Elliptic Curve Cryptography

Xin JIANG @PPLabs

June 5 2019

About Us

PPIO is a decentralized platform for developers who value speed, affordability, and privacy in data storage and distribution.

Xin JIANG, PPIO Blockchains Senior Engineer with rich development experiences of Andriod system, security system, and blockchains.
Master degree of Nanjing University.

Agenda

1. Background
2. What's Elliptic Curve
3. Groups
4. The group law for elliptic curves
5. Point addition
   - 5.1 Geometric addition
   - 5.2 Algebraic addition
   - 5.3 Speeding up addition
6. The update elliptic curve model
7. Domain parameters
8. Discrete logarithm
9. Elliptic Curve Cryptography
   - 9.1 ECDH
   - 9.2 ECDSA
10. RSA vs ECC
11. References

Background

Many blockchains use ECC instead of RSA, why?

Many new technologies are based on ECC, such as Ring Signature, zkSNARKs.

Some technologies are related to ECC, such as BLS Signature.

.....

We need to know how it works.

Background - RSA

In RSA, this maximum value (***n***) is obtained by multiplying two random prime numbers. The public and private keys are two specially chosen numbers that are greater than zero and less than the ***n***, call them ***pub*** and ***priv***. To encrypt a number you multiply it by itself ***pub*** times, making sure to wrap around when you hit the maximum. To decrypt a message, you multiply it by itself ***priv*** times and you get back to the original number. It sounds surprising, but it actually works. This property was a big breakthrough when it was discovered.

1. Generate two distinct random prime ***p*** and ***q***, and calculate ***n = pq***
2. Calculate ***φ = φ(n) = (p-1)(q-1)***
3. Choose ***e***, arbitrary, but less than ***n*** and relatively prime to ***φ(n)***
4. Calculate ***d***, inverse of ***e*** modulo ***φ***: ***ed mod φ = 1***
5. ***pub: (e, n), priv: (d, n)***

