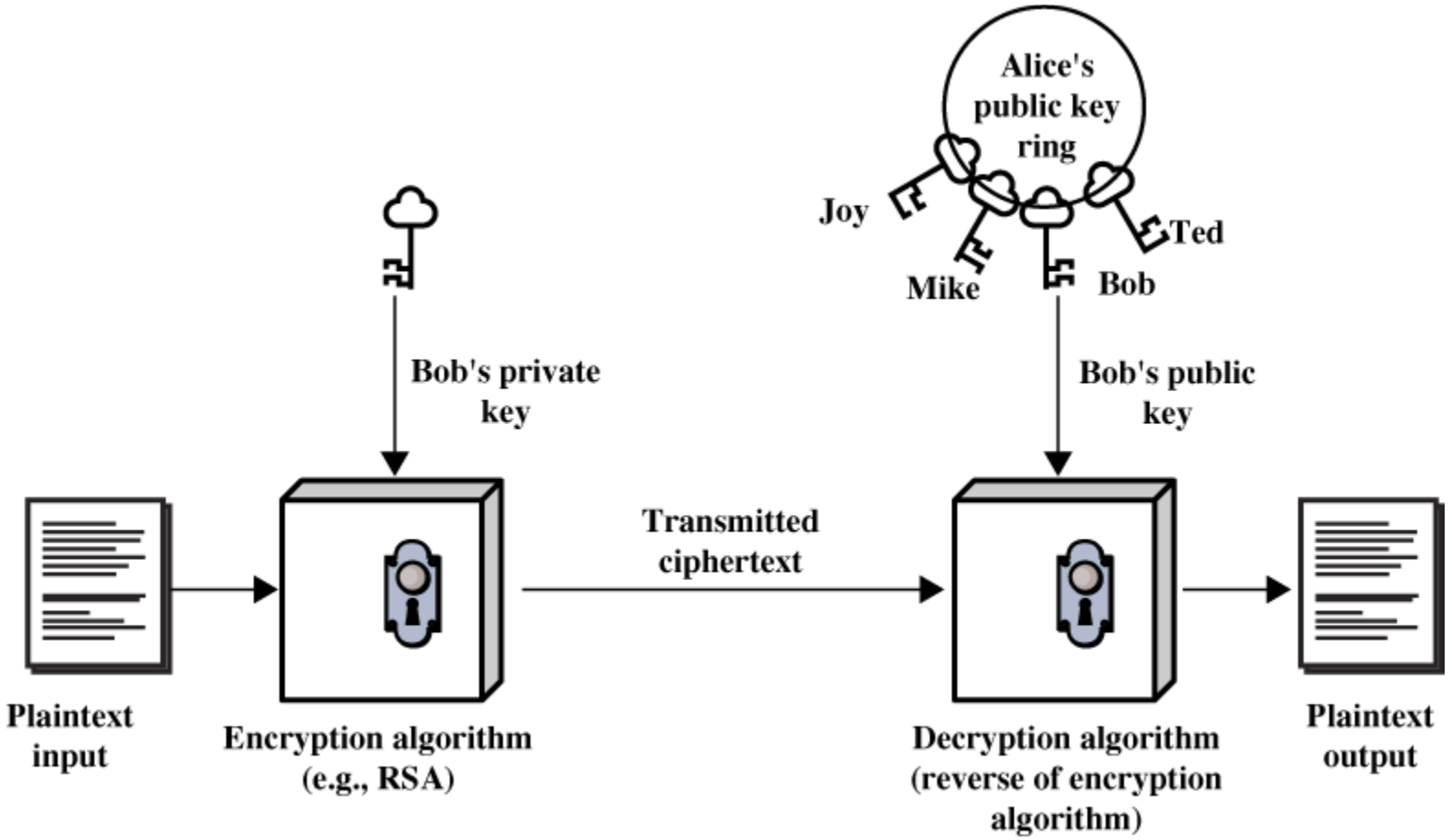
- Encryption



(a) Encryption

Public-key Cryptography (2/2)

