Diyala University
College of Engineering
Computer & Software
Engineering Department

Fourth Year 2012/2013

Public Key Cryptography

Chapter 3/Part1

PRESENTED BY
DR. ALI J. ABBOUD

3.1 Objectives

- History
- Mathematical Background
  - Arithmetic Modulus
  - Remainder Arithmetic
  - Factors
  - Prime and Compound Numbers
  - Power and base
  - Laws of Indices
  - Modular Inverse
- Public Key Cryptography
- Markel's Knapsack Algorithm
- RSA Algorithm

3.2 History

For many centuries secret messages had to be transmitted by using a key and/or method known only to those who were meant to share in the contents of those messages. Clearly, with such systems, there were always difficulties in distributing these keys or systems so that they did not fall into the wrong hands.

A breakthrough was made (in 1977) by Rivest, Shamir and Adleman (which is why the initials RSA are often attached to this system), when they devised a sytem using two keys. One key is used to put the message into cipher, and this key can be broadcast to the world so there is no distribution problem. This key is known as the **Public Key**. In addition to the Public Key another number (known as the *modulus)* is also published. The other **Key**, which is needed to decipher the message, is kept secret by the individual(s) for whom the message(s) is, or are, intended.

The system, based on some relatively simple ideas in **modulo arithmetic**, will be explained here by means of a numerical example, using only the smallest numbers it is possible to use. First of all it is necessary to set up the necessary numbers which will be used, by following this routine.

3.3 Mathematical Background

- Arithmetic Modulus
- Remainder Arithmetic
- Factors
- Prime and Compound Numbers
- Power and Base
- Laws of Indices
- Modular Inverse

3.3.1 Arithmetic Modulus

Modulo arithmetic is also known as **clock arithmetic**, or **remainder arithmetic**, and there is a very good reason for both of those names as we shall see.
Modulo arithmetic is a form of arithmetic which uses only a limited set of the whole numbers {0, 1, 2, 3, 4, 5, 6, 7 . . . . . }. It is always defined by the size of the limited set to be used, and that size is called the *modulus.*
A modulus of n means that the first n elements of the whole-number set must be used.

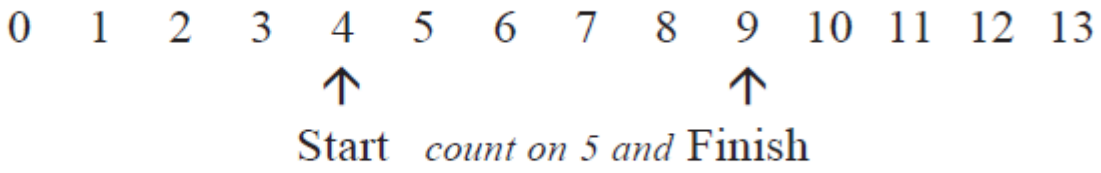
For example: A modulus of 3 means use 0, 1, 2
A modulus of 6 means use 0, 1, 2, 3, 4, 5
A modulus of 20 means use 0, 1, 2, 3, 4, 5 . . . . 17, 18, 19

Note

- The set to be used always starts with 0
- No numbers may be left out
- The set ends with the number which is **1 less** than the modulus

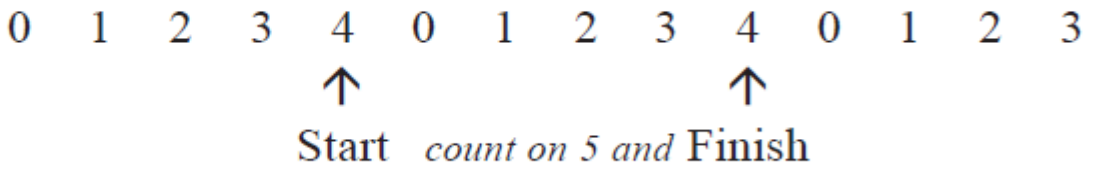
3.3.1 Arithmetic Modulus

For example: 4 + 5 can be modelled as

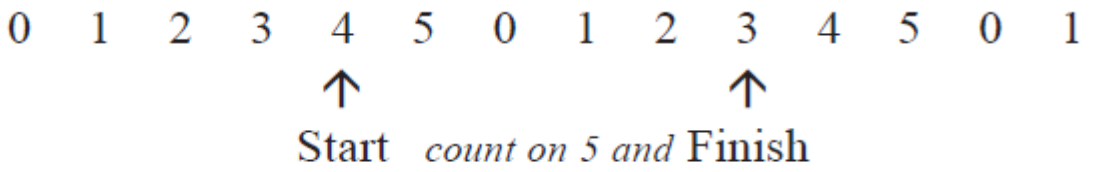


In modulo arithmetic the equivalent arrangement of the number line requires the same limited set of numbers to be repeated

For example: In modulo 5



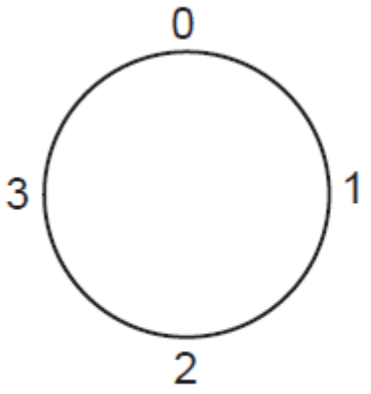
or in modulo 6



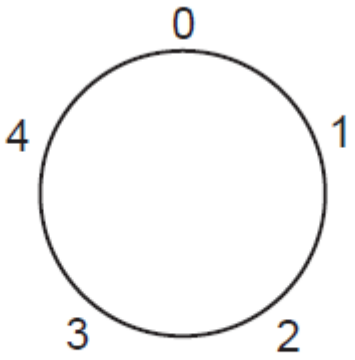
Note

- The answer clearly depends upon the size of the modulus
- The starting number must be less than the modulus *(for the moment)*
- The number of places to be counted on can be bigger than the modulus

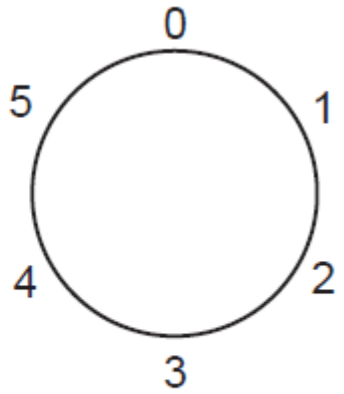
3.3.1 Arithmetic Modulus



modulo 4



modulo 5



modulo 6

For example: What is 6 + 8 in modulo 5?

