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

BÁO CÁO BÀI TẬP

Môn học: NT219

Kỳ báo cáo: Buổi 04 (Session 04)

Tên chủ đề: Thuật toán mã hoá Elliptic Curve và hàm băm

Ngày báo cáo: 08/05/2023

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT219.N22.ATCL.1

STT	Họ và tên	MSSV	Email
1	Võ Sỹ Minh	21521146	21521146@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá	Người đóng góp
1	Bài tập 1	100%	
2	Bài tập 2	100%	
3	Bài tập 3	100%	
4	Bài tập 4	100%	
5	Bài tập 6 (Không có bt5)	90%	
6	Bài tập 7	90%	

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

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

1. **Kịch bản 01/Câu hỏi 01**

2. **Kịch bản 02**
   - Tài nguyên:
   - Mô tả/mục tiêu:
   - Các bước thực hiện/ Phương pháp thực hiện (Ảnh chụp màn hình, có giải thích)
3. **Kịch bản 03**
   - Tài nguyên:
   - Mô tả/mục tiêu:
   - Các bước thực hiện/ Phương pháp thực hiện (Ảnh chụp màn hình, có giải thích)

Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

Bài tập 1: Sử dụng code mẫu sample_ecdsa.cpp được cung cấp, chỉnh sửa và ký tập tin UIT.png đính kèm.

```
//get the message from the file
string getMessageFromFile(string filename)
{
    string plaintext;
    std::ifstream file(filename);
    if (file.is_open())
    {
        getline(file, plaintext);
        file.close();
    }
    else
    {
        std::wcout << L"Unable to open file!" << endl;
        exit(1);
    }
    return plaintext;
}
```

Tạo một hàm để lấy thông tin file UIT.png

Sử dụng các công cụ có sẵn của sample code.

```
result = GeneratePrivateKey(CryptoPP::ASN1::secp160r1(), privateKey);
assert(true == result);
if (!result) { return -1; }

result = GeneratePublicKey(privateKey, publicKey);
assert(true == result);
if (!result) { return -2; }

// Sign and Verify a message
string message = getMessageFromFile("UIT.png");
string signature;

result = SignMessage(privateKey, message, signature);
if (true == result)
    std::wcout << "Signed successfully\n";
else
    std::wcout << "Failed while signing!\n";

result = VerifyMessage(publicKey, message, signature);
if (true == result)
    std::wcout << "Verified successfully\n";
else
    std::wcout << "Failed verifying! Unauthentic\n";
```

```
Microsoft Visual Studio Debug Console
Signed successfully
Verified successfully

Modulus:
 1461501637330902918203684832716283019653785059327.
Coefficient A:
 1461501637330902918203684832716283019653785059324.
Coefficient B:
 1632357913061681105466049194032715795305483454l3.
Base Point:
 X: 425826231723888350446541592701409065913635568770.
 Y: 203520114162904107873991457957346892027982641970.
Subgroup Order:
 1461501637330902918203687197606826779884643492439.
Cofactor:
 1.

Private Exponent:
 481522441152682903546140697064939654837934621818.

Public Element:
 X: 1297385226444512117853281203214833205562296612931.
 Y: 570590527037307591033387406310837071271560173058.

D:\Studying\A Code Space\CryptoLab04\x64\Release\CryptoLab04.exe
```

➔ Hoạt động tốt

Mục tiêu tiếp theo: Thêm một số hàm nhằm lưu file chứa chữ ký

```
// convert byte string to hex wstring cryptopp::byte
void convertor(string byteString)
{
    string encodedCode = "";
    StringSource(byteString, true,
        new HexEncoder(
            new StringSink(encodedCode)));
    wstring wstr = stringToWString(encodedCode);
    wcout << wstr << endl;
}
```

```
void setUpSignature(string filePrivateKey, string fileMessage, string& signature)
{
    ECDSA<ECP, SHA1>::PrivateKey privateKey;

    loadPrivateKey(filePrivateKey, privateKey);

    string message = getMessageFromFile(fileMessage);
    double timeCounter = 0.0;

    signature = signMessage(message, privateKey);
    if (signature.empty())
    {
        wcout << L"Signature is empty!" << endl;
        exit(1);
    }

    wcout << L"Signature: ";
    convertor(signature);
}
```

```
// function to sign the file
void putSignatureToFile(string filename, const string& signature)
{
    ofstream file(filename);
    try
    {
        file << signature;
        file.close();
    }
    catch (const std::exception& e)
    {
        wcout << e.what() << '\n';
        exit(1);
    }
}
```

