Lecture Notes, Week 6

Math 480A2: Mathematics of Blockchain Protocols, Fall 2022

Lecturer: Bryan Gillespie

Elliptic Curves Over Finite Fields

In the following, we will write F_q for the finite field of order $q = p^r$. Elliptic curves taken over finite fields are a bountiful source of finite abelian groups which are used in modern cryptography.

A first obvious observation about elliptic curves taken over F_q is that their solution sets consist of pairs of field elements, and thus are finite. A natural question then is: how many elements can an elliptic curve group over F_q have. An answer to this is given by the famous Hasse's theorem.

Theorem 1 (Hasse's Theorem)**.** *Let E be an elliptic curve over F_q, and let $N(E)$ be the number of elements in the elliptic curve group associated with E. Then*

$$|N(E) - (q + 1)| \leq 2\sqrt{q}$$

For a prime number p, the value of $N(E)$ for an elliptic curve E over F_p can be any number in the range given by Hasse's theorem. For prime powers of higher degree, almost every group size in the Hasse range can be achieved except for a small number of possibilities which can be explicitly described.

When q is not a power of 2, we can give a nice heuristic reason for why Hasse's theorem holds. In this case, E can be expressed in a Weierstrass form that is given by $y^2 = f(x)$ for some cubic polynomial f. If we didn't know anything about the values of $f(x)$ for different values of x in F_q, we might guess that $f(x)$ is a "random" element of F_q. If we make this assumption, then we can identify three possible outcomes:

- If $f(x) \neq 0$ is a square in F_q, then letting y be a square root of this value, we get two points on E, (x, y) and $(x, -y)$.
- If $f(x) \neq 0$ is a non-square in F_q, then no value of y gives a solution of $y^2 = f(x)$, so no point on E has x as its x-coordinate.
- If $f(x) = 0$, then $(x, 0)$ is a single point on E.

For odd q, it can be shown that exactly half of the nonzero elements of F_q are squares. If we assume that $f(x)$ is "random" and distributed uniformly across F_q, then this means that the expected number of points on E for each point $x \in F_q$ is 1, giving on average q points of E plus 1 for the point at infinity.

Stretching this heuristic further, the bound of $2\sqrt{q}$ in Hasse's theorem can be interpreted probabilistically as an instance of the well-known *central limit theorem*. In particular, the scale of the normal distribution associated with the central limit theorem is $\sqrt{n}$ where n is the number of random variables in a sum. Thus $\sqrt{q}$ is the "right size" to expect if the sampling of squares and non-squares for $f(x)$ was actually random and independent.

Once we have the size of a finite abelian group, another natural question to ask is: what is its structure? This has a nice answer for elliptic curve groups over a finite field.

Proposition 2. *Let E be an elliptic curve over F_q. Then E is either a cyclic group or a sum of two cyclic groups. In the latter case, there are integers m, n with $m \mid n$ such that $E \cong (\mathbb{Z}/m\mathbb{Z}) \oplus (\mathbb{Z}/n\mathbb{Z})$.*

The divisibility part of the above result implies that the factorization of the order $N(E)$ of an elliptic curve group over a finite field is strongly related to its group structure. For instance, if the group order has no repeated prime factors, then this implies that the group itself is cyclic! In general, at most half of the copies of a prime number in the factorization of the elliptic curve group order can be used as part of the size m of the smaller cyclic part in the above representation.

A computational operation which is frequently useful when working with elliptic curve groups is that of taking a repeated sum of a point with itself many times. This repeated sum can be computed efficiently as follows.

Let K be a field and let E be an elliptic curve defined over K. For positive $m \in \mathbb{N}$, write m in its binary representation via

$$m = \sum_{i=0}^{d-1} a_i 2^i$$

where $a_i \in \{0, 1\}$ is the coefficient of 2^i in this representation. We will assume that $m > 0$ and the largest coefficient a_{d-1} is 1. Then the repeated sum mP can be computed by repeated doubling and addition d times, as follows.

Algorithm 1 Fast computation of mP in an elliptic curve group by a positive integer m

```
function MULT($m = 2^{d-1} + 2^{d-2}a_{d-2} + \cdots + 2a_1 + a_0 : \mathbb{N}_+,\ P : E(K)) \to E(K)$
    $Q : E(K) \leftarrow P$
    for $i \in 1 .. (d-1)$ do
        Q $\leftarrow Q + Q$
        if $a_{d-1-i} = 1$ then
            Q $\leftarrow Q + P$
    return Q
```

Lemma 3. *Algorithm 1 computes mP for $m \in \mathbb{N}$ and $P \in E(K)$ using at most $2\log_2(m) - 2$ group operations in $E(K)$.*

Proof. The binary representation of an integer m uses at most $\log_2(m)$ digits, so this gives an upper bound on d. Each iteration of the for loop includes the computation of $Q+Q$, and possibly the computation of $Q + P$ if the corresponding digit a_i is 1, so we use at most two group operations per repetition of the for loop, resulting in at most $2(d-1) < 2\log_2(m) - 2$ group operations in total.

