Assignment 2

Goal

The goal in the second assignment is to familiarize students with the use of cryptographic libraries. In particular, the objectives of this assignment are:

- Successfully use a variety of crypto suites from the standard Java crypto libraries
- Gain appreciation for the practical challenges of using cryptography in real systems
- Analyze the relative performance of different crypto schemes

This is an assignment for **groups of two**. Discussions among students in different groups should remain at a high level, and no code should be shared outside of your group. If you have any questions, review the collaboration policy in the syllabus, and if you are still unclear ask a question on Piazza.

You should keep your code private and only share it with instructional staff. If you use a repository service like GitHub or Bitbucket, your repositories should be private. Posting solutions publicly will be considered a violation of academic integrity.

Description

Due: October 2, 2017
Points: 100
In this homework you will compare the performance tradeoffs of various encryption/decryption/hashing algorithms using the javax.crypto libraries provided by the Java. Details of this assignment are given below.

- You will write a single program, *cryptotest*, in Java and use the java.security and javax.crypto libraries.
- You may structure your class files as you see fit.
- You must provide a working Makefile to build the program with make.
- The program *cryptotest* must take as a command line input a filename that will be used as input to various cryptographic functions.

- Cryptotest takes an input file of arbitrary size and performs the following types of operations on it:
    - AES-128 CTR Mode
    - AES-256 CTR Mode
    - Encrypt with RSA-1024
    - Encrypt with RSA-4096
    - HMAC MD5
    - HMAC SHA1
    - HMAC SHA256
    - Digital Signature with RSA-4096 and SHA-256
- The goal of this assignment is to time the performance of these actions on a large input file. In your tests, you should use a 100MB input file. (One will be provided for you in the VCL image for your convenience). Note that the TA's will test your code with a variety of file sizes, so *you should not hardcode file sizes*. During development, you may find it easier to test on small files.

High Level:

Keys for symmetric encryption must be generated by using the "SecureRandom" class with "NativePRNG."

You will need to efficiently read in the file. There are a lot of ways to do this, but we used the 'BufferedInputStream' class. If you are clever, you should only need to read the input file once.

All times reported should be in seconds.

Encryption

For the AES ciphers, you will encrypt the entire file and then decrypt it 100 times. For the AES ciphers, you will encrypt the entire file and then decrypt it **15 times.** You will time each of these operations, and then present the mean and median times for each cipher (separately for encryption and decryption). You must generate a new key for each iteration, but timing the key generation is unnecessary.

Note that RSA does not have a block mode, so you should implement ECB for the purposes of this assignment. (This is OK as the point of the exercise is to evaluate the computation time, not to actually encrypt anything.)

You do not need to save the ciphertexts to disk.

In your report, you must show a screenshot showing all means and medians.

HMAC

For the three hash algorithms, you will HMAC the entire file. As with encryption, you will time 100 iterations of each algorithm and report the mean and median times for each run. In your report, you must show a screenshot showing all means and medians.

Use a 256-bit key for each HMAC. Do not time the key generation.

The HMAC size should be 32 bytes. You do not need to print the HMAC or save it to disk.

Signatures

You will then generate a digital signature (using SHA-256 and RSA-4096) for your large file. Key generation time should not be reported.
You should provide mean and median times for 100 signatures and 100 verifications.

For the signature, you can generate just a single key. For the signature key, you will need to create the keypair folder in the current folder including the public and private key using RSA 4096. You will not need to turn this in.

Report

You will write a report in which you present screenshots of your results. The report must be written using LaTeX. All group members should put their name and email on top of the first page. Screenshots and their explanation should be well organized and understandable. See the rubric for the required items in the report.

Submission

To turn in, you will create a tar file named < *lastname* >*-assign2.tgz* which contains a single directory < *lastname* >*-assign2* then upload it to Moodle.

This .tgz file must contain the Makefile, all Java source, an executable JAR file named *cryptotest.jar*, and your report files in .tex and .pdf.

All source code and the Makefile should be extensively commented.

The instructional staff should be able to run the following commands:

```
tar xvfz < lastname >-assign2.tgz
cd < lastname >-assign2
make
java -jar cryptotest.jar < input file name >
< display results >
```

Logistics

Submission Instructions: Students must turn in their tar file to Moodle by 11:55pm on October 2.

Testing: We have developed a VCL image for you to test your code on, and we will grade your code using this image. If we cannot build and/or run your code successfully on the VCL image, *you will not receive a passing grade.* You should test early and often on VCL.

- The image is named 'CSC574_Fall17_Ubuntu16.04' image, which will be used in our grading.
- It is better to submit an incomplete project that compiles and runs than to have all of the code "working" but not building into a final product.