Flow trong hàm main:

```
string fileName = "UIT.png";
string fileSignatureName = "signature.txt";
string filePrivateKey = "." + slash + "eccPrivate.key";
string fileMessage = "." + slash + fileName;
string fileSignature = "." + slash + fileSignatureName;

wcout << "Signing " << stringToWString(fileName) << " file\n";
try
{
    setUpSignature(filePrivateKey, fileMessage, signature);
    putSignatureToFile(fileSignature, signature);
    wcout << L"Signature saved successfully in " << stringToWString(fileSignatureName) << endl;
}
catch (const CryptoPP::Exception& e)
{
    wcout << L"Error when signing message!" << endl;
    wcout << e.what() << endl;
    exit(1);
}

pause();

return 0;
```

```
D:\Studying\A_Code_Space\CryptoLab04\x64\Release\CryptoLab...
Signing UIT.png file
Signature: 6F586BA7E1A5E9D234EA4FC85EF60C001939E255C710A45301B1
FF1BC23586E822330ADD2C065A7F9DF586EA9C340857F96F6203D7C3250839C
670D96BE30D22
Signature saved successfully in signature.txt
Press any key to continue . . .
```

Bài tập 2: Thay đổi thuật toán mã hoá sha1 thành sha256 và mô tả điểm khác biệt trong chương trình.

Ở SHA1, thì đường cong Elliptic Curve được sử dụng ở đây là đường cong secp160r1. Tạo ra một khóa private 160 bits.

```
// Private and Public keys
ECDSA<ECP, SHA1>::PrivateKey privateKey;
ECDSA<ECP, SHA1>::PublicKey publicKey;

//////////////////////////////////////////////
// Generate Keys
result = GeneratePrivateKey(CryptoPP::ASN1::secp160r1(), privateKey);
assert(true == result);
if (!result) { return -1; }

result = GeneratePublicKey(privateKey, publicKey);
assert(true == result);
if (!result) { return -2; }
```

```
Private Exponent:
 48156614822962897124111355050229875761512556234 9.

Public Element:
 X: 10591992876328961274788331602253460176854792618 43.
 Y: 20918471987143489871427063657905347326634712042 7.
Press any key to continue . . .
```

Ở SHA256 thì sử dụng private key có độ dài 256 bits

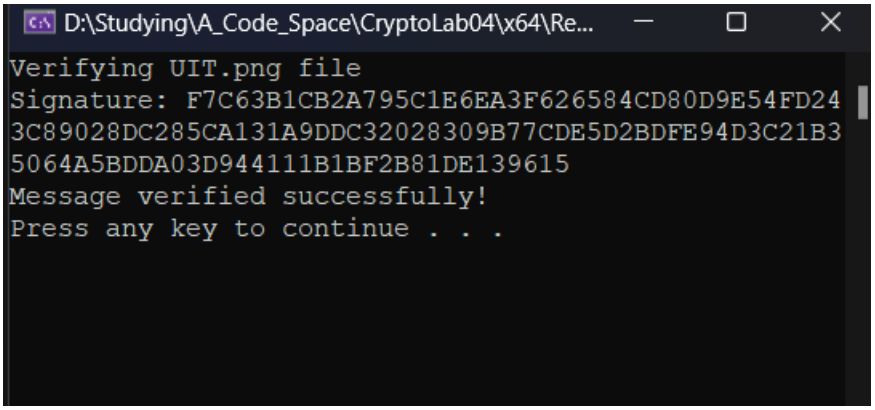
```
// Private and Public keys
ECDSA<ECP, SHA256>::PrivateKey privateKey;
ECDSA<ECP, SHA256>::PublicKey publicKey;

//////////////////////////////////////////
// Generate Keys
result = GeneratePrivateKey(CryptoPP::ASN1::secp256r1(), privateKey);
assert(true == result);
if (!result) { return -1; }

result = GeneratePublicKey(privateKey, publicKey);
assert(true == result);
if (!result) { return -2; }
```

```
Private Exponent:
 42333762487210818751860852954053927628443876004304119879777445460412628546513.

Public Element:
 X: 10136284136971273534359124581095846138775734547389286772534689143425166569715 7.
 Y: 29478719735785266447923145605166478469013522341797735586118232901165549460722.
Press any key to continue . . .
```

Bài tập 3: Load chữ ký số (r, s) và văn bản m từ file và xác thực.