- Authentication

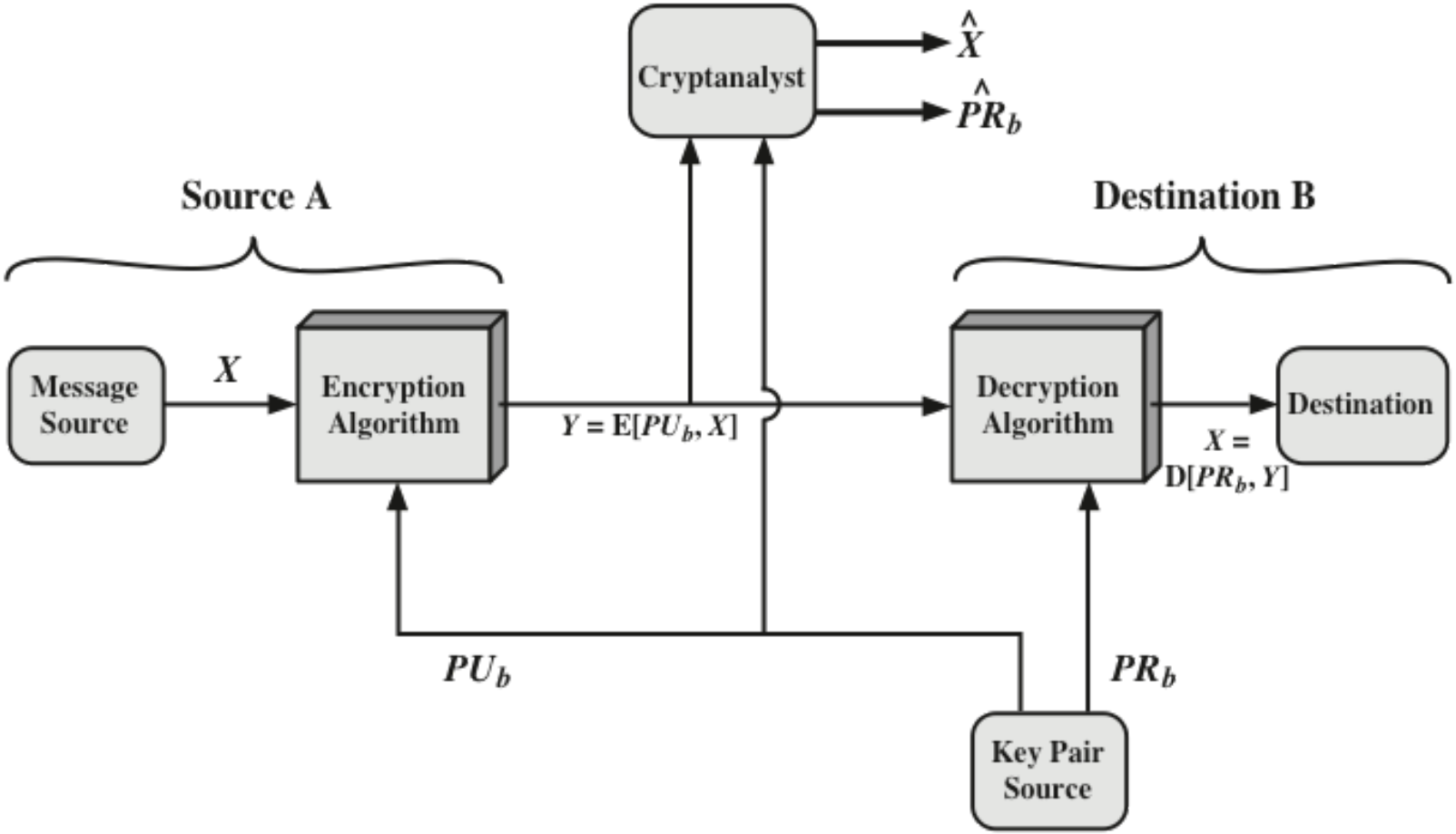


(b) Authentication

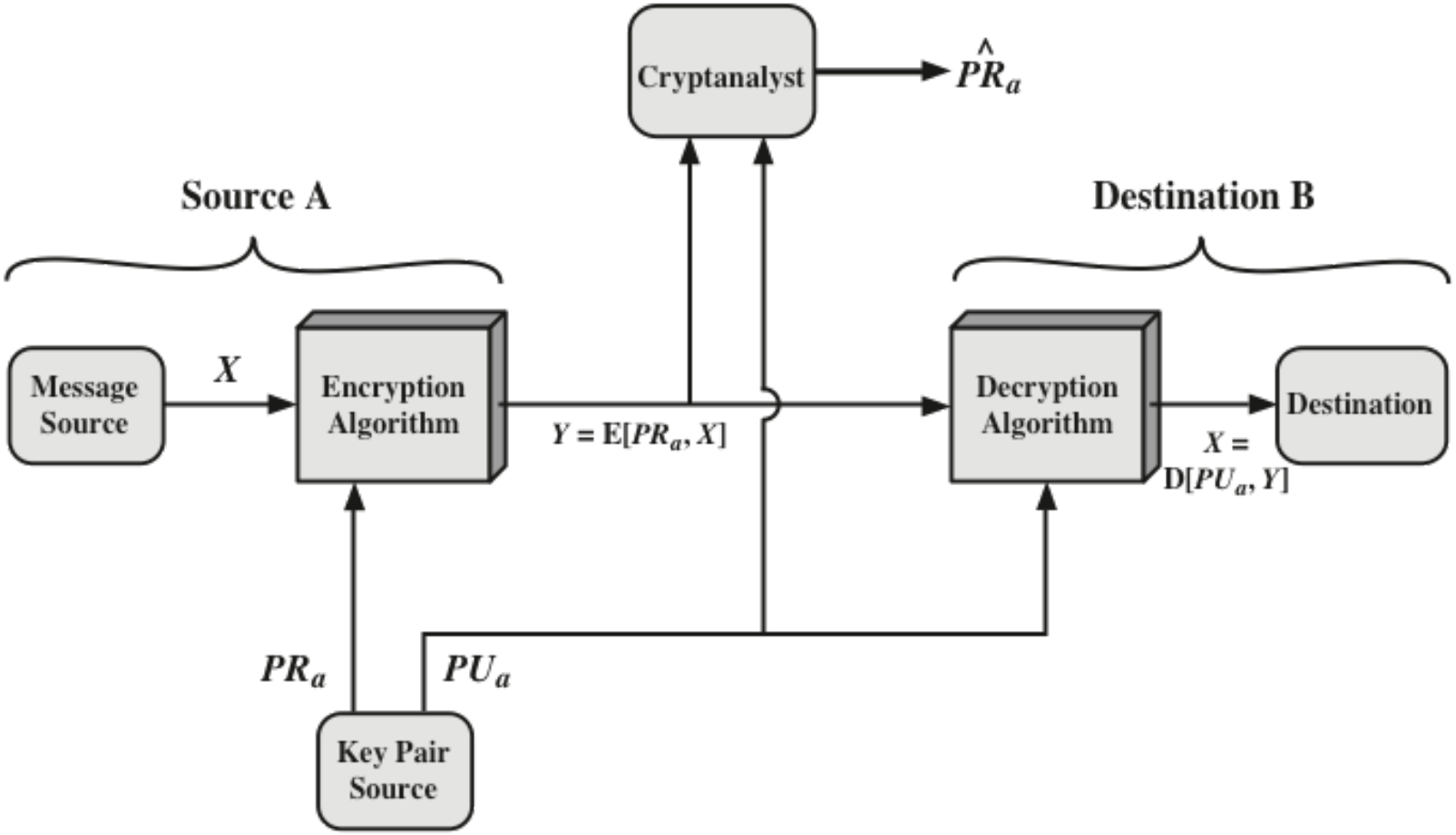
Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
Needed to Work: 1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key. *Needed for Security:* 1. The key must be kept secret. 2. It must be impossible or at least impractical to decipher a message if the key is kept secret. 3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.	*Needed to Work:* 1. One algorithm is used for encryption and a related algorithm for decryption with a pair of keys, one for encryption and one for decryption. 2. The sender and receiver must each have one of the matched pair of keys (not the same one). *Needed for Security:* 1. One of the two keys must be kept secret. 2. It must be impossible or at least impractical to decipher a message if one of the keys is kept secret. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

