Elliptic Curve Cryptography

1. **Introduction**

   Elliptic Curve Cryptography (ECC) represents a cutting-edge approach to cryptographic security, leveraging the mathematical properties of elliptic curves over finite fields. These curves, defined by equations such as $y^2 = x^3 + ax + b$, where a and b are constants satisfying $4a^3 + 27b^2 \neq 0$), form the basis of ECC's robustness and efficiency. By harnessing these curves, ECC provides a secure foundation for key exchange, encryption, and decryption processes in digital communication. Renowned for its ability to offer high levels of security with reduced computational overhead, ECC stands as a pivotal advancement in modern cryptography, addressing the evolving challenges of securing sensitive data in diverse applications.

   The adoption of ECC arises from its remarkable efficiency and security advantages over traditional cryptographic systems like RSA. ECC requires smaller key sizes to achieve equivalent security levels, making it particularly suitable for resource-constrained environments such as mobile devices and Internet of Things (IoT) devices. Its ability to offer high levels of security with reduced computational overhead makes ECC an attractive choice for securing sensitive data in various applications.

   Elliptic Curve Cryptography (ECC) operates within finite fields, where both variables and coefficients are constrained. Two common choices are prime curves over Z_p and binary curves over $GF(2^m)$. The elliptic curve equation over finite fields is expressed as $y^2 \bmod p = (x^3 + ax + b) \bmod p$, where a and b are curve-specific coefficients. This approach enhances security and efficiency by confining computations to bounded value ranges, making ECC suitable for diverse cryptographic applications.

   The addition operation in ECC defines a group law, allowing for the addition of two points on the elliptic curve. This operation is geometrically intuitive and involves drawing a line through two points on the curve, finding the third point of intersection, and reflecting it across the x-axis to obtain the result of the addition. This process forms the foundation for ECC operations such as key generation and encryption.

   Key generation in ECC involves selecting a private key d and deriving a corresponding public key Q by performing scalar multiplication of a base point on the curve G by the private key. Encryption and decryption processes utilize the properties of elliptic curves to ensure confidentiality and authenticity of transmitted data. Encryption involves generating a shared secret using the recipient's public key and combining it with the plaintext message to produce the ciphertext. Decryption, on the other hand, requires the recipient's private key to derive the shared secret and recover the original plaintext message.

2. Objectives

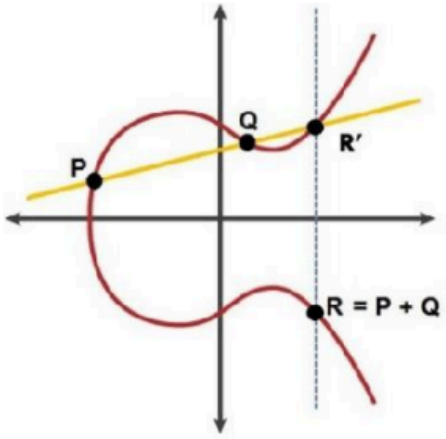
- Implement secure and efficient key generation mechanisms to generate public and private key pairs for use in Elliptic Curve Cryptography (ECC), ensuring robust cryptographic operations.
- Develop encryption and decryption algorithms based on ECC principles to enable secure and reliable transmission of confidential data over digital networks, safeguarding against unauthorized access and interception.

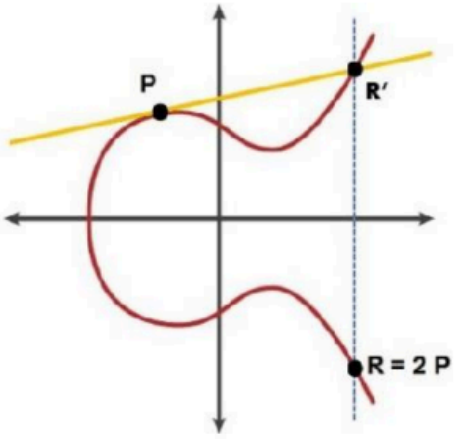
3. Implementation and Results analysis

Elliptic Curve Cryptography (ECC) is implemented step by step, beginning with the establishment of the curve, defining points on a finite field, and conducting operations such as addition and multiplication within the finite field. The process involves key generation, encryption, and decryption, ensuring secure communication over digital networks.

Curve and Points on Finite Field

ECC operates within the framework of elliptic curves defined over finite fields. The curve is defined by the equation $y^2 = x^3 + ax + b \bmod p$, where a and b are curve parameters, and p is a prime number or an integer of the form 2^m. Points on the curve satisfy this equation, forming the basis for cryptographic operations.





