Cryptography

RSA
Cryptography and Network Security William Stallings

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
The Chinese Remainder Theorem
Computational Aspects
The Security of RSA

Introduction

- Public key algorithms = asymmetric algorithms
- Public key cryptography relies on number theory
- **Misconception**: asymmetric crypto is more secure than symmetric crypto
  - The security relies on the the length of the key and the computational work to break
- **Misconception**: asymmetric crypto made symmetric crypto obsolete
  - Symmetric crypto is still in use, sometimes preferable over the asymmetric crypto
- **Misconception**: it’s easy to distribute the keys of public key crypto
  - Some protocols are still needed involving a central agent

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
The Chinese Remainder Theorem
Computational Aspects
The Security of RSA

Principles of Public-Key Cryptosystems

- Public key cryptography evolved to solve two issues:

1. Key distribution under symmetric encryption
    - The communicants already share a key
    - Or the use of a key distribution center
2. Digital signatures:
    - Signing electronic documents

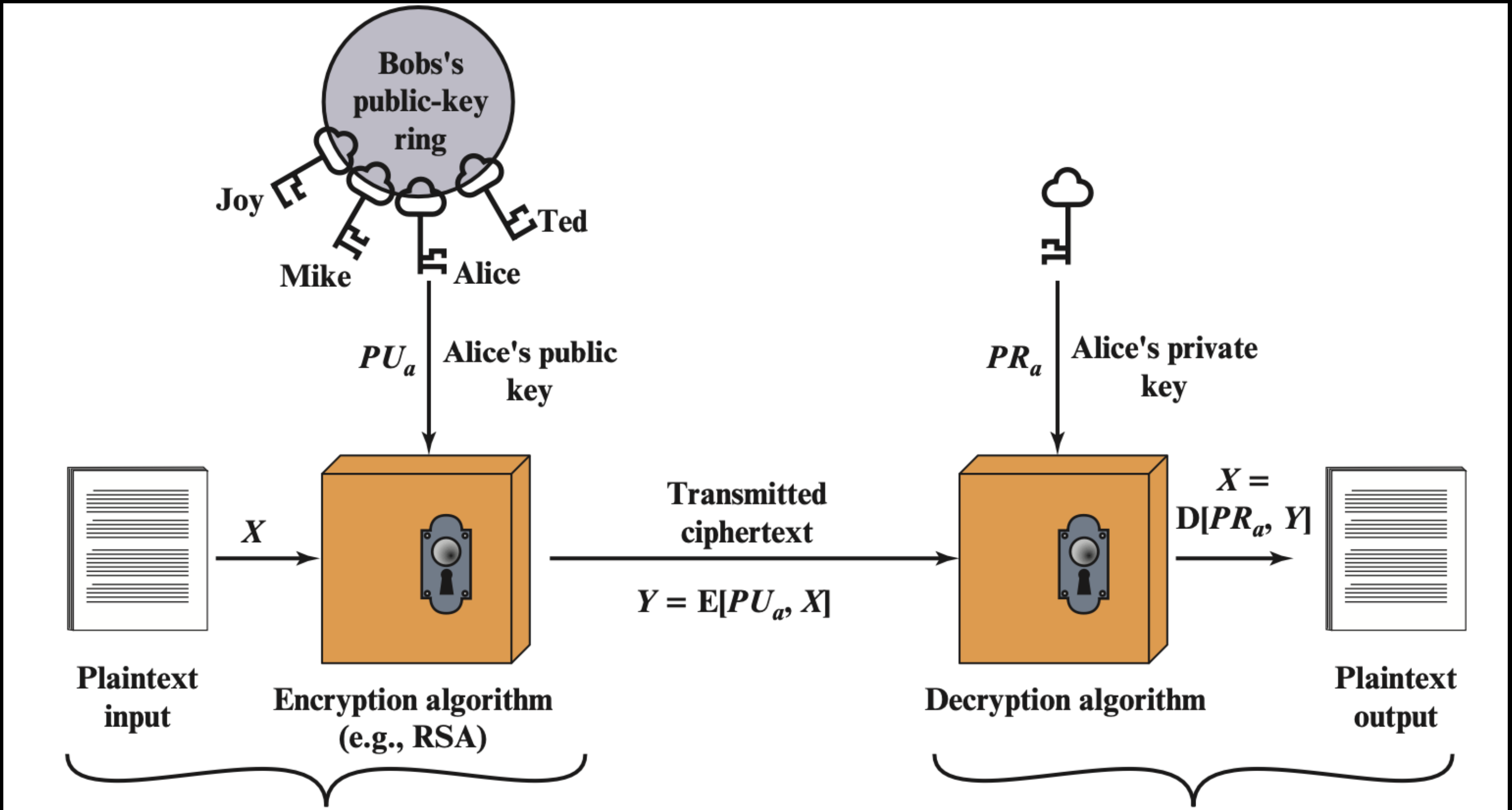
Principles of Public-Key Cryptosystems

- Asymmetric crypto uses two keys:
  - One for encryption
  - Another different **but related** key for decryption

- Characteristics:
  - Cannot compute the decryption key given only the algorithm and the encryption key
  - Either of the two keys can be used for encryption, with the other used for decryption
    - RSA exhibits this characteristic

Principles of Public-Key Cryptosystems

- A public-key encryption scheme has six ingredients:
  - Plaintext
  - Encryption algorithm
  - Public and private keys
  - Ciphertext
  - Decryption algorithm



(a) Encryption with public key

Principles of Public-Key Cryptosystems

- Basic steps

Key generation

Publish public key
Keep private key

Bob sends to Alice:
Bob encrypts the message using Alice's PK

Alice decrypts it using her private key

Any user can update his key pair:

