SYSC 4810: Introduction to Network and Software Security
Assignment #1
Fall 2019

Dr. J. Jaskolka
Carleton University
Department of Systems and Computer Engineering

Posted: September 6, 2019

Due: October 4, 2019

Due on Friday, October 4, 2019 by 11:55PM

This assignment contains 16 pages (including this cover page) and 7 problems. You are responsible for ensuring that your copy of the assignment is complete. Bring any discrepancy to the attention of your instructor.

Special Instructions:

1. **Do as many problems as you can.**
2. Start early as this assignment is much more time consuming than you might initially think!
3. The burden of communication is upon you. Solutions not properly explained will not be considered correct. Part of proper communication is the appearance and layout. If we cannot "decode" what you wrote, we cannot grade it as a correct solution.
4. You may consult outside sources, such as textbooks, but *any use* of *any source* **must** be documented in the assignment solutions.
5. You are permitted to discuss *general aspects* of the problem sets with other students in the class, but you must hand in your own copy of the solutions.
6. Your assignment solutions are due by 11:55PM on the due date and must be submitted on cuLearn.
   - Late assignments will be graded with a late penalty of 20% of the full grade per day *up to 48 hours past the deadline.*
7. You are responsible for ensuring that your assignment is submitted correctly and without corruption.

Problem	1	2	3	4	5	6	7	Total
Points:	10	10	5	10	5	10	10	60

In this assignment, you will participate in activities related to the operation and use of cryptographic tools and techniques. This assignment aims to assess your understanding of the basic principles underlying the main cryptographic concepts and technologies available today, including symmetric and asymmetric encryption, and digital signatures.

Acknowledgment

This assignment is based off the SEED Labs: "Secret-Key Encryption Lab" and "RSA Public-Key Encryption and Signature Lab" developed by Wenliang Du at Syracuse University.

Submission Requirements

Please read the following instructions very carefully and follow them precisely when submitting your assignment!

The following items are required for a complete assignment submission:

1. **PDF Assignment Report**: Submit a detailed report that carefully and concisely describes what you have done and what you have observed. Include appropriate code snippets and listings, as well as screenshots of program outputs and results. You also need to provide an adequate explanation of the observations that are interesting or surprising. You are encouraged to pursue further investigation beyond what is required by the assignment description.
2. **ZIP Archive of Source Code**: In addition to embedding source code listings in your assignment report, create and submit a ZIP archive of all programs that you write for this assignment. Please name each of your source code files with the problem number to which they correspond (e.g., for Problem 7(a), the source code file should be named `Problem7a.c`). Your source code must compile and run, producing the desired output. Also, please remember to provide sufficient comments in your code to describe what it does and why.
3. **ZIP Archive of Screenshot Image Files**: In addition to embedding screenshots of program outputs and results in your assignment report, create and submit a ZIP archive of all of the raw screenshot images that you capture for this assignment.

Grading Notes

An important part of this assignment is following instructions. As such, the following grade **penalties** will be applied for failure to comply with the submission requirements outlined above:

- Failure to submit an Assignment Report will result in a grade of 0 for the assignment.
- Failure to submit the Source Code files will result in deduction of 10% of the full grade of the assignment.
- Failure to submit the Screenshot Image files will result in deduction of 10% of the full grade of the assignment.
- Failure of Source Code to compile/run will result in a grade of 0 for the corresponding problem(s).
- Failure to submit any deliverable in the required format (PDF or ZIP) will result in deduction of 5% of the full grade of the assignment.

Part I Environment Setup

This assignment will be conducted using a pre-built virtual machine (VM) image. All of the necessary tools, software, and libraries that are needed for the assignment have been installed on the virtual machine image. In this part of the assignment, we will walk-through the setup of this virtual machine image to allow you to complete the rest of the assignment problems and tasks.

Important Note You are ***NOT*** required to submit any formal documentation of the steps you have taken to set up the virtual machine. This is merely preliminary work to facilitate the remainder of the assignment.

