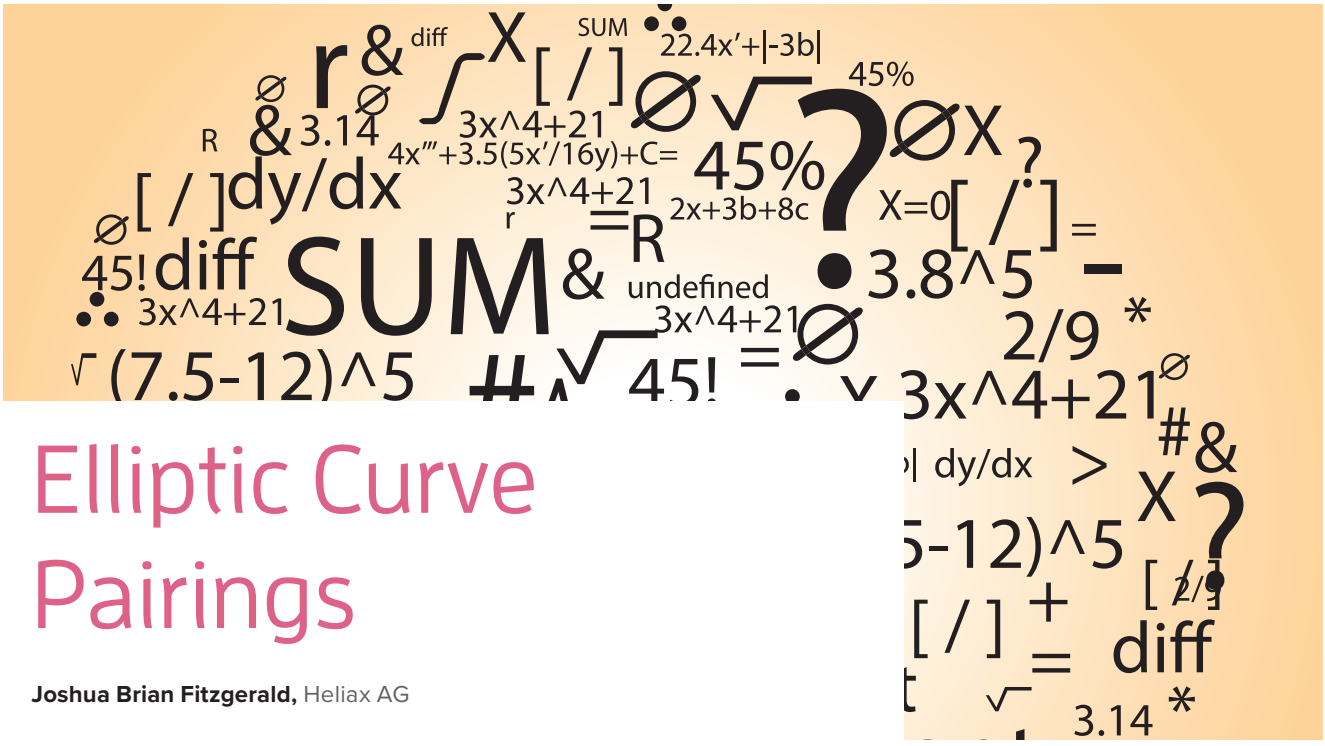

Elliptic Curve Pairings

Joshua Brian Fitzgerald, Heliax AG

Elliptic curve pairings are a powerful tool and a popular way to construct zero-knowledge proofs, which are beginning to be used in blockchains as a way to provide privacy in the transaction ledger.

A recent trend in cryptography is to leverage zero-knowledge proof systems that allow one party to convince another that they have information that satisfies certain conditions without revealing the information itself. This technology is particularly useful when multiple parties—who may not trust each other—nevertheless need to coordinate to produce some useful information. For example, the Zcash cryptocurrency uses zero-knowledge cryptography to hide the amounts and recipients in Zcash transactions while allowing an observer to verify the resulting balances have been adjusted correctly.

The heart of many of these zero-knowledge proof systems is the elliptic curve pairing, a useful function that can test if two points on an elliptic curve are related in the same way as another pair of points.