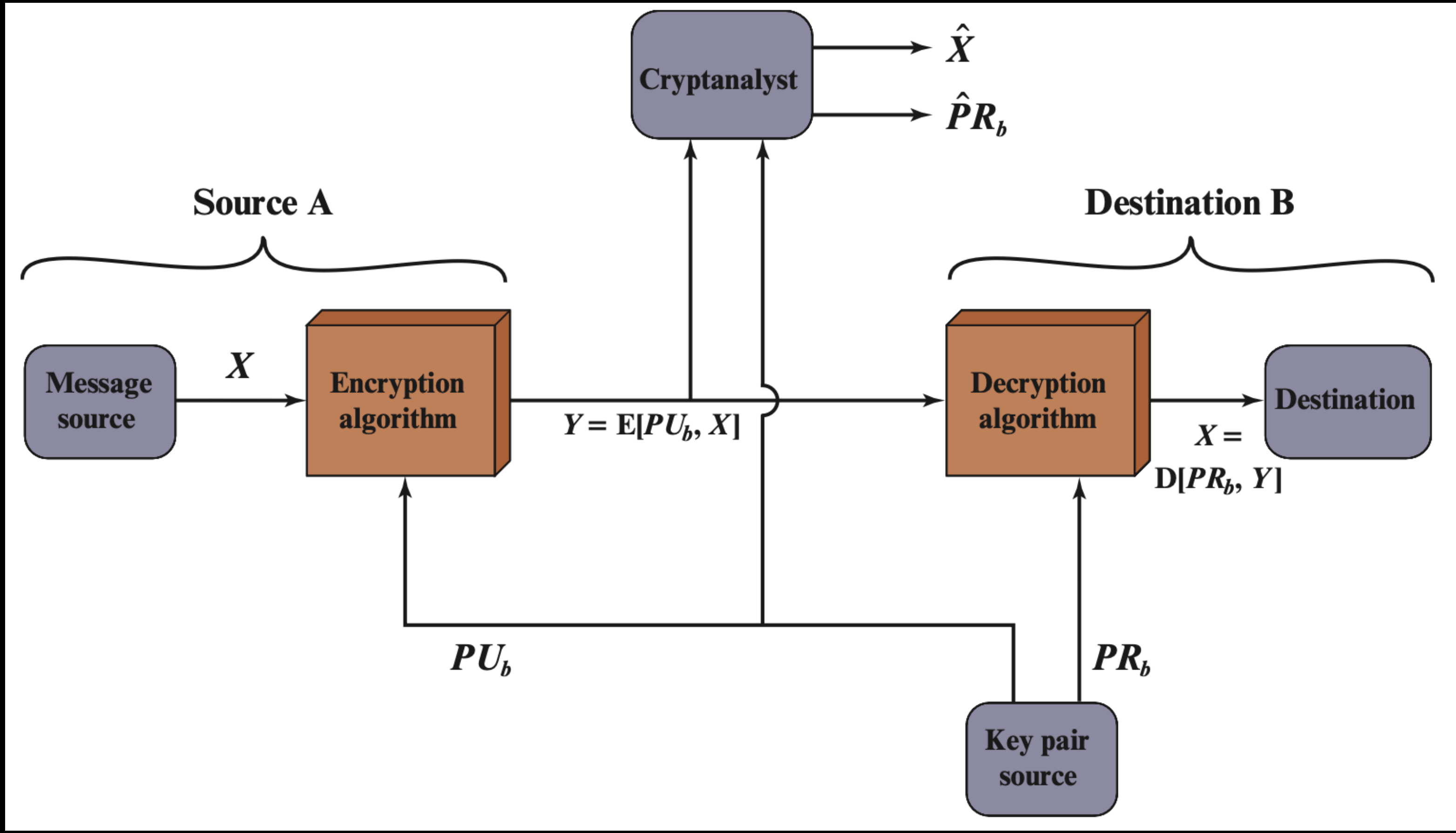
- Change the private key
- Keep the private key secret
- Publish the companion public key

Principles of Public-Key Cryptosystems

- A public-key encryption: confidentiality

- Encryption → Public key
- Decryption → Private key

- The cryptanalyst tries to:
  - Guess the plaintext, $\hat{X}$
  - Guess the private key, $\widehat{PR_b}$

