Elliptic curve arithmetic

Hendrik

April 27, 2020

1 Arithmetic over the real numbers

Consider the elliptic curve

$$y^2 = x^3 + 7 \tag{1}$$

over the real numbers, and denote a point g by its projective coordinates $(x_g, y_g, 1) \sim (\alpha x, \alpha y, \alpha)$. We also write the point reflected over the x-axis as $-g = (x_g, -y_g, 1)$. In this projective space, the line of infinities is given by $(x, y, 0)$ from which we further denote $\infty = (0, 1, 0)$.

Given the set of points that are on the elliptic curve plus ∞, we introduce a group structure as follows. For each g and h we let $g + h = r$ if g, h and $-r$ are colinear. This definition implies that $g + \infty = \infty + g = g$, as well that that $g + (-g) = \infty$. Hence, $e \equiv \infty$ is the unit element whereas $-g$ is the inverse of g. Lastly, the sum $g + g$ is taken to be the inverse of the point that is colinear with the tangent to the curve at g. It is easily verified that this group operation is commutative, since the colinearity is independent of the order of it's generating points. The operation is also associative, although this is not a trivial observation.

For any positive integer n we can then define multiplication on the elliptic curve by $n \cdot g = g + \cdots g$ where summation was repeated $n - 1$ times.

Note that the geometric definition given above ensures that the group is well-defined in the sense that it is closed under addition. In preperation of extending this definition to elliptic curves over closed fields, we first derive the algebraic group operations for elliptic curves over the real numbers.

Addition of inverse elements are equal to e by definition. We then consider two points $g(x_g, y_g)$ and $h(x_h, y_h)$ on the elliptic curve (1), which are not eachother's inverses. These two points define a straight line

$$y = m(x - x_g) + y_g \tag{2}$$

where $m = (y_g - y_h)/(x_g - x_h)$ if $g \neq h$, and $m = 3x_g^2/2y_g$ otherwise. The sum of g and h is then the point $-r$ where r is the intersection of (1) and (2). Solving for x_r is done by consecutively factoring out $(x - x_g)$ and $(x - x_h)$. Not suprisingly, the solutions for different and coincident points is the same if

expressed in terms of the gradient m, namely, $x_r = m^2 - x_g - x_h$. We thus find that

$$\begin{aligned} g(x_g, y_g) + h(x_h, y_h) &= (g+h)(m^2 - x_g - x_h, -m(m^2 - 2x_g - x_h) - y_g) \\ &= (g+h)(m^2 - x_g - x_h, -m(m^2 - x_g - 2x_h) - y_h). \end{aligned} \tag{3}$$

These equations make the commutativity of the operation manifest.

2 Arithmetic over a finite field

2.1 Group structure

In this section we consider the elliptic curve (1) over the finite field $\mathbb{F}_p$ where p is a prime number. The field $\mathbb{F}_p$ is the set of integers modulo p, and since p is taken prime, this field forms a group with respect to addition and multiplication. We thus consider all points $g(x_g, y_g) \in \mathbb{F}_p^2$ that satisfy

$$y^2 \equiv x^3 + 7 \mod p. \tag{4}$$

Equivalently to what we did in the previous section, we introduce a group structure over the points on (4) by adding a unit element e and defining $g + e = e + g = g$, $g + (-g) = e$ and

$$\begin{aligned} g + h = (&m^2 - x_g - x_h \mod p, \\ &- m(m^2 - 2x_g - x_h) - y_g \mod p) \end{aligned} \tag{5}$$

where $m = (y_g - y_h)(x_g - x_h)^{-1} \mod p$ if $g \neq h$, and $m = 3x_g^2(2y_g)^{-1} \mod p$ otherwise.

The order of the group is the number of elements in the group, which can be calculated using Schoof's algorithm in polynomial (in the logarithm of) time. It is worth noting that not every value y_g^2 in (4) will have a square root in $\mathbb{F}_p$, whereas there are two in case it exists.

2.2 Scalar multiplication

Similarly to what we did in the context of real numbers, scalar multiplication can be defined as $n \cdot g = g + \cdots g$. This computation can be sped up significantly by considering the binary represention $n = \sum_k b_k 2^k$ such that

$$n \cdot g = \sum_k b_k (2^k \cdot g). \tag{6}$$

Since each term in this sum requires a doubling of the point obtained in the previous term, we have at most two times $\log_2 n$ additions to compute, which is a huge gain compared to the n additions in the brute-force calculation.