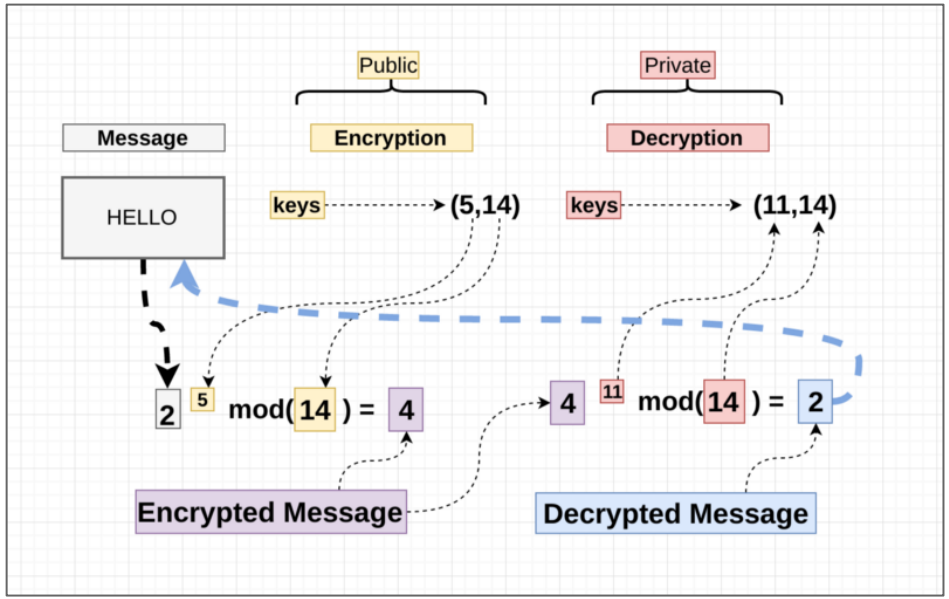
Background - RSA

Encryption

$$c = m^e \pmod n$$

Decription

$$m = c^d \pmod n$$



Proof

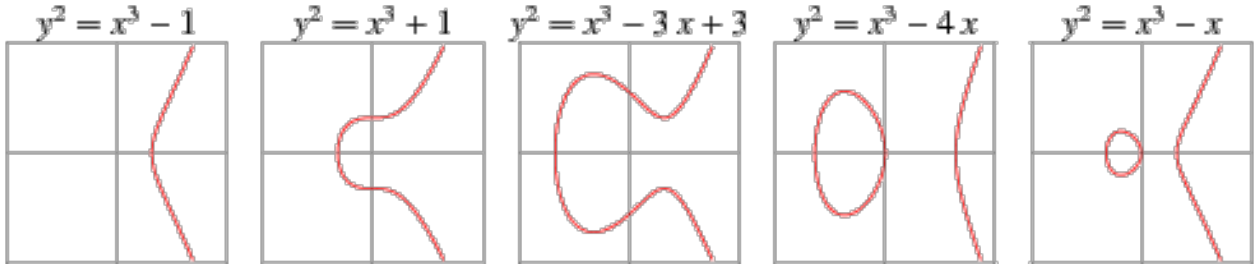
$$m = c^d \pmod n = m^{ed} \pmod n = m^{k\varphi+1} \pmod n = m \cdot (m^{\varphi})^k \pmod n$$

because $m^{\varphi} = 1 \pmod n$ *(Euler Theorem)*

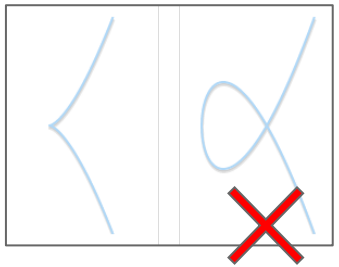
therefore $m = m \cdot 1^k \pmod n = m \pmod n$

What's Elliptic Curve

$$y^2 = x^3+ax+b$$



$4a^3+27b^2 \neq 0$ (to exclude singular curves)



Groups

A **group** in mathematics is a set for which we have defined a binary operation that we call "addition" and indicate with the symbol $+$.
In order for the set G to be a group, addition must defined so that it respects the following four properties:

Closure: if a and b are members of G, then $a + b$ is a member of G;

Associativity: $(a + b) + c = a + (b + c)$;

There exists an **identity element** 0 such that $a + 0 = 0 + a = a$;

every element has an **inverse**, that is: for every a there exists b such that $a + b = 0$.

If we add a fifth requirement:

Commutativity: $a + b = a + b$,

then the group is called **abelian** group.

Groups

The set of integer numbers ***Z*** is an abelian group.

The set of natural numbers ***N*** however is not a group, as the fourth property can't be satisfied.

Some properties of groups:

- the **identity element** is unique; also the **inverses** are unique
- that is: for every a there exists only one b such that $a + b = a + b = 0$ (and we can write b as $-a$).

The group law for elliptic curves

We can **define** a group over elliptic curves. Specifically:

the elements of the group are the points of an elliptic curve;

the **identity element** is the point at infinity ***O***;

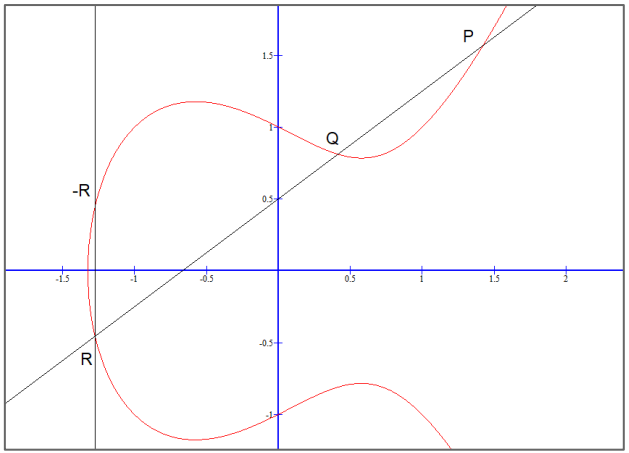
the **inverse** of a point ***P*** is the one symmetric about the ***x-axis***;

addition is given by the following rule:

given three aligned,

non-zero points ***P***, ***Q*** and ***R***,

their sum is ***P*** + ***Q*** + ***R*** = ***O***.



