SIL765: Networks and System Security

Semester II, 2021-2022

Assignment-2

January 24, 2022

Problem: Evaluating Cryptographic Primitives

Background

In this assignment, you will evaluate and compare the computational, communication, and storage cost of the following cryptographic algorithms for encryption and/or authentication. The computational cost refers to the time (in msec) to execute the algorithm, the communication cost refers to the length (in bits) of the packet communicated from the sender to the receiver in the algorithm, and the storage cost refers to the length (in bits) of the key that needs to be securely stored in this algorithm.

1. Setup
   - 1.1. Generate keys for different algorithms
   - 1.2 Generate nonces for different algorithms
2. Encryption
   - 2.1. AES with a 128-bit key in the CBC mode
     - 2.1.1. Encryption: **AES-128-CBC-ENC**
     - 2.1.2. Decryption: **AES-128-CBC-DEC**
   - 2.2. AES with a 128-bit key in the CTR mode
     - 2.2.1. Encryption: **AES-128-CTR-ENC**
     - 2.2.2. Decryption: **AES-128-CTR-DEC**
   - 2.3. RSA with a 1024-bit key (Use a 128-bit symmetric key as the plaintext input)
     - 2.3.1. Encryption: **RSA-2048-ENC**
     - 2.3.2. Decryption: **RSA-2048-DEC**
3. Authentication
   - 3.1. AES with a 128-bit key in the CMAC mode
     - 3.1.1. Authentication Tag (MAC) Generation: **AES-128-CMAC-GEN**
     - 3.1.2. Authentication Tag (MAC) Verification: **AES-128-CMAC-VRF**
   - 3.2. SHA3 with a 256-bit output
     - 3.2.1. Authentication Tag (HMAC) Generation: **SHA3-256-HMAC-GEN**
     - 3.2.2. Authentication Tag (HMAC) Verification: **SHA3-256-HMAC-VRF**
   - 3.3. RSA with a 1024-bit key and SHA3 with a 256-bit output
     - 3.3.1. Authentication Tag(Signature) Generation: **RSA-2048-SHA3-256-SIG-GEN**
     - 3.3.2. Authentication Tag(Signature) Verification: **RSA-2048-SHA3-256-SIG-VRF**

3.4. Elliptic Curve Digital Signature Algorithm (ECDSA) with a 256-bit key and SHA3 with a 256-bit output

3.4.1. Authentication Tag(Signature) Generation: **ECDSA-256-SHA3-256-SIG-GEN**

3.4.2. Authentication Tag(Signature) Verification: **ECDSA-256-SHA3-256-SIG-VRF**

4. Authenticated Encryption

4.1. AES with a 128-bit key in the GCM mode

4.1.1. Encrypt and MAC Generation: **AES-128-GCM-GEN**

4.1.2. Decrypt and MAC Verification: **AES-128-GCM-VRF**

To-Do Tasks

You need to complete three fundamental tasks.

1. You need to prototype the given algorithms.
2. You need to provide the three different costs for each algorithm.
3. You need to discuss the pros and cons of each algorithm in comparison to other relevant algorithms.

Given Files

- **execute_crypto.py** - the file providing the framework for your code.
- **setup_env.sh** - the file providing the framework for adding commands for installing required packages.
- **original_plaintext** - the file containing the plaintext to be encrypted and/or authenticated in different algorithms (except RSA encryption).
- **example.zip** - Illustrative examples of keys, ciphertexts and authentication tags

Expected Submission

- **setup_env.sh** - the file containing the command for installing any required packages. If you are using Python, please follow and update the given setup_env.sh.
- **execute_crypto** (the source code in any appropriate format) containing the functions corresponding to the algorithms. If you are using Python, please follow and update the given execute_crypto.py.
- **example_test** (the source code in any appropriate format) - the file containing the code to call the functions given in "execute_crypto".
- **original_plaintext** - the file containing the plaintext to be encrypted and/or authenticated.
- **readme.pdf** : This should contain the information about the used libraries and about your code. You can provide your results in a table similar to Table 1. You should discuss your findings. You can also add bar plots for highlighting specific results.
- You can also share any other relevant files. For instance,
  - **Makefile** (the make file to cleanly execute your source code)

Algorithm	Key Length	Execution Time	Packet Length
AES-128-CBC-ENC			
AES-128-CBC-DEC			
AES-128-CTR-ENC			
AES-128-CTR-DEC			
RSA-2048-ENC			
RSA-2048-DEC			
AES-128-CMAC-GEN			
AES-128-CMAC-VRF			
SHA3-256-HMAC-GEN			
SHA3-256-HMAC-VRF			
RSA-2048-SHA3-256-SIG-GEN			
RSA-2048-SHA3-256-SIG-VRF			
ECDSA-256-SHA3-256-SIG-GEN			
ECDSA-256-SHA3-256-SIG-VRF			
AES-128-GCM-GEN			
AES-128-GCM-VRF			

Table 1: Illustrative table for presenting results.

Grading

- You will be graded on your performance in the three to-do tasks discussed above.
- If your script is not in python, you need to be present in a scheduled session to demonstrate the functionality of your code.
- If your script is in python, Gradescope has been configured to auto-grade your submission. To facilitate that, please follow the following steps.
  - In the given file "execute_crypto.py", there are two sets of comments "Do not change this" and "Write your script here". Please follow them and add your code.
  - At any time, you can upload your submission on Gradescope to see if your code is passing the tests given there.
  - It is important to understand the directory structure utilized in the Gradescope auto-grading tool. You can check your code by creating a folder "example_autograder"" and placing all sources files (e.g., execute_crypto.py, setup_env.sh, original_plaintext.txt) in the folder "example_autograder/source/submissions". There are two important things to note.
    - You must run your code (e.g., "example_test.py") from the "example_autograder" folder.
    - The path to any file must be relative to the "example_autograder" folder.
  - For more help in setting up your folder structure, you can refer to the "example_autograder" provided in the Assignment-1 of this course.