CSE 565 Fall 2019
Homework 2
Due on September 26, 2019, in class at 9:30am

This exercise will allow you to obtain experience with working with cryptographic libraries. For this purpose, you are to write a program in a programming language of your choice and you can use a cryptographic library or libraries of your choice, but the program should run on the UB CSE Fall 2019 virtual machine image (Ubuntu) which can be downloaded from `https://cse.buffalo.edu/~eblanton/misc/vm/`. The image can be used from any operating system using a suitable VM environment software such as, e.g., VirtualBox.

Once you start the VM, you can run commands from a terminal and directly write your program in the VM (using an editor such as emacs) or transfer your program written elsewhere to the VM.

While you are not constrained in the programming language or cryptographic library choice, this handout provides help with writing and running your program in C, Java, or python, as described later in this document.

## Implementation and execution tasks

Your program is to call different cryptographic functions and measure the speed of different cryptographic operations. For that purpose, it should create or read a small file of size 1KB and a large file of size 10MB. These can be randomly generated data or existing files of any type (of the specified size). The program is to implement the following functionalities:

(a) Create a 128-bit AES key, encrypt and decrypt each of the two files using AES in the CBC mode. AES implementations must be based on hardware implementation of AES, so ensure that your libraries are chosen or configured properly.

(b) Repeat part (a) using AES in the CTR mode.

(c) Repeat part (b) with a 256-bit key.

(d) Compute a hash of each of the files using hash functions SHA-256, SHA-512, and SHA3-256.

(e) Create a 2048-bit RSA key, encrypt and decrypt the files above with PKCS #1 v2 padding (at least v2.0, but v2.2 is preferred if available; it may also be called OAEP).

(f) Repeat part (e) with a 3072-bit key.

(g) Create a 2048-bit DSA key, sign the two files and verify the corresponding signatures. If creating a key takes two parameters, use 224 bits for the exponent sizes. If the hash function algorithm needs to specified separately, use SHA-256.

(h) Repeat part (g) with a 3072-bit DSA key (if the second parameter is required, use 256).

Include simple checking for correctness, namely, that computed ciphertexts decrypt to the original data and that signed messages properly verify.

For each **encryption algorithm**, measure (i) the time it take to generate a new key, (ii) the total time it takes to encrypt each of the two files, (iii) the total time it takes to decrypt each file, and also compute (iv) encryption speed per byte for both files and (v) decryption speed per byte for both files. For the **signature scheme**, measure (i) the key generation time, and (ii) the time to produce a signature for both files, (iii) the time to verify a signature on both of the files,

and compute per-byte time for (iv) signing and (v) signature verification for both files. Finally, for the **hash functions**, measure the total time to compute the hash of both files and compute per-byte timings. Make sure that you only measure the operations as specified and not any other operations (such as, e.g., reading or creating files) and retrieve time using enough precision (e.g., in microseconds or nanoseconds).

### Programming language specific instructions

If you choose to use python, the VM should come with all necessary libraries. Make sure that you use `python3` command to have access to the latest version instead of older `python` (implementation of SHA3 is only available starting from version 3.6).

If you choose to use C, using OpenSSL is a good option. The VM comes with command-line functions, but not programming interfaces. To install the rest of OpenSSL, run the following commands at a shell prompt:

```
sudo apt-get update
sudo apt-get install libssl-dev
```

If you choose to use Java, you will need to install the necessary libraries on the VM. Most of the cryptographic functions come with the standard Java distribution, while Bouncy Castle can be used for SHA3. To install the necessary libraries and programs, run the following commands at a shell prompt:

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk libbcpkix-java
```

### Performance report and submission instructions

Once you run your program and produce the necessary measurements, you will need to turn in your code electronically and also turn in a printed report in class. To turn in your code electronically, pack it in a single ZIP file named `hw2.zip` and use the CSE 565 submit command to turn in your code from any CSE machine (e.g., from timberlake.cse.buffalo.edu) as in

```
submit_cse565 hw2.zip
```

It is important that you verify that running `unzip hw2.zip` on timberlake correctly unpacks all of your source code. If we are unable to unpack your ZIP file, you risk getting no credit for the homework.

To turn in your homework in class, create a report that (i) includes the results of your measurements and computations as instructed above, (ii) lists all source code, (iii) specifies what modifications need to be done to the VM (if any) in order to reproduce your environment and run your program, and (iv) provides your comments about the expected and observed performance with respect to the performance aspects below and any justification of the difference (if any) between the expected and observed performance:

1. how per byte speed changes for different algorithms between small and large files;
2. how encryption and decryption times differ for a given encryption algorithm;
3. how key generation, encryption, and decryption times differ with the increase in the key size;
4. how hashing time differs between the algorithms and with increase of the hash size;
5. how performance of symmetric key encryption (AES), hash functions, and public-key encryption (RSA) compare to each other.