NIS ASSIGNMENT TASK 1 Daniel Collins CLLDAN008

This document will describe and show an example of how symmetric key distribution can be done using asymmetric cryptography. The screenshots provided show Task1A on the left and Task1B on the right.

**Language:** Python

**API libraries:** Python Cryptography library

# Task 1 Brief

Task1A and Task1B parties are programs that use RSA encryption, which is secure based on the assumption of the difficulty of factoring large numbers. So initially both parties will have a public and a private key. The public key is used to encrypt messages and can be used by anyone who has access to it to encrypt and send messages to the owner of the associated private key. The owner of the private key is the only person who can decrypt these messages.

Once parties have exchanged public keys and have verified that the other is the true owner of their associated private keys, a symmetric secret key is exchanged. First, party A will generate and encrypt the secret key with B’s public key and send that message to B. Party A will then sign the same message with their private key to prove that the message was sent by them. Party B will then check this signed message using A’s public key. This ensures that it was truly A that sent the secret key.

Both parties also use OAEP (Optimal Asymmetric Encryption Padding), which is a padding scheme [proven to be secure](https://cseweb.ucsd.edu/~mihir/papers/oae.pdf) against several attack types, including chosen cipher text attack and chosen plaintext attack. It also prevents partial decryption or other information leakage. This is the [recommended padding algorithm](http://www.daemonology.net/blog/2009-06-11-cryptographic-right-answers.html) for RSA encryption, according to the Python Cryptography library. This padding scheme using the SHA256 hashing algorithm ensures that messages cannot be tampered with in transmission.

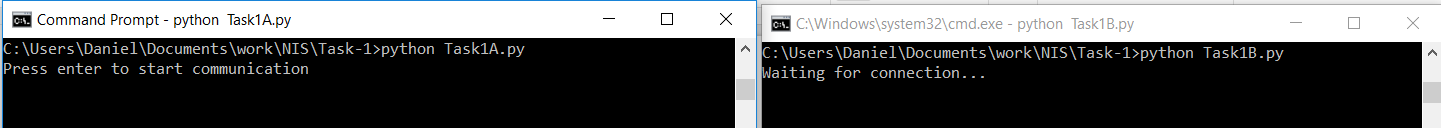
This padding also ensures that repeatedly sending the same encrypted message does not give an attacker any advantage.

For the signing of the secret key by party A, PSS (Probabilistic Signature Scheme) is used. This scheme is often used for verifying RSA signatures.

# Program Screenshots and Descriptions

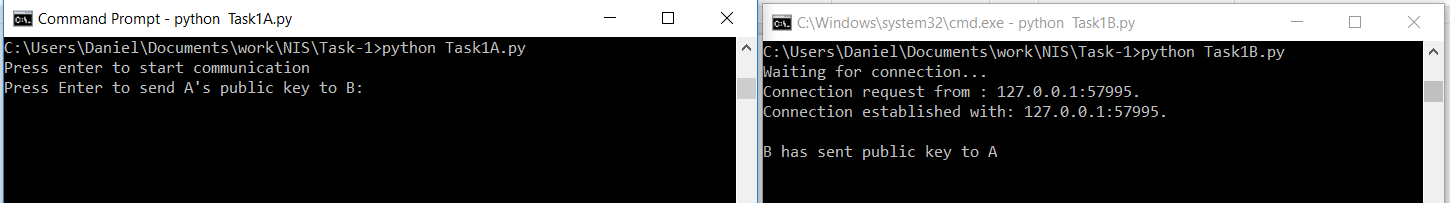
Step 1:

Prompt user to start the communication



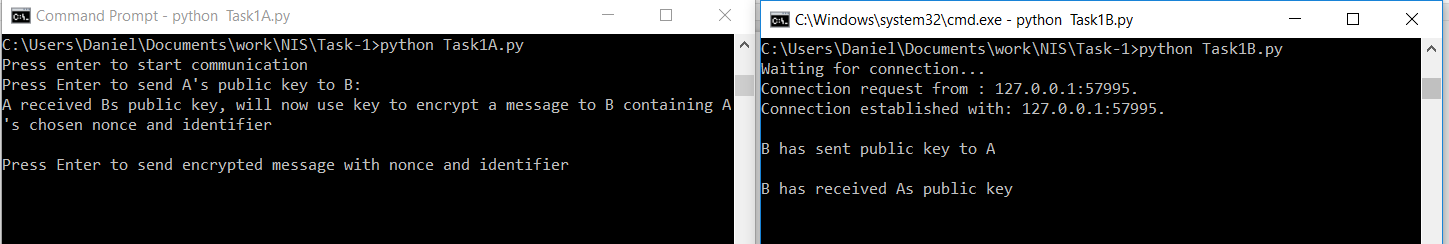
Step 2:

B accepts the communication requests and starts a new thread to handle communication with A. B then immediately sends A a message containing B’s public key. A then sends B its public key.