The group law for elliptic curves

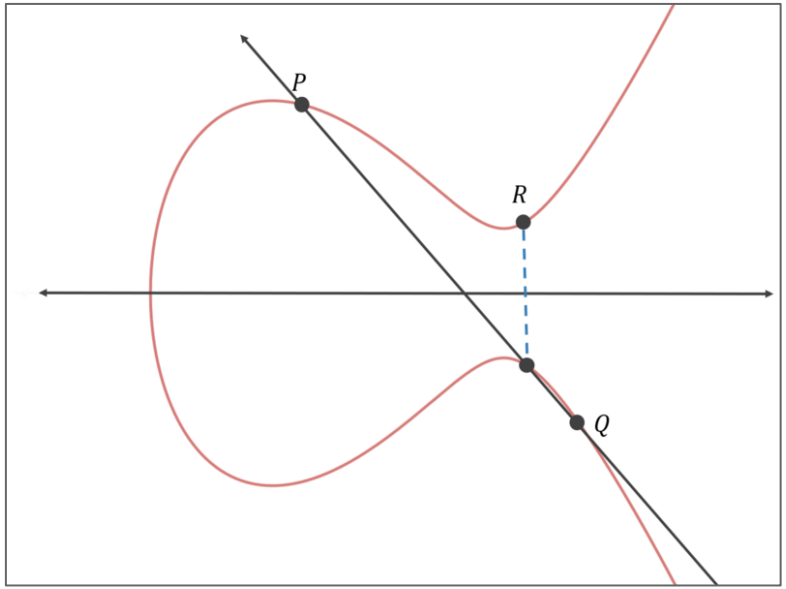
Note that with the last rule, we only require three aligned points, and three points are aligned without respect to order.

This means that, if P, Q and R are aligned, then $P+(Q+R)=Q+(P+R)=R+(P+Q)=\cdots=0$.

This way, we have intuitively proved that our + operator is both **associative** and **commutative**: we are in an **abelian** group.

