Đại học Quốc gia Thành phố Hồ Chí Minh
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO BÀI TẬP

Môn học: Mật mã học

Kỳ báo cáo: Buổi 03 (Session 03)

Tên chủ đề: Thuật toán mã hoá RSA

Ngày báo cáo: 23/04/2023

1. THÔNG TIN CHUNG:

Lớp: NT219.N22.ATCL.1

STT	Họ và tên	MSSV	Email
1	Đoàn Hải Đăng	21520679	21520679@gm.uit.edu.vn
2	Lê Thanh Tuấn	21520518	21520518@gm.uit.edu.vn
3	Phan Thị Hồng Nhung	21521250	21521250@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:

STT	Công việc	Kết quả tự đánh giá	Người đóng góp
1	Chậm lại và suy nghĩ 1	100%	Hồng Nhung
2	Bài tập 1	100%	Hồng Nhung
3	Bài tập 2	100%	Hồng Nhung
4	Bài tập 3	100%	Hải Đăng
5	Bài tập 4	100%	Hải Đăng
6	Bài tập 5	100%	Hải Đăng
7	Bài tập 6	90%	Thanh Tuấn
8	Bài tập 7	70%	Thanh Tuấn
9	Bài luyện tập 1	100%	Hải Đăng Hồng Nhung
10	Bài luyện tập 2	100%	Hồng Nhung Hải Đăng, Thanh Tuấn

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

BÁO CÁO CHI TIẾT

1. Chậm lại và suy nghĩ 1

☞ *GenerateRandomWithKeySize and InvertibleRSAFunction?*

- The GenerateRandomWithKeySize function is a function in a programming language or a library related to RSA encryption algorithm, and it is used to generate a random number with a specified length (size) for the key.
- InvertibleRSAFunction is responsible for generating a pair of RSA keys, including a public key and a private key. It performs necessary steps in the process of RSA key generation, including generating random numbers, calculating the public and private keys, and validating the generated keys for their validity.

2. Bài tập 1

☞ *Encrypt the following plaintext: "RSA Encryption Schemes".*

Here's what we have fixed:

```
static const int SECRET_SIZE = 23; // Update size to include null terminator
SecByteBlock plaintext( SECRET_SIZE );
memcpy(plaintext,"RSA Encryption Schemes", SECRET_SIZE); // Update plaintext
string ciphertextStr(reinterpret_cast<const char*>(ciphertext.data()), ciphertext.size());

// Print ciphertext to screen
cout << "Ciphertext: " << ciphertextStr << endl;
string recoveredText((const char*)recovered.data(), recovered.size());
cout << "Recovered plain text: " << recoveredText << endl;
```

Result:

```
PS D:\NT219_Cryptography\Crypto> .\sample.exe
Ciphertext: '▬▼M3╛▄Jò3]zn¥=SởK
↓ô╤«▀F╒╠]§-@f\)╫C!↓╚┤á╛♠╚s╥│|╛r T≈▒±DzÅS♠+VZ"┴ºN╗⌡E≈ö=
=û┘≡·╫ï.σ¬▀┴¢s└→>Ä╠╓U‼ùGÅ└f&n5╓µüN♥‼Ω3'╒¬▲╚üL º`huG
Recovered plain text: RSA Encryption Schemes
```

3. Bài tập 2

☞ *Use private key for encryption and public key for decryption.*

```
PS D:\NT219_Cryptography\Crypto> .\bt2_sample_RSA-SS.exe
Signature on message verified
PS D:\NT219_Cryptography\Crypto>
```

4. Bài tập 3

☞ *Ciphertext input from file.*

```
PS D:\Selfwork\Crypto> .\ex3.exe
Plaintext: RSA Encryption Schemes
V♠oo┘)⌐)≥▐0╥╩↔%O§|αo≤┘Z╝>ϕöBï╢╪ó╪]╚╬+Φ|↔%╢CL┐àKD┬▬÷û$c├Éuw╪EΣ0ê├g=)¡/⌂¡z└Ö▼┘C&@¥↔┴;♥gv6■¬└═♥<%X»╗║@ªí¶!┌æ╨l|o
Ciphertext read from file:
V♠oo┘)⌐)≥▐0╥╩↔%O§|αo≤┘Z╝>ϕöBï╢╪ó╪]╚╬+Φ|↔%╢CL┐àKD┬▬÷û$c├Éuw╪EΣ0ê├g=)¡/⌂¡z└Ö▼┘C&@¥↔┴;♥gv6■¬└═♥<%X»╗║@ªí¶!┌æ╨l|o
Recovered plain text: RSA Encryption Schemes
```

5. Bài tập 4

☞ *Plaintext UTF-16.*

```
PS D:\Selfwork\Crypto> .\ex4-5.exe
1. Input string using utf-8
2. Input string using utf-16
2
Enter plaintext: Mèo méo meo mèo meo
Plaintext: Mèo méo meo mèo meo
Encoded Ciphertext: NQO7wB/kbsSM0cYnVb0y57H5SX+VecJXgK71a9MrRUT9Ft8wwAGbwxIBumDrBIcroGN0vD5X
nusGqhX4OlejtgTlanRJvC5u+q+sMTXSr/aj3KheOh53Lzg1+NOptsn+fV46jpRXsTAvHOQa
YT8hWxAyicbGTxeCAjX5Y1NsE0o=

Recovered Text as UTF-16: Mèo méo meo mèo meo
```