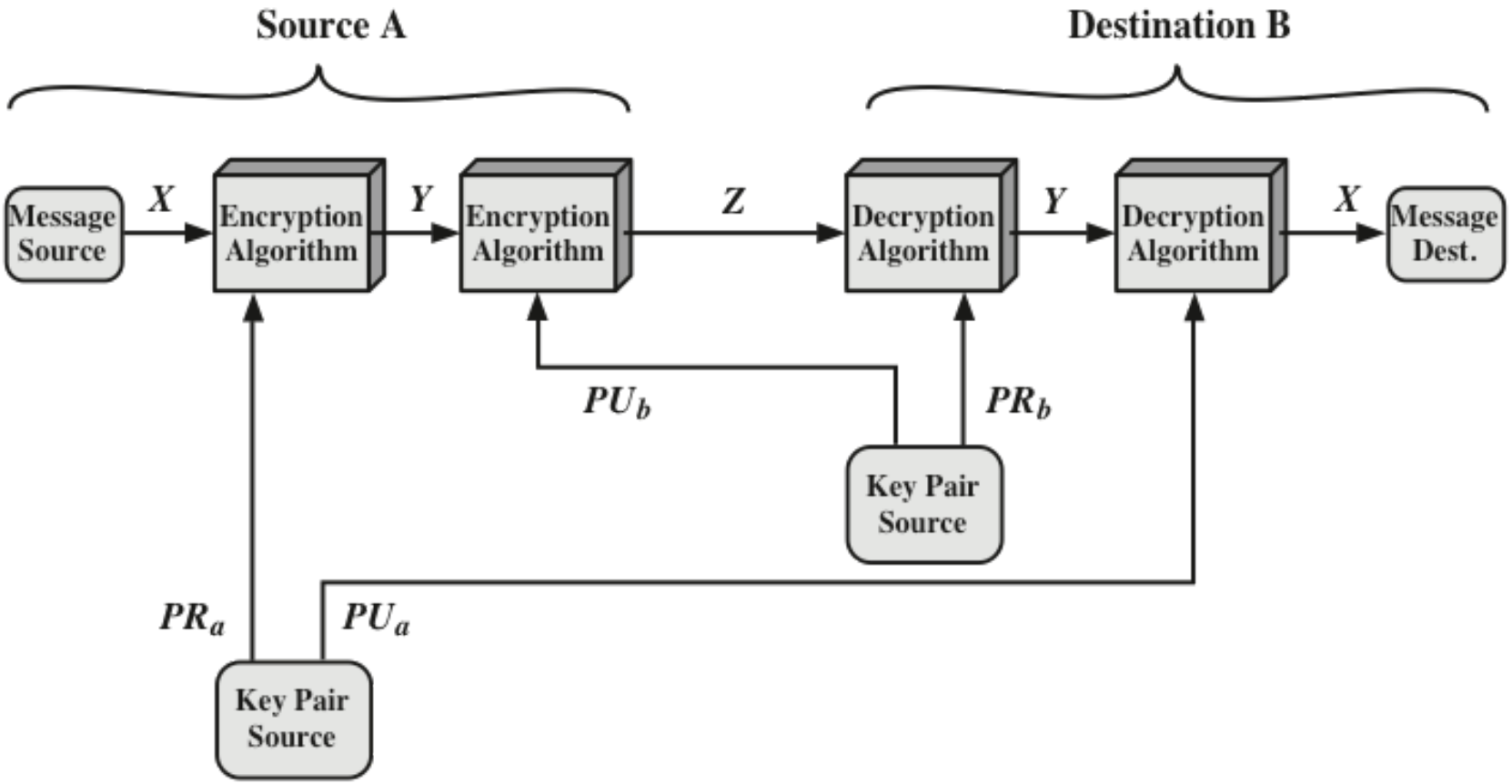
Public-Key Cryptosystem: Secrecy



Public-Key Cryptosystem: Authentication



Public-Key Cryptosystem: Authentication and Secrecy



Public-Key Requirements

- Conditions that these algorithms must fulfill:
  - It is computationally easy for a party B to generate a pair (public-key PU_b, private key PR_b)
  - It is computationally easy for a sender A, knowing the public key and the message to be encrypted, to generate the corresponding ciphertext
  - It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message
  - It is computationally infeasible for an adversary, knowing the public key, to determine the private key
  - It is computationally infeasible for an adversary, knowing the public key and a ciphertext, to recover the original message
  - The two keys can be applied in either order

Public-Key Requirements

- Need a trap-door one-way function
  - A one-way function is one that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy, whereas the calculation of the inverse is infeasible
    - $Y = f(X)$ easy
    - $X = f^{-1}(Y)$ infeasible
- A trap-door one-way function is a family of invertible functions f_k, such that
  - $Y = f_k(X)$ easy, if k and X are known
  - $X = f_k^{-1}(Y)$ easy, if k and Y are known
  - $X = f_k^{-1}(Y)$ infeasible, if Y known but k not known
- A practical public-key scheme depends on a suitable trap-door one-way function

Mathematics in Public-key Cryptosystems

- Most public key algorithms are based on **modular arithmetics**.
- Modular addition

  $(a+b) \bmod n$
- Modular multiplication

  $(a \times b) \bmod n$
- Modular exponentiation

  $a^b \bmod n$

Modular Addition

Addition Modulo 10

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

a's additive inverse is -*a*

Modular Multiplication (1/2)

Multiplication Modulo 10

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

If $(a \times b) \bmod n = 1$, b is the muliplicative inverse of a, denoted as a^{-1}

Given a and n, **Euclid's algorithm** can efficiently find a^{-1}.

Modular Multiplication (2/2)

- What's special about the numbers {1, 3, 7, 9}?
  - They are **relatively prime** to 10.
  - Only they have multiplicative inverse.
  - *a* and 10 do not share any common factors other than 1, i.e. gcd(*a*, 10) = 1

×	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

- How many numbers less than *n* are relatively prime to *n*, denoted as $\varphi(n)$?
  - φ is called the **totient function**.
  - If *n* is prime, all numbers in {1, 2, …, *n*-1} are relatively prime to *n*, i.e., $\varphi(n) = n-1$
  - If *n* is a product of two distinct primes, say *p* and *q*, $\varphi(n) = (p-1)(q-1)$

Proof:

There are $n = pq$ numbers in {0,1, 2, …, *n*-1}.

There are $p+q-1$ numbers in {0, 1, 2, …, *n*-1} are not relatively prime to *n* because

1. the numbers that are either multiples of *p* or of *q* are not relatively prime to *n*, and
2. there are *p* multiples of *q* less or equal than *pq*, and *q* multiples of *p* less than *pq*.

Hence $\varphi(n) = \varphi(pq) = pq - (p+q-1) = (p-1)(q-1)$

Modular Exponentiation (1/2)

Exponentiation Modulo 10

x^y	0	1	2	3	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

Modular Exponentiation (2/2)

- $x^y \bmod n = x^{y \bmod \varphi(n)} \bmod n$ when one of the following conditions holds
  - n is a prime
  - n is a product of distinct primes
  - n does not have p^2 as a factor for any prime p, such n is known as **square free**

- Special case: if $y = 1 \bmod \varphi(n)$, then $x^y = x \bmod n$.

Setting up an RSA Cryptosystem

- RSA key setup
  - Select two large primes p and q at random
  - Compute $n = pq$. Note that $\varphi(n) = (p-1)(q-1)$.
  - Select encryption key e $(1<e<\varphi(n))$ such that e is relatively prime to $\varphi(n)$.
  - Compute the decryption key: $d = e^{-1} \bmod \varphi(n)$ using Euclid's algorithm (i.e., $ed = 1 \bmod \varphi(n)$)
  - Public key: PU = (n, e). Private key: PR = (n, d)
  - Important: p, q, and $\varphi(n)$ must be kept secret. Why?

