A picture containing icon

Description automatically generated

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐHQG-HCM

**KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG**

BÁO CÁO THỰC HÀNH

NT101 – An toàn mạng máy tính

**LAB 03: Hash function and digital certificate**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thành viên (Nhóm TH1.02):**  20520890 – Hoàng Văn Anh Đức  20520245 – Lê Quang Minh  20520421 – Nguyễn Mạnh Cường  20520392 – Nguyễn Trần Đức Anh |  | |  | | --- | | Điểm tự đánh giá | |  | |

|  |  |
| --- | --- |
| Tổng thời gian thực hiện Lab trung bình | Vài ngày |
| Phân chia công việc | Làm chung |
| Ý kiến *(nếu có)*  + Khó khăn  + Đề xuất, kiến nghị |  |

**MỤC LỤC**

[**A.** BÁO CÁO CHI TIẾT 2](#_Toc117765123)

[1. 1. Generating message digests (hash values) and HMAC 2](#_Toc117765124)

[a. Generate the hash values of the arbitrary message which contains your student ID. Then compare the results with other tools to verify. 2](#_Toc117765125)

[2. Hash properties: One-way vs Collision-free 8](#_Toc117765126)

[a. Nội dung :  *Consider two HEX messages as follow:* 8](#_Toc117765127)

[*b.* *Có bao nhiêu bits/bytes khác nhau giữa 2 message . Dùng MD5 hash cho mỗi message xem kết quả hash có giống nhau hay không* 8](#_Toc117765128)

[*c.* *Download 2 file và dùng sha1 để hash và xem kết quả :* 8](#_Toc117765129)

[*d.* *Kết luận ? Giải thích tại sao lại tồn tại collision attack trong 2 hàm hash này ?* 8](#_Toc117765130)

[3. Generating Two Different Files with the Same MD5 Hash 9](#_Toc117765131)

[*a.* *Nếu tạo file prefix có nội dung (bytes) không phải là bội 64 thì chuyện gì sẽ xảy ra ?* 10](#_Toc117765132)

[b. Tạo file có 64bytes và chạy tool collision và xem chuyện gì xảy ra 11](#_Toc117765133)

[*c.* *Những dữ liệu (output) mà đc md5collgen tạo ra có khác nhau giữa 2 file không ?* 13](#_Toc117765134)

[4. Generating Two Executable Files with the same MD5 Hash 14](#_Toc117765135)

[5. Manually Verifying an X.509 Certificate 18](#_Toc117765136)

[*a.* *Bước 1 : Tải chứng chỉ xuống từ máy chủ web* 18](#_Toc117765137)

[*b.* *Bước 2: Trích xuất khóa công khai (e, n) từ chứng chỉ của người phát hành* 19](#_Toc117765138)

[*c.* *Bước 3: Trích xuất chữ ký từ chứng chỉ của máy chủ* 20](#_Toc117765139)

[*d.* *Bước 4: Giải nén nội dung chứng chỉ của máy chủ* 20](#_Toc117765140)

[6. Your application 26](#_Toc117765141)

[a. Với ứng dụng đã phát triển trong nhiệm vụ 3 của lab trước đó, hãy nâng cao tính toàn vẹn của dữ liệu trong quá trình vận chuyển và đề xuất một tính năng mới liên quan đến mật mã. 26](#_Toc117765142)

# BÁO CÁO CHI TIẾT

## 1. Generating message digests (hash values) and HMAC

### Generate the hash values of the arbitrary message which contains your student ID. Then compare the results with other tools to verify.

Thiết kế giao diện :

Graphical user interface, text, application, email

Description automatically generated

Data Format là 1 combobox cho phép chọn 3 dạng hash là “File” , “Text string”,”Hex string”

* ***Sự kiện nhấn nút open file :***

Text

Description automatically generated

Cho phép chọn đường dẫn đến file và hiển thị đường dẫn lên txbData

* ***Sự kiện nhấn nút calculate :***

Text

Description automatically generated

Hàm này đầu tiên sẽ kiểm tra dữ liệu nhập vào ở ô txbData có trống không , nếu có thì báo lỗi .Tiếp theo sẽ kiểm tra xem dữ liệu người dùng chọn ở combobox là gì để tiếp tục xử lí .

