ELLIPTIC CURVE CRYPTOGRAPHY

Elliptical curve cryptography (ECC) is a public key encryption technique based on elliptic curve theory that can be used to create faster, smaller and more efficient cryptographic keys.

ECC is an alternative to the Rivest-Shamir-Adleman (RSA) cryptographic algorithm and is most often used for digital signatures in cryptocurrencies, such as Bitcoin and Ethereum, as well as one-way encryption of emails, data and software.

An elliptic curve is not an ellipse, or oval shape, but it is represented as a looping line intersecting two axes, which are lines on a graph used to indicate the position of a point. The curve is completely symmetric, or mirrored, along the x-axis of the graph.

Public key cryptography systems, like ECC, use a mathematical process to merge two distinct keys and then use the output to encrypt and decrypt data. One is a public key that is known to anyone, and the other is a private key that is only known by the sender and receiver of the data.

ECC generates keys through the properties of an elliptic curve equation instead of the traditional method of generation as the product of large prime numbers. From a cryptographic perspective, the points along the graph can be formulated using the following equation:

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

ECC is like most other public key encryption methods, such as the RSA algorithm and Diffie-Hellman. Each of these cryptography mechanisms uses the concept of a one-way, or trapdoor, function. This means that a mathematical equation with a public and private key can be used to easily get from point A to point B. But, without knowing the private key and depending on the key size used, getting from B to A is difficult, if not impossible, to achieve.

ECC is based on the properties of a set of values for which operations can be performed on any two members of the group to produce a third member, which is derived from points where the line intersects the axes as shown with the green line and three blue dots in the below diagram labeled A, B and C. Multiplying a point on the curve by a number produces another point on the curve (C). Taking point C and bringing it to the mirrored point on the opposite side of the x-axis produces point D. From here, a line is drawn back to our original point A, creating an intersection at point E. This process can be completed *n* number of times within a defined max value. The *n* is the private key value, which indicates how many times the equation should be run, ending on the final value that is used to encrypt and decrypt data. The maximum defined value of the equation relates to the key size used.

The **private keys** in the ECC are integers (in the range of the curve's field size, typically **256-bit** integers). Example of 256-bit ECC private key (hex encoded, 32 bytes, 64 hex digits) is: `0x51897b64e85c3f714bba707e867914295a1377a7463a9dae8ea6a8b914246319`.

The **key generation** in the ECC cryptography is as simple as securely generating a **random integer** in certain range, so it is extremely fast. Any number within the range is valid ECC private key.

The **public keys** in the ECC are **EC points** - pairs of integer coordinates {***x***, ***y***}, laying on the curve. Due to their special properties, **EC points** can be **compressed** to just one coordinate + 1 bit (odd or even). Thus the **compressed public key**, corresponding to a 256-bit ECC private key, is a **257-bit** integer. Example of ECC public key (corresponding to the above private key, encoded in the Ethereum format, as hex with prefix `02` or `03`) is: `0x02f54ba86dc1ccb5bed0224d23f01ed87e4a443c47fc690d7797a13d41d2340e1a`. In this format the public key actually takes 33 bytes (66 hex digits), which can be optimized to exactly 257 bits.

Elliptic Curves

In mathematics **elliptic curves** are plane algebraic curves, consisting of all points {***x***, ***y***}, described by the equation:

$$A x^3 + B x^2 y + C x y^2 + D y^3 + E x^2 + F x y + G y^2 + H x + I y + J = 0,$$

