CSE 462

Introduction To Computer Security and Forensics Lab

Assignment 02

Objective:

The goal of this assignment is to implement a hybrid cryptosystem using both symmetric and asymmetric cryptography mechanisms. We will use AES as the symmetric cryptography algorithm and RSA as the asymmetric cryptography algorithm.

Overview of the hybrid cryptosystem

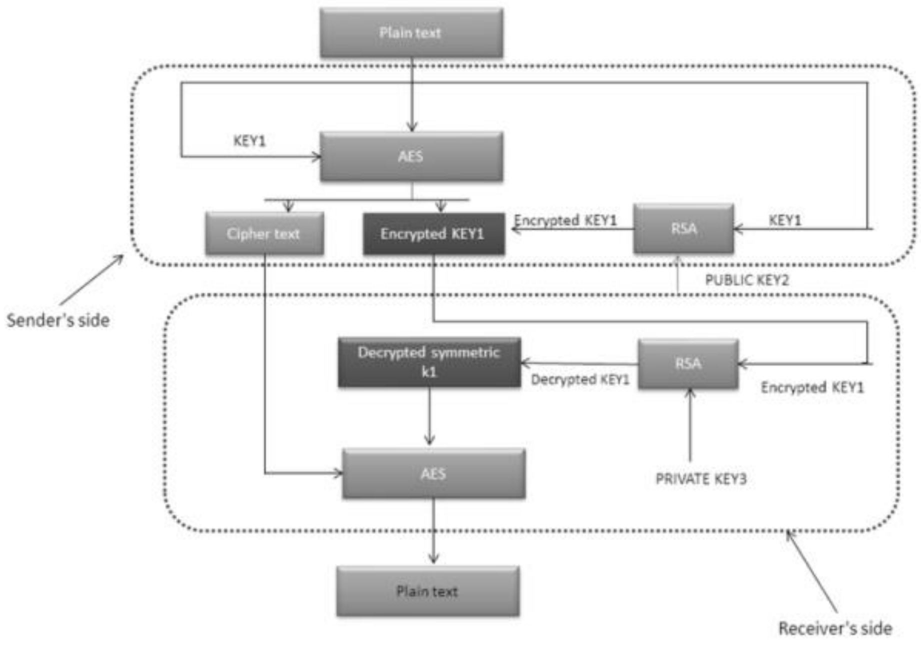


Figure 1: Hybrid Cryptosystem using AES and RSA

As shown in Figure 1, we will at first encrypt the plain text using AES. The key used to encrypt the plain text will be encrypted using RSA. The AES ciphertext and the encrypted AES key will be sent through any communication channel where the key will be at first RSA decrypted to be used for the AES decryption of the ciphertext.

Tasks 01

Independent Implementation of AES

a. ***Key Expansion:*** The Encryption key will be provided by the user as an ASCII string of any length. But the key will always be of length 16. Your program must handle keys of any other size (i.e. discard chars after 16th **or** pad necessary number of chars with it). Choose the *padding char* as you like.

b. ***Encryption:*** The program will encrypt a block of text of 128 bits with the keys generated in the previous step. Do the same for other blocks.

c. ***Decryption:*** Decrypt that encrypted text block and observe if they match with the original text. Do the same for other blocks.

d. Report time-related performance in the code.

N.B.:

❖ A helper file named "***BitVector_Helper.py***" is attached for your convenience. You may import it if you want to. Think of it as a **Plug & Play** type file.

❖ I have made this file for you which will direct you in detail to implement sub+InvSubBytes, Mix+InvMixColumn steps.

Sample I/O:

Sample Input 01:

Key:

In ASCII: Thats my Kung Fu

In HEX: 5468617473206d79204b756e67204675

Plain Text:

In ASCII: Two One Nine Two

In HEX: 54776f204f6e65204e696e652054776f

Cipher Text:

In ASCII:)ÃP_W¶ ö@"³→☻×:

In HEX: 29c3505f571420f6402299b31a2d73a

Decipher Text:

In ASCII: Two One Nine Two

In HEX: 54776f204f6e65204e696e652054776f

Execution Time:

Key Scheduling: 0.00040140000005554408 sec

Encryption Time: 0.22660769999492913 sec

Decryption Time: 0.29179450002266094 sec

Sample Input 02:

Key:

In ASCII: SUST CSE19 Batch

In HEX: 53555354204353453139204261746368

Plain Text:

In ASCII: IsTheirCarnivalSuccessful

In HEX: 497354686569724361726e6976616c537563636573736756c

Cipher Text:

In ASCII: }♣ÄÍ→▲´ÊBÈw∟◄▼3QeP"¦♫

Sã6Đ☻o

In HEX:
7d58e0c4cd1a1eb4ca42c88d771c111f3351655022a6ea53e33682d026f

Decipher Text:

In ASCII: IsTheirCarnivalSuccessful

In HEX: 497354686569724361726e6976616c537563636573736756c

Execution Time:

Key Scheduling: 0.00021090000518597662 sec

Encryption Time: 0.3442144000000553 sec

Decryption Time: 0.5365762000001268 sec

Sample Input 03:

Key:

In ASCII: SUST CSE19 Batch

In HEX: 53555354204353453139204261746368

Plain Text:

In ASCII: YesTheyHaveMadeItAtLast

In HEX: 5965735468657948617665 4d616465497441744c617374

Cipher Text:

In ASCII: §T§w¶X6Á☺Eq♠@YÖ↑ñW▼qûî^½m:Ï

In HEX:
1554157714583 67c11457168f4059d618f1571f8e719bb2fbee5ebd6d3acf

Decipher Text:

In ASCII: YesTheyHaveMadeItAtLast

In HEX: 5965735468657948617665 4d616465497441744c617374

Execution Time:

Key Scheduling: 0.00023350000265054405 sec

Encryption Time: 0.33451689999492373 sec

Decryption Time: 0.5017264999914914 sec

Sample Input 04:

Key:

In ASCII: BUETCSEVSSUSTCSE

In HEX: 42554554435345565353555354435345

Plain Text:

In ASCII: BUETnightfallVsSUSTguessforce

In HEX:
425545546e6967687466616c6c567353555354677565737366f726365

Cipher Text:

In ASCII: 64¡¼é-ÔÌDCÍî¿½2

In HEX:
368d9d9134a1bce994add4cc954443cd8beebfbd98df329dee10b817ecf8f

Decipher Text:

In ASCII: BUETnightfallVsSUSTguessforce

In HEX:
425545546e6967687466616c6c567353555354677565737366f726365

Execution Time:

Key Scheduling: 0.0001728000061120838 sec

Encryption Time: 0.35366069999872707 sec

Decryption Time: 0.5599523999990197 sec

Task 02

Independent Implementation of RSA

a. Your program will at first generate the key pairs (both public and private) with an input of K (Take K = 16, 32, 64, 96). You must **randomly** generate two distinct prime numbers p and q each being $\boldsymbol{K/2}$ bits for given key length K. Choose such algorithms which can generate faster prime numbers.
b. Take plain text as input and encrypt it character by character.
c. Decrypt the ciphertext and match it with the original plain text.
d. Report time-related performance in the following format:

Sample I/O:

Bit Size = 16

Public Key: (e,n)=(7,60491)

Private Key: (d,n)=(17143,60491)

Plain Text:

BUETCSEVSSUSTCSE

Encrypted Text(ASCII):

[53296, 32514, 13868, 59107, 55418, 47636, 13868, 21544, 47636, 47636, 32514, 47636, 59107, 55418, 47636, 13868]

Decrypted Text:

BUETCSEVSSUSTCSE

Execution Time:

Key Generation: 4.220000118948519e-05 sec

Encryption Time: 2.370000584051013e-05 sec

Decryption Time: 3.3899996196851134e-05 sec

Bit Size = 32

Public Key: (e,n)=(11,4292870399)

Private Key: (d,n)=(1560996131,4292870399)

Plain Text:

BUETCSEVSSUSTCSE

Encrypted Text(ASCII):

[1268954923, 3419348944, 3190642909, 1478590601, 1579240218, 2079935148, 3190642909, 1466721226, 2079935148, 2079935148, 3419348944, 2079935148, 1478590601, 1579240218, 2079935148, 3190642909]

Decrypted Text:

BUETCSEVSSUSTCSE

Execution Time:

Key Generation: 6.839999696239829e-05 sec

Encryption Time: 4.909999552182853e-05 sec

Decryption Time: 0.00044709999992918223 sec

Bit Size = 64

Public Key: (e,n)=(3,18446743979220271189)

Private Key: (d,n)=(12297829313753557747,18446743979220271189)

Plain Text:

BUETCSEVSSUSTCSE

Encrypted Text(ASCII):

[287496, 614125, 328509, 592704, 300763, 571787, 328509, 636056, 571787, 571787, 614125, 571787, 592704, 300763, 571787, 328509]

Decrypted Text:

BUETCSEVSSUSTCSE

Execution Time:

Key Generation: 0.05144229999859817 sec

Encryption Time: 3.860000288113952e-05 sec

Decryption Time: 0.0003893999964930117 sec

Bit Size = 96

Public Key: (e,n)=(3,79228162514229434696431832827)

Private Key: (d,n)=(52818775009485914497652274427,79228162514229434696431832827)

Plain Text:

BUETCSEVSSUSTCSE

Encrypted Text(ASCII):

[287496, 614125, 328509, 592704, 300763, 571787, 328509, 636056, 571787, 571787, 614125, 571787, 592704, 300763, 571787, 328509]

Decrypted Text:

BUETCSEVSSUSTCSE

Execution Time:

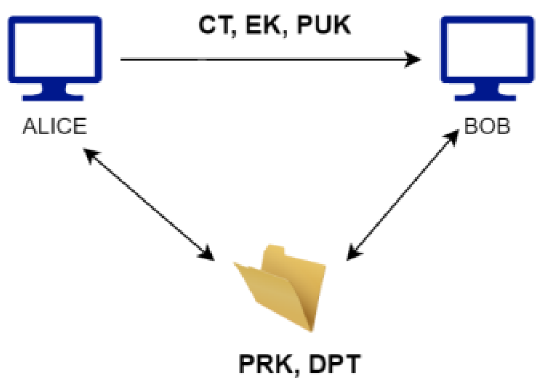
Key Generation: 7.056494499993278 sec

Encryption Time: 1.990000600926578e-05 sec

Decryption Time: 0.0005146999901626259 sec

Task 03

Implementation of the Hybrid Cryptosystem



You need to implement the Hybrid Cryptosystem just as in Figure 1 (1st Page).For that

a. Firstly, encrypt the plaintext using AES algo provided the AES key and then encrypt the key using RSA algo. **N.B.:** After generating private key and public key using RSA, store the private key (**PRK**) inside a folder called "**Don't Open This**".
b. Suppose, ALICE is the sender and BOB is the receiver. ALICE just needs to send the AES encrypted ciphertext (**CT**), the RSA encrypted key (**EK)** to BOB keeping the public key (**PUK**) within herself. So, she will do it by storing them in that secret folder named "**Don't Open This**".
c. BOB will read them from the folder, decrypt the encrypted key using RSA using his private key (can also be found in that same folder), then decrypt the ciphertext using AES decryption using that RSA decrypted key, and then write the decrypted plain text (**DPT**) in the same folder and finally match it with the original plain text.
d. To clear some confusion, you can dedicate two file like alice.py and bob.py to simulate the user's corresponding activities to perform.
e. It would be very much appreciated if you reuse the file from task 1& task 2 for task 3 without readding the same codes for this task.

N.B.: It would be very much realistic if you knew the concept of **"Socket Programming"** (should have been covered from ***Java - Thread Programming*** but what you can do, huh! 😊) Where we could communicate client to client (actually one of the clients is the Server) and run them separately in two different thread and make transactions (send & receive) messages (data).

<u>Some Guidelines:</u>

- ❖ I hope Cryptography Slide can help you with required knowledge to implement these tasks. Moreover, you can contact me for more info.
- ❖ Sample I/O 01 for task 1 will help you cross-validating each subSteps of AES algo which you can find from slide no. 59 to slide 76 of crypto slide.
- ❖ Time mentioned in sample I/O are mere approximations. You can have better efficiency than mine.
- ❖ For task 2, as you will generate p & q randomly. So, my sample I/O's p & q may differ with you. Thus, you can put your own I/O in this regard.
- ❖ You need to submit corresponding source files (py/c/cpp/java etc as there is no restriction over choosing a programming language but python is recommended).
- ❖ Also, you need to submit a report containing required successfully tested sample I/O in each of the tasks and also mentioning the issues you've faced during implement this humongous task.
- ❖ To clear some confusion and be on the same page, possible files I am expecting from all of you, can be like these i.e. aes.py (for task 1), rsa.py (for task 2), alice.py & bob.py (for task 3).
- ❖ You can use library for some data manipulations or very simple functions like int(value,base), hex(val), gcd(val1,val2), perf_counter(). But not the whole algorithm itself.
- ❖ You need to make one single short video covering all the tasks which should contain,
    - o Code walkthrough in brief (what functionalities you've implemented and what they do in high level) in a functional time-flow based chronological order of your algorithm.
    - o Successful Testing of the sample I/O 's I've provided.
    - o You may add your own sample I/O. I won't mind😊.
    - o Mention the issues you've faced during implement this humongous task.
- ❖ Lastly, put all of the files along with video drive link inside a folder. Zip it and upload.
- ❖ Please **DON'T FOREGT TO Hand In in the Google Classroom.**
- ❖ **Do not copy from any web source or friend or seniors. The persons involved in such activities will be penalized by -100% of the total marks and will be marked *<u>*RED*</u>* for future assignments. [FYI, some are penalized and also marked already].**

Mark Distribution:

Task	**Marks**
1	60
2	20
3	20
Total	100

Deadline: 26th of April 2024. Friday 11.59 pm.