Group Grading Policy: The responsibility for completion of the assignment falls on both group members jointly, and the grade assigned will be equal for both. If issues arise (interpersonal conflict, a group member disappears), you should contact the instructional staff *well before the deadline.* This is yet another good reason to start early on the assignment.

Friendly Advice: This assignment is deceptively challenging and time consuming. The TAs (with extensive prior programming experience and security courses) spent roughly 40 hours combined doing the initial implementation. You should plan on spending *at least* as much time on this project, so *start early*. Keep in mind: running your experiments will take a significant amount of time just to execute.

Academic Integrity: Like all assignments in this class you are prohibited from copying any content from the Internet or sharing ideas, code, configuration, text or anything else or getting help from anyone in or outside of the class. Consulting online sources is acceptable, but under no circumstances should anything be copied. The exception is short snippets from official documentation *with citation in comments.* Failure to abide by this requirement will result in academic integrity proceedings.

Useful References

- Java cryptography architecture : http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html
- SecureRandom class : https://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html
- Key generator : http://www.java2s.com/Code/Java/Security/Generatea1024bitRSAkeypair.htm
- HMAC MD5/SHA1/SHA256 : http://www.supermind.org/blog/1102/generating-hmac-md5-sha1-sha256-etc-in-java
- Digital Signatures: http://srcrr.com/java/oracle/openjdk/6/reference/sun/security/rsa/RSASignature.SHA256withRSA-source.html
- Create Makefile for Java : https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html
- Create Jar file in Linux and Eclipse :
    - http://www.skylit.com/javamethods/faqs/createjar.html
    - https://www.cs.utexas.edu/~scottm/cs307/handouts/Eclipse%20Help/jarInEclipse.htm
- How to make tar file : https://www.tecmint.com/18-tar-command-examples-in-linux/

-

Rubric

Note: If the Makefile does not work, your maximum grade will be 50%.

If the JAR file does not work, your maximum grade will be 50%.
If the JAR file does not take the input file name in a command line, your grade will be deducted 25%.

AES Ciphers (20 pts)

- Use SecureRandom correctly for key generation (4 points)
- Use correct key size and mode (10 points)
- Present median and mean for both ciphers (6 points)

RSA Ciphers (20 pts)

- Correctly generate keys (4 points)
- Use correct key size and mode (10 points)
- Present median and mean for both ciphers and encryption/decryption (6 points)

Hash (30 pts)

- Use SecureRandom correctly for key generation (3 points)
- Use correct key size (6 points)
- Use correct algorithm (12 points)
- Present median and mean for all algorithms (9 points)

Signature (10 pts)

- Correctly generate keys (2 points)
- Use correct algorithms (4 points)
- Present separate median and mean for signatures and validations (4 points)

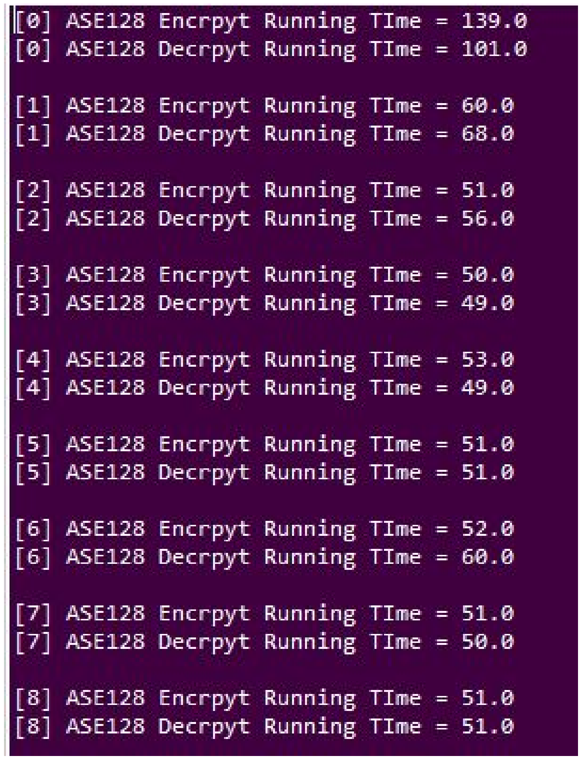
Report (20 pts)

- Include screenshots with clear explanations of means and medians of all tests (8 points total)
- Details of the testing environment (2 points)