**Đối với việc người dùng muốn hash 1 file :**

Text

Description automatically generated

Text

Description automatically generated

Tham số truyền vào là đường dẫn của file đó

Đầu tiên ta dùng stream để mở file ra đọc

Sau đó hash và lưu vào string “hash”

Kết quả trả về chuyển thành dạng chữ thường

Tương tự cho 2 hàm với kiểu hash là : MD5 và SHA3

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

**Đối với hash 1 Text string :**

Text

Description automatically generated

Khởi tạo đối tượng “sha1” dùng để hash

Hash string được nhập vào và lưu dưới dạng byte

Khởi tạo đối tượng “sb” kiểu StringBuilder biểu diễn chuỗi kí tự có thể thay đổi có độ dài x2 độ dài của chuỗi byte ở trên

Dùng vòng lặp for thêm từng kí tự ở chuỗi byte”hash” đã thành kí tự viết thường vào “sb”

Cuối cùng trả về chuỗi đó dưới dạng string

Tương tự có ta có hàm

Text

Description automatically generated

Text

Description automatically generated

Chuyển string nhập vào thành mảng byte”encodedPassword” để xử lí

Hash mảng đó và lưu vào mảng “hash”

Trả về kết quả dưới dạng string (kí tự thường) sau khi đã bỏ các dấu “-“

Vd kết quả sau khi hash : 2-4-g-rh-l

=>Kết quả trả về : 24grhl

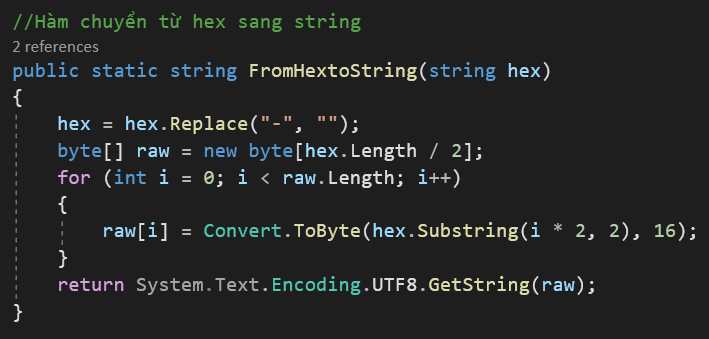
***Đối với việc hash dạng “Hex string”:***

Text

Description automatically generated

Đầu tiên ta phải kiểm tra dữ liệu đầu vào là số lượng byte nhập vào phải là chẵn và kí tự đó có dưới dạng hex hay không

Nếu thỏa mãn 2 điều kiện trên thì ta chuyển từ hex sang string kí tự và hash nó



Khởi tạo mảng byte = ½ hex

Xử lý 2 bytes của string hex cho đến khi khi hết string đó : chuyển lần lượt 2 bytes của hex sang string kí tự bình thường

Kết quả đc chuyển thành string trc khi trả về

* Sau khi đã chuyển sang string kí tự thì dùng các hàm của “Text string” (có đề cập ở trên ) để hash

Text

Description automatically generated

## Hash properties: One-way vs Collision-free

### Nội dung : *Consider two HEX messages as follow:*

**Message 1**

d131dd02c5e6eec4693d9a0698aff95c2fcab5**8**712467eab4004583eb8fb7f89

55ad340609f4b30283e4888325**7**1415a085125e8f7cdc99fd91dbd**f**280373c5b

d8823e3156348f5bae6dacd436c919c6dd53e2**b**487da03fd02396306d248cda0

e99f33420f577ee8ce54b67080**a**80d1ec69821bcb6a8839396f965**2**b6ff72a70

**Message 2**

d131dd02c5e6eec4693d9a0698aff95c2fcab5**0**712467eab4004583eb8fb7f89

55ad340609f4b30283e4888325**f**1415a085125e8f7cdc99fd91dbd**7**280373c5b

d8823e3156348f5bae6dacd436c919c6dd53e2**3**487da03fd02396306d248cda0

e99f33420f577ee8ce54b67080**2**80d1ec69821bcb6a8839396f965**a**b6ff72a70

### *Có bao nhiêu bits/bytes khác nhau giữa 2 message . Dùng MD5 hash cho mỗi message xem kết quả hash có giống nhau hay không*

