CS 4920/5920 Applied Cryptography

Chapter 9 Public Key Cryptography and RSA

Secret-Key Cryptography

- traditional **symmetric/secret/single key** cryptography uses **one** key
- shared by both sender and receiver
- if this key is disclosed, communications are compromised
- also is **symmetric**, parties are equal
  - does not protect sender from receiver forging a message & claiming is sent by sender
  - does not protect receiver from sender denying sending a message

Public-Key Cryptography

- significant advance in the 3000 year history of cryptography
- uses **two** keys – a public key & a private key
- **asymmetric** since parties are not equal
- uses clever application of number theoretic concepts to function
- complements rather than replaces private key cryptography

Misconceptions about Public-Key Cryptography

- public-key encryption is more secure
  - there is nothing in principle to show one is superior to another from the point of view of resisting cryptanalysis.
- public-key encryption is a general-purpose technique that has made symmetric encryption obsolete.
  - because of the computational overhead of current public-key encryption schemes, symmetric encryption is more popular
- key distribution is trivial in public-key encryption
  - the procedures involved are not simpler nor any more efficient than those required for symmetric encryption

Why Public-Key Cryptography?

- developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a third party (e.g., KDC) with your key
  - **digital signatures** – how to protect two parties against each other
- public invention due to Whitfield Diffie & Martin Hellman at Stanford Univ. in 1976
  - known earlier in classified community