Geometric addition

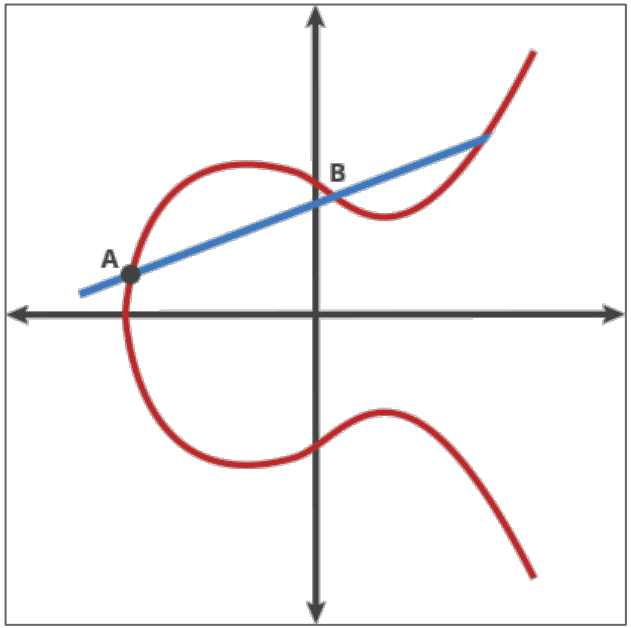
$P + Q = R$



$A + B = C$

$A + C = D$

$A + D = E$

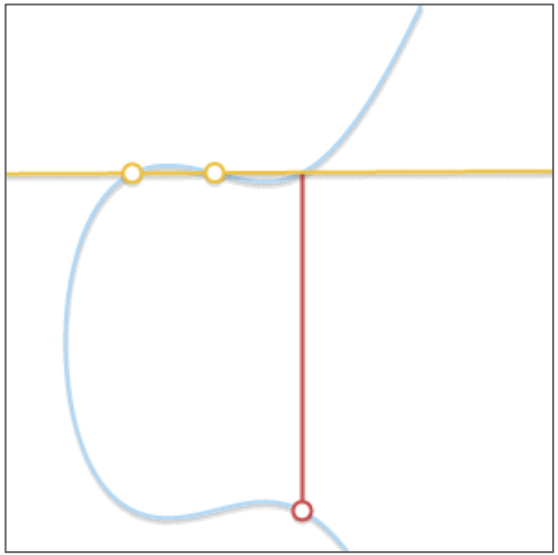


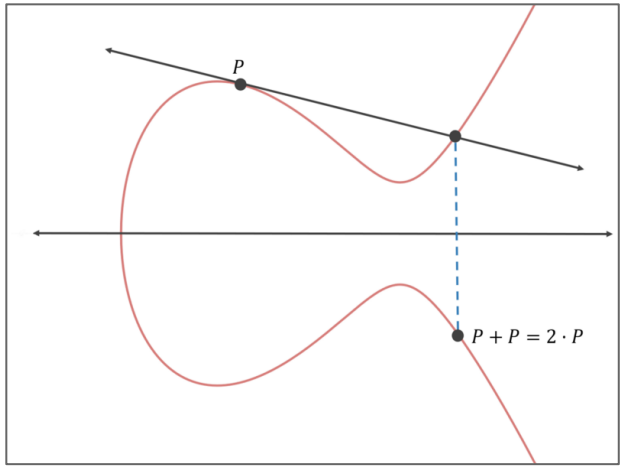
Geometric addition

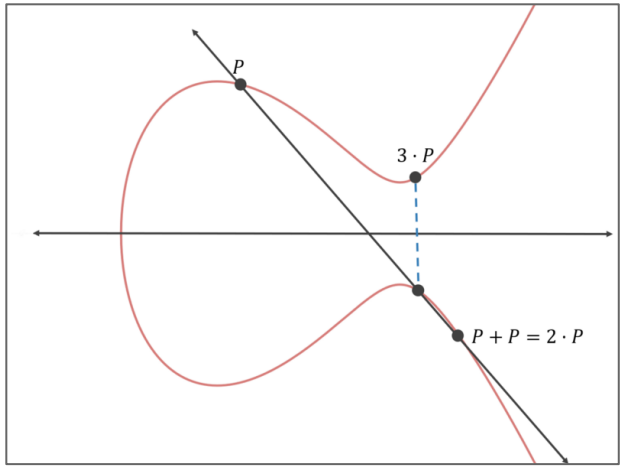
$P + P = 2P$

$P + 2P = 3P$

.....







Algebraic addition

Two non-zero, non-symmetric points $P=(x_P,y_P)$ and $Q=(x_Q,y_Q)$.

If P and Q are distinct $(x_P \neq x_Q)$, the line through them has **slope**:

$$m=(y_P-y_Q)/(x_P-x_Q)$$

The intersection of this line with the elliptic curve is a third point $R=(x_R,y_R)$:

$$x_R = m^2 - x_P - x_Q$$

$$y_R = y_P + m(x_R - x_P)$$

Algebraic addition

If $P = Q$, we must use a diferent equation for the slope:

$$m = (3x_P^2 + a)/(2y_P)$$

The intersection of this line with the elliptic curve is a third point $R=(x_R,y_R)$:

$$x_R = m^2 - 2x_P$$

$$y_R = y_P + m(x_R - x_P)$$

Speeding up addition

$aP = a_n 2^n P + \ldots + a_j 2^j P + \ldots a_o 2^o P$ $(a_i = 1 \text{ or } 0,\ i = 0, \ldots n)$

1. calculate $2^i P = 2^{i-1} P + 2^{i-1} P$
2. calculate $a_n 2^n P + \ldots + a_j 2^j P + \ldots a_o 2^o P$