ELLIPTIC CURVES

An elliptic curve is a polynomial that can be written as $y^2 = x^3 + ax + b$. If the y^2 on the left side were simply y, then this would be a typical cubic polynomial, familiar from a high school algebra class. Squaring the y on the left side turns this cubic into an elliptic curve with a pleasing symmetry across the x-axis.

Amazingly, the points on an elliptic curve can be used to define an algebraic group structure. Draw a straight line diagonally across an elliptic curve and, being cubic, it will intersect the line at up to three points. These intersection points "add up to zero" in the elliptic curve group structure. If $P = (x_1, y_1)$, $Q = (x_2, y_2)$, and $R = (x_3, y_3)$, then having a straight line $y = mx + b$ passing through these three points tells us that $P + Q + R = 0$ in the elliptic curve group. This can be rearranged as $P + Q = -R$, which gives us a formula for the sum of two points on an elliptic curve and a geometric method for calculating it. To add two points: draw a line through them and look for the third intersection point. The *inverse* of this point is the desired sum.

Digital Object Identifier 10.1109/MC.2022.3146745
Date of current version: 8 April 2022

0018-9162/22©2022IEEE

The identity element, or zero, of the elliptic curve group is an abstract point at infinity. Every line can be thought of as passing through this point as the line heads off the page and out to infinity. (The point at infinity can be made explicit using a projective coordinate system where it becomes a regular point with integer coordinates.)

The inverse of an elliptic curve point is its reflection across the x-axis. The line through these two points will be vertical and not intersect the curve proper in any other place. If $P = (x_1, y_1)$, then the vertical line $x = x_1$ passing through P will also intersect the curve at $Q = (x_1, -y_1)$. The equation representing these intersections is $P + Q = 0$, which can be rearranged to show that $Q = -P$. Thus, the additive inverse of a point is its reflection across the x-axis.

Using some calculus, we can find tangent lines on an elliptic curve. A line that is tangent to a point P on the elliptic curve can be thought of as intersecting the curve twice at that point. There will be a third intersection point, Q, showing that $P + P + Q = 0$ or, equivalently, $2P = -Q$.

Therefore, with some algebra and a little calculus, we can add two different points or double any single point on an elliptic curve. This gives us an efficient way to find an nth multiple of any point on the curve using a double-and-add style algorithm requiring $O(\log n)$ steps.

ELLIPTIC CURVES IN CRYPTOGRAPHY

An elliptic curve is often introduced using the real numbers R as the base field for which x and y are members. But an elliptic curve can be constructed over any field, including finite fields, such as the integers modulo a prime number. When constructed over a finite field, an elliptic curve group will have a finite number of elements, and its group structure will be either cyclic or a product of two cyclic groups.[2]

Over a finite field, each point on an elliptic curve has finite order, so for any point P there is some whole number r such that $rP = 0$. All the points P such that $rP = 0$ form a subgroup of the elliptic curve group called the *r-torsion subgroup*.

Now we can see how elliptic curve groups can be used in cryptography. Suppose you have a secret number n. Take some publicly known point P and compute nP using the efficient algorithm mentioned earlier. The point nP is just a coordinate pair (x, y) and can be published freely. Someone else who knows both P and nP can discover n by naively multiplying P by itself repeatedly until reaching nP, but this will take ages if n is large. Faster algorithms for discovering n exist, but all are exponential in the number of digits of n. This is the elliptic curve discrete logarithm problem (ECDLP) that is used as the basis for elliptic curve cryptography.

Elliptic curve cryptography is fast and uses smaller keys than other cryptosystems, such as Rivest–Shamir–Adleman. For these reasons, it is the primary public-key cryptography used to secure Internet communication today. It is the reason I can send my credit card number or email password over the public Wi-Fi at a coffee shop, and no one else in the vicinity, including the coffee shop owner or their Internet service provider, can snoop on it and see my information.