RSA Encryption and Decryption

- Suppose Bob wants to send a secret message m to Alice
- To encrypt the message m, Bob does the following steps:
  - Obtain Alice's public key $PU_{Alice} = (n, e)$.
  - Encrypt m as $c = m^e \bmod n$.
- To decrypt the ciphertext c, Alice computes $m = c^d \bmod n$, using her private key $PR_{Alice} = (n, d)$.
  - Because $ed = 1 \bmod \varphi(n)$, $c^d = (m^e)^d = m^{ed} = m \bmod n$
  - Here m<n must hold.
- Question: how to prevent message modification attacks?
- Answer: Can be easily prevented by message formatting in both encryption and signatures.
  - See PKCS standards later.

RSA Example: Key Setup

- Select two primes: $p = 17$, $q = 11$.
- Compute the modulus $n = pq = 187$.
- Compute $\varphi(n) = (p-1)(q-1) = 160$.
- Select e satisfying $1<e<160$ and gcd $(e, 160) = 1$.

  Let $e = 7$.
- Compute the decryption key

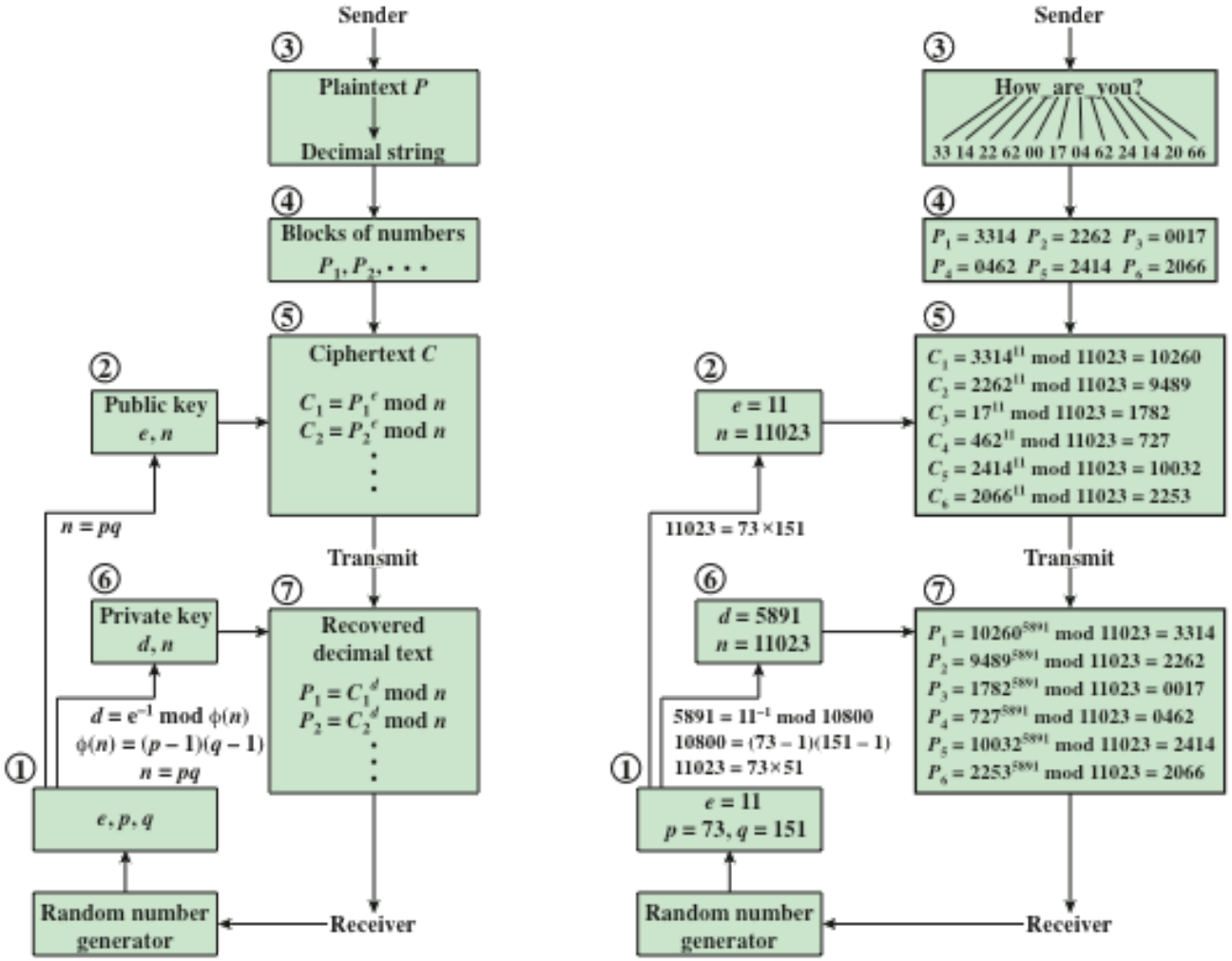
  $d = e^{-1} \bmod \varphi(n) = 7^{-1} \bmod 160 = 23$

  (using Euclid's algorithm)
- Public key: PU = (187, 7).
- Private key: PR = (187, 23).

