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

BÁO CÁO BÀI TẬP

Môn học: xxx

Kỳ báo cáo: Buổi 01 (Session 01)

Tên chủ đề: chủ đề môn học

Ngày báo cáo: xx/xx/20xx

Nhóm: XX (nếu không có xoá phần này)

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT101.XXXX.YYYY

STT	Họ và tên	MSSV	Email
1	Phạm Công Lập	21522281	21522281@gm.uit.edu.vn
2	Huỳnh Nguyễn Anh Khoa	21522221	21522221@gm.uit.edu.vn
3	Dương Phú Cường	21521900	21521900@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%	Lập, Khoa
2	Bài tập 2	100%	Khoa
3	Bài tập 3	100%	Lập, Khoa
4	Bài tập 4	100%	Cường
5	Bài tập 6	100%	Lập
6	Bài tập 7	100%	Lập

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

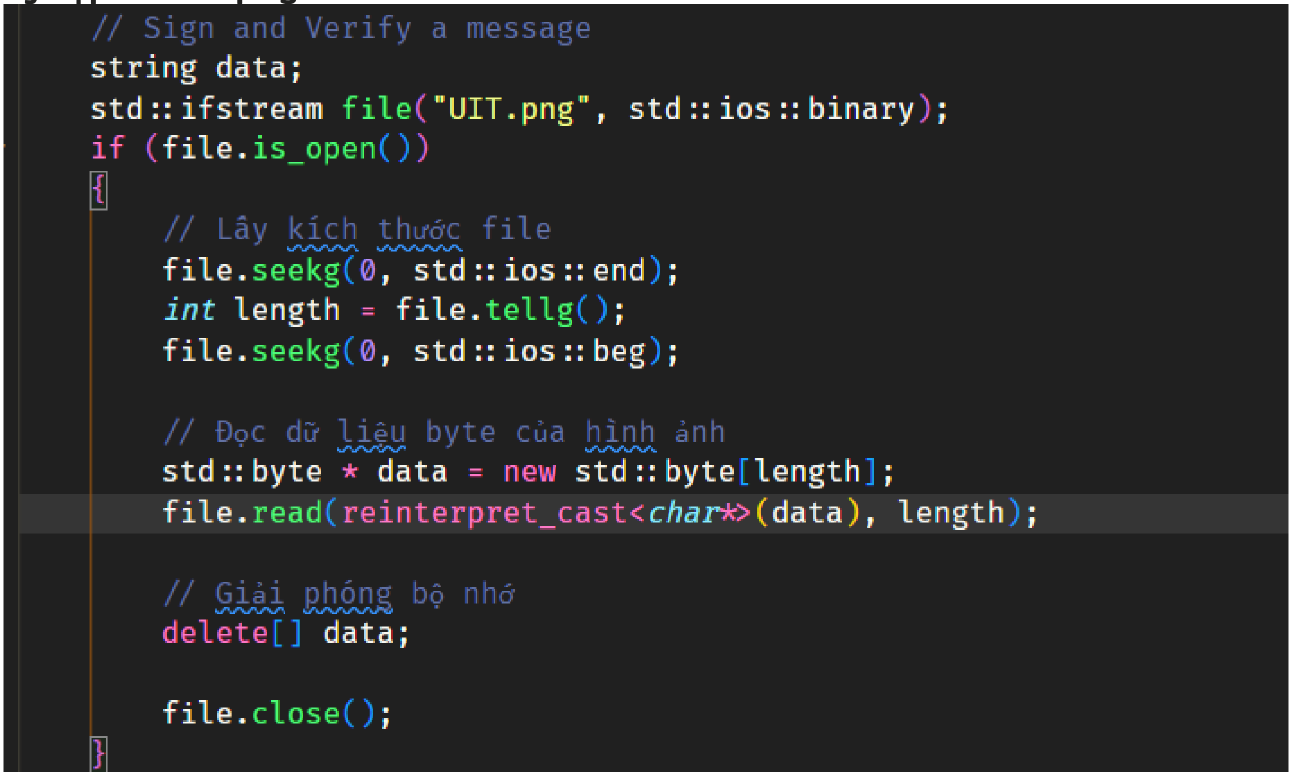
BÁO CÁO CHI TIẾT

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)

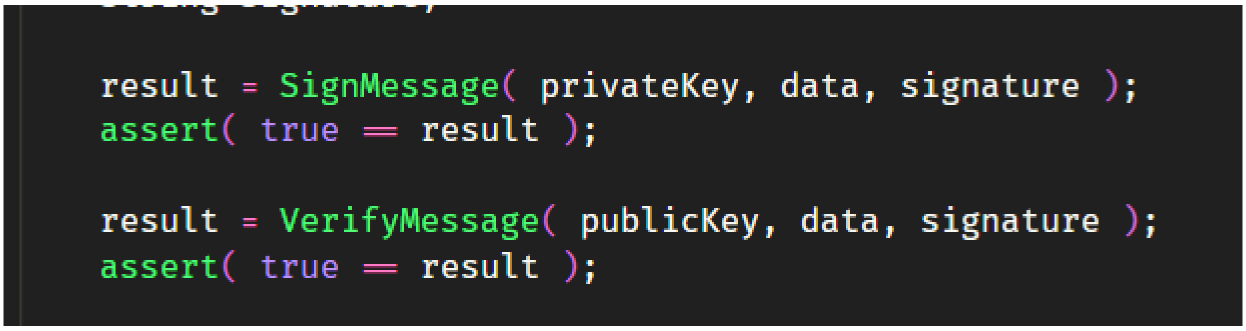
Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này

