Cryptography

Asymmetric Cryptography

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
Knapsack Cryptosystem

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
Knapsack Cryptosystem

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

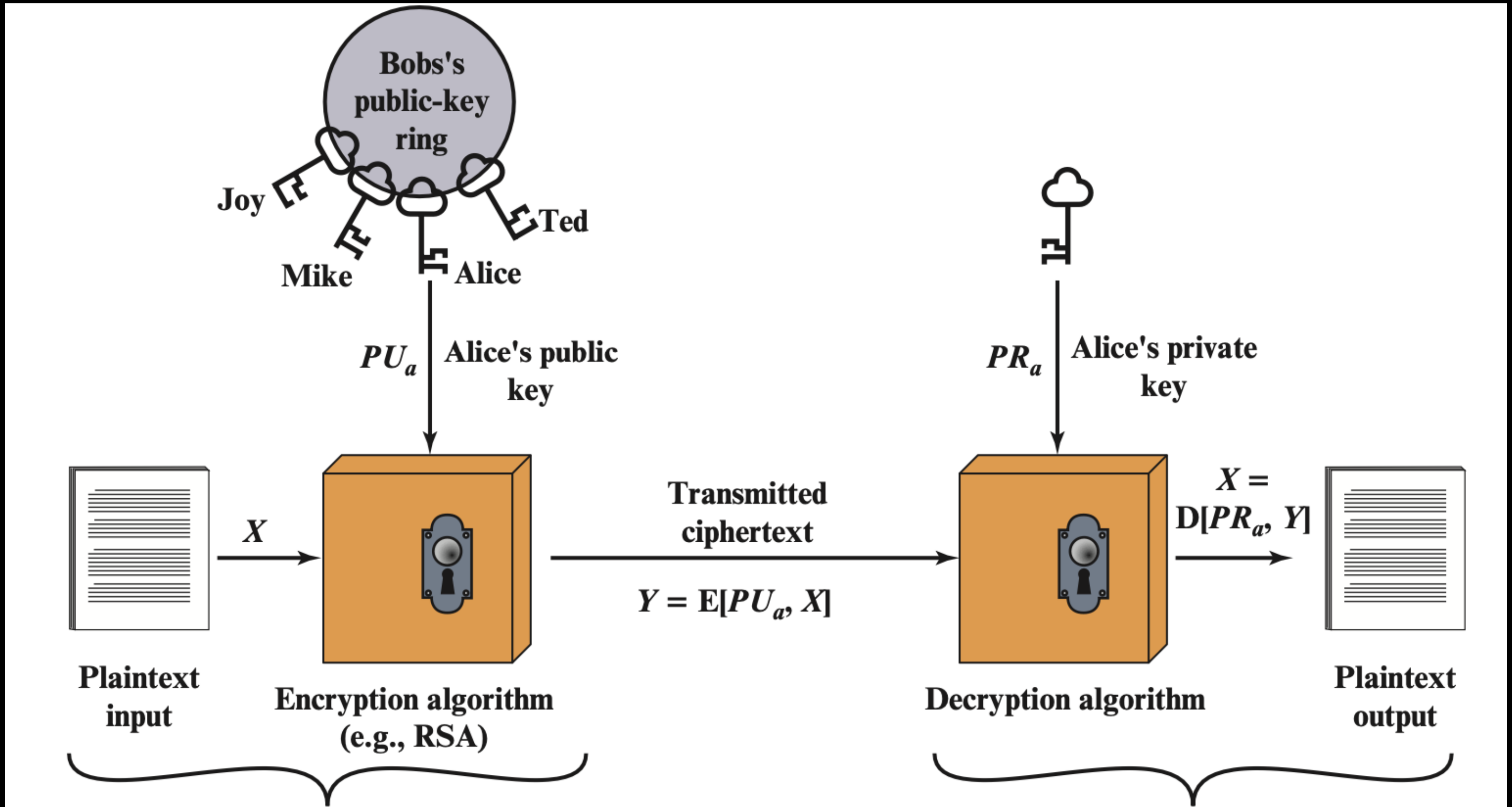
Principles of Public-Key Cryptosystems

- Asymmetric cryptography 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:
Encrypted message
using Alice's PK

Alice decrypts it
using her private
key

Any user can update his key pair:

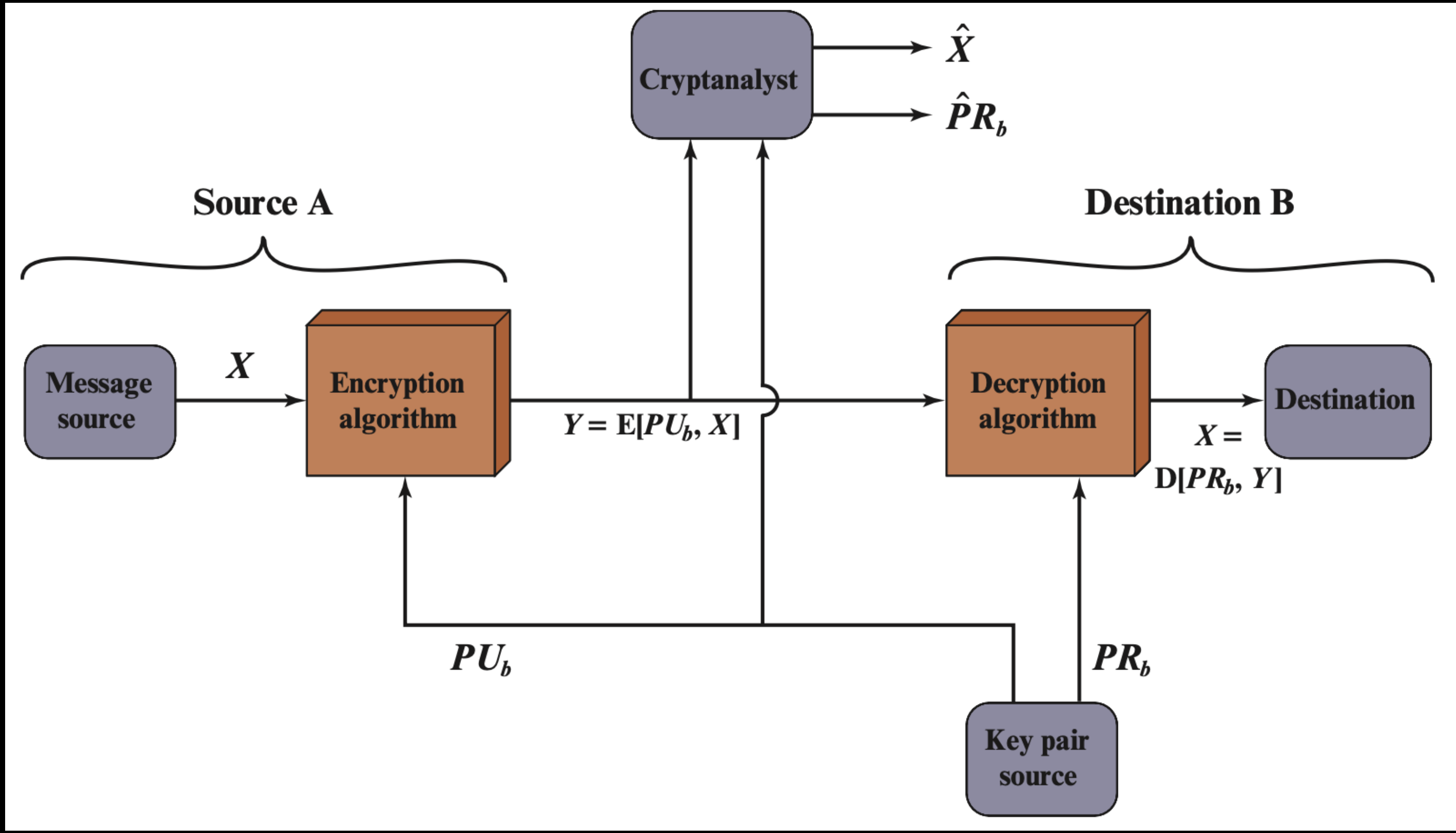
- Change the private key
- Keep the private key secret
- Publish the companion public key

