CSCE4853/5853 Homework 3 (Programming)

Due date: February 26, 2019

Full Grade: 160 pts

I. Task Description

In this assignment, you will implement encrypted communications between two parties, Alice and Bob, and evaluate the performance of AES and RSA. You will also implement authenticated communications between them, and evaluate the performance of HMAC and RSA digital signature. For simplicity, Alice and Bob will be simulated by two programs running on the same computer. When Alice sends a message to Bob, she writes the message to a file. Bob receives the message through reading from the file.

Part 1: Implement encryption and decryption using AES with 128-bit key. Assume that Alice and Bob already have a shared secret key *k* (e.g., they can read the key from the same file). Alice encrypts an 18-byte message *m* (the message is manually input from command line), and writes the ciphetext into a file named *ctext*. Bob reads the ciphertext from the file, decrypts it, and prints the message *m*. The encryption should use the CBC mode.

Part 2: Implement encryption and decryption using RSA with 2048-bit key. Assume that Alice already has got Bob's public key (you need to figure out a way to do this). Alice encrypts an 18-byte message *m* (the message is manually input from command line) using Bob's public key, and writes the ciphetext into a file named *ctext*. Bob reads the ciphertext from the file, decrypts it, and prints the message *m*.

Part 3: Measure the performance of AES and RSA. Take a 7-byte message manually input from command line. Run the AEC encryption over the 7-byte message and decryption of its ciphertext for one hundred times, measure the average time needed for one encryption, and measure the average time needed for one decryption. Run the RSA encryption over the 7-byte message and decryption of its ciphertext for one hundred times, measure the average time needed for one encryption, and measure the average time needed for one decryption. Print the average time of encryption and the average time of decryption for AES and RSA separately.

Part 4: Implement authentication/integrity protection using HMAC with SHA-256 as the underlying hash algorithm. Assume that Alice and Bob already have a 16-byte shared secret key *k* (e.g., they can read the key from the same file). Alice generates the HMAC of an 18-byte message *m* (the message is manually input from command line) using key *k*, and writes the message m as well as the HMAC into a file named *mactext*. Bob reads the message and HMAC from the file, verifies the HMAC, and prints whether the verification succeeds.

Part 5: Implement digital signature using RSA with 2048-bit key. Assume that Bot already has got Alice's public key (you need to figure out a way to do this). Alice signs an 18-byte message *m* (the message is manually input from command line) using her private key to get the signature *s*, and writes the message *m* as well as the signature *s* into a file named *sigtext*. Bob reads the

message and signature from the file, verifies the signature using Alice's public key, and prints whether the verification succeeds.

Part 6: Measure the performance of HMAC and digital signature. Take a 7-byte message *m* manually input from command line. Implement HMAC with SHA-256 as the underlying hash algorithm. Generate the HMAC of the 7-byte message using a 16-byte key for one hundred times, and measure the average time needed for one HMAC generation. Implement RSA with 2048-bit keys. Generate the digital signature of the 7-byte message and verify it for one hundred times, measure the average time needed for signature generation, and the average time needed for signature verification. Print the average time of HMAC generation, signature generation, and signature verification separately.

II. Tests

You need to demo your program to the Grader. A demo sign-up sheet will be distributed in class later. During the demo, your program will be tested in the following ways.

Test of Part 1: For the encryption function, your program needs to take a manually input plaintext message from the command line, and print the derived ciphertext.
For the decryption function, your program needs to print the received ciphertext and the deciphered plaintext.

Test of Part 2: Same as Part 1.

Test of Part 3: Your program needs to take a manually input plaintext message from the command line, and print the average time of encryption and the average time of decryption for AES and RSA.

Test of Part 4: Your program needs to take a manually input plaintext message from the command line, print the derived HMAC, and print whether the verification succeeds. Then a manual change will be made to the *mactext* file (either to the message m or to the HMAC), and print whether the verification succeeds.

Test of Part 5: Your program needs to take a manually input plaintext message from the command line, print the derived RSA signature, and print whether the verification succeeds. Then a manual change will be made to the *sigtext* file (either to the message m or to the signature), and print whether the verification succeeds.

Test of Part 6: Your program needs to take a manually input plaintext message from the command line, and print the average time of HMAC generation, signature generation, and signature verification separately.

III. Other Instructions

Any programming language is fine.

Messages must be manually input from command line (not from a file). Results must be printed to the command line (not to a file).

Submit your code and necessary support files (including an empty *ctext/mactext/sigtext* file and, if any, the file(s) used to exchange shared secret key and public key) to Blackboard as a .zip file named in this format: HW3.YourLastName.YourFirstName.zip.