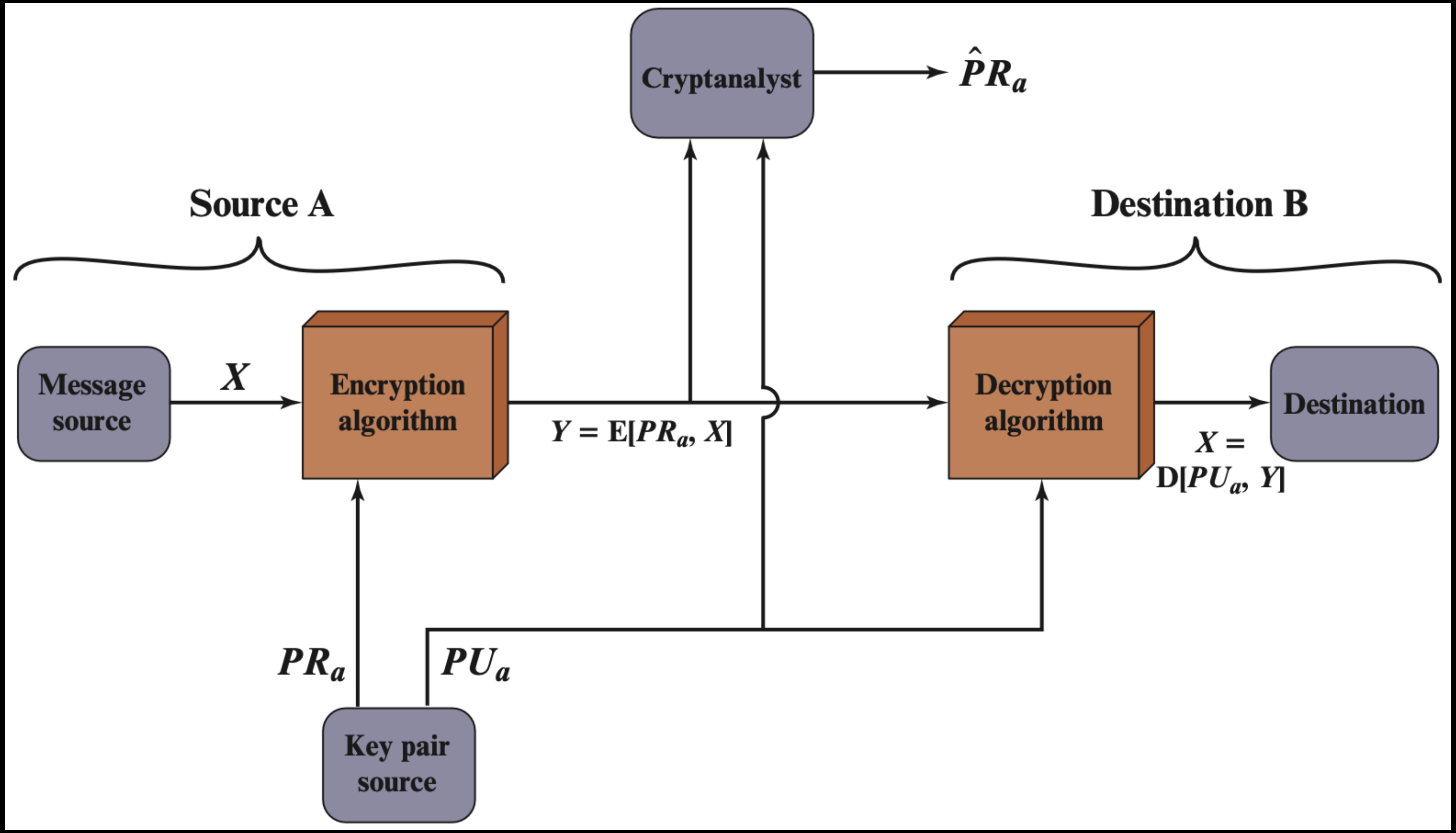


Principles of Public-Key Cryptosystems

- A public-key encryption: authentication – **Digital Signature**

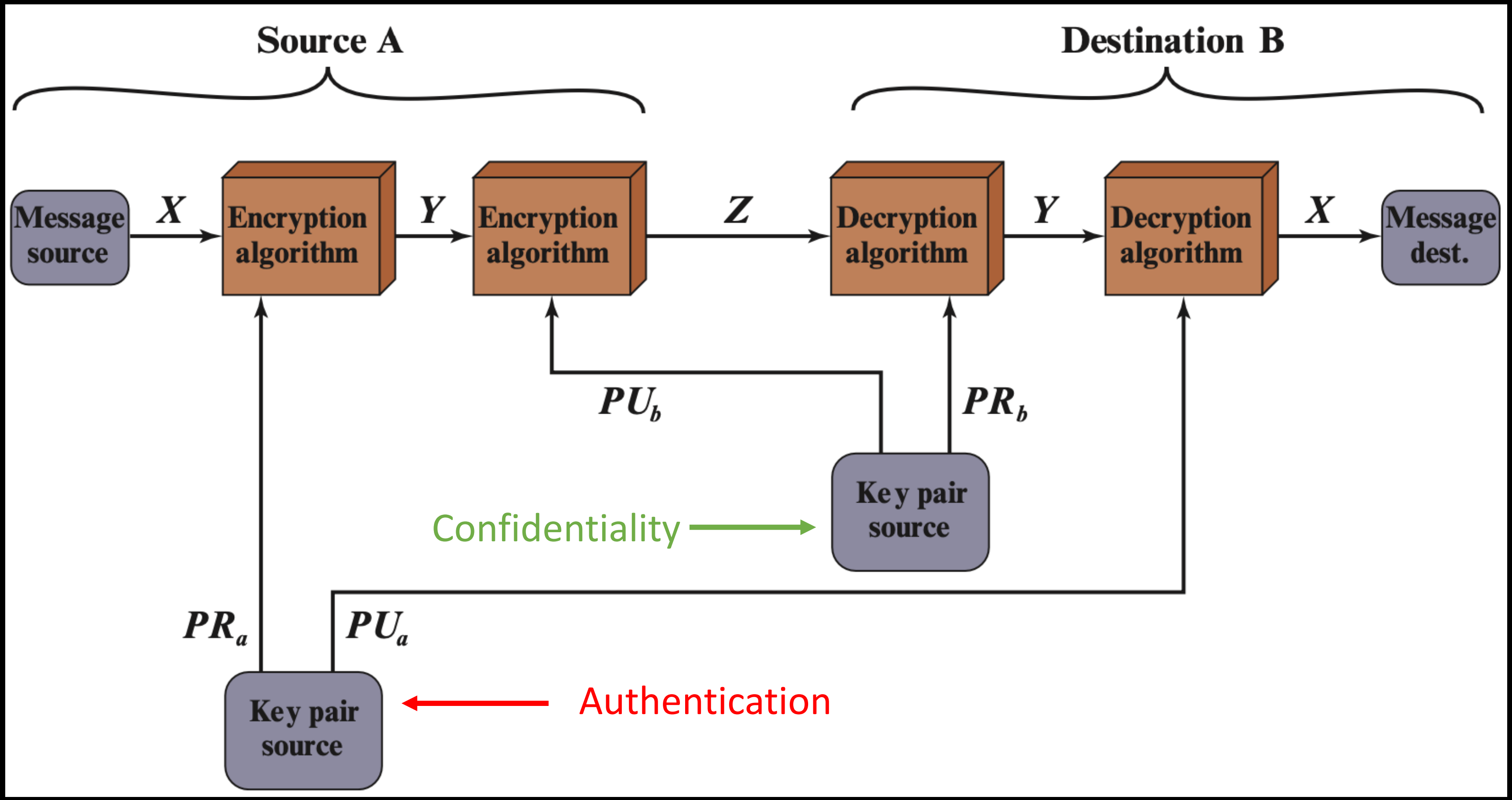
- Encryption → Private key
- Decryption → Public key

- The cryptanalyst tries to:
  - Guess the private key, $\widehat{PR}_a$



Principles of Public-Key Cryptosystems

- A public-key encryption: authentication and confidentiality



Principles of Public-Key Cryptosystems

What are the requirements for public key cryptography?

Principles of Public-Key Cryptosystems

1. Computationally easy for party B to generate key pair (PU_B, PR_B)

Principles of Public-Key Cryptosystems

1. Computationally easy for party B to generate key pair (PU_B, PR_B)
2. Computationally easy for a sender to encrypt a message: $C = E(PU_B, M)$

Principles of Public-Key Cryptosystems

1. Computationally easy for party B to generate key pair (PU_B, PR_B)
2. Computationally easy for a sender to encrypt a message: $C = E(PU_B, M)$
3. Computationally easy for a receiver to decrypt a message: $M = D(PR_B, C)$

Principles of Public-Key Cryptosystems

1. Computationally easy for party B to generate key pair (PU_B, PR_B)
2. Computationally easy for a sender to encrypt a message: $C = E(PU_B, M)$
3. Computationally easy for a receiver to decrypt a message: $M = D(PR_B, C)$
4. Computationally hard to get PR_B given only the PU_B

Principles of Public-Key Cryptosystems

1. Computationally easy for party B to generate key pair (PU_B, PR_B)
2. Computationally easy for a sender to encrypt a message: $C = E(PU_B, M)$
3. Computationally easy for a receiver to decrypt a message: $M = D(PR_B, C)$
4. Computationally hard to get PR_B given only the PU_B
5. Computationally hard to decrypt a C using PU_B