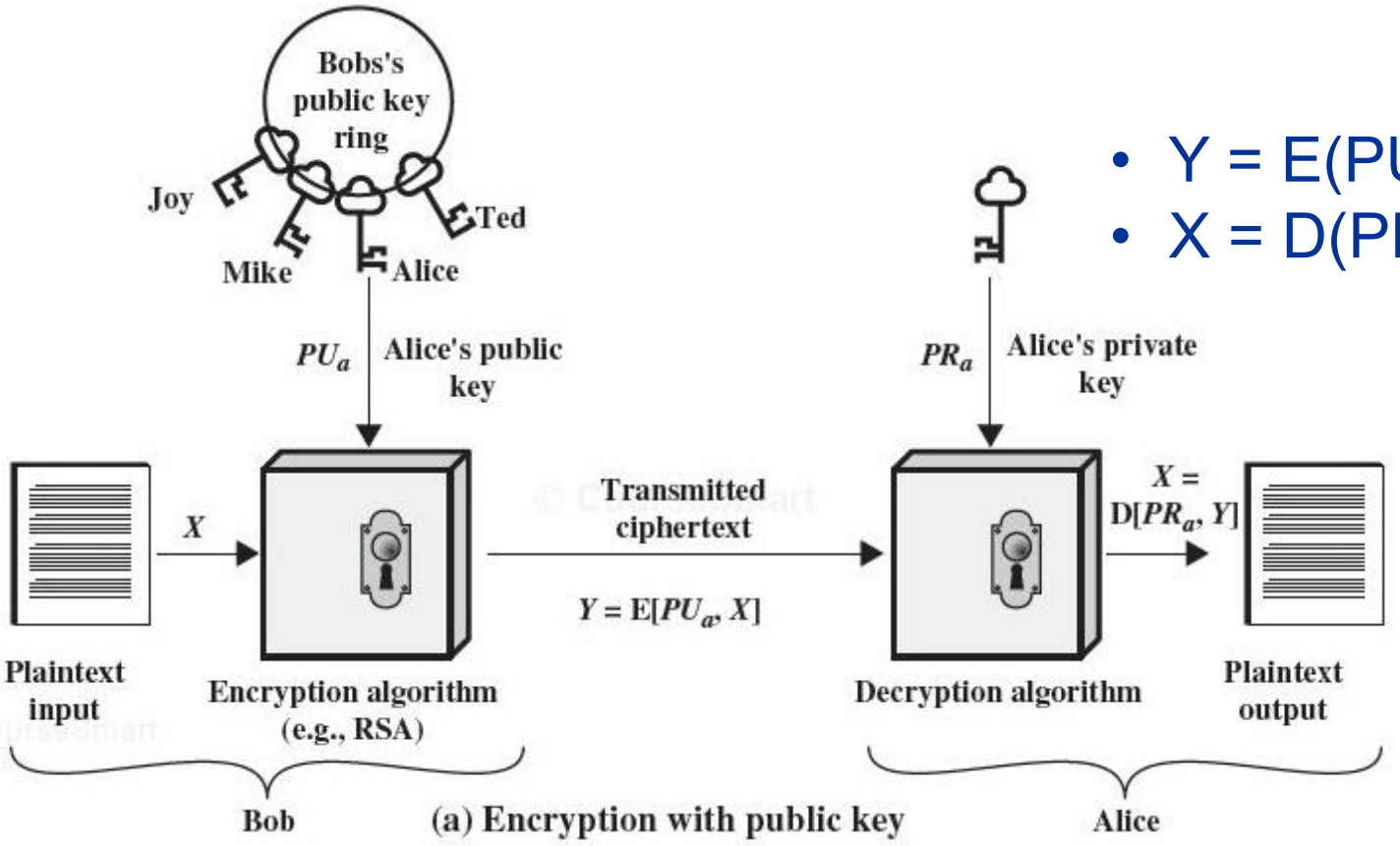
Public-Key Cryptography

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt** messages (provide confidentiality), or **decrypt** messages (verify signatures)
  - a related **private-key**, known only to one party, used to **decrypt** messages (get the plaintext), or **encrypt** messages (create signatures**)**
- **infeasible to determine private key from public key**
- is **asymmetric** because
  - different keys are used for encryption and decryption

Symmetric vs Public-Key

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 no other information is available. 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 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 no other information is available. 3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.

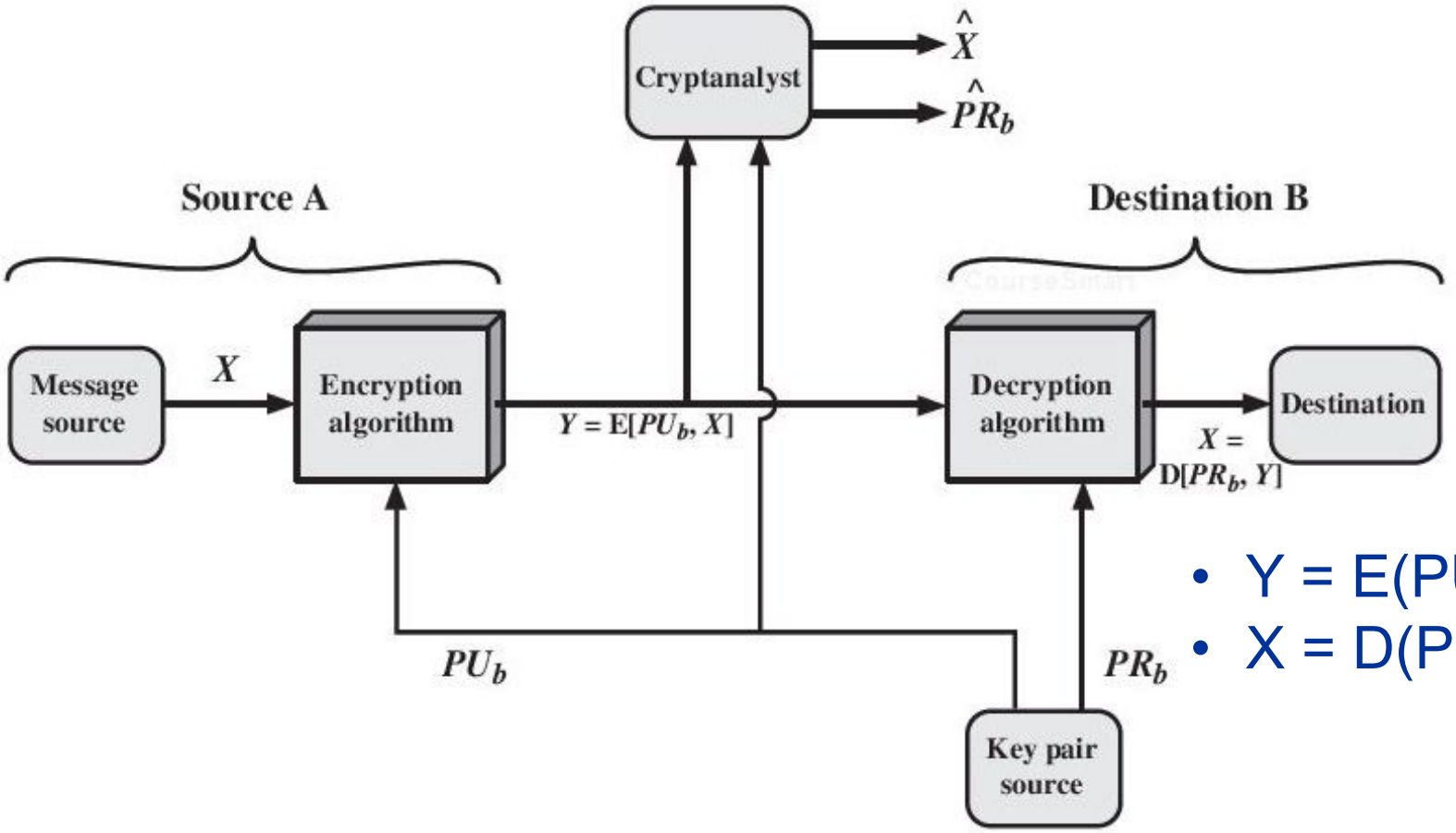
Encryption with Public Key



- $Y = E(PU_a, X)$
- $X = D(PR_a, Y)$

(provides confidentiality)

