4

Lab

**BÁO CÁO BÀI THỰC HÀNH SỐ 4**

**Block ciphers: Hash Functions and Digital Certificates**

**Môn học: Security network**

**Lớp: CS4243.O11.CTTT**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn** | Nguyễn Xuân Hà |
| **Sinh viên thực hiện** | Phạm Thùy Dung (20521214) |
| Lương Gia Hân (19521468) |
| **Mức độ hoàn thành** | Hoàn thành |
| **Thời gian thực hiện** | 22/09/2019 – 29/09/2019 |
| **Tự chấm điểm** | 9/10 |

* 1. **CÁC BƯỚC THỰC HÀNH**

**Gợi ý:** *Ghi rõ từng bước thực hành, chụp hình ảnh screenshot để báo cáo thêm trực quan*

* 1. **TRẢ LỜI CÁC CÂU HỎI**

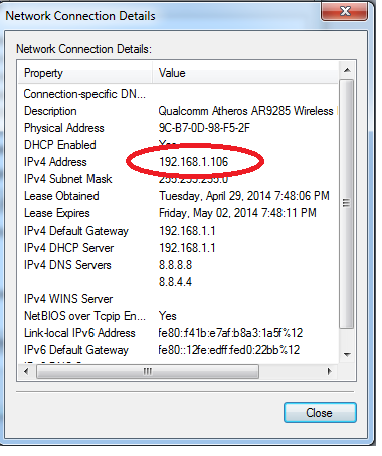
**Gợi ý:** *Trả lời câu hỏi đúng, đầy đủ, cần giải thích lý do tại sao có được đáp án, có các hình ảnh, bằng chứng để chứng minh tính đúng đắn.*

**Ví dụ:**

*Câu 1. Địa chỉ IP máy tính của bạn là gì?*

***Trả lời:*** *192.168.1.106*

Để xem địa chỉ IP của máy tính trên Windows, mở **Control Panel** và chọn **View network status and tasks.** Chọn mạng tương ứng đang sử dụng để kết nối Internet, chọn **Details** trong cửa sổ trạng thái. Xem địa chỉ IP trong Ipv4 Address



C.Answer

Exercise 1:

A screenshot of a computer

Description automatically generated

Here is my answer:

I use python the solve this problem:

A screen shot of a computer screen

Description automatically generated

Here is my explanation about the code:

A screen shot of a computer code

Description automatically generated

Purpose: To compute hashes of the provided data using specified hashing algorithms.

A black screen with white text

Description automatically generated

Purpose: To compute hashes of the contents of a specified file using different hashing algorithms.

With Main Script: it create to get input and give the result of the output

A screen shot of a computer program

Description automatically generated

Here is my file text created:

A screenshot of a computer

Description automatically generated

Here is my Result when run the code:

A screenshot of a computer program

Description automatically generated

Text Hashes:

MD5: 70f81d3c93b74af35201538a7068be34

SHA1: f73f57a04d0dce50b899cdd8252529c0327ef7de

SHA256: 81a2cd72bc1c7ff058ac25d56252371464d1849d6c3d471f403cc9c86e4229d9

Hex String Hashes:

MD5: 66eaadc618a18802aadaaabd84b22c58

SHA1: 1bac77b2c94d3cee1ec9632be651c3b68f7a78bc

SHA256: 17226b1f68aebacdef0746450f642874638b295707ef73fb2c6bb7f88e89929f

File Hashes:

MD5: d0331f2217ffb5e914213ab8ca5b7656

SHA1: 7298c6b0c7e6b98a954bb543b3f29fb7ba30f93b

SHA256: 5cbb9173718c1e787f002e50541702f312d73e9c85540066be7165fb5c2872dc

Here is illustration upload and download file from google drive

A screenshot of a computer

Description automatically generated

And next, is my code running to check the result again:

A screen shot of a computer screen

Description automatically generated

Here is result of the code with downloaded file:

A screenshot of a computer program

Description automatically generated

Comparing 2 results:

A black screen with white text

Description automatically generated

A computer screen with numbers

Description automatically generated

So, you can see the hash values of downloaded file is the same with original file:

For the file hashes, ideally, they remain the same before and after uploading/downloading, assuming the file content hasn't changed. This is because hash functions are designed to produce the same output for the same input and are sensitive to even minor changes in input.