RSA Example: Encryption and Decryption

- Suppose $m = 88$
- Encryption: $c = m^e \bmod n = 88^7 \bmod 187 = 11$.
- Decryption: $m = c^d \bmod n = 11^{23} \bmod 187 = 88$.
  - We do not first compute 11^{23} and then reduce it modulo 187
  - Instead, we reduce the intermediate results modulo 187 whenever it gets bigger than 187

RSA Processing of Multiple Blocks



Algorithm Requirements

- For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:
  1. It is possible to find values of e, d, n such that $M^{ed} \bmod n = M$ for all $M < n$
  2. It is relatively easy to calculate $M^e \bmod n$ and $C^d \bmod n$ for all values of $M < n$
  3. It is infeasible to determine d given e and n

Why is RSA Secure?

- The real promise behind RSA's security is
  - The assumption that **factoring a big number is hard**
  - The best knowing factoring methods are really slow:
    - To factor a 512-bit number with the best known techniques would take about 30,000 MIPS-years.
    - MIPS-year=# of steps processed for one year at one million instructions per second=31.5 trillion instructions.

Progress in RSA Factorization

Number of Decimal Digits	Number of Bits	Date Achieved
100	332	April 1991
110	365	April 1992
120	398	June 1993
129	428	April 1994
130	431	April 1996
140	465	February 1999
155	512	August 1999
160	530	April 2003
174	576	December 2003
200	663	May 2005
193	640	November 2005
232	768	December 2009

MIPS-Years Needed to Factor

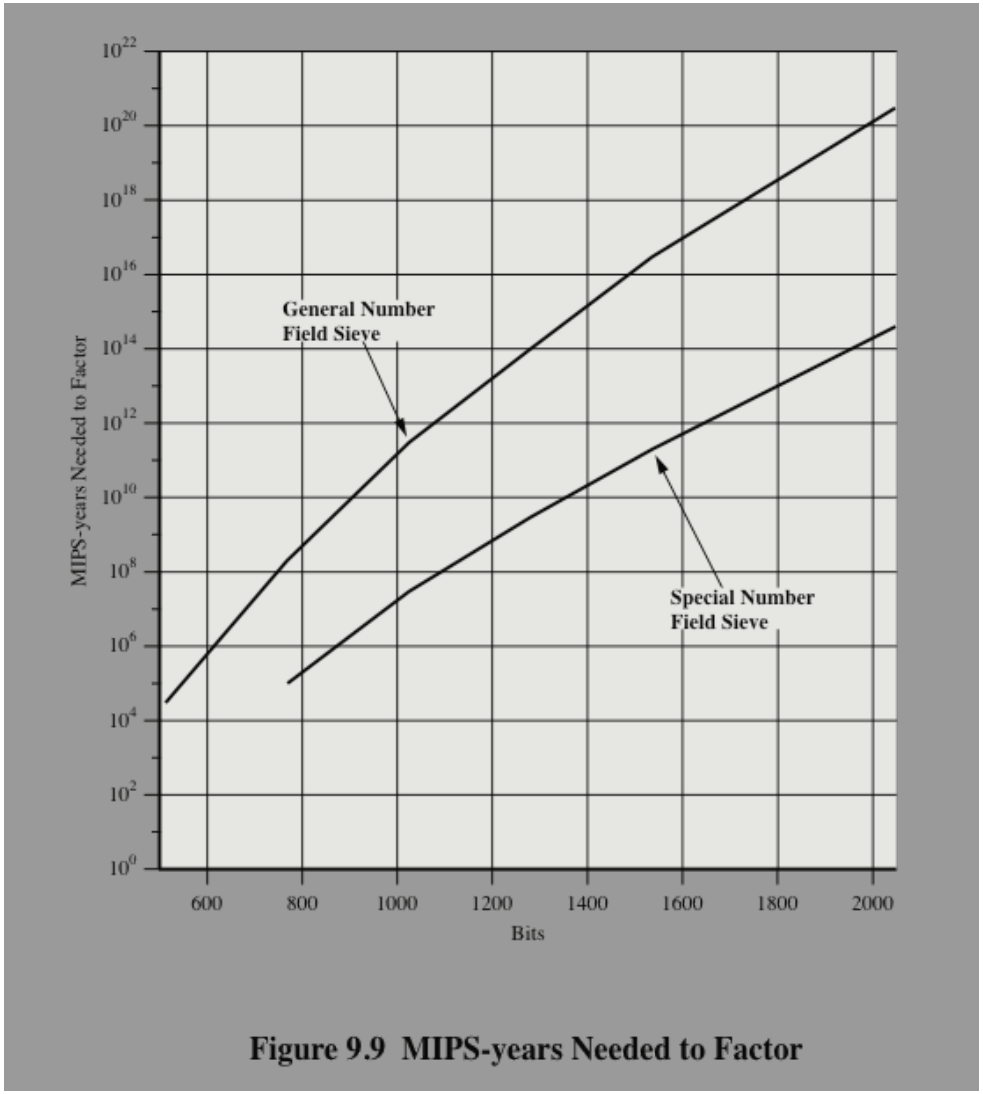


Figure 9.9 MIPS-years Needed to Factor

Exponentiating with big numbers

- Directly computing $x^e \bmod n$ is very slow
- An efficient algorithm is called **square-and-multiply** algorithm

