Introduction to Elliptic Curve Cryptography

Elliptic-curve cryptography (ECC) is a public-key cryptography method based on the elliptic curve over a finite field , which was well known as smaller, faster, and more efficient than other non-EC cryptography. It was introduced in the 1985 and began to be widely adopted around 2004 and 2005.[1] The study report will introduce the ECC from the following four aspects: Fundamental Concepts of ECC, ECC Introduction based on ECC Diffie-Hellman(ECDH), Coding Experiment Example of ECDH , Advantages and disadvantages of ECC Cryptography, and try to unveil a corner of her mysterious veil.

1. Fundamental Concepts of ECC

1.1 Public-Key Cryptography

Public-key Cryptography, also known as asymmetric cryptography, is a system that uses pairs of related keys: public and private.[2] As their name shows, the public key may be shared widely, and the owner only keeps the private key. The public key is used to encrypt messages, and the private key is used to decrypt messages. ECC is only one type of Public-key cryptography. Diffie-Hellman (DH) and Rivest-Shamir-Alemans (RSA) are also widely used public-key cryptography.

1.2 Trapdoor Function

The foundation of public-key cryptography is a trapdoor function that is easy to compute in one direction but difficult to compute in the opposite direction without particular information. Different types of public-key cryptography have different trapdoor functions. The following figure displays the idea of the trapdoor function: f is the trapdoor function, and t is the trapdoor. It is easy to compute f(x) from x, but the inverse of the f(x), which means known f(x) to get x, is very hard without trapdoor t.

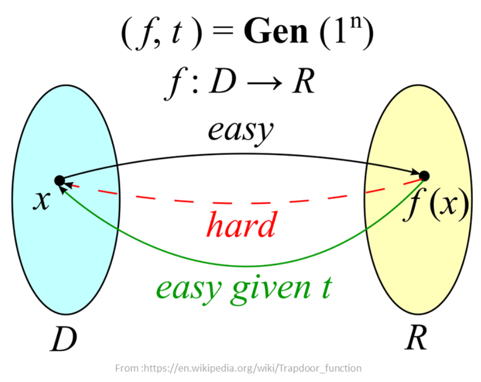
$(f, t) = \mathbf{Gen}(1^n)$
$f : D \to R$
easy
x
$f(x)$
hard
easy given t
D
R
From :https://en.wikipedia.org/wiki/Trapdoor_function

Figure 1: The Idea of Trapdoor Function

1.3 Elliptic Curve

The trapdoor function of ECC is based on an elliptic curve that can be described as equation $y^2 = x^3+ax+b$; a and b are coefficients of the equation, which will change the shape of the curve.[3] Figure 2 right plot is the graphical representation of an elliptic curve in R (real numbers) with a= -1 and b= 1, left plot is the elliptic curve in R with a= -4 and b= 7.

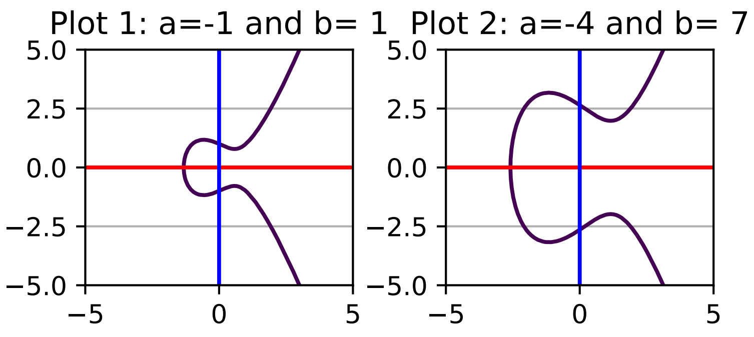

Figure 2: Elliptic Curves of Different Coefficients

The elliptic curve also has an additional point O at the infinity field, which is infinitely high (and low) on the y-axis.[4] Thus, the more exact equation to describe an elliptic curve is

$$E(R) = (x, y) \in R^2 : y^2 = x^3 + ax + b \cup O$$

However, the shape changes when considering an elliptic curve over the complex numbers(C). The curve, denoted by E(C), forms a complex structure that can be visualized as a torus, which may look like the following figure.[4] Since the E(C) is much more complex, the report will not dig into the elliptic curve over the complex number field.