Exercise 2:

A screenshot of a computer program

Description automatically generated

1. Consider two HEX messages as follows:

Message 1

d131dd02c5e6eec4693d9a0698aff95c2fcab58712467eab4004583eb8fb7f89

55ad340609f4b30283e488832571415a085125e8f7cdc99fd91dbdf280373c5bd8

823e3156348f5bae6dacd436c919c6dd53e2b487da03fd02396306d248cda0e99f

33420f577ee8ce54b67080a80d1ec69821bcb6a8839396f9652b6ff72a70

Message 2

d131dd02c5e6eec4693d9a0698aff95c2fcab50712467eab4004583eb8fb7f8

955ad340609f4b30283e4888325f1415a085125e8f7cdc99fd91dbd728037

3c5bd8823e3156348f5bae6dacd436c919c6dd53e23487da03fd02396306d

248cda0e99f33420f577ee8ce54b67080280d1ec69821bcb6a8839396f965a

b6ff72a70

How many bytes are the difference between two messages?

To do it we determine the byte difference between two given hexadecimal messages.

And then compare the two messages byte by byte

I use tool WinHex to compare two messages

First, I create 2 file text

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

And the open it in WinHex

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Then I click Tool > Compare data to compare 2 text

A screenshot of a computer

Description automatically generated

And then set up the output file with name report with search for differences engines and click OK

A screenshot of a computer

Description automatically generated

Here is result of the output

A screenshot of a computer

Description automatically generated

So, we have 6 different byte can be found.

Let’s generate MD5 hash values for each message. Please observe whether

these MD5 are similar or not and describe your observations in the lab report

Here is my code to do it

A screenshot of a computer program

Description automatically generated

When I run the code, I get the result:

A black background with white letters and numbers

Description automatically generated

With the result, we can see that these MD5 hashes are not similar; they are different. MD5 hashes are designed to change significantly even with small changes in the input data. Therefore, two slightly different messages will result in vastly different MD5 hashes.

In the context of cryptographic hash functions, MD5 is considered to be broken in terms of collision resistance, meaning that it's relatively easy to find two different inputs that produce the same MD5 hash (a collision). As a result, it's not recommended for security-critical applications, and more secure hash functions like SHA-256 are preferred.

2. Consider two executable programs named hello and erase:

• If you are using Windows, you can download these .exe files here.

• If you are using Linux, you can download the similar: hello and erase.

Run these programs and observe what happens. Note these programs must

be run from the console. Let’s generate MD5 hash values for these programs

and report your observations

I downloaded the file in <https://www.mscs.dal.ca/~selinger/md5collision/>

A screenshot of a computer

Description automatically generated

Here is my result:

A computer screen shot of white text

Description automatically generated

Report:

The first command calculates the MD5 hash value of the "hello.exe" file, and the resulting MD5 hash is cdc47d670159eef60916ca03a9d4a007.

The second command runs the "hello.exe" executable, which prints the message "Hello, world! (press enter to quit)" to the console.

A computer screen shot of white text

Description automatically generated

Report:

The first command calculates the MD5 hash value of the "erase.exe" file, and the resulting MD5 hash is cdc47d670159eef60916ca03a9d4a007. This MD5 hash can be used to verify the integrity of the "erase.exe" file.

The second command runs the "erase.exe" executable, which displays a humorous message indicating that it is an "evil" program that pretends to erase the hard drive but does not actually perform any data erasure.

In this case, both "hello.exe" and "erase.exe" have the same MD5 hash value (cdc47d670159eef60916ca03a9d4a007), which indicates that the content of these two files is the same. This is why they have the same MD5 hash.

The difference in behavior when executing these files is due to the instructions and logic contained within the programs themselves. The content of the files is the same, but the code inside each executable file determines how they behave when run. In the case of "hello.exe," it simply prints "Hello, world!" to the console, while "erase.exe" displays a humorous message about erasing the hard drive but does nothing harmful. The difference in behavior is determined by the code within the executables, not their MD5 hash values.