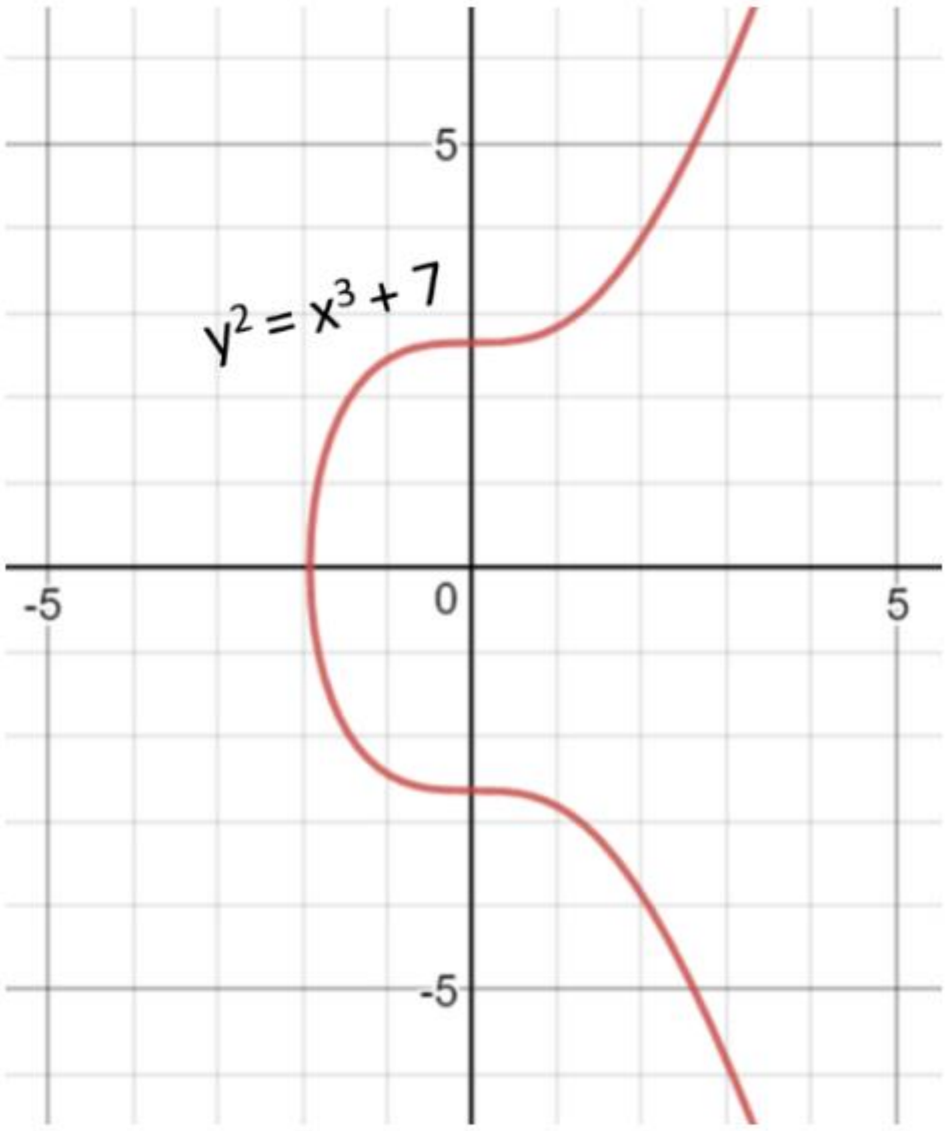
Cryptography uses **elliptic curves** in a simplified form (Weierstras form), which is defined as:

- y2 = x3 + _**a**_x + ***b***

For example, the NIST curve `secp256k1` (used in Bitcoin) is based on an elliptic curve in the form:

- y2 = x3 + ***7*** (the above elliptic curve equation, where ***a*** = **0** and ***b*** = **7**)

This is a visualization of the above elliptic curve:



Elliptic Curves over Finite Fields

The **elliptic curve cryptography (ECC)** uses **elliptic curves over the finite field 𝔽p** (where ***p*** is prime and ***p*** > 3) or **𝔽2*m*** (where the fields size ***p*** = 2_**m**_). This means that the field is a **square matrix** of size ***p*** x ***p*** and the points on the curve are limited to **integer coordinates** within the field only. All algebraic operations within the field (like point addition and multiplication) result in another point within the field. The elliptic curve equation over the finite field **𝔽p** takes the following modular form:

- y2 ≡ x3 + _**a**_x + ***b*** (mod ***p***)

Respectively, the "Bitcoin curve" `secp256k1` takes the form:

- y2 ≡ x3 + ***7*** (mod ***p***)

