Lab 1: Basic Cryptography - AES, RSA and Kyber

Due on Sept. 27^{th} (F), 2024 Noon on Github

1 Overview

In this Lab, you will learn to use AES, RSA, and CRYSTALS-Kyber for secure network communications. The first two cryptographic algorithms are used in our daily life (e.g. HTTPS protocol), and the third is a new post-quantum-cryptography algorithm just adopted as standard. You will need to analyze the performance of software implementations of the algorithms, and then you will use them to set up secure communication between yourself (client) and our server to obtain a secret.

We recommend you to start working on this lab as soon as possible, especially if you are not familiar with working in Linux or Unix like environment. If you do not have a Linux machine, you could use the COE Linux server(gateway.coe.neu.edu) for lab modules 1 and 2. However, module 3 cannot run on the COE server and you have to get your own Linux machine ready for it.

2 Accept your Assignment on Github

The class adopts the GitHub Classroom platform for code development and submission. For each lab assignment, we make a repository for the necessary files.

For this lab, please click the following link to accept assignment: `https://classroom.github.com/a/g8NGDFPn` (*Note that you should have a Github account with your Northeastern Email Address, so that it is convenient for the TA to grade your submissions.*). Clone the assignment repository to your computer and you will have a root directory for assignment 1. If you are new to git commands, you can find a Git tutorial at `https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners`: For this assignment, start from Step 2.

3 Using OpenSSL Library

OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used in Internet web servers, serving a majority of all web sites. You will need to checkout the repository from `https://github.com/openssl/openssl.git` to your local computer and compile. However, as OpenSSL library is big, you should NEVER put it under the root directory for assignment 1 for submission (if you are pushing the entire directory to the GitHub repository for submission).
To checkout the repository:

```
$ git clone https://github.com/openssl/openssl.git
```

To compile it on your machine:

```
$ cd openssl
$ ./config
$ make
```

Once it is compiled, you will see a static library *libcrypto.a* in the OpenSSL directory. This library will be statically linked against your executable if you are going to use its RSA and AES implementations.

4 Using PQClean Library

Because CRYSTALS-Kyber is very new, it is not a part of OpenSSL's standard release yet, so we will be using code from an open-source repository for the source code, PQClean, from which many other implementations are forked. We focus on Kyber512 (the KEM algorithm) in this lab. The necessary Kyber 512 files will be provided in your assignment, but you can check out the full repository from `https://github.com/PQClean/PQClean.git` if you'd like to view other PQC algorithms. Again, you should NEVER put this repository under the root directory for assignment 1 (if you are pushing the entire directory to the GitHub repository for submission).
To compile the kyber512 files on your machine:

```
$ cd kyber512/
$ make
```

Once it is compiled, you will see a static library *libkyber512_clean.a* in the kyber512 directory.

5 Lab Module 1: Performance of Ciphers (30 pt)

The OpenSSL library has the following APIs for RSA:

1. RSA_public_encrypt
2. RSA_public_decrypt
3. RSA_private_decrypt
4. RSA_private_encrypt

and for AES:

1. AES_encrypt
2. AES_decrypt

The PQClean library has the following APIs for Kyber:

1. PQCLEAN_KYBER512_CLEAN_crypto_kem_keypair
2. PQCLEAN_KYBER512_CLEAN_crypto_kem_enc
3. PQCLEAN_KYBER512_CLEAN_crypto_kem_dec

First understand how to use these ciphers, and write your programs to measure the performance of *RSA_public_encrypt*, *AES_encrypt*, and *PQCLEAN_KYBER512_CLEAN_crypto_kem_enc* functions. Use these functions to encrypt data and time the functions. Collect one million timing samples for each function. You will need to look up the appropriate timer and also the parameters for AES_encrypt(), RSA_public_encrypt(), and PQCLEAN_KYBER512_CLEAN_crypto_kem_enc.