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, $\widehat{PR_b}$

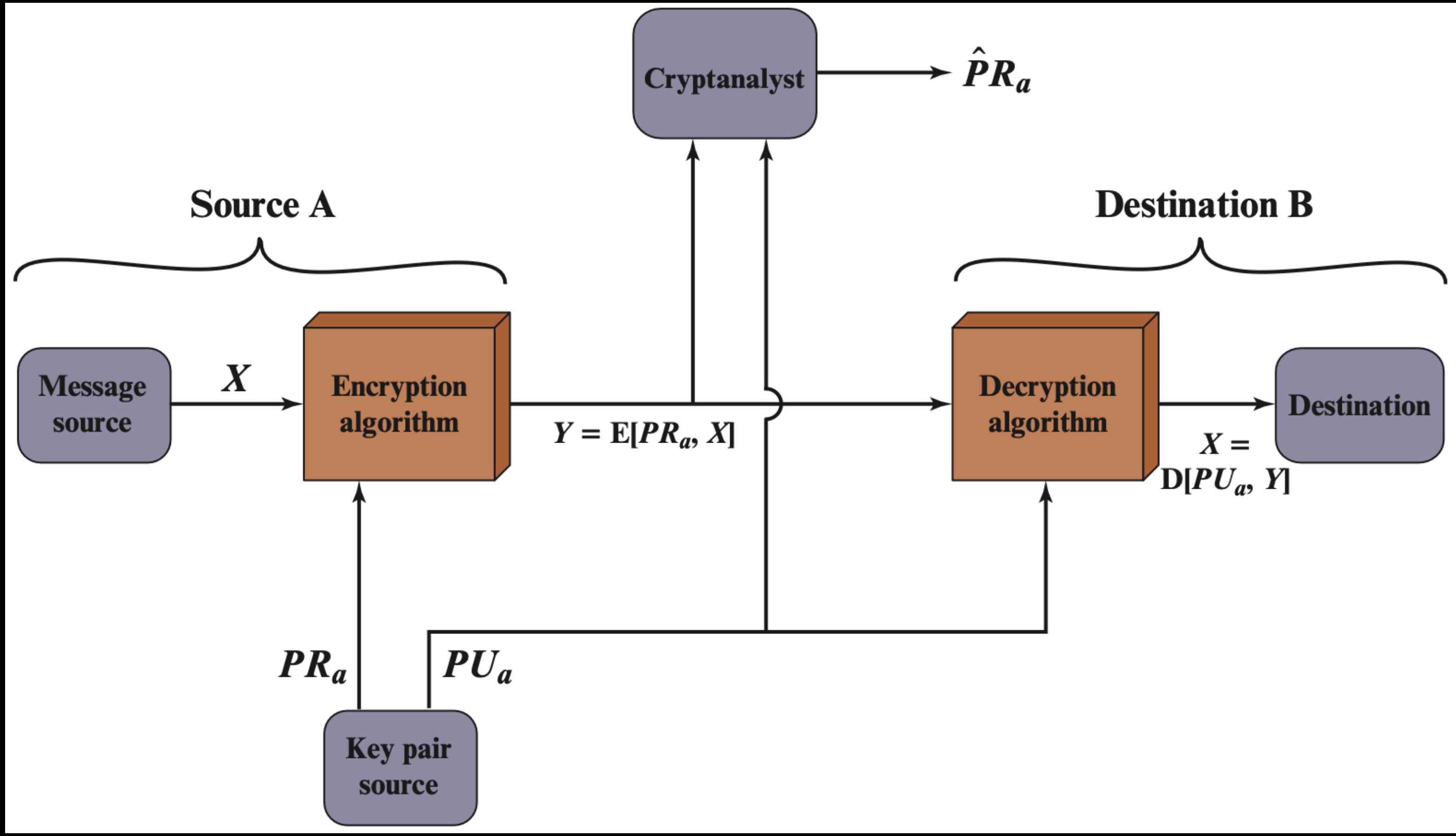


Principles of Public-Key Cryptosystems

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

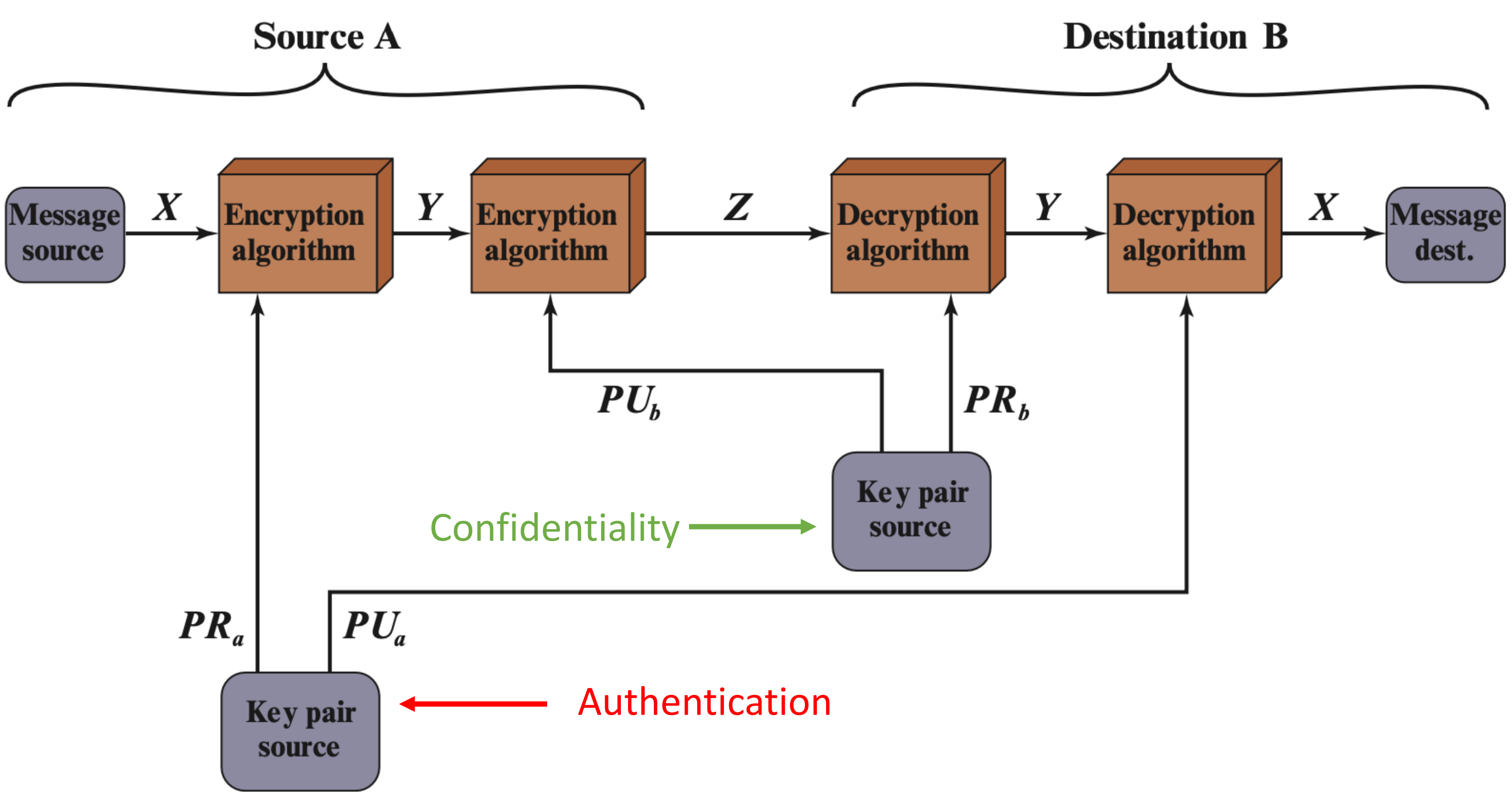
- Encryption → Private key
- Decryption → Public key

