



ehalr24 / 386MK



<> Code

Issues

Pull requests

Actions

Projects

Security

Insights

386MK / Karicher_MP.md



mtkaricher Update Karicher_MP.md

e5f8d95 · now

History



Update Karicher_MP.md
e5f8d95

Fixed formatting error

329 lines (251 loc) · 14 KB

386MK / Karicher_MP.md

↑ Top

Preview

Code

Blame

Raw



Abstract


For my mini project I will be completing the SEED Lab on RSA encryption and digital signature. I will be using this document for documentation and attach all programs in my submission on canvas along with a presentation and video. This lab will require programming in C and use of imported libraries to perform mathematical functions.

Set Up

To begin this lab I will find all of the neccassary files to download from the [SEED Labs Library](#).

The necessary files include the PDF document outlining the lab and all tasks required for completion, along with a pre-filled Ubuntu virtual machine.

Create Virtual Machine



> Name and Operating System

> Unattended Install

> Hardware

▼ Hard Disk

☒ Create a Virtual Hard Disk Now

Hard Disk File Location and Size

C:\Users\mtkar\VirtualBox VMs\SEED Lab\SEED Lab.vdi

4.00 MB

25.00 GB

2.00 TB

Hard Disk File Type and Variant

VDI (VirtualBox Disk Image)

☐ Pre-allocate Full Size

☐ Split into 2GB parts

☐ Use an Existing Virtual Hard Disk File

SEED-Ubuntu20.04.vdi (Normal, 80.00 GB)

