Asymmetric Encryption Systems

Kalp Rawat

June 23, 2024

1 Introduction

Unlike symmetric encryption systems where the encryption and decryption are carried out using the same key, asymmetric encryption systems use two different keys that are linked to each other in some manner. Some of the famous asymmetric encryption systems are **RSA, Paillier, and ElGamal encryptions**.

2 RSA Encryption System

Rivest-Shamir-Adleman (RSA) is a well-known public-key or asymmetric cryptographic algorithm. It protects sensitive data through encryption and decryption using a private and public key pair. First introduced in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman of the Massachusetts Institute of Technology, RSA is named after their last initials.

2.1 Key Generation

1. The system requires two sufficiently large prime numbers, let us say p and q, and we form the product $N = p \cdot q$ from them.
2. Then, we select an integer e such that $\gcd(e, (p-1)(q-1)) = 1$. The pair (e, N) is the **public key**.
3. We then compute d such that d is the modular inverse of e, i.e., $d \cdot e \equiv 1 \pmod{(p-1)(q-1)}$.
4. The pair (d, N) is the **private key**.

2.2 Encryption

The encryption works as follows:

$$c = m^e \pmod{N}$$

2.3 Decryption

The decryption works as follows:

$$m = c^d \pmod{N}$$

2.4 Working Mechanisms and Security Features

RSA utilizes a public and private key pair. The private key is kept secret, and the other key is made public. The RSA encryption system is based on the fact that factoring a product into its prime components is a challenging task. The security of the RSA algorithm heavily relies on large, difficult-to-factor prime numbers used for the key generation process. As the key length increases, the computational power required to break the code also increases, making the encryption more secure.

```python
import random
def gcd_eu(a,b):
    if(a==0):
        return b
    return gcd_eu(b%a,a)

def not_equal(p,q):
    if(p==q):
        q=random.randint(0,200)


def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

Figure 1: Code Snippet for RSA

```python
p=random.randint(0,200)
q=random.randint(0,200)
while (is_prime(p)==False or is_prime(q)==False):
    p=random.randint(0,200)
    q=random.randint(0,200)
    not_equal(p,q)

n=p*q
phi=(p-1)*(q-1)
e=random.randint(1,200)

while(e<phi):
    if(gcd_eu(e,phi)==1):
        break
    e+=1                    #this is to see if the e that we selected at once is not a factor of n

def mod_inverse(e, phi):
    phi0, x0, x1 = phi, 0, 1  # Initializing variables
    while e > 1:
        q = e // phi           # Compute the quotient
        phi, e = e % phi, phi  # Update phi and e
        x0, x1 = x1 - q * x0, x0  # Update x0 and x1
    return x1 + phi0 if x1 < 0 else x1  # Ensure the result is positive
d=mod_inverse(e,phi)

message=int(input("Enter message to encrypt:"))
encrypted_message=(pow(message,e))%(n)
print(encrypted_message)
```

Figure 2: Code Snippet for RSA

3 Paillier Encryption System

The Paillier cryptosystem, invented by and named after Pascal Paillier in 1999, is a probabilistic asymmetric algorithm for public key cryptography.

3.1 Key Generation

1. The system requires two sufficiently large prime numbers, let us say p and q, and we form the product $N = p \cdot q$ from them.
2. Then, we select an integer e such that $\gcd(e, (p-1)(q-1)) = 1$.
3. We then compute g such that g is equal to $N + 1$.

3.2 Encryption

The encryption works as follows:

$$c = (g^m) \cdot (e^N) \pmod{N^2}$$

3.3 Decryption

The decryption works as follows:

$$m = L(c^\lambda \mod N^2) \cdot \mu \mod N$$

where $\lambda = \text{lcm}(p-1, q-1)$ and $L(x) = \frac{x-1}{N}$.

$$\mu := \left(L(g^\lambda \mod N^2)\right)^{-1} \mod N$$

3.4 Working Mechanisms and Security Features

The homomorphic properties of the Paillier cryptosystem make it very useful for secure multi-party computations: if you encrypt two messages m_1 and m_2, and multiply the ciphertexts together, the result will decrypt to $m_1 + m_2$.

Moreover, the ciphertexts are randomized: if you encrypt the same number twice, the two ciphertexts will be different. Only the person with the private key can see that they contain the same thing.

