**UNIVERSITY OF SCIENCE – HCMVNU**

**FACILITY OF INFORMATION TECHNOLOGY**

Logo

Description automatically generated

**LAB**

**Subject: Application and Cryptography**

*RSA Encryption and Signature Lab*

Tp. Hồ Chí Minh – 03/2022

Mục lục

[Overview: 3](#_Toc98587214)

[Task 1: Deriving the Private Key 3](#_Toc98587215)

[Task 2: Encrypting a Message 4](#_Toc98587216)

[Task 3: Decrypting a Message 4](#_Toc98587217)

[Task 4: Signing a Message 5](#_Toc98587218)

[Task 5: Verifying a Signature 7](#_Toc98587219)

[Task 6: Manually Verifying an X.509 Certificate 8](#_Toc98587220)

# Overview:

|  |  |
| --- | --- |
| **Student ID** | **Full name** |
| 19127525 | Nguyễn Thanh Quân |
| 19127625 | Lâm Chí Văn |

# Task 1: Deriving the Private Key

Function **Task1(…)** in file **task1.c** will return *private key d*

Text

Description automatically generated

Follow [Wiki/RSA\_(cryptosystem)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

A picture containing text, watch

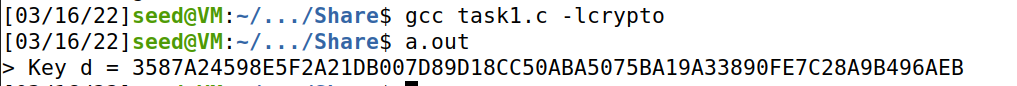
Description automatically generated

With

First, I *calculate* (line 19 to 28)

Finnally, use **BN\_mod\_inverse(…)** function of **openssl/bn.h** library (line 31) to *calculate* ***d***

Result: d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB



# Task 2: Encrypting a Message

Function **Task2\_Encrypt(…)** in file **task2.c** will return *encrypt message (ciphertext)* with Hex format

Use formula to calculate ciphertext with Hex format

First, *convert a* ***message*** *from ASCII string format to Hex string* by **string2hexString(…)**(line 46)*.*

Then init its to BIGNUM variable (m) (line 47)

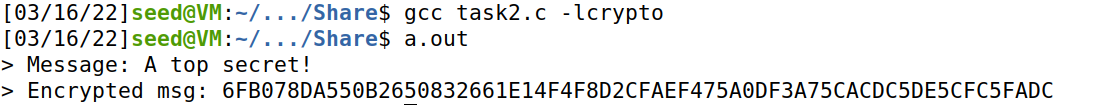
Finally, use formula (line 50) to *find* ***ciphertext***

Text

Description automatically generated

Result:

Encrypt message: 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC



# Task 3: Decrypting a Message

Function **Task3\_Decrypt(…)** in file **task3.c** will print *Decrypted message (plaintext)* with ASCII string format

Use formula to *calculate* ***plaintext*** with Hex format

First, use formula to *find* ***Decrypt message*** *with Hex string* format by **BN\_mod\_exp(..)** (line 41)

Finally, *convert* ***m*** *from Hex string to ASCII string* by **textFromHexString(…)** (line 47)

* Find origin message (plaintext)



Result:

Origin message: Password is dees

Text

Description automatically generated

# Task 4: Signing a Message

Function **Task4\_RSAsign(…)** in file **task4.c** will return *Sign of message* in Hex format

With instruction in <https://cryptobook.nakov.com/digital-signatures/rsa-signatures>

Graphical user interface, text, application, email

Description automatically generated

We must hash msg first but follow the instruction in **Task 4**. We do not need to hash this message.

So, we *convert this* ***message*** *from string to Hex string* by **string2hexString(…)** (line 46) (consider this value is h(msg))

Finally, like Decrypt message, *use private key d to* ***sign this message*** by formula to by **BN\_mod\_exp(..)** (line 50)

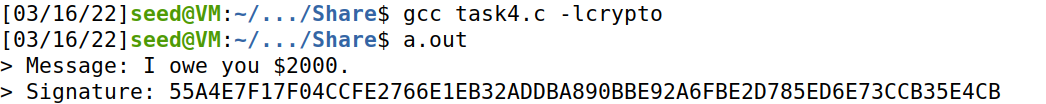
* We had *Sign this message*

Text

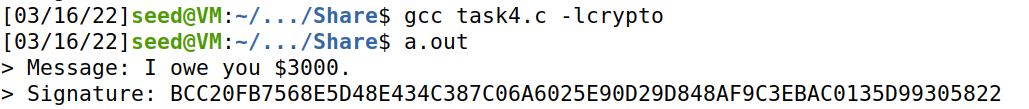
Description automatically generated

Result:

With Message: “I owe you $2000.”, we will have sign is 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB



With Message: “I owe you $3000.”, we will have sign is BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822



From 2 result above, we can see, these sign is **complete different**

55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CCB35E4CB

and

BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D99305822

# Task 5: Verifying a Signature

Function **Task5\_VerifySignature(…)** in file **task5.c** will *return 0 or 1 with (0 is True and 1 is False)*

With instruction in <https://cryptobook.nakov.com/digital-signatures/rsa-signatures>

Graphical user interface, text, application, email

Description automatically generated

We must hash msg first but follow the instruction in **Task 5**, and we do not need to hash this message.

First, we *calculate* ***h’*** by formula (line 45) (here, I assign h2 as h’)

Then, we *calc* ***h = h(M)*** by *converting* ***M*** *from ASCII string to hex string* by **string2hexString(…)** (line 48)

Finally, we *compare* ***h*** *and* ***h’*** (line 54).

* If return value is 0, this Sign is True, and else this Sign is False

Text

Description automatically generated

With S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB680**2F**

We have h and h’ value is 4C61756E63682061206D697373696C652E

h and h’ is equal Result: This signature is indeed Alice's

Text

Description automatically generated

With S = 643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB680**3F**

We have h value is: 4C61756E63682061206D697373696C652E

And h’ value is: 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294

h and h’ is different Result: This signature is NOT indeed Alice's

Text, application

Description automatically generated

# Task 6: Manually Verifying an X.509 Certificate

Firstly, this task will use the website [www.mozilla.org](http://www.mozilla.org) for certificate validation.

Using command *openssl s\_client -connect www.mozilla.org:443 -showcerts* to download the certificate for extraction. → got c0.pem and c1.pem

A close-up of a document

Description automatically generated with medium confidence

C0.pem

A picture containing text, newspaper

Description automatically generated

c1.pem

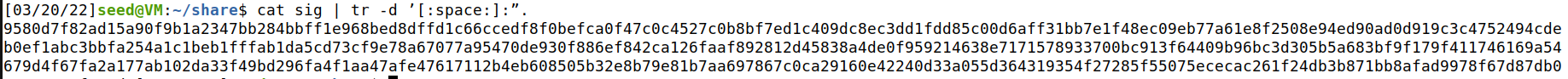
From c1.pem, we can read and get the public key, with the command “openssl x509 -in c1.pem -text -noout” to print-all data we got from c1.pem. We can extract exponent (e) and modulus (n) from that.

A screenshot of a computer

Description automatically generated with medium confidence

*RSA public key*

We got the signature key from c0.pem, using the command “openssl x509 -in c0.pem -text -noout”. However, We need to remove the spaces and colons from the data to get a hex string, which can be done with the “cat signature | tr -d ’[:space:]:”.



signature

Last step, using “openssl asn1parse -i -in c0.pem -strparse 4 -out c0\_body.bin -noout” to get the body of certificate, then hash it with “sha256sum c0\_body.bin”.

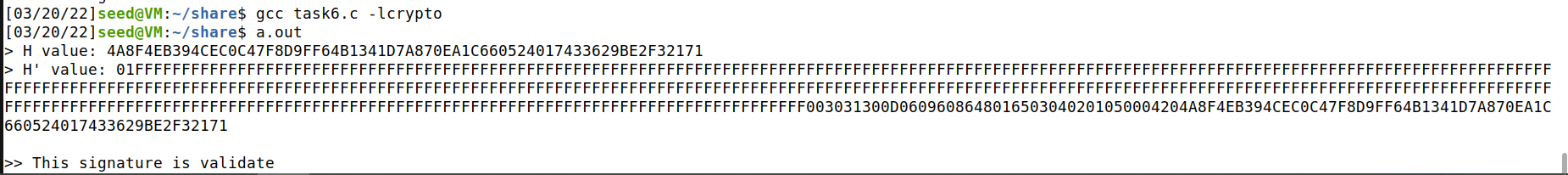


Body of certificate after hasing

After having all needed information, verify the signature

The final coding to verify the above signature was done in task 5.

Result:



*result*

This is the result after calculation; however, the first part looks different because it is padding. The last part is the same, so it returns a TRUE value.

So this is a validated certificate.