The pseudo-code for measuring encryptions is:

```
for (i = 0; i < 1000000; i++){
  t1 = timer_start()
  AES_encrypt() | RSA_public_encrypt() | PQCLEAN_KYBER512_CLEAN_crypto_kem_enc() // test each
      cipher separately
  t2 = timer_stop()
  timearray[i] = t2 - t1
}
plot_distribution(timearray);
```

To allow your programs to use these cryptographic functions included in the OpenSSL library, you can refer to the following Makefile example for header file inclusion and static library linking. Note to update **OPENSSL** setting to the directory of your downloaded OpenSSL:

```
CC=gcc
OPENSSL=../../openssl /* This line sets the directory of the OpenSSL you have downloaded and
    installed, relative to your current directory*/
INCLUDE=$(OPENSSL)/include/
KYBER=./kyber512
KEXTRAS=$(KYBER)/fips202.c $(KYBER)/randombytes.c

all: program program_kyber //assume you have aes_test.c, rsa_test.c, and kyber_test.c for program.c

program: program.c. // for RSA and AES testing
  $(CC) program.c -I$(INCLUDE) -L$(OPENSSL) -o program $(OPENSSL)/libcrypto.a -ldl

program_kyber: program_kyber.c. // for Kyber testing
  $(CC) program_kyber.c -I$(INCLUDE) -L$(OPENSSL) -L$(KYBER) -o program_kyber
      $(OPENSSL)/libcrypto.a $(KYBER)/libkyber512_clean.a $(KEXTRAS) -ldl -lkyber512_clean

clean:
  rm -rf program program_kyber
```

Plot timing distributions of these samples and find the mean for each function.

Question 1: First compare the performance of RSA, AES-128, and Kyber512 (on the same size of plaintext - 16 bytes). Choose the appropriate key size for RSA to achieve the same security level as AES-128. How much slower is RSA/Kyber than AES? With this implementation cost, discuss what scenarios RSA, AES, and Kyber are mainly used for, respectively. Why would someone want to use Kyber rather than RSA?

6 Lab Module 2: AES Encryption Mode (20 pt)

There are several operation modes of AES. In this section, you will explore two of them: ECB and CBC. Using each of these two modes to encrypt an image file and report your findings. You can experiment with any images, and are provided with two samples in the GitHub repository.

To simplify the experiment, we will be using an image in the PPM format. First, separate the header and body sections of the image:

```
$ head -n 3 penguin.PPM > header.txt
$ tail -n +4 penguin.PPM > body.bin
```

Encrypt the body sections and reconstruct the image using the header and encrypted body sections.

```
$ openssl enc -aes-128-ecb -nosalt -pass pass:"A" -in body.bin -out enc_body.bin
$ cat header.txt enc_body.bin > ecb_penguin.ppm
```

Question 2: Compare these two encrypted images and comment on the security. What is the downside of CBC mode in terms of performance? Suggest one operation mode you think is the best and give your reason.

7 Lab Module 3: Secure Communication (50 pt)

For this module, you can choose to use the traditional public cipher (RSA), or the new PQC standard algorithm Kyber, to establish secure communications with our class server and retrieve a secret message. The class server is *hardware-security.nueess.tk or 10.75.12.66*, with two different services running on different ports. This domain address cannot be accessed from the public Internet. You would need the NEU VPN(click here for instructions). Please reach NEU ITS for support should you have any questions about VPN. The client-server communication is based on socket programming. You can find a good tutorial on socket programming at: `https://www.geeksforgeeks.org/socket-programming-cc/`.

Go to the respective subsection below for the option you choose.

7.1 Option 1: Secure Communication with RSA

You will need to use both AES and RSA to communicate with a service we host on the class server at port *12000* and obtain a 16-byte secret. The following is the communication protocol between the client (you) and the server:

```
server -> [size of server's public key] -> client
server -> [server's public key] -> client
server <- [size of the client's AES key encrypted using server's public key] <- client
server <- [encrypted client's AES key]  <- client
server -> [encrypted secret message using client's AES key] -> client
connection close
```

You can test on your own computer first. The server.c is given in the GitHub - you can pick your own message "secret", and debug/test with your own client program.