6. Bài tập 5

☞ *Plaintext input from user.*

```
PS D:\Selfwork\Crypto> .\ex4-5.exe
1. Input string using utf-8
2. Input string using utf-16
1
Enter plaintext: Hello World
Plaintext: Hello World
Encoded Ciphertext: OVlZ2Z1/0aIryBau3AfoZnshFnmkeatLlXEaLNA7tsCjQdcWZ71aJlZfCnc1dpR/eJxgGTPk
jpGFoJAu7DfH8o6IjI2tR/Na0R4bOneB3VSa74ryQLAPdC/LaWo3sCwBKROIvLdIO0ku/66L
viGNR37i0jujm2Z5MZx/C3oZ/00=

Recovered Text: Hello World
```

7. Bài tập 6

☞ *Key load from file.*

```
PS D:\Selfwork\Crypto> .\ex6.exe
Enter plaintext: Lab3 RSA
Plain Text : Lab3 RSA
Cipher Text : 856DA47237805BA1FFAD296309AD39DF78AECBCF8518C7E1A5CE5D557A52C59FF72E4672A6692483B55D7A0EB5FAD1E372C23CBBB53F7E31E34ADBC8C2EAF8BEFB118E796FB5FA31731A
75C20485396202BCFF10C46F30F53FA5D4622332272F71AD1D8B88526752460D9A8BEB9ABF0268B942ED32D17A8C6EE26BA27680DE840BDD90D2923CB94B81DCBC2427019AE6B50641C2A6DEDBA87AD139
9CCEBEFFFB3368BE4788595E03491D0C512C75374E94A653A14819C62843D626B9DD8647B31BF67F4CE791144FEA6FCF08E25C222472D788C9F214D4FD9504698D2CE755095B9BC3C3A93B35F1CE542103
90C0298FF46F827E31F08D011736C5F2F14DB8AF4A160A357202FD93DAE540F35919842C8F02FE0DC2D7E48D7510EFFEF161178C83CD35A9B01125AAC72FC99979755C592F0BCC8B749E1AE8307E05938B
4C50B052CD130613A3C6ABFD4789EC61B53FFCF90ED675A283DCB55D266468CCF0F5C153913BEB1A14C43E0FA99FAAA8AB66A66E01C9A28CA528FC3460B2862E86317D
Recovered Text : Lab3 RSA
```

8. Bài tập 7

☞ *Prime Factorization Attack:*

The idea behind this attack is to factorize the modulus to obtain the private key, which can then be used to decrypt the encrypted messages.

Prime Factorization Attack is only effective against RSA with weak or small prime factors, and it is not feasible for large modulus values used in real-world RSA encryption.

1. Write a function to check if a number is prime and a function to find the greatest common divisor (GCD).

```cpp
bool isPrime(int num) {
    if (num <= 1) {
        return false;
    }
    for (int i = 2; i <= sqrt(num); i++) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}
```

```cpp
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
```

2. Write a function to perform the actual attack.

```
int primeFactorizationAttack(int n, int e) {
    int p, q; // prime factors of n
    int phi; // Euler's totient function value
    int d; // private exponent

    // Find prime factors p and q of n
    for (int i = 2; i < n; i++) {
        if (n % i == 0 && isPrime(i)) {
            p = i;
            q = n / i;
            break;
        }
    }
    phi = (p - 1) * (q - 1);

    // Find the multiplicative inverse of e modulo phi
    for (int i = 2; i < phi; i++) {
        if (gcd(i, phi) == 1) {
            d = i;
            break;
        }
    }

    return d;
}
```

3. In the main function, enter the modulus (n) and public exponent (e) values. Display the value of d as the output, which represents the private key.

```
int main() {
    int n, e, d; // RSA parameters: modulus (n), public exponent (e), private exponent (d)

    cout << "Enter modulus (n): ";
    cin >> n;

    cout << "Enter public exponent (e): ";
    cin >> e;

    // Perform prime factorization attack to find the private exponent (d)
    cout << "Performing prime factorization attack..." << endl;
    d = primeFactorizationAttack(n, e);
    cout << "Private Exponent (d): " << d << endl;

    return 0;
}
```

9. Bài luyện tập 1

☞ *Benchmark RSA algorithm*

- **Utf-16 data**

```
PS D:\Selfwork\Crypto> .\benchmark_utf16.exe
RSA/OAEP-MGF1(SHA-1) encrypt benchmarks...
  3.3 GHz cpu frequency
  1985.69 cycles per byte (cpb)
  1.58491 MiB per second (MiB)
```

- **100 MB data**

- It is difficult to encrypt a large file with an asymmetric algorithm like RSA
- It is easy to encrypt a large file with a symmetric algorithm like AES, but both sides must have the same key, and that key exchange is difficult
- The solution is to use AES to encrypt the file, and use RSA to encrypt the AES key.
- Essentially, use the asymmetric RSA encryption to protect and exchange the AES key, and use AES to do the actual file encryption. You could even generate a new AES key each time you do this.