- 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 C using PU_B

Principles of Public-Key Cryptosystems

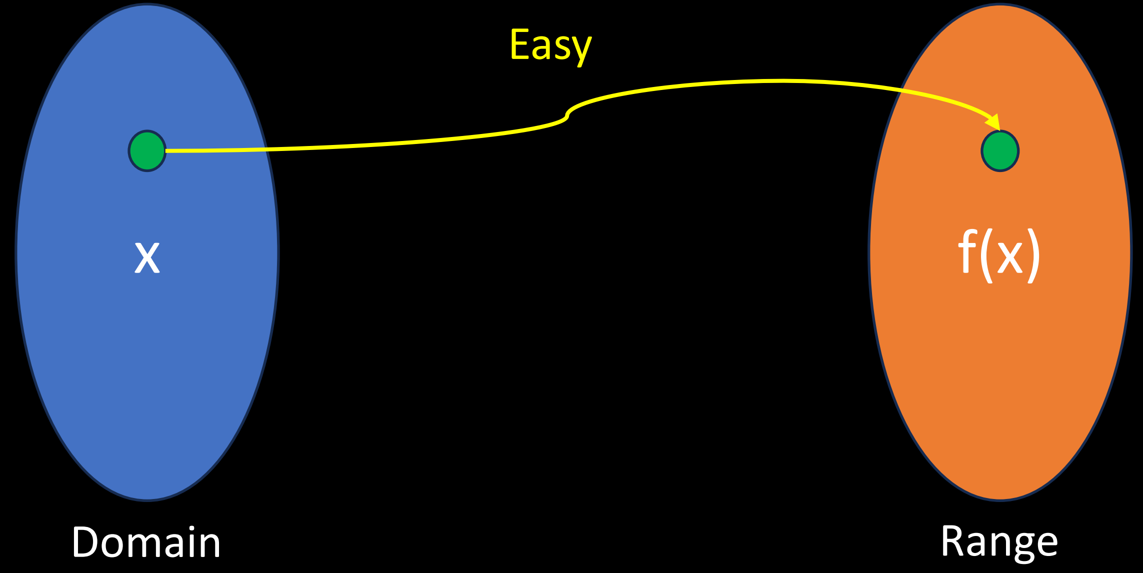
1. Computationally easy for party B to generate key pair (PU_B, PR_B)
2. Computationally easy for a sender to encrypt a message: $C = E(PU_B, M)$
3. Computationally easy for a receiver to decrypt a message: $M = D(PR_B, C)$
4. Computationally hard to get PR_B given only the PU_B
5. Computationally hard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