☐ Do Not Add a Virtual Hard Disk

Help

Guided Mode

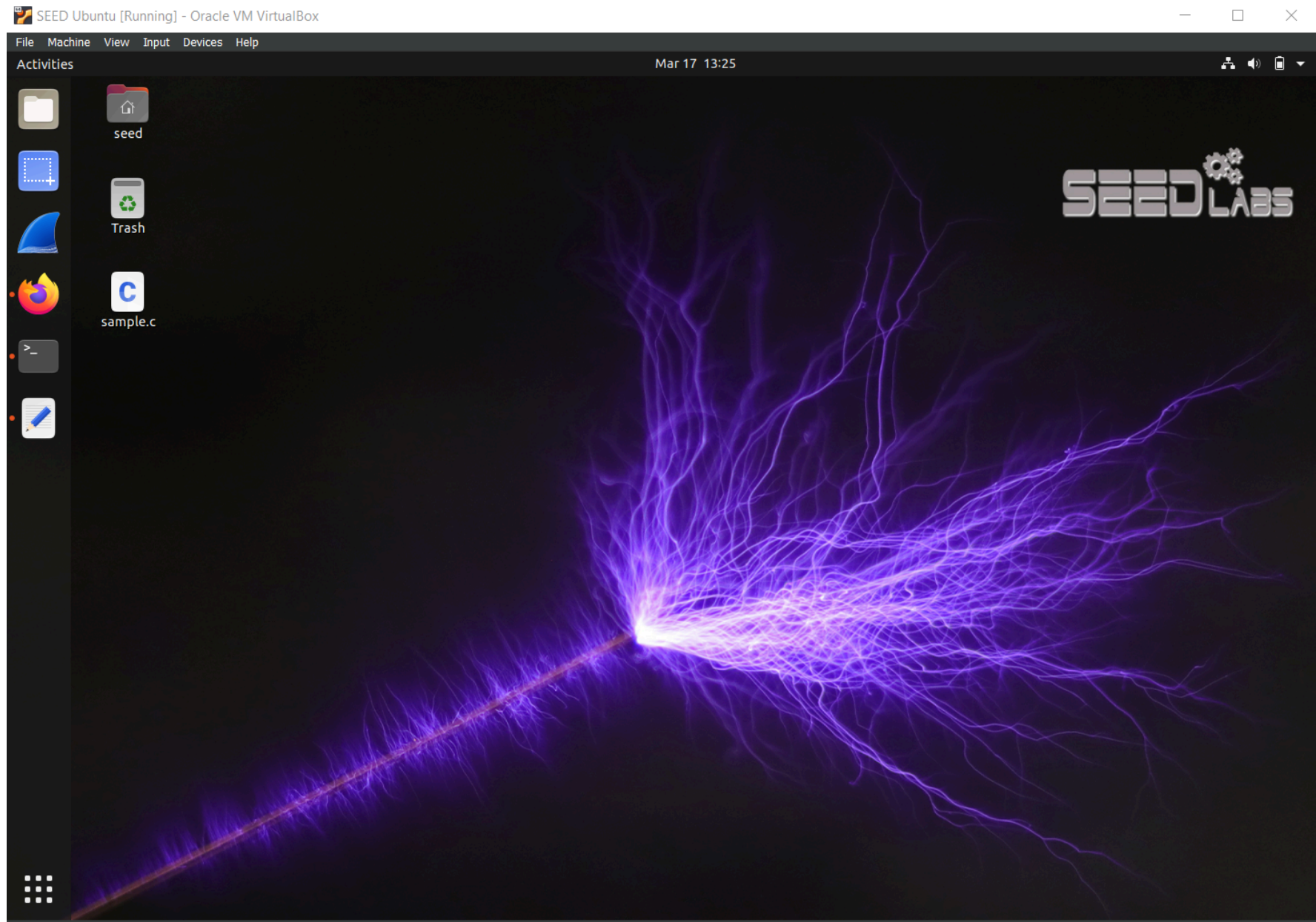
Back

Finish

Cancel

Once everything is downloaded, I will use my VM software to create a new virtual machine with the SEED Labs Ubuntu image ready to load.

Once this machine is booted up we are greeted by the seeds lab install screen.



Now I will want to keep everything organized so I will create a directory for this project and then various empty files that I can use for programming. This lab will be done with C programming.

Input:

```
rmdir Project
cd Project
touch task1.c
touch task2.c
touch task3.c
touch task4.c
touch task5.c
touch task6.c
```



None of these commands have output. I can verify their completion by running the ls command between each input. I am now ready to begin Task 1.

Task 1: Deriving the private key

Task 1: Let p , q , and e be three prime numbers. Let $n = p \cdot q$. We will use (e, n) as the public key. Please calculate the private key d . The hexadecimal values of p , q , and e are listed in the following. It should be noted that although p and q used in this task are quite large numbers, they are not large enough to be secure. We intentionally make them small for the sake of simplicity. In practice, these numbers should be at least 512 bits long (the one used here are only 128 bits).

For this task:

```
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3
```



To begin, I referenced my notes on how to do this. To derive the key, I want to use the formula, $d = e^{-1} \bmod \phi(n)$. First I should calculate $\phi(n)$ (will use variable `pon`).

First, I used the `print BN` function given to me in the documentation for the lab setup. This will be used when printing any of the numbers stored in the BN data type.

In the main function, I will need to create all variables that will be used as Big number data types.

```
BN_CTX *CTX = BN_CTX_new(); //context, to keep data in line
BIGNUM *p = BN_new(); //p
BIGNUM *q = BN_new(); //q
BIGNUM *e = BN_new(); //e
BIGNUM *d = BN_new(); //secret key
BIGNUM *n = BN_new(); //n
BIGNUM *pmo = BN_new(); //p minus one
BIGNUM *qmo = BN_new(); //q minus one
BIGNUM *pon = BN_new(); // phi of n for mod operation
BIGNUM *one = BN_new(); //need one to create qmo and pmo
```



I will be programming an implementation of the algorithm that we learned about in class. To do this with the big number library is simple when you understand all of the programs in the library.

Once the variables are created they need to be initialized with the data given to us in the seed lab documentation.

```
BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
BN_hex2bn(&e, "0D88C3");
BN_dec2bn(&one, "1");
```



To begin the formula, we need to create our `p minus one`, `q minus one`, and `phi of n`.

```
BN_sub(pmo, p, one);
BN_sub(qmo, q, one);
```



```
BN_mul(pon, pmo, qmo, ctx);
```

With all of the needed variables created, we can run the mod inverse command and get $d = e^{-1} \bmod \text{pon}$.

```
BN_mod_inverse(d, e, pon, ctx);
```

Now that we should have our private key, we can call the function created earlier to print our key out in hexadecimal form.

```
printBN("", d);
```

FULL INPUT OF TASK ONE PROGRAM:

```
#include <stdio.h>
#include <openssl/bn.h>

#define NBITS 256

//First, I need to create the method to print any big numbers
void printBN(char *msg, BIGNUM *a) {
    //convert the BN type to number string then print
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    //finish by freeing the DAM
    OPENSSL_free(number_str);
}

int main() {
    //for main, I will be implementing the secret key formula
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *p = BN_new(); //p
    BIGNUM *q = BN_new(); //q
    BIGNUM *e = BN_new(); //e
    BIGNUM *d = BN_new(); //secret key
```

```

    BIGNUM *n = BN_new(); //n
    BIGNUM *pon = BN_new(); //pi of n (p-1)*(q-1)
    BIGNUM *one = BN_new(); //create a 1 to use in bn operations
    BIGNUM *pmo = BN_new(); //p minus one
    BIGNUM *qmo = BN_new(); //q minus one

    //now, initialize the given variables
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");
    BN_dec2bn(&one, "1");

    //create pi of n for use in later mod operation
    //I will have to create extra variables to store p-1 and q-1
    BN_sub(pmo, p, one);
    BN_sub(qmo, q, one);

    //create pi of n
    BN_mul(pon, pmo, qmo, ctx);

    //perform mod operation, d = e^(-1) mod pon
    BN_mod_inverse(d, e, pon, ctx);

    //call the function created at the top to print the private key
    printBN("",d);

    return 0;
}