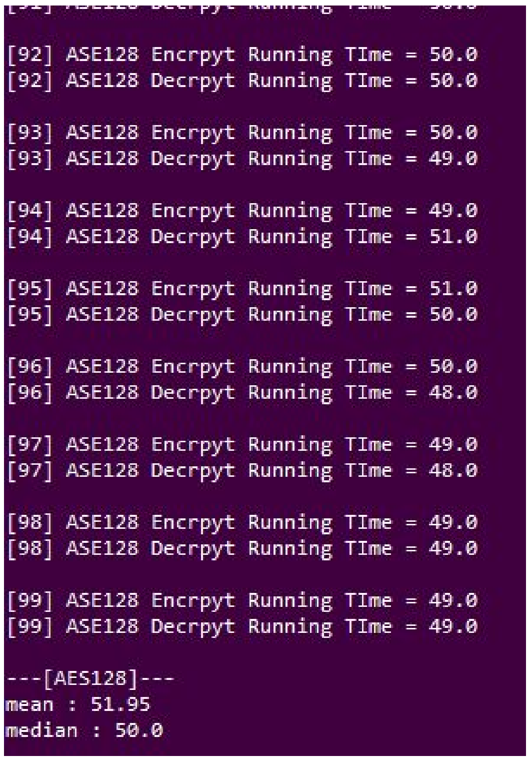
- Discussion of relative performance of all algorithms. Compare each encryption algorithm to each other, each MAC algorithm, and the differences in encryption/decryption and sign/verify. (6 points)
- Produce a table presenting each operation (enc/dec/mac/sign/verify) for each relevant algorithm in a rank order by median time. (2 points)
- Provide .tex and .pdf files. (2 points)

Appendix: Sample Screenshots

- AES128 : input file size **1MB**

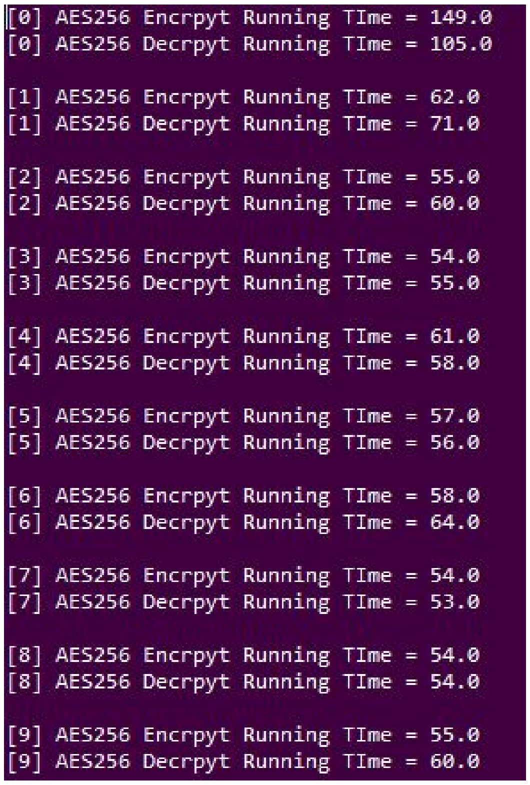


```
[0] ASE128 Encrpyt Running TIme = 139.0
[0] ASE128 Decrpyt Running TIme = 101.0

[1] ASE128 Encrpyt Running TIme = 60.0
[1] ASE128 Decrpyt Running TIme = 68.0

[2] ASE128 Encrpyt Running TIme = 51.0
[2] ASE128 Decrpyt Running TIme = 56.0

[3] ASE128 Encrpyt Running TIme = 50.0
[3] ASE128 Decrpyt Running TIme = 49.0

[4] ASE128 Encrpyt Running TIme = 53.0
[4] ASE128 Decrpyt Running TIme = 49.0

[5] ASE128 Encrpyt Running TIme = 51.0
[5] ASE128 Decrpyt Running TIme = 51.0

[6] ASE128 Encrpyt Running TIme = 52.0
[6] ASE128 Decrpyt Running TIme = 60.0

[7] ASE128 Encrpyt Running TIme = 51.0
[7] ASE128 Decrpyt Running TIme = 50.0

[8] ASE128 Encrpyt Running TIme = 51.0
[8] ASE128 Decrpyt Running TIme = 51.0
```

