UNIVERSITY OF INFORMATION TECHNOLOGY – VNU-HCM
FACULTY OF COMPUTER NETWORKS AND COMMUNICATIONS

4. Hash Functions and Digital Certificates

A. OVERVIEW

1. Introduction and learning objective.

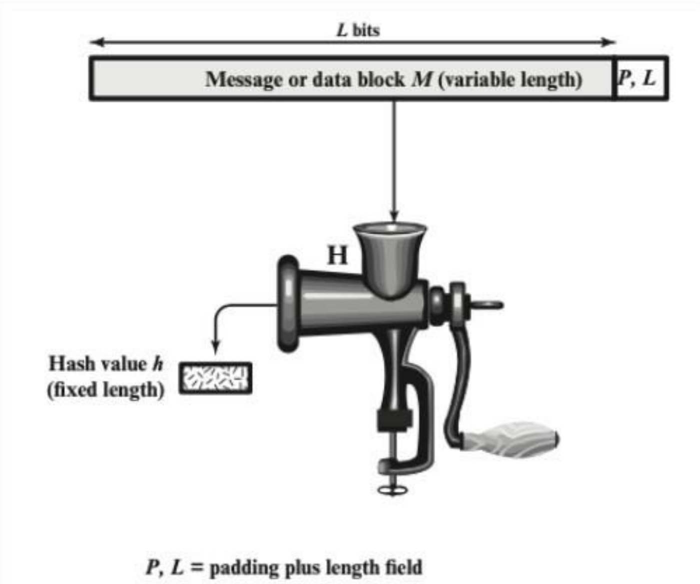

Figure 1 Hash Function

For cryptographic applications, need one or more of these properties:

- **The one-way property**: Given h, it's infeasible to find x such that H(x)=h. (Also called the "Preimage resistance")
- **The collision-free property**:
    - **Weak collision resistance**: Given x, it's infeasible to find y = x such that H(x) = H(y). (Also called "Second preimage resistance")
    - **Strong collision resistance**: It's infeasible to find any two x and y such that x /= y and H(x) = H(y). (Also called "Collision resistance")

> A secure one-way hash function needs to satisfy two properties: the one-way property and the collision-resistance property. The one-way property ensures that given a hash value h, it is computationally infeasible to find an input M, such that hash(M) = h. The collision-resistance property ensures that it is computationally infeasible to find two different inputs M1 and M2, such that hash(M1) = hash(M2).
>
> Several widely used one-way hash functions have trouble maintaining the collision-resistance property. At the rump session of CRYPTO 2004, Xiaoyun Wang and co-authors demonstrated a collision attack against MD5 [3]. In February 2017, CWI Amsterdam and Google Research announced the SHAttered attack, which breaks the collision-resistance property of SHA-1 [4]. While many students do not have trouble understanding the importance of the one-way property, they cannot easily grasp why the collision-resistance property is necessary, and what impact these attacks can cause. (SEED Labs - Wenliang Du, Syracuse University)

The **learning objective of this lab** is for students to get familiar with one-way hash functions and Message Authentication Code (MAC). After finishing the lab, in addition to gaining a deeper understanding of the concepts, students should be able to use tools and write programs to generate hash value and MAC for a given message. Besides, another goal is for students to really understand the impact of collision attacks, and see first hand what damages can be caused if a widely-used one-way hash function's collision-resistance property is broken (MD5 and SHA-1 collision).

2. Backgrounds and Prerequisites

To complete this lab well, you are expected to know how cryptographic hash functions work. If you are not familiar with them, you should find out in more detail in:

Chapter 11: Cryptographic Hash Functions W. Stallings, CS book - Cryptography and network security: Principles and practice, 7th ed. Boston, MA, United States: Prentice Hall, 2017

3. Lab environment and Tools

Operating system:

- 1 PC running Window

Programming languages and IDE: Flexible, you are free to choose any programming language you wish (Python and Golang are highly recommended)

B. LAB TASKS

1. Generating message digests (hash values) and HMAC

Task 1.1

Your task is to write an application to calculate hash values (at least 3 different types: MD5, SHA-1, SHA-2) for an input, which could be:

- Text string
- Hex string
- File

You are able to use the hash library for your own programming language. Then, test your application with the following exercise:

- Generate the hash values of "UIT Cryptography" in Text string and Hex string format. Then, compare the results with other tools to verify.
- Create a text file and put your name and student's ID inside. For example, "Nguyen Van An - 19521234". Generate hash values H1 of these files (using both MD5 and SHA-1). Subsequently, send this file to your friend via email or upload it to Google Drive and download it. Calculate the hash values of the downloaded file and compare them to those of the original file. Please observe whether these hash values are similar or not.