PAIRINGS ON ELLIPTIC CURVES

The elliptic curve group used as the basis for cryptography is not a general algebraic group. Elliptic-curve-based groups have additional structure and particularities that are specific to elliptic curves. This means that elliptic curves could have specialized attacks that exploit their extra structure to make the ECDLP easier to break.

One such structure is the elliptic curve pairing. An elliptic curve pairing is a nondegenerate bilinear map e: $E[r] \times E[r] \rightarrow \mu_r$, where $E[r]$ is the r-torsion subgroup of an elliptic curve group, and μ_r is the multiplicative group of rth roots of unity in the field F_{q^k} (q is the characteristic of the field over which the elliptic curve is constructed). The exponent k is called the *embedding degree* and is an important security parameter for pairings.

Bilinearity means that $e(nP, mQ)_r = e(P, mQ)_r^n = e(nP, Q)_r^m = e(P, Q)_r^{nm}$, where the r subscript means "in μ_r." Nondegeneracy means that $e(P, Q)_r \neq 1$ as long as neither P nor Q is zero (the point at infinity).

The existence of a pairing on an elliptic curve means that one can check to see if two pairs of points are related in the same way. Suppose we have a secret number n, and we also have four elliptic curve points: P, nP, Q, and nQ. Using a pairing, we can test to see if nP is the same multiple of P as nQ is of Q by checking the following pairing equation:

$$e(P, nQ)_r = e(nP, Q)_r.$$

Checking this equation involves computing the resulting root of unity on each side and comparing the results. At no point do we discover n, but we can tell that both nP and nQ are nth multiples of P and Q. This leaks a little bit of information about these points that somewhat weakens security.

This difference in security can be formalized using computational hardness assumptions. Some relevant hardness assumptions for elliptic curve pairings are the computational Diffie–Helman (CDH) problem and the decisional Diffie–Hellman (DDH) problem as well as the discrete logarithm problem discussed earlier.[3] The CDH problem asks whether it is feasible to compute nmP only given P, nP, and mP. The CDH problem is infeasible for carefully chosen elliptic curves. (Notice that the CDH problem would be feasible if the discrete logarithm were also feasible to compute.)

The DDH problem doesn't ask that nmP be computed, only that it be

recognizable. That is, given only P, nP, mP, and Q, decide whether Q is actually equal to nmP. This is easy to do with a pairing by checking this pairing equation:

$$e(nP, mP)_r = e(P, Q)_r.$$

COMPUTING A PAIRING

A few different definitions of pairings exist, but the most commonly used pairings are based on Tate's definition, which allows several optimizations. Tate's pairing is defined as follows[1]:

$$e(P,Q)_r = f_{r,p}(Q)^{\frac{q^k-1}{r}}.$$

In this formula, r (on the right-hand side) is the order of an r-torsion group to which P must belong, q is the characteristic of the field over which the elliptic curve is constructed, and k is the embedding degree discussed in the section "Pairings on Elliptic Curves."

Computing the pairing has two major phases: the Miller loop, where function $f_{r,P}(Q)$ is computed using a double-and-add style algorithm, and then the final exponentiation, in which $f_{r,P}(Q)$ is taken to the power of $(q^k-1)/r$.

MILLER'S ALGORITHM

Function $f_{r,P}$ is a rational function in x and y with a zero of multiplicity r at P (meaning $rP = 0$) and a pole of multiplicity r at the point at infinity. This function can be constructed iteratively out of linear factors. At each point in this process, we need to know how many zeros and poles our function has and at which places. A *divisor* is the mathematical construction that takes care of this bookkeeping for us. Divisors are out of the scope of this article but are implicitly used in this section for counting multiplicities of zeros and poles.

Suppose, for some n, the line $ax + by + c = 0$ passes through points P and nP. Because of the group law on elliptic curves, this same line will also pass through $-(n + 1)P$. This line will have a zero of order 1 at each of P, nP, and $-(n + 1)P$ as well as a pole of order 3 at the point at infinity. We will denote this line through P and nP as $l_{P,nP}$.

Suppose we also find the vertical line $x + d = 0$ passing through nP and $-nP$. This line has zeros of order 1 at both nP and $-nP$ and a pole of order 2 at the point at infinity. We will denote the vertical line through nP as v_{nP}.

