REPORT: A description of RSA's encryption and decryption procedure.

RSA is an algorithm used in modern day computers, it's used to generate keys which are used to encrypt and decrypt data, RSA creates a sense of confidentiality in communication. RSA is asymmetric. Asymmetric means that there are two keys that every user receives, a public key, that everyone can see, to encrypt messages and a corresponding private key, that's hidden, to decrypt the message. RSA is hard to break because it's extremely hard to factorise, meaning that it's hard to find the prime number factors of a large composite number.

Generating the Keys

RSA needs two keys, a public key, to decrypt the data and a private key, used for decrypting data. However, data encrypted with a public key can only be decrypted using it's corresponding private key.

Public key generation:

- First pick two random prime numbers (**p, q**).
- Then multiply **p** and **q** to get the product **N** (**N = p*q**). **N** is important because it's used in the encryption lock and it's needed through the whole RSA system.
- Calculate the totient **phi** by subtracting one from **p** and **q** and multiplying them together **(phi = (p – 1)(q – 1)).** The totient shows you how many numbers are coprime with **N** and smaller than **N**.
- Now the public key **e** needs to be picked. **e** has two conditions, it must be between **1** and **phi**. And **e** must be coprime with **N** and with **phi**, this means that **e** must share no factors with **N** and **phi** other than **1**.
- Finally, once **e** has been picked, we now have our encryption lock. The encryption lock consists of **e** and **N**. It's normally represented as (**e, N**).

Private key generation:

- Once we have the encryption lock, we can calculate the private key **d** from it. **d** is calculated using the equation **d = ((1 + phi)/e).** This is because **d*e = 1 (mod(phi)).**
- After calculating the **d**, we now have our decryption lock. The decryption lock consists of **d** and **N**. it's normally represented as (**d, N**) and it's kept a secret.

Generating large primes: The first part of generating the public key is picking two large primes. The problem with this is how do you know a number is prime without doing exhaustive calculations. The most common way to test this is by doing a probabilistic test for primality on a number using a Fermat primality test.

In a Fermat test, a number **p** is prime if **a^(p-1) = 1 (mod p)** where **a** is a random number between **a** and **(p – 1)**. **p** would be tested against multiple **a** using the equation.

Encrypting a message:

Using the encryption lock that was generated, we can encrypt messages. Firstly, we do this by taking a plaintext message and converting it to a number (usually it can be using ASCII to decimal format). Once the plaintext is represented as a number, we compute the ciphertext using the following formula:

$$c = m^e \ (mod\ N)$$

Where m is the message as a number, e is the encryption key, N is the product of the two initial prime numbers that were generated, and c is the ciphertext which is unreadable and a by a computer or human. e and N are found within the encryption lock.

Decrypting a ciphertext:

Using the decryption lock that was generated from the encryption key, we can decrypt a ciphertext. We can compute the plaintext from the ciphertext by using the following formula:

$$m = c^d (mod\ N)$$

Where c is the ciphertext, d is the decryption key, N is the product of the two initial prime numbers and m is the plaintext message represented as a number. The decryption key and N are kept within the decryption lock.

Normal Scenario:

Two users, Alice and Bob want to send a secret message. First, they would both have to generate their own key pair, which consists of a public key and a private key. Alice and Bob would then make their public keys publicly available to see by everyone.

If Alice wanted to send a message to Bob, she would have to take Bob's public key and encrypt her message with Bob's public key, she then sends the ciphertext to Bob. This means that only Bob can decrypt the ciphertext because his private key is the only key that can decrypt the ciphertext. Once Bob receives the ciphertext, he can decrypt the message using the decryption formula shown above.

This system shows that every user would have to generate a key pair and they would have to keep their decryption key secret.

Extra security:

Raw RSA can easily be broken, traffic analysis could reveal information about the ciphertext. For example, someone could tell when the message changes, if RSA was used to send secret information in a warzone, the message encrypt("stay put") will be sent continuously and it will always show the same ciphertext but as soon as they send out encrypt("attack"), the adversary will see that the message has changed and they'll know something is going on.

Padding can add another level of randomness or security to RSA. Padding is done by adding a bunch of random values to the beginning of the plaintext before encrypting it. This will guarantee that the ciphertext will always be different and it'll never reveal information. After the message is decrypted, there must be a system that removes the padding.