6 = 0 + 6 so, starting at 0 and counting on 6 places finishes at 1

8 = 0 + 8 so, starting at 0 and counting on 8 places finishes at 3

And 6 + 8 becomes 1 + 3 in module 5 which is 4.

Another way it can be done is by first adding the given numbers (6 + 8) in the usual way (= 14) and then changing the answer into modulo 5

14 = 0 + 14 so, starting at 0 and counting on 14 places finishes at 4

3.3.1 Arithmetic Modulus

modulo **4**

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

modulo **5**

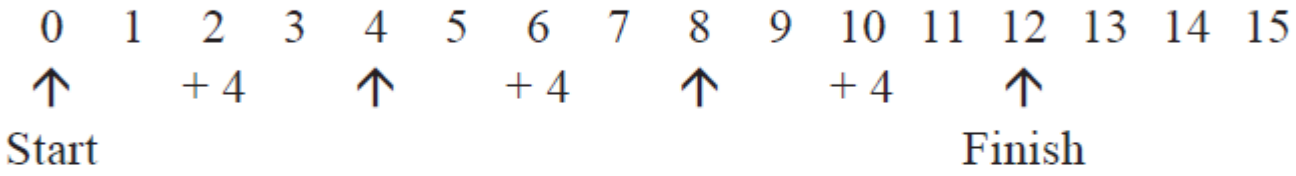
+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

modulo **6**

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

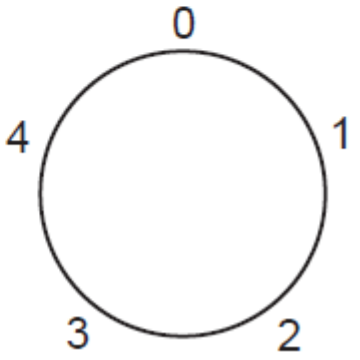
3.3.1 Arithmetic Modulus

As with addition, we must first see how multiplication works in 'normal' arithmetic.
Consider the statement 3×4
This means, "put together 3 lots of 4" *(or 4 lots of 3)*
In other words, 3×4 is a short way of writing $4 + 4 + 4$ *(or 3 + 3 + 3 +3)*
On the number line, 4 + 4 + 4 can be modelled as



And we can see that the answer is 12 *(which is hardly a surprise!)*
To do the same sum in modulo arithmetic needs the modulus to be stated.

We will evaluate 3×4 in modulo 5 by counting 4 + 4 + 4 on a modulo 5 clock
First, 4 + 4 takes us to 3
then, 3 + 4 takes us to 2
So, 3×4 (*mod* 5) is 2



[*One for you.* Count 3 + 3 + 3 + 3 on the same clock]

3.3.1 Arithmetic Modulus

modulo **4**

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

modulo **5**

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

modulo **6**

×	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

modulo **4**

x	x^2	x^3	x^4	x^5
0	0	0	0	0
1	1	1	1	1
2	0	0	0	0
3	1	3	1	3

modulo **5**

x	x^2	x^3	x^4	x^5
0	0	0	0	0
1	1	1	1	1
2	4	3	1	2
3	4	2	1	3
4	1	4	1	4

modulo **6**

x	x^2	x^3	x^4	x^5
0	0	0	0	0
1	0	1	1	1
2	4	2	4	2
3	3	3	3	3
4	4	4	4	4
5	1	5	1	5

3.3.2 Remainder Arithmetic

Suppose we want 4×5 in modulo 6

We know that $4 \times 5 = 20$, but what is it in modulo 6?

Thinking of the modulo 6 clock, starting at 0, every time we move 6 places we get back to 0

So, 6, 12, 18 will all get us back to 0,
which leaves only 2 places more to get to 20

This is the same as saying,

"Count in 6's and stop when you are about go past the number you have (in this case 20), then whatever you have left (in this case 2) will be the number you want."

Or, in a much shorter phrase:

"Divide by 6 and keep the **remainder**."

$20 \div 6 = 3$ remainder 2

It is this 'trick' which gives modulo arithmetic its other name of **remainder arithmetic**

Formally it is written:

$$20 \equiv 2 \pmod 6$$

Note the symbol is $\equiv$ which is read as "is congruent to"
and not $=$ which is read as "equals"

3.3.2 Remainder Arithmetic

One number is said to be a **multiple** of another number if the first number is equal to the second number multiplied by some whole number.

For example: 12 is a multiple of 4 since $12 = 4 \times 3$
20 is a multiple of 5 since $20 = 5 \times 4$

Note

- A number is considered to be a multiple of itself since $x = x \times 1$

Any division sum is made up of 4 parts, all of which are named.
The number which has to be divided, or shared out, is called the **dividend**.
The number which must do the dividing, is called the **divisor**.
The number giving the answer, is called the **quotient**.
The number giving the amount left over, is called the **remainder**.

dividend ÷ divisor = quotient + remainder

For example: In the sum $27 \div 4 = 6$ with 3 left over
27 is the dividend
4 is the divisor
6 is the quotient
3 is the remainder.

Note

- The remainder can be zero.

3.3.3 Factors

One number is said to be a **factor** of another number if it divides into it exactly.

For example: 3 is a factor of 6; 4 is a factor of 12;
2 is a factor of 18; and so on

Note

- 1 is a factor of ALL other numbers.
- Every number is a factor of itself.

A number may have several factors.

For example: 12 has the factors 1, 2, 3, 4, 6, 12
16 has the factors 1, 2, 4, 8, 16
25 has the factors 1, 5, 25
17 has the factors 1, 17

Note

- Every number, except 1, has **at least two** factors.

3.3.4 Prime and Compound Numbers

A **prime number** is a number which has two, and only two, factors.

For example: The first 15 prime numbers are

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Note

- 1 is NOT a prime number (it has only one factor).
- There is no end to the list of prime numbers.
- Numbers (other than 1) which are NOT prime are **compound numbers**.
- Prime numbers are usually called just 'primes'.

Primes are thought of as the 'building blocks' for numbers in the sense that all the other numbers can be made from them by using multiplication.

For example:

$12 = 2 \times 2 \times 3$ which can be written $2^2 \times 3$

$20 = 2 \times 2 \times 5$ or $2^2 \times 5$

$25 = 5 \times 5$ or 5^2

$15600 = 2 \times 2 \times 2 \times 2 \times 3 \times 5 \times 5 \times 13$ or $2^4 \times 3 \times 5^2 \times 13$

Note

- There is only ever **one** way this can be done.
- A re-arrangement of the primes is NOT a different way.

