**Elliptical Curve Cryptography**

**History [1]:**

Elliptical curves date back to the times in ancient Greece, there origins are wrapped up in the age of Number Theory and the dawn of arithmetic.

Elliptical curves did not see their way into the space of cryptography until their use was suggested by Neal Koblitz and Victor S. Miller in 1985.

They would not see a wider case of use until 2004 -2005, and their most prominent use would not be utilised until 2009 by Satoshi Nakamoto.

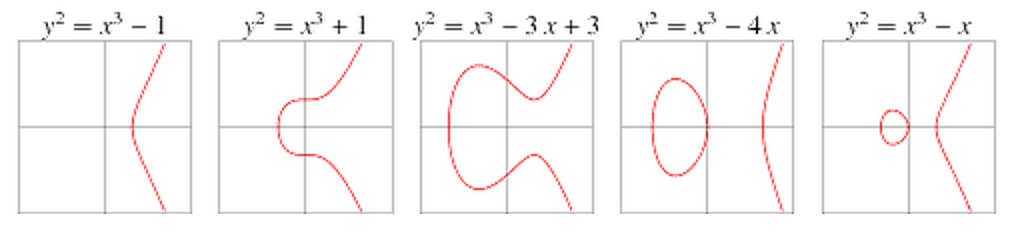
**What is an Elliptical Curve [3]:**

An elliptical curve on an x y plane consists of all the points that satisfy the following equation,

y² = x³ + ax + b

where 4a³ + 27b² ≠ 0, this is needed to avoid any singular points. A singular point is when an algebraic curve has a point of self-intersection or a cusp. All elliptical curves are symmetrical on the x-axis.

Below are a few images of elliptical curves



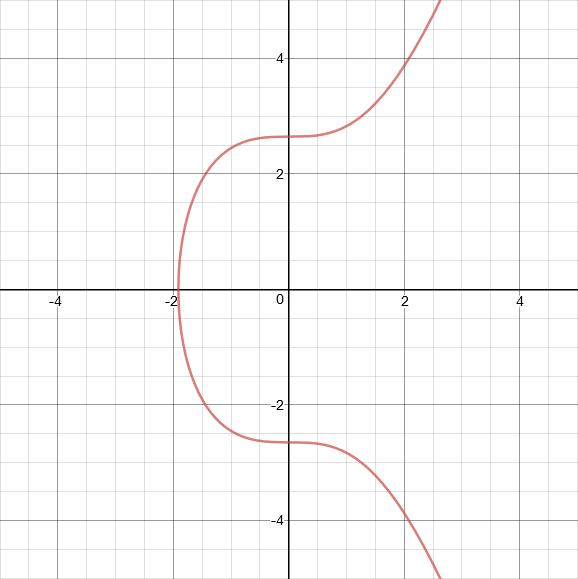
**A now famous use for elliptical curve cryptography [2][4]:**

To best explain how elliptical curve cryptography works, looking at its use in cryptocurrency blockchains is the best option. Bitcoin and Ethereum along with many other cryptocurrencies use the elliptic curve called secp256k1.

The equation for this particular curve is y² = x³+7,

With Bitcoin being one of the very first cryptocurrencies, it’s safe to assume it was one of the first use cases for this curve in the blockchain space. Its been cited in many sources that Satoshi Nakamotos choice to use this curve was for no particular reason.

Below is an image of the Bitcoin elliptical curve.



**The workings of the Elliptical Curve [2]:**

The first note will be towards point addition, if a point P to a point Q on the curve is added, the sum will be a third point on the curve, a point R.

P + Q = R

Two points need to be picked on the curve and then a line that goes through the two points is found, next where the line intersects the curve at a third point needs to be determined. The third point then needs to be reflected across the x-axis, this can be done by multiplying the y-coordinate by -1.

The point that is obtained from following those steps should be the same R found in P + Q = R.

In Elliptical Curve Cryptography a base point P on the curve is specified and is then added to itself, adding two arbitrary points together is not practiced in this area. To add P to itself the equation of the line that goes through P and P must be found. There is an infinite number of these lines but in this example the tangent line will be used.

Once this is done, the third point that the line intersects can be found and reflected across the x-axis, giving the result,

P + P = 2\*P

P can be added to itself again if desired, repeating this will cause the computing of P + P + P, the result will be 3\*P. Alternatively P and 2\*P can be added to give the result 3\*P.