2 message có 6 bytes khác nhau

Hash mess1 và 2 :

Mess1 : a482f36e3cf46428cbb0afb0594812e7

Mess2 : 3600560a5dec057e82321a169afe6a47

### *Download 2 file và dùng sha1 để hash và xem kết quả :*

shattered-1:

SHA1: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a

shattered-2:

SHA1: 38762cf7f55934b34d179ae6a4c80cadccbb7f0a

* Kết quả hash ra là giống nhau

### *Kết luận ? Giải thích tại sao lại tồn tại collision attack trong 2 hàm hash này ?*

Hàm băm có thể băm 2 input khác nhau và cho ra output giống nhau

* Tồn tại collision attack bởi vì khi số lượng băm tăng lên, nhiều khả năng các cặp dữ liệu khác nhau có thể đc tạo ra khi dùng cùng một hàm băm

Vd : Tỉ lệ 2 người có cùng ngày sinh nhật trong nhóm 1000 người sẽ cao hơn trong nhóm 10 người

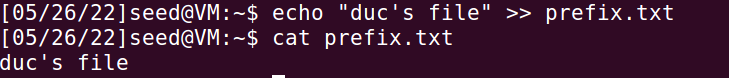
* Ứng dụng : bằng cách tạo hai tệp PDF đối chiếu thành hai hợp đồng thuê nhà với giá thuê khác nhau, có thể lừa ai đó tạo chữ ký hợp lệ cho hợp đồng thuê nhà cao bằng cách yêu cầu người đó ký hợp đồng thuê nhà thấp.

## Generating Two Different Files with the Same MD5 Hash

Text

Description automatically generated

Tạo prefix.txt và kiểm tra xem đã tạo đc chưa



Thêm nội dung vào prefix.txt và kiểm tra



Tạo 2 file với file tiền tố là prefix.txt

Text

Description automatically generated

Text

Description automatically generated

Kiểm tra có tạo thành công hay không

Text

Description automatically generated

Kiểm tra 2 file có khác nhau hay không (Có)

Text

Description automatically generated

Có thể dùng “md5sum <tên file>” để kiển tra file hash của nó

### *Nếu tạo file prefix có nội dung (bytes) không phải là bội 64 thì chuyện gì sẽ xảy ra ?*

Nếu như vậy thì khi dùng câu lệnh tạo ra 2 file thì tool sẽ tự động padding cho đủ 64bytes

xxd <tên file>

để kiểm tra

Text

Description automatically generated

Ta có thể thấy rất nhiều số 0 đc pad vào để đủ 64 bytes

### Tạo file có 64bytes và chạy tool collision và xem chuyện gì xảy ra

Text

Description automatically generated

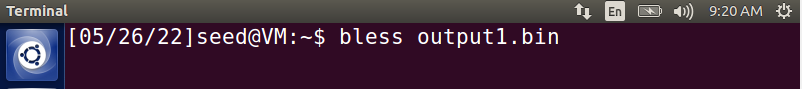
Tạo file prefix\_64.txt có 64 bytes

Text

Description automatically generated

Chạy tool và lưu vào 2 file out1\_1.bin và out2\_1.bin

* Prefix\_64.txt : out1\_1.bin ,out2\_1.bin
* Prefix.txt : output1.bin,output2.bin