3.3.5 Power and Base

In working only with positive whole numbers a **power**, or **index**, is written as a superscript to some other number to indicate how many of the other numbers are to be multiplied together. The other number is called the **base**.

For example: In 2^3 the power is 3 and the base is 2;

so it means that 3 lots of 2 have to be multiplied together.

$2 \times 2 \times 2$ which = 8

3^2 means 3×3 which = 9

5^2 means 5×5 which = 25

3^3 means $3 \times 3 \times 3$ which = 27

7^5 means $7 \times 7 \times 7 \times 7 \times 7$ which = 16807

Note

- If the power is 0, the answer is always 1 ($958^0 = 1$)
- If the power is 1, the answer is the number itself ($37^1 = 37$)

3.3.6 Laws of Indices

An operation (such as + - × ÷) which combines two numbers is said to be **commutative** if the order in which the two numbers are placed makes no difference to the answer.

For example: addition **is** commutative since $3 + 4 = 4 + 3$
multiplication **is** commutative since $2 \times 5 = 5 \times 2$
subtraction **is not** commutative since $7 - 1 \neq 1 - 7$
division **is not** commutative since $6 \div 3 \neq 3 \div 6$

The three principal rules which determine how numbers written using index notation may be combined are known as the **laws of indices**.

They are

$$b^m \times b^n = b^{m+n} \qquad b^m \div b^n = b^{m-n} \qquad (b^m)^n = b^{mn}$$

Two special cases which follow from these are

$$b^0 = 1 \qquad b^{-n} = \frac{1}{b^n}$$

For example: $2^3 \times 2^6 = 2^{3+6} = 2^9 = 512$ $\qquad 2^5 \div 2^2 = 2^{5-2} = 2^3 = 8$

Note

- The two numbers being combined must have the same values for b

3.3.7 Modular Inverse

In modular arithmetic, the **modular multiplicative inverse** of an integer *a* modulo *m* is an integer *x* such that

$$a^{-1} \equiv x \pmod{m}.$$

That is, it is the multiplicative inverse in the field of integers modulo *m*, denoted $\mathbb{Z}_m$. This is equivalent to

$$ax \equiv aa^{-1} \equiv 1 \pmod{m}.$$

Some applications may include *x* outside $\mathbb{Z}_m$.

The multiplicative inverse of *a* modulo *m* exists if and only if *a* and *m* are coprime (i.e., if gcd(*a*, *m*) = 1). If the modular multiplicative inverse of *a* modulo *m* exists, the operation of division by *a* modulo *m* can be defined as multiplying by the inverse, which is in essence the same concept as division in the field of reals.

Suppose we wish to find modular multiplicative inverse *x* of 3 modulo 11.

$$3^{-1} \equiv x \pmod{11}$$

This is the same as finding *x* such that

$$3x \equiv 1 \pmod{11}$$

Working in $\mathbb{Z}_{11}$ we find one value of *x* that satisfies this congruence is 4 because

$$3(4) = 12 \equiv 1 \pmod{11}$$

and there are no other values of *x* in $\mathbb{Z}_{11}$ that satisfy this congruence. Therefore, the modular multiplicative inverse of 3 modulo 11 is 4.

Once we have found the inverse of 3 in $\mathbb{Z}_{11}$, we can find other values of *x* in $\mathbb{Z}$ that also satisfy the congruence. They may be found by adding multiples of *m* = *11* to the found inverse. Generalizing, all possible *x* for this example can be formed from

$$4 + (11 \cdot z), z \in \mathbb{Z}$$

yielding {...,-18,-7,**4**,15,26,...}.

3.4 Public Key Cryptography

- ***Public-key algorithms*** are *asymmetric* algorithms and, therefore, are based on the use of two different keys, instead of just one. In public-key cryptography, the two keys are called the *private key* and the *public key*

- **Private key**: This key must be know *only* by its owner.

- **Public key**: This key is known to everyone (it is *public*)

- ***Relation between both keys***: What one key encrypts, the other one decrypts, and vice versa. That means that if you encrypt something with my public key (which you would know, because it's public :-), I would need my private key to decrypt the message.
- ***Public Key Cryptography can be used for***:
  1) Data Encryption.
  2) Digital signatures.
  3) Digital certificates.
  4) Key Exchange.

3.4 Public Key Cryptography

Table 9.2. Applications for Public-Key Cryptosystems

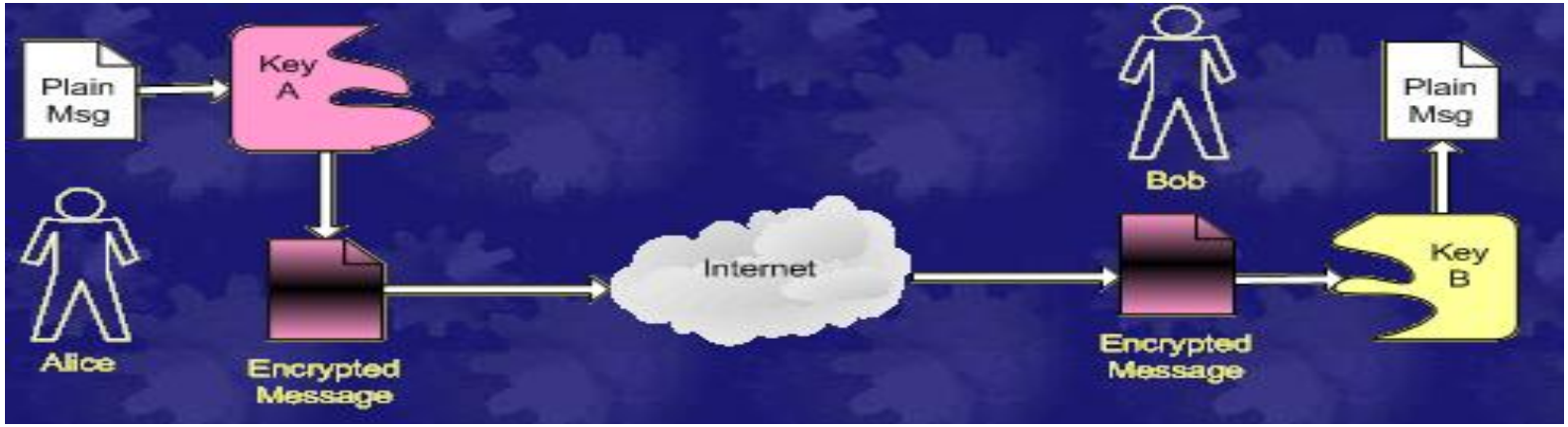
Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