Trap-door one-way function

Principles of Public-Key Cryptosystems

- Trapdoor function: maps the domain to the range

$$y = f_k(x)$$

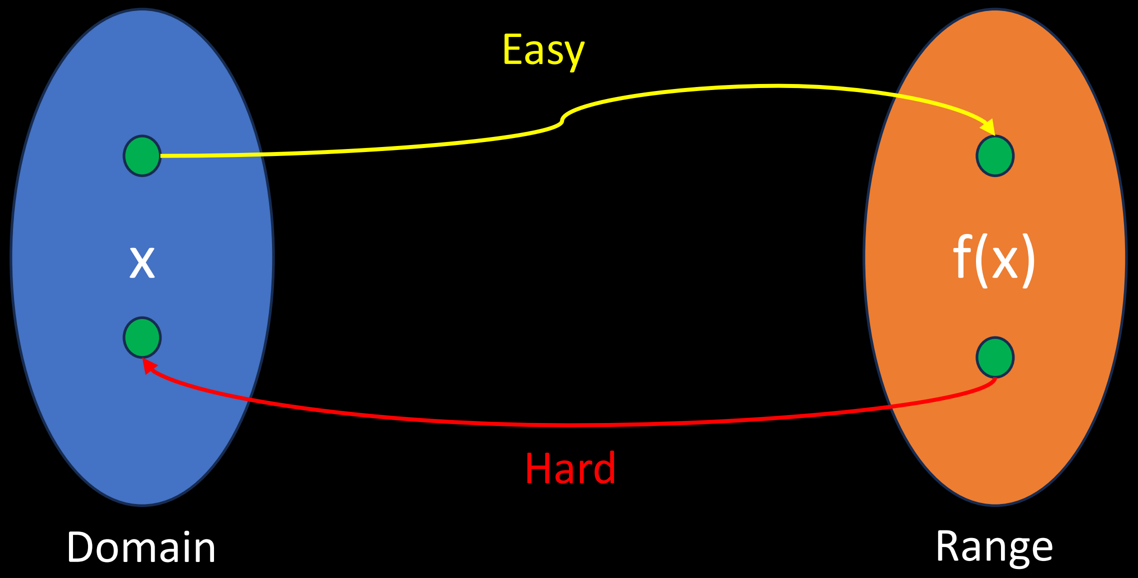


Principles of Public-Key Cryptosystems

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

$$y = f_k(x)$$

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



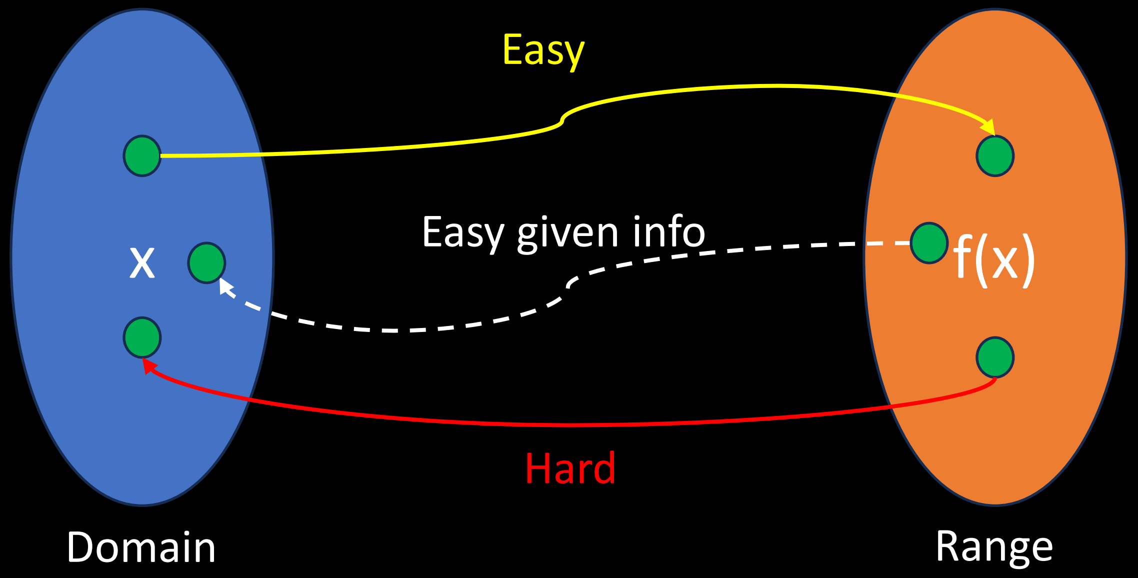
Principles of Public-Key Cryptosystems

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

$$y = f_k(x)$$

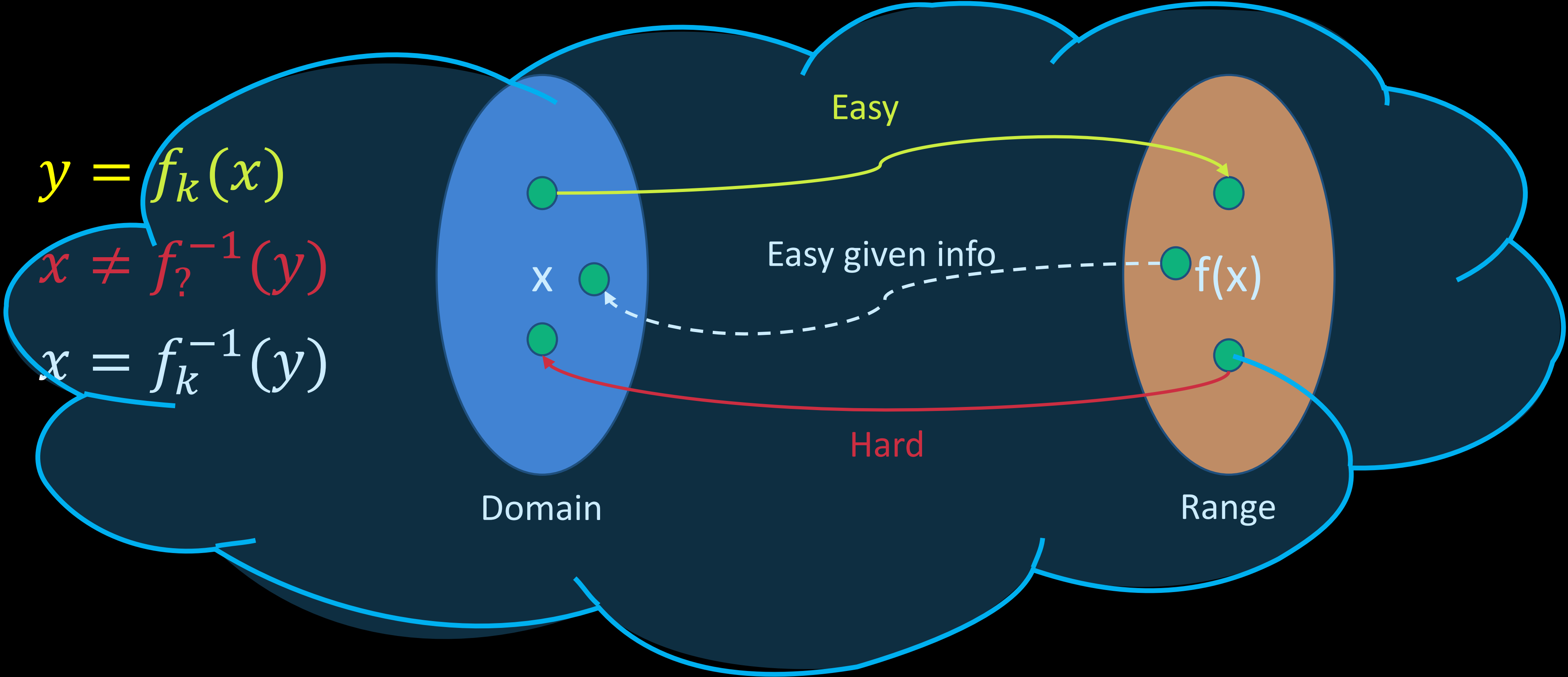
$$x \neq f_?^{-1}(y)$$

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



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

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
Knapsack Cryptosystem

The RSA Algorithm

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

24725387912226386773406422770506824337943430362146117585611811
53322971499614407180042227635152596218510405414532496746537437
34734188672138481874162250814304104733926303051948325997875058
56430815348087510866371728812805464582241735489450115288528172