Principles of Public-Key Cryptosystems

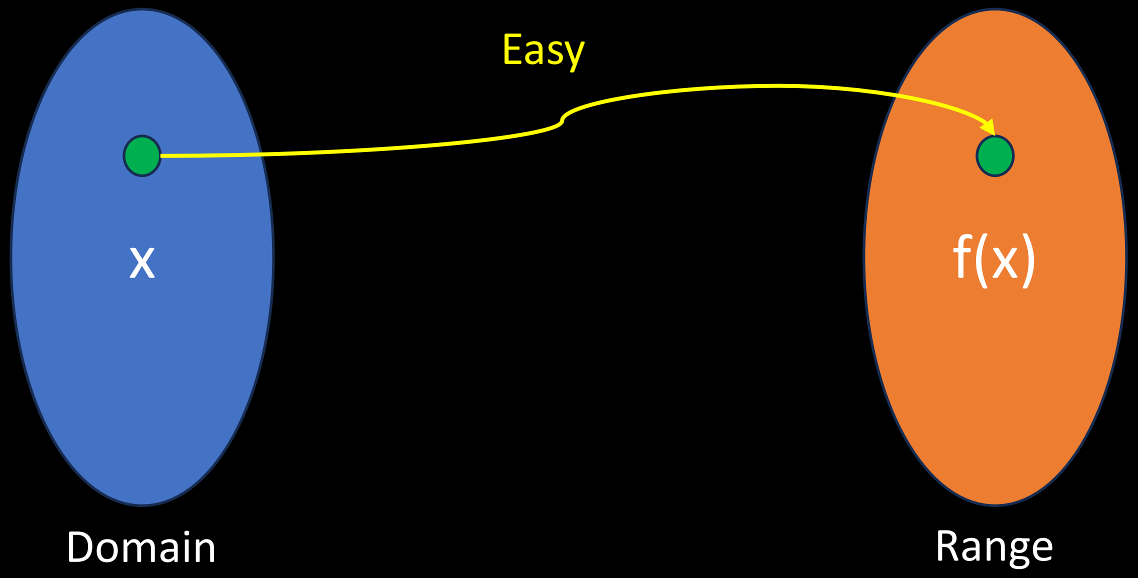
1. Computationally easy for party B to generate key pair (PU_B, PR_B)
2. Computationally easy for a sender to encrypt a message: $C = E(PU_B, M)$
3. Computationally easy for a receiver to decrypt a message: $M = D(PR_B, C)$
4. Computationally hard to get PR_B given only the PU_B
5. Computationally hard to decrypt a C using PU_B

$\Big\}$ Trap-door one-way function

Principles of Public-Key Cryptosystems

- Trapdoor function: maps the domain to the range

$$y = f_k(x)$$

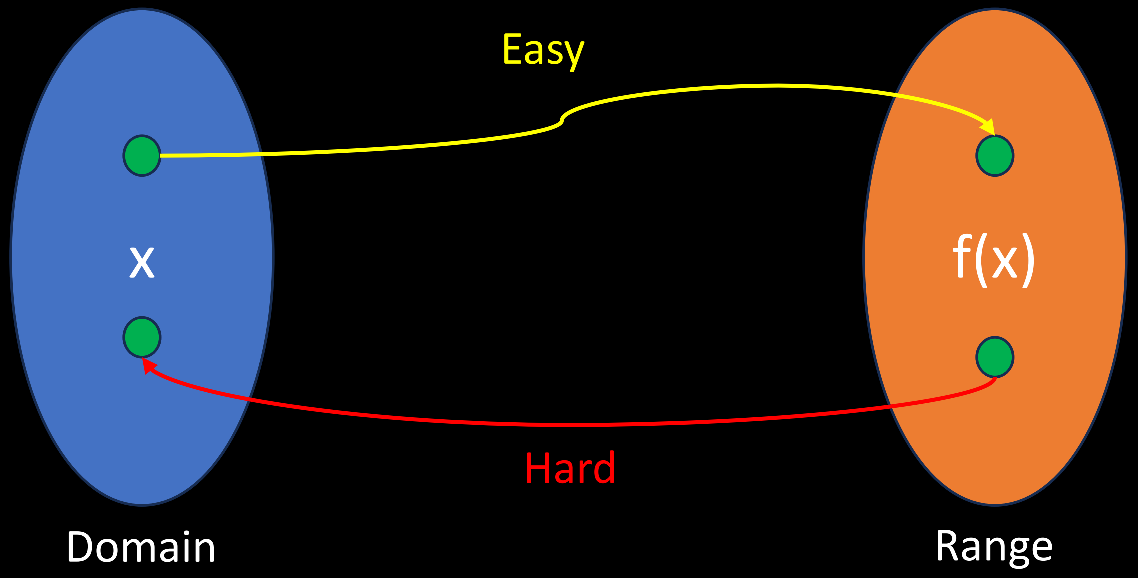


Principles of Public-Key Cryptosystems

- Trapdoor function: maps the domain to the range but impossible to calculate from the other direction

$$y = f_k(x)$$

$$x \neq f_?^{-1}(y)$$



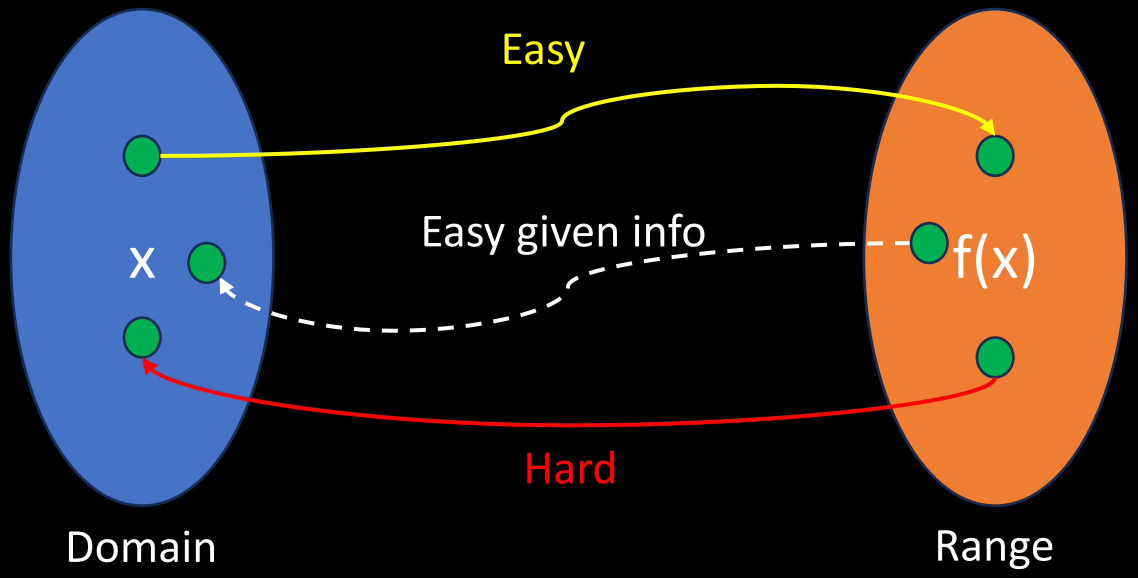
Principles of Public-Key Cryptosystems

- Trapdoor function: maps the domain to the range but impossible to calculate from the other direction unless some information is known.