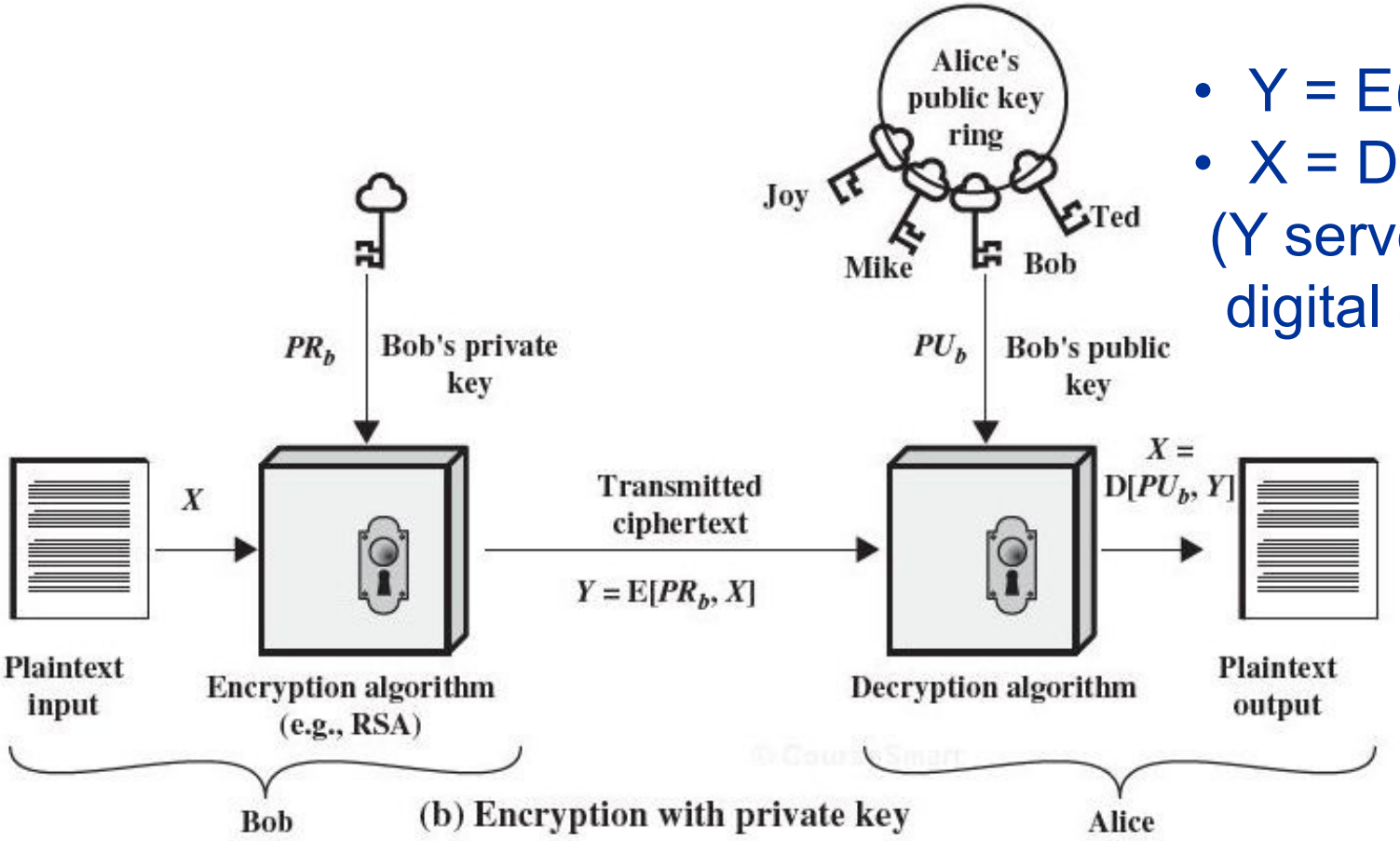
Encryption with Public Key (general form)