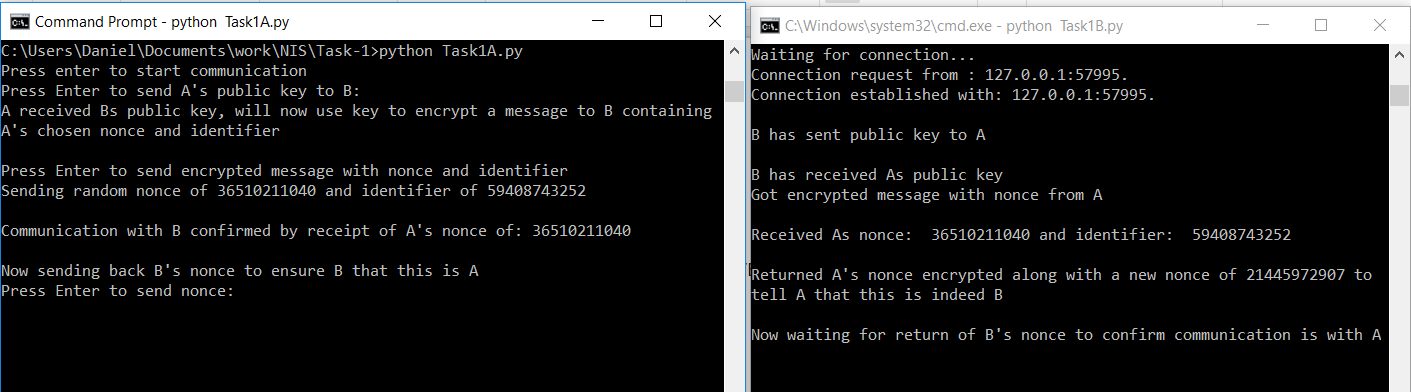
Step 3:

A receives B’s public key, and chooses a random number, the nonce, and an identifier, which is also random for the sake of this assignment. A then concatenates (with a colon as a delimiter) and encrypts these two numbers with B’s public key and sends this to B.



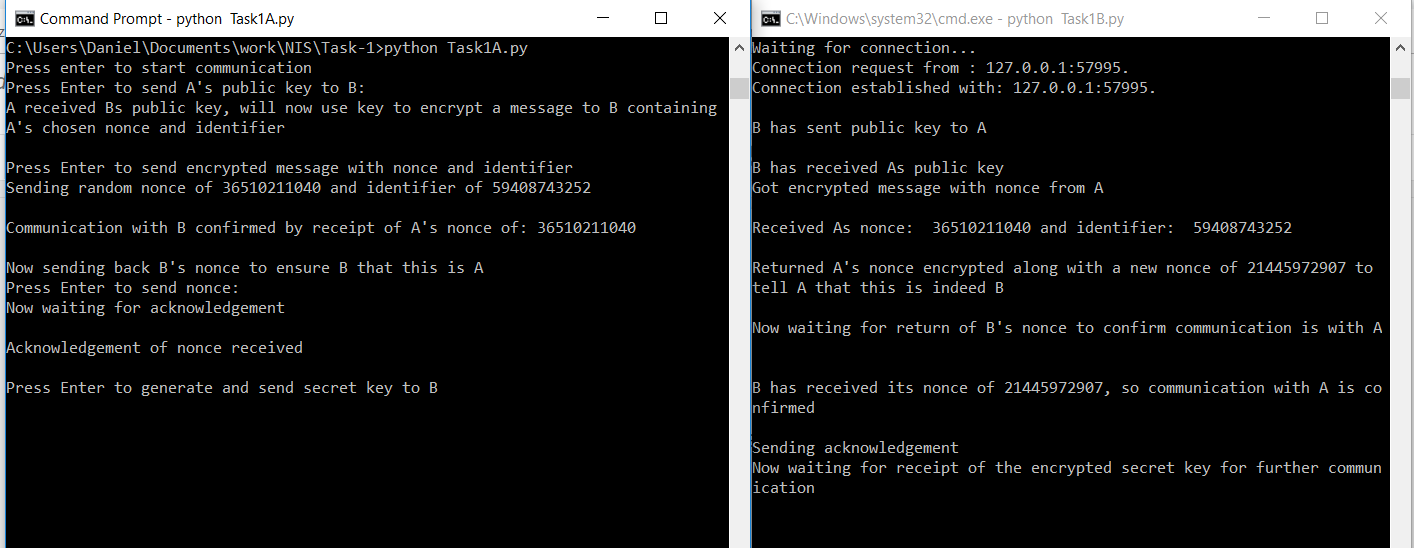
Step 4:

B then decrypts this message with its private key and saves the nonce and identifier. B then chooses its own random number as a nonce, concatenates this nonce with A’s nonce as done previously, and encrypts this with A’s public key. This message is then sent to A. A receives this message and decrypts it, and checks if the nonce in the first half of the message is the same as the one it previously sent. If it is the same, then this confirms that it is communicating with B, because only B would have been able to read the nonce A sent by decrypting it with its private key. A then encrypts B’s given nonce with B’s public key and sends this to B. This will tell B that is it truly communicating with A, because only A would be able to decrypt the message with its private key to read the nonce.



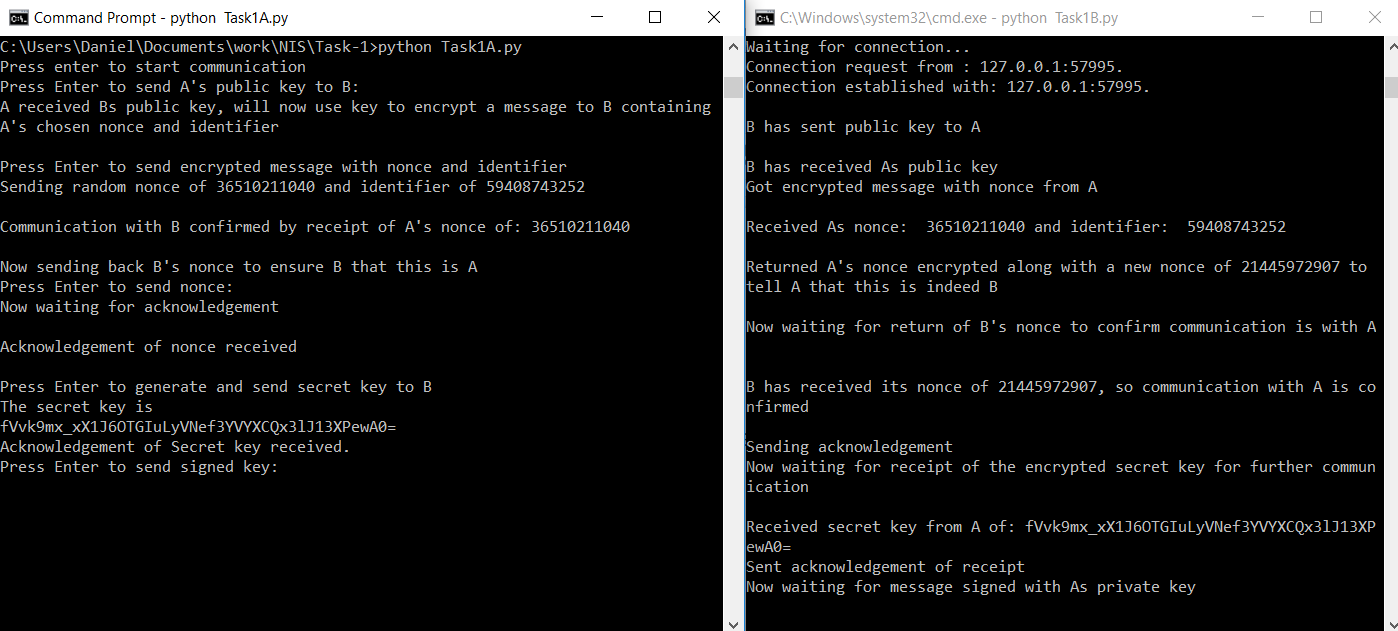
Step 5:

B decrypts the message containing its nonce and compares it to its previously chosen nonce, and if they match then communication with A is confirmed. B then sends acknowledgement of the nonce back to A.