Tương tự với việc ký, dựa vào flow trên để tạo nên hàm chứng thực chữ ký

```
//function to set up the verification
void setUpVerification(string filePublicKey, string fileMessage, string fileSignature)
{
    ECDSA<ECP, SHA1>::PublicKey publicKey;

    loadPublicKey(filePublicKey, publicKey);

    string message = getMessageFromFile(fileMessage);

    string signature;
    getSignatureFromFile(fileSignature, signature);

    if (verifyMessage(publicKey, message, signature) == false)
    {
        wcout << L"Verification failed!" << endl;
        exit(1);
    }

    //printDomainParameters(publicKey);
    //printPublicKey(publicKey);
    wcout << L"Signature: ";
    convertor(signature);
}
```



Bài tập 4: Viết chương trình C++ sử dụng switch case với các trường hợp hàm băm sau MD5, SHA224, SHA256, SHA384, SHA512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256

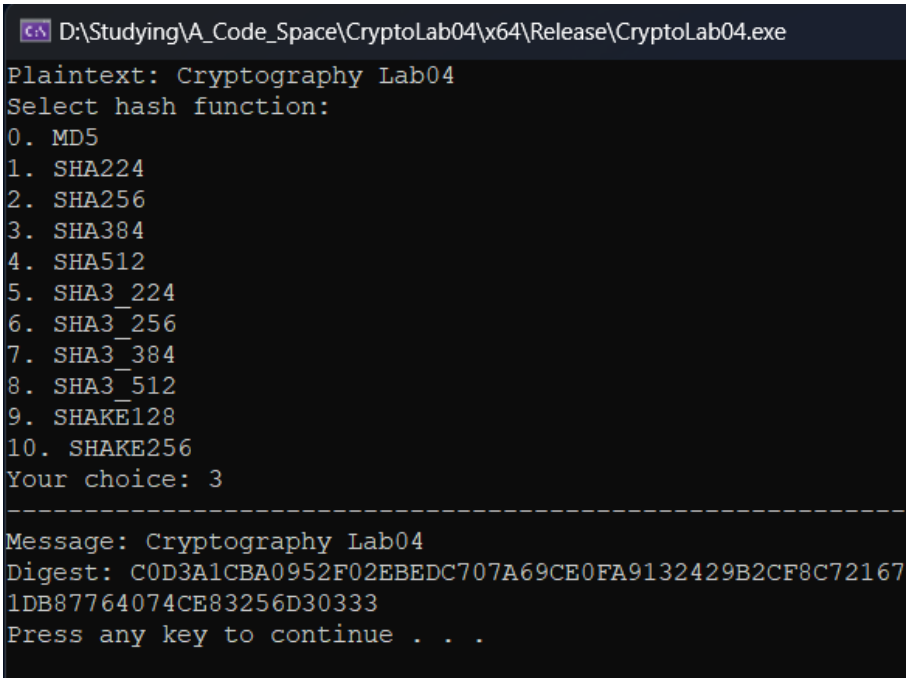
Các hàm dùng để tái sử dụng cho các loại SHA và SHAKE

```cpp
//create hash function
template <class HASH>
string hashFunc(const string& message)
{
    HASH hash;
    string digest;
    hash.Restart();
    hash.Update((const CryptoPP::byte*)message.data(), message.size());
    digest.resize(hash.DigestSize());
    hash.TruncatedFinal((CryptoPP::byte*)&digest[0], digest.size());
    return digest;
}

//create shake function
template <class SHAKE>
string shakeFunc(const string& message, int digestSize)
{
    SHAKE hash;
    string digest;
    hash.Restart();
    hash.Update((const CryptoPP::byte*)message.data(), message.size());
    digest.resize(digestSize);
    hash.TruncatedFinal((CryptoPP::byte*)&digest[0], digest.size());
    return digest;
}
```