```

Now that we have the program we will compile it using the provided command by seed: `gcc -o task1.exe task1.c -lcrypto` and `./task1.exe` to be given the output of:

3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB



```
seed@VM: ~/Project
[03/17/24] seed@VM:~/Project$ ls
Documentation.txt  task1.c
[03/17/24] seed@VM:~/Project$ gcc -o task1.exe task1.c -lcrypto
[03/17/24] seed@VM:~/Project$ ls
Documentation.txt  task1.c  task1.exe
[03/17/24] seed@VM:~/Project$ ./task1.exe
3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[03/17/24] seed@VM:~/Project$
```

With the private key being displayed, task 1 has been successfully completed.

Task 2: Encrypting a Message

Task 2: let (e, n) be the public key. Please encrypt the message "A top secret!" (the quotations are not included). We need to convert this ASCII string to a hex string, and then convert the hex string to a BIGNUM using the hex-to-bn API `BN_hex2bn()`. The following python command can be used to convert a plain ASCII string to a hex string.


For this task I will take the given message and use the e we were previously given to encrypt the message. To set this up I will begin by converting the message to hexadecimal using [This online tool](#).

From

Text

To

Hexadecimal

 Open File



Paste text or drop text file

A top secret!


Character encoding


ASCII

Output delimiter string (optional)

Space

 Convert

 Reset

 Swap

41 20 74 6F 70 20 73 65 63 72 65 74 21

Now, I will begin coding the c program to perform the encryption. I will start the same way as the previous code by creating a function that will print the final piece when needed.

```
void printBN(char *msg, BIGNUM *a) {  
    //convert the BN type to number string then print  
    char * number_str = BN_bn2hex(a);  
    printf("%s %s\n", msg, number_str);  
    //finish by freeing the DAM  
    OPENSSL_free(number_str);  
}
```



The next step will to again declare all variables needed using the same code as the previous task. For this task we will need e, n, the message, and a variable for the final cipher.

```
//declare variables e, n, original message, cipher text  
    BN_CTX *ctx = BN_CTX_new(); //we need this context to temporarily store variables while performing  
multi function operations  
    BIGNUM *n = BN_new();  
    BIGNUM *e = BN_new();  
    BIGNUM *mesHex = BN_new();  
    BIGNUM *cipHex = BN_new();
```



With all of the variables declared, they will be initialized with the data given to us by SEED labs.

```
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");  
BN_hex2bn(&e, "010001");  
BN_hex2bn(&mesHex, "4120746f702073656372657421"); //derived from the seed document above
```



Now, we are going to compute the ciphertext with the formula we already know; $\text{cipHex} = \text{mesHex}^e \bmod n$.

```
BN_mod_exp(cipHex, mesHex, e, n, ctx);
```



FULL INPUT OF TASK 2

```
#include <stdio.h>
#include <openssl/bn.h>

//function to print big number
void printBN(char *msg, BIGNUM *a) {
    //convert the BN type to number string then print
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    //finish by freeing the DAM
    OPENSSL_free(number_str);
}

int main() {

    //declare variables e, n, d, original message, cipher text
    BN_CTX *ctx = BN_CTX_new(); //context to temporarily store variables for multi function ops.
    BIGNUM *n = BN_new(); //result of p*q
    BIGNUM *e = BN_new(); //public key for encryption
    BIGNUM *d = BN_new(); //private key for decryption, used to verify
    BIGNUM *mesHex = BN_new(); //hexadecimal form of the ASCII plaintext
    BIGNUM *cipHex = BN_new(); //final ciphertext

    //now initialize variables that were given to us from the seed document
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&mesHex, "4120746f702073656372657421"); //derived from the seed document above

    //Use the formula cipHex = mesHex^e mod n to find ciphertext
    BN_mod_exp(cipHex, mesHex, e, n, ctx);

    //now call the print function
```



```

    printBN("", cipHex);

    return 0;
}