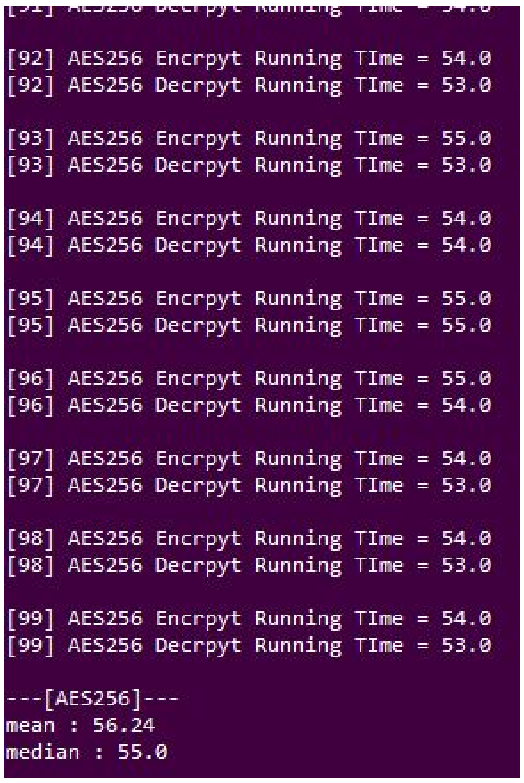


```
[92] ASE128 Encrpyt Running TIme = 50.0
[92] ASE128 Decrpyt Running TIme = 50.0

[93] ASE128 Encrpyt Running TIme = 50.0
[93] ASE128 Decrpyt Running TIme = 49.0

[94] ASE128 Encrpyt Running TIme = 49.0
[94] ASE128 Decrpyt Running TIme = 51.0

[95] ASE128 Encrpyt Running TIme = 51.0
[95] ASE128 Decrpyt Running TIme = 50.0

[96] ASE128 Encrpyt Running TIme = 50.0
[96] ASE128 Decrpyt Running TIme = 48.0

[97] ASE128 Encrpyt Running TIme = 49.0
[97] ASE128 Decrpyt Running TIme = 48.0

[98] ASE128 Encrpyt Running TIme = 49.0
[98] ASE128 Decrpyt Running TIme = 49.0

[99] ASE128 Encrpyt Running TIme = 49.0
[99] ASE128 Decrpyt Running TIme = 49.0

---[AES128]---
mean : 51.95
median : 50.0
```

- AES256 : input file size **1MB**



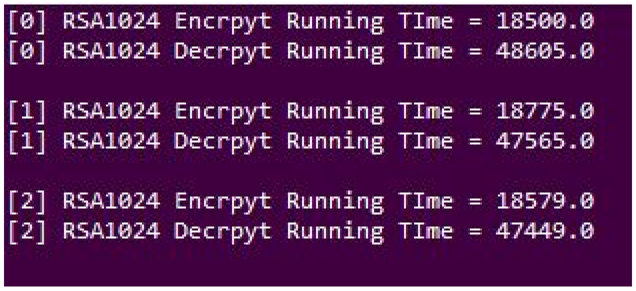
```
[0] AES256 Encrpyt Running TIme = 149.0
[0] AES256 Decrpyt Running TIme = 105.0

[1] AES256 Encrpyt Running TIme = 62.0
[1] AES256 Decrpyt Running TIme = 71.0

[2] AES256 Encrpyt Running TIme = 55.0
[2] AES256 Decrpyt Running TIme = 60.0

[3] AES256 Encrpyt Running TIme = 54.0
[3] AES256 Decrpyt Running TIme = 55.0

[4] AES256 Encrpyt Running TIme = 61.0
[4] AES256 Decrpyt Running TIme = 58.0

[5] AES256 Encrpyt Running TIme = 57.0
[5] AES256 Decrpyt Running TIme = 56.0

[6] AES256 Encrpyt Running TIme = 58.0
[6] AES256 Decrpyt Running TIme = 64.0

[7] AES256 Encrpyt Running TIme = 54.0
[7] AES256 Decrpyt Running TIme = 53.0

[8] AES256 Encrpyt Running TIme = 54.0
[8] AES256 Decrpyt Running TIme = 54.0

[9] AES256 Encrpyt Running TIme = 55.0
[9] AES256 Decrpyt Running TIme = 60.0
```



```
[92] AES256 Encrpyt Running TIme = 54.0
[92] AES256 Decrpyt Running TIme = 53.0

[93] AES256 Encrpyt Running TIme = 55.0
[93] AES256 Decrpyt Running TIme = 53.0

[94] AES256 Encrpyt Running TIme = 54.0
[94] AES256 Decrpyt Running TIme = 54.0

[95] AES256 Encrpyt Running TIme = 55.0
[95] AES256 Decrpyt Running TIme = 55.0

[96] AES256 Encrpyt Running TIme = 55.0
[96] AES256 Decrpyt Running TIme = 54.0

[97] AES256 Encrpyt Running TIme = 54.0
[97] AES256 Decrpyt Running TIme = 53.0

[98] AES256 Encrpyt Running TIme = 54.0
[98] AES256 Decrpyt Running TIme = 53.0

[99] AES256 Encrpyt Running TIme = 54.0
[99] AES256 Decrpyt Running TIme = 53.0

---[AES256]---
mean : 56.24
median : 55.0
```