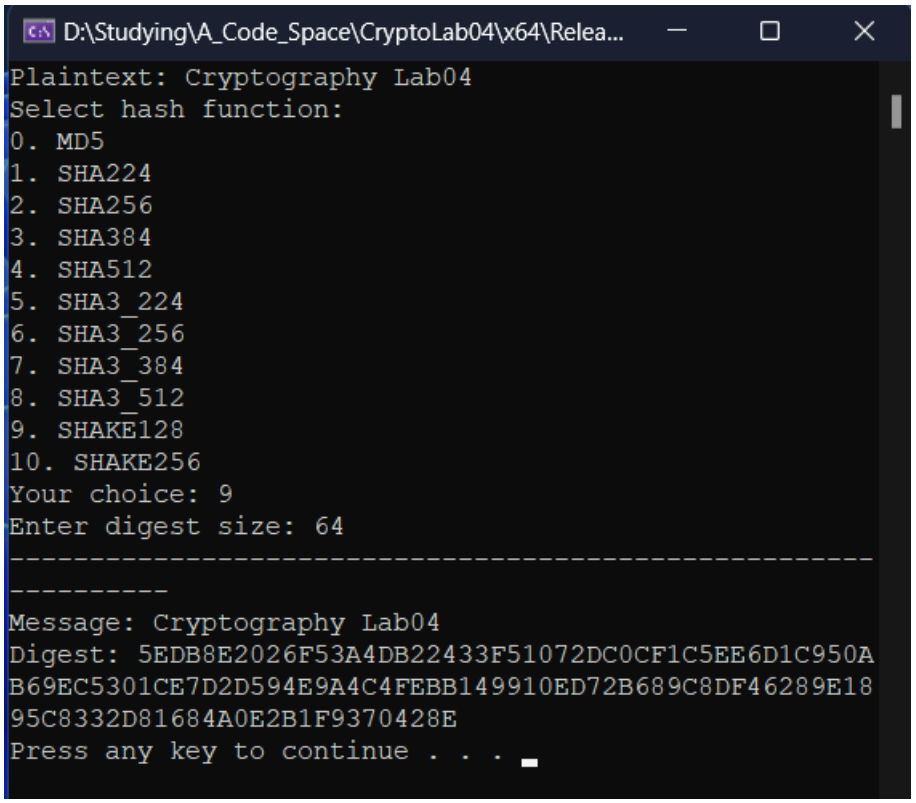
```
    //selectHashFunction
    int typeOfHashFunction = selectHashFunction();
    if (typeOfHashFunction == 9 || typeOfHashFunction == 10)
    {
        std::wcout << L"Enter digest size: ";
        wcin >> digestSize;
    }

    switch (typeOfHashFunction)
    {
    case 0:
        digest = hashFunc<CryptoPP::MD5>(message);
        break;
    case 1:
        digest = hashFunc<CryptoPP::SHA224>(message);
        break;

    case 2:
        digest = hashFunc<CryptoPP::SHA256>(message);
        break;

    case 3:
        digest = hashFunc<CryptoPP::SHA384>(message);
        break;
```

```
case 7:
    digest = hashFunc<CryptoPP::SHA3_384>(message);
    break;

case 8:
    digest = hashFunc<CryptoPP::SHA3_512>(message);
    break;

case 9:
    digest = shakeFunc<CryptoPP::SHAKE128>(message, digestSize)
    break;

case 10:
    digest = shakeFunc<CryptoPP::SHAKE256>(message, digestSize)
    break;

}
std::wcout << L"-----------------------------------------------
std::wcout << L"Message: " << s2ws(message) << endl;
std::wcout << L"Digest: ";
convertor(digest);
```

Kết quả: SHA384 làm ví dụ



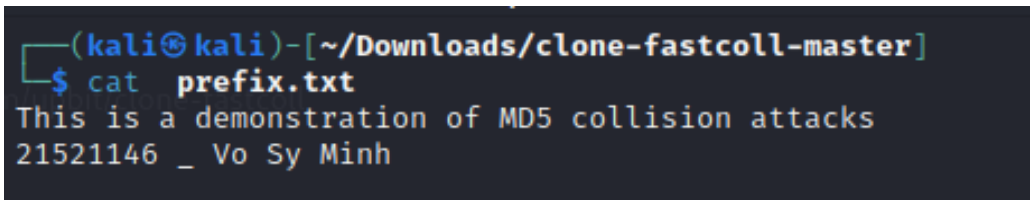
SHAKE128, digest size 64 bits làm ví dụ



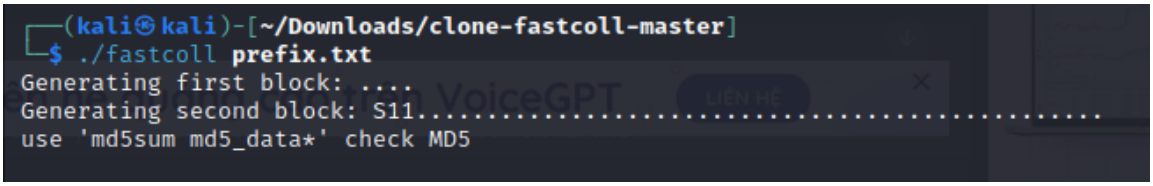
Bài tập 6: Tìm hiểu phương pháp tấn công collision trong hàm băm và mô tả chi tiết lại quá trình tấn công sử dụng hàm băm md5.