$$y = f_k(x)$$

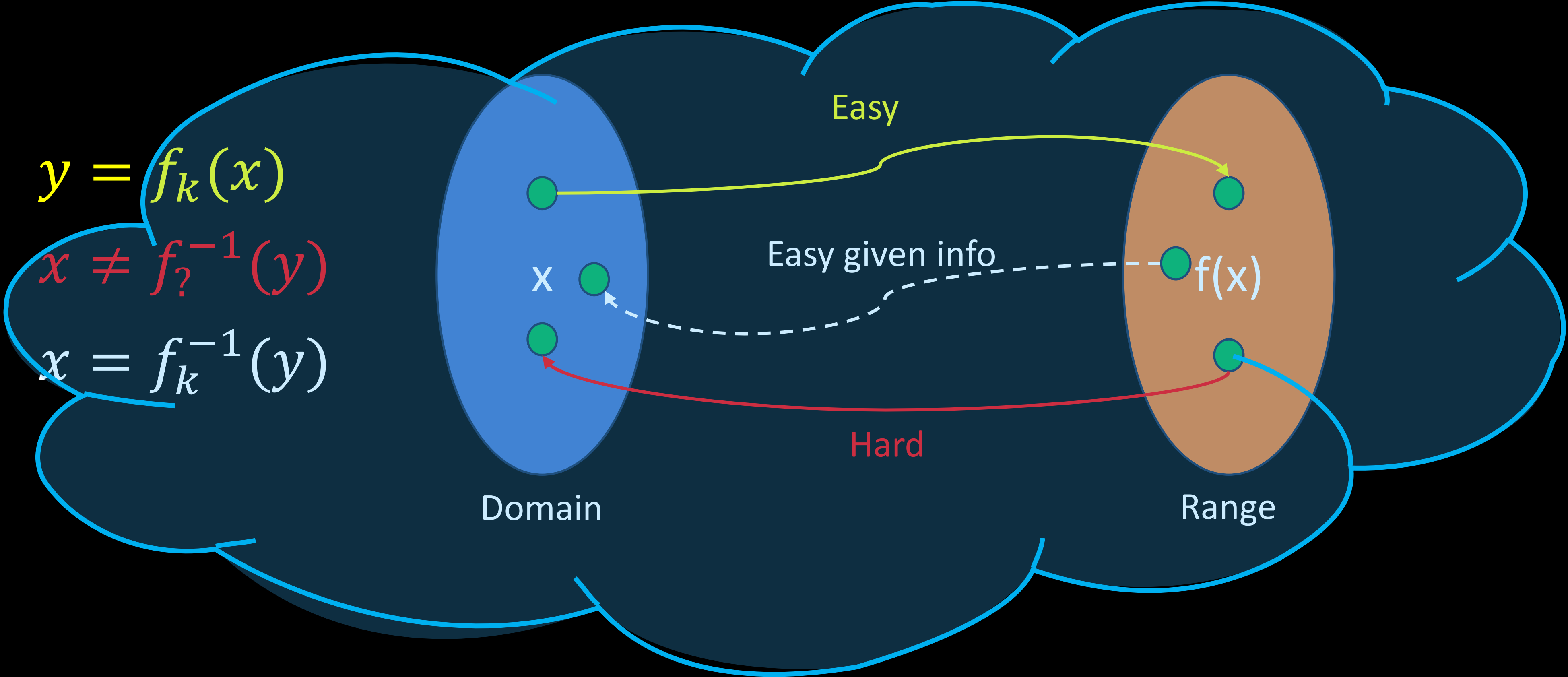
$$x \neq f_?^{-1}(y)$$

$$x = f_k^{-1}(y)$$



Principles of Public-Key Cryptosystems

- Trapdoor function: maps the domain to the range but impossible to calculate from the other direction unless some information is known.



To build a public key cryptosystem, **discover a trapdoor function**

Content

The RSA Algorithm

- RSA sees the plaintext and ciphertext as integers between $[0: n-1]$
- n is a large integer value; of size 1024 bits
  - A number that is 309 digits
  - $n < 2^{1024}$

24725387912226386773406422770506824337943430362146117585611811
53322971499614407180042227635152596218510405414532496746537437
34734188672138481874162250814304104733926303051948325997875058
56430815348087510866371728812805464582241735489450115288528172
8600701643105745461025953612473530075689406770397864088629194

The RSA Algorithm

- RSA math depends on **modular exponentiation**.
- The plaintext block must be less than n.

0 $\log_2(n)$+1

...10010101110101101000...

- For example, if $n = 13$, the plaintext block size = $\log_2(13) + 1 \cong 4$ bits
- Thus, the possible values can be $[0{:}\,12] \rightarrow [0000 : 1100]$

The RSA Algorithm

- Encryption:

$$C = M^e \; mod \; n$$

- Decryption:

$$M = C^d \; mod \; n = (M^e)^d \; mod \; n = M^{ed} \; mod \; n$$

- Public key: $PU = \{e, n\}$
- Private key: $PR = \{d, n\}$
- Both sender and receiver know e, n
- <u>Only receiver knows d</u>

The RSA Algorithm

- RSA requirements:

1. Possible to find value $e, d, n \mid M^{ed}\ mod\ n = M \quad \forall M < n$

2. Easy to compute $M^e\ mod\ n$ and $C^d\ mod\ n \qquad \forall M < n$

3. Infeasible to determine d given e and n

The RSA Algorithm

- RSA requirements:

1. Possible to find value $e, d, n \mid M^{ed} \; mod \; n = M \quad \forall M < n$

   How to find values e, d to satisfy $M^{ed} \; mod \; n$?

2. Easy to compute $M^e \; mod \; n$ and $C^d \; mod \; n \qquad \forall M < n$

3. Infeasible to determine d given e and n

The RSA Algorithm

- $M^{ed}\ mod\ n$ → if e and d are multiplicative inverses modulo $\phi(n)$

$$ed\ mod\ \phi(n) = 1$$
$$ed \equiv 1\ mod\ \phi(n)$$
$$d = e^{-1}\ mod\ \phi(n)$$

  - $\gcd\big(e, \phi(n)\big) = 1$ and $\gcd\big(d, \phi(n)\big) = 1$

- $\phi(n)$ is Euler totient function
  - If $n = p \times q$, where p and q are primes → $\phi(pq) = (p-1)(q-1)$

The RSA Algorithm

- RSA parameters set

Prime numbers p, q	Private	Chosen
$n = pq$	Public	Calculated
$e \mid \gcd(\phi(n), e) = 1; 1 < \phi(n)$	Public	Chosen
$d \equiv e^{-1} \left(mod\ \phi(n)\right)$	Private	Calculated