To see that the algorithm computes the right sum, we argue that after the j-th repetition of the for loop, the value of the variable Q is $(2^j + \sum_{i=1}^{j} 2^{j-i} a_{d-1-i})P$. We can interpret the base case $j = 0$ as the state before the algorithm has executed the for loop at all, and where the summation beside P in our induction formula is just 1, giving $1P = P$. This is the value that Q is assigned before executing the for loop, so all is well.

Now suppose that the algorithm has executed the for loop $j > 0$ times, and that the inductive hypothesis holds for $j - 1$ repetitions of the for loop. Then at the end of the previous repetition, Q has value $(2^{j-1} + \sum_{i=1}^{j-1} 2^{j-1-i} a_{d-1-i})P$. Adding this value of Q to itself has the effect of multiplying the coefficient by 2, giving

$$Q + Q = \big(2^j + \sum_{i=1}^{j-1} 2^{j-i} a_{d-1-i}\big)P$$

The subsequent if statement has the effect of adding $a_{d-1-j}P$ to Q, giving

$$Q + a_{d-1-j}P = \big(2^j + \sum_{i=1}^{j-1} 2^{j-i} a_{d-1-i} + a_{d-1-j}\big)P = \big(2^j + \sum_{i=1}^{j} 2^{j-i} a_{d-1-i}\big)P$$

Thus Q again has the desired value after the j-th repetition of the for loop. The resulting value of Q returned after $d - 1$ repetitions can be seen to be mP as desired. □

The takeaway from this algorithm is that the operation of "multiplying P by m" can be accomplished in time logarithmic in the number m being multiplied — fast! We will see next that the inverse operation is (probably) computationally much more difficult, and that this (probable) fact is the foundation of a large body of cryptographic practice.

The Discrete Logarithm Problem

The fundamental concept which makes elliptic curve groups useful in modern cryptography is the so-called *discrete logarithm*.

Definition 4. Let G be an abelian group, and let $g, h \in G$. The **discrete logarithm problem** for G is the problem of finding an exponent m such that $g^m = h$. Such an exponent is sometimes denoted $\log_g(h)$. If $G = E(F_q)$ is the elliptic curve group for some elliptic curve over a finite field F_q, then finding an m, given $P, Q \in E(F_q)$, such that $Q = mP$, is called the **elliptic curve discrete logarithm problem**, or **ECDLP**.

In a variety of abelian groups, the discrete logarithm problem is considered to be *hard* to compute. In a group where this is the case, exponentiating a generator g to a power a representing a piece of data can be thought of as a sort of "hiding" operation on this data; it faithfully represents the value of a inside the group, but the actual value of a can't be retrieved from this representation.

The meaning of the word "hard" often depends on the application, and typically is related to the computational capabilities of potential adversaries of a cryptographic system. It will be helpful to recall a few definitions describing the asymptotic growth of functions in order to give a reasonable definition.

Definition 5. Let $f : \mathbb{R}_+ \to \mathbb{R}$. Then f is said to have:

- **At most polynomial growth**, or sometimes just **polynomial growth**, if there exists $k \in \mathbb{N}$ such that for all sufficiently large x, $|f(x)| \leq x^k$
- **Superpolynomial growth** if for every $k \in \mathbb{N}$, for all sufficiently large x, $|f(x)| \geq x^k$
- **Exponential growth** if there exist real numbers $r, s > 1$ such that for all sufficiently large x, $r^x \leq |f(x)| \leq s^x$
- **Subexponential growth** if for every real number $r > 1$, for all sufficiently large x, $|f(x)| \leq r^x$

If $g : \mathbb{R}_+ \to \mathbb{R}$, then f is said to be:

- **Big-O of** g, written $f = O(g)$, if there exists a positive constant $C \in \mathbb{R}_+$ such that for all sufficiently large x, $|f(x)| \leq C\,|g(x)|$
- **Little-o of** g, written $f = o(g)$, if for every positive constant $c \in \mathbb{R}_+$, for all sufficiently large x, $|f(x)| \leq c\,|g(x)|$

A moderately encompassing, if theoretical, definition for difficulty of the discrete logarithm problem is that a family of groups (G_α) has hard discrete logarithm if the complexity of computing the discrete logarithm by any known means is super-polynomial as a function of $\log |G_\alpha|$. Note that in general if a function f describes the time complexity of an algorithm in terms of $|G|$, then the complexity in terms of $\log |G|$ is given by the function $g(n) = f(2^n)$.