Dùng tool bless để đọc file nhị phân

Graphical user interface, text, application, table, email

Description automatically generated

Ta thấy file đc padding để đủ 64 bytes

Graphical user interface, text, application, email

Description automatically generated

bless out1\_1.bin

khi đã đủ 64 bytes thì k cần padding thêm nữa

### *Những dữ liệu (output) mà đc md5collgen tạo ra có khác nhau giữa 2 file không ?*

A screenshot of a computer

Description automatically generated with low confidence

Lưu dữ liệu 2 file nhị phân và so sánh

Text

Description automatically generated

Text

Description automatically generated

## Generating Two Executable Files with the same MD5 Hash

Tổng hợp các bước để làm :

B1: Tạo mảng kí tự 200 “A” và thực thi

B2: Dùng bless để xem file binary của file thực thi

* Lấy prefix và suffix

B3: Tạo 2 file khi dùng md5collgen hash file prefix (thành 2 file a và b)

B4: Tạo p và q là 128 bytes cuối của a và b

B5: Nối p và q là 128 bytes giữa của prefix và suffix => ( thành c và d)

B6: Kiểm tra file binary của c và d (giống nhau thì thành công)

Text

Description automatically generated

Tạo mảng kí tự “A” (0x41) 200 bytes

Tạo file mới “task3.c” copy paste code c vào

A screenshot of a computer

Description automatically generated with medium confidence

Lưu file lại

Background pattern

Description automatically generated with low confidence

File out.txt

Copy ma trận và gõ :

sudo nautilus và thêm ma trận vào file task3.c

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Compile chương trình , lưu thành file task3.o và kiểm tra

Graphical user interface, text

Description automatically generated

bless task3.o để xem file

Ta để ý thấy Offset :4160 là thứ tự đầu tiên của mảng trong file binary

* Ta +64 =4224
* Lấy 4224 bytes đầu chính là file prefix

Text

Description automatically generated

Ta tạo file prefix

Và dùng nó để tạo ra 2 file output có cùng MD5 hash value

Text

Description automatically generated

Tạo thành công

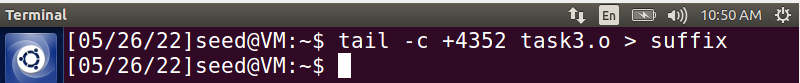
Bây giờ ta sẽ tạo file suffix :

Timeline

Description automatically generated with low confidence

Ta lấy prefix + 128 = byte đầu tiên của suffix

4224+128=4352



Tạo file chứa nội dung từ byte 4352 đến hết ở file task3.o và lưu vào suffix

* Đã prefix và suffix

Text

Description automatically generated

Tạo ra p và q (128 bytes) là 128 bytes cuối task3\_a.bin và task3\_b.bin ( là 2 file đc tạo ra khi dùng md5 collgen lên prefix )

Nối prefix + p + suffix = task3\_1

Nối prefix + q + suffix = task3\_2

Và kiểm tra file hash của nó

* 2 File hash hoàn toàn giống nhau

## Manually Verifying an X.509 Certificate

Xác minh chứng chỉ X.509 theo cách thủ công.

### *Bước 1 : Tải chứng chỉ xuống từ máy chủ web*

Chúng ta sử dụng máy chủ **www.example.org** . Chúng ta có thể tải các chứng chỉ xuống bằng trình duyệt hoặc sử dụng lệnh sau:

**openssl s\_client -connect www.example.org:443 -showcerts Text

Description automatically generated**

Text

Description automatically generated

Lưu 2 certificate vào 2 file gọi là c0.pem và c1.pem

### *Bước 2: Trích xuất khóa công khai (e, n) từ chứng chỉ của người phát hành*

Openssl cung cấp các lệnh để trích xuất các thuộc tính nhất định từ các chứng chỉ x509.