Tips: *You can refer to a similar application like* [SlavaSoft HashCalc - Hash, CRC, and HMAC Calculator]

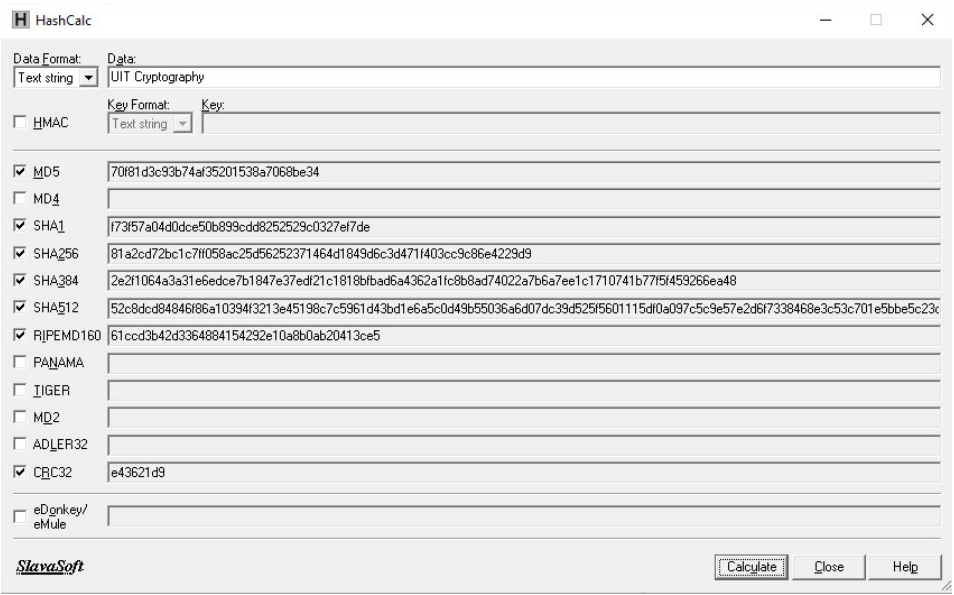

Figure 2 HashCalc application on Windows OS

2. Hash properties: One-way vs Collision-free

Task 2.1

It is now well-known that the cryptographic hash function MD5 and SHA-1 has been clearly broken (in terms of collision-resistance property). We will find out about MD5 and SHA-1 collision in this task by doing the following exercises:

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?

   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.

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

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.

Draw the conclusion based on your observations, explain the reasons for the existence of collision in MD5 and SHA-1.

Advanced Task 2.2

In this task, we will generate two different files with the same MD5 hash values. The beginning parts of these two files need to be the same, i.e., they share the same prefix. We can achieve this using the md5collgen program, which allows us to provide a prefix file with any arbitrary content. The way the program works is illustrated in Figure 3. The following command generates two output files, out1.bin and out2.bin, for a given prefix the prefix.txt:

```
$ md5collgen -p prefix.txt -o out1.bin out2.bin
```

We can check whether the output files are distinct or not using the diff command. We can also use the md5sum command to check the MD5 hash of each output file. See the following command.

```
$ diff out1.bin out2.bin

$ md5sum out1.bin

$ md5sum out2.bin
```

Since out1.bin and out2.bin are binary, we cannot view them using a text-viewer program, such as cat or more; we need to use a binary editor to view (and edit) them. You can use the hex editor called bless. (Figure 4)

1. If the length of your prefix file is not multiple of 64, what is going to happen?
2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

3. ***Can one make 2 different files get the same hash by appending stuff? Explain.***