1 Obtaining the Virtual Machine Image and Software

To set up the necessary virtual machine environment for this assignment you will need the following resources:

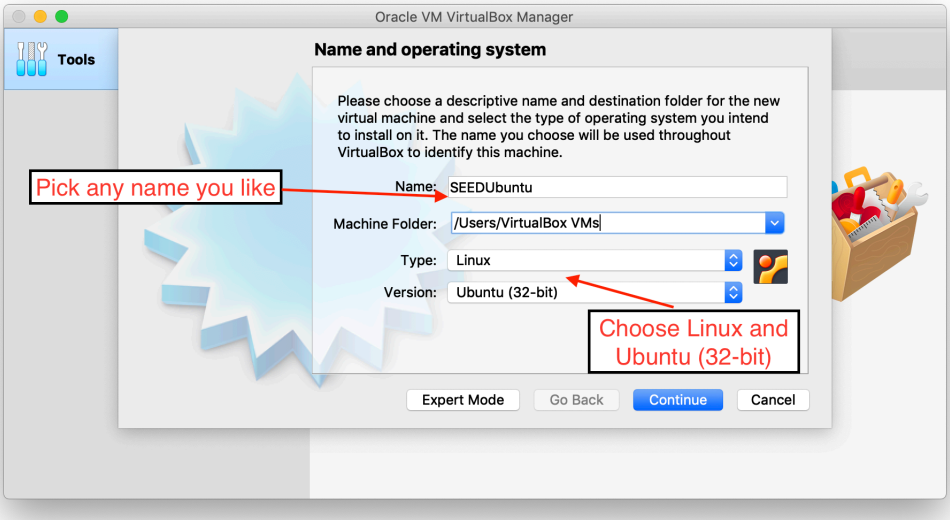
1. Download the pre-built Ubuntu Virtual Machine Image.
2. Download and install the free VirtualBox software.
   - We recommend **Version 6.0.4**. Newer versions are known to have some issues with the pre-built Ubuntu Virtual Machine Image.

2 Setting Up the Virtual Machine Image in VirtualBox

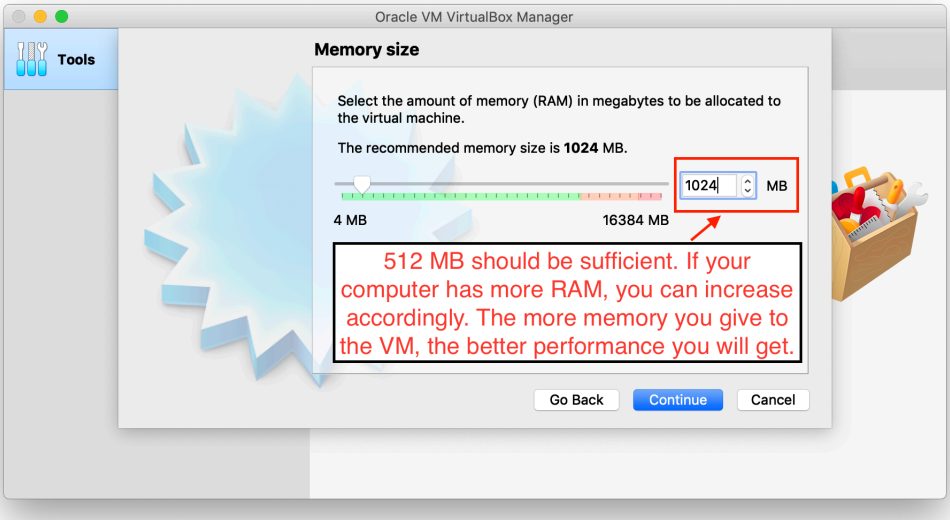
To set up the pre-built virtual machine image in Virtual Box, please follow the steps below. Note that the screenshots may look different depending on the version of VirtualBox and the host operating system.

1. **Create a New Virtual Machine in VirtualBox**

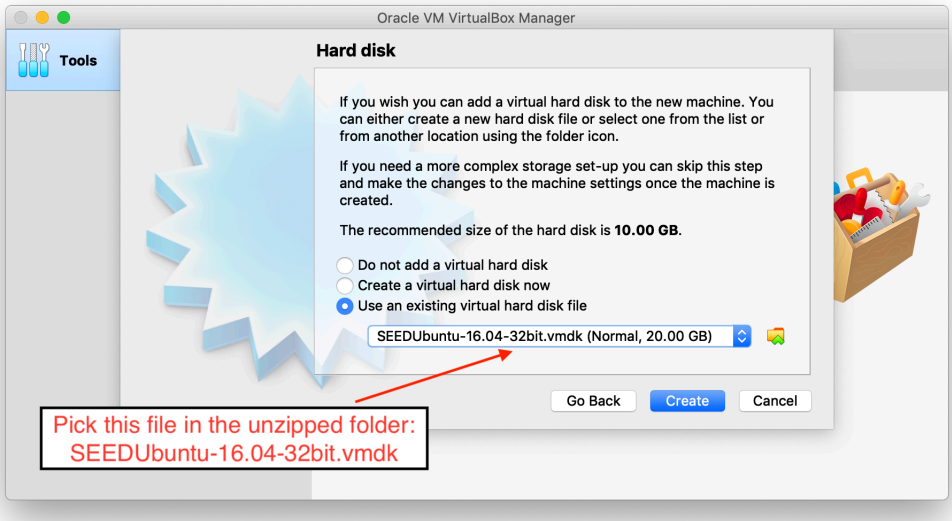
2. **Provide a Name and Select the OS Type and Version**: Do NOT pick Ubuntu (64-bit), even though your machine is 64-bit. The pre-built virtual machine is 32-bit Ubuntu.