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



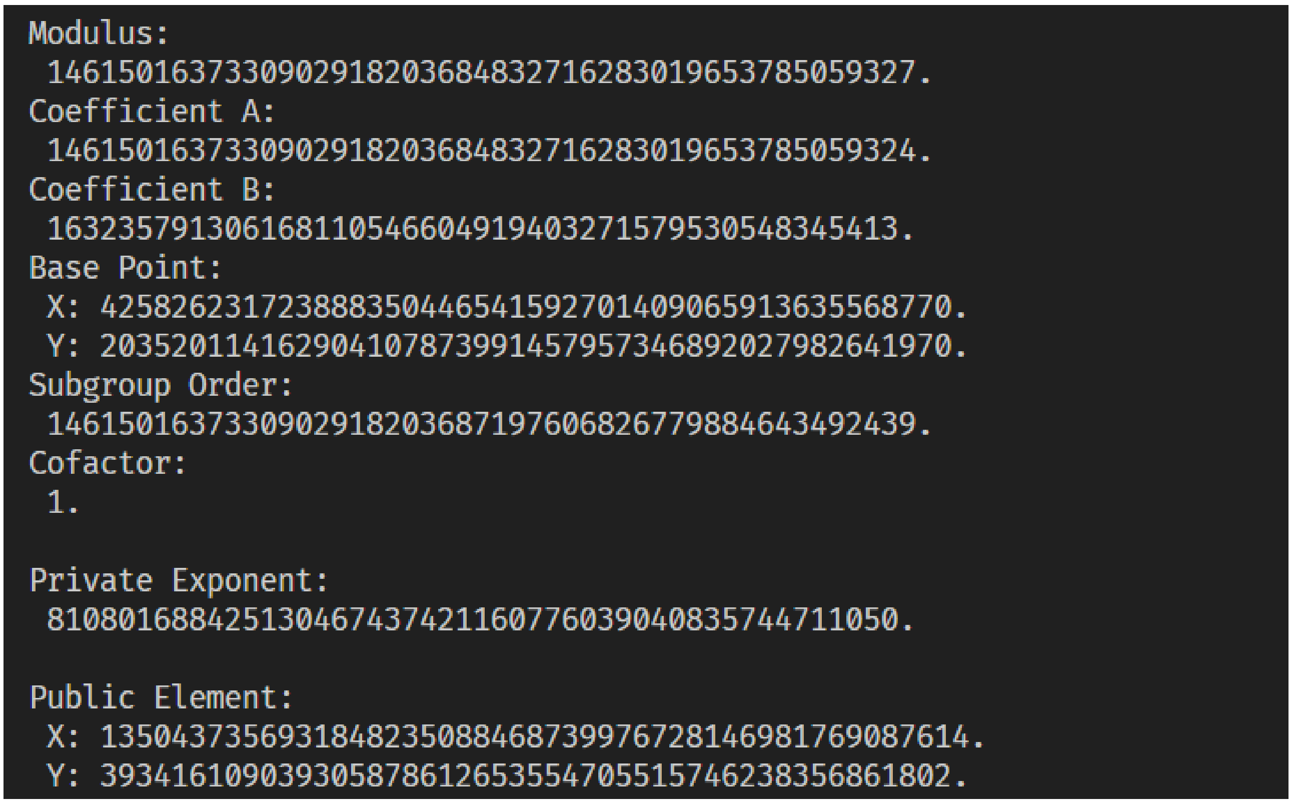
```
// Sign and Verify a message
string data;
std::ifstream file("UIT.png", std::ios::binary);
if (file.is_open())
{
    // Lấy kích thước file
    file.seekg(0, std::ios::end);
    int length = file.tellg();
    file.seekg(0, std::ios::beg);

    // Đọc dữ liệu byte của hình ảnh
    std::byte * data = new std::byte[length];
    file.read(reinterpret_cast<char*>(data), length);

    // Giải phóng bộ nhớ
    delete[] data;

    file.close();
}
```

Create a std::ifstream object named file, open the file “UIT.png” in binary mode to read the data.
Check if the file is opened successfully using the is_open() function.
Use the seekg() function to get the read pointer to the end of the file by changing the cursor position to the end of the file, then use the tellg() function to get the size of the file.
Use the seekg() function again to return the read pointer to the beginning of the file.
Creates a std::byte array of length equal to the size of the file to store the byte data of the image.
Use the read() function to read the byte data of the image and store it in the data array.
Frees the memory allocated for the data array via the delete[] operator.
Close the file by calling the file object’s close() method.
After this code is executed, the contents of the file ”UIT.png” are read and stored into the data variable as an array of bytes.