— Let $e = e_k e_{k-1} \ldots e_0$ denote the binary expression of e

— $e = ((((e_k \times 2 + e_{k-1}) \times 2 + e_{k-2}) \times 2 + e_{k-3}) \times 2 \ldots + e_1) \times 2 + e_0$

square-and-multiply algorithm

$z \leftarrow 1$
for $i \leftarrow k$ downto 0 do
    $z \leftarrow z^2 \bmod n$
        $z \leftarrow z \times x^{e_i} \bmod n$
return z

Example: $11^{23} \bmod 187$

$23 = 10111_b$

$z \leftarrow 1$

$z \leftarrow z^2 \cdot 11 \bmod 187 = 11$ (square and multiply)

$z \leftarrow z^2 \bmod 187 = 121$ (square)

$z \leftarrow z^2 \cdot 11 \bmod 187 = 44$ (square and multiply)

$z \leftarrow z^2 \cdot 11 \bmod 187 = 165$ (square and multiply)

$z \leftarrow z^2 \cdot 11 \bmod 187 = 88$ (square and multiply)

Generating RSA Keys (1/3)

- Finding big primes p and q
  - Choose a random number and then test if it is prime
    - The probability of a randomly chosen number n being prime $\approx 1/\ln(n)$
    - **Euler's theorem**:

      For any a relatively prime to n, $a^{\varphi(n)}=1 \bmod n$
    - **Fermat's theorem**:

      If p is a prime and $0<a<p$, $a^{p-1} = 1 \bmod p$. (Note: $\varphi(p) = p-1$)
    - If p is not a prime, the probability that $a^{p-1} = 1 \bmod p$ is $1/10^{13}$.
    - Choose m values for a, we can reduce the false positive ratio to $1/10^{(13*m)}$.

Generating RSA Keys (2/3)

- How to test $a^{p-1} = 1 \bmod p$?
  - For any prime p, we can find $b \geq 1$ and an odd number c such that $p-1 = 2^b c$.
  - Thus, for any prime p, we have $1 = a^{p-1} \bmod p = (a^c \bmod p)^{2^b}$
  - If p is a prime, then for any $1 \leq i \leq b$, $(a^c \bmod p)^{2^i} = \pm 1$
  - **Miller and Rabin Test**: if there exists $1 \leq i \leq b$ such that $(a^c \bmod p)^{2^i} \neq \pm 1$, then p is not a prime.

Generating RSA Keys (3/3)

- Finding d and e
  - Choose any number e that is relatively prime to $\varphi(n)= (p-1)(q-1)$. Two methods:
  1. First choose p and q. Then choose e at random; test whether $\gcd(e, (p-1)(q-1)) = 1$; if not, choose another e.
  2. First choose e. Then select p and q carefully such that $\gcd(e, (p-1)) = 1$ and $\gcd(e, (q-1)) = 1$
     - This is easier because we can choose e to be small.
  - Use Euclid's algorithm to find d such that $ed = 1 \mod \varphi(n)$

Having a small encryption key

- RSA is no less secure (as far as anyone knows) if e is always chosen to be the same number.

- To speed up encryption, small values are usually used for e
  - Encryption and signature verification become very efficient
  - Also it becomes easier to find p and q.

- Popular choices are 3, $17 = 2^4 + 1$, and $65537 = 2^{16} + 1$
  - These values have only two 1's in their binary expression.
  - Save time for computing x^e using the square-and-multiply algorithm (i.e., reduce the number of multiplications).

Low Encryption Exponent Attack

- A message m sent to e users who employ the same encryption exponent e is not protected by RSA.
- Suppose $e = 3$, and Bob sends m to 3 recipients encrypted as $c_1 = m^3 \bmod n_1$, $c_2 = m^3 \bmod n_2$, and $c_3 = m^3 \bmod n_3$
- After intercepting c_1, c_2, and c_3, an attacker can compute message $m^3 = c \bmod n_1 n_2 n_3$ from $m^3 = c_1 \bmod n_1$, $m^3 = c_2 \bmod n_2$, and $m^3 = c_3 \bmod n_3$ using Chinese Remainder Theorem
  - If $m^3 < n_1 n_2 n_3$, then $m^3 = c$, i.e., $m = c^{1/3}$
- Question: How to prevent?
- Answer: include a random number in m

Can we use a small decryption key?

- One may be tempted to use a small d to speed up decryption
- Unfortunately, that is risky.
- Wiener's attack
  - If $d < n^{1/4}/3$ and $p<q<2p$, then d can be computed from (n, e).
  - In practice, the condition $p<q<2p$ often holds.

Message Forgery Attacks

- You sign two messages m_1 and m_2 and get m_1^d, m_2^d.
- Attackers can see m_1^d, m_2^d, and compute $m_1^d \, m_2^d = (m_1 m_2)^d$, which is the signature of $m_1 m_2$.
- Similarly, they can compute the signature of m_1/m_2, $(m_1 m_2)^j$ for any j.
- How to prevent such attacks?
  - Easy: by message formatting.