86007016431057454610259536124735300756894067703978640886291 94

The RSA Algorithm

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

0 $\log_2(n)$+1

...1001010111010110100...

- 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 $C^d \; mod \; n \quad \forall M < n$

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
$e \mid \gcd(\phi(n), e) = 1; 1 < \mathrm{e}$	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 $PR = \{d, n\}$

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

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

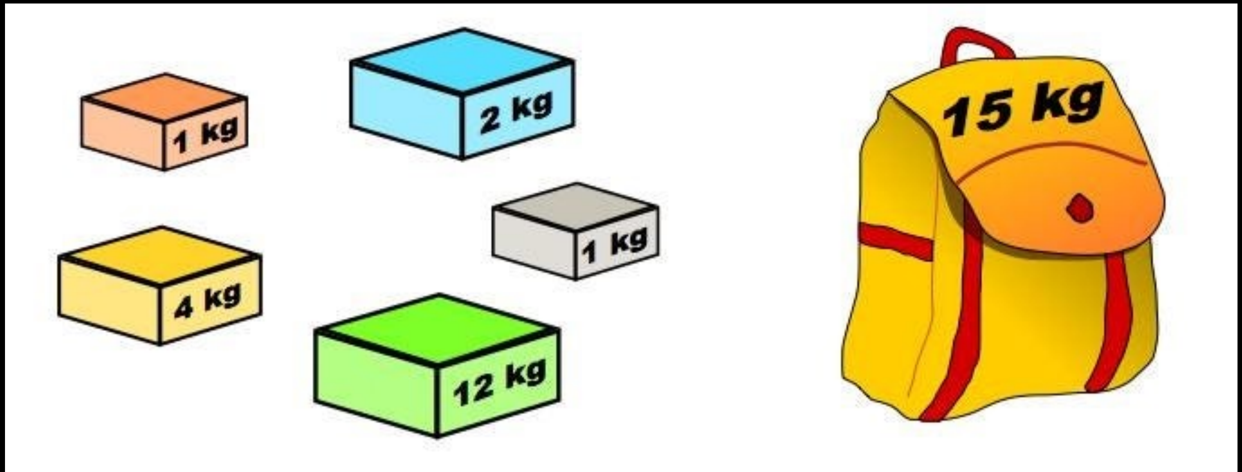
3. The plaintext is “ABC”

Content

Content
Introduction
Principles of Public-Key Cryptosystems
The RSA Algorithm
Knapsack Cryptosystem

Knapsack Cryptosystem

- The first public-key cryptosystem
  - Broken, not secure with today's standards
- Based on the knapsack problem:
  - Assume we have a knapsack that a set of numbers
  - If we are told what the numbers are → easy to compute the sum
  - If we are told what the sum is → hard to tell what numbers in the knapsack