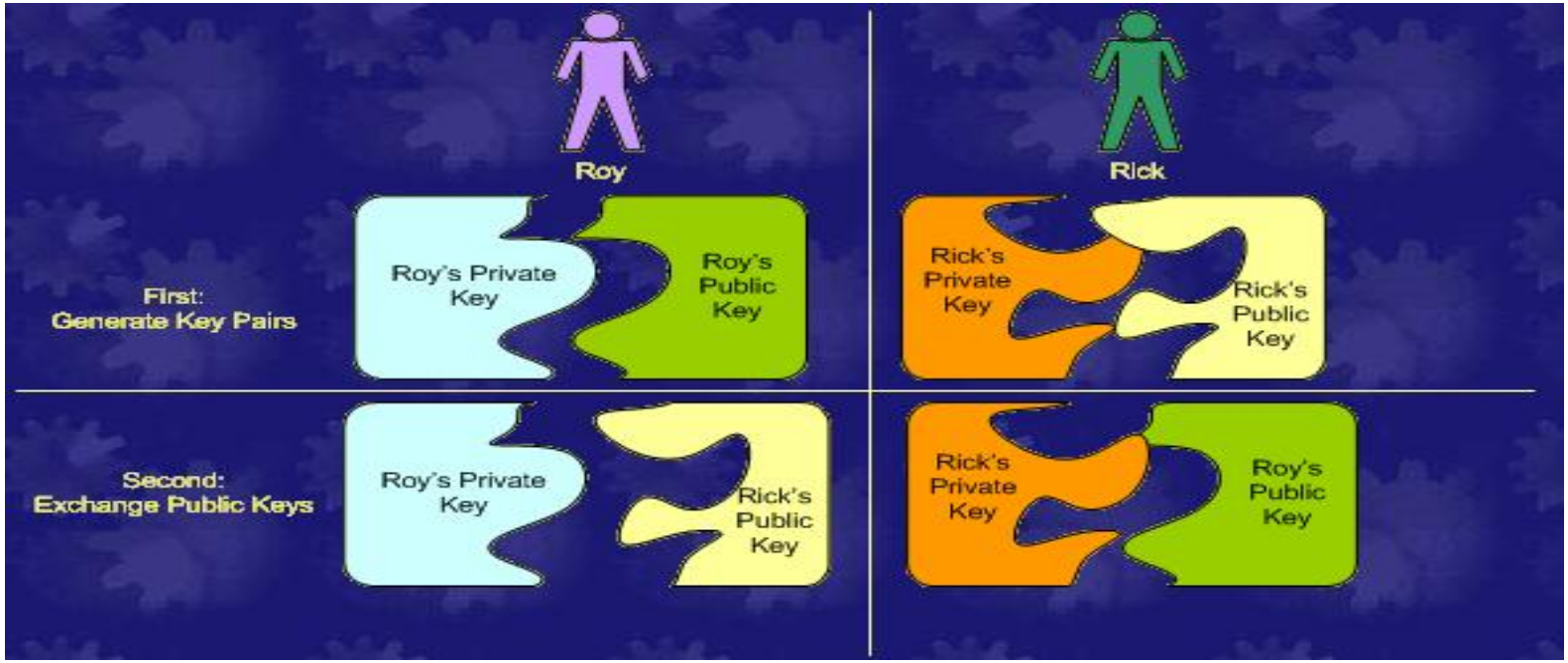
3.4 Public Key Cryptography

Table 9.1. Conventional and Public-Key Encryption

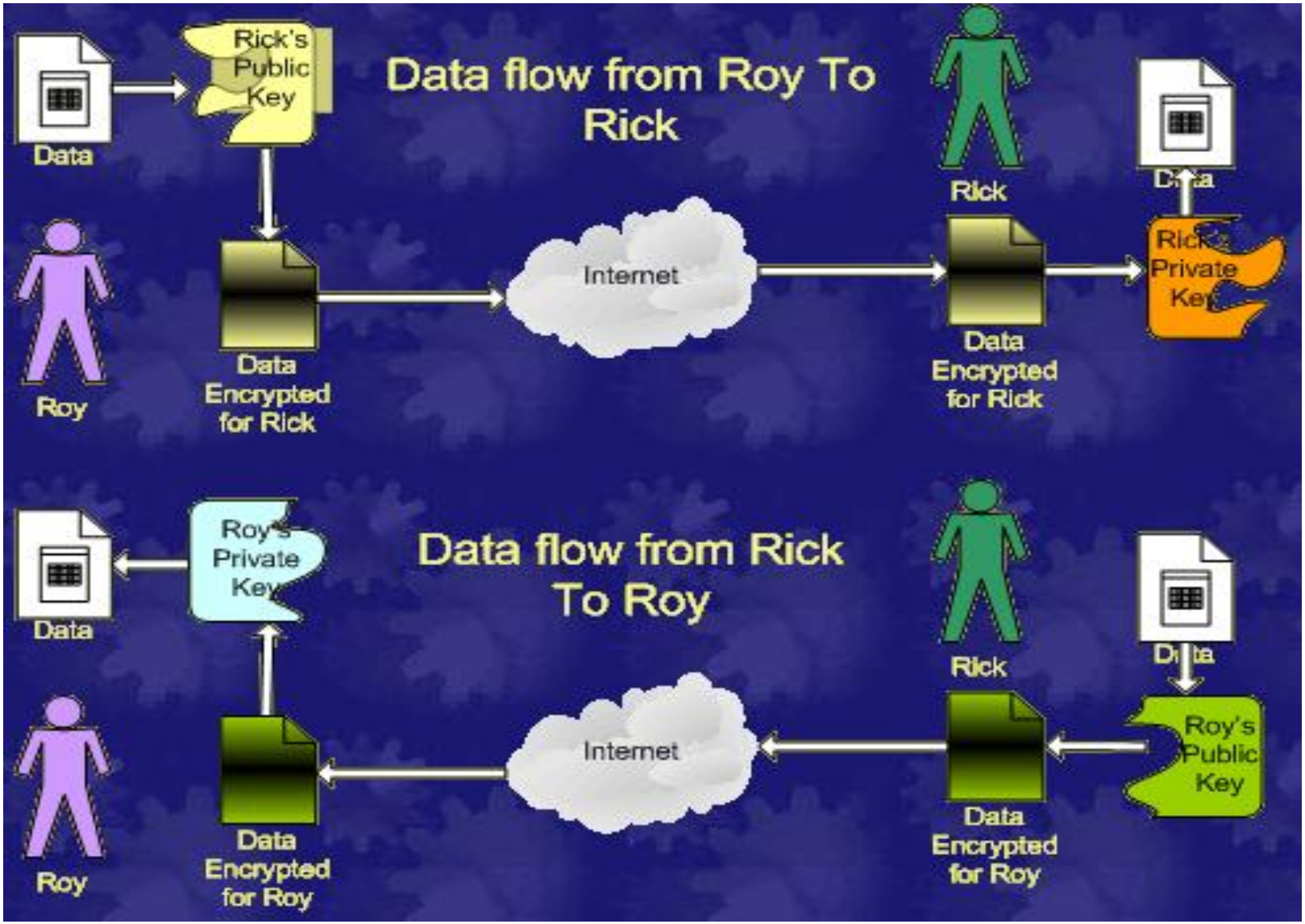
Conventional Encryption	Public-Key Encryption
Needed to Work:	**Needed to Work:**
1. The same algorithm with the same key is used for encryption and decryption. 2. The sender and receiver must share the algorithm and the key.	1. One algorithm is used for encryption and decryption with a pair of keys, 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:	**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.	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.