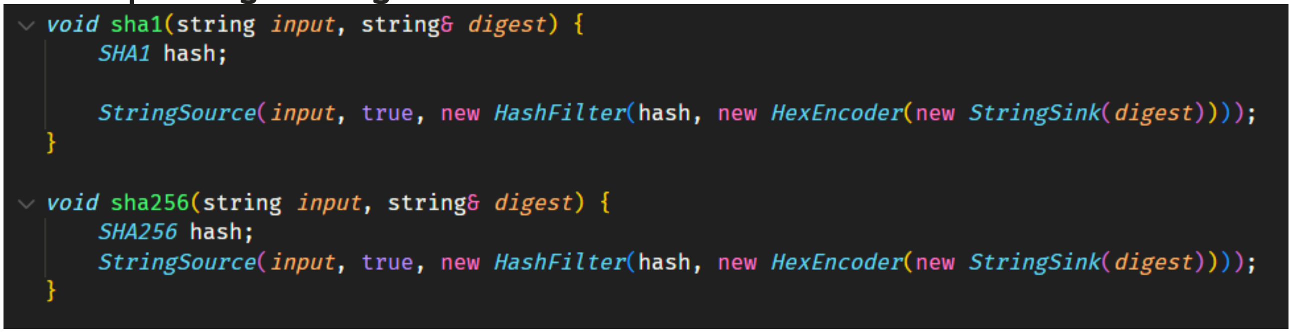
```
    result = SignMessage( privateKey, data, signature );
    assert( true == result );

    result = VerifyMessage( publicKey, data, signature );
    assert( true == result );
```

The above code is using the SignMessage() and VerifyMessage() functions to sign and verify the signature for a message.
The first line generates a message string containing a statement by Yoda.
Then the signature variable is declared to store the generated signature.
Next, the SignMessage() function is called with the private key as input, the data to be signed, and the signature variable to store the signature. The result of the function is stored in the variable result. If result is true then the signing was successful.
Then, the VerifyMessage() function is called with the input parameters being the public key (public key), the data to be authenticated (data), and the signature variable containing the signature. The result of the function is stored in the variable result. If result is true then the signature validation was successful.
In a nutshell, the above code is performing signing and signature validation for a message. This process uses a public and private key pair to ensure the integrity of the message and the origin of the message.



```
Modulus:
 1461501637330902918203684832716283019653785059327.
Coefficient A:
 1461501637330902918203684832716283019653785059324.
Coefficient B:
 163235791306168110546604919403271579530548345413.
Base Point:
 X: 425826231723888350446541592701409065913635568770.
 Y: 203520114162904107873991457957346892027982641970.
Subgroup Order:
 1461501637330902918203687197606826779884643492439.
Cofactor:
 1.

Private Exponent:
 810801688425130467437421160776039040835744711050.

Public Element:
 X: 1350437356931848235088468739976728146981769087614.
 Y: 393416109039305878612653554705515746238356861802.
```

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

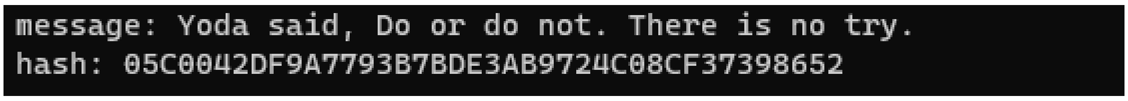


```
void sha1(string input, string& digest) {
    SHA1 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha256(string input, string& digest) {
    SHA256 hash;
    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}
```

The sha1 function is used to calculate the hash value of the input string, and stores the result into the string digest variable through the reference of the string& digest variable.
Similarly, the sha256 function is also used to calculate the hash value of the input string and store the result in the string digest variable.

<u>Point of difference:</u>

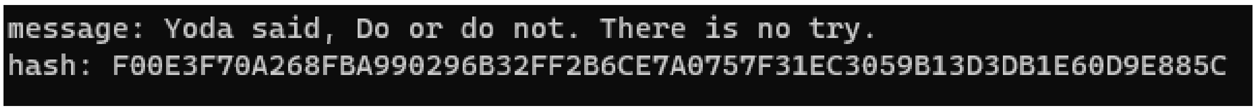
SHA-1:



```
message: Yoda said, Do or do not. There is no try.
hash: 05C0042DF9A7793B7BDE3AB9724C08CF37398652
```

- When using ECDSA with SHA-1, the 160-bit hash is used as input to the ECDSA algorithm to generate the digital signature. However, due to weaknesses in the security of SHA-1, it is no longer considered secure for use in cryptographic applications.