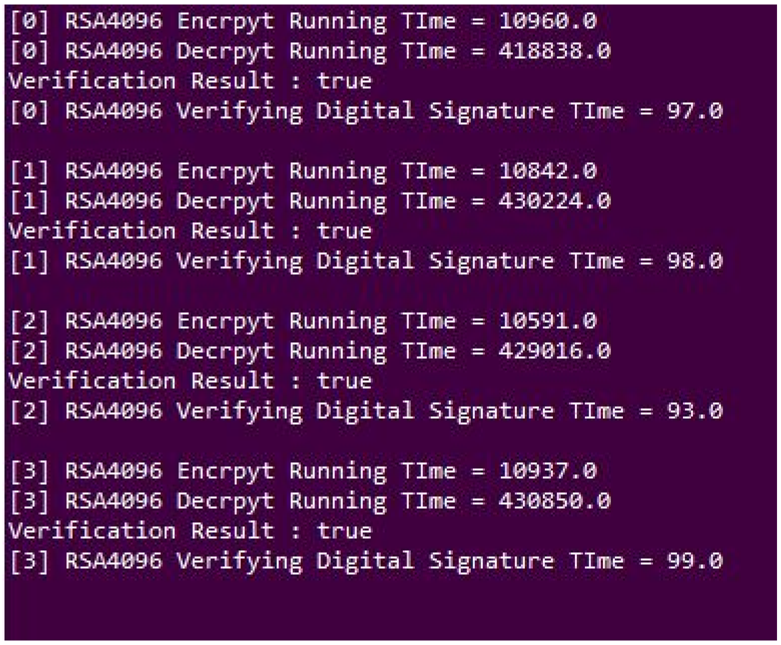
- RSA1024 : input file size **1MB**

* RSA is Block cipher. Plaintext has to be split on block size by key length in order to encrypt/decrypt. So, it would take a long time to encrypt/decrypt by RSA.



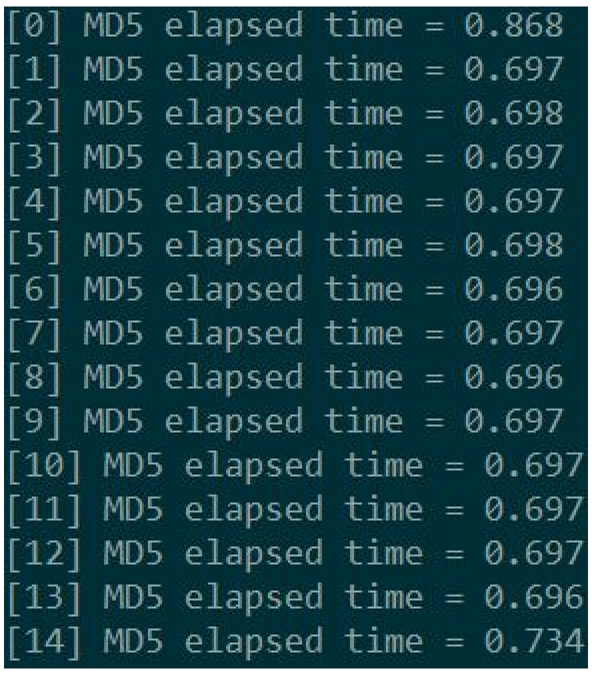
```
[0] RSA1024 Encrpyt Running TIme = 18500.0
[0] RSA1024 Decrpyt Running TIme = 48605.0

[1] RSA1024 Encrpyt Running TIme = 18775.0
[1] RSA1024 Decrpyt Running TIme = 47565.0

[2] RSA1024 Encrpyt Running TIme = 18579.0
[2] RSA1024 Decrpyt Running TIme = 47449.0
```

- RSA4096(with signing and verifying digital signature) : input file size **1MB**

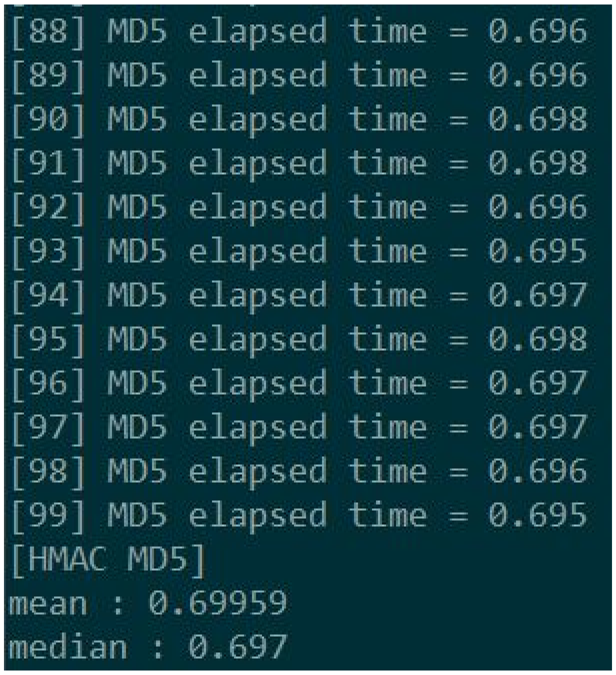


```
[0] RSA4096 Encrpyt Running TIme = 10960.0
[0] RSA4096 Decrpyt Running TIme = 418838.0
Verification Result : true
[0] RSA4096 Verifying Digital Signature TIme = 97.0

[1] RSA4096 Encrpyt Running TIme = 10842.0
[1] RSA4096 Decrpyt Running TIme = 430224.0
Verification Result : true
[1] RSA4096 Verifying Digital Signature TIme = 98.0

[2] RSA4096 Encrpyt Running TIme = 10591.0
[2] RSA4096 Decrpyt Running TIme = 429016.0
Verification Result : true
[2] RSA4096 Verifying Digital Signature TIme = 93.0

[3] RSA4096 Encrpyt Running TIme = 10937.0
[3] RSA4096 Decrpyt Running TIme = 430850.0
Verification Result : true
[3] RSA4096 Verifying Digital Signature TIme = 99.0
```