3.4 Public Key Cryptography





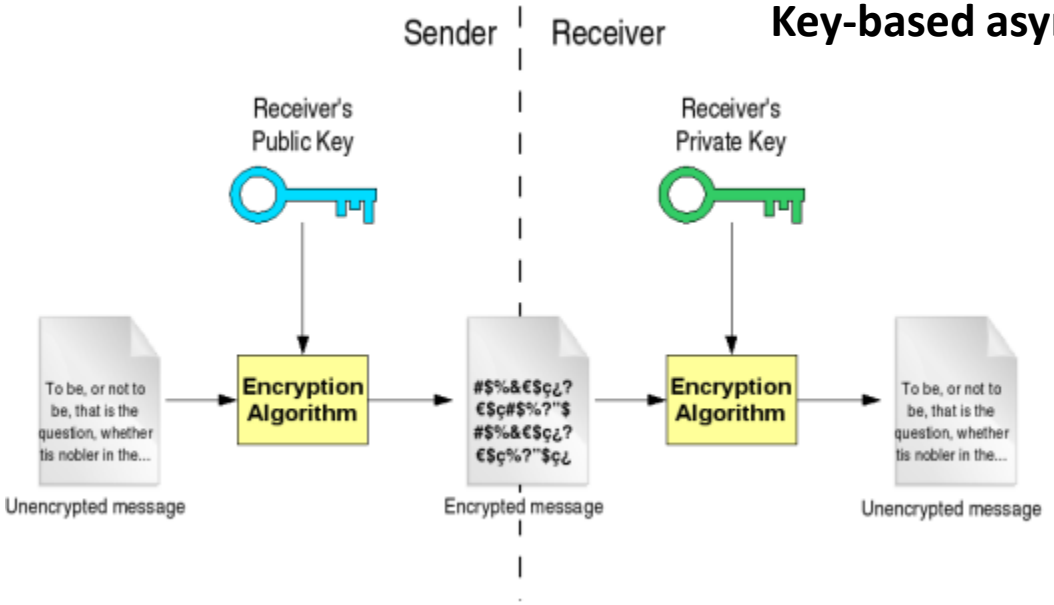
3.4 Public-Key Cryptography



3.4.1 A secure conversation using public-key cryptography

- In a basic secure conversation using public-key cryptography, the sender encrypts the message using the receiver's public key. Remember that this key is known to everyone.
- The encrypted message is sent to the receiving end, who will decrypt the message with his private key. Only the receiver can decrypt the message because no one else has the private key. Also, notice how the encryption algorithm is the same at both ends: what is encrypted with one key is decrypted with the other key using the same algorithm.

Key-based asymmetric algorithm

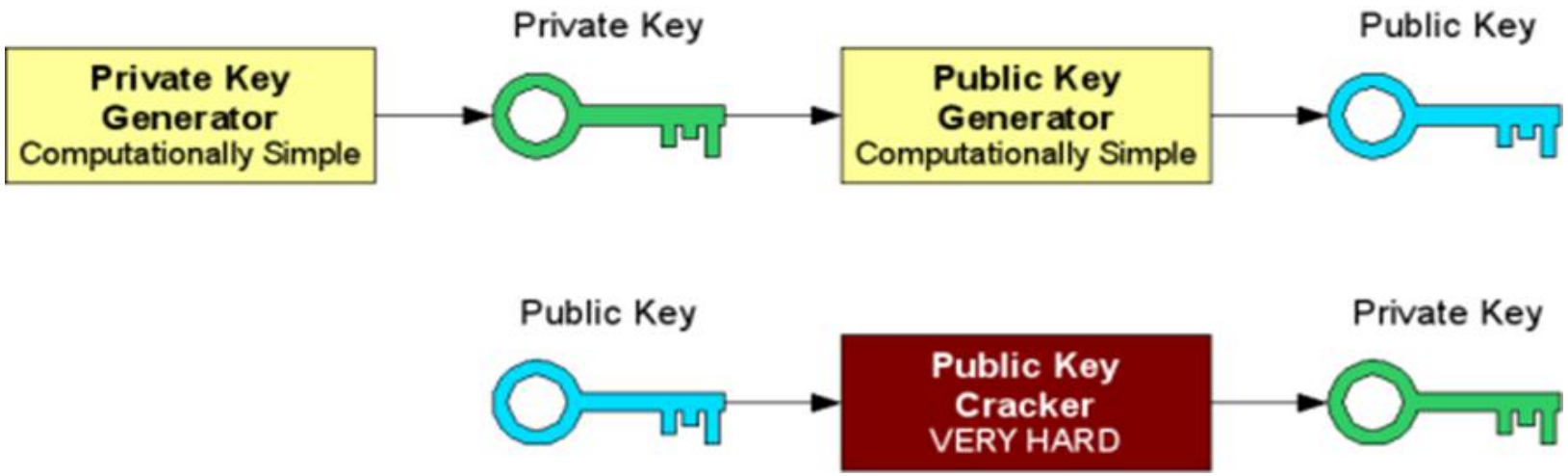


3.4.2 Pros and Cons of Public-Key Systems

- Public-key systems have a clear advantage over symmetric algorithms: there is no need to agree on a common key for both the sender and the receiver.
- As seen in the previous example, if someone wants to receive an encrypted message, the sender only needs to know the receiver's public key (which the receiver will provide; publishing the *public* key in no way compromises the secure transmission).
- As long as the receiver keeps the private key secret, no one but the receiver will be able to decrypt the messages encrypted with the corresponding public key.
- This is due to the fact that, in public-key systems, it is relatively easy to compute the public key from the private key, but *very hard* to compute the private key from the public key (which is the one everyone knows). In fact, some algorithms need several *months* (and even years) of constant computation to obtain the private key from the public key.