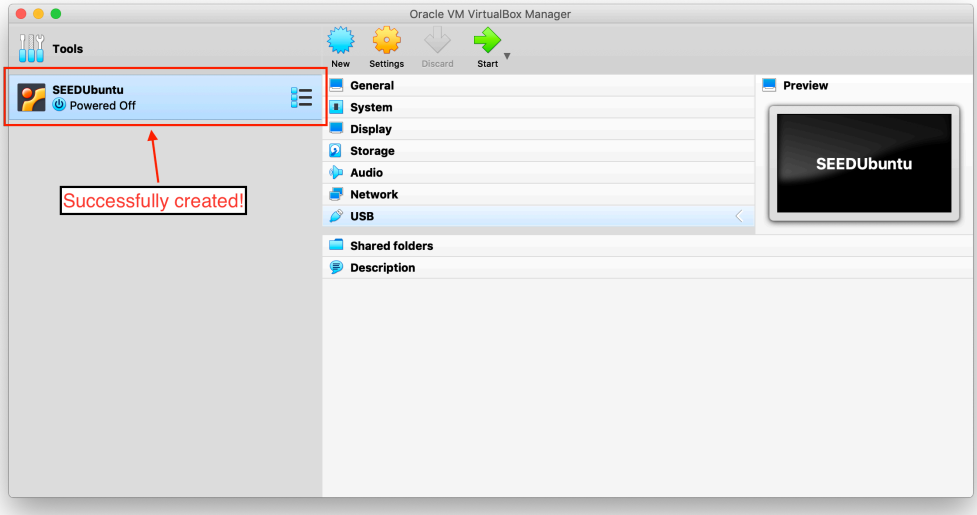


3. **Set the Memory Size**

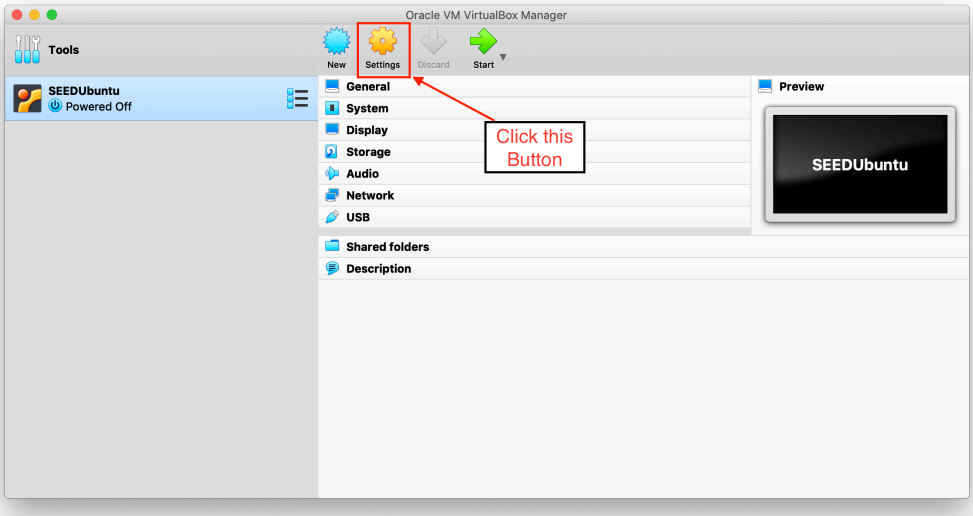


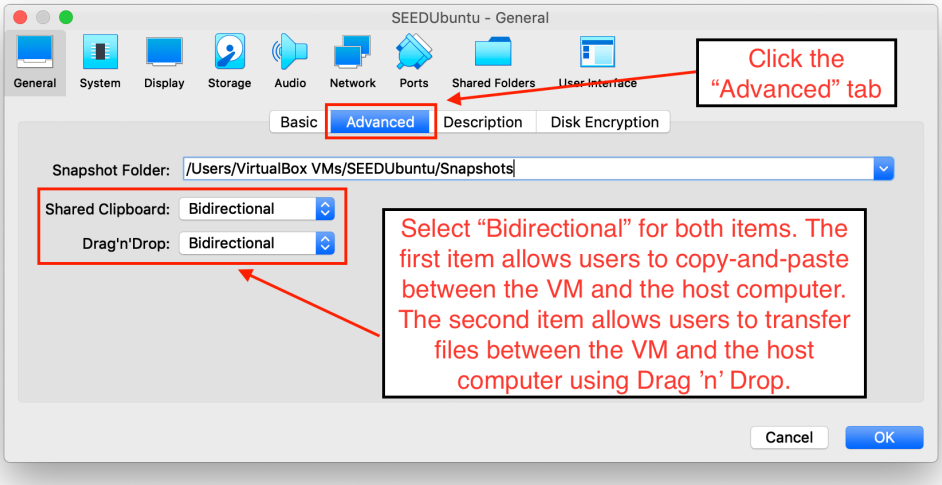
4. **Select the Pre-built Virtual Machine File Downloaded Earlier**