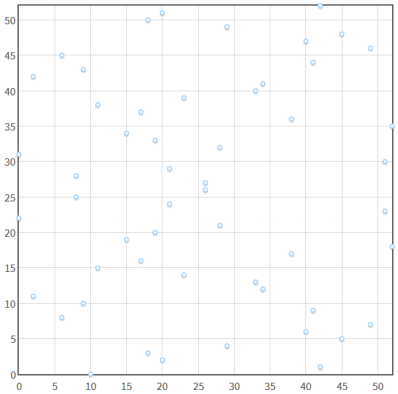
The update elliptic curve model

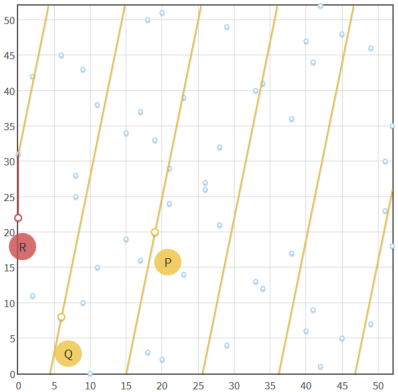
$$y^2 = x^3+ax+b \quad \Rightarrow \quad y^2 \bmod p = (x^3 + ax + b) \bmod p$$

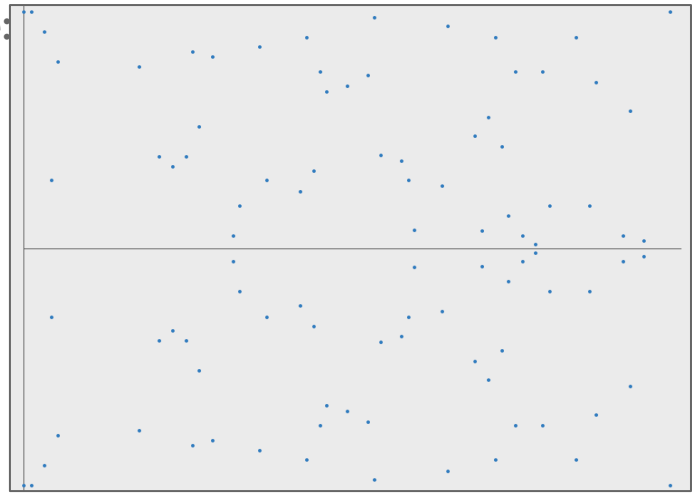
$\mathbf{p}$ is some prime number.

In secp256k1, $\mathbf{p}$ is the largest prime that is smaller than 2^{256}.

The previous elliptic curve now looks something like:







Domain parameters

Our elliptic curve algorithms will work in a cyclic subgroup of an elliptic curve over a finite field. Therefore, our algorithms will need the following parameters:

The **prime** p that specifies the size of the finite field.

The **coefficients** a and b of the elliptic curve equation.

The **base point** G that generates our subgroup.

The **order** n of the subgrouop.

The **cofactor** h of the subgroup.

In conclusion, the **domain parameters** for our algorithms are the **sextuple** (p,a,b,G,n,h).

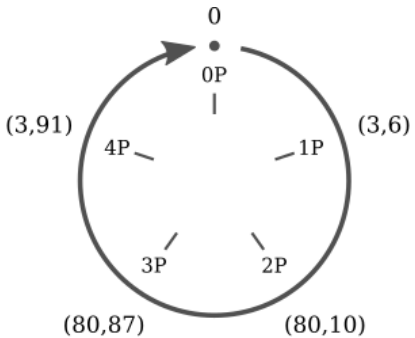
Subgroup order n

$nP=P+P+\cdots+P$ (n times)

The order of P is the smallest positive integer n such that $nP=O$

The point P is called generator or base point of the cyclic subgroup $\{O, P, 2P....(n-1)P\}$

$y^2 \equiv x^3+2x+3 \pmod{97}$ and the point $P=(3,6)$. The order of P is 5



Subgroup cofactor h

$h = N/n$ (h is always an integer, because n is a divisor of N by Lagrange's theorem)