3.4.2 Pros and Cons of Public-Key Systems

- Another important advantage is that, unlike symmetric algorithms, public-key systems can guarantee integrity and authentication, not only privacy. The basic communication seen above only guarantees privacy. We will shortly see how integrity and authentication fit into public-key systems.
- The main disadvantage of using public-key systems is that they are not as fast as symmetric algorithms.



3.5 Merkle's Knapsack Algorithm

- The first algorithm for generalized public-key encryption was the knapsack algorithm developed by Ralph Merkle and Martin Hellman.

- It could only be used for encryption, although Adi Shamir later adapted the system for digital signatures.

- Knapsack algorithms get their security from the knapsack problem, an NP complete problem. Although this algorithm was later found to be insecure, it is worth examining because it demonstrates how an NP-complete problem can be used for public-key cryptography.

3.5 Merkle's Knapsack Algorithm

- The knapsack problem is a simple one. Given a pile of items, each with different weights, is it possible to put some of those items into a knapsack so that the knapsack weighs a given amount?

- More formally: Given a set of values M1 , M2 ,..., Mn, and a sum S, compute the values of bi such that S= b1M1+ b2M2+ ...+ bnMn

- The values of bi can be either zero or one. A one indicates that the item is in the knapsack; a zero indicates that it isn't.

- For example, the items might have weights of 1, 5, 6, 11, 14, and 20. You could pack a knapsack that weighs 22; use weights 5, 6, and 11. You could not pack a knapsack that weighs 24. In general, the time required to solve this problem seems to grow exponentially with the number of items in the pile.

- The idea behind the Merkle-Hellman knapsack algorithm is to encode a message as a solution to a series of knapsack problems. A block of plaintext equal in length to the number of items in the pile would select the items in the knapsack (plaintext bits corresponding to the b values), and the ciphertext would be the resulting sum.

3.5 Merkle's Knapsack Algorithm

- What is the easy knapsack problem? If the list of weights is a superincreasing sequence, then the resulting knapsack problem is easy to solve. A superincreasing sequence is a sequence in which every term is greater than the sum of all the previous terms. For example, {1, 3, 6, 13, 27, 52} is a superincreasing sequence, but {1, 3, 4, 9, 15, 25} is not.

- The solution to a superincreasing knapsackis easy to find. Take the total weight and compare it with the largest number in the sequence. If the total weight is less than the number, then it is not in the knapsack. If the total weight is greater than or equal to the number, then it is in the knapsack.

- Reduce the weight of the knapsack by the value and move to the next largest number in the sequence. Repeat until finished. If the total weight has been brought to zero, then there is a solution. If the total weight has not, there isn't.

3.5 Merkle's Knapsack Algorithm

For example, consider a total knapsack weight of 70 and a sequence of weights of {2, 3, 6, 13, 27, 52}.

- The largest weight, 52, is less than 70, so 52 is in the knapsack.
- Subtracting 52 from 70 leaves 18. The next weight, 27, is greater than 18, so 27 is not in the knapsack. The next weight, 13, is less than 18, so 13 is in the knapsack.
- Subtracting 13 from 18 leaves 5. The next weight, 6, is greater than 5, so 6 is not in the knapsack.
- Continuing this process will show that both 2 and 3 are in the knapsack and the total weight is brought to 0, which indicates that a solution has been found.
- Were this a Merkle-Hellman knapsack encryption block, the plaintext that resulted from a ciphertext value of 70 would be 110101.

3.5.1 Creating the Public Key from the Private Key of Merkle's Knapsack Algorithm

- To get a normal knapsack sequence, take a superincreasing knapsack sequence, for example {2, 3, 6, 13, 27, 52}, and multiply all of the values by a number n,mod m. The modulus should be a number greater than the sum of all the numbers in the sequence: for example, 105. The multiplier should have no factors in common with the modulus: for example, 31.

- The normal knapsack sequence would then be

  2 * 31 mod 105 = 62
  3 * 31 mod 105 = 93
  6 * 31 mod 105 = 81
  13 * 31 mod 105 = 88
  27 * 31 mod 105 = 102
  52 * 31 mod 105 = 37

- The knapsack would then be {62, 93, 81, 88, 102, 37}.

- The superincreasing knapsack sequence is the private key.

- The normal knapsack sequence is the public key.

3.5.2 Encryption for Merkle's Knapsack Algorithm

To encrypt a binary message, first break it up into blocks equal to the number of items in the knapsack sequence. Then, allowing a one to indicate the item is present and a zero to indicate that the item is absent, compute the total weights ofthe knapsacks—one for every message block.

For example, if the message were 0110001101011011 10 in binary, encryption using the previous knapsack would proceed like this:

message = 011000 110101 101110

011000 corresponds to 93 + 81 = 174

110101 corresponds to 62 + 93 + 88 + 37 = 280

101110 corresponds to 62 + 81 + 88 + 102 = 333

The ciphertext would be
174,280,333

3.5.3 Decryption for Merkle's Knapsack Algorithm

- A legitimate recipient of this message knows the private key: the original superincreasing knapsack, as well as the values of n and m used to transform it into a normal knapsack. To decrypt the message, the recipient must first determine n-1 such that n(n-1) 1 (mod m). Multiply each of the ciphertext values by n-1 mod m, and then partition with the private knapsack to get the plaintext values.

- In our example, the superincreasing knapsack is {2, 3, 6, 13, 27, 52}, m is equal to 105, and n is equal to 31. The ciphertext message is 174, 280, 333. In this case n-1 is equal to 61, so the ciphertext values must be multiplied by 61 mod 105.

  174 * 61 mod 105 = 9 = 3 + 6, which corresponds to 011000

  280 * 61 mod 105 = 70 = 2 + 3 + 13 + 52, which corresponds to 110101

  333 * 61 mod 105 = 48 = 2 + 6 + 13 + 27, which corresponds to 101110

- The recovered plaintext is 011000 110101 101110.