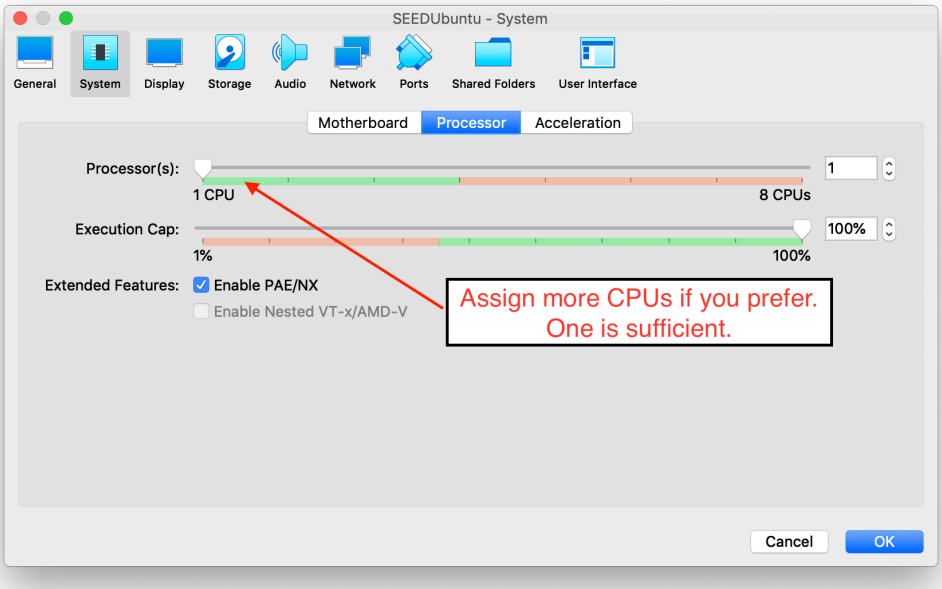


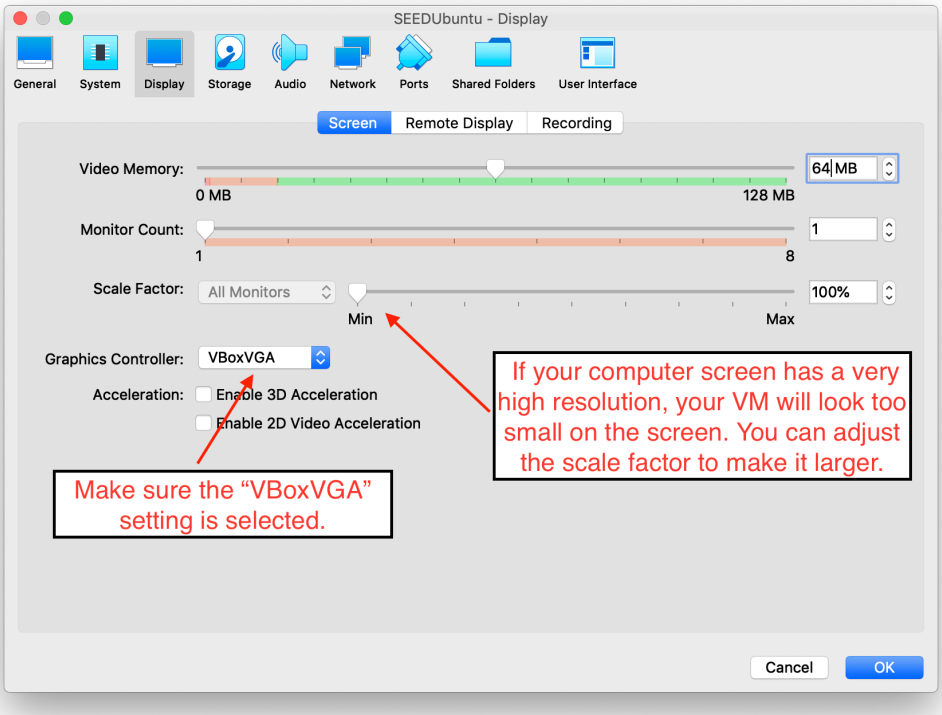


5. Configure the Virtual Machine

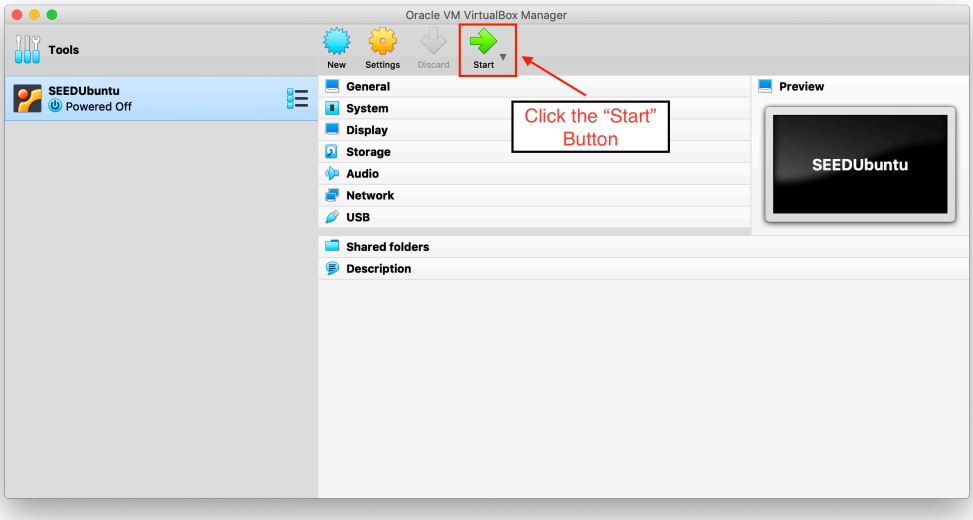




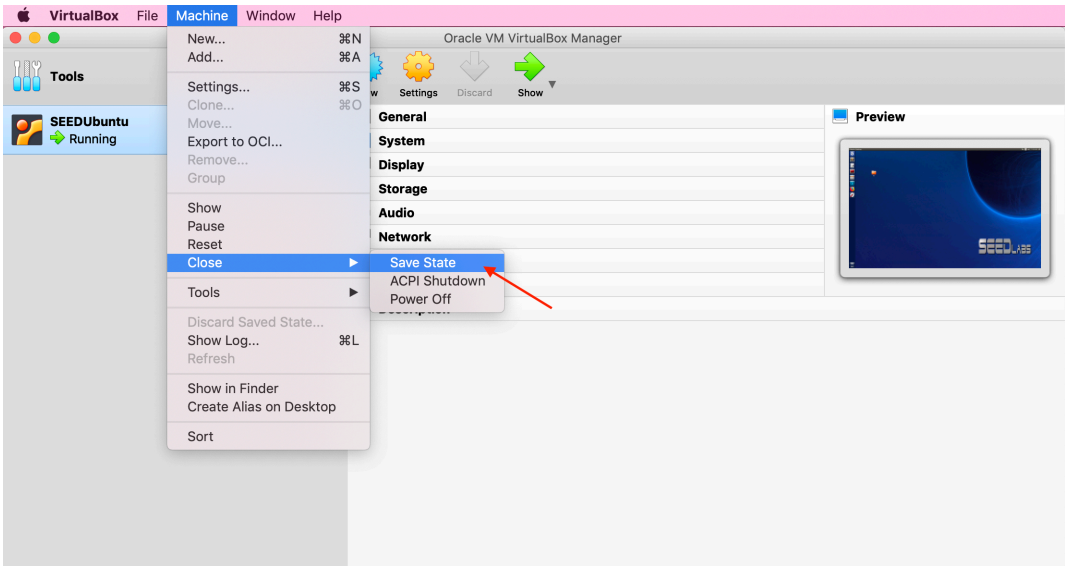

SEEDUbuntu - System
General
System
Display
Storage
Audio
Network
Ports
Shared Folders
User Interface
Motherboard
Processor
Acceleration
Processor(s):
1
1 CPU
8 CPUs
Execution Cap:
100%
1%
100%
Extended Features:
Enable PAE/NX
Enable Nested VT-x/AMD-V
Assign more CPUs if you prefer. One is sufficient.
Cancel
OK