We take a moment now to demonstrate that the discrete logarithm problem is not always difficult.

Example 6. (Easy discrete logarithm) Let $G = (\mathbb{Z}/p\mathbb{Z}, +)$ be the additive group of the integers modulo p for a prime number p, and let $a, b \in \mathbb{Z}/n\mathbb{Z}$. If $b = 0$ then we can write $b = 0 \cdot a$, giving a discrete logarithm of 0. If $b \neq 0$ but $a = 0$, then the discrete log of b with respect to a doesn't exist. If $a, b \neq 0$, then we can compute the discrete log of b by computing the multiplicative inverse of a in $\mathbb{Z}/p\mathbb{Z}$, and multiplying b by this element: $(a^{-1}b)a = b$. Computing a multiplicative inverse takes time logarithmic in p using the extended Euclidean algorithm, so this approach is polynomial in $\log p$.

The task of identifying abelian groups which have "hard" discrete logarithm is, in fact, challenging and quite subtle. Note that our definition for difficulty refers to the "fastest algorithm known" for solving a particular instance of the discrete log problem. This hints at a fundamental weakness in our current collective understanding, namely, that there is a lack of theoretical results giving computational lower bounds on computing discrete logs. Thus, the best meter that we have available is our algorithmic state-of-the-art for solving the problem, which is intrinsically a messy ruler to measure groups against. We discuss in operational terms only a sampling of these algorithms.

In the following, consider the DLP for a group G containing n elements. Some approaches for solving the discrete logarithm problem apply for arbitrary abelian groups, without regard to any other features of their structure. For instance, the naive approach to solving the DLP for $g, h \in G$ is to raise g to successive powers until you find an exponent giving h; this will take, on average, $n/2$ group operations to compute.

Several algorithms for solving the discrete logarithm problem for general groups make use of variants of the combinatorial "birthday paradox" in order to simplify the computation. These include:

- The *baby-step giant-step* algorithm solves the DLP using time and space $O(\sqrt{n})$
- The *Pollard rho* and *Pollard kangaroo* algorithms are randomized algorithms which solve the DLP using time $O(\sqrt{n})$, and very low space

Importantly, an algorithm running in time $O(\sqrt{n})$ does not violate "hardness" of the discrete logarithm problem, since $\sqrt{n}$ still grows asymptotically more quickly than $(\log n)^k$ for any fixed k. In fact, $\sqrt{n}$ is exponential as a function of $\log n$:

$$\sqrt{n} = \sqrt{2^{\log n}} = (\sqrt{2})^{\log n}$$

Thus these general algorithms for solving the DLP for any abelian group work in *exponential* time as a function of $\log n$.

Another algorithm for the DLP working for arbitrary abelian groups is the *Pohlig-Hellman* algorithm, which makes use of the Chinese remainder theorem to reduce the complexity of the computation based on the factorization of the group order. Specifically, if $n = p_1^{e_1} \cdots p_k^{e_k}$, then

- The *Pohlig-Hellman* algorithm solves the DLP using time $O(\sum_i e_i(\log n + \sqrt{p_i}))$

Thus, when the group order n has only small prime factors (say no larger than $(\log n)^k$ for some integer k), Pohlig-Hellman is a polynomial-time algorithm to solve the DLP on G, meaning that G does not have hard discrete logarithm problem. However, if G has prime order, or if the order at least has a large prime factor, then this approach is ineffective.

The situation is a bit different if we specialize to the multiplicative subgroup of a finite field. In this case, there are specialized algorithms which run much faster, in *subexponential* time. These include:

- The *index calculus* algorithm solves the DLP for $F_q^\times$ in time

$$\exp\left(\left(\sqrt{2} + o(1)\right)(\log q)^{1/2}(\log\log q)^{1/2}\right)$$

- The *function field sieve* algorithm solves the DLP for $F_q^\times$ in time estimated (but not proven) to be approximately

$$\exp\left(\left(\sqrt[3]{64/9} + o(1)\right)(\log q)^{1/3}(\log\log q)^{2/3}\right)$$

This latter class of algorithms explains in part why elliptic curve groups are useful in modern cryptography as a source of abelian groups with hard discrete logarithms: they lack the structure present in finite field multiplicative groups enabling specialized solutions, so no algorithms are known which solve the ECDLP for arbitrary elliptic curve groups in subexponential time.