Chúng ta có thể trích xuất giá trị của n bằng cách sử dụng -modulus. Không có lệnh cụ thể để trích xuất e, nhưng chúng ta có thể in ra tất cả các trường và có thể dễ dàng tìm thấy giá trị của e.

n: **openssl x509 -in c1.pem -noout -modulus**

Text

Description automatically generated

**e: openssl x509 -in c1.pem -text**

Text

Description automatically generated



### *Bước 3: Trích xuất chữ ký từ chứng chỉ của máy chủ*

Không có lệnh openssl cụ thể nào để trích xuất trường chữ ký. Tuy nhiên, chúng ta có thể in ra tất cả các trường, sau đó sao chép và dán khối chữ ký vào một tệp

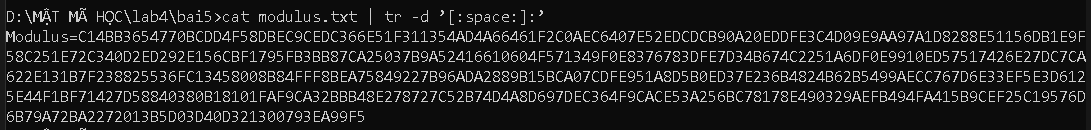
**openssl x509 -in c1.pem -text**

A black screen with white text

Description automatically generated with low confidence

* Lưu vào file modulus.txt

Chúng ta cần xóa khoảng trắng và dấu hai chấm khỏi dữ liệu, để chúng ta có thể nhận được một chuỗi hex mà chúng ta có thể thực hiện trong chương trình của mình. Các lệnh sau có thể đạt được mục tiêu này. Lệnh tr là một công cụ tiện ích của Linux cho các hoạt động chuỗi. Trong trường hợp này, tùy chọn -d được sử dụng để xóa “:” và “khoảng trắng” khỏi dữ liệu.

******

### *Bước 4: Giải nén nội dung chứng chỉ của máy chủ*

Tổ chức phát hành chứng chỉ (CA) tạo chữ ký cho server certificate bằng cách tính toán hàm băm của chứng chỉ, sau đó ký vào hàm băm. Để xác minh chữ ký, chúng ta cũng cần tạo hàm băm từ một chứng chỉ.

Kể từ khi hàm băm được tạo ra đến trước khi chữ ký đc tính toán, chúng ta cần loại trừ khối chữ ký của một chứng chỉ khi tính toán băm. Tìm hiểu phần nào của chứng chỉ được sử dụng để tạo hàm băm là một việc khá khó khăn nếu không có hiểu biết tốt về định dạng của chứng chỉ.

Chứng chỉ X.509 được mã hóa bằng ASN.1 (Ký hiệu cú pháp Notion.One) chuẩn, vì vậy nếu chúng ta có thể phân tích cú pháp cấu trúc ASN.1, chúng ta có thể dễ dàng trích xuất bất kỳ trường nào từ một chứng chỉ. Openssl có một lệnh gọi là asn1parse được sử dụng để trích xuất dữ liệu từ Dữ liệu được định dạng ASN.1 và có thể phân tích cú pháp chứng chỉ X.509 của chúng ra.

**openssl asn1parse -i -in c0.pem**

Text

Description automatically generated

Dòng đc highlight là phần thân của chứng chỉ (certificate ) được dùng để tạo hàm băm

Text

Description automatically generated

Còn đây là khối chữ ký (signature block)

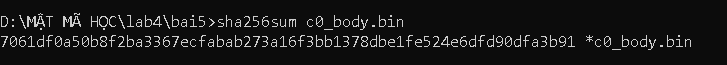
Trong trường hợp của chúng ta,phần thân chứng chỉ là từ offset 4 đến 1590, trong khi khối chữ ký là từ 1591 đến cuối tệp. Đối với chứng chỉ X.509, phần offset bắt đầu luôn giống nhau (tức là 4), nhưng phần cuối phụ thuộc vào độ dài nội dung của một chứng chỉ. Chúng ta có thể sử dụng tùy chọn -strparse để lấy trường từ độ lệch 4, sẽ cung cấp cho chúng ta phần thân của chứng chỉ, ngoại trừ khối chữ ký.