N : The elliptic curve group order

n: The subgroup order of generator P

The requirement for domain parameters

p, more bigger more better, but more slower (~200 bits)

n must be a prime number

$h \leq 4,\ p \neq n \times h,\ pt \neq 1 \pmod{n}\ (1 \leq t < 20)$

$4a^3 + 27b^2 \neq 0 \pmod{p}$

secp256k1

p = 0xffffffff ffffffff ffffffff ffffffff ffffffff ffffffff fffffffe fffffc2f (the largest prime that is smaller than 2^{256})

a = 0

b = 7

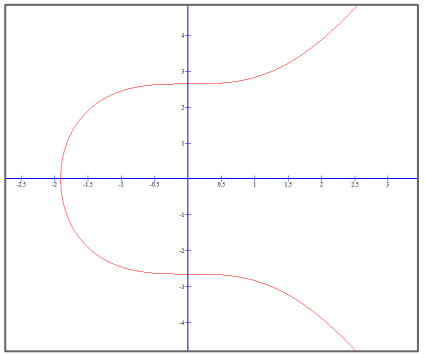
x_G = 0x79be667e f9dcbbac 55a06295 ce870b07 029bfcdb 2dce28d9 59f2815b 16f81798

y_G = 0x483ada77 26a3c465 5da4fbfc 0e1108a8 fd17b448 a6855419 9c47d08f fb10d4b8

n = 0xffffffff ffffffff ffffffff fffffffe baaedce6 af48a03b bfd25e8c d0364141

h = 1

Bitcoin, Ethereum and many other blockchains use **secp256k1**



Discrete logarithm

- If we know P and Q, what is k such that $Q=kP$?
- This problem, which is known as the **discrete logarithm problem** for elliptic curves, is believed to be a "**hard**" problem, in that there is no known **polynomial time** algorithm that can run on a classical computer.
- This problem is also analogous to the discrete logarithm problem used with other cryptosystems such as the **Digital Signature Algorithm (DSA)**, the **Diffie-Hellman key exchange (D-H)** and the **ElGamal** algorithm. (if we know a and b, what's k such that $b=a^k \bmod p$?)

Elliptic Curve Cryptography

- The private key is a random integer d chosen from $\{1,\ldots,n-1\}$ (where n is the order of the subgroup).
- The public key is the point $H=dG$ (where G is the base point of the subgroup).
- If we know d and G (along with the other domain parameters), finding H is "easy". But if we know H and G, finding the private key d is "hard", because it requires us to solve the discrete logarithm problem.
- Now we are going to describe two public-key algorithms based on that: ECDH (Elliptic curve Diffie-Hellman), which is used for encryption, and ECDSA (Elliptic Curve Digital Signature Algorithm), used for digital signing.

ECDH

- ECDH is short for Elliptic Curve Diffie-Hellman, which is used for encryption
- The problem it solves is the following: two parties (the usual Alice and Bob) want to **exchange information securely**, so that a third party (the Man In the Middle) may intercept them, but may not decode them.
- Usually, ECDH is used to exchange symmetric encryption keys

ECDH

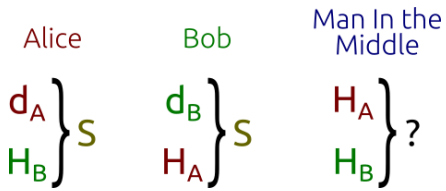
First, Alice and Bob generate their own private d and public keys H.

$H_A=d_AG$ for Alice, $H_B=d_BG$ for Bob.

Alice and Bob exchange their public keys H_A and H_B over an insecure channel.

Alice calculates $S=d_AH_B$ and Bob calculates $S=d_BH_A$

$S=d_AH_B=d_A(d_BG)=d_B(d_AG)=d_BH_A$



The Man In the Middle, would not be able to find out the shared secret S.

Now that Alice and Bob have obtained the shared secret, they can exchange data with symmetric encryption.

ECDHE

Some of you may have heard of **ECDHE** instead of **ECDH**.
The "**E**" in **ECHDE** stands for "**Ephemeral**" and refers to the fact that the keys exchanged are **temporary**, rather than **static**.
For example in **TLS**.

ECDSA

ECDSA is short for **Elliptic Curve Digital Signature Algorithm**

The scenario is the following:

Alice wants to sign a message with her private key (d_A), and Bob wants to validate the signature using Alice's public key (H_A).

Nobody but Alice should be able to produce valid signatures.

Everyone should be able to check signatures.

ECDSA - signing

ECDSA is short for **Elliptic Curve Digital Signature Algorithm**

ECDSA works on the **hash** of the message, rather than on the message itself. **The hash of the message ought to be truncated** so that the bit length of the hash is the same as the bit length of n (the order of the subgroup). The truncated hash is an integer and will be denoted as z.