7.2 Option 2: Quantum Secure Communication with Kyber

You can also use AES and Kyber to communicate with a service we host on the class server at port *13000* and obtain a 16-byte secret. (Note: Kyber is designed to encapsulate and decapsulate messages and establish a shared 32-byte/256-bit secret, which can be used as the user key to generate the set of 16-byte round keys for AES when you specify AES_set_decrypt_key(aes_key, 128, expanded). You can find the full protocol description at: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf`). The following is the communication protocol:

```
server <- [size of client's Kyber public key] <- client
server <- [client's Kyber public key] <- client
server -> [size of the server's ciphertext encrypted using client's public key] -> client
server -> [encrypted server's ciphertext] -> client  (client then decapsulates)
server -> [encrypted secret message using the shared secret key with AES encryption] -> client
connection close
```

You can test on your own computer first. The server_kyber.c is given in the GitHub - you can pick your own message "secret", and debug/test with your own client program.

8 What You Need to Turn In

When you accept the assignment and clone the Lab 1, you are provided with *handout* folder that includes: a sample timer.c for Module 1, penguin.PPM and puppy.PPM for Module 2, and a sample server.c file for Module 3. You are also given a hints file for the flow and an example Makefile.

For Lab 1 assignment, you will write programs for Module 1 to use AES, RSA, and Kyber for encryption and evaluate their performance; generate two different encrypted images under different AES modes for the given penguin.ppm and understand the pro and con of each mode; write your client program for Module 3 to communicate with the service running on the lab server, compile and run it and retrieve (decrypt) the secret message.

For the lab submission, you will need to put two things under your *submission* directory and push them to our Github Classroom repository. One is a PDF writeup which contains:

1. Plots showing the timing distribution for AES, RSA, and Kyber functions. Your Makefile should contain a target *p1*. When we run the command *make p1*, it should compile and run your program, which generates *aes.txt*, *rsa.txt*, and *kyber.txt*. Each text file should contain timing samples you used to generate your submitted plots.

2. Answers to **Question 1** and **Question 2**.

3. The secret messages in characters. Your Makefile should contain one target *p3* (choose to use RSA or Kyber though). When we run the command *make p3*, it should compile and run you program, which will output the text message to a file *secret.txt*. Note that what the client receives from the server will be 16 bytes (after decryption). Look up the ASCII codes to convert each byte to a character for the message.

The other thing to submit is a folder (named *codes*) which should include all your code files, Makefile, and a README (how to compile and run your programs). Please commit and push these two things to your Github repository. Please be mindful of the submission deadline - the TA will grade the last commit (with the time stamp).

Please do NOT push OpenSSL library and PQCLean to Github, because they are huge libraries and will clutter the server. The TA will test your code on his local computer with his own OpenSSL and PQCLEAN library.

9 Extra credit (20 pts)

You get extra credits of 20 points for doing the other option of Module 3.

10 Resources

Your best friend, when working with programming, software, and security, is Google. The following resource links may be useful in addition to the ones included in lecture slides.

- Getting started with Git: `https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners`
- Getting started with Linux Commands: `http://www.usm.uni-muenchen.de/people/puls/lessons/intro_general/Linux/Linux_for_beginners.pdf`
- Getting started with C programming: `https://www.programiz.com/c-programming`
- Getting started with Makefile: `http://faculty.chas.uni.edu/~wallingf/teaching/cs4550/support/make/Makefiles.pdf`
- Getting started with Matlab programming: `http://mayankagr.in/images/matlab_tutorial.pdf`
- Getting started with Python programming: `https://www.python.org/about/gettingstarted/`