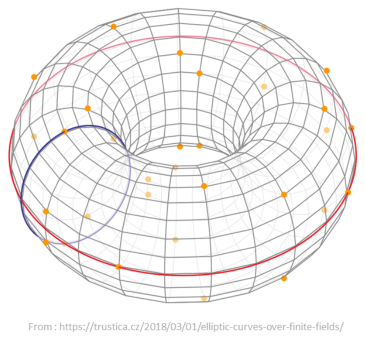
From : https://trustica.cz/2018/03/01/elliptic-curves-over-finite-fields/

Figure 3: Elliptic Curve E(C) of the Weierstrass Form

1.4 Finite Field

As the previous part shows, the elliptic curve with real number is more straightforward to explain and understand, but it is not how cryptography works in the real world. The ECC cryptography was performed on a finite field of integers modulo p, in which p is a prime number. The math function [5] can be described as

$$y^2 \equiv x^3 + ax + b \ (mod\ p)$$

2. Elliptic Curve Cryptograph

2.1 The Operation of Elliptic Curve

The elliptic curve has two properties: one is the curve's horizontal symmetry with the x-axis, and one is any non-vertical line that will intersect the curve in at most three points.[6]

(1) Point Addition

According to the elliptic curve properties, given two points A and B on the curve, the line through A and B will intersect the curve at a third point C, and reflecting this point over the x-axis gives point *-C*, which is also on the curve. We can describe this process as A+B = *-C*. Then, taking another move from point *-C* to point D, the process can be described as *-C* + D = *-E*. These two processes are called point additions on the curve. The following figure shows the previous point addition geometrically.

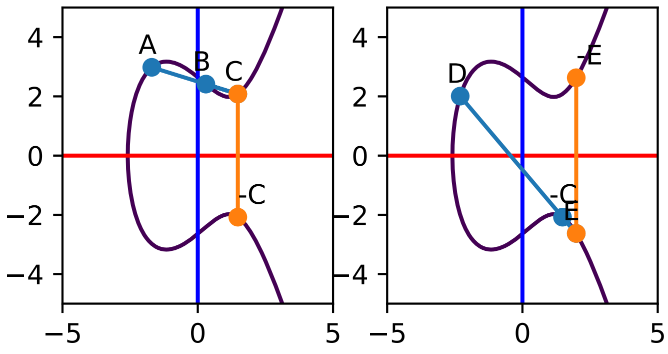

Figure 4: Elliptic Curves Point Addition

The point addition process can be repeated by adding a new point to bounce to a new sum, as long as the new line is not vertical.

To make the whole process easier to repeat and understand, we can take a particular point addition process,given two same points,named P. The line that goes through the P is a tangent line of P, which will yield a new point on the curve and can be described as P+P = 2P. Then, repeat the same step from 2P to the P, yielding a new point 2P+P=3P. In the same additional step, go through points 3P and P,3P+P= 4P. The following figure 5 plot 1- plot 3 show the process geometrically.[7]

(2) Scalar Multiplication as the Trapdoor Function

Theoretically, the process of adding point P to itself can repeat N times to yield an ending point NP. NP= P+P+P+..+P_n =N*P. The operation NP is the point P adds to itself N times,as figure 5 plot 4 shows. Computing NP is straightforward and efficient, but the inverse problem, known as the Elliptic Curve Discrete Logarithm Problem(ECDL), is finding N given start point P and ending Point NP, which is extremely difficult, especially N is big. So, EC scalar multiplication inverse problem (ECDL) be utilized as the trapdoor function.

The security of elliptic curve cryptography mostly depends on the ability to compute a point multiplication and the inability to compute the multiplicand given the original point and product point.