The algorithm performed by Alice to sign the message works as follows:

1. Take a random integer k chosen from $\{1,\ldots,n-1\}$ (where n is still the subgroup order).
2. Calculate the point $P=kG$ (where G is the base point of the subgroup).
3. Calculate the number $r = x_P \bmod n$ (where x_P is the x coordinate of P).
4. If $r = 0$, then choose another k and try again.
5. Calculate $s=k^{-1}(z+rd_A) \bmod n$ (where d_A is Alice's private key and k^{-1} is the multiplicative inverse of k modulo n).
6. If $s=0$, then choose another k and try again.

The pair (r,s) is the signature.

ECDSA - verifying

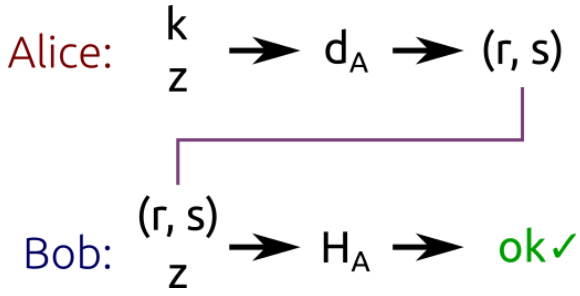
In plain words, this algorithm first generates a secret (k). This secret is hidden in r thanks to point multiplication. r is then bound to the message hash by the equation $s=k^{-1}(z+rd_A) \bmod n$.

In order to verify the signature we'll need Alice's public key H_A, the (truncated) hash z and, obviously, the signature (r,s).

Calculate the integer $u_1 = s^{-1}z \bmod n$.

Calculate the integer $u_2 = s^{-1}r \bmod n$.

Calculate the point $P = u_1G+u_2H_A$.



The signature is valid only if $r = x_P \bmod n$.

$$P = u_1G+u_2H_A = u_1G+u_2d_AG = (u_1+u_2d_A)G = (s^{-1}z + s^{-1}rd_A)G = s^{-1}(z + rd_A)G$$

because $s=k^{-1}(z+rd_A)$ mod n, therefore $k=s^{-1}(z + rd_A)$

$$P = kG$$

If a subgroup has a non-prime order, ECDSA can't be used.

ECDSA - The importance of k

When generating ECDSA signatures, it is important to keep the secret k really secret. If we used the same k for all signatures, or if our random number generator were somewhat predictable, an attacker would be able to find out the private key!

This is the kind of mistake made by Sony a few years ago. Basically, the PlayStation 3 game console can run only games signed by Sony with ECDSA. The problem is: all the signatures made by Sony were generated using a static k.

In this situation, we could easily recover Sony's private key d_S by buying just two signed games, extracting their hashes (z_1 and z_2) and their signatures ((r_1,s_1) and (r_2,s_2)), together with the domain parameters. Here's how:

First off, note that $r_1=r_2$ (because $r=x_P \bmod n$ and $P=kG$ is the same for both signatures).

Consider that $(s_1-s_2) \bmod n = k^{-1}(z_1-z_2) \bmod n$ (this result comes directly from the equation for s).

Now multiply each side of the equation by k: $k(s_1-s_2) \bmod n=(z_1-z_2) \bmod n$.

Divide by $(s_1-s_2)$to get $k=(z_1-z_2)(s_1-s_2)^{-1} \bmod n$.

The last equation lets us calculate k using only two hashes and their corresponding signatures. Now we can extract the private key using the equation for s:

$$s=k^{-1}(z+rd_S) \bmod n \Rightarrow d_S=r^{-1}(sk-z) \bmod n$$

Similar techniques may be employed if k is not static but predictable in some way.