The RSA Algorithm

- Alice generates her keypair

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

The RSA Algorithm

- Bob sends a message to Alice

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

- Alice decrypts the message

Decryption by Alice with Alice's Private Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find e | $\gcd(\phi(n), e) = 1$
$d \equiv e^{-1} \ mod\ n$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17$$
$$q = 11$$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find e | $\gcd(\phi(n), e) = 1$
$d \equiv e^{-1} \ mod \ n$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17, \ q = 11$$
$$n = p \times q = 17 \times 11 = 187$$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find $e \mid \gcd(\phi(n), e) = 1$
$d \equiv e^{-1} \ mod \ n$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17, \ \ q = 11$$
$$n = p \times q = 17 \times 11 = 187$$
$$\phi(187) = (17-1)(11-1) = 160$$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find $e \mid \gcd(\phi(n), e) = 1$
$d \equiv e^{-1} \ mod\ n$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17,\ \ q = 11$$
$$n = p \times q = 17 \times 11 = 187$$
$$\phi(187) = (17-1)(11-1) = 160$$
$$e = 7\ \ \mid \gcd(160, 7) = 1$$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find $e \mid \gcd(\phi(n), e) = 1$
$d \equiv e^{-1} \ mod\ \phi(n)$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17, \ \ q = 11$$
$$n = p \times q = 17 \times 11 = 187$$
$$\phi(187) = (17-1)(11-1) = 160$$
$$e = 7 \ \ \mid \gcd(160, 7) = 1$$
$$d = egcd\big(e, \phi(n)\big) = egcd(7, 160) = 23$$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find $e \mid \gcd(\phi(n), e) = 1$
$d \equiv e^{-1}\ mod\ n$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17,\ \ q = 11$$
$$n = p \times q = 17 \times 11 = 187$$
$$\phi(187) = (17-1)(11-1) = 160$$
$$e = 7\ \ \mid \gcd(160, 7) = 1$$
$$d = egcd\big(e, \phi(n)\big) = egcd(7, 160) = 23$$
$$PU = \{7, 187\}$$

The RSA Algorithm

- Key generation example

Find primes p, q
$n = p \times q$
$\phi(n) = (p-1)(q-1)$
Find $e \mid \gcd(\phi(n), e) = 1$
$d \equiv e^{-1} \ mod\ n$
Publish $PU\{e, n\}$
Keep $PR = \{d, n\}$

$$p = 17, \ \ q = 11$$
$$n = p \times q = 17 \times 11 = 187$$
$$\phi(187) = (17-1)(11-1) = 160$$
$$e = 7 \ \ \mid \gcd(160, 7) = 1$$
$$d = egcd\big(e, \phi(n)\big) = egcd(7, 160) = 23$$
$$PU = \{7, 187\}$$
$$PR = \{23, 187\}$$

The RSA Algorithm

- Encrypt the message: "ABC" given $PU = \{7, 187\}$

1. Convert the string to numerical values:

A	B	C
65	66	67

2. Encrypt each character by computing: $M^7 \ mod \ 187$

65	66	67
$65^7 \ \% \ 187 = 142$	$66^7 \ \% \ 187 = 110$	$67^7 \ \% \ 187 = 67$

3. Send ciphertext $\{142, 110, 67\}$ to Alice

The RSA Algorithm

- Decrypt the message: $\{142, 110, 67\}$ given $PR = \{23, 187\}$

1. Read each ciphertext block and compute: $C^{23}\ mod\ 187$

142	110	67
$142^{23}\ \%\ 187 = 65$	$110^{23}\ \%\ 187 = 66$	$67^{23}\ \%\ 187 = 67$

2. Decode the encrypted values

65	66	67
A	B	C

3. The plaintext is “ABC”

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
The Chinese Remainder Theorem
Computational Aspects
The Security of RSA

The Chinese Remainder Theorem

- CRT: Solves a set of congruences with one variable but different moduli, which are relatively prime.

$$\begin{cases} x \equiv a_1 \ \ (mod\ m_1) \\ x \equiv a_2 \ \ (mod\ m_2) \\ x \equiv a_3 \ \ (mod\ m_3) \end{cases}$$

The Chinese Remainder Theorem

- CRT: Solves a set of congruences with one variable but <u>different</u> <u>moduli</u>, which are relatively prime.

$$\begin{cases} x \equiv a_1 \pmod{\boxed{m_1}} \\ x \equiv a_2 \pmod{\boxed{m_2}} \\ x \equiv a_3 \pmod{\boxed{m_3}} \end{cases}$$

Each of these is called a modulus, $m_1 \neq m_2 \neq m_3$

The Chinese Remainder Theorem

- CRT: Solves a set of congruences with one variable but different moduli, which are <u>relatively prime</u>.

Relatively prime =
coprime =
no common factors =
$\gcd(m_1, m_2) = 1$

e.g., $\gcd(5, 7) = 1$

$$\begin{cases} x \equiv a_1 \pmod{\boxed{m_1}} \\ x \equiv a_2 \pmod{\boxed{m_2}} \\ x \equiv a_3 \pmod{\boxed{m_3}} \end{cases}$$

The Chinese Remainder Theorem

- Example:

$$\begin{cases} x \equiv 2 \ \ (mod\ 3) \\ x \equiv 3 \ \ (mod\ 5) \\ x \equiv 2 \ \ (mod\ 7) \end{cases}$$

Has a unique solution which is $x = 23$. Because:

$$23\ \%\ 3 = 2 \ \rightarrow 23 \equiv 2\ (\%\ 3)$$
$$23\ \%\ 5 = 3 \ \rightarrow 23 \equiv 3\ (\%\ 5)$$
$$23\ \%\ 7 = 2 \ \rightarrow 23 \equiv 2\ (\%\ 7)$$

The Chinese Remainder Theorem

- How CRT works to find the solution:

Find the common modulus: $M = m_1 \times m_2 \times \cdots \times m_k$

Find $M_1 = M/m_1, M_2 = M/m_2, \ldots, M/m_k$

Find the multiplicative inverses: $M_1^{-1}, M_2^{-1}, \ldots, M_k^{-1}$ using their corresponding moduli

Compute the solution: $x = \left(a_1 \times M_1 \times M_1^{-1} + a_2 \times M_2 \times M_2^{-1} + \cdots + a_k \times M_k \times M_k^{-1}\right) \mod M$

The Chinese Remainder Theorem

- Example:

$$\begin{cases} x \equiv 2 \ \ (mod\ 3) \\ x \equiv 3 \ \ (mod\ 5) \\ x \equiv 2 \ \ (mod\ 7) \end{cases}$$

1. $M = 3 \times 5 \times 7 = 105$
2. $M_1 = 105/3 = 35, 105/5 = 21, 105/7 = 15$
3. $M_1^{-1} = Inv(35, 3) = 2, M_2^{-1} = Inv(21, 5) = 1, M_3^{-1} = mInv(15, 7) = 1$
4. $x = (2 \times 35 \times 2 + 3 \times 21 \times 1 + 2 \times 15 \times 1)\ mod\ 105 = 23$

The Chinese Remainder Theorem

- CRT can be used to represent large integers as a list of smaller integers.