- $Y = E(PU_b, X)$
- $X = D(PR_b, Y)$

(provides confidentiality)

Encryption with Private Key



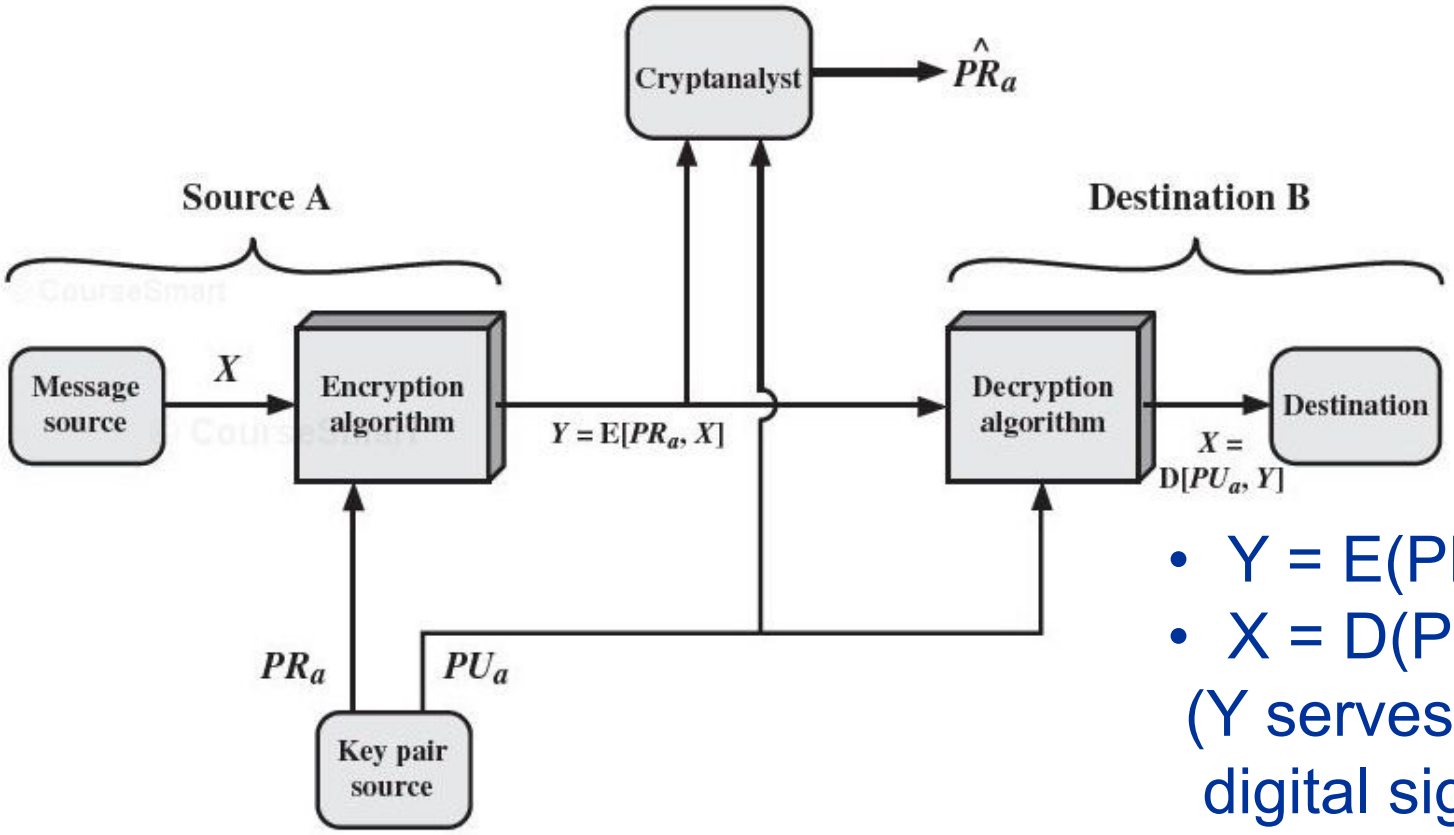
(b) Encryption with private key

- $Y = E(PR_b, X)$
- $X = D(PU_b, Y)$

(Y serves as a digital signature)

(provides authentication: source and the integrity of the message)

Encryption with Private Key (general form)