☞ We use AES/CTR Mode to encrypt the input (100MB file), then RSA is used to encrypt AES key and iv:

```
PS D:\Selfwork\Crypto> .\benchmark_100mb.exe
RSA/OAEP-MGF1(SHA-1) encrypt benchmarks...
  3.3 GHz cpu frequency
  1518.24 cycles per byte (cpb)
  2.07287 MiB per second (MiB)
AES/CTR plaintext encrypt benchmarks...
  3.3 GHz cpu frequency
  9.05676e-06 cycles per byte (cpb)
  3.47489e+08 MiB per second (MiB)
```

10. Bài luyện tập 2

☞ *Use the private key to encrypt sign text using RSA (input can be from a file).*

- The SaveSignatureToFile function writes the signature in binary format to a file:

```
void SaveSignatureToFile(SecByteBlock signature , string filename){
    std::ofstream file;
    file.open(filename,std::ios_base::binary);
    assert(file.is_open());
    string s = HexEncode(signature);
    int len = s.length();
    char* char_array = new char[len + 1];
    strcpy(char_array, s.c_str());
    for (int i = 0; i < len; i++)
    {
        file.write(reinterpret_cast<char *>(&char_array[i]),sizeof(char_array[i]));
    }
    delete[] char_array;
    file.close();
}
```

- The GetFileData function reads the plaintext message from a file:

```
string GetFileData(string filename)
{
    cin.ignore();
    string data;
    FileSource file(filename.data(), true, new StringSink(data));
    cout << "Reading plaintext from file " << filename << endl;
    return data;
}
```

- In the main function, the message is signed using the SignMessage method of the signer object, and the resulting signature is stored in the buffer.
- The program then resizes the buffer to the actual size of the signature and prints the signature in hexadecimal format.
- Finally, the program saves the signature to a file named ”sample.bin” in binary format.
- If any exceptions are thrown during the process, they are caught and the program prints an error message.

```
        // Message
        string message = GetFileData("DigitalSignature.txt");
        cout << "Message : " << message << endl;
        // Signer object
        RSASS<PSS, SHA256>::Signer signer(privateKey);

        // Create signature space
        size_t length = signer.MaxSignatureLength();
        SecByteBlock signature(length);

        // Sign message
        length = signer.SignMessage(rng, (const byte*) message.c_str(),
            message.length(), signature);

        // Resize now we know the true size of the signature
        signature.resize(length);
        cout << "Signature : ";
        PrintSecByteBlock(signature);
        SaveSignatureToFile(signature,"sample.bin");

    }

    catch( CryptoPP::Exception& e ) {
        std::cerr << "Error: " << e.what() << std::endl;
    }

    return 0;
}
```

- After encrypting sign text using RSA:

```
Reading plaintext from file DigitalSignature.txt
Message : 378d34fe57ab4b61276634ac2e5aa20adf206203e7a3ed9b58b89a46db8278ffb453c0bd1fc3d8b9d24f67f3c3a5bb08c4fce4ae982daf44c2b14bdfbc68b73b7c1b0cf4e6ce5b13f3a5996bf2
5728119e247c302e0a3cbbfe7984c899e4fbd1252dc27f175699d4b386ba52b0d80ec1f542396c7e23ebe236a9b76d87999598
Signature : 9D6D0585D3D420465D2D1A1AAE3B360F49DF5E495DEAD61A3078F1E14CEF66C1C147EC1BFBD35A8CC1CB57C63E81AA5BEE25CD298B0EA7A475DA2F0CBB02FBDBC1A9248DF37226992FA6F0C2
D66ADC66BC2556485C19ED71BC5D4BCC1B02BEB3CDB0803D8CC89CE7F6262D75EBB0E7BBF17DB1944542E84FBBF4BD955E1C3D198ABBFA56F113D90334020CFA2E4F5CC518CD1C91752AA3B72B591F663467
D68E85D6949E83540662E785072285604D04311C7A80CD4F40384120CB5F12730A6FFF294AC8C47B31FA910BCCC6C2373C192B46D94A550EAD7222CEA4D74EFA40523E6A8A40A1330975D3E9647B374BB0CA
B3C1F500B583A6C7F46DD3B87BE59174196B0361A0E38E5C45A7F5CDAE7D53BDED4AD32DAF906DF44F42B525B55900BE36C0967E75BA44712371E0F96C1921931EA3AEAB176E361DBFE33FA4D6E56D9C57C1
CA9A28E05B973866EC7A446727B0D60F63E58F32AFEEFF948266C7D8C608FC2AAB19986E6EE3D6EC08FF7842C2800C49FB2332FC41B5C7E3410EAB0A1A45
PS D:\NT219 Cryptography\Crypto>
```