- Example: Assume we need to calculate $z = x + y$ where $x = 123$ and $y = 334$, but our system accepts only numbers less than 100.

The Chinese Remainder Theorem

- $x = 123$ and $y = 334$, the limit is 100

1. Represent the numbers as congruences

$$\begin{cases} x \equiv \alpha_1 \ (mod\ 99) \\ x \equiv \alpha_2 \ (mod\ 98) \\ x \equiv \alpha_3 \ (mod\ 97) \end{cases} \qquad \begin{cases} y \equiv \beta_1 \ (mod\ 99) \\ y \equiv \beta_2 \ (mod\ 98) \\ y \equiv \beta_3 \ (mod\ 97) \end{cases}$$

The Chinese Remainder Theorem

- $x = 123$ and $y = 334$, the limit is 100

1. Represent the numbers as congruences

$$\begin{cases} x \equiv \alpha_1 \ (mod\ 99) \\ x \equiv \alpha_2 \ (mod\ 98) \\ x \equiv \alpha_3 \ (mod\ 97) \end{cases} \qquad \begin{cases} y \equiv \beta_1 \ (mod\ 99) \\ y \equiv \beta_2 \ (mod\ 98) \\ y \equiv \beta_3 \ (mod\ 97) \end{cases}$$

$$\alpha_1 = 123\ mod\ 99 = 24 \qquad \beta_1 = 334\ mod\ 99 = 37$$
$$\alpha_2 = 123\ mod\ 98 = 25 \qquad \beta_2 = 334\ mod\ 98 = 40$$
$$\alpha_3 = 123\ mod\ 97 = 26 \qquad \beta_3 = 334\ mod\ 97 = 43$$

The Chinese Remainder Theorem

- $x = 123$ and $y = 334$, the limit is 100

1. Represent the numbers as congruences

$$\begin{cases} x \equiv 24 (mod\ 99) \\ x \equiv 25\ (mod\ 98) \\ x \equiv 26\ (mod\ 97) \end{cases} + \begin{cases} y \equiv 37\ (mod\ 99) \\ y \equiv 40\ (mod\ 98) \\ y \equiv 43\ (mod\ 97) \end{cases} = \begin{cases} z \equiv 61\ (mod\ 99) \\ z \equiv 65\ (mod\ 98) \\ z \equiv 69\ (mod\ 97) \end{cases}$$

The Chinese Remainder Theorem

- $x = 123$ and $y = 334$, the limit is 100

1. Represent the numbers as congruences

$$\begin{cases} x \equiv 24 (mod\ 99) \\ x \equiv 25\ (mod\ 98) \\ x \equiv 26\ (mod\ 97) \end{cases} + \begin{cases} y \equiv 37\ (mod\ 99) \\ y \equiv 40\ (mod\ 98) \\ y \equiv 43\ (mod\ 97) \end{cases} = \begin{cases} z \equiv 61\ (mod\ 99) \\ z \equiv 65\ (mod\ 98) \\ z \equiv 69\ (mod\ 97) \end{cases}$$

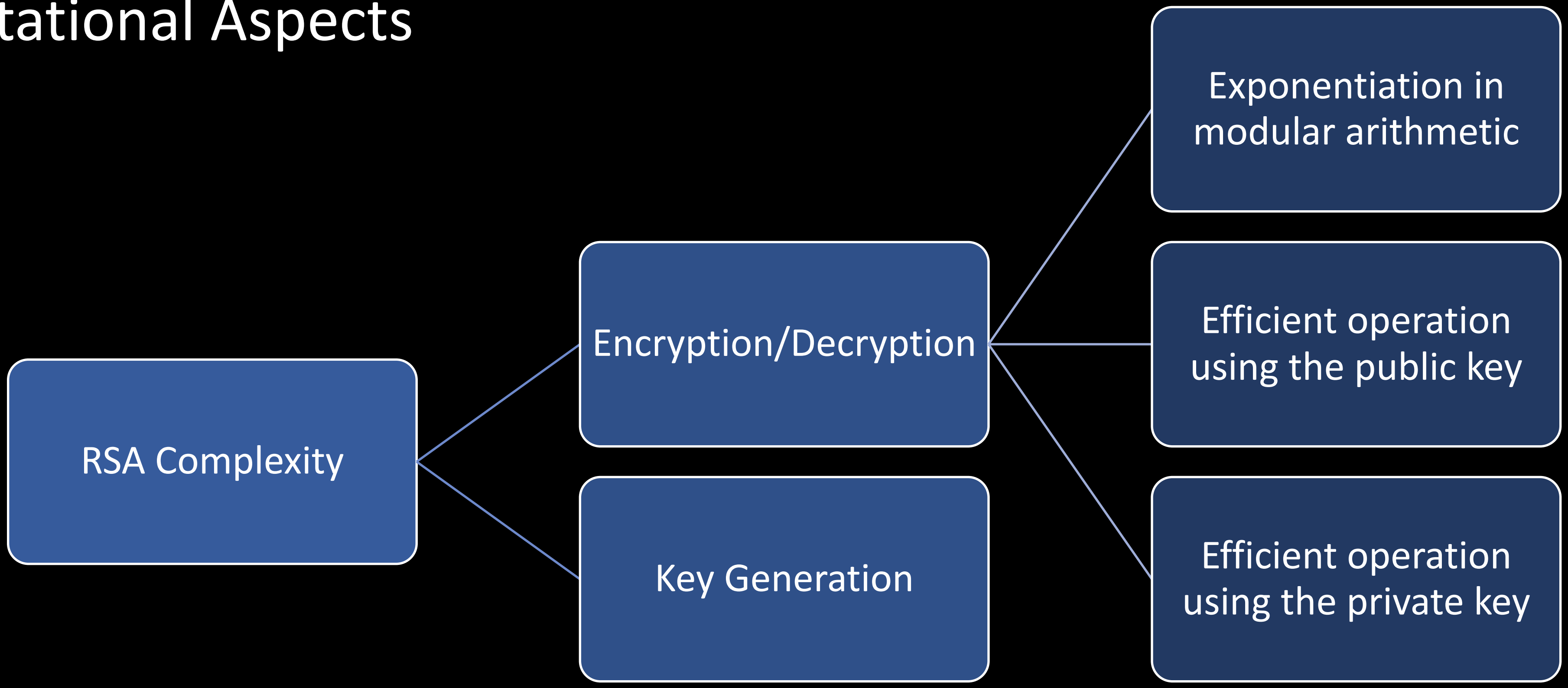
2. Solve the z congruences using CRT to get $z = 457$

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
The Chinese Remainder Theorem
Computational Aspects
The Security of RSA

Computational Aspects

- Computational Aspects



Computational Aspects

Exponentiation in modular arithmetic

- $(large\ integer)^{large\ integer}\ mod\ (large\ integer)$ → The intermediate values are too large to handle!