Dividing these gives us a rational function

$$g_{n,P}(x,y) = \frac{l_{P,nP}}{v_{nP}} = \frac{ax+by+c}{x+d}$$

which will have zeros of order 1 at P and nP but poles of order 1 at $(n + 1)P$ and the point at infinity. We can compute these functions for each n from 1 to r. When multiplied together, these functions have a telescoping-like behavior, where the linear factor in the numerator of $g_{n,P}$ creating a zero at nP cancels out the linear factor in the denominator of the previous function $g_{n-1,P}$ that had created a pole at nP.

The target function can be found by taking the product of these rational functions like so:

$$f_{r,P}(x,y) = \prod_{i=1}^{r} g_{i,P}(x,y),$$

where $g_{i,P}(x, y)$ is the rational function whose numerator is a line passing through P and iP, and whose denominator is a vertical line through iP, as described previously.

This suggests an algorithm for computing $f_{r,P}(Q)$ by successively evaluating $g_{i,P}$ at Q and accumulating the results by multiplying the current state by $g_{i,P}(Q)$ for each i up to r. This algorithm's runtime complexity is $O(r)$, and so it is not suitable for our purposes where large r will be used.

However, we can make this into an $O(\log r)$ algorithm by using a double-and-add style approach. The preceding product formula suggests this iterative formula for computing $f_{m+1,P}$ from $f_{m,P}$:

$$f_{m+1,P} = f_{m,P} \cdot \frac{l_{P,nP}}{v_{nP}}.$$

Similarly, we can also get a crucial formula for $f_{2m,P}$ from $f_{m,P}$:

$$f_{2m,P} = (f_{m,P})^2 \cdot \frac{l_{nP,nP}}{v_{2mP}}.$$

Combining these gives us a double-and-add method for computing $f_{r,P}$ that runs in $O(\log r)$ time.

THE FINAL EXPONENTIATION

The only remaining piece of the pairing formula left to compute is the final exponentiation, in which the result of $f_{r,P}(Q)$ is taken to the power $(q^k-1)/r$. In practice, the characteristic of the field q is very large, say 256 bits, and the embedding degree k is greater than 1. (Two commonly used elliptic curve families, Barreto–Naehrig (BN) and Barreto–Lynn–Scott, use $k = 12$ and $k = 24$, respectively.[1]) A greater k increases security but makes the final exponent much greater and harder to compute. To make things even worse, a large k also means the field elements we are working with are members of a kth-degree extension field, where multiplication is more difficult than in the base field. "Pairing-friendly" curves have an embedding degree that is large enough to provide security but not too large that efficiency is compromised.

Thankfully, a few optimizations can help. For BN elliptic curves, the embedding degree will always be 12. Then, $(q^{12} - 1)/r$ can be factored as $(q^6 - 1)(q^2 + 1)((q^4 - q^2 + 1)/r)$. The two binomial factors can be computed easily with the Frobenius operation, leaving the remaining $(q^4 - q^2 + 1)/r$ part with a much lower exponent.

Since the exponent $(q^k-1)/r$ is based on the parameters of an elliptic curve and known well in advance, precomputation can speed up the final exponentiation process.[4]

Furthermore, occasionally multiple pairing values are computed and multiplied together according to some formula. The Miller loop portion of each pairing can be computed first, these results multiplied together using the formula, and then the final exponentiation can be applied once at the very end of the process.

APPLYING PAIRINGS TO A ZERO-KNOWLEDGE PROOF IN A CRYPTOCURRENCY BLOCKCHAIN

Suppose there is a public ledger that records every transaction of a new cryptocurrency. A transaction would include a sender address, recipient address, and the amount being transferred:

$$T = (S, R, a).$$

Miners or validators of this cryptocurrency can compute the balance b in any account by adding up all of the amounts sent to that account number and subtracting the amounts sent from that account number. To check that the sender has enough funds to make the transaction, the validators check the following inequality:

$$b - a \geq 0.$$

Each transaction in this new cryptocurrency is public, so the amounts being transferred are viewable, and all account balances can be easily computed from the ledger.