SEEDUbuntu - Display
General
System
Display
Storage
Audio
Network
Ports
Shared Folders
User Interface
Screen
Remote Display
Recording
Video Memory:
64 MB
0 MB
128 MB
Monitor Count:
1
1
8
Scale Factor:
All Monitors
100%
Min
Max
Graphics Controller:
VBoxVGA
Acceleration:
Enable 3D Acceleration
Enable 2D Video Acceleration
If your computer screen has a very high resolution, your VM will look too small on the screen. You can adjust the scale factor to make it larger.
Make sure the "VBoxVGA" setting is selected.
Cancel
OK

6. **Start the Virtual Machine**



7. **Stop the Virtual Machine or Save the Virtual Machine's State**: When you are done with your virtual machine, you can always shut it down (from inside Ubuntu). A better alternative is to "freeze" the computer, so everything is saved. When you need it again, you can "unfreeze" it, and resume from where you left off. This is much faster and more convenient than shutting down and rebooting the virtual machine. To achieve this, you can use the "Save State" option.



3 User Accounts

The virtual machine has two user accounts. The usernames and passwords are listed below:

1. User ID: `root`, Password: `seedubuntu`.
    - Ubuntu does not allow root to login directly from the login window. You have to login as a normal user, and then use the command `su` to login to the `root` account.
2. User ID: `seed`, Password: `dees`.
    - This account is already given the root privilege, but to use the privilege, you need to use the `sudo` command.

Important Note It is essential that you set up the virtual machine as early as possible to ensure that you have time to address any technical difficulties that you may face. The instructor and the TA will not be able to provide adequate technical support close to the assignment due date.

Part II Symmetric Cryptography

1 Introduction

The most commonly used symmetric encryption algorithms such as DES, 3DES and AES are block ciphers. Block ciphers process plaintext input in fixed-size blocks and produces a block of ciphertext of equal size for each plaintext block. In the case of DES and 3DES, the block length is 64 bits, and for AES, the block size is 128 bits. For longer amounts of plaintext, it is necessary to break the plaintext into blocks (padding the last block if necessary). To apply a block cipher in a variety of applications, five modes of operation have been defined and are intended to cover virtually all the possible applications of encryption for which a block cipher could be used. These modes are intended for use with any symmetric block cipher.

In this part of the assignment, you will get familiar with fundamental concepts of symmetric encryption, including symmetric encryption algorithms and symmetric block cipher modes using `openssl`.

2 Background

The `openssl enc` command can be used to encrypt and decrypt files.

```
$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin -K key -iv intial_vector
```

- `-ciphertype` stands for the cipher and mode to be used. Examples: `-aes-128-cbc`, `-bf-cbc`, `-aes-128-cfb`, etc.
- `plain.txt` is the input file to be encrypted
- `cipher.bin` is the output file containing the ciphertext resulting from the encryption
- `key` is the key used for encryption (in hexadecimal). Example: `00112233445566778899AABBCCDDEEFF`
- `initial_vector` is the initialization vector to be used (in hexadecimal). Example: `0102030405060708`

Some common options for the `openssl enc` command are provided below:

```
-in <file>        input file
-out <file>       output file
-e                encrypt
-d                decrypt
-K/-iv            key/initialization vector in hexadecimal
-[pP]             print the key/initialization vector (then exit -P)
```

You can find the meaning of the command-line options and all of the supported cipher types by typing `man enc` or `man openssl`.

3 Problems and Tasks

Problem 1 [10 points]

Symmetric Encryption using Different Ciphers and Modes: To complete this problem, do the following:

(a) [1 point] **Create a plaintext file**: For example, the contents of the file may be: "Only my friends can read these secrets." Do not forget to describe the file and its contents in your report.

(b) [4 points] **Encrypt the file**: Using the `openssl enc` command, encrypt the file that you created. Do this using at least THREE (3) different cipher types but use the same key and initialization vector. Be sure to clearly state the ciphers and the chosen modes that you have selected.

(c) [5 points] **Verify the output**: Once the files are encrypted, most of the data in the file will not be printable. To observe the contents of the output file, use the command-line hex viewing tool `xxd` as follows:

```
$ xxd cipher.bin
```

Explain your findings and clearly identify what you notice about each of the ciphertexts that are generated.

Problem 2 [10 points]

Encryption Mode: ECB vs. CBC: The file `original.bmp` contains a simple picture and can be downloaded from the assignment resources for this assignment on cuLearn. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. To complete this problem, do the following:

(a) [2 points] **Encrypt the file**: Using the `openssl enc` command as in Problem 1, encrypt `original.bmp` using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes using the AES-128 bit cipher.