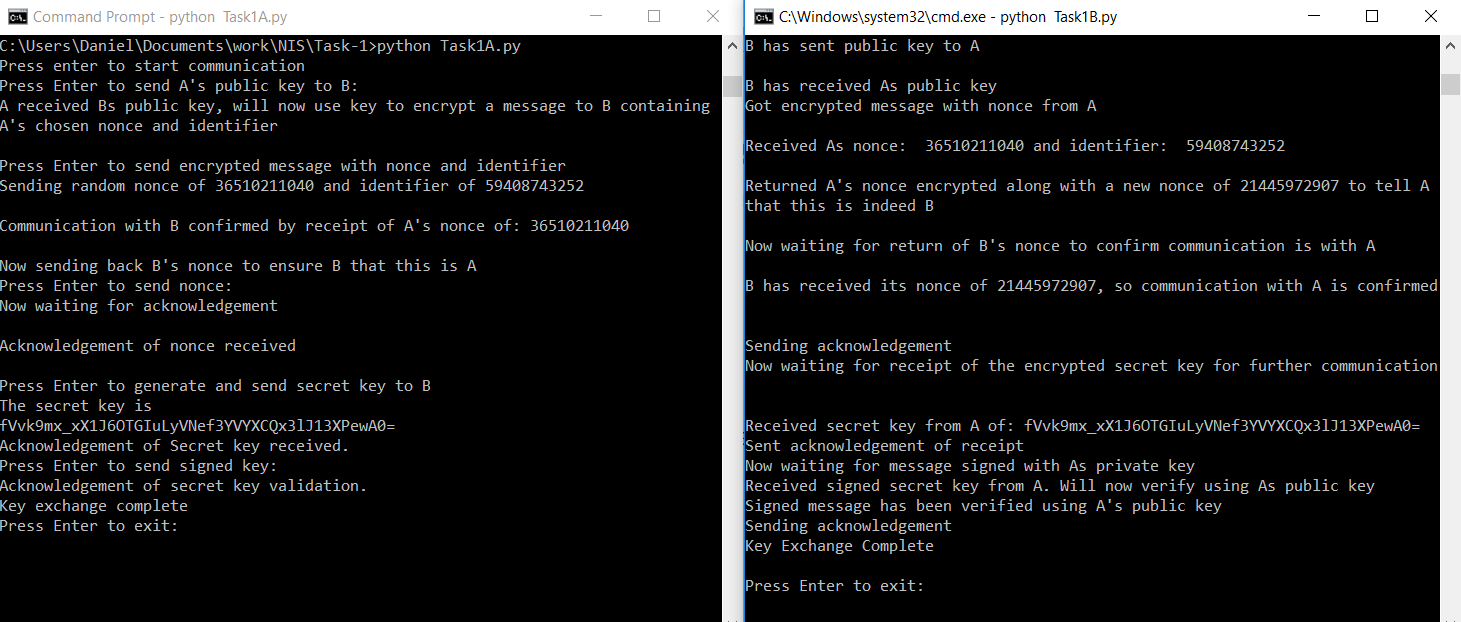
Step 6:

A then generates a random secret key and encrypts it with B’s public key, and sends this to B. B then decrypts this message with this private key to read the secret key and saves this key. This key is safe because only B can decrypt this message to read the secret key. B then sends acknowledgement of receipt back to A and waits for A to send the signed private key.



Step 7:

A then signs the message encrypted with B’s public key containing the secret key with A’s own private key. This signed message is then sent to B. B then uses A’s public key to verify that the message was signed with A’s private key. Only A has access to its private key, so this ensures B that the secret key was sent and signed by A. The secret key is now known by both parties and it has been proven that A was the sender of the key, and B was the receiver. B then sends acknowledgement of the verification to A.



Now that both parties have the same secret key they can use this key to symmetrically encrypt and decrypt messages securely.

# Features Implemented To Ensure Security

The application, with Task1A and Task1B programs, used Python cryptography API libraries with RSA encryption to implement security features that ensure security of the Asymmetric keys, their storage and distribution between sending and receiving parties using a set of security services. The security services implemented as features in the application entail the following.

**Authentication and Authenticity**

A Nonce and identifier were generated on Python to implement authentication and authenticity features for sender and receiver programs to uniquely identify and authenticate each other. This entailed padding the Nonce and identifier data with OAEP and SHA256 algorithms to be exchanged between sender and receiver parties for verification before encrypted messages are exchanged.

**Data Access Control**

The generated keys are locate inside the program and are retrieved safely with the use certificates and access is restricted to the authenticated program user to ensure that no unrecognized access, such as read, write and execute, is made to illegally obtain the keys or illegally execute a send/receive program. This is crucial because if the private and secret keys are accessed, they can be used to perform misuse of resources and data breach.

**Non-Repudiation - undeniable send or receive**

Signing of the Asymmetric keys is performed which ensures that receiver A can verify that it was sender B who sent the message and sender B can verify that receiver A did receive a message from B proven with the signature of a secret key transferred between these communicating parties and thus either party cannot deny that certain actions were not performed by them.

**Confidentiality and Integrity**

The incorporation of OAEP padding scheme and SHA256 hashing algorithm on the programs ensures that a third party intercepting the connection to read and modify data or keys on transit will not be able to achieve data breach and theft as the data is encrypted on the programs and sent as cipher text to the other party. This protects the privacy of data from possible security attacks. Integrity is achieved with a scheme that uses SHA-256 hashing algorithm to ensure that data or key modification on transit is useless.