Knapsack Cryptosystem

- If we are given two k-tuples: $a = [a_1, a_2, \dots, a_k]$ and $x = [x_1, x_2, \dots, x_n]$
  - a is a tuple that includes a set of elements, x is a tuple in which x_i is either 0 or 1
  - 0 = the element a_i is not in the knapsack, 1 = the element is in the knapsack
- The sum of elements in the knapsack is

$$s = knapsackSum(a, x) = x_1 a_1 + x_2 a_2 + \cdots + x_n a_n$$

- Given x and a → easy to compute s
- Given s → hard to compute x and a
  - $inv\_knapsackSum$ is infeasible
- We say that $knapsackSum$ is a one-way function

Knapsack Cryptosystem

- Easy to compute $knapsackSum$ and its inverse if a is **superincreasing**
- In a superincreasing tuple, $a_i \geq a_1 + a_2 + \cdots + a_{i-1}$
  - each element (except a_1) is greater than or equal to the sum of all previous elements

```
knapsackSum (x [1 … k], a [1 … k])
{
    s ← 0
    for (i = 1 to k)
    {
      s ← s + a_i × x_i
    }
    return s
}
```

```
inv_knapsackSum (s, a [1 … k])
{
    for (i = k down to 1)
    {
        if s ≥ a_i
        {
            x_i ← 1
            s ← s − a_i
        }
        else x_i ← 0
    }
    return x [1 … k]
}
```

Knapsack Cryptosystem

- Example: assume that $a = [17, 25, 46, 94, 201, 400]$ and $s = 272$ are given. Show how the tuple x is found using $inv\_knapsackSum$ routine.

i	a_i	s	$s \geq a_i$	x_i	$s \leftarrow s - a_i \times x_i$
6	400	272	false	$x_6 = 0$	272
5	201	272	true	$x_5 = 1$	71
4	94	71	false	$x_4 = 0$	71
3	46	71	true	$x_3 = 1$	25
2	25	25	true	$x_2 = 1$	0
1	17	0	false	$x_1 = 0$	0

- So, $x = [0, 1, 1, 0, 1, 0]$: 25, 46, and 201 are in the knapsack.

Knapsack Cryptosystem

Knapsack key generation

1. Create a superincreasing k-tuple $b = [b_1, b_2, \dots, b_k]$
2. Choose a modulus n, such that $n > b_1 + b_2 + \ \dots + b_k$
3. Select a random integer r that is co-prime with n and $1 \leq r \leq n - 1$
4. Create a temporary k-tuple $t = [t_1, t_2, \dots, t_k]$ in which $t_i = r \times b_i \ mod\ n$
5. Select a permutation of k objects and find a new tuple $a = permute(t)$
6. The public key is the k-tuple a
7. The private key is n, r, and the k-tuple b

Knapsack Cryptosystem

Encryption

Suppose Alice needs to send message to Bob.

1. Alice converts her message to a k-tuple $x = [x_1, x_2, \ldots, x_k]$, in which x_i is either 0 or 1. The tuple x is the plaintext
2. Alice uses the $knapsackSum$ routine to calculate s
3. Alice sends the value of s as the ciphertext

Knapsack Cryptosystem

Decryption

Bob receives the ciphertext s.

1. Bob calculates $s` = r^{-1} \times s \; mod \; n$
2. Bob uses $inv\_knapsackSum$ to create $x`$
3. Bob permutes $x`$ to find x
4. The tuple x is the recovered plaintext

Knapsack Cryptosystem

- Example:

Key generation
Create $b = [b_1, b_2, \dots, b_k]$
Choose a modulus n
Select r that is co-prime with n and $1 \leq r \leq n - 1$
Create $t = [t_1, t_2, \dots, t_k] \mid t_i = r \times b_i \ mod\ n$
Select $a = permute(t)$
The public key is the k-tuple a
The private key is n, r, and the k-tuple b

The superincreasing tuple

$$b = [7, 11, 19, 39, 79, 157, 313]$$

Knapsack Cryptosystem

- Example:

Key generation
Create $b = [b_1, b_2, \ldots, b_k]$
Choose a modulus n
Select r that is co-prime with n and $1 \leq r \leq n-1$
Create $t = [t_1, t_2, \ldots, t_k] \mid t_i = r \times b_i \ mod \ n$
Select $a = permute(t)$
The public key is the k-tuple a
The private key is $n, r,$ and the k-tuple b

The superincreasing tuple

$$b = [7, 11, 19, 39, 79, 157, 313]$$

$$n = 900$$

Knapsack Cryptosystem

- Example:

Key generation
Create $b = [b_1, b_2, \dots, b_k]$
Choose a modulus n
Select r that is co-prime with n and $1 \leq r \leq n - 1$
Create $t = [t_1, t_2, \dots, t_k] \mid t_i = r \times b_i \; mod \; n$
Select $a = permute(t)$
The public key is the k-tuple a
The private key is n, r, and the k-tuple b

The superincreasing tuple

$$b = [7, 11, 19, 39, 79, 157, 313]$$

$n = 900$

$r = 37$

Knapsack Cryptosystem

- Example:

Key generation
Create $b = [b_1, b_2, \dots, b_k]$
Choose a modulus n
Select r that is co-prime with n and $1 \leq r \leq n-1$
Create $t = [t_1, t_2, \dots, t_k] \mid t_i = r \times b_i \ mod \ n$
Select $a = permute(t)$
The public key is the k-tuple a
The private key is n, r, and the k-tuple b

