3. Public-key Cryptography

A. OVERVIEW

1. Introduction and learning objective

Symmetric encryption has evolved from classical to modern but also leaves a lot to be desired. Two of the most difficult problems associated with symmetric encryption are:

- *Key distribution* -the issue of secret key exchange between sender and receiver: Key distribution under symmetric encryption requires either (1) that two communicants already share a key, which somehow has been distributed to them; or (2) the use of a key distribution center. A secure channel is required for key exchange so that the key must be kept secret and known only to the sender and receiver. This will be difficult to implement and establishing such a secure channel will be costly and time-consuming.
- *Digital signatures* and confidentiality of keys: If the use of cryptography was to become widespread, not just in military situations but for commercial and private purposes, then electronic messages and documents would need the

equivalent of signatures used in paper documents. Besides, there is no basis to blame if the key is revealed.

Diffie and Hellman achieved an astounding breakthrough in 1976, by coming up with a method that addressed both problems and was radically different from all previous approaches to cryptography, going back over four millennia. That is public-key cryptography or asymmetric cryptography. The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography.

To discriminate between the two, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to as the **public key** (are often denoted as *PU*) and the **private key** (are often denoted as *PR*). The plaintext is denoted by M, the ciphertext is denoted by C

There are 2 main approaches in Public-key cryptography:

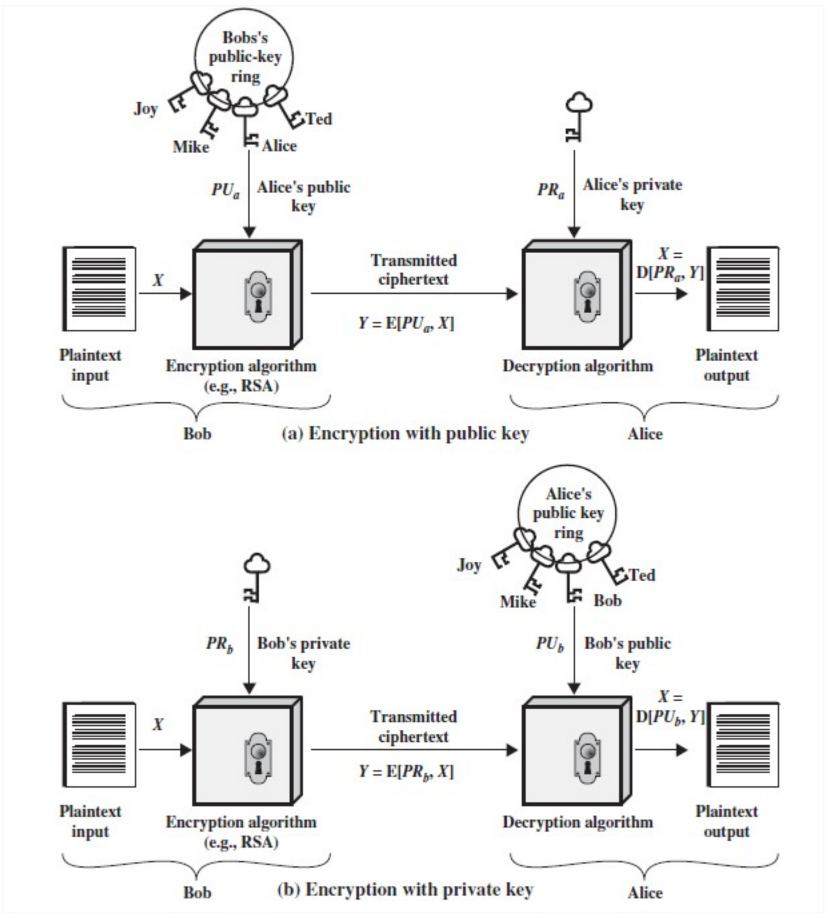

Figure 1: Two approaches in Public-Key Cryptography

- **Public-Key Cryptosystem for Confidentiality:**

  Encryption with the public key (Figure 1). If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

  • Encryption: **C = E(M,** ***PU*****)**

  • Decryption: **M = D(C,** ***PR*****)**

- **Public-Key Cryptosystem for Authentication:**

  Encryption with public key (Figure 3.1). In this case, Alice prepares a message to Bob and encrypts it using Alice's private key before transmitting it. Bob can decrypt the message using Alice's public key. Because the message was encrypted using Alice's private key, only Alice could have prepared the message. Therefore, the entire encrypted message serves as a digital signature.