- HMAC MD5 : input file size **100MB**

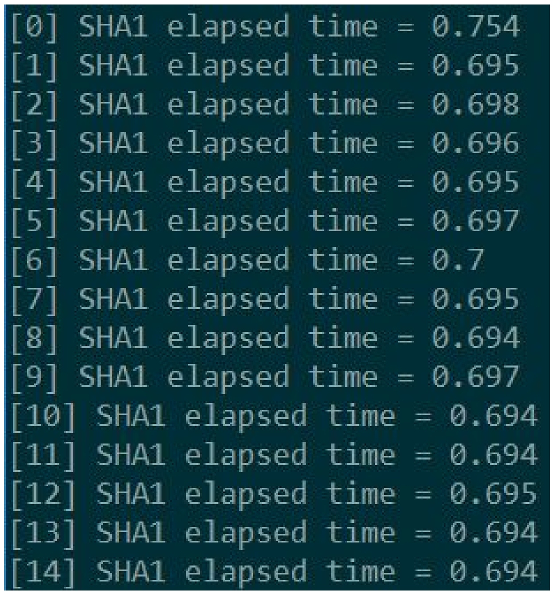


```
[0] MD5 elapsed time = 0.868
[1] MD5 elapsed time = 0.697
[2] MD5 elapsed time = 0.698
[3] MD5 elapsed time = 0.697
[4] MD5 elapsed time = 0.697
[5] MD5 elapsed time = 0.698
[6] MD5 elapsed time = 0.696
[7] MD5 elapsed time = 0.697
[8] MD5 elapsed time = 0.696
[9] MD5 elapsed time = 0.697
[10] MD5 elapsed time = 0.697
[11] MD5 elapsed time = 0.697
[12] MD5 elapsed time = 0.697
[13] MD5 elapsed time = 0.696
[14] MD5 elapsed time = 0.734
```

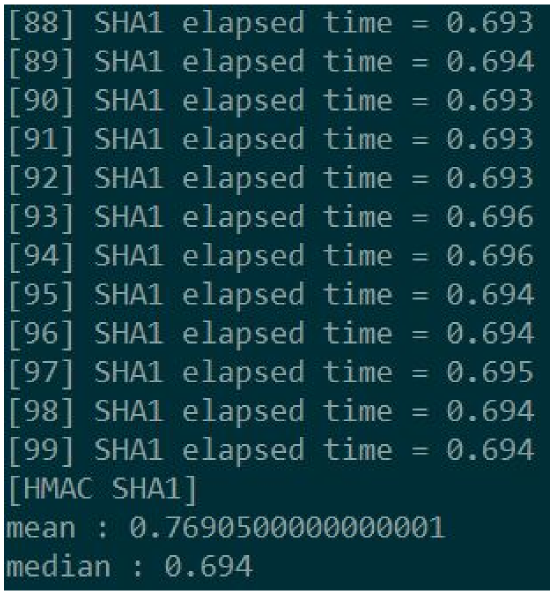


```
[88] MD5 elapsed time = 0.696
[89] MD5 elapsed time = 0.696
[90] MD5 elapsed time = 0.698
[91] MD5 elapsed time = 0.698
[92] MD5 elapsed time = 0.696
[93] MD5 elapsed time = 0.695
[94] MD5 elapsed time = 0.697
[95] MD5 elapsed time = 0.698
[96] MD5 elapsed time = 0.697
[97] MD5 elapsed time = 0.697
[98] MD5 elapsed time = 0.696
[99] MD5 elapsed time = 0.695
[HMAC MD5]
mean : 0.69959
median : 0.697
```