P can be continuously added to itself to compute 4\*P, 5\*P, 6\*P, 7\*P and so on.

This point addition method can get quite slow when larger repeats of adding P to itself is reached. A small example would be P\*10 needing nine point addition operations, to speed this up instead of calculating P + P + P + P + P + P + P + P + P + P,

First calculate P + P = 2\*P

Next instead of P + P + P or P + P + 2\*P use 2\*P + 2\*P = 4\*P

Then do 4\*P + 4\*P = 8\*P

Finally do 2\*P + 8\*P = 10\*P

Taking the best approach will lower the addition operations to 4 instead of 9 which is a lot more efficient.

Note that on paper methods will not work in a computing system so an equation n\*P + r\*P = (n + r)\*P = q\*P will not work. To show using numbers make n = 4, r = 6 and q = n + r.

Shortening the points of addition operations is important when the amount of steps needed starts to head towards large numbers, consider how many steps it would take to compute x\*P, where x is a random 256-bit integer.

In this instance x can range anywhere from 0 to 1.1579209e+77, if x is random it’ll never take more than 510 point addition operations, if the series 2^0\*P, 2^1\*P,…..,2^255\*P is computed, it can be calculated with 255 point addition operations. Because there is 256 points the system can move from one point to the next by adding the current point to itself.

Note that the quicker method cannot be used because x is random, if the goal was to reach 2^255\*P by using the smallest amount of addition operations it would not take 255 point addition operations.

Next the binary expansion of x must be found, assuming x is found to be equal 246, the binary will be 11110110 making the binary expansion be 2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1. The binary expansion must then be multiplied by P giving the following expression 2^7\*P + 2^6\*P + 2^5\*P + 2^4\*P + 2^2\*P + 2^1\*P, these individual points do not need to be calculated as they would have been calculated prior to the binary expansion.

Similarly the binary expansion of x will contain 256 elements 2^0 -> 2^255 so no more than 255 point addition operations will be required. Decimal point addition operations plus binary expansion point addition operations is:

255 + 255 = 510

**Private and Public Keys in Elliptic Curve Cryptography [2]:**

If x\*P where x is a random 256-bit integer is computed, the result will be some point on the curve, label that point K. If K was publicly distributed, would it be possible to determine what x was? Would it be possible to determine how many times P was added to itself to get to K?

While in theory it is possible, it is not feasible to figure out what x is, even knowing what P is, the curve being used and access to a super computer. No algorithm exists to find x, the only existing options are to keep adding P to itself until K is found or to keep subtracting P from K until you get P.

To give x an average it’ll be somewhere between 0 -> 2^256-1, so around 2^128, it will take an average of 2^128 point addition operations to find x no matter what approach is taken. To put a perspective of time and computational power on this, if since the beginning of the universe until now a computer was doing one trillion point addition operations per second, only 2^98 point addition operations would be completed, likely leaving x still unknown.

Note the starting point for finding x does not matter as x is completely random.

Knowing that x cannot be figured out when K is given, where K = x\*P, it is then logical to make x the private key and K the public key. So the private key will be a random 256-bit integer and a public key will be the x and y coordinates of a point on an elliptic curve.

To satisfy the properties of public and private keys, it must be computationally infeasible to derive a private key which corresponds to a given public key. Elliptical Curve Cryptography satisfies this.

With all that has been discussed, the current elliptical curve model is met with a problem, the current model x\*P could result with a points x- and y-coordinates being too long to be stored in the standard 512-bit public key.

Knowing this the elliptic curve must be defined over a finite field, it must be restricted to only having integer points and a limit on how large the coordinates of a point can get. To do this the equation:

y² = x³ + ax + b

must be changed to:

y² mod p = (x³ + ax + b) mod p

where p is some prime number to ensure that addition and multiplication operations can always be undone. In secp256k1 p is the largest prime that is smaller than 2^256.

**Bibliography:**

[1]<https://en.wikipedia.org/wiki/Elliptic-curve_cryptography> [Accessed on 15/02/2019]

[2] <https://hackernoon.com/what-is-the-math-behind-elliptic-curve-cryptography-f61b25253da3> [Accessed on 15/02/2019]

[3]<http://mathworld.wolfram.com/SingularPoint.html> [Accessed on 15/02/2019]

[4]<https://bitcointalk.org/index.php?topic=2699.msg37328#msg37328> [Accessed on 16/02/2019]