(b) [1 point] **Replace the encrypted headers**: We need to treat the encrypted picture as a picture, and use a picture viewing software to display it. However, for the `.bmp` file, the first 54 bytes contain the header information about the picture. This header must be set correctly so the encrypted file can be treated as a legitimate `.bmp` file. We will replace the header of the encrypted picture with that of the original picture. We can use the `bless` hex editor tool (already installed on the virtual machine) to directly modify binary files. We can also use the following commands to get the header from `p1.bmp`, the data from `p2.bmp` (from offset 55 to the end of the file), and then combine the header and data together into a new file as follows:

```
$ head -c 54 p1.bmp > header
$ tail -c +55 p2.bmp > body
$ cat header body > new.bmp
```

(c) [2 points] **View the encrypted images and draw conclusions**: Display the encrypted picture using a picture viewing program; an image viewer program called `eog` is installed on the virtual machine). Can you derive any useful information about the original picture from the encrypted picture? Explain your observations.

(d) [5 points] Select a picture of your choice, repeat Parts (a)-(c) and report your observations.

HINT: This problem will be much easier if you select a `.bmp` picture!

Part III Asymmetric Cryptography

1 Introduction

RSA (Rivest Shamir Adleman) is one of the first public-key cryptosystems and is widely used for secure communication. The RSA algorithm first generates two large random prime numbers, and then use them to generate public and private key pairs, which can be used to do encryption, decryption, digital signature generation, and digital signature verification. The RSA algorithm is built upon number theories, and it can be quite easily implemented with the support of libraries.

In this part of the assignment, you will gain hands-on experience with asymmetric cryptography and the RSA algorithm by generating public/private keys, performing encryption/decryption, and signature generation/verification. Essentially, you will be implementing the RSA algorithm using the C program language.

2 Background

The RSA algorithm involves computations on large numbers. These computations cannot be directly conducted using simple arithmetic operators in programs, because those operators can only operate on primitive data types, such as 32-bit integer and 64-bit long integer types. The numbers involved in the RSA algorithms are typically more than 512 bits long. For example, to multiply two 32-bit integer numbers a and b, we just need to use $(a \times b)$ in our program. However, if they are big numbers, we cannot do that anymore; instead, we need to use an algorithm (i.e., a function) to compute their products.

There are several libraries that can perform arithmetic operations on integers of arbitrary size. In this assignment, we will use the Big Number library provided by `openssl`. To use this library, we will define each big number as a `BIGNUM` type, and then use the APIs provided by the library for various operations, such as addition, multiplication, exponentiation, modular operations, etc.

2.1 BIGNUM APIs

All the big number APIs can be found from https://linux.die.net/man/3/bn. In the following, we describe some of the APIs that are needed for this assignment.

- Some of the library functions require temporary variables. Since dynamic memory allocation to create BIGNUMs is quite expensive when used in conjunction with repeated subroutine calls, a `BN_CTX` structure is created to hold `BIGNUM` temporary variables used by library functions. We need to create such a structure, and pass it to the functions that require it.

```
BN_CTX *ctx = BN_CTX_new();
```

- Initialize a `BIGNUM` variable

```
BIGNUM *a = BN_new();
```

- There are a number of ways to assign a value to a `BIGNUM` variable.

```
// Assign a value from a decimal number string
BN_dec2bn(&a, "12345678901112231223");
// Assign a value from a hex number string
BN_hex2bn(&a, "2A3B4C55FF77889AED3F");
```

```
// Generate a random number of 128 bits
BN_rand(a, 128, 0, 0);
// Generate a random prime number of 128 bits
BN_generate_prime_ex(a, 128, 1, NULL, NULL, NULL);
```

- Print out a big number.

```
void printBN(char *msg, BIGNUM * a) {
    // Convert the BIGNUM to number string
    char * number_str = BN_bn2dec(a);
    // Print out the number string
    printf("%s %s\n", msg, number_str);
    // Free the dynamically allocated memory
    OPENSSL_free(number_str);
}
```

- Compute $res = a - b$ and $res = a + b$:

```
BN_sub(res, a, b);
BN_add(res, a, b);
```

- Compute $res = a \times b$. It should be noted that a `BN_CTX` structure is needed in this API.

```
BN_mul(res, a, b, ctx);
```

- Compute $res = (a \times b) \bmod n$:

```
BN_mod_mul(res, a, b, n, ctx);
```

- Compute $res = a^c \bmod n$:

```
 BN_mod_exp(res, a, c, n, ctx);
```

- Compute modular multiplicative inverse, i.e., given a, find b, such that $(a \times b) \bmod n = 1$.

```
BN_mod_inverse(b, a, n, ctx);
```