- HMAC SHA1 : input file size **100MB**

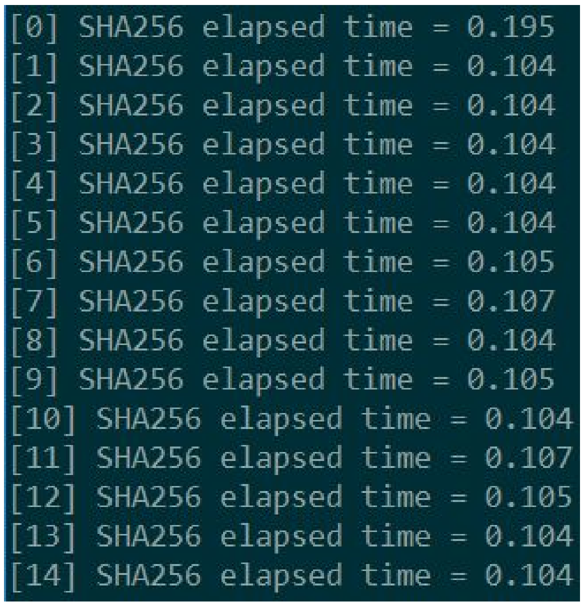


```
[0] SHA1 elapsed time = 0.754
[1] SHA1 elapsed time = 0.695
[2] SHA1 elapsed time = 0.698
[3] SHA1 elapsed time = 0.696
[4] SHA1 elapsed time = 0.695
[5] SHA1 elapsed time = 0.697
[6] SHA1 elapsed time = 0.7
[7] SHA1 elapsed time = 0.695
[8] SHA1 elapsed time = 0.694
[9] SHA1 elapsed time = 0.697
[10] SHA1 elapsed time = 0.694
[11] SHA1 elapsed time = 0.694
[12] SHA1 elapsed time = 0.695
[13] SHA1 elapsed time = 0.694
[14] SHA1 elapsed time = 0.694
```

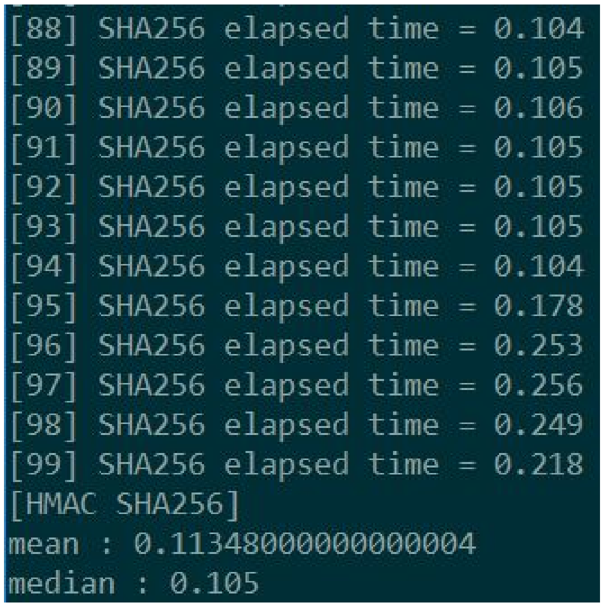


```
[88] SHA1 elapsed time = 0.693
[89] SHA1 elapsed time = 0.694
[90] SHA1 elapsed time = 0.693
[91] SHA1 elapsed time = 0.693
[92] SHA1 elapsed time = 0.693
[93] SHA1 elapsed time = 0.696
[94] SHA1 elapsed time = 0.696
[95] SHA1 elapsed time = 0.694
[96] SHA1 elapsed time = 0.694
[97] SHA1 elapsed time = 0.695
[98] SHA1 elapsed time = 0.694
[99] SHA1 elapsed time = 0.694
[HMAC SHA1]
mean : 0.7690500000000001
median : 0.694
```

- HMAC SHA256 : input file size **10MB**



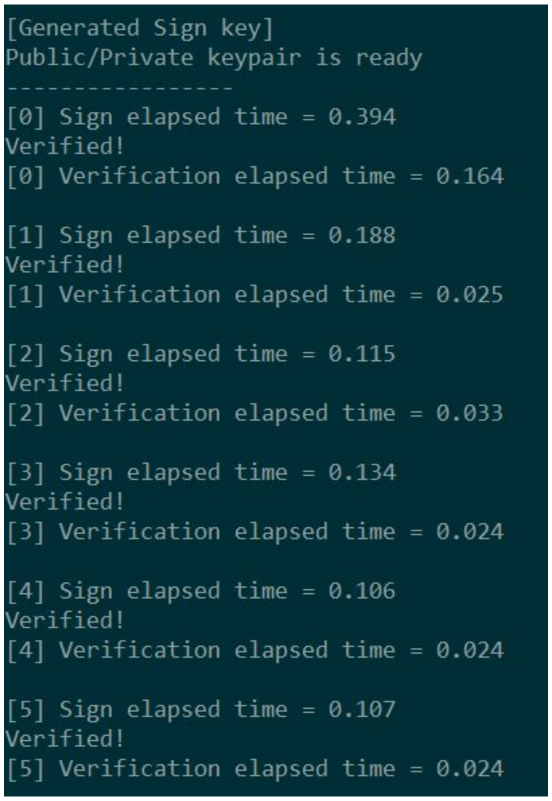
```
[0] SHA256 elapsed time = 0.195
[1] SHA256 elapsed time = 0.104
[2] SHA256 elapsed time = 0.104
[3] SHA256 elapsed time = 0.104
[4] SHA256 elapsed time = 0.104
[5] SHA256 elapsed time = 0.104
[6] SHA256 elapsed time = 0.105
[7] SHA256 elapsed time = 0.107
[8] SHA256 elapsed time = 0.104
[9] SHA256 elapsed time = 0.105
[10] SHA256 elapsed time = 0.104
[11] SHA256 elapsed time = 0.107
[12] SHA256 elapsed time = 0.105
[13] SHA256 elapsed time = 0.104
[14] SHA256 elapsed time = 0.104
```