SHA-256:



```
message: Yoda said, Do or do not. There is no try.
hash: F00E3F70A268FBA990296B32FF2B6CE7A0757F31EC3059B13D3DB1E60D9E885C
```

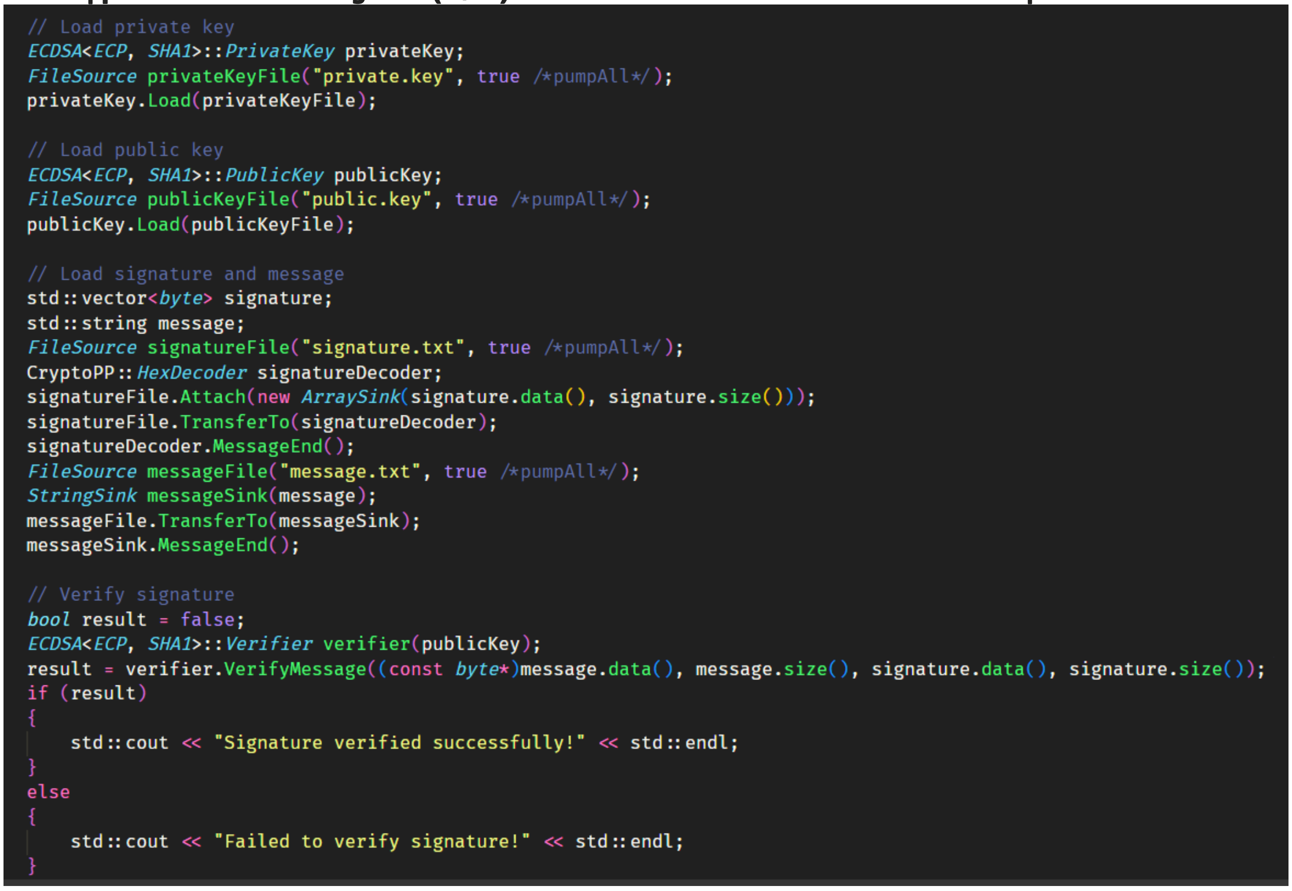
SHA-256 generates a 256-bit value

- When using ECDSA with SHA-256, the 256-bit hash is used as input to the ECDSA algorithm to generate the digital signature. SHA-256 is considered to be a stronger and more secure hashing algorithm than SHA-1, and is widely used in modern cryptographic applications.

<u>Overview:</u>

SHA-1 produces output of length 160 bits, while SHA-256 produces output of length 256 bits. SHA-1 has been evaluated as a serious security flaw and should no longer be used. Meanwhile, SHA-256 is considered secure until now and is commonly used in security applications.

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



```
// Load private key
ECDSA<ECP, SHA1>::PrivateKey privateKey;
FileSource privateKeyFile("private.key", true /*pumpAll*/);
privateKey.Load(privateKeyFile);

// Load public key
ECDSA<ECP, SHA1>::PublicKey publicKey;
FileSource publicKeyFile("public.key", true /*pumpAll*/);
publicKey.Load(publicKeyFile);

// Load signature and message
std::vector<byte> signature;
std::string message;
FileSource signatureFile("signature.txt", true /*pumpAll*/);
CryptoPP::HexDecoder signatureDecoder;
signatureFile.Attach(new ArraySink(signature.data(), signature.size()));
signatureFile.TransferTo(signatureDecoder);
signatureDecoder.MessageEnd();
FileSource messageFile("message.txt", true /*pumpAll*/);
StringSink messageSink(message);
messageFile.TransferTo(messageSink);
messageSink.MessageEnd();

// Verify signature
bool result = false;
ECDSA<ECP, SHA1>::Verifier verifier(publicKey);
result = verifier.VerifyMessage((const byte*)message.data(), message.size(), signature.data(), signature.size());
if (result)
{
    std::cout << "Signature verified successfully!" << std::endl;
}
else
{
    std::cout << "Failed to verify signature!" << std::endl;
}
```

First, the code redefines the byte data type as an unsigned char for use during ECDSA digital signature verification.