Message Guessing Attacks

- John and Alice are lovers.
  - When they are happy, the message between them is often “I love you”.
  - When they are unhappy, the message between them is often “I hate you”.

- For encryption, an attacker can guess the message *m* and test each guess because *e* is public.

- How to prevent this attack?
  - Include a random number in the message.

Optimizing RSA Private Key Operations

- In RSA, $d,n \approx 512$ bits, $p,q \approx 256$ bits

- Decryption: $c^d \bmod n$

  Instead of computing $c^d \bmod n$ directly, we do

  - Compute $c_1 = c \bmod p$ and $c_2 = c \bmod q$
  - Compute $m_1 = (c_1)^d \bmod p$ and $m_2 = (c_2)^d \bmod q$
  - Compute $c^d \bmod pq$ using Chinese Remainder Theorem

$$\begin{cases} c^d \equiv m_1 \bmod p \\ c^d \equiv m_2 \bmod q \end{cases}$$

Cube Root Attacks: e=3

- For encryption: attackers can see m^3, they can try to calculate m.
  - This is infeasible when m is large.
  - But if the message m is a small number, calculating m from m^3 is easy.
- For signature: if short messages are padded randomly at least significant bits, attackers can sign any short message x:
  - Legal sender: (m|random bits)^d
  - Legal receiver: (m|random bits)^{d*3}= m|random bits
  - Attacker wants receiver to receive m'|random bits by:
    - 1. Let x=m'|zeros.
    - 2. Compute the cube root of x.
    - 3. Round the root to the nearest integer r.
    - 4. Send r to the legal receiver.
    - When receiver receives it, compute r^3= m'|random bits.
- Question: how to prevent?
- Answer: by proper message formatting.

Public-Key Cryptography Standard (PKCS)

- PKCS standards define the encodings for things such as
  - An RSA public key
  - An RSA private key
  - An RSA signature
  - A short RSA-encrypted message (typically a secret key)
  - A short RSA-signed message (typically a message digest)
  - Password-based encryption

- Encryption

PKCS#1 defines a standard for formatting a message to be encrypted with RSA

To encrypt a message *m*:

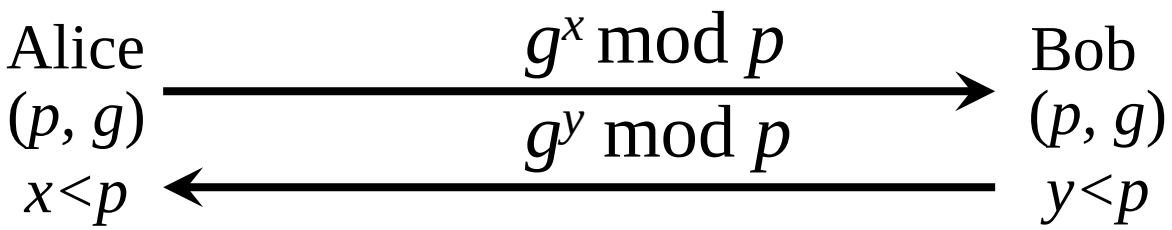
0	2	At least eight random nonzero octets (*r*)	0	Message (*m*)

- Pad *m* so that *m'* = 0|2|*r*|0|*m*
- The ciphertext $c = \text{RSA}(m') = (m')^e \bmod n$
- 2 indicates encryption and1 indicates signature.
- The first byte 0 guarantees that m'<n. Note that the first bit of n must be 1.
- The second 0 delimit the random numbers and the message m.
- The random numbers prevent message guessing attacks.
- The formatting prevents message modification attacks.
- Padding being on the left side prevents the cube root attacks for both encryption and signature.

Diffie-Hellman Key Exchange

- It is the oldest public-key system still in use
- It is less general than RSA
  - Neither support encryption
  - Nor support signature
- Diffie-Hellman allows two individuals to agree on a shared key
  - Even though they can only exchange messages in public

Suppose (p, g) are publicly known, where p is a large prime and $g<p$



- Both parties compute the shared key

  $K=g^{xy} \bmod p = (g^x)^y \bmod p = (g^y)^x \bmod p$
- Assumption: infeasible to calculate discrete logarithms for large primes.

Diffie-Hellman Example

- Users Alice & Bob who wish to swap keys:
- Agree on prime p=353 and g=3
- Select random secret keys:
  - A chooses x=97, B chooses y=233
- Compute respective public keys:
  - $T_A = g^x = 3^{97} \bmod 353 = 40$ (Alice)
  - $T_B = g^y = 3^{233} \bmod 353 = 248$ (Bob)
- Compute shared session key as:
  - $K = T_B^{\ x} \bmod 353 = 248^{97} \bmod 353 = 160$ (Alice)
  - $K = T_A^{\ y} \bmod 353 = 40^{233} \bmod 353 = 160$ (Bob)

Homework

- Textbook Chapter 9, problems:
  - 9.2(c)(e), 9.10, 9.13 and 9.15.