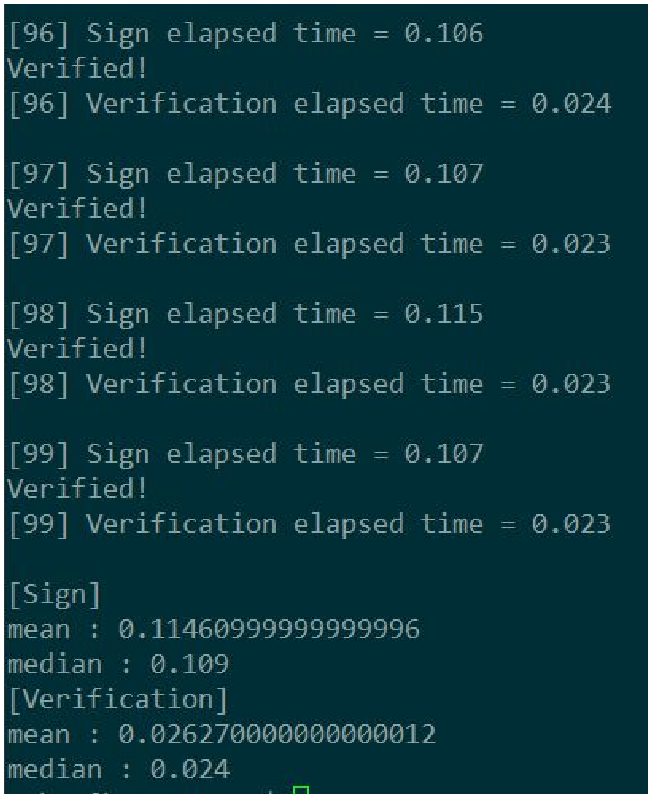
```
[88] SHA256 elapsed time = 0.104
[89] SHA256 elapsed time = 0.105
[90] SHA256 elapsed time = 0.106
[91] SHA256 elapsed time = 0.105
[92] SHA256 elapsed time = 0.105
[93] SHA256 elapsed time = 0.105
[94] SHA256 elapsed time = 0.104
[95] SHA256 elapsed time = 0.178
[96] SHA256 elapsed time = 0.253
[97] SHA256 elapsed time = 0.256
[98] SHA256 elapsed time = 0.249
[99] SHA256 elapsed time = 0.218
[HMAC SHA256]
mean : 0.11348000000000004
median : 0.105
```

- Signature : input file size **16.5MB**

- Time to sign and verification for 100 times



```
[Generated Sign key]
Public/Private keypair is ready
-----------------
[0] Sign elapsed time = 0.394
Verified!
[0] Verification elapsed time = 0.164

[1] Sign elapsed time = 0.188
Verified!
[1] Verification elapsed time = 0.025

[2] Sign elapsed time = 0.115
Verified!
[2] Verification elapsed time = 0.033

[3] Sign elapsed time = 0.134
Verified!
[3] Verification elapsed time = 0.024

[4] Sign elapsed time = 0.106
Verified!
[4] Verification elapsed time = 0.024

[5] Sign elapsed time = 0.107
Verified!
[5] Verification elapsed time = 0.024
```



```
[96] Sign elapsed time = 0.106
Verified!
[96] Verification elapsed time = 0.024

[97] Sign elapsed time = 0.107
Verified!
[97] Verification elapsed time = 0.023

[98] Sign elapsed time = 0.115
Verified!
[98] Verification elapsed time = 0.023

[99] Sign elapsed time = 0.107
Verified!
[99] Verification elapsed time = 0.023

[Sign]
mean : 0.11460999999999996
median : 0.109
[Verification]
mean : 0.026270000000000012
median : 0.024
```