CSI 3680 Final Project

<https://github.com/CSIWill/csi_3680_final>

# RSA Encryption

A data encryption and decryption tool

* e.g., Implement RSA algorithm from scratch, and the RSA modulus must be at least 1024 bits long.
* Randomly generate private key and public key from RSA algorithm, therefore to encrypt and decrypt any file.

# Requirements

* Python based.
  + Written only in Python
* A topic with potential practical usage.
  + Creating RSA private and public keys
* At least define and use TWO functions in a meaningful way.
  + Function for test prime numbers using Miller-Rabin algorithm
  + Function to generate keys
  + Function to encrypt and decrypt files
* At least import ONE third-party library/package to support your implementation.
  + Encrypt PDF file by reading with pypdf
* At least ONE file I/O operation, either read or write or both.
  + Use exceptions to check whether each file was opened without an error.
  + Read content of plaintext and ciphertext files for encryption and decryption
* Visualize your outcomes
  + Plot, image, webpage,GUI … something other than console output!
  + Tkinter to provide GUI

# Introduction/Motivation

RSA encryption will produce a public and private key. These keys can provide confidentiality and authentication. The private key is only known by the owner while the public keys are shared with everyone. The keys can be reversed in order where the encryption/decryption with public/private keys was equivalent to encryption/decryption with private/public keys.

For confidentiality, the user will encrypt with the recipient’s public key. The intended recipient should be able to decrypt with their own private key.

For authentication, the user will encrypt with their own private key. The intended recipient will use the sender’s public key to decrypt. Successful decryption ensures that the user proves their identity.

The keys are pairs and cannot be treated separately. For example, I cannot encrypt and decrypt the same file with only the public key and vice versa.

# Methods

I am using pypdf to read PDF format files. You can encrypt a plaintext pdf file. The text will be extracted into a ciphertext file in a .txt file. Decryption will still be in a .txt file.

The modulus needs to be at least 1024 bits. First, I tried just randint() method from random to generate integer numbers. Then, I shifted to getrandbits() method to get to the required sizes. I used 1024-bit size for the random number generation of p and q. This ensures that the modulus is at least 1024 bits.

The most difficult part in creating the private and public keys was the primality test. First, I tried to use brute force to check that the random numbers generated were prime. That worked initially as I was only testing very small numbers between 1 to 100. Once I started scaling up, especially to a modulus of at least 1024 bits, that was no longer feasible.

My research led to a Stanford resource (Lynn, n.d.) that discussed using the Miller-Rabin Primality test to test large prime numbers. It is an extension of the Fermat Test. The idea is to randomly test multiple smaller numbers, a, instead of a singular value. The variable, a, should be able to satisfy the condition that:

The variable n is the “prime” number we are evaluating. The variable is the largest power of 2 that can divide n-1. The variable q is an odd number that satisfies the previous conditions.

From Stanford’s material and pseudocode from Wikipedia (Wikimedia Foundation, 2023), I was eventually able to develop a function that would determine if a number was most likely prime. I did 100 iterations of this primality test to improve accuracy.

Since the values were so large, it was more effective to use pow(x,y,z) method that it was to manually write out the equivalent .

For key generation, the user can specify the value of public exponent, e used. The default value is 65337. This is commonly used in RSA (Wikimedia Foundation, 2022). If the user wants to randomize e, they can type 0 in the textbox.

# Results

Visualization was achieved with tkinter to provide GUI. There will a be a pop up for each status whether successful or not. Files will be created where main.py is ran from.

# Instructions

Install pypdf with: pip install pypdf

1. Run the Python file, main.py
   1. You will see the following screen:

A screenshot of a computer

Description automatically generated

1. First, input a number for e or leave blank if you would like to use the default value of 65537
   1. Select “Generate Keys”
   2. It will create two files, privateKey.txt and publicKey.txt
      1. Stored in the same folder as main.py
      2. Holds private key and public key respectively
      3. Do not share the private key
2. Encrypt File (simulating sending to another user)
   1. Select the file you want to encrypt to send to another user
      1. “Browse”, under Selected File to Encrypt
   2. Assume that the other user has shared their public key
      1. Select the public key file that they have shared with you
         1. For example, publicKey.txt
      2. “Browse”, under
   3. Select “Submit”
   4. If successful, you will see the following window

A screen shot of a computer

Description automatically generated

* 1. It will be stored in the same folder
     1. “Encrypted – filename.txt”

1. Encrypt File (proving your identity)
   1. Same steps as 3, except, you would select your private key file
2. Decrypt File (Receiving a file from another user sent to you)
   1. Select the file sent by another user that you want to decrypt
      1. “Browse”, under Selected File to Decrypt
   2. Assume that the other user has shared their public key
      1. Select the public key file that they have shared with you
         1. For example, publicKey.txt
      2. “Browse”, under
   3. Select “Submit”
   4. If successful, you will see the following window

A screenshot of a computer error

Description automatically generated

* 1. It will be stored in the same folder
     1. “Decrypted – filename.txt”

1. Decrypt File (Authenticating sender’s identity)
   1. Same steps as 5, except, you would select their public key
2. The final result will look like this:

A screenshot of a computer

Description automatically generated

# References

Arizona State University. (n.d.). *RSA Cryptosystem.* Retrieved from School of Mathematical and Statistical Sciences: https://math.asu.edu/sites/default/files/rsa\_0.pdf

Lynn, B. (n.d.). *Primality Tests*. Retrieved from Number Theory - Primality Tests: https://crypto.stanford.edu/pbc/notes/numbertheory/millerrabin.html

Python Software Foundation. (n.d.). *Built-in Functions*. Retrieved from Python 3.12.1 Documentation: https://docs.python.org/3/library/functions.html?highlight=pow#pow

Python Software Foundation. (n.d.). *Random - generate pseudo-random numbers*. Retrieved from Python 3.12.1 Documentation: https://docs.python.org/3/library/random.html?highlight=random#module-random

Wikimedia Foundation. (2022, July 26). *65,537*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/65,537

Wikimedia Foundation. (2023, December 10). *Miller–Rabin primality test*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin\_primality\_test

Wikimedia Foundation. (2023, December 10). *RSA (cryptosystem)*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/RSA\_(cryptosystem)