**openssl asn1parse -i -in c0.pem -strparse 4 -out c0\_body.bin -noout**

Khi chúng ta có được phần thân của chứng chỉ, chúng ta có thể tính toán băm của nó bằng cách sử dụng câu lệnh :

**sha256sum c0\_body.bin**

* Kết quả :



Bây giờ chúng ta có tất cả thông tin, bao gồm khóa công khai của CA, chữ ký của CA, và nội dung chứng chỉ của máy chủ. Chúng ta có thể chạy chương trình của riêng mình để xác minh chữ ký có hợp lệ hay không.

* Dùng pk của CA verify certificate của máy chủ mà giống với chữ kí của CA là ok

Text

Description automatically generated

***Cách 2 :***

**Bước 1 : Lưu toàn bộ chứng chỉ.**

openssl s\_client -connect medium.com:443 -showcerts

Chúng ta sẽ thấy hai chứng chỉ. Lưu cái đầu tiên trong medium.com.crt và cái thứ hai trong root.crt. Định dạng được sử dụng là PEM. Nó bao gồm BEGIN CERTIFICATE và END CERTIFICATE

Text

Description automatically generated

Text

Description automatically generated

Tại sao phải lưu hai chứng chỉ? Vì tất cả chúng cùng nhau tạo thành một chuỗi nên chứng chỉ được ký bằng khóa riêng của chứng chỉ gốc (parent’s certificate’s private key) , do đó xác thực chứng chỉ của con cho khi chứng chỉ gốc là chứng chỉ được cài đặt trên máy tính: do đó đáng tin cậy. Chuỗi chứng chỉ được cho là đáng tin cậy, khi và chỉ khi tất cả các chứng chỉ được xác thực bởi cha mẹ của nó. Một chuỗi có thể có một chứng chỉ - self-signed - nhiều - thường là 2 hoặc 3.

**Bước 2: Giải nén khóa công khai từ root.crt.**

openssl x509 -in root.crt -noout -pubkey > root.key

Text

Description automatically generated with medium confidence

**Bước 3: Trích xuất chữ ký từ medium.com.crt.**

openssl x509 -noout -text -in medium.com.crt

Text

Description automatically generated with low confidence

Text

Description automatically generated

Nó cho chúng ta biết, chữ ký được mã hóa bằng RSA và hàm băm đã được tính bằng sha256.

Một cách để trích xuất chữ ký là sử dụng dd. Nhưng trước tiên chúng ta cần tìm nơi để trích xuất dữ liệu thô. Chứng chỉ phải ở định dạng DER sau đó chúng ta cần phân tích cú pháp nó bằng cách sử dụng ans.1

openssl x509 -in medium.com.crt -outform der | openssl asn1parse -inform der

0:d=0 hl=4 l=1901 cons: SEQUENCE  
4:d=1 hl=4 l=1621 cons: SEQUENCE  
...  
1629:d=1 hl=2 l= 13 cons: SEQUENCE  
1631:d=2 hl=2 l= 9 prim: OBJECT :sha256WithRSAEncryption  
1642:d=2 hl=2 l= 0 prim: NULL  
1644:d=1 hl=4 l= 257 prim: BIT STRING

Theo RFC 3280 phần 4.1 (<https://datatracker.ietf.org/doc/html/rfc3280#section-4.1>

), cấu hình asn.1 trông giống như sau:

Diagram, text

Description automatically generated with medium confidence

d = là chiều sâu, hl = là chiều dài tiêu đề và l = là chiều dài nội dung. Vì vậy d = 0 là đối tượng gốc, d = 1 tiếp theo là đối tượng con đầu tiên cho đến d = 1 tiếp theo, v.v. Nhìn vào cấu hình x.509 asn.1, signatureValue là con cuối cùng từ gốc - vì vậy d = 1 cuối cùng.

Trích xuất chữ ký:

openssl x509 -in medium.com.crt -outform der |   
 dd skip=$((4+4+1621+2+13+4+1)) bs=1 > medium.com.sig

“4+4+1621+2+13+4+1” là chữ ký RSA! Để giải nén nó, chúng ta phải yêu cầu dd loại bỏ rất nhiều dữ liệu: tiêu đề của mỗi đối tượng và các đối tượng - tbsCertificate, signatureAlgorith và tiêu đề signatureValue.