Elliptic Curve Pairings

We now discuss in general terms a construction, the *elliptic curve pairing*, which draws an important connection between the ECDLP for certain elliptic curve groups, and the DLP in the multiplicative group of a related finite field. We will see that this produces a valuable tool for cryptographic applications, but also a new attack vector for discrete log hardness in the relevant elliptic curve groups.

Definition 7. Let G and G_t be cyclic groups of equal cardinality n. A map $e : G \times G \to G_t$ is said to be **bilinear** if for all $u, v \in G$ and all $a, b \in \{0, 1, \ldots, n-1\}$,

$$e(u^a, v^b) = e(u, v)^{ab}$$

A bilinear map is called **non-degenerate** if for any generator g of G, we do not have $e(g, g) = 1_{G_t}$. A non-degenerate bilinear map is sometimes called a **pairing**, since it associates an element of G_t to each pair of elements in G.

In some settings, the domain of a pairing may be chosen to be a tuple of elements from two separate groups G_1 and G_2 with the same cardinality as G_t, as this flexibility can allow for more efficiently computable maps. If $G_1 = G_2$ as in the above definition, then the pairing is called *symmetric*, and otherwise it is called *asymmetric*.

On the topic of constructing pairings, note that two cyclic groups of the same order are isomorphic, so theoretically a pairing can be defined on any cyclic group by mapping it to a group with easy discrete logarithm such as $\mathbb{Z}/n\mathbb{Z}$, and implementing an appropriate pairing in this group. However, this is essentially finding a discrete logarithm in the original group, which as we have seen, is thought to be computationally difficult for some groups. Thus this approach does not typically produce an efficiently computable pairing for the original group. In particular, there are some (many) concrete cyclic groups for which a computationally efficient pairing has yet to be identified, and whose discovery would be surprising to cryptographers.

If E is an elliptic curve group over a finite field F_q, then the group G for a pairing is typically taken to be a cyclic subgroup E with prime order different than the characteristic of F_q. In this case, a pairing on G is usually derived from one of two theoretical constructions on elliptic curves, the Weil pairing and the Tate pairing, whose background and derivation are highly technical and beyond the scope of our discussion.

The group G_t associated with an elliptic curve pairing is usually a subgroup of the multiplicative group of F_{q^k}, for an appropriate choice of k. Since G_t has prime order p', this choice of k has to satisfy the relation $p' \mid q^k - 1$, and the smallest k such that p' divides $q^k - 1$ is called the **embedding degree** of p' in F_q. In particular, pairing computations using the Weil and Tate pairings involve doing some arithmetic in terms of elements in F_{q^k}, so if k is large, these computations can be infeasible. This gives rise to the idea of a "pairing friendly" elliptic curve over a finite field, whose group order has a large prime divisor with small embedding degree such that pairings can be computed efficiently.

It has been shown under "plausible assumptions" that for most elliptic curves E over F_q, a large prime divisor p' of $\#E$ has embedding degree in F_q proportional to p', meaning that pairing computations are not feasible for elliptic curve groups of cryptographic size. Thus pairing-friendly elliptic curve groups are special.

Example 8. A commonly employed elliptic curve group with hard discrete logarithm is Curve25519. It is defined by a so-called *montgomery equation*[1]

$$y^2 = x^3 + 486662x^2 + x$$

over a prime field F_p, where $p = 2^{255} - 19$. The elliptic curve group has order $8 * p'$ where p' is a large prime which can be written as $2^{252} + (2.7742\ldots) \cdot 10^{37}$. The subgroup of order p' is generated by a point with x-coordinate 9.

The embedding degree k of p' over F_p is a very large integer, $(1.2062\ldots) \cdot 10^{75}$, which is about $p'/6$, and so is of the expected order of magnitude. This means that computing an elliptic curve pairing on Curve25519 would require working in a field represented by a tuple of over 10^{75} coefficients in F_p.[2] This is, of course, not computationally feasible.

The existence of elliptic curve pairings leads to some very interesting consequences for cryptographic applications.

