**PROJECT: CRYPTOGRAPHIC ALGORITHMS’ IMPLEMENTATION**

**REPORT:**

***Report on implementation of RSA algorithm:***

* At first, I tried to code in C, but due to int’s limitations couldn’t do well. [rsa\_basic.c]
* Next, I used strings to store the numbers. Had to implement basic arithmetic; addition, subtraction, multiplication and division, all with strings. It did run but was impractical to use due to the bad efficiency. [rsa\_string.c]
* Then, learnt a bit of basic python and coded the RSA in python. It was little better but it gave the below error when used primes greater than 9 digits each. [RSA\_python.txt]

OverflowError: Python int too large to convert to C long

* Unfortunately, I couldn’t get rid of that error.
* The python version is really fast as we use logarithmic calculation time for calculating mod function.
* I’ll use the python scripted code in the trial runs.
* Then, I switched to implementing AES-128 bit.

***Report on implementation of AES-128 algorithm:***

* All thanks to the creator the this PDF: <http://www.moserware.com/assets/stick-figure-guide-to-advanced/A%20Stick%20Figure%20Guide%20to%20the%20Advanced%20Encryption%20Standard%20%28AES%29.pdf> and Wikipedia.
* Implemented in python, this code could handle pretty much everything we throw at it to encrypt and decrypt.
* Note that the performance can be further improved (by a little) by usage of look-up tables for Galois Multiplication for Mix Columns step, but it would make the code bigger, but better nevertheless. (I didn’t implement that, but can be done the same way I used look-up table for s-box and inverse s-boxes)
* I made 3 codes, one being the main code and other two are just derived from the first by commenting most outputs which give insight of what is happening in each round. The code breaks the input as 16 blocks (each has 16 characters). Last file, outputs only the encrypted hex and decrypted text of each block. And also, finally, it gives the encrypted hex and decrypted text of the whole input.
* [AES-128.txt] – Primary code which prints a lot as output (details of every block).
* [AES-128-commented.txt] – Commented version of the previous one.
* [AES-128-ENC-DEC.txt] – Outputs very little (only important outputs). I will use this in the trial runs.

**Some trial runs of the codes: (run on jupyter)**

**RSA:**

***Trial 1:***

***Output:***

p= 86026921 q= 179424697

n= 15435354234267937 N= 15435353968816320 k= 2 d= 2374669841356357 e= 13

Give the message:

Trial run for rsa code(python).

Plain text= Trial run for rsa code(python).

31

The Cipher is :

[84, 114, 105, 97, 108, 32, 114, 117, 110, 32, 102, 111, 114, 32, 114, 115, 97, 32, 99, 111, 100, 101, 40, 112, 121, 116, 104, 111, 110, 41, 46]

The secret message is:

Trial run for rsa code(python).

Total time : 0.038895368576049805

***Trial 2:*** I used a random text copied from a text generator online (1376 words/4328 characters)

The output was really fast.

***Output:*** In short, Total time : 3.6364047527313232

The total output is attached as a different file. [Big\_output\_rsa.txt]

***Trial 3:*** I used the same text 15 times(copied over and over again) (64920 characters)

***Output:*** In short, Total time : 78.39845371246338

The total output is attached as a different file. [Very\_Big\_output\_rsa.txt]

**AES:**

***Trial 1:***

***Output:***

Give the message(no limit for input's number of bits) :

Trial run of aes code(python).

Plain text= Trial run of aes code(python).

length of input/plain text is 30 so you are going to have 2 part(s)

key(128 bit is mandatory) :

1234567891234567

Key matrix= [['1', '5', '9', '4'], ['2', '6', '1', '5'], ['3', '7', '2', '6'], ['4', '8', '3', '7']]

The whole Encrypted text(secret message) in hex is 96 2 c7 d6 87 39 15 d3 2c f 27 64 6 87 d3 e5 f7 ef 74 13 86 9c c1 24 cb 8e 3e df 91 89 83 fc

The whole Decrypted text(secret message) is Trial run of aes code(python)...

Note that if your input is not a multiple of 16 alphabets/numbers final decrypted text will have .'s(dots) instead of unfilled inputs

Total time : 0.03391909599304199

***Trial 2:*** Same input as in rsa trial 2(1376 words/4328 characters)

***Output:*** In short, Total time : 3.0348732471466064

The total output is attached as a different file. [Big\_output\_aes.txt]

***Trial 3:*** Same input as in rsa trial 3(64920 characters)

***Output:*** In short, Total time : 56.3631796836853

The total output is attached as a different file. [Very\_Big\_output\_aes.txt]

**Few observations:**

* Many online compilers were not decoding ascii greater than 128(some did decode).
* UnicodeDecodeError: 'ascii' codec can't decode byte 0xe8 in position 1: ordinal not in range(128).
* The reason why I had to comment parts of AES code is to avoid this ; online compilers were giving an error - output limit reached.
* RSA has been broken, AES is yet to be broken. AES is quite fast compared to RSA. But, RSA is still used for key exchange given that you use really big primes.
* RSA banks on the tough math behind factorisation of large numbers. This page has a graph which can help us to estimate the strength of the RSA algorithm. Input the value of n. <https://www.khanacademy.org/labs/explorations/time-complexity>
* AES banks on the basic principles like diffusion, confusion and secrecy only in the key. The following read puts into perspective the mathematical strength of AES. <https://www.eetimes.com/document.asp?doc_id=1279619>

**Thank you!**