Cryptography Project  
  
Detailed Report

**Name Section Bench No.**

Yousef Alwaer 2 37

Supervised by

Dr. Samir Shaheen

Eng. Kaled moataz

April 2023

Table of Contents

[**Implementation Details**](#_dpv7yjpi1f2n)

[RSA Algorithm](#_w4gfnk9ztf7r) 3

RSA World End-to-End Encryption 4

[RSA Mathematical Brute-force Attack Test](#_ex8t62c7m5f8) 5

[RSA Analysis](#_cv0kx6mp7teo) Results & Conclusion 6

## Implementation Details

### RSA Algorithm

RSA (Rivest-Shamir-Adleman) is a public-key encryption algorithm widely used in modern cryptography. It was invented by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. The security of RSA is based on the fact that it is difficult to factor large prime numbers, making it infeasible to determine the private key given only the public key. RSA is widely used in various applications, such as SSL/TLS, email encryption, and digital signatures.

The RSA algorithm involves three main steps: key generation, encryption, and decryption. Key generation is the process of creating the public and private keys that will be used for encryption and decryption. Encryption is the process of encoding a message using the public key, while decryption is the process of decoding the encrypted message using the private key.

In the key generation process, two distinct prime numbers p and q are selected. These prime numbers are used to compute the modulus n = p \* q, which is part of both the public and private keys. The Euler totient function phi(n) is also computed, where phi(n) = (p-1) \* (q-1). A public exponent e is selected, typically a small prime number such as 65537, that is coprime to phi(n). The private exponent d is computed using the extended Euclidean algorithm, such that e \* d = 1 (mod phi(n)). The public key is (e, n), while the private key is (d, n).

Encryption of a message m involves converting it to an integer using a reversible encoding function, which maps a string of characters to a unique integer. The encoded message m is then encrypted using the public key, by computing the ciphertext c = m^e (mod n).

Decryption of the ciphertext c involves computing the plaintext message m, by computing m = c^d (mod n) using the private key. The encoded message m is then converted back to the original message using a decoding function that maps the unique integer back to the original string of characters.

The implementation provided in the code includes functions for encoding and decoding messages, as well as generating the public and private keys. The encode\_message() function takes a string of characters and converts it to a unique integer using a base-37 encoding. The decode\_message() function performs the reverse operation, converting the unique integer back to the original string of characters.

The generate\_keypair() function takes two prime numbers p and q as input, and generates the public and private keys for the RSA algorithm. The function chooses a public exponent e that is coprime to phi(n), using a random number generator. It then computes the private exponent d using the extended Euclidean algorithm, such that e \* d = 1 (mod phi(n)).

The encrypt() function takes a message m and a public key (e, n) as input, and encrypts the message using the RSA algorithm. It computes the ciphertext c = m^e (mod n) using modular exponentiation.

The decrypt() function takes a ciphertext c and a private key (d, n) as input, and decrypts the ciphertext using the RSA algorithm. It computes the plaintext message m = c^d (mod n) using modular exponentiation.

### RSA World: End-to-end Encrypted Chat

### Server

* The server starts by creating a socket object using socket.socket().
* It gets the name of the local machine using socket.gethostname().
* It reserves a port number for the service and binds the socket to that port using server\_socket.bind((host, port)).
* It listens for incoming connections from clients using server\_socket.listen(2).
* Once two clients connect, it accepts the connections and receives the public keys from each client. The public keys are received as a JSON string that has been encoded using base64.
* It then sends each client the other client's public key so that they can encrypt messages to each other.
* The server then receives the names of each client and sends each client the other client's name.
* Finally, it starts a new thread that receives messages from one client and sends them to the other.

### Client

* The client starts by creating a socket object using socket.socket().
* It gets the name of the local machine using socket.gethostname().
* It connects to the server using client\_socket.connect((host, port)).
* It generates a public-private key pair using the RSA algorithm implemented in a separate file called RSA.py.
* It sends the public key to the server, encoded as a JSON string that has been base64 encoded.
* It receives the other client's public key from the server, which is also a JSON string that has been base64 encoded. It then decodes the string and evaluates it as a Python object to get the public key.
* It sends its name to the server.
* It receives the other client's name from the server.
* Finally, it starts a new thread that sends messages to the other client after encrypting them using the other client's public key, and receives messages from the other client, decrypting them using its own private key

### RSA Mathematical Attack Test

In the mathematical attack, we attempt to factorize a set of public keys, which range from 8 to 1024 bits (14, 16, 18, ..., 1024), and measure the time for each case. We employed a simple algorithm for integer factorization, which searches for divisors of the given integer by considering only those odd ones lying in the range up to the integer's square root. (Please see the appendix for why looking in this range suffices.) Once a divisor is discovered, it is guaranteed to be prime (as we iterate from small numbers to larger ones), and dividing the public key by it should produce the other prime.

However, as demonstrated in the recursive algorithm below, we keep searching for primes if the remaining number is composite after division, and we report that the key is not due to RSA in a wrapper function if more than two primes are discovered.

def factorize(n):

   if (n % 2) == 0:

      return [2] + factorize(n//2)

   integer = 3

   while integer <= (n\*\*0.5):

      if n % integer == 0:

         return [integer] + factorize(n // integer)

      else:

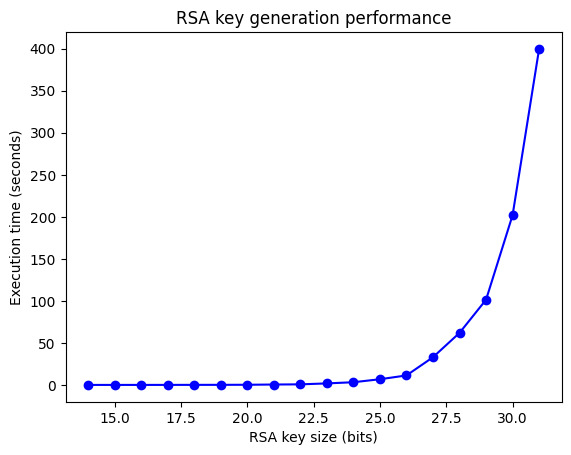
         integer += 2                        # Since all primes are odd.

   return [n]

### 

## Analysis Results & Conclusions

### RSA MATH ATTACK



The key size of RSA is directly proportional to the time required to break it. As the key size increases, the time required to factorize the large number used in RSA encryption also increases exponentially. This is due to the fact that factoring large numbers is a computationally intensive task that becomes exponentially more difficult as the size of the numbers increase.