Xem xét kỹ độ dài nội dung: nó dài 257 byte. Hoặc chữ ký RSA chỉ nên dài 256 byte. Byte đầu của BIT STRING được sử dụng để đệm. Có nghĩa là nếu nội dung không phải là bội số của 8 bit thì byte này sẽ bù cho nó. Vì byte đầu là 0x00 nên chúng ta có thể loại bỏ nó một cách an toàn.

Kiểm tra : openssl x509 -in medium.com.crt -outform der   
| dd skip=$((4+4+1621+2+13+4)) count=1 bs=1   
| xxd -ps -c 1

**Bước 4: Giải mã chữ ký.**

openssl rsautl -verify -pubin -inkey root.key -in medium.com.sig | hexdump

Text

Description automatically generated with medium confidence

* Hàm băm để ta xác thực là : fcca7ea7fc1dbb08f608b55a198ce0323d6c8a8103e9b9e9fca65068070910ee

Bước 5: xác minh hàm băm.

Trường signatureValue chứa chữ ký điện tử được tính toán dựa trên tbsCertificate được mã hóa ASN.1 DER. TbsCertificate được mã hóa ASN.1 DER được sử dụng làm đầu vào cho hàm chữ ký.

tbs = to be signed . không thể ký toàn bộ chứng chỉ có chứa chữ ký

Chúng ta sẽ cần dd một lần nữa. Lần này chúng ta sẽ giải nén tbsCertificate.

openssl x509 -outform der -in medium.com.crt \  
| dd bs=1 count=$((4+1621)) skip=4 \  
| shasum -a 256

=>Kết quả : fcca7ea7fc1dbb08f608b55a198ce0323d6c8a8103e9b9e9fca65068070910ee

=>Verify thành công

## Your application

### Với ứng dụng đã phát triển trong nhiệm vụ 3 của lab trước đó, hãy nâng cao tính toàn vẹn của dữ liệu trong quá trình vận chuyển và đề xuất một tính năng mới liên quan đến mật mã.

Thêm mã hóa ECC

Graphical user interface, application, email

Description automatically generated

Dùng để mã hóa key của AES

* Trao đổi khóa an toàn hơn
* Khóa dùng để mã hóa và giải mã (AES) sẽ đc mã hóa

Demo :

Graphical user interface

Description automatically generated with medium confidence

Khi Sender muốn gửi message thì message sẽ được mã hóa bằng AES

Graphical user interface, text, application, email

Description automatically generated

Bên nhận tạo cặp khóa để bên gửi dùng PK để mã hóa

Graphical user interface, application

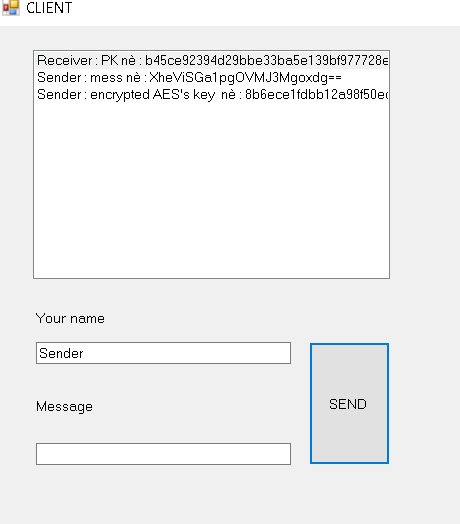
Description automatically generated

Receiver gửi PK để Sender dùng nó để mã hóa ( như vậy thì chỉ có Receiver ms giải mã đc)

Graphical user interface, text, application, email

Description automatically generated

Dùng ECC và PK của Receiver để mã hóa key của AES và gửi cho Receiver



Sender gửi key và mess đã đc mã hóa cho Receiver

Graphical user interface, text, application, email

Description automatically generated

Rceiver giải mã key của AES bằng PR của mình

Graphical user interface, application

Description automatically generated

Receiver dùng key đã đc giải mã để giải mã message mà Sender gửi cho