2.2 A Complete Example

The following code sample shows a complete example where we initialize three `BIGNUM` variables, a, b, and n. We then compute $(a \times b)$ and $(a^b \bmod n)$.

```
/* bn_sample.c */
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

void printBN(char *msg, BIGNUM * a) {
    // Use BN_bn2hex(a) for hex string
    // Use BN_bn2dec(a) for decimal string
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}
```

```c
int main () {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *a = BN_new();
    BIGNUM *b = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *res = BN_new();

    // Initialize a, b, n
    BN_generate_prime_ex(a, NBITS, 1, NULL, NULL, NULL);
    BN_dec2bn(&b, "273489463796838501848592769467194369268");
    BN_rand(n, NBITS, 0, 0);

    // res = a*b
    BN_mul(res, a, b, ctx);
    printBN("a*b = ", res);

    // res = a^b mod n
    BN_mod_exp(res, a, b, n, ctx);
    printBN("a^b mod n = ", res);
    return 0;
}
```

Compilation

We can use the following command to compile `bn_sample.c`.

```
$ gcc bn_sample.c -lcrypto
```

NOTE: the character after - is the letter ℓ, not the number 1; it tells the compiler to use the `crypto` library.

3 Problems and Tasks

Problem 3 [5 points]

Deriving the Private Key: Let p, q, and e be three prime numbers. Let $n = p \times q$. We will use (e, n) as the public key. Write a C program to calculate the private key d. The hexadecimal values of p, q, and e are listed below.

```
p = CC11522701A01C8C7C0441BAF445D92F
q = B18394BD6917DA79842CFD4AF57637E1
e = C5575
```

It should be noted that although p and q used in this problem 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 ones used here are only 128 bits).

Problem 4 [10 points]

Encrypting a Message: Let (e, n) be the public key. Write a C program to encrypt the message "`Here is the new secret intel!`" (the quotations are not included). We need to convert this ASCII string to a hexadecimal string, and then convert the hexadecimal 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 hexadecimal string.

```
$ python -c 'print("My ASCII string.".encode("hex"))'
4D79204153434949920737472696E672E
```

The public keys are listed below in hexadecimal. We also provide the private key d to help you verify your encryption result. Be sure to explain how you verified the result.

```
n = 20856D1A2A1DAF9AD883B59AC6F9C3029C80CC75F91AAC66B9A8E626C0BFC60F
e = 0E755
M = Here is the new secret intel!
d = 0F99F1AA2A46C1E2CBDAB4AEF23688F7E6FB4CAB543709A30DA38E899A1D29BD
```

Problem 5 [5 points]

Decrypting a Message: The public/private keys used in this problem are the same as the ones used in Problem 4. Write a C program to decrypt the following ciphertext C.

```
C = 076D7C8038F80C7B2F6FBF54B6F4C81D59923ED147CEF45DB5E61EFED9C6C197
```

You will need to convert the result back to a plain ASCII string. The following `python` command can be used to convert a hexadecimal string back to to a plain ASCII string.

```
$ python -c 'print("4D79204153434949920737472696E672E".decode("hex"))'
My ASCII string.
```

Problem 6 [10 points]

Signing a Message: The public/private keys used in this problem are the same as the ones used in Problem 4.

(a) [5 points] Write a C program to generate a signature for the following message:

```
M = Authorize $1000 withdraw.
```

You should directly sign this message, instead of signing its hash value.

(b) [5 points] Make a slight change to the message M, such as changing \$1000 to \$2000, and sign the modified message. Compare both signatures and describe what you observe.

Problem 7 [10 points]

Verifying a Signature: Bob receives a message $M =$ "`Freeze all client accounts.`" from Alice, with her signature S. We know that Alice's public key is (e, n).

(a) [5 points] Write a C program to verify whether or not the signature is indeed Alice's. The public key and signature (in hexadecimal) are listed below:

```
M = Freeze all client accounts.
S = 1BE03821BBB463A9D47DE59FAF51E8DDA90310B65E2F521065FF2E3F6009B4B0
e = 0E755
n = 20856D1A2A1DAF9AD883B59AC6F9C3029C80CC75F91AAC66B9A8E626C0BFC60F
```

(b) [5 points] Suppose that the signature is corrupted, such that the last byte of the signature changes from `B0` to `B1`, i.e, there is only one bit of change. Repeat Part (a) of this problem and describe what happens to the verification process.

END OF ASSIGNMENT