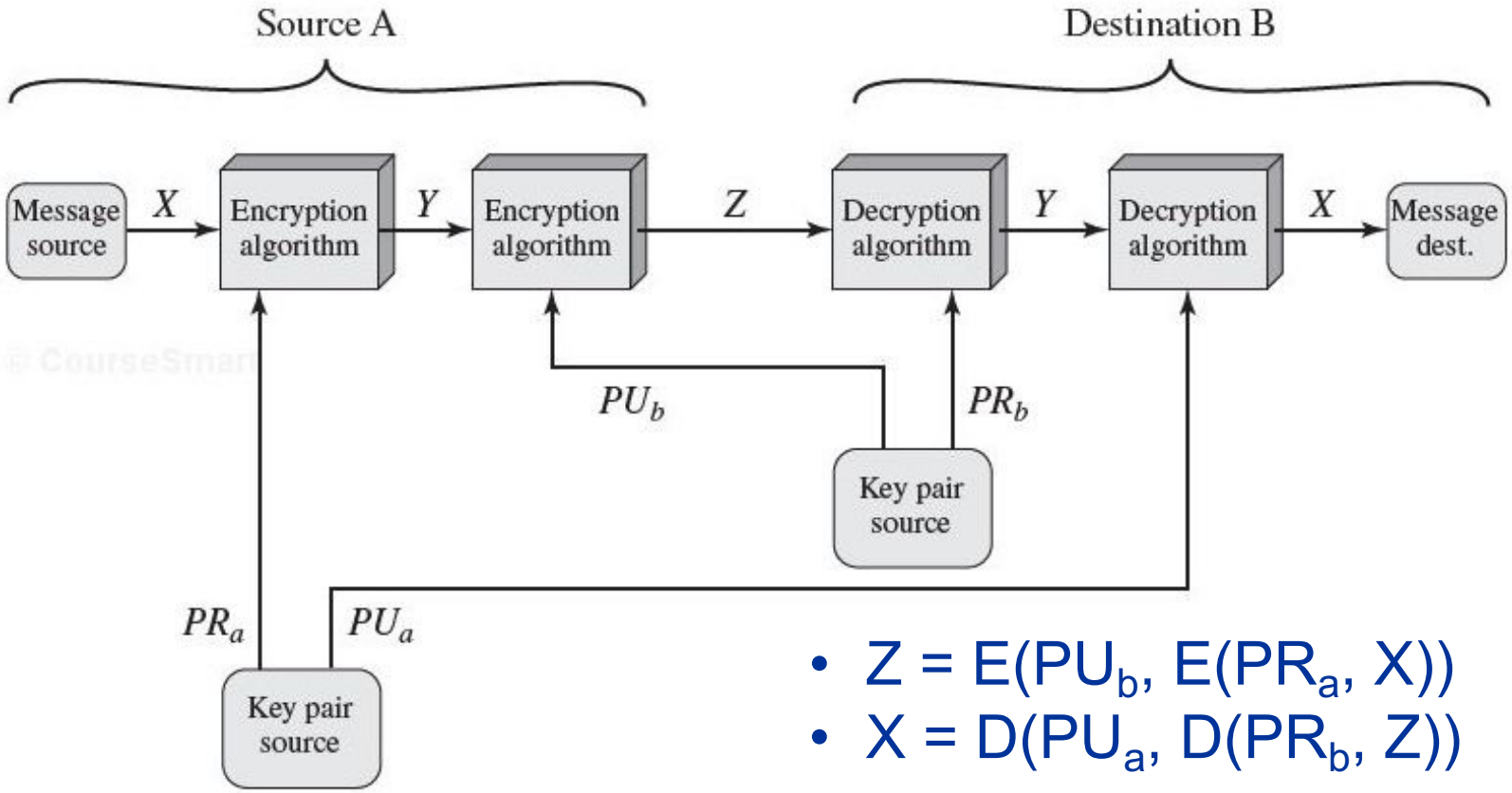


- $Y = E(PR_a, X)$
- $X = D(PU_a, Y)$

(Y serves as a digital signature)

(provides authentication: source and the integrity of the message)

Double Use of Public-key Scheme



- $Z = E(PU_b, E(PR_a, X))$
- $X = D(PU_a, D(PR_b, Z))$

(provides both confidentiality and authentication, not used in practice due to high computational cost)

Applications for Public-Key Algorithms

- can classify uses into 3 categories:
  - **encryption/decryption** (provide confidentiality)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- some algorithms are suitable for all uses, others are specific to one

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Public-Key Cryptography Requirements [DIFF76b]

1. It is computationally easy for a party B to generate a pair (public key PU_b, private key PR_b).
2. It is computationally easy for a sender A, knowing the public key and the message to be encrypted, *M*, to generate the corresponding ciphertext: $C = E(PU_b, M)$
3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using the private key to recover the original message: $M = D(PR_b, C) = D[PR_b, E(PU_b, M)]$
4. It is computationally infeasible for an adversary, knowing the public key, PU_b, to determine the private key, PR_b
5. It is computationally infeasible for an adversary, knowing the public key, PU_b, and a ciphertext, C, to recover the original message, M.