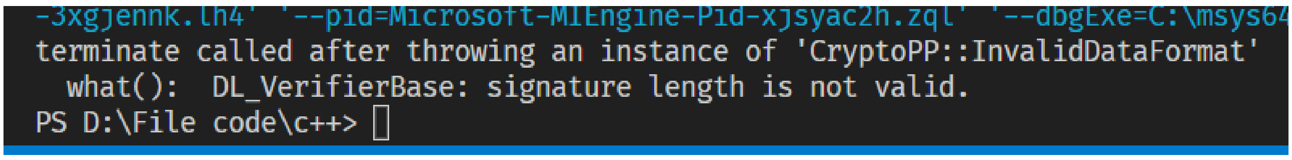
Next, the code uses the PrivateKey class to load the secret key from the private.key file. Similarly, the PublicKey class is used to load the public key from the file public.key.

The code then uses the HexDecoder and ArraySink classes to load the signature from the signature.txt file. This signature will be stored in the signature vector.

Similarly, the code uses StringSink to load the message to be verified from the message.txt file. This message will be stored in the message string.

Finally, the code uses the Verifier class to verify the ECDSA signature. The VerifyMessage method is used to verify the message and signature. The result of the verification is stored in the variable result. If the verification is successful, the program will display the message "Signature verified successfully!", otherwise it will display the message "Failed to verify signature!".

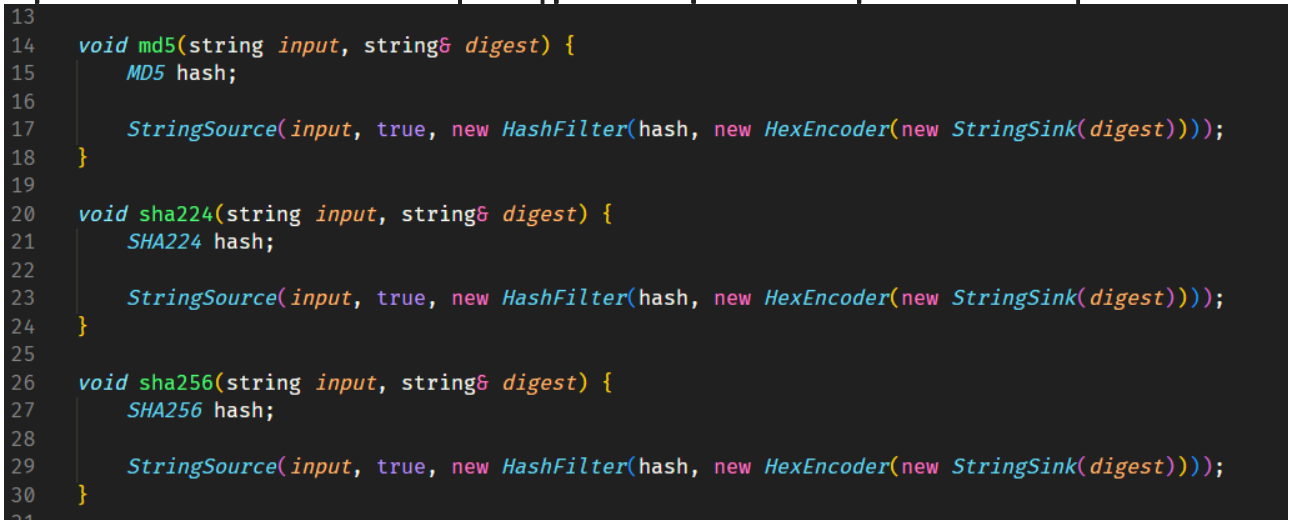
In a nutshell, this snippet loads the key, message, and signature, and then uses them to verify the ECDSA signature and display the verification results.