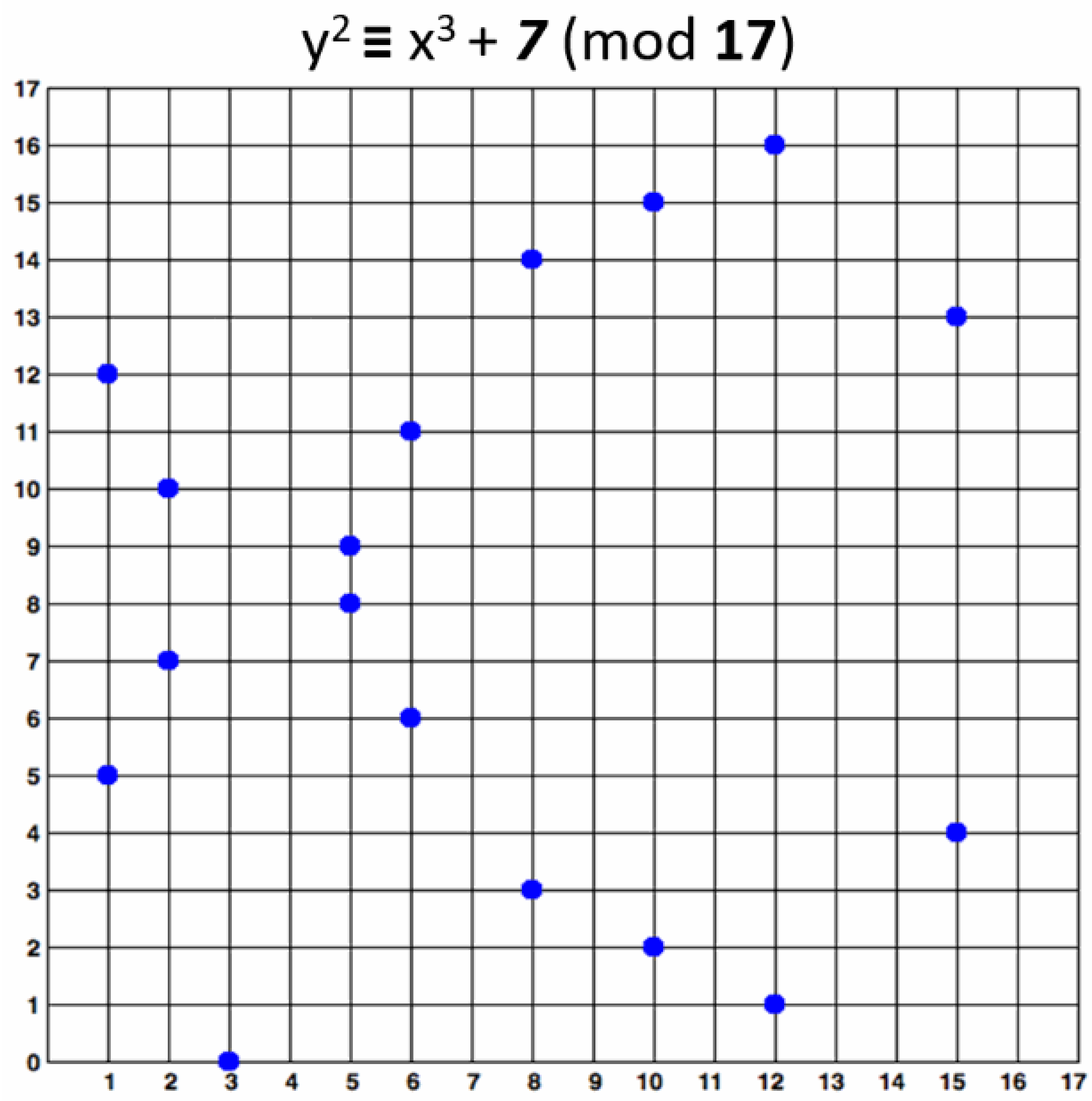
Unlike **RSA**, which uses for its key space the **integers** in the range [0...***p***-1] (the field ℤp), the **ECC** uses the **points** {*x*, *y*} within the Galois field **𝔽p** (where ***x*** and ***y*** are integers in the range [0...***p***-1]). An **elliptic curve over the finite field 𝔽p** consists of:

- a set of integer coordinates {***x***, ***y***}, such that **0** ≤ ***x***, ***y*** < ***p***
- staying on the elliptic curve: _**y**_2 ≡ x3 + _**a**_x + ***b*** (mod **p**)

Example of elliptic curve over the finite field **𝔽17**:

- y2 ≡ x3 + **7** (mod **17**)

This elliptic curve over **𝔽17** looks like this:


$y^2 \equiv x^3 + 7 \pmod{17}$