Public-Key Cryptography Requirements (cont.)

6. The two keys can be applied in either order:
   - $M = D[PU_b , E(PR_b, M)] = D[PR_b, E(PU_b, M)]$
   - a useful requirement, not necessary for all algorithms
   - RSA meets this requirement

These are formidable requirements which only a few algorithms have satisfied.

Public-Key Requirements

- need a *trapdoor one-way function*
- one-way function has
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- a trap-door one-way function has
  - $Y = f_k(X)$ 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
- development of a practical public-key scheme depends on discovery of a suitable trap-door one-way function

Public-key Cryptanalysis

- like secret key schemes **brute force** exhaustive key search attack is always theoretically possible
  - a tradeoff on key size

- Computing the private key given the public key could be feasible
  - it has not been mathematically proven this type of attack is infeasible for a particular public-key algorithm (computational hardness assumption => computationally secure as long as assumption holds)
  - any given algorithm, including RSA, is suspect

RSA

- by Rivest, Shamir & Adleman of MIT in 1977
- best known & widely used public-key scheme
- a block cipher, the plaintext and ciphertext are integers between 0 and n-1 for some n
  - a typical size for n is 1024 bits or 2^1024
  - (longer 2048 or 4096 bits can be used for greater security)
- security is based on the difficulty of finding the prime factors of a large composite number
- the inventors: https://www.youtube.com/watch?v=bQ8NR1Vx4e8
- the cipher overview: https://www.youtube.com/watch?v=b57zGAkNKIc

RSA En/decryption

- to encrypt a message M the sender:
  - obtains **public key** of recipient $PU=\{e,n\}$
  - computes: $C = M^e \bmod n$, **where** $0 \leq M < n$
- to decrypt the ciphertext C the owner:
  - uses the private key $PR=\{d,n\}$
  - computes: $M = C^d \bmod n$

$$= (M^e \bmod n)^d \bmod n$$

$$= (M^e)^d \bmod n$$

$$= M^{ed} \bmod n$$

RSA Key Setup

- each user generates a public/private key pair by:
  selecting two large primes at random: p, q
- computing $n=p.q$
  - note $\varnothing(n)=(p-1)(q-1)$ // Euler Totient Function
- selecting at random the encryption key e
  - where $1<e<\varnothing(n)$, $\gcd(e,\varnothing(n))=1$
- solve following equation to find decryption key d
  - $e.d \equiv 1 \bmod \varnothing(n)$ and $0 \leq d \leq n$
- publish the public encryption key: PU={e,n}
- keep secret private decryption key: PR={d,n}

Why RSA Works

- because of Euler's Theorem:
  - $a^{\phi(n)} \bmod n = 1$ where $\gcd(a,n)=1$
- in RSA have:
  - $n=p.q$
  - $\phi(n)=(p-1)(q-1)$
  - carefully chose e & d to be inverses $\bmod \phi(n)$
  - hence $e.d=1+k.\phi(n)$ for some k
- Proof:

$$M^{e.d} \bmod n = M^{1+k.\phi(n)} \bmod n = M \bmod n = M$$

  - $M^{1+k.\phi(n)} \bmod p = M \bmod p \rightarrow p \mid (M^{1+k.\phi(n)} - M)$
  - $M^{1+k.\phi(n)} \bmod q = M \bmod q \rightarrow q \mid (M^{1+k.\phi(n)} - M)$
  - $n \mid (M^{1+k.\phi(n)} - M) \rightarrow M^{1+k.\phi(n)} \bmod n = M \bmod n$

Proof of $M^{1+k.\phi(n)} \bmod p = M \bmod p$

First we show that $M^{k(p-1)(q-1)+1} \bmod p = M \bmod p$. There are two cases to consider.

Case 1: M and p are not relatively prime; that is, p divides M. In this case, $M \bmod p = 0$ and therefore $M^{k(p-1)(q-1)+1} \bmod p = 0$. Thus, $M^{k(p-1)(q-1)+1} \bmod p = M \bmod p$.