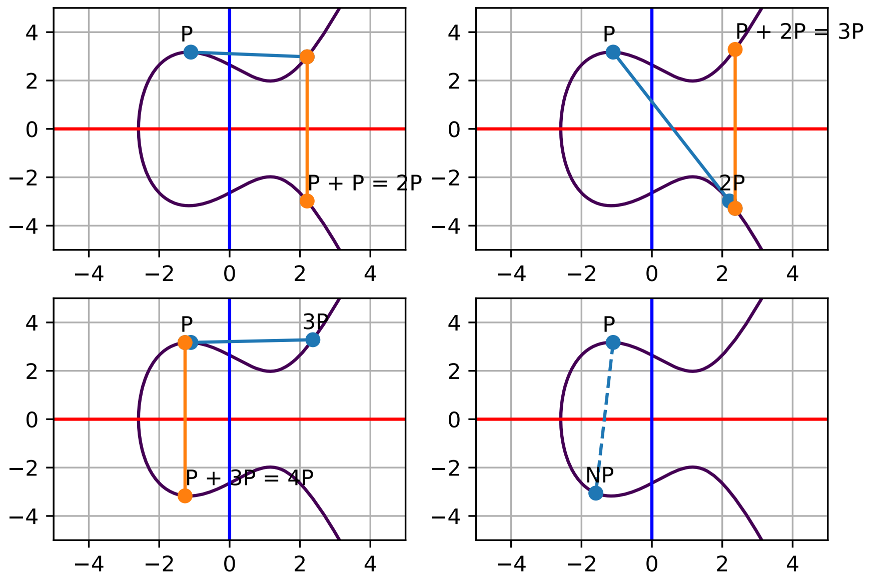

Figure 5: Elliptic Curves Point Addition

2.2 Key Generation

In ECC, a private key is a randomly selected scalar N, and the corresponding public key is the point NP (where P is a predefined point on the curve known to everyone in the system). Due to the complexity of the Elliptic Curve Discrete Logarithm Problem (ECDLP), it's practically impossible for an attacker, knowing only P and the public key NP, to calculate the private key N.

2.3 ECC Implementation

Elliptic curve cryptography widely used in the real world, such as online banking and payments,email encryption and digital signatures,etc.

(1) ECC Algorithms

Elliptic Curve Cryptography (ECC) has a variety of algorithm groups, each tailored for specific applications and all based on elliptic curves over finite fields.

-Digital Signatures: Elliptic Curve Digital Signature Algorithm (ECDSA) and Edwards-curve Digital Signature Algorithm(EdDSA)

-Data Encryption: Elliptic Curve Integrated Encryption Scheme (ECIES) and EC-based ElGamal Elliptic Curve Cryptography

-Key Agreement: Elliptic-curve Diffie-Hellman (ECDH) and Fully Hashed Menezes-Qu-Vanstone(FHMQV)

These algorithms generate keys and encrypt data using specific elliptic curves, such as secp256K1, curve25519,and so on. Their security is anchored in the complexity of the Elliptic Curve Discrete Logarithm Problem (ECDLP), which makes decrypting these algorithms without the proper keys exceedingly difficult, thus ensuring robust network security. The next part will further explain the process of key agreement protocol and message encrypt and decrypt flow based on ECDH .

(2) ECC Work Flow (ECDH)

Alice wants to send a message securely to Bob. They perform ECDH for key exchange and get shared key SK. Firstly, they all agree to use point P as the starting point and select their private key. Alice's private key is N, and Bob selects M as his private key. Then, they leverage a special EC algorithm to yield their own public key NP and MP. NP is the point P after N times of sum operation, and MP is the point of P after M times of sum operation.

They exchange their public key and combine them with their private key to produce a sharing key (SK). Alice takes Bob's public-key MP and adds the point MP to N times: MP+MP+..+MP = N* MP. Bob takes Alices public-key NP, adding the point NP to M times: $NP+NP+..NP = M*NP$. Finally, they will get the same sharing key (SK). $SK = N*MP = M*NP$. For example, if N=7, M=11, $SK = 7*11*P = 11*7*P = 77P$. Both are going to end at 77P Point.

After they both have SK, Alice can use any secure symmetric encryption method (such as Advanced Encryption Standard 256) to encrypt the data with the SK. When Bob receives the data, he can decrypt the encrypted data with the same SK. The following figure shows the ECDH encryption flow.