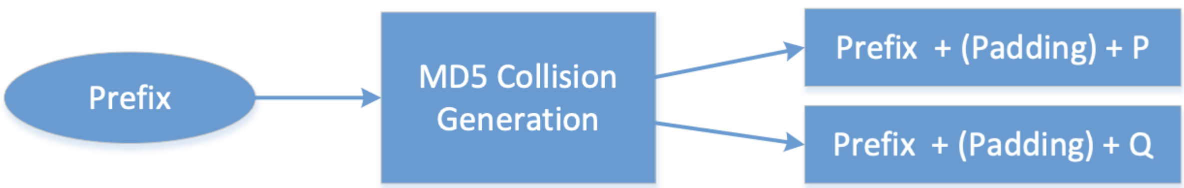

Figure 3 MD5 collision generation from a prefix

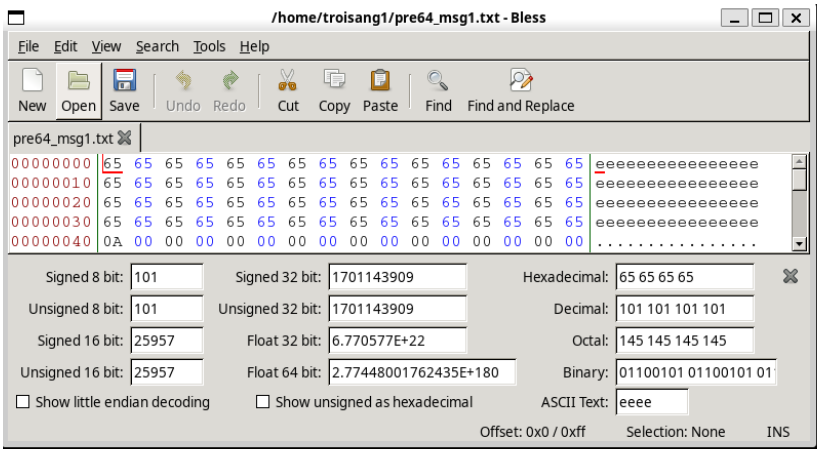

Figure 4 The UI from "bless" command.

Advanced Task 2.3

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

Take an example with files in different formats and explain if the file remains valid.

Is there any way that a hacker can abuse Hash collision? Give me an example.

Tips: You can use this tool for a demo. This tool also has a video to explain in YouTube. You can search it if you want to know more about Hash Collision.

3. Manually Verifying an X.509 Certificate

In this task, we will manually verify an X.509 certificate. An X.509 contains data about a public key and an issuer's signature on the data. We will download a real X.509 certificate from a web server, and get its issuer's public key, and then use this public key to verify the signature on the certificate.

Step 1: Download a certificate from a real webserver

We use the www.example.org server in this document. Students should choose a different web server that has a different certificate than the one used in this document (it should be noted that www.example.com maybe share the same certificate with www.example.org). We can download certificates using browsers or use the following command:

```
$ openssl s_client -connect www.example.org:443 -showcerts

Certificate chain
 0 s:/C=US/ST=California/L=Los
Angeles/O=Internet\xC2\xA0Corporation\xC2\xA0for\xC2\xA0Assigned\xC2\xA0
Names\xC2\xA0and\xC2\xA0Numbers/CN=www.example.org
  i:/C=US/O=DigiCert Inc/CN=DigiCert TLS RSA SHA256 2020 CA1
```

```
-----BEGIN CERTIFICATE-----
MIIHRzCCBi+gAwIBAgIQD6pjEJMHvD1BSJJkDM1NmjANBgkqhkiG9w0BAQsFADBP
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSkwJwYDVQQ
DEyBE
.....
0XELBnGQ666tr7pfx9trHniitNEGI6dj87VD+laMUBd7HBtOEGsiDoRSlA==
-----END CERTIFICATE-----
 1 s:/C=US/O=DigiCert Inc/CN=DigiCert TLS RSA SHA256 2020 CA1
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Global Root CA
-----BEGIN CERTIFICATE-----
MIIEvjCCA6agAwIBAgIQBtjZBNVYQ0b2ii+nVCJ+xDANBgkqhkiG9w0BAQsFADBh
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQ
QLExB3
.....
A7sKPPcw7+uvTPyLNhBzPvOk
-----END CERTIFICATE----
```

The result of the command contains two certificates. The subject field (the entry starting with s:) of the certificate is www.example.org, i.e., this is www.example.org's certificate. The issuer field (the entry starting with i:) provides the issuer's information. The subject field of the second certificate is the same as the issuer field of the first certificate. Basically, the second certificate belongs to an intermediate CA. In this task, we will use CA's certificate to verify a server certificate.

Copy and paste each of the certificates (the text between the line containing "Begin CERTIFICATE" and the line containing "END CERTIFICATE", including these two lines) to a file. Let us call the first one c0.pem and the second one c1.pem.