  • Encryption: **C = E(M,** ***PR*****)**

  • Decryption: **M = D(C,** ***PU*****)**

The learning objective of this lab is for students to gain hands-on experience with the RSA and ECDSA algorithm. From lectures, students should have learned the theoretical part of these algorithms, so they know mathematically how to generate public/private keys and how to perform encryption/decryption and signature generation/verification. This lab enhances students' understanding of algorithms by requiring them to go through every essential step of the algorithm on actual numbers, so they can apply the theories learned from the class. Essentially, students will be implementing the RSA and ECDSA algorithm using a programming language.

2. Backgrounds and Prerequisites

To complete this task well, you are expected to gain knowledge about:

- Number Theory:
  - Modular Arithmetic.
  - Divisor, the greatest common divisor (Euclidian Algorithm).
  - Prime numbers, Testing for Primality.
- RSA Cipher.
- ECDSA Cipher.

B. LAB TASKS

1. Number Theory

Before starting with the public key encryption algorithm, we will kick-off with some number theory revision, specifically check for prime numbers, greatest common divisor (GCD), modulo.

Write a program (with C/C++/C#) to fulfil the following requirements:

1. Primary number:
   + Generate random prime numbers with 2 bytes, 8 bytes, and 32 bytes long.
   + Check if an arbitrary integer less than 2^{89}*-1* is prime or not.
2. Determine the greatest common divisor (gcd) of 2 arbitrary "large" integers (which are as large as possible that you can handle)
3. Compute the modular exponentiation $a^x \bmod p$. Your program should be able to compute in case of "large" exponents (x > 40), for example $7^{40} \bmod 19$

Tips:

- *To check large prime numbers, you can find out and use the Miller-Rabin[1] algorithm.*
- *To determine the greatest common divisor, you should use Euclidian algorithm.*
- *Modular multiplications has the following property:*

  $(a \times b) \equiv [(a \bmod n) \times (b \bmod n) \bmod n]$

  It can be applied in compute modular exponentiation in case of "large" exponents.

2. RSA Public-Key Encryption

RSA is one of the first public-key cryptosystems and is widely used for secure communication. This cipher was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978. In RSA scheme, 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 21024. RSA algorithm can be summarized as follows.

1. Select two "large" prime numbers p and q ($p \neq q$), then calculate n = pq
2. Calculate $\phi(n) = (p-1)(q-1)$
3. Select e such that e is relatively prime to $\phi(n)$ and less than $\phi(n)$

[1] https://asecuritysite.com/encryption/rabin

4. Determine d such that $e.d \equiv 1 \; mod \; \phi(n)$ *(d can be calculated using the extended Euclid's algorithm)*
5. The resulting keys are public key $PU = (e,n)$ and private key $PR = (d,n)$
6. To encrypt a plaintext input of M:

   + Encryption for Confidentiality: $C = E(M,PU) = M^e \; mod \; n$

   + Encryption for Authentication: $C = E(M,PR) = M^d \; mod \; n$
7. To decrypt ciphertext input of C:

   + Decryption for Confidentiality: $M = D(C, PR) = C^d \; mod \; n$

   + Decryption for Authentication: $M = D(C,PU) = C^e mod \; n$

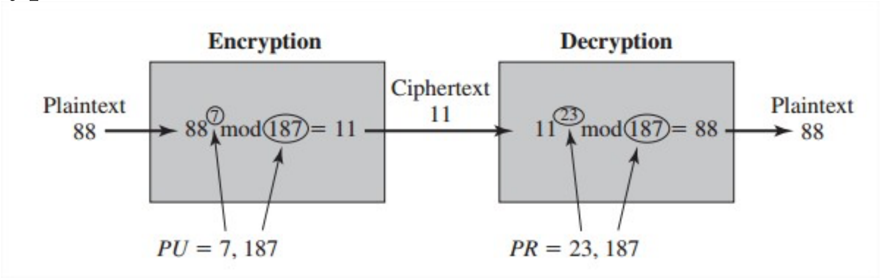

Figure 2: Example of RSA Algorithm

When using RSA to process multiple blocks of data, each plaintext symbol could be assigned a unique code of two decimal digits (e.g., a = 00, A = 26). A plaintext block consists of four decimal digits, or two alphanumeric characters. Figure 3.3.a illustrates the sequence of events for the encryption of multiple blocks, and Figure 3.3.b gives a specific example. The circled numbers indicate the order in which operations are performed.

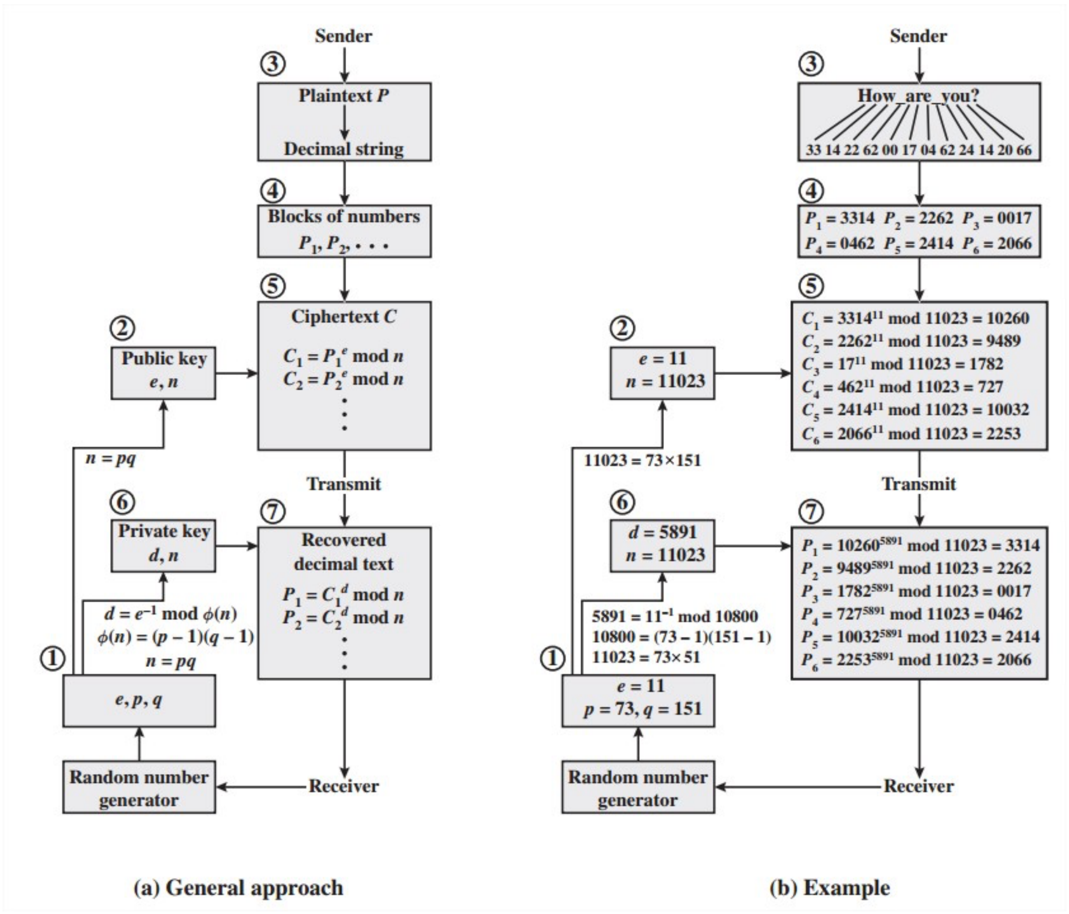

Figure 3: RSA Processing of Multiple Blocks

To get acquainted with RSA, use the useful tools to perform the following experiments:

1. Determine public key *PU* and private key *PR*, if:

   + p1 = 11, q1 = 17, e1 = 7 (decimal)

   + p2 = 20079993872842322116151219,
   q2 = 676717145751736242170789, e2 = 17 (decimal)

   + p3 = F7E75FDC469067FFDC4E847C51F452DF,
   q3 = E85CED54AF57E53E092113E62F436F4F,e3 = 0D88C3 (hexadecimal)
   (Note: Remember to check if the above values are prime numbers or not before calculating the keys)
2. Using key which generated by p1, q1, e1 to encrypt and decrypt the plaintext M=5 in both case Encryption for Confidentiality and Ecryption for Authentication
3. Use the keys above to encrypt the following message (ciphertext in hexadicimal)
   The Faculty of Computer Networks and Communications

3. RSA Application

Write a program to ***illustrate how simple RSA Cipher work***, meet the following requirements:

1. Generate a keypair (PU, PR) using the given "valid" inputs p, q, e or generate a randomized keypair if p, q, e are not given.
   Note that the keypair is as large as possible.
2. Use the generated keys to encrypt/decrypt the message. The message can be numeric or string

Using any cryptographic libraries is not allowed.

Tips:

- *You need to solve some problems when writing this application such as: generating/testing large prime number (can use Miller-Rabin algorithm); find the greatest common divisor (can use Euclidian's algorithm); applying modular multiplication and exponentiation properties to calculate "large" $a^x \bmod n$,…*

4. Programming using the Crypto Library (required)

Writing an encryption application (C++, using Crypto++ library) to fulfil the following requirements:

1. Support: ECDSA algorithm.
2. Support: signing function and verify function.
3. ECC curve: should select from standard curves.
4. Message to sign/Signature to verify:
   - Input from files (using filename)
   - *Support Vietnamese (using setmode, UTF-16) (bonus points)*
5. Secret key/public key
   - The keys load from files (for both two functions)
   - The public key: >=256 bits

Generate a set of different input sizes (at least 3 inputs). Execute your code and check the computation time on average 100 running times. Summarize the results in a table including the size of the input, mode of operation, encryption time and decryption time.

C. REQUIREMENTS

You are expected to complete all tasks in section B (Lab tasks). Advanced tasks are optional, and you could get bonus points for completing those tasks. We prefer you work in a team of 2 members to get the highest efficiency.

Your submission must meet the following requirements:

- You need to submit a **detailed lab report in .docx** *(Word Document)* format, **using the report template** provided on the UIT Courses website.
- Either Vietnamese or English report is accepted. That's up to you. Using more than one language in the report is not allowed (except for the untranslatable keywords).
- When it comes to **programming tasks** *(require you to write an application or script),* please attach all source-code and executable files (if any) in your submission. Please also list the important code snippets followed by explanations and screenshots when running your application. Simply attaching code without any explanation will not receive points.
- Submit work you are proud of – don't be sloppy and lazy!

  Your submissions must be your own. You are free to discuss with other classmates to find the solution. However, copying reports is prohibited, even if only a part of your report. Both reports of owner and copier will be rejected. Please remember to cite any source of the material (website, book,…) that influences your solution.

Notice: Combine your lab report and all related files into a single **ZIP file (.zip)**, name it as follow:

StudentID1_StudentID2_ReportLabX.zip

D. REFERENCES

[1] William Stallings, *Cryptography and network security: Principles and practice, 7th ed*, Pearson Education, 2017. *Chapter 4, chapter 6, chapter 7*

[2] Wenliang Du (Syracuse University), *SEED Cryptography Labs*
https://seedsecuritylabs.org/Labs_20.04/Files/Crypto_Encryption

[3] Crypto++ Library
https://cryptopp.com/index.html

Attention: *Don't share any materials (slides, readings, assignments, labs, etc..) out of our class without my permission!*