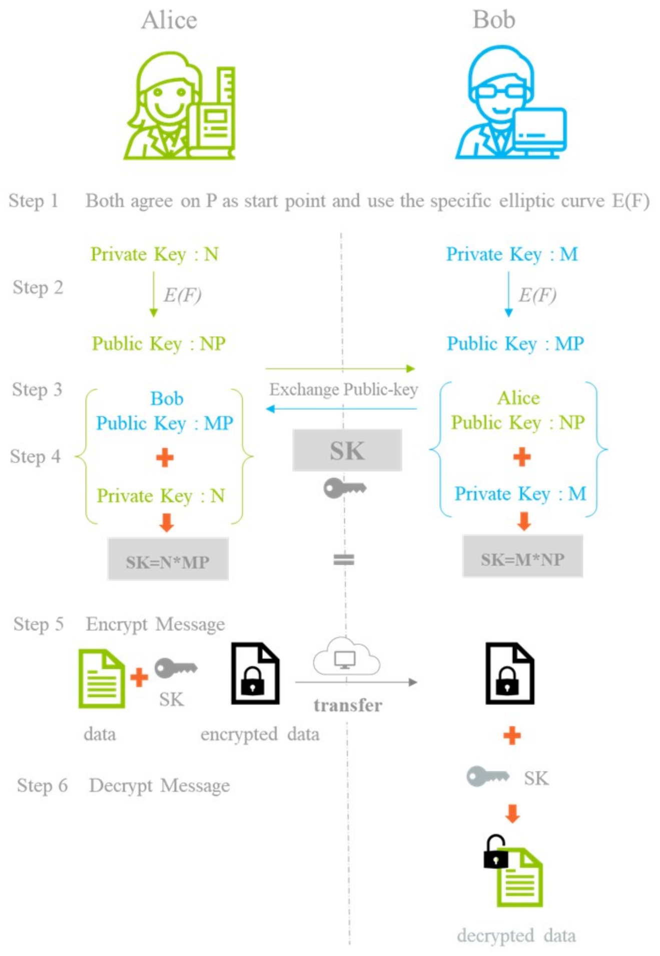

Figure 6: Elliptic Curve Encryption Flow

3. Coding Experiment Example of ECC Application

3.1 Key Generation and Exchange

The following code is an experiment of implementing Elliptic Curve Diffie-Hellman (ECDH) to create a shared key for securely exchanging cryptography keys over a public channel.[8]

(1) Prerequisite Library

-*tinyec* is used for elliptic curve operations.

-*registry* is a Python module working with elliptic curves.

-*secrets* is a Python module for generating strong random numbers suitable for managing data such as passwords, account authentication, security tokens, and related secrets.

(2) Coding

Due to the randomly produced private key, executing the following code, the SK output will differ, but the decryption and encryption keys will remain the same.

```python
from tinyec import registry
import secrets

# Function to convert the point of elliptic curves to a hexadecimal string
# to reduce the size the key

def compress(publicKey):
    return hex(publicKey.x) + hex(publicKey.y % 2)[2:]
# The elliptic curve which is used for the ECDH calculations
# 'brainpoolP256r1' is a standardized elliptic curve.
curve = registry.get_curve('brainpoolP256r1')
# Generation of private key and public key (Alice)
# curve.g is generator point or starting point
PrviateKey_a = secrets.randbelow(curve.field.n)
PublicKey_a = PrviateKey_a * curve.g
print("Public Key of Alice:", compress(PublicKey_a))

# Generation of private key and public key (Bob)
```

```
## Public Key of Alice: 0x6b8d010286716f05999ca10cc5ff9012958a0de930be2adea26e7b8f880ced810
```

```python
PrviateKey_b = secrets.randbelow(curve.field.n)
PublicKey_b = PrviateKey_b * curve.g
print("Public Key of Bob  :", compress(PublicKey_b))
```

```
## Public Key of Bob  : 0x8dca802ea868446c0fa0373925624ca087201615e62350f0e0cc2efdbee8ead1
```

```python
print("Currently exchange the public key (e.g. through the Internet)")

# Generation of shared key and public key (SK)
# (A_SharedKey): represents user Alice
# (B_SharedKey): represents user Bob
```

```
## Currently exchange the public key (e.g. through the Internet)
```

```
A_SharedKey = PrviateKey_a * PublicKey_b
print("A shared key :", compress(A_SharedKey))
```

```
## A shared key : 0x89d9b35e938d2e19758e075dcd71030da4e997f26ea8a4e6cb37fca5d613fe331
```

```
B_SharedKey = PrviateKey_b * PublicKey_a
print("B shared key :", compress(B_SharedKey))

# The both SK are same
```

```
## B shared key : 0x89d9b35e938d2e19758e075dcd71030da4e997f26ea8a4e6cb37fca5d613fe331
```

```
print("Equal shared keys:", A_SharedKey == B_SharedKey)
```

```
## Equal shared keys: True
```