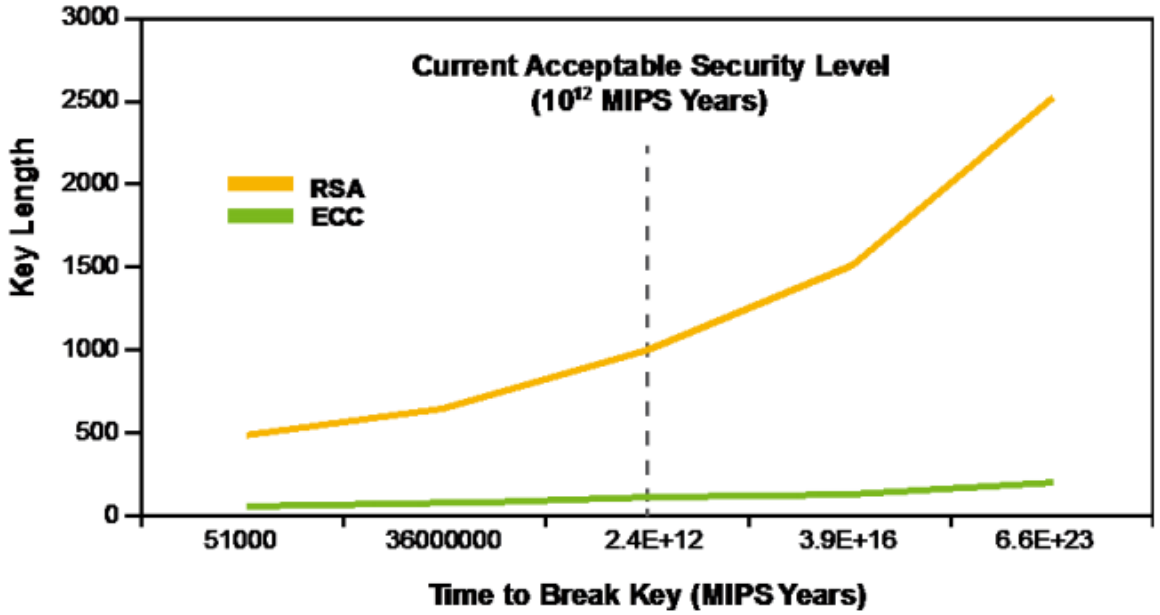
RSA vs ECC

Security Comparison for Various Algorithm-key Size Combinations

Security Bits	Symmetric Encryption Algorithm	Minimum Size (bits) of Public Keys	
		RSA	ECC
80	Skipjack	1024	160
112	3DES	2048	224
128	AES-128	3072	256
192	AES-192	7680	384
256	AES-256	15360	512

RSA vs ECC

RSA and ECC Performance



This chart presents another way to look at the performance of RSA and ECC. It compares what key lengths of each algorithm will provide a level of security measured in the time in MIPS-years to break the security. It is clear that ECC is more efficient.

RSA vs ECC

- To visualize how much harder it is to break, Lenstra recently introduced the concept of "**Global Security**." You can compute how much energy is needed to break a cryptographic algorithm, and compare that with how much water that energy could boil. This is a kind of cryptographic carbon footprint.
- By this measure, breaking a **228-bit RSA** key requires less energy to than it takes to boil **a teaspoon of water**. Comparatively, breaking a **228-bit elliptic curve** key requires enough energy to boil **all the water on earth**. For this level of security with RSA, you'd need a key with **2,380-bits**.

RSA vs ECC

Advantage of RSA:

- Well established.

Advantages of elliptic curve:

- Shorter keys are as strong as long key for RSA
- Low on CPU consumption.
- Low on memory usage.

References

Primer on Elliptic Curve Cryptography - 中文翻译版

椭圆曲线密码算法（ECC）

ECC椭圆曲线详解(有具体实例)

Elliptic Curve Cryptography: a gentle introduction

HTML5/JavaScript visual tool

An Introduction to the Theory of Elliptic Curves

What is the math behind elliptic curve cryptography?

RSA vs ECC Comparison for Embedded Systems

Public-Key Encryption by RSA Algorithm Objective

Exercise

Calculate the address of an ethereum private key

1. private_key → public_key
2. Keccak256(public_key) → hash
3. hash[12:] → address

PPIO/ppio-blockchain-code-talks/ecc

Thanks for listening!

官方网站: pp.io

微信: PPIO

知乎: PPIO

GitHub: PPIO

Medium: PPIO

让数据存储分发更便宜、更快速、更安全