One is that it introduces another method to attack the DLP, by using a pairing to translate an instance of the discrete log problem in the original elliptic curve group into an instance in the finite field F_{q^k}. As we have seen, there are sub-exponential algorithms for solving the DLP over a finite field, so if the embedding degree k of an elliptic curve group E is too small, then the DLP in F_{q^k} may be easier to solve than the DLP in E, reducing the security of protocols relying on the difficulty of this computation.[3]

In particular, when choosing an elliptic curve group to work with which admits an efficient pairing, it is important to select a group whose embedding degree is small enough for efficient computation, but large enough that the discrete logarithm in the corresponding finite field F_{q^k} is at least as hard as in the original elliptic curve group. Usually, parameters are selected so that these discrete logarithms are about equally difficult with the best known algorithms, which optimizes for computational efficiency while at the same time preserving the discrete log difficulty of the group.

Another important consequence of pairings is that if G is a cyclic cryptographic group which admits an efficient pairing e to G_t, then the pairing allows us to compare products "in the exponent". In more detail, if g is a generator for G, then difficulty of the discrete logarithm means that for an integer a, it is computationally infeasible to determine a when given only the group element g^a. This means that if a represents some secret data, then raising g to the a-th power acts as a sort of "hiding" operation on the data.

For integers a, b and c, if given g^a, g^b, and g^c, it is possible to check that $c = a+b$ without knowing these numbers, by checking that the product $g^a g^b$ is equal to g^c. However, it is not generally possible to check whether $c = ab$ given only these values. If you are able to solve

[1] This representation is in contrast with the standard short Weierstrass equation, and elliptic curves which can be expressed by an equation of this form allow a nice fixed-time algorithm for computing products, called the Montgomery ladder. This is important for various applications in which variable time execution of elliptic curve computations allow for a type of attack called a "side-channel attack" which uses variations in computation time to exfiltrate cryptographic secrets from otherwise secure protocols.

[2] Some work has been done to reduce the size of computations in the corresponding group G_t by using the fact that G_t itself is only of size p, and so in theory could be represented using p bits. However, the best that has been accomplished reduces the size of representation by at most a small integer factor, which does not change the feasibility of the computation.

[3] One attack making use of this observation is called the "MOV" attack.

the discrete log problem with base g, then this problem is easy; compute the discrete logs a, b and c, and check if the product relation is satisfied. Lacking access to an efficient solution to the discrete logarithm, the task becomes more difficult.

The following notions are useful when talking about the security properties of a cryptographic group related to these product comparisons. The *computational Diffie-Hellman (CDH) problem* asks us to compute a product in the exponent for a given group. More specifically, given a random generator g of a cyclic group G, as well as powers g^a and g^b chosen randomly, we are asked to compute the group element g^{ab}. The related *decisional Diffie-Hellman (DDH) problem* asks us to instead only decide whether a given group element is equal to g^{ab}: given a random generator g of G, random powers g^a and g^b, and an element h which is with equal probability either g^{ab} or a random power g^c, decide whether $h = g^{ab}$. The group G is said to satisfy the **CDH assumption** if no efficient algorithm solves the computational Diffie-Hellman problem, and it is said to satisfy the **DDH assumption** if no efficient algorithm solves the decisional Diffie-Hellman problem.[4]

Thus, in these terms, an efficient solution to the discrete log problem immediately solves the computational and decisional Diffie-Hellman problems. On the other hand, an efficient pairing on a cryptographic group with hard discrete logarithm produces an interesting middle-ground between the two problems: to check if $h = g^{ab}$ in the decisional Diffie-Hellman problem, it is enough to check that

$$e(g^a, g^b) = e(g, h)$$

Thus the pairing provides a ready solution to the DDH problem, but says nothing useful about the CDH problem, i.e. this computation doesn't give an obvious way to find g^{ab} when given only g^a and g^b. A group admitting an efficient pairing therefore does not satisfy the decisional Diffie-Hellman assumption, but may still satisfy the computational Diffie-Hellman assumption. For a group satisfying only the CDH assumption, we therefore find ourselves at an interesting state of affairs: somebody given elements g, g^a, g^b, and h can easily verify that $h = g^{ab}$, but only somebody knowing the exponents a and b themselves can actually compute g^{ab}. This property will be useful later when we will use discrete logarithms in such groups to produce efficient *polynomial and vector commitment schemes*, which will be central to the succinctness part of SNARK protocols.

[4]A bit more work needs to go into defining these assumptions rigorously in order to capture the fundamentally asymptotic property of an algorithm being "efficient". Specifically, the property must be defined with respect to an infinite family of cyclic groups and a probabilistic random selection function which generates, for a given n, a "random group from the family of size at most n and a random generator of this group".