```

Now that we have the cipher text, we can compile the program and execute to get the encrypted message.

```

seed@VM: ~/Project
[03/17/24] seed@VM: ~/Project$ ls
Documentation.txt  task1.c  task1.exe  task2.c
[03/17/24] seed@VM: ~/Project$ gcc -o task2.exe task2.c -lcrypto
[03/17/24] seed@VM: ~/Project$ ls
Documentation.txt  task1.c  task1.exe  task2.c  task2.exe
[03/17/24] seed@VM: ~/Project$ ./task2.exe
6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
[03/17/24] seed@VM: ~/Project$

```

Task 3: Decrypting a Message

Task 3: The public/private keys used in this task are the same as the ones used in Task 2. Please decrypt the following ciphertextC, and convert it back to a plain ASCII string.

C = 8C0F971DF2F3672B28811407E2DABBE1DA0FEBBDFC7DCB67396567EA1E2493F

Now that we have the encrypted message from task two, we are given the secret key and are able to test our encryption by decrypting it.

Again, we begin by importing the neccassary header files and creating a function to print the Big Numbers as string numbers in hex form.

From there, I created a main method and followed the same steps as before in declaring variables then declaring them with the values given in the SEED labs documentation. For this task we use all the same variables as the previous task.

```
//Declare variables
BN_CTX *ctx = BN_CTX_new();
BIGNUM *n = BN_new();
BIGNUM *d = BN_new();
BIGNUM *cipHex = BN_new();
BIGNUM *mesHex = BN_new();

//Initialize variables with given documentation
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
BN_hex2bn(&cipHex, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBBDFC7DCB67396567EA1E2493F");
```

The difference for task 3 is that the ciphertext is not the ciphertext we generated in task 2, but a different ciphertext provided.

Now, I use the formula of $\text{plaintext} = \text{ciphertext}^d \bmod n$ to get the hexadecimal form of the plaintext.

```
BN_mod_exp(mesHex, cipHex, d, n, ctx);
```

Once this is gotten, I will call the print function to print it out for me.

```
printBN("", mesHex);
```

FULL INPUT CODE TASK 3:

```
#include <stdio.h>
#include <openssl/bn.h>
```

```

//function to print big number
void printBN(char *msg, BIGNUM *a) {
    //convert the BN type to number string then print
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    //finish by freeing the DAM
    OPENSSL_free(number_str);
}

int main() {

    //declare all big numeber variables
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *cipHex = BN_new();
    BIGNUM *mesHex = BN_new();

    //Initialize variables with given documentation
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&cipHex, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBD7FC7DCB67396567EA1E2493F");

    //decrypt message with formula from class mesHex = cipHex^d mod n
    BN_mod_exp(mesHex, cipHex, d, n, ctx);

    //now that we have the plaintext stored in mesHex, we will print it using the above function
    printBN("", mesHex);

    return 0;
}

```

Once we compile and run the program from the terminal, we are given a hexadecimal number of **50617373776F72642069732064656573**. Running it through [the website we used in task 2](#), we are given the ASCII message, "The password is dees". This is also the password to access the seeds virtual machine.

Hex to ASCII Text String Converter

Enter hex bytes with any prefix / postfix / delimiter and press the *Convert* button

(e.g. 45 78 61 6d 70 6C 65 21):

From


Hexadecimal



To

Text



 Open File



Paste hex numbers or drop file


50617373776F72642069732064656573


Character encoding

ASCII



 Convert

 Reset

 Swap

Password is dees

```
seed@VM: ~/Project
[03/17/24] seed@VM:~/Project$ ls
Documentation.txt task1.c task1.exe task2.c task2.exe task3.c
[03/17/24] seed@VM:~/Project$ gcc -o task3.exe task3.c -lcrypto
[03/17/24] seed@VM:~/Project$ ls
Documentation.txt task1.c task1.exe task2.c task2.exe task3.c task3.exe
[03/17/24] seed@VM:~/Project$ ./task3.exe
50617373776F72642069732064656573
[03/17/24] seed@VM:~/Project$
```

Final Report

Through completion of this lab I was able to reinforce skills in programming and RSA encryption. By developing programs to perform various steps of the RSA encryption process I will be able to implement these practices in future developments. This was also performed in Ubuntu with all text editing, compilation, and execution run through the command line.

Resources

https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_RSA/

<https://github.com/seed-labs/seed-labs/blob/master/manuals/vm/seedvm-manual.md>

https://seedsecuritylabs.org/Labs_20.04/Files/Crypto_RSA/Crypto_RSA.pdf

<https://www.rapidtables.com/convert/number/hex-to-ascii.html>