Implementation of

RSA-1024

For Securing AES-128 keys

# Introduction

This project is to demonstrate knowledge on how to generate an RSA private/public key pair and use in conjunction with AES-128 to encrypt / decrypt messages. As per project specifications, the AES-128 Engine will be used to encrypt/decrypt plaintext and ciphertext. Whereas the RSA key pairs will be used to “wrap” the AES “red” (unencrypted) keys and turn them “black” (encrypted).

RSA-1024 generates 2 prime numbers which are tested 256 times using the Miller-Rabin Primality tests to minimize false-positives (composite numbers passing off as prime numbers).

RSA-1024 also uses a hard-coded public exponent 65537 – which has been acknowledged to use:

<https://piazza.com/class/lcgjdtutgvz7o9/post/152_f2>

AES-128 ECB has been imported via Python’s Cryptography library. Random 16-byte (128-bits) AES red key is generated when a user chooses the “encrypt” option

# Academic Integrity Disclaimer:

The following algorithm implementations have been pulled and modified from online resources. I have taken the time to acknowledge this within my code under <WARN> <WARN> tags to provide proper references to the resources. Also, comments have been heavily populated to demonstrate understanding.

1. Algorithm for modular inverse wrapper function has been viewed and modified from StackOverFlow. I have decided to implement Extended Euclidean algorithm via non-recursive methods.
   1. <https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python>
2. Extended Euclidean algorithm for calculating modular inverses without recursion has been derived using pseudocode from the Wikipedia:
   1. <https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm>
3. Miller-Rabin primality test has been derived from the Wikipedia pseudocode:
   1. <https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>

# Environment

For my environment, I am running PyCharm 2022.3.2 Professional Edition on Windows 11 Home version 22H2. Python Interpreter 3.7

# Python Packages Required

To run my code, the following need to be installed:

1. Genkeys.py
   1. **Base64** – All RSA/AES Keys/CT/PT have been converted to base64.
   2. **Os** – Random function
   3. **Sys** – Argument parser
   4. **Random** – Random range
2. Crypt.py
   1. **Base64 -** All RSA/AES Keys/CT/PT have been converted to base64.
   2. **Os –** Random Function
   3. **Sys –** Argument Parser
   4. **cryptography.hazmat.primitives.ciphers** – AES-128 Engine
      1. Cipher, algorithms, modes
   5. **cryptography.hazmat.primitives** – AES-128 Padding Scheme (PKCS7)
      1. padding

# Block Diagram

The block diagram below illustrates the interaction and the passage of data between the python scripts

Graphical user interface, diagram, application, Excel

Description automatically generated

# Regular operation

Users should first generate RSA-1024 private/public keys using genkeys.py. Then call crypt.py to encrypt/decrypt plaintext/ciphertext.

1. Genkeys.py alice

Text

Description automatically generated

1. Genkeys.py bob

Text

Description automatically generated

1. Cat message.txt

Text

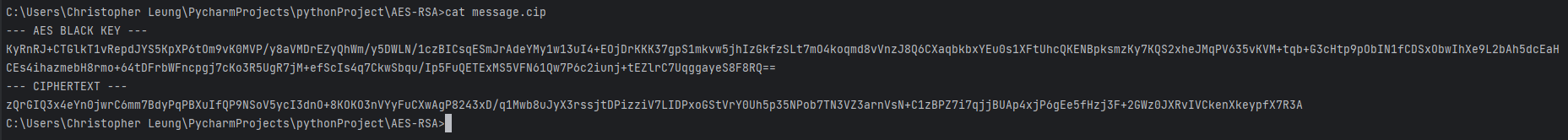
Description automatically generated

1. Crypt.py -e bob.pub message.txt message.cip

Text

Description automatically generated

1. Cat message.cip



1. Crypt.py -d bob.priv message.cip message.txt

Text

Description automatically generated

1. Cat message.txt

Text

Description automatically generated

1. Cat message2.txt

Text

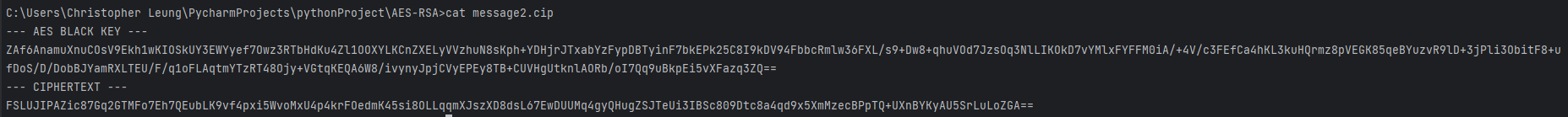
Description automatically generated

1. Crypt.py -e alice.pub message2.txt message2.cip

Text

Description automatically generated

1. Cat message2.cip



1. Crypt.py -d alice.priv message2.cip message2.txt

Text

Description automatically generated

1. Cat message2.txt

Text

Description automatically generated

# Usage

I have generated 2 files for the grader:

1. Genkeys.py
2. Crypt.py

## Genkeys.py Functional Description

### Input:

python genkeys.py <name>

### Output:

<NAME.pub> and <NAME.priv>

### Notice:

As mentioned in the academic integrity header, I have pulled and modified Miller-Rabin primality test, modular inverse wrapper, and extended Euclidean algorithm. I have sourced my references for verification and commented the code to demonstrate understanding. Please search for <WARN><WARN> tags to quickly find these functions.

### Description:

This python script will generate RSA public/private key pairs. This script will name them <NAME.pub> and <NAME>.priv

The output for public (e, n) and private (d, n) key pairs will be in Base64 format.

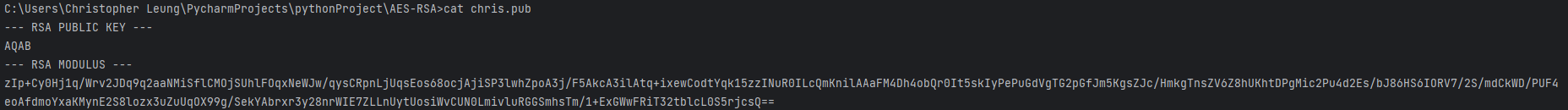
Public exponent has been set to 65537 (2^16 + 1)

**Key Generation:**

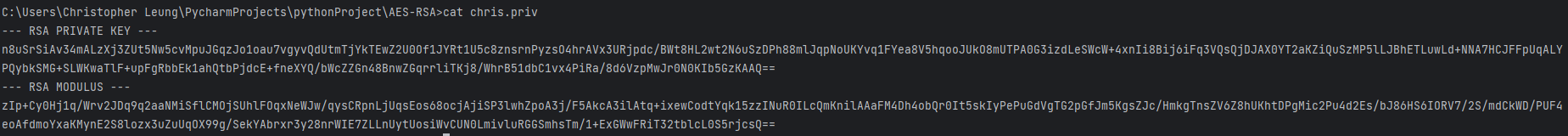
Text

Description automatically generated

**Public Key format:**



**Private Key Format:**



## Crypt.py Functional Description

### Input:

Python crypt.py <-e/-d> <keyFile> <inputFile> <outputFile>

### Output:

Plaintext or Ciphertext depending on -e / -d flag

### Description:

This script will either encrypt or decrypt a message based on the input flag.

AES-128 ECB mode has been used as encrypt/decrypt engine. This has also been imported via Cryptographic library.

**If Encrypt mode is selected:**

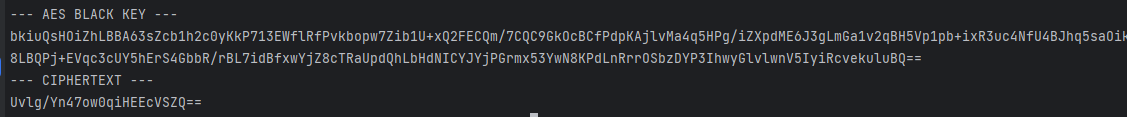
* Generate random 128-bit AES red key used for encrypting message
* Encrypt message
* Encrypt AES red key using RSA and turn it into an AES black key
* Write ciphertext and AES black key to file

**If Decrypt mode is selected:**

* Pull AES Black key and ciphertext from the passed in input file
* Decrypt AES Black key using RSA and turn this into AES red key
* Decrypt ciphertext
* Write Plaintext to file

**File format for ciphertext:**

* “--- AES BLACK KEY ---" header
* AES Black Key (Base64)
* “--- CIPHERTEXT ---" header
* Ciphertext (Base64)



# Resources

Extended Euclidean Algorithm

<https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm>

Miller-Rabin Primality Test

<https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test>

Modular Inverse Wrapper for Python 3.7

<https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python>

RSA Algorithm Overview

<https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>

RSA Example

<https://www.cs.utexas.edu/~mitra/honors/soln.html>