Adam Chehadi
ECEN 2703
Final Project

RSA Encryption

Scope

The purpose of this project is to develop an RSA encryption implementation with Python. Two variations of RSA encryption were developed: asymmetric encryption and hybrid encryption. Hybrid RSA encryption is the extra credit extension for this project.

Asymmetric RSA encryption involves the receiver of encrypted messages creating a public and private key. The public key is visible to everyone and utilized for encryption by the entity sending the message. The private key is only known by the receiver, and utilized to decrypt the message. The public key is created by generating two large prime number, p and q, and an exponent e that is relatively prime to $(p-1)*(q-1)$. The cyphertext C is computed with modular exponentiation, and that encrypted message is sent to the receiver. The receiver decrypts the message by utilizing a private key with d and n. d is computed by taking the inverse of e modulo $(p-1)*(q-1)$. To decrypt the message and retrieve the plaintext value, the receiver utilizes modular exponentiation. Asymmetric RSA encryption is implemented in *RSA.py*.

Hybrid RSA encryption combines the techniques of asymmetric and symmetric encryption. Instead of utilizing public key encryption to encrypt the message, the private symmetric key encrypts the message. The symmetric key is encrypted using public key encryption. The steps of hybrid RSA encryption are:

1. Generate a symmetric key. The symmetric key needs to be kept a secret.
2. Encrypt the data using the secret symmetric key.
3. The person to whom we wish to send a message will share their public key and keep the private key a secret.
4. Encrypt the symmetric key using the public key of the receiver.
5. Send the encrypted symmetric key to the receiver.
6. Send the encrypted message text.
7. The receiver decrypts the encrypted symmetric key using their private key and gets the symmetric key needed for decryption.
8. The receiver uses the decrypted symmetric key to decrypt the message, getting the original message.

The asymmetric encryption involved with encrypting and decrypting the symmetric key utilizes nearly the same code and techniques that have been outlined above and contained in *RSA.py*. This project utilizes the python cryptography library to generate a symmetric key, encrypt the message with the symmetric key, and decrypt the message with the symmetric key. Hybrid RSA encryption is implemented in *RSA_hybrid.py*.

Motivation

Data is increasingly central to our personal lives, and that data must be kept secure. Similar to locking physical assets, encryption is utilized to protect data from being accessed by unintended groups of people. Encryption is important in our society due to the increased adoption of digital practices and data transmission.

The motivation for this project is an interest in cryptography and understanding the underlying concepts of cryptography and how to securely transmit data.

Goals

The goals for this project are to create an asymmetric RSA encryption and hybrid RSA encryption implementation with Python. The asymmetric RSA encryption will utilize public and private keys to encrypt and decrypt a message. The hybrid RSA encryption will utilize public and private keys to encrypt and decrypt a symmetric key, and the symmetric key will be utilized to encrypt and decrypt a message being transmitted.

Achievements

The achievements for this project are that the asymmetric RSA encryption and the hybrid RSA encryption Python implementations are successful. Both implementations are functional and work as intended.

The asymmetric RSA encryption implementation successfully creates a public key, creates a private key, accepts user input, encrypts the user input message with the public key, and decrypts the message with the receiver's private key.

The hybrid RSA encryption implementation successfully creates a public key, creates a private key, accepts user input, creates a symmetric key, encrypted the symmetric key with the public key, encrypted the user input message with the symmetric key, decrypts the symmetric key with the receiver's private key, and decrypted the message with the symmetric key.

Using The Code

The code for this project is a command line tool, and *RSA.py* and *RSA_hybrid.py* were developed using Python 3.8. The Python scripts utilize the sympy, math, and cryptography Python libraries, so those must be installed in order to use both scripts. Both scripts ask the user to input a message of their choice, and both scripts return the encrypted message and decrypted message on the terminal console.

While testing *RSA.py* and *RSA_hybrid.py*, two errors have been found on occasion: "Exception: No modular inverse," and "UnicodeEncodeError surrogates not allowed." The cause for these uncommon errors are unknown, and they have occurred very infrequently while testing the functionality of the Python scripts. If one of these errors are encountered, restart the script once or twice, and it should work successfully.

```
(base) ➜  ECEN_2703_Final_Project git:(main) ✗ python RSA_hybrid.py
Secret Key: b'r3plWSy6Johggf2HmWwiL_4SpMaSOGSwqDvzR_sRqbA='

Enter Message: hello world
Alice Encrypted Message:  b'gAAAAABghxoTwarPUepiYzEGlzhq4x9f0QlP1ruH1Bmdm9fZJLfXZ_l-9Cw4ehrgrFDvsTkE4mza6C02GuiY2m7fL45MnQrO
XQ=='

Bob Public Key (n, e): (2479, 239)
Bob Private Key (n, d): (2479, 2207)
Traceback (most recent call last):
  File "RSA_hybrid.py", line 116, in <module>
    encrypted_secret_key = numeric_string_conversion(str(secret_symmetric_key), True, Bob_n, Bob_e, 0)
  File "RSA_hybrid.py", line 56, in numeric_string_conversion
    return(''.join([chr(encrypt_decrypt_message(ord(char), enc_dec_bool, n, e, d)) for char in list(message)]))
  File "RSA_hybrid.py", line 56, in <listcomp>
    return(''.join([chr(encrypt_decrypt_message(ord(char), enc_dec_bool, n, e, d)) for char in list(message)]))
  File "RSA_hybrid.py", line 42, in encrypt_decrypt_message
    C = inverse_mod(char_val**e, n)
  File "RSA_hybrid.py", line 35, in inverse_mod
    raise Exception('No modular inverse')
Exception: No modular inverse
(base) ➜  ECEN_2703_Final_Project git:(main) ✗
```

No modular inverse, error example

```
(base) ➜  ECEN_2703_Final_Project git:(main) ✗ python RSA_hybrid.py
Secret Key: b'I_IL2GGXrL8A6L9Fodjt2p2YXiDZAco8WTFi9HgfSps='

Enter Message: i have a secret message
Alice Encrypted Message:  b'gAAAAABghxpM15Ll5VFsuzgfCLcyFA3cs-67zaE8D2tn_TbEHabdtMcyvJ8_l2owaAdSGtEzYoCoTZxbq4Uus2OvY5dhdwu7OF3xRXzZ48AOQ_ZYG83nEgk='

Bob Public Key (n, e): (368383, 197)
Bob Private Key (n, d): (368383, 337237)

Alice Encrypted Secret Key: Traceback (most recent call last):
  File "RSA_hybrid.py", line 117, in <module>
    print('\nAlice Encrypted Secret Key:', encrypted_secret_key)
UnicodeEncodeError: 'utf-8' codec can't encode character '\ud955' in position 27: surrogates not allowed
```

UnicodeEncodeError surrogates not allowed, error example

Sources

The listed sources were utilized during the development of this project.

Math 114 Discrete Mathematics
Cryptography and the number theory behind it
D Joyce, Spring 2018
https://mathcs.clarku.edu/~djoyce//ma114/crypto.pdf

numberTheory-h.pdf

Encryption, Part 3: Hybrid Encryption
https://dzone.com/articles/encryption-part-3-hybrid-encryption-symmetric-publ

How to Encrypt and Decrypt Strings in Python?
https://www.geeksforgeeks.org/how-to-encrypt-and-decrypt-strings-in-python/