- **Solution**: reduce the intermediate results modulo n:
  - For multiplication: $(a \times b)\ mod\ n = [(a\ mod\ n) \times (b\ mod\ n)]\ mod\ n$
  - For exponentiation: $x^{16} = x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x \times x$ → $x^{(10000)_b}$; use the square-multiply algorithm, performs 4 squares and 1 multiplication.

Computational Aspects

Efficient operation using the public key

- To allow faster computations during encryption, specific e values can be chosen: 65537, 3, 17

- Each of these choices has only two 1 bits, so the number of multiplications required to perform exponentiation is minimized

- Using small e values makes the RSA vulnerable to attacks

Computational Aspects

Efficient operation using the public key

- Three different RSA users use $e = 3$ but have unique values of n: n_1, n_2, n_3.
- If user A sends the same encrypted message M to all three users, then the ciphertexts are $C_1 = M^3\ mod\ n_1$, $C_2 = M^3 mod\ n_2$, $C^3 = M^3\ mod\ n_3$.
- It is likely that n_1, n_2, n_3 are pairwise relatively prime. Therefore, one can use the CRT to compute $M^3\ mod\ (n_1 n_2 n_3)$.
- $M = \sqrt[3]{M^3}$
- **Solution**: pad each M with a unique random bit string to be encrypted

Computational Aspects

Efficient operation using the private key

- Small values of d → Brute force attacks
- To speed up the computation of $M = C^d \; mod \; n$:

1. Define: (1) $V_p = C^d \; mod \; p, \qquad V_q = C^d \; mod \; q$

   (2) $X_p = q \times (q^{-1} \; mod \; p), \quad X_q = p \times (p^{-1} \; mod \; q)$
2. Compute $M = \left(V_p X_p + V_q X_q\right) mod \; n$

Computational Aspects

Efficient operation using the private key

- Small values of d → Brute force attacks
- To speed up the computation of $M = C^d \mod n$:

1. Define: (1) $V_p = C^d \mod p, \quad V_q = C^d \mod q$

   (2) $X_p = q \times (q^{-1} \mod p), \quad X_q = p \times (p^{-1} \mod q)$

2. Compute $M = \left(V_p X_p + V_q X_q\right) \mod n$

- Calculating V_p and V_q is 4X faster with **Fermat's little theorem**:
  - $V_p = C^d \mod p = C^{d \mod (p-1)} \mod q, \quad V_q = C^d \mod q = C^{d \mod (q-1)} \mod q$

Computational Aspects

Key generation

Determining two prime numbers, p and q

- Selecting large primes to prevent factorizing n into p and q
- Use **probabilistic primality test** algorithms to test p, q
  - e.g., Miller-Rabin algorithm

Selecting either e or d and calculating the other

- We need to select e such that $\gcd(\phi(n), e) = 1$ and calculate $d \equiv e^{-1}\, mod\ \phi(n)$
- The Extended Euclidean Algorithm can determine the gcd and the inverse at the same time

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
The Chinese Remainder Theorem
Computational Aspects
The Security of RSA

The Security of RSA

- Five possible approaches to attacking the RSA algorithm

Brute force

- Trying all possible keys

Mathematical attacks

- Efforts to factorize n into p, q

Timing attacks

- Determine the run-time of the decryption algorithm to determine the decryption key

Hardware fault-based attack

- Inducing hardware faults in the processor that is generating digital signatures

Chosen Ciphertext attack

- Exploiting properties of the RSA algorithm

The Security of RSA

The factoring problem

- Determining the factors of $n = pq$ → calculating $\phi(n) = (p-1)(q-1)$
- Thus, it's easy to compute $d \equiv e^{-1}\ mod\ \phi(n)$
- Fast algorithms to compute the factorization:
  - Generalized number field sieve (GNFS)
  - Specialized number field sieve (SNFS)
- Researchers broke the modulus n up to 768-bits
- Governmental standards recommend using n of size 2048 or larger

The Security of RSA

Timing attacks

- Attackers observe the time to decrypt a message to recover the private key
- Exploit the square-multiply algorithm and similar algorithms that don't run in a fixed time
- Countermeasures:
  1. **Constant exponentiation time**: Ensure that all exponentiations take the same amount of time before returning a result
  2. **Random delay**: add a random delay to the exponentiation algorithm to confuse the timing attack
  3. **Blinding**: Multiply the ciphertext by a random number before performing exponentiation

The Security of RSA

- Blinding prevents knowing what ciphertext bits are being processed and therefore prevents the bit-by-bit analysis.

Generate a secret random number r between 0 and $n - 1$

Compute $C` = C \times r^e \; mod \; n$

Compute $M' = C`^d \; mod \; n$

Compute $M = M`r^{-1} \; mod \; n \mid r^{-1}$ is the multiplicative inverse of r

The Security of RSA

Fault-based attacks

- Reduces the power to the processor → computing invalid signatures → attackers can recover the private key

- Can take 100 hours to extract 1024-bit private key

- It requires the attacker to have physical access to the computer → not very practical

The Security of RSA

Chosen Ciphertext attacks

- We can decrypt $C = M^e \; mod \; n$ using CCA as follows (we don't know M):

1. Compute $X = (C \times 2^e) \; mod \; n$
2. Submit X as a chosen ciphertext and receive back $Y = X^d \; mod \; n$
3. Now, $Y = X^d \; mod \; n \;\rightarrow (\mathrm{C \; mod \; n}) \times (2^e \; mod \; n)$
   $= (M^e \; mod \; n) \times (2^e \; mod \; n) \;\rightarrow (2M)^e \; mod \; n$
4. The attacker deduces M

- Check this demo: https://asecuritysite.com/encryption/c_c

The Security of RSA

Chosen Ciphertext attacks

- The goal of CCA is to model the attackers' capabilities
- Decrypting something is not always enough to break a system!
  - Example: Video-protection devices perform encryption/decryption queries using the device's chip.
  - But attackers are interested in the key to redistribute it;
  - In this case, decrypting "for free" isn't sufficient to break the system
  - Check this: https://crypto.stackexchange.com/questions/26689/easy-explanation-of-ind-security-notions/26738#26738
- To prevent CCA, use the **optimal asymmetric encryption padding (OAEP)**

The Security of RSA

- Write a python function to return a prime number of specific bit size.
  - Use $sympy.\mathrm{isprime}()$ function to check if a random value is prime or not.

- Implement the EGCD algorithm to return the modular inverse of a number.
- Implement a function that generate two RSA key pairs.
  - Assume a default value for $e = 65537$

- Implement a function to encrypt an integer value using RSA.
- Implement a function to decrypt an integer value using RSA.