However, using an elliptic curve pairing, we can transform this simple public ledger into a private one, where all amounts are encrypted, and balances cannot be computed from the information in the ledger. (The simple protocol that follows was chosen to illustrate the usefulness of an elliptic curve pairing in a blockchain environment, not for its security. This protocol suffers from a few security flaws that would need to be carefully addressed before implementing it safely.)

Our new private cryptocurrency will use elliptic curve cryptography to hide the amount of each transaction. The designers of this private protocol will choose a pairing-friendly elliptic curve E and a point P on the curve that generates a large prime-order subgroup. Then, when the sender wants to send the amount a in a transaction, it includes aP in the amount field rather than a. Now, the amount of the transaction, a, cannot be computed from the information in the transaction because of the ECDLP. Since elliptic curve arithmetic respects addition, an encrypted version of the balance, bP, can be computed by adding all of the encrypted amounts together just as before.

But how can we check that the sender has enough funds in the account to complete the transaction? Because of the ECDLP, no validator can check directly that the sender's balance exceeds the transaction amount. But if the sender includes some extra information, it ought to be able to convince a validator that the balance is high enough.

The sender chooses some point Q on the curve which it keeps to itself. It then computes $b' = b - a$, the new balance after the transaction is complete, and computes its bit decomposition $(b'_0, b'_1, \ldots, b'_n)$. The sender includes $(b'_0Q, b'_1Q, \ldots, b_nQ')$ in the transaction data.

From this encrypted bit decomposition, a validator can compute $b'Q$ using the equation:

$$b'Q = b'_0Q + 2b'_1Q + 4b'_2Q + \cdots + 2^n b'_nQ.$$

By providing a bit decomposition of b', the sender can show that b' is positive by the simple fact that there aren't enough bits to overflow the order of the elliptic curve subgroup we are working in.

Next, the validator chooses a random number k and sends it to the sender. The sender computes kQ and sends this back to the validator. The validator uses this elliptic curve pairing equation to check that the two versions of the balance are consistent with each other:

$$e(kP, b'Q) = e(bP, kQ).$$

Because of the bilinearity of the pairing, this equation shows that $b'k = bk$ in the order-r target group for the pairing. With no foreknowledge of k, it is extremely unlikely that the sender can choose a bit decomposition that satisfies the pairing equation that is not the true decomposition.

Now that we've verified that the sender has enough to complete the transaction, the transaction data $T = (S, R, aP, (b'_0Q, b'_1Q, \ldots, b'_nQ), k, kQ)$ are appended to the ledger, and the transaction is complete.

This protocol is easily checkable with pairing equations because of the highly arithmetical nature of the statement we are checking. Elliptic curve pairings can be used to check more complicated statements than this, however. Using a much more complex construction called a *zk-SNARK*, a few pairing equations can check any statement in NP.

One major flaw in the protocol described previously is that the sender's and recipient's addresses are public, and anyone reading the ledger can see that the two parties are transacting, even if they can't tell how much is being transferred. In Zcash, the sender and recipient addresses are also hidden, and a zk-SNARK is used to verify the addresses and link the amounts to the correct accounts without revealing the addresses.

REFERENCES

1. C. Costello, *Pairings for Beginners*. 2012. [Online]. Available: https://www.craigcostello.com.au
2. J. H. Silverman, *The Arithmetic of Elliptic Curves*, 2nd ed. New York, NY, USA: Springer-Verlag, 2009.
3. M. Bellare and P. Rogaway, *Introduction to Modern Cryptography*. Boca Raton, FL, USA: CRC Press, 2020.
4. T. Kim, S. Kim, and J. H. Cheon, "Accelerating the final exponentiation in the computation of the Tate pairings," Cryptology ePrint Archive, Rep. 2012/119, Mar. 2012. [Online]. Available: https://eprint.iacr.org/2012/119

JOSHUA BRIAN FITZGERALD
is a cryptographer and protocol developer for Heliax AG, Zug 6300, Switzerland. Contact him at joshuabfitzgerald@gmail.com.