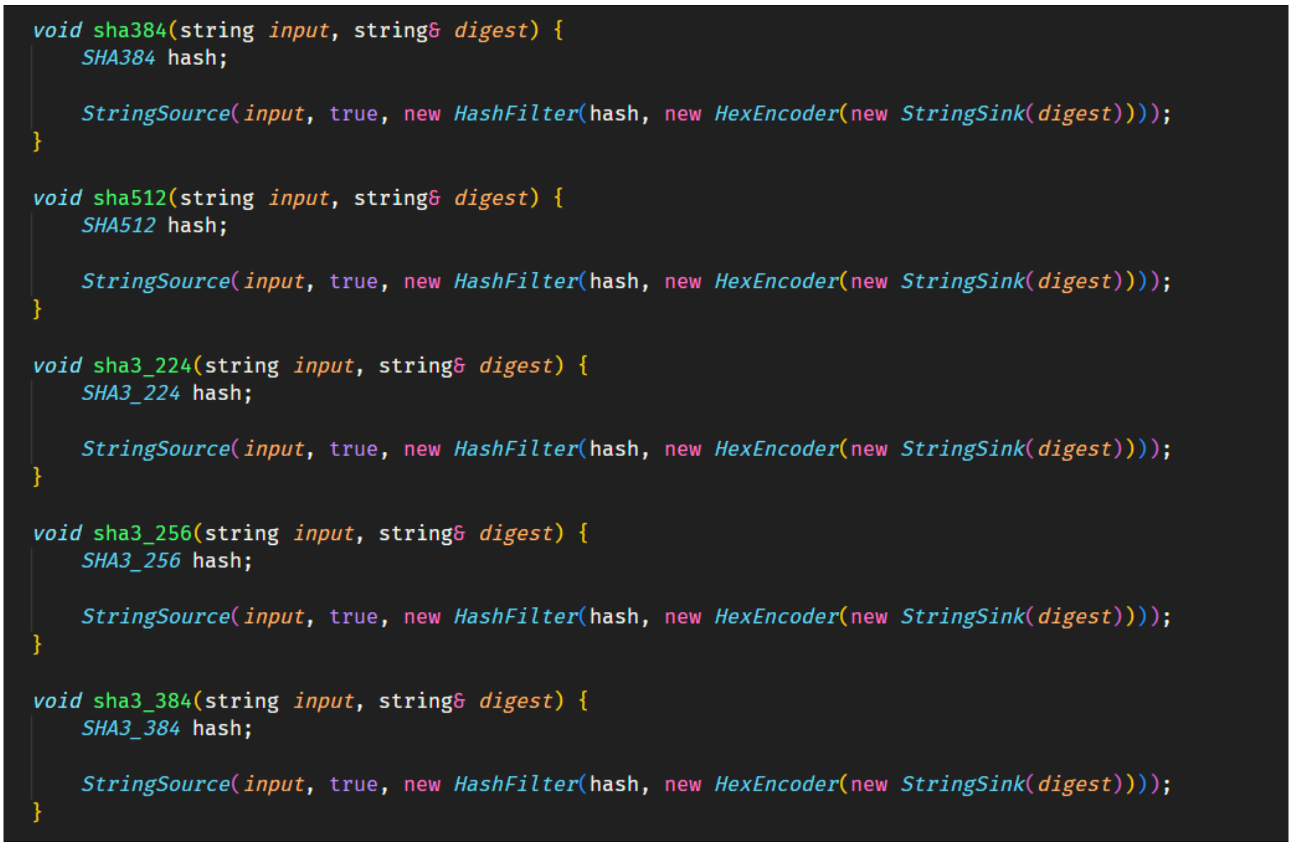
```
-3xgjennk.lh4' '--pid=Microsoft-MIEngine-Pid-xjsyac2h.zql' '--dbgExe=C:\msys64
terminate called after throwing an instance of 'CryptoPP::InvalidDataFormat'
  what():  DL_VerifierBase: signature length is not valid.
PS D:\File code\c++> 
```

Message "DL_VerifierBase: signature length is not valid" means that the length of the signature is not valid.
Because I created fake signature and did not fit the length, the above error appeared.

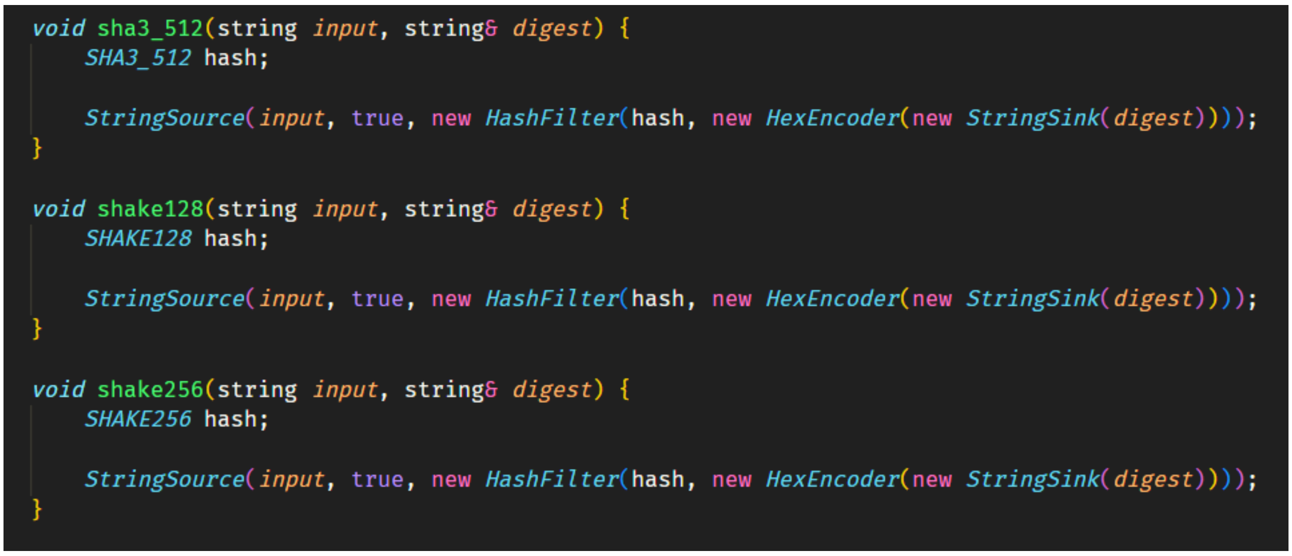
Bài tập 4: Viết chương trình C++ sử dụng switch case với các trường hợp hàm băm sau MD5, SHA224, SHA256, SHA384, SHA512, SHA3-224, SHA3-256, SHA3-384, SHA3-512, SHAKE128, SHAKE256.
Input đầu vào cần băm sẽ được nhập từ bàn phím, kết quả xuất ra đoạn mã hash.



```
void md5(string input, string& digest) {
    MD5 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha224(string input, string& digest) {
    SHA224 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha256(string input, string& digest) {
    SHA256 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}
```



```
void sha384(string input, string& digest) {
    SHA384 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha512(string input, string& digest) {
    SHA512 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha3_224(string input, string& digest) {
    SHA3_224 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha3_256(string input, string& digest) {
    SHA3_256 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void sha3_384(string input, string& digest) {
    SHA3_384 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}
```