To produce the same result, we can use the following codes. While, In real-world cryptography applications, the randomness should not be predictable, so a fixed seed should not be used.

```
## X: 0xb63ea938c428149e7c376805e4d3266ec333b3b9790b880017aeea9612762511

## Y: 0x94ac9ceb11b74624ef378f6f5e71e862983db1709bbff20e3939159bf06296b71

## Currently exchange the public key (e.g. through the Internet)

## A shared key : 0x754ccb62fb08d101a3305260809e222622ca91efb6723ee1a2023f8d884ff89a0

## B shared key : 0x754ccb62fb08d101a3305260809e222622ca91efb6723ee1a2023f8d884ff89a0

## Equal shared keys: True
```

(3) Detailed output of Shared Key

The output shows the elliptic curve point (x and y coordinates) and the elliptic curve function used.

```
print(A_SharedKey)
```

```
## (62351504031334451819626570586044802465195847422191961497334220066935612374579, 117439610199522025558
```

As the previous part(2) output shows, the hexadecimal representation looks like the following string.

```
compress(A_SharedKey)
```

```
## '0x89d9b35e938d2e19758e075dcd71030da4e997f26ea8a4e6cb37fca5d613fe331'
```

3.2 Encrypt and Decrypte Message

Elliptic Curve Diffie-Hellman (ECDH) is typically used for key exchange rather than directly encrypting and decrypting messages. After having a shared key, a symmetric encryption algorithm, such as AES, can encrypt and decrypt messages.

```python
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
import os
from cryptography.hazmat.primitives import padding

# Updated encrypt function with padding
def encrypt(message, key):
    padder = padding.PKCS7(128).padder()  # 128-bit padding for AES block size
    padded_data = padder.update(message.encode()) + padder.finalize()
    iv = os.urandom(16)  # Generate a random IV
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    ct = encryptor.update(padded_data) + encryptor.finalize()
    return (iv, ct)

# Updated decrypt function with unpadding
def decrypt(iv, ciphertext, key):
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    decrypted_padded_data = decryptor.update(ciphertext) + decryptor.finalize()
    unpadder = padding.PKCS7(128).unpadder()
    return unpadder.update(decrypted_padded_data) + unpadder.finalize()

# Define a salt
salt = os.urandom(16)  # You should generate a secure random salt

# Define shared_secret and other necessary variables
# Replace this with your actual implementation of shared_secret
shared_secret = b'\x01' * 32  # Replace with your shared_secret

# Derive a symmetric key from the shared secret
hkdf = HKDF(
    algorithm=hashes.SHA256(),
    length=32,  # Length for AES-256 key
    salt=salt,
    info=b'handshake data',
    backend=default_backend())
aes_key = hkdf.derive(shared_secret)

# Encrypting the message "hello, wonderful world!"
iv, ciphertext = encrypt("hello, wonderful world!", aes_key)

# Decrypting the message
decrypted_message = decrypt(iv, ciphertext, aes_key).decode()
print(f"Encrypted: {ciphertext}")
```

```
## Encrypted: b'\xf9fX\x84\xa4\x06\x01\xe8j\xad\xe4\x8b\x90\x15\x1c.\xf7\xb7z\x11\xf6h\xd3AC\xc4\xec.\x
```

```python
print(f"Decrypted: {decrypted_message}")
```

```
## Decrypted: hello, wonderful world!
```

4. Advantage and Disadvantage of ECC

4.1 Advantages of ECC

(1) Smaller Key Sizes

ECC offers a high level of security with relatively small key sizes compared to other public-key systems like RSA. This leads to less computational burden and lower power consumption. The following table shows the key size comparison of ECC vs RSA.[9]

Security (Bits)	RSA required key size	ECC required key size
80	1024	160-223
128	3072	256-383
256	15360	512+

(2) Efficiency and Bandwidth Saving

Due to smaller key sizes, ECC operations are generally faster, which is beneficial in environments with limited processing power and memory, such as mobile devices and IoT devices. Smaller key sizes mean less data is transmitted during cryptographic operations, saving bandwidth.