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

Openssl provides commands to extract certain attributes from the x509 certificates. We can extract the value of n using -modulus. There is no specific command to extract e, but we can print out all the fields and can easily find the value of e.

```
For modulus (n):
$ openssl x509 -in c1.pem -noout -modulus

Print ou all the fields, find the exponent (e):
$ openssl x509 -in c1.pem -text
```

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

There is no specific openssl command to extract the signature field. However, we can print out all the fields and then copy and paste the signature block into a file (note: if the signature algorithm used in the certificate is not based on RSA, you can find another certificate)

```
$ openssl x509 -in 1.pem -text
...
```

```
Signature Algorithm: sha256WithRSAEncryption
        aa:9f:be:5d:91:1b:ad:e4:4e:4e:cc:8f:07:64:44:35:b4:ad:
        3b:13:3f:c1:29:d8:b4:ab:f3:42:51:49:46:3b:d6:cf:1e:41:
        ......
        0e:84:52:94
```

We need to remove the spaces and colons from the data, so we can get a hex-string that we can deed into our program. The following commands can achieve this goal. The tr command is a Linux utility tool for string operations. In this case, the -d option is used to delete ":" and "space" from the data.

```
$ cat signature | tr -d '[:space:]:'

84a89a11a7d8bd0b267e52247bb2559dea30895108876fa9ed10ea5b3e0bc7
......
5c045564ce9db365fdf68f5e99392115e271aa6a8882
```

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

A Certificate Authority (CA) generates the signature for a server certificate by first computing the hash of the certificate, and then sign the hash. To verify the signature, we also need to generate the hash from a certificate. Since the hash is generated before the signature is computed, we need to exclude the signature block of a certificate when computing the hash. Finding out what part of the certificate is used to generate the hash is quite challenging without a good understanding of the format of the certificate.

X.509 certificates are encoded using the ASN.1 (Abstract Syntax Notation.One) standard, so if we can parse the ASN.1 structure, we can easily extract any field from a certificate. Openssl has a command called asn1parse used to extract data from ASN.1 formatted data, and is able to parse our X.509 certificate.

`

```
$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1522 cons: SEQUENCE
    4:d=1  hl=4 l=1242 cons:  SEQUENCE          ❶
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER
   :0E64C5FBC236ADE14B172AEB41C78CB0
   ... ...
 1236:d=4  hl=2 l=  12 cons:     SEQUENCE
 1238:d=5  hl=2 l=   3 prim:      OBJECT            :X509v3 Basic Constraints
 1243:d=5  hl=2 l=   1 prim:      BOOLEAN           :255
 1246:d=5  hl=2 l=   2 prim:      OCTET STRING      [HEX DUMP]:3000
 1250:d=1  hl=2 l=  13 cons:  SEQUENCE          ❷
 1252:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1263:d=2  hl=2 l=   0 prim:   NULL
 1265:d=1  hl=4 l= 257 prim:  BIT STRING
```

The field starting from ❶ is the body of the certificate that is used to generate the hash; the field starting from ❷ is the signature block. Their offsets are the numbers at the beginning of the lines. In our case, the certificate body is from offset 4 to 1249, while the signature block is from 1250 to the end of the file. For X.509 certificates, the starting offset is always the same (i.e., 4), but the end depends on the content length of a certificate. We can use the -strparse option to get the field from the offset 4, which will give us the body of the certificate, excluding the signature block.

```
$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
```

Once we get the body of the certificate, we can calculate its hash using the following command:

```
$ sha256sum c0_body.bin
```

Task 3.1

Now we have all the information, including the CA's public key, the CA's signature, and the body of the server's certificate. Use your own way to verify the certificate.

Tips: You can practice using this tutorial.

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 two or three to get the highest efficiency.

Your submission must meet the following requirements:

- You need to submit a **detailed lab report in .pdf** format, **using the report template** provided on the UIT Courses website.
- Either Vietnamese or English report is accepted, that's up to you. The report written in the mixing of multiple languages 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 in your report. Simply attaching code without any explanation will not receive points.
- 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 the owner and the 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 3, chapter 4, chapter 6, chapter 7*

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

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

[4] Collisions. corkami/collisions: Hash collisions and exploitations (github.com)

Training platforms and related materials

- ASecuritySite-https://asecuritysite.com
- Cryptopals-https://cryptopals.com

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