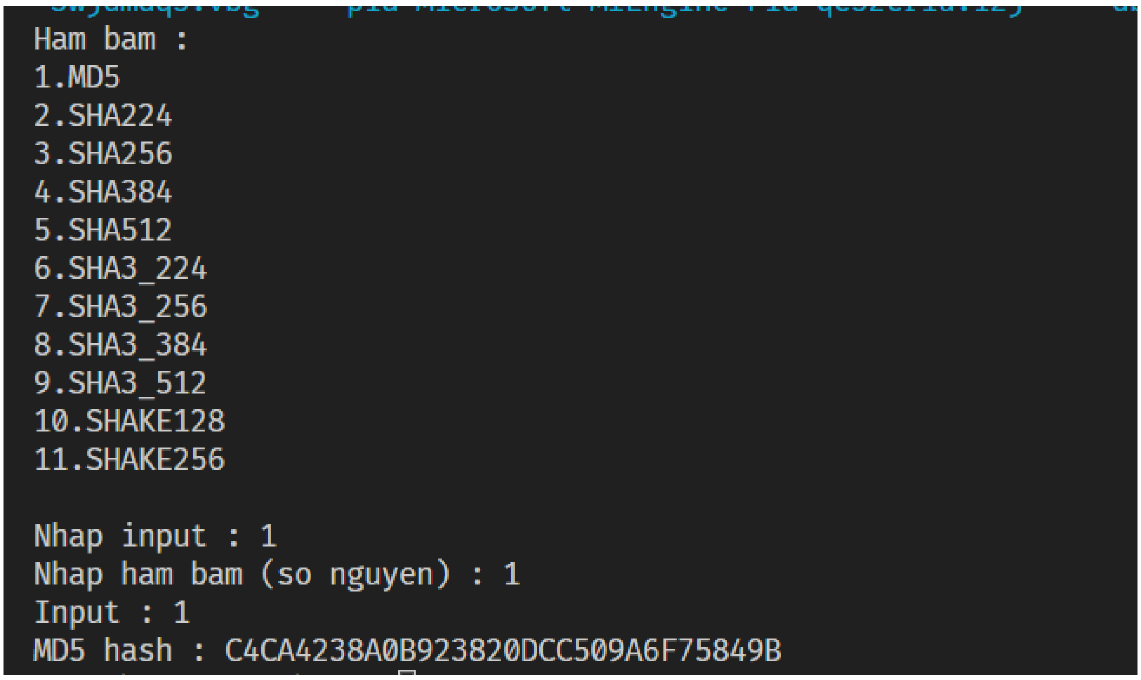


```
void sha3_512(string input, string& digest) {
    SHA3_512 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void shake128(string input, string& digest) {
    SHAKE128 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}

void shake256(string input, string& digest) {
    SHAKE256 hash;

    StringSource(input, true, new HashFilter(hash, new HexEncoder(new StringSink(digest))));
}
```

This is a set of functions to compute the hash (hash) of an input string using various hashing algorithms such as MD5, SHA-2, SHA-3 and SHAKE.

Each function will have two input parameters: the string to calculate the hash and the destination string to store the result of the calculation. In the function, the corresponding hash algorithm is selected and initialized with the hash object. Then, the StringSource will be used to give the input string for hashing, followed by the HashFilter to calculate the hash result. Finally, the resulting hash is encoded in hex to avoid non-visible characters and saved to the target string via StringSink.

In a nutshell, the above code defines functions to compute the hash of an input string using different hashing algorithms and encode the result of the calculation into a hex code.



```
Ham bam :
1.MD5
2.SHA224
3.SHA256
4.SHA384
5.SHA512
6.SHA3_224
7.SHA3_256
8.SHA3_384
9.SHA3_512
10.SHAKE128
11.SHAKE256

Nhap input : 1
Nhap ham bam (so nguyen) : 1
Input : 1
MD5 hash : C4CA4238A0B923820DCC509A6F75849B
```

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. Tham khảo https://en.wikipedia.org/wiki/Collision_attack

The MD5 hash collision attack is an attack method that secures the system by trying to generate two different data, but with the same MD5 hash value. Having found two such data, the hacker can use them to create a data spoofing attack, by replacing an original with a forged copy.