Case 2: If M and p are relatively prime, by Euler's theorem, $M^{\phi(p)} \bmod p = 1$. We proceed as

$$
\begin{aligned}
M^{k(p-1)(q-1)+1} \bmod p &= [(M)M^{k(p-1)(q-1)1}] \bmod p \\
&= [(M)(M^{(p-1)})^{k(q-1)}] \bmod p \\
&= [(M)(M^{\phi(p)})^{k(q-1)}] \bmod p \\
&= (M \bmod p) \times [(M^{\phi(p)}) \bmod p]^{k(q-1)} \\
&= (M \bmod p) \times (1)^{k(q-1)} \qquad \text{(by Euler's theorem)} \\
&= M \bmod p
\end{aligned}
$$

Quiz 12a

RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$
2. Calculate $n = pq$
3. Calculate $\varnothing(n)$
4. Select e: $\gcd(e,\varnothing(n))=1$; choose $e=7$
5. Determine d: $d.e \equiv 1 \mod \varnothing(n)$ and $d < \varnothing(n)$
   And show $d.e \equiv 1 \mod \varnothing(n)$
   (extended Euclid's algorithm $ax+by=1$)
6. Publish public key $PU=\{e,n\}$
7. Keep secret private key $PR=\{d,n\}$

Quiz 12a

RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\phi(n) = (p-1)(q-1) = 16*10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $d.e \equiv 1 \mod 160$ and $d < 160$
   Value is d=23 since $23*7=161= 1*160+1$
   (extended Euclid's algorithm $ax+by=1$)
6. Publish public key PU={7,187}
7. Keep secret private key PR={23,187}

Quiz 12a

RSA Example - Key Setup

1. Select primes: $p=17$ & $q=11$ Keep p, q private
2. Calculate $n = pq = 17 \times 11 = 187$
3. Calculate $\phi(n) = (p-1)(q-1) = 16*10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $d.e \equiv 1 \mod 160$ and $d < 160$
   Value is $d=23$ since $23*7=161= 1*160+1$
   (extended Euclid's algorithm $ax+by=1$)
6. Publish public key $PU=\{7,187\}$
7. Keep secret private key $PR=\{23,187\}$

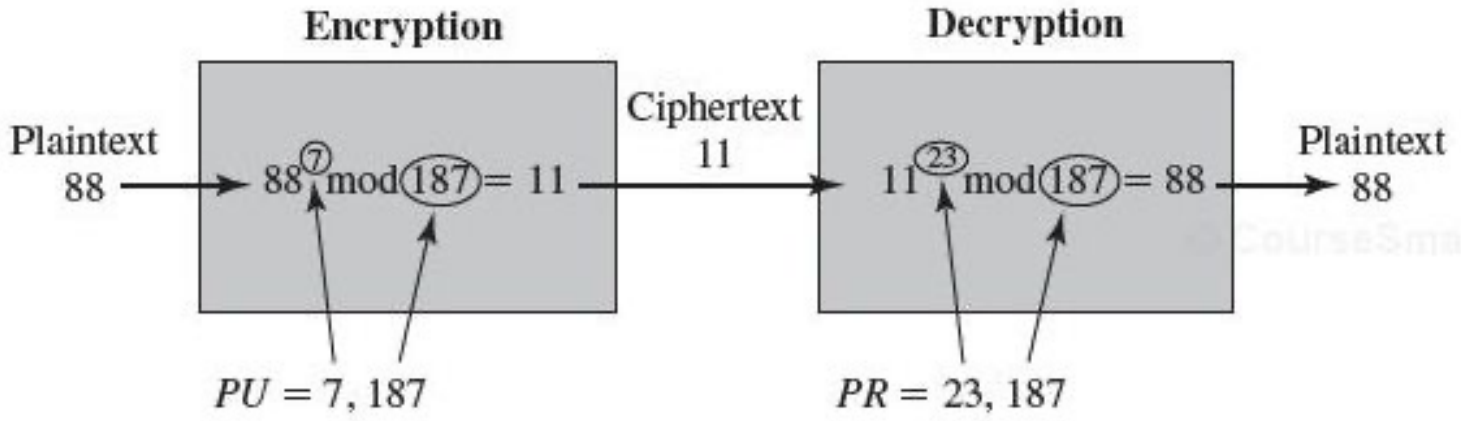
Quiz 12b

RSA Example - En/Decryption

- sample RSA encryption/decryption
  - PU={7,187}, PR={23,187}
- given message M = 88 (note 88<187)
- encryption: C ?
- decryption: M ?