3.6 RSA Algorithm

- One of the first successful responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. The Rivest-Shamir-Adleman (**RSA**) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

- The **RSA** scheme is a block cipher in which the plaintext and ciphertext are integers between (0) and (n – 1) for some (n). A typical size for (n) is (1024) bits, or (309) decimal digits. That is, n is less than 2^{1024} . We examine RSA in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of **RSA.**

- RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to $\log_2(n) + 1$; in practice, the block size is l bits, where $2^i < n \leq 2^{i+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C.

3.6.1 Greatest Common Divisor (gcd)

- The positive integer d is the greatest common divisor of integers a and b, denoted d=gcd(a,b) if
  - It is a divisor of both a and b
  - Any other divisor of a and b is a divisor of d
- Example: gcd(8,12)=4, gcd(24,60)=12
- Integers a and b are called *relatively prime* if gcd(a,b)=1
- Computing gcd(a,b): Euclid's algorithm
  - Based on the following fact: gcd(a,b)=gcd(b,a mod b)
- **Euclid's Algorithm** to compute gcd(a,b): Euclid(a,b)
  - `If b=0 then return a`
  - `Else return Euclid(b,a mod b)`
- Note: the algorithm always terminates

3.6.2 Euclidean Algorithm

Example: d=gcd(1970,1066)

1970 = 1 x 1066 + 904	d= gcd(1066, 904)
1066 = 1 x 904 + 162	d= gcd(904, 162)
904 = 5 x 162 + 94	d= gcd(162, 94)
162 = 1 x 94 + 68	d= gcd(94, 68)
94 = 1 x 68 + 26	d= gcd(68, 26)
68 = 2 x 26 + 16	d= gcd(26, 16)
26 = 1 x 16 + 10	d= gcd(16, 10)
16 = 1 x 10 + 6	d= gcd(10, 6)
10 = 1 x 6 + 4	d= gcd(6, 4)
6 = 1 x 4 + **2**	d= gcd(4, 2)
4 = 2 x 2 + 0	d= **2**

Result: *gcd(1970,1066)=2, i.e., the last nonzero residue in the above computation*

3.6.3 RSA Terms

The following notation is used consistently throughout:

- A public key is represented by (**e**).
- A private key is represented by (**d**).
- Plaintext message is represented by (**M**).
- Ciphertext message is represented by (**C**).
- Public key pair $PU = \{e, n\}$
- Private key pair $PR = \{d, n\}$
- Parameter is used to adjust RSA algorithm (**n**).

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

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of $PU = \{e, n\}$ and a private key of $PR = \{d, n\}$. 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.

3.6.4 RSA Key Generation

Key Generation Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p-1)(q-1)$	
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	$PR = \{d, n\}$

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
5. Determine d such that $de \equiv 1 \pmod{160}$ and $d < 160$. The correct value is $d = 23$, because $23 \times 7 = 161 = (1 \times 160) + 1$; d can be calculated using the extended Euclid's algorithm ·

3.6.5 RSA Encryption and Decryption

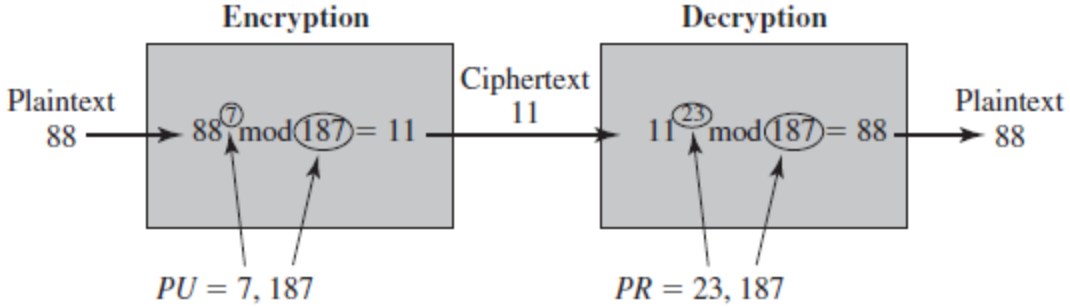
Encryption by Bob with Alice's Public Key

Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key

Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

The RSA Algorithm



Example of RSA Algorithm

3.6.6 RSA Encryption and Decryption

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$. The example shows the use of these keys for a plaintext input of $M = 88$. For encryption, we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows.

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$$

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

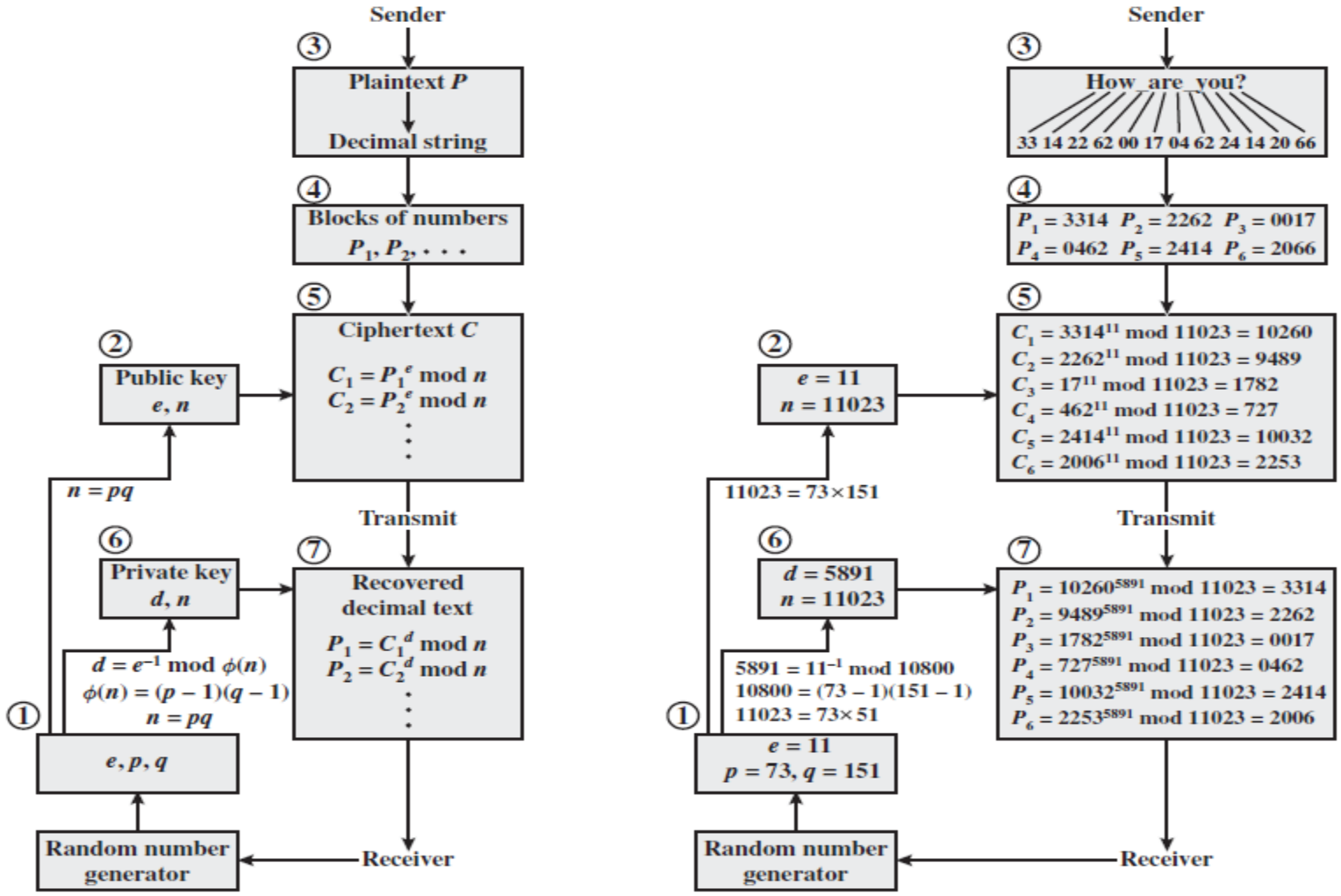
$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$$