The superincreasing tuple

$$b = [7, 11, 19, 39, 79, 157, 313]$$

$$n = 900$$

$$r = 37$$

$$t = [259, 407, 703, 543, 223, 409, 781]$$

Knapsack Cryptosystem

- Example:

Key generation
Create $b = [b_1, b_2, \dots, b_k]$
Choose a modulus n
Select r that is co-prime with n and $1 \leq r \leq n-1$
Create $t = [t_1, t_2, \dots, t_k] \mid t_i = r \times b_i \ mod\ n$
Select $a = permute(t)$
The public key is the k-tuple a
The private key is n, r, and the k-tuple b

The superincreasing tuple

$$b = [7, 11, 19, 39, 79, 157, 313]$$

$n = 900$

$r = 37$

$$t = [259, 407, 703, 543, 223, 409, 781]$$

$$a = permute\ (t) = [543, 407, 223, 703, 259, 781, 409]$$

Knapsack Cryptosystem

- Example:

Key generation
Create $b = [b_1, b_2, \dots, b_k]$
Choose a modulus n
Select r that is co-prime with n and $1 \leq r \leq n - 1$
Create $t = [t_1, t_2, \dots, t_k] \mid t_i = r \times b_i \ mod \ n$
Select $a = permute(t)$
The public key is the k-tuple a
The private key is n, r, and the k-tuple b

The superincreasing tuple

$$b = [7, 11, 19, 39, 79, 157, 313]$$

$n = 900$

$r = 37$

$t = [259, 407, 703, 543, 223, 409, 781]$

$a = permute\ (t) = [543, 407, 223, 703, 259, 781, 409]$

The public key:

$a = [543, 407, 223, 703, 259, 781, 409]$

The private key:

$n = 900, r = 37, b = [7, 11, 19, 39, 79, 157, 313]$

Knapsack Cryptosystem

- Example: Suppose Alice wants to send a single character “g” to Bob.

Encryption
Convert the message to k-tuple x
Calculate s using $kanpsackSum$
Send the value of s as the ciphertext

Use the 7-bit ASCII of the letter “g”

$x = g = [1,1,0,0,1,1,1]$

Knapsack Cryptosystem

- Example: Suppose Alice wants to send a single character "g" to Bob.

Encryption
Convert the message to k-tuple x
Calculate s using $kanpsackSum$
Send the value of s as the ciphertext

Use the 7-bit ASCII of the letter "g"
$x = g = [1,1,0,0,1,1,1]$

We have $a = [543, 407, 223, 703, 259, 781, 409]$
$s = kanpsackSum(a, x)$
$s = 1\times 543 + 1\times 407 + \ \ldots + 1\times 409 = 2165$

Knapsack Cryptosystem

- Example: Suppose Alice wants to send a single character “g” to Bob.

Encryption
Convert the message to k-tuple x
Calculate s using $kanpsackSum$
Send the value of s as the ciphertext

Use the 7-bit ASCII of the letter “g”
$x = g = [1,1,0,0,1,1,1]$

We have $a = [543, 407, 223, 703, 259, 781, 409]$
$s = kanpsackSum(a, x)$
$s = 1\times543 + 1\times407 + \ ... + 1\times409 = 2165$

Ciphertext = $s = 2165$

Knapsack Cryptosystem

- Example: Bob receives $s = 2165$

Decryption
Calculate $s` = r^{-1} \times s \ mod \ n$
Compute $x` = inv\_kanpsackSum$
Compute plaintext = $x = permute(x`)$

We have $r = 37, \mathrm{n} = 900$

$r^{-1} = modInv(37, 900) = 73$

$s` = 73 \times 2165 \ mod \ 900 = 527$

Knapsack Cryptosystem

- Example: Bob receives $s = 2165$

Decryption
Calculate $s` = r^{-1} \times s \ mod \ n$
Compute $x` = inv\_kanpsackSum$
Compute plaintext = $x = permute(x`)$

We have $r = 37, \mathrm{n} = 900$

$$r^{-1} = modInv(37, 900) = 73$$

$$s` = 73 \times 2165 \ mod \ 900 = 527$$

$$x` = Inv\_kanpSack(s`, b) = [1, 1, 0, 1, 0, 1, 1]$$

Knapsack Cryptosystem

- Example: Bob receives $s = 2165$

Decryption
Calculate $s` = r^{-1} \times s \ mod \ n$
Compute $x` = inv\_kanpsackSum$
Compute plaintext = $x = permute(x`)$

We have $r = 37, \mathrm{n} = 900$

$r^{-1} = modInv(37, 900) = 73$

$s` = 73 \times 2165 \ mod \ 900 = 527$

$x` = inv\_kanpSack(s`, b) = [1, 1, 0, 1, 0, 1, 1]$

$x = permute(x`) = [1,1, 0,0,1,1,1]$
Plaintext = (1100111) = ”g”

TASK

- Implement the Knapsack cryptosystem.
- Implement RSA.