RSA Example - En/Decryption

- sample RSA encryption/decryption
  - PU={7,187}, PR={23,187}
- given message M = 88 (note 88<187)
- encryption: $C = 88^{7} \bmod 187 = 11$
- decryption: $M = 11^{23} \bmod 187 = 88$



Exponentiation in Modular Arithmetic

- can use the *Square and Multiply Algorithm*
  - a fast, efficient algorithm for computing: $a^b \bmod n$
  - $[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$
  - look at binary representation of exponent b
- example:
  - $a^{11} \bmod n$

    $= (a^8 * a^2 * a^1) \bmod n$

    $= [(a^8 \bmod n) * (a^2 \bmod n) * (a^1 \bmod n)] \bmod n$

Square and Multiply Algorithm: $a^b \bmod n$ (b is in binary format: $b_k b_{k-1} \ldots b_0$)

```
c = 0; f = 1
for i = k downto 0
    do   c = 2 x c
         f = (f x f) mod n
    if b_i == 1 then
         c = c + 1
         f = (f x a) mod n
 return f
```

Quiz 12c

e.g., $a^{11} \bmod n$

b = 1011

k =3

f ?

c ?

Square and Multiply Algorithm: $a^b \bmod n$ (b is in binary format: $b_k b_{k-1} \ldots b_0$)

```
c = 0; f = 1
for i = k downto 0
    do   c = 2 x c
         f = (f x f) mod n
    if b_i == 1 then
         c = c + 1
         f = (f x a) mod n
 return f
```

Quiz 12c

e.g., $a^{11} \bmod n$

b = 1011

k =3

f will be $a^{11} \bmod n$

c will be 11

Square and Multiply Algorithm: a^b mod n (b is in binary format: $b_k b_{k-1} \ldots b_0$)

```
c = 0; f = 1
for i = k downto 0
    do   c = 2 x c
         f = (f x f) mod n
    if b_i == 1 then
         c = c + 1
         f = (f x a) mod n
return f
```

Quiz 12c

e.g., a^{11} mod n
b = 1011
k =3

f will be a^{11} mod n
c will be 11

Additional computation if b_i==1

Efficient Encryption

- encryption $C = M^e \bmod n$, faster for smaller e
  - often choose e=65537 ($2^{16}+1$)
  - also see choices of e=3 or e=17
- but if e too small (e.g., e=3), vulnerable to attack
- if e is selected first, to ensure $\gcd(e, \varnothing(n)) = 1$
  - may need to select new p and q

Efficient Decryption

- decryption $M = C^d \bmod n$, faster for smaller d
  - small d is vulnerable to brute force and other attacks
- can use large d, but use Chinese Remainder Theorem (CRT) to speed up
  - compute mod p & mod q separately. then combine to get desired answer
  - approximately 4 times faster than doing directly
- only owner of private key who knows values of p & q can use this technique

RSA Key Generation

- users of RSA must:
  - determine two primes at random `p, q`
  - select either `e` or `d` and compute the other
- `p,q` must not be easily derived from `n=p.q`
  - means `p,q` must be sufficiently large
  - typically guess and use probabilistic test (e.g., Miller Rabin algorithm); about 0.5 ln(N) trials
- gcd(e,ø(n))=1,and e.d ≡ 1 mod ø(n)
  - `extended Euclidean algorithm`
  - Prob(two random numbers are relatively prime): 0.6

RSA Security

- possible approaches to attacking RSA are:
  - brute force key search - infeasible given size of numbers
  - mathematical attacks - based on difficulty of factoring n to primes p and q
  - timing attacks - on running of decryption

Factoring Problem

- mathematical approach takes 3 forms:
  - factor $n=p.q$, hence compute $\varnothing(n)$ and then d
  - determine $\varnothing(n)$ directly and compute d
  - find d directly
- currently believe all equivalent to factoring n
  - have seen slow improvements over the years
    - as of May-2005, key size (length of n) 663-bit was achieved with LS
  - biggest improvement comes from improved algorithm
    - Quadratic sieve, Generalized number field sieve, lattice sieve (LS)
    - currently assume 1024-2048 bit RSA is secure
    - ensure p, q of similar size and matching other constraints, e.g., both (p-1) and (q-1) should contain a large prime factor., GCD(p-1, q-1) should be small

Progress in Factoring

Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve
160	530	April 2003	—	Lattice sieve
174	576	December 2003	—	Lattice sieve
200	663	May 2005	—	Lattice sieve

It has not been shown mathematically that
such factorization into primes is necessarily difficult.

“Factoring could turn out to be easy.” - Rivest

Timing Attacks

- developed by Paul Kocher in mid-1990's
- exploit timing variations in operations
  - e.g., multiplying by small vs large number
- infer operand size based on time taken
- for RSA, exploits time taken in exponentiation
- countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Summary

- principles of public-key cryptography
- RSA algorithm, implementation, and security