3.6.7 RSA Processing of Multiple Blocks



RSA Processeing of Multiple Blocks

3.6.8 Security of RSA Algorithm

Four possible approaches to attacking the RSA algorithm are

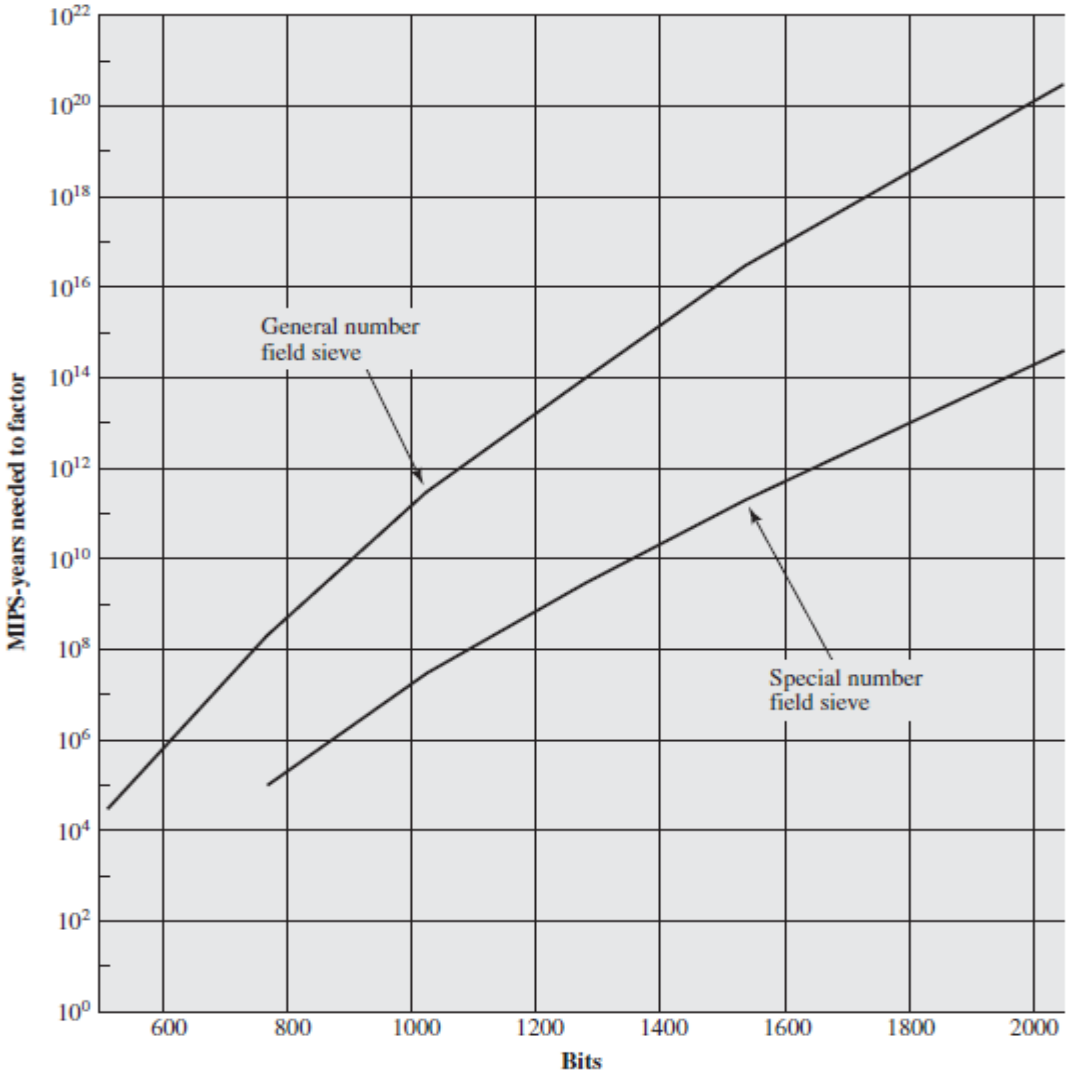
- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

3.6.8 Security of RSA Algorithm

Progress in Factorization

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

3.6.8 Security of RSA Algorithm



MIPS-years Needed to Factor

3.6.9 Procedure for Picking Random Number

The procedure for picking a prime number is as follows:

1. Pick an odd integer *n* at random (e.g., using a pseudorandom number generator).

2. Pick an integer $a < n$ at random.

3. Perform the probabilistic primality test, such as Miller-Rabin, with *a* as a parameter. If *n* fails the test, reject the value *n* and go to step 1.

4. If *n* has passed a sufficient number of tests, accept *n*; otherwise, go to step 2.

In next lecture, we will explain how:

1. To do primality test using Miller-Rabin method.

2. To find modular inverse using extended Euclid's algorithm.

End of Chapter 3/Part1