Let us then consider the elements generated by g, i.e., the set $\hat{g} = \{i \cdot g\}_{i=0\ldots n-1}$, where n is the minimum number for which $n \cdot g = e$, indeed implying

that n is the number of elements in $\hat{g}$. Note that $\hat{g}$ is a subgroup on the elliptic curve, since $i \cdot g + j \cdot g = (i+j) \cdot g$ is also an element of $\hat{g}$, and we also have that $i \cdot g + (n-i) \cdot g = e$. It can be shown that for any commutative multiplicative group, the order of $\hat{g}$ divides the order of the supergroup.[1] Hence, if the elliptic curve has N points, every point generates a subgroup of order n where n divides N. This also means that all generated subgroups are finite and cyclic. In particular, if the order of the elliptic curve is prime, the two subgroups are the trivial one (generated by the unit element) and the whole group itself.

In cryptography we will look for subgroups with prime order n, in which case we are sure that the corresponding subgroup does not have proper subgroups itself. Given such a large prime number all points $n' \cdot g$, where $n' = N/n$ and for any g on the elliptic curve, are generators of the considered subgroup. This is so because $N \cdot g = 0$ as any subgroup order divides N.

2.3 Discrete logarithm

Given that we have a point $n \cdot g$ and the generating point g, the discrete logarithm of $n \cdot g$ is the number n. Currently, the fastest way to compute this is brute-force, i.e., to repeatedly add the inverse of g until you get g, which is $O(n)$. Therefore, the time needed to invert scalar multiplication is exponentially longer than the multiplication itself.

In a cryptographic setting, we are considering a finite field for a very large prime number. For example, in `secp256k1` the elliptic curve (4) is considered over $\mathbb{F}_p$ where[2]

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 2^0.$$

Given a private key n of the order of 2^{256} and a generating point g, a public key is quickly calculated from $n \cdot g$. Reverting this would take a huge amount of time, namely, of the order of 2^{248} more slowly than the key generation itself. It is this discrepancy in calculation time which is the backbone of the security of private-public key pairings.

3 Digital signatures

In this section we will discuss the Elliptic Curve Digital Signature Algorithm (ECDSA), which is a digital signature protocol used for signing transactions in various cryptocurrencies. First let us observe that a digital signature is a mathematical scheme that ensures the authenticity of digital messages.[3] The

[1] This follows from Lagrange's theorem. Indeed, consider all right coset spaces $\hat{g} + h$ on the elliptic curve. Each coset has the same order as $\hat{g}$, because they are all isomorphic to it. Furthermore, the cosets are disjoint. This is so because if a point were in two cosets, all points would be related by the left action of $\hat{g}$, so that the two cosets must be the same. Therefore, the right cosets form a partition and have the same order, from which it follows that the order of $\hat{g}$ divides the order of the group on the elliptic curve. The division thus equals the number of right cosets of $\hat{g}$.

[2] `http://www.secg.org/sec2-v2.pdf`

[3] `https://en.wikipedia.org/wiki/Digital_signature`

authenticity is derived from the following three crucial properties of a digital signature.

Authentication Any recepient of the messages that knows the private key of the sender, can verify the signature of the message.

Non-repudiation Any agent that sees a message can verify the signature of the message. Note that this property implies authentication.

Integrity A message that is signed cannot be altered without rendering the signature invalid.

Non-repudiation thus means that the signature can be verified without having access to the private key. As anyone can verify the signature, the latter cannot be denied by the authorizer.

Let m be the transaction or message and (k, K) a private-public key pair, i.e., $K = k \cdot G$ for a generator G. To sign the message, an ephemeral private key t is considered. According to ECDSA, the signature is then given by

$$s(m, k, r) \equiv t^{-1}\text{SHA}(m) + rk \mod p, \tag{7}$$

where $(r, \_) = t \cdot G$ and SHA is a secure hash algorithm, e.g., keccak256.

Given the digital signature s, the private key r and the public key K of the authorizer, anyone can check the signature by verifying whether

$$(r, \_) = u_1 \cdot G + u_2 \cdot K, \tag{8}$$

where

$$u_1 \equiv \text{SHA}(m)s^{-1} \mod p \quad \text{and} \quad u_2 \equiv rs^{-1} \mod p. \tag{9}$$

Indeed, the verification will only work if K is the public key corresponding to k, which can be easily checked by substituting for the relevant equations.