3. Download two PDF files: shattered-1.pdf and shattered-2.pdf. Open these files

to check the difference. Then generate SHA-1 hash for them, and observe the

result.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Here is my code to check the SHA-1 file:

A computer screen shot of a computer code

Description automatically generated

Here is my result:



You can see the output with the same characteristics, so it demonstrates:

This result is a demonstration of a collision attack against the SHA-1 cryptographic hash function, where two different inputs produce the same output hash. This is a significant vulnerability because hash functions like SHA-1 are supposed to generate a unique hash for each unique input; collisions undermine this principle and can compromise the security of systems relying on the hash function for integrity verification or other security guarantees.

A close-up of text

Description automatically generated

Can one make 2 different files with arbitrary contents and the same hash?

Creating two different files with the same hash, especially with arbitrary contents, is a significant challenge. This is known as a hash collision. While it is theoretically possible, especially with weaker hash functions like MD5 and SHA-1, it is computationally intensive and requires a deep understanding of the hash function's structure. The process typically involves crafting a specific section of the file that is designed to produce the same hash output despite changes in other parts of the file. The SHAttered attack demonstrated that this is possible with SHA-1.

For files in different formats to have the same hash and remain valid, one would need to exploit specific vulnerabilities in the file format's structure to insert the collision-producing data in a way that doesn't invalidate the file. This is much more complex and is not typically feasible with arbitrary content since file formats have specific requirements and structures that must be adhered to.

Can a hacker abuse Hash collision?

Yes, hash collisions can be abused by hackers in several ways. Here’s an example:

Example of Hash Collision Abuse: Digital Certificates

Consider digital certificates that use a hash function to ensure the integrity of the certificate contents. If a hacker can create two different certificates with the same hash value (a collision), they could get a legitimate certificate authority (CA) to sign a valid certificate and then replace it with a fraudulent one that has the same hash. The fraudulent certificate would appear valid because it has a valid signature from the CA. This could enable a wide range of security breaches, such as man-in-the-middle attacks, where the hacker could impersonate websites or services.

In real-world scenarios, a hash collision attack requires significant computational resources and expertise. Security algorithms are constantly being updated to address vulnerabilities, and the use of stronger hash functions like SHA-256 and SHA-3 is becoming more widespread precisely because they are currently resistant to known collision attacks.

A close-up of a certificate

Description automatically generated

A blue background with black text

Description automatically generated

Answer:

Step 1: Download a certificate from a real webserver

A screen shot of a computer

Description automatically generated

Next I copy 2 certificates into 2 file

c0.pem:

A screenshot of a computer screen

Description automatically generated

c1.pem

A screen shot of a computer screen

Description automatically generated

Step 2: Extract the public key (e, n) from the issuer’s certificate

A screen shot of a computer

Description automatically generated

Step 3: Extract the signature from the server’s certificate

A computer screen shot of a black screen

Description automatically generated

Here is my signature file based the result of previous comment:

A screenshot of a computer screen

Description automatically generated

Then I run the code to get this file

A screen shot of a computer

Description automatically generated

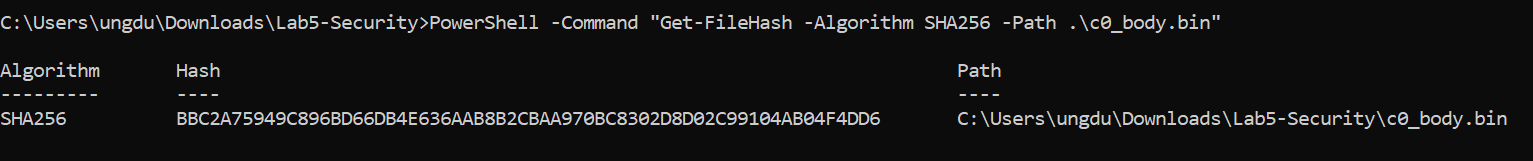
Step 4: Extract the body of the server’s certificate

A screenshot of a computer screen

Description automatically generated



So, I get the output of algorithm SHA256 with has value and the path of hash values



Here is my link on GitHub: <https://github.com/amenosakura/Lab5-Security>