RSA Algorithm

1. Modulus: p\*q

P=23

Q=13

N=23\*13=299

1. Fuler Totient

Formula: Ꝋ(n) = (p-1) \* (q-1)

N = (23-1) \* (13-1)

= 264

1. 1 < e < Ꝋ(n) 1 and e is co-prime with Ꝋ(n)

5,7,13,17,19,23,25,29,31,35

1 < e < Ꝋ(n) and gcd (e, Ꝋ(n) 1 = 1 for Ꝋ(n) =264

1. A. d \* e = 1 (mod Ꝋ(n) )/ d\*13 = 1 (mod 264)

264 = 20\*13+4

13 = 13\*4+1

4 = 4\*1+0

Re = 0; gcd is 1

B. sub 4 = 264-20\*13

1=13-3(264-20\*13)

1=13-3\*264+60\*13

1 = 61\*13-3\*264

d=61 satisfies 13\*61=1(mod 264)

C. d \* 23 = 1(mod 264)

264 = 11\*23+11

23 = 2\*11+1

11 = 11\*1+0

D. 11 = 264 -11 \* 23

1 = 23-2\*(264-11\*23)

1= 23 -2\*264+22\*33

1=23\*23-2\*264

D = 23 satisfies 23\*23 = 1(mod 264)

E. e= 13 d=61/e=23 d=23

1. A computer screen with white text

   Description automatically generated
2. A computer screen with text

   Description automatically generated

RSA is a public-key algorithm for establishing, generating, and verifying digital signatures. The term RSA comes from the initials of its creators, Rivest, Shamir, and Adleman, who developed this cryptosystem. RSA is a form of asymmetric encryption, meaning it uses two different but linked keys to secure sensitive data, mainly when transmitted over the Internet. The security of the RSA algorithm relies on the difficulty of factoring large integers, especially those that are the product of two large prime numbers. RSA is used for secure communication by encrypting data between two parties over an insecure network. It is also used for digital signatures to verify the authenticity of messages or documents, for digital certificates to confirm the identity of individuals or organizations behind websites, and in software protection mechanisms, such as web browsers, to establish secure connections.

Building the RSA algorithm involves several steps, beginning with calculating the modulus. First, the modulus is calculated by multiplying two prime numbers together. This is done using the formula where the modulus equals the product of the two primes. After determining the modulus, the next step is calculating Euler’s Totient, which represents the number of integers coprime with the modulus. The formula used to calculate Euler’s Totient involves multiplying one less than each of the prime numbers used to calculate the modulus.

Next, an encryption exponent is selected. This exponent must be a number greater than one but less than Euler’s Totient, and it must also be coprime with Euler’s Totient, meaning it has no common factors with it other than one. The most commonly used encryption exponents are 3, 5, and 65537 because they usually meet the coprime requirement.

After selecting the encryption exponent, the next step is to compute the decryption exponent. The decryption exponent is calculated using a formula that ensures that multiplying the encryption and decryption exponents together will yield a number congruent to one when divided by Euler’s Totient. The Extended Euclidean Algorithm is used to find this decryption exponent.

For encryption, a function is created that uses a formula to raise the message to the power of the encryption exponent and then reduce it by dividing it by the modulus. Before this calculation can be done, the message is converted into an integer format, such as by using ASCII encoding.

For decryption, a separate function is created that uses a similar formula. Still, this time, the ciphertext is raised to the power of the decryption exponent and then reduced by dividing it by the modulus. After this calculation, the resulting integer is converted to its original format to recover the original message. However, a challenge was encountered during this decryption process, where an error occurred when attempting to convert the decrypted integer back into a string. The **UnicodeDecodeError** error indicated that the program could not correctly decode the decrypted data, resulting in an “unexpected end of data” issue.

Finally, the program is tested by encrypting and decrypting a sample message. By comparing the original message with the decrypted message, the correctness of the algorithm is verified. Example inputs and outputs demonstrate that the encryption and decryption processes function as expected.