Mục đích của việc tấn công collision trên hàm băm MD5 là để tạo ra hai thông điệp khác nhau có cùng giá trị băm. Điều này có thể được sử dụng để tạo ra các chữ ký giả mạo hoặc các tài liệu giả mạo.

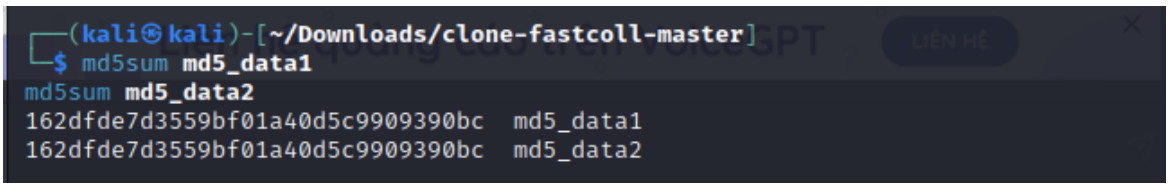
Công cụ dùng cho việc demo: Fastcoll, một công cụ nhanh chóng triển khai collision attack cho MD5 và SHA1.



Tạo một file prefix.txt để làm mẫu cho giá trị hàm băm mà collision attack hướng tới



Sau khi đọc thông tin file prefix, fastcoll sẽ tạo ra 2 file có chung giá trị hàm băm



Dùng md5sum xem xét và so sánh. Đã có 2 file chứa chứa giá trị hàm băm như nhau

Bài tập 7: Tìm hiểu tấn công length extension trong hàm băm và mô tả lại chi tiết quá trình sử dụng hàm băm sha256.

Mục tiêu của tấn công length extension là để kẻ tấn công sử dụng Hash (message1) và độ dài của message1 để tính toán Hash (message1 ‖ message2) cho message2 được điều khiển bởi kẻ tấn công mà không cần biết nội dung của message1. Điều này có thể được sử dụng để tạo ra một chữ ký giả mạo cho một tệp hoặc để xác minh tính toàn vẹn của một tệp.

Công cụ dùng cho việc demo: **Hashdump**

Cần những đầu vào sau:

- Chữ ký tin nhắn ban đầu, được tạo ra bằng một khóa bí mật và một hàm băm.
- Nội dung message ban đầu (plaintext).
- Chiều dài của khóa bí mật (theo bit).
- Dữ liệu mà bạn muốn thêm vào đằng sau message ban đầu (plaintext).

Ngữ cảnh demo:

- Chữ ký tin nhắn ban đầu: \`f0a47c878ddc33e1c1035b86a351189f01a3d06a8a73483325e91fa627a042e8\`
- Message ban đầu: \`Hello\`
- Chiều dài của khóa bí mật: \`256\`
- Dữ liệu để thêm vào: \` world!\`

```
┌──(kali㉿kali)-[~/Downloads/HashPump]
└─$ ./hashpump
Input Signature: f0a47c878ddc33e1c1035b86a351189f01a3d06a8a73483325e91fa627a042e8
Input Data: Hello
Input Key Length: 256
Input Data to Add: world!
```

Ta có kết quả trả về là một chữ ký (cho mesage mới) và message mới bao gồm message ban đầu, một byte null, các byte đệm và dữ liệu thêm vào. Đây là chữ ký được tạo ra mà không cần có quyền truy cập vào khóa bí mật (private key).

```
┌──(kali㉿kali)-[~/Downloads/HashPump]
└─$ ./hashpump
Input Signature: f0a47c878ddc33e1c1035b86a351189f01a3d06a8a73483325e91fa627a042e8
Input Data: Hello
Input Key Length: 256
Input Data to Add: world!
d94798b56cb0eaff38593f971fbce727378f98a5a631ad350f3c688af96dfaaf
Hello\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x08(world!
```

YÊU CẦU CHUNG

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
- Sinh viên báo cáo kết quả thực hiện và nộp bài.

Báo cáo:

- File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
- Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
- Đặt tên theo định dạng: [Mã lớp]-SessionX_GroupY. (trong đó X là Thứ tự buổi Thực hành, Y là số thứ tự Nhóm Thực hành/Tên Cá nhân đã đăng ký với GV).

  *Ví dụ: [*NT101.K11.ANTT*]-Session1_Group3.*
- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm.
- Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Đánh giá: Sinh viên hiểu và tự thực hiện. Khuyến khích:

- Chuẩn bị tốt.
- Có nội dung mở rộng, ứng dụng trong kịch bản/câu hỏi phức tạp hơn, có đóng góp xây dựng.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

HẾT