To perform a collision attack on an MD5 hash, a hacker must follow these steps:
Step 1: Select the source message (plaintext)

To perform a collision attack on the MD5 hash, the hacker needs to select a source message to generate the collision. This message can be any data, from text to images, audio files or any other type of data. The hacker will feed this message into the MD5 hash to calculate its hash value.

For example, a hacker wants to create a collision for the message "hello world".

Step 2: Look for a collision message

After obtaining the source message, the hacker needs to find another message, called the collision message, such that its hash is equal to the hash of the source message. This means that when both these messages are included in the MD5 hash, their hash value will be the same.

The collision message search can be done through techniques such as random search or dynamic programming search. The random search technique is simpler, but requires more computational time and resources. Dynamic programming search techniques are more complex, but allow faster searches.

For example, after the computation, the hacker finds another collision message whose hash is also d41d8cd98f00b204e9800998ecf8427e. This clash message could be a "holy cow".

Step 3: Use collision generation

One of the most popular collision message generation techniques is simple optimization. This technique uses changing some bits in the collision message to produce a new message with a hash value equal to the source message.

Specifically, the hacker will perform the following steps:

Split the two source and collision messages into 16-byte blocks.
Look for blocks in the collision message that are different from the source message.
Change the bits in those blocks to produce a new message with the same hash value as the source message.
For example, in our case, the "holy cow" collision message contains the following blocks:

holy cow => 686f6c7920636f77 => 686f6c7920636f770000000000000000

Meanwhile, the "hello world" source bulletin contains the following blocks:

hello world => 68656c6c6f20776f726c64 =>
68656c6c6f20776f726c640000000000

The hacker finds the block that differs between these two messages is "y c". To create a collision for the "hello world" message, the hacker changes the bits of this block.

Specifically, the hacker will change 3 bits of block " y c" to create a new block called " yz". This will not change the hash of the collision message, but when concatenated with other blocks in the collision message, will produce a new message with the same hash as the source message.

So the new collision message will take the form:

holy coz => 686f6c7920636f7a => 686f6c7920636f7a0000000000000000

After putting these blocks together, the hacker created a new message "hello coz" with the same hash value as "hello world".

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. Tham khảo https://en.wikipedia.org/wiki/Length_extension_attack

The length extension attack is a type of hash function attack, in which an attacker uses the known hash value of a particular input data to compute the hash of a new data that the attacker cannot obtain. unknown attack. This attack works for simple hash functions like MD5 and SHA-1, and has been fixed in later versions of these hash functions, such as SHA-256.

To understand the length extension attack, consider an example with a SHA-256 hash. Suppose an attacker knows the hash value of an input is "hello world", and wants to compute the hash of a new string by adding a continuation to the original string that the attacker does not know. To do this, the attacker needs to calculate padding for the new string, to ensure that the new string length is also a multiple of the block length (512 bits) and matches the hash function's requirements. SHA-256.

To calculate the padding for the new string, the attacker uses the hash of the original string and the length values to calculate the initial padding. The attacker then adds a continuation to the original string and computes the hash of the new string using the hash of the original string and the calculated padding.

For example:
Given the original string "hello world" and the corresponding hash value:
8b1a9953c4611296a827abf8c47804d7 28 4473
9d90e52b8fd6d36a9810224e592f73f4

Where the first part is the hash value of the string "hello world", and the next parts are the length value and padding value for the original string.

Let's say the attacker wants to calculate the hash for the new string "hello world, how are you today?". To do this, the attacker computes a padding for the new string using the initial hash of the "hello world" string and the length of the original string. The attacker then computes the hash for the new string using the original hash of the "hello world" string and the newly computed padding.

The calculation of the new padding and the new hash will be done as follows:

Calculate new padding value for new string:
The new string length is 29 characters (including spaces), so the new length value will be 29 * 8 = 232 bits.

With the initial length value of the string "hello world" being 11 characters, the initial padding length is 1 bit (because just add 1 bit 1 to the end of the original string to get a length that is a multiple of 512 bits) .
To compute the new padding, the attacker will input the following values into the SHA256 hash:
Empty string (no characters).
The initial padding value of the original string.
Some characters (doesn't matter, chosen by the attacker).
New length (232 bits).
The SHA256 hash function will calculate the hash for this string, and the new padding will be the last 128 bits of that hash.
Calculate the new hash for the new string:
The initial hash of the string "hello world" is
0x6cd3556deb0da54bca060b4c39479839c
5f5bc9602e9aa0ad56f8ad4ebb8465e.
To compute a new hash value for the new string, the attacker will input the following values into the SHA256 hash:
The new string is "hello world, how are you today?".
The new buffer value is calculated.
The SHA256 hash function will calculate the hash for this string, and that will be the new hash for the new string.
The result of this attack is that the attacker can compute the hash for the new string without knowing the initial value of the string or the hash of the original string. The length extension attack can be used to generate new strings

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