```python
import random
def gcd_eu(a,b):
    if(a==0):
        return b
    return gcd_eu(b%a,a)

def not_equal(p,q):
    if(p==q):
        q=random.randint(0,200)


def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

Figure 3: Code Snippet for Paillier

```python
p=random.randint(0,200)
q=random.randint(0,200)
while (is_prime(p)==False or is_prime(q)==False):
    p=random.randint(0,200)
    q=random.randint(0,200)
    not_equal(p,q)

n=p*q
phi=(p-1)*(q-1)
g=n+1
e=2

while(e<n):
    if(gcd_eu(e,n)==1):
        break
    e+=1

def mod_inverse(e, phi):
    phi0, x0, x1 = phi, 0, 1
    while e > 1:
        q = e // phi
        phi, e = e % phi, phi
        x0, x1 = x1 - q * x0, x0
    return x1 + phi0 if x1 < 0 else x1

msg=int(input("Enter message:"))
encrypted_message=((pow(g,msg))*(pow(e,n)))%(pow(n,2))
print(encrypted_message)
```

Figure 4: Code Snippet for Paillier

4 ElGamal Encryption System

In cryptography, the ElGamal encryption system is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie–Hellman key exchange. It was described by Taher Elgamal in 1985.

4.1 Key Generation

1. We select a prime number p and a primitive root of p, α.
2. We then select a random number in the range 1 to $p-1$, say a.
3. The value of a is kept **private**.
4. We then compute $\beta \equiv \alpha^a \pmod{p}$.
5. The triplet (p, α, β) is then made **public**.

4.2 Encryption

The encryption works as follows: Suppose we want to send a plain message P. We first choose a random number $1 \leq k \leq p-2$. Then the encryption is carried out as follows:

$$E(P) \equiv (\alpha^k, P \cdot \beta^k) \pmod{p}$$

This produces a pair which constitutes the ciphertext.

4.3 Decryption

The decryption works as follows: To decrypt the ciphertext block (γ, δ), we claim that:

$$P \equiv \gamma^{p-1-a} \cdot \delta \mod p$$

This works because of the following, where the negative exponent means the multiplicative inverse:

$$\gamma^{p-1-a} \cdot \delta = (\alpha^k)^{p-1-a} \cdot P \cdot \beta^k \mod p$$

$$= (\alpha^{p-1})^k \cdot (\alpha^a)^{-k} \cdot \beta^k \cdot P \mod p$$

$$= (1)^k \cdot \beta^{-k} \cdot \beta^k \cdot P \mod p$$

$$= P \mod p$$

$$\alpha \text{ is a primitive root of } p \implies \alpha^{p-1} \equiv 1 \pmod{p}$$

4.4 Working Mechanisms and Security Features

In search of random prime for the algorithm and then taking a random primitive root of it , the larger its value is the more it is beneficial for the algorithm to be strong.
It is based on discrete logarithms mechanism of encryption and is indeed a very secure system.

```python
import random

def not_equal(p,q):
    if(p==q):
        q=random.randint(1,p-1)

def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

p=random.randint(0,200)
while (is_prime(p)==False):
    p=random.randint(0,200)
```

Figure 5: Code Snippet for ElGamal

```python
def mod_inverse(e, phi):
    phi0, x0, x1 = phi, 0, 1  # Initializing variables
    while e > 1:
        q = e // phi          # Compute the quotient
        phi, e = e % phi, phi  # Update phi and e
        x0, x1 = x1 - q * x0, x0  # Update x0 and x1
    return x1 + phi0 if x1 < 0 else x1  # Ensure the result is positive

a=1
for i in range(1,p):
    for j in range(1,p):
        if ((pow(a,i))%p==(pow(a,j))%p):
            a=i
        else :
            a+=1

#we get "a" which is the primitive root
h=random.randint(0,p)
b=(pow(a,h))%p
print("public key:",p,a,b)
k=random.randint(1,p-1)
msg=int(input("enter message:"))
encrypted_message_1=pow(a,k)%p
encrypted_message_2=(msg*(pow(b,k)))%p
print(encrypted_message_1,encrypted_message_2)
```

Figure 6: Code Snippet for ElGamal