Finite Field Addition

The addition of two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on the curve within a finite field involves calculating a third point $R = (x_3, y_3)$. The coordinates of R are determined by the formulas:

$x_3 = (\lambda^2 - x_1 - x_2) \bmod p$

$y_3 = (\lambda(x_1 - x_3) - y_1) \bmod p$

Here, $\lambda = (\{y_2 - y_1\} \div \{x_2 - x_1\}) \bmod p$, ensuring that the addition operation remains within the finite field.

Key Generation Formulas

Key generation in ECC involves selecting private keys and deriving corresponding public keys. Each user generates their key pair using a common base point G on the curve. The process is as follows:

User A:

- Selects a private key n_A such that $n_A < n$, where n is the order of the base point.
- Computes the public key $P_A = n_A * G$.

User B:

- Selects a private key n_B such that $n_B < n$.
- Computes the public key $P_B = n_B * G$.

The shared secret key is calculated by each user using the other user's public key:

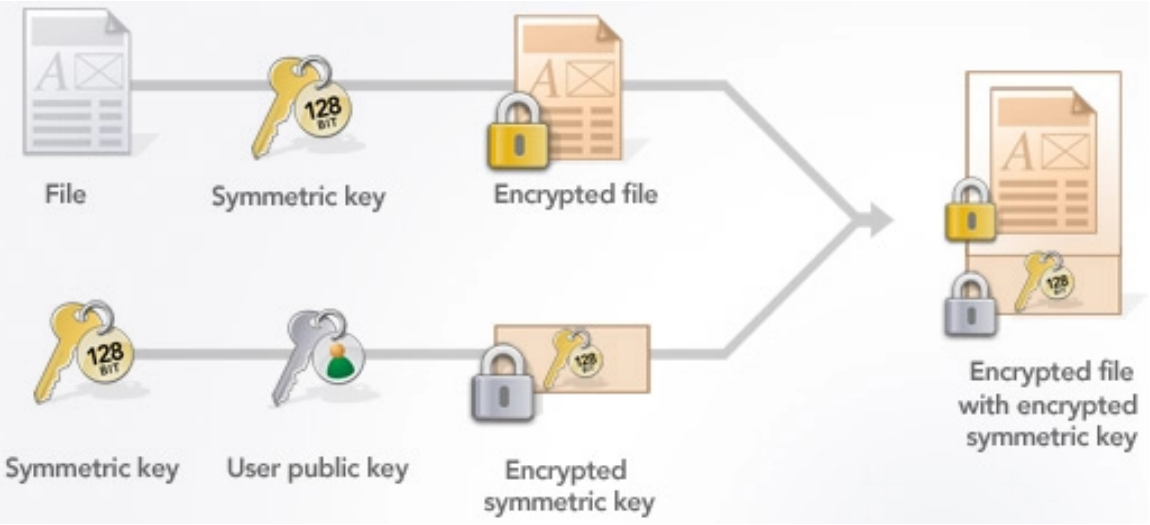
- User A: $K = n_A * P_B$
- User B: $K = n_B * P_A$

Encryption and Decryption

Encryption and decryption in ECC involve encoding plaintext messages into points on the curve and conducting operations to generate ciphertexts and recover plaintexts. The process is as follows:

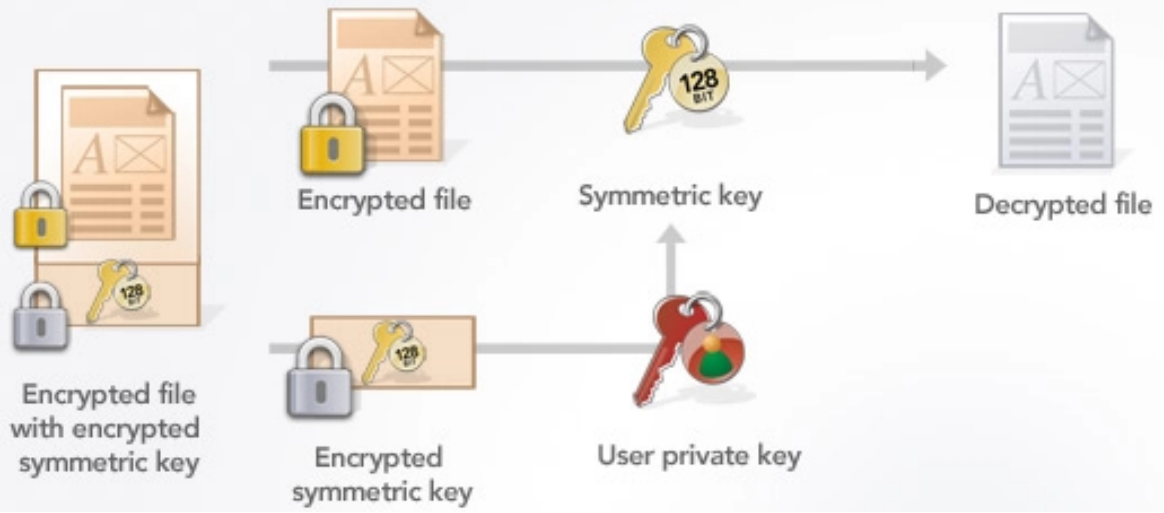
Encryption:

- Encode plaintext message M into a point P_M on the curve.
- Choose a random positive integer k.
- Calculate the cipher point $C_M = \{k * G, P_M + k * P_B\}$.



Decryption:

- Multiply kG * n_B.
- Subtract $P_M + k * P_B - (kG * n_B)$
  we know that $P_B = n_B * G$
  $Pm + k * n_B * G - (kG * n_B) = P_M$



This process ensures secure and efficient communication between users, safeguarding the confidentiality and integrity of transmitted data.

4. Conclusion

Elliptic Curve Cryptography (ECC) stands as a formidable solution for secure digital communication, providing robust security and efficient operations. By leveraging the mathematical properties of elliptic curves within finite fields, ECC offers a balanced approach to encryption and decryption. Its step-by-step implementation ensures secure key generation, encryption, and decryption processes, fostering confidentiality and authenticity in data transmission.

In summary, ECC is a cornerstone of modern cryptography, offering a reliable and scalable solution for safeguarding digital information. Its adoption ensures the integrity and privacy of digital transactions, making it indispensable in today's interconnected world.

5. Learning outcomes

- Gained understanding of the mathematical principles underlying Elliptic Curve Cryptography (ECC).
- Learned the process of key generation, encryption, and decryption in ECC for secure digital communication.
- Understood the significance of finite field operations and point arithmetic in ECC implementations.
- Acquire skills in implementing ECC algorithms for key generation, encryption, and decryption.

6. Source code:

i. Adding points on curve

```
def add_points(P,Q,p):
    x1,y1=P
    x2,y2=Q
    if x1==x2 and y1==y2 :
        beta=(3*x1*x2+a)*pow(2*y1,-1,p)
    else:
        beta=(y2-y1)*pow(x2-x1,-1,p)
    x3=(beta*beta-x1-x2)%p
    y3=(beta*(x1-x3)-y1)%p
    is_on_curve((x3,y3),p)
    return x3,y3
```

ii. Key Generation

```
def make_keypair():
"""Generates a random private-public key pair."""
    private_key = random.randrange(1, curve.n)
    public_key = scalar_mult(private_key, curve.g)
    return private_key, public_key
print("Basepoint:\t", curve.g)

aliceSecretKey, alicePublicKey = make_keypair()
bobSecretKey, bobPublicKey = make_keypair()

print("Alice\'s secret key:\t", aliceSecretKey)
print("Alice\'s public key:\t", alicePublicKey)
print("Bob\'s secret key:\t", bobSecretKey)
print("Bob\'s public key:\t", bobPublicKey)
sharedSecret1 = scalar_mult(bobSecretKey, alicePublicKey)
sharedSecret2 = scalar_mult(aliceSecretKey, bobPublicKey)
print("Alice\'s shared key:\t", sharedSecret1)
print("Bob\'s shared key:\t", sharedSecret2)
print("The shared value is the x-value: \t", (sharedSecret1[0])
```

iii. Encryption - Decryption

```
ka = aliceSecretKey
Qa = apply_double_and_add_method(G = G, k = ka,p= p)
# bob
m= 43424217
s = apply_double_and_add_method(G = G, k =m,p= p)
r = random.getrandbits(128)
c1 = apply_double_and_add_method(G = G, k =r,p= p)
c2= apply_double_and_add_method(G = Qa, k = r, p=p)
c2 = add_points(c2, s, p)
(c1,c2)
c1_prime = (c1[0], (-1*c1[1]) %p)
s_prime=apply_double_and_add_method(G=c1_prime,k=ka,p=p)
s_prime=add_points(P=c2,Q=s_prime,p=p)
s_prime
s
```

References

1. Python Documentation : https://docs.python.org/3/
2. NumPy Documentation : https://numpy.org/doc/
3. Book : Cryptography and Network Security: Principles and Practice 7th Global Edition