CSE 410/565 SPRING 2021 HOMEWORK 2

Due on March 4, 2021, at 11:59pm

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 Spring 2021 virtual machine image (Ubuntu). The VM image 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 and measure the runtime of 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) 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). This experiment can use a 1MB file for the second file size to reduce the runtime.

(e) Repeat part (d) with a 3072-bit key. This experiment can use a 1MB file for the second file size to reduce the runtime.

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

(g) (CSE 565 only) 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 size. If the hash function algorithm needs to specified separately, use SHA-256.

(h) (CSE 565 only) Repeat part (g) with a 3072-bit DSA key (if the second parameter is required, use 256).

Include simple checking of correctness of your code, namely, that computed ciphertexts decrypt to the original data and that signed messages properly verify. (There is no need to test whether the library functions themselves work correctly.)

Your program will need to measure the following execution times:

1. For each *encryption* experiment (a)–(e), 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.
2. For each *hash function* experiment listed in part (f), measure the total time to compute the hash of both files and compute per-byte timings.
3. (CSE 565 only) For each *signature* experiment , 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.

Make sure that you only measure the operations as specified and not any other operations (such as, e.g., reading or creating files). Also make sure that you retrieve the times using sufficient precision (e.g., in microseconds or nanoseconds); reporting 0 is not valid.

Program interface

Your program needs to be written to produce a single executable that executes all of the experiments above and prints the specified information to standard output for each experiment. It should be non-interactive and not prompt for information such as files to use.

Name your program `cryptotools` and make sure that we can execute it by typing `./cryptotools` in your main submission directory. That is, if the source code requires compilation, include a Makefile that compiles the program by typing `make`. If the program cannot be executed in the above form, create a shell script named `cryptotools` that properly invokes your program.

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

Create a report that you will turn in as your homework. The report needs to include:

- Any special information needed to run your program. This can include any modifications to be done to the VM in order to reproduce your environment and run your program and whether your program requires compilation.

- The timing results that your program measured as specified above.
- Your comments about the expected and observed performance for 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.

Submission instructions

Submit your report as a PDF document via UBlearns. To submit your code, pack all files in a single ZIP file named `hw2.zip` and use the CSE 410 or CSE 565 submit command to turn in your code from any CSE machine (e.g., from timberlake.cse.buffalo.edu) as in

```
submit_cse410 hw2.zip
```

or

```
submit_cse565 hw2.zip
```

depending on the course in which you are enrolled.

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