(3) Strong Security

ECC provides robust security against various cryptographic attacks. The difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP) is the core of its security.ECC is considered resistant to the most powerful forms of cryptographic attack that are effective against other forms of public-key cryptography, such as RSA or Diffie-Hellman.

(4) Widespread Industry Adoption

ECC is widely adopted in various security protocols and standards, including SSL/TLS for secure web browsing and in many modern encryption systems.

4.2 Disadvantages of ECC

(1) Implementation Complexity

The mathematical inner workings of ECC cryptography and cryptanalysis security (e.g., the Weierstrass equation that describes elliptical curves, group theory, quadratic twists, quantum mechanics behind the Shor attack and the elliptic-curve discrete-logarithm problem) are complex.[10] Implementing ECC correctly requires a deep understanding of its mathematical foundations and potential vulnerabilities. Incorrect implementation can lead to significant security flaws.In many IT departments, there might be a lack of familiarity with ECC compared to more traditional methods like RSA. This can make troubleshooting, maintenance, and effective implementation more challenging.

(2)Patent Issues

Some aspects of ECC were patented, which hindered its adoption. Although most of these patents have expired, it initially slowed the widespread acceptance of ECC in cryptography implementations.

(3)Security Issues

There are some potential vulnerabilities to elliptic curves, such as side-channel attacks and twist-security attacks. These attacks threaten to invalidate the security ECC aims to provide for private keys. Incorrect implementation of ECC will produce significant vulnerabilities in the cryptography software.[10] Meanwhile, like other public-key cryptosystems, ECC could be broken by quantum computers.

In conclusion, ECC is a high secure and efficient form of cryptography when properly implemented and used, and it is a critical component of modern cryptographic practices. However, as with any cryptographic system, its security is contingent on various factors, including algorithm choice, key size, implementation, and the evolving of cryptography and computing capabilities.

Reference

1. Wikipedia.*Elliptic-curve Cryptography.* Archived from https://en.wikipedia.org/wiki/Elliptic-curve_cryptography on 2023-12-04.
2. Wikipedia.*Public-key Cryptography.* Archived from https://en.wikipedia.org/wiki/Public-key_cryptography on 2023-12-04.
3. Chunck Easttom (2016),*Modern Cryptography Applied Mathematics*,p254.
4. Gaël Foppolo (Jun 2018),*Elliptic-curve cryptography (ECC).* Archived from https://medium.com/@gaelfoppolo/elliptic-curve-cryptography-ecc-6af6142b0dd1.
5. Svetlin Nakov (2018), *Elliptic Curve Cryptography (ECC)*, Practical Cryptography for Developers. Archived from https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc
6. Wikipedia.*Elliptic Curve.* Archived from https://en.wikipedia.org/wiki/Elliptic_curve
7. Fangpen lin. Archived from https://github.com/fangpenlin/elliptic-curve-explained/blob/master/elliptic-curve.ipynb on 2023-12-04.
8. Monika,Tanya Tomar (March 2020),*Implementation of Elliptic - Curvecryptography*,International Journal of Electrical Engineering and Technology (IJEET), p181. Archived from https://iaeme.com/MasterAdmin/Journal_uploads/IJEET/VOLUME_11_ISSUE_2/IJEET_11_02_022.pdf
9. *Blockchain - Elliptic Curve Cryptography*,Nov 2022.Archived from https://www.geeksforgeeks.org/blockchain-elliptic-curve-cryptography/
10. Veronika Stolbikova (May 2016), *Can Elliptic Curve Cryptography be Trusted? A Brief Analysis of the Security of a Popular Cryptosystem.* Archived from https://www.isaca.org/resources/isaca-journal/issues/2016/volume-3/can-elliptic-curve-cryptography-be-trusted-a-brief-analysis-of-the-security-of-a-popular-cryptosyste