COS 533: Advanced Cryptography — Princeton University
Lecture 6 (February 23, 2021)
Lecturer: Mark Zhandry — Scribe: Geoffrey Mon

Notes for Lecture 6

1 Introduction

We continue to discuss elliptic curves; in particular, we discuss how they are applied in cryptography and why they are useful, and we will also cover some known attacks on elliptic curves, especially using pairings. We also discuss other applications of pairings and how they can also be useful for building cryptography.

2 Elliptic curves

Definition 1 *An* elliptic curve *E over a field $\mathbb{F}$ is the set of solutions (x, y) to the equation $y^2 = x^3 + ax + b$.*

If we have $\mathbb{F} = \mathbb{R}$, then for two points P, Q, we can define addition $P+Q$ geometrically as drawing a line that includes both P and Q, finding the third point where the line intersects E, and mirroring this point across the x-axis. This operation can be computed efficiently, and the details and corner cases are discussed in the previous lecture. In particular, the addition operation is associative (due to some intuition from algebraic geometry, out of scope for this course), and so we can define a group using the points of E as the elements, using addition as the group law, and using a point at $y = \infty$ as the identity.

For cryptographic purposes, we will use elliptic curves over *finite fields*, because we need to be able to represent field elements on a computer with a finite number of bits. In doing so, we lose the geometric intuition from using $\mathbb{R}$ as the base field, but the algebraic formulas for the addition group law still make sense over any field. In order to apply elliptic curves to cryptography, we would like to know more about the structure of an elliptic curve E over a finite field $\mathbb{F}$. Specifically, how big is E, and does it have large prime-order cycles, which would be useful for constructing cyclic subgroups on which we can generate discrete log instances?

Theorem 2 (Hasse) *If the field size is $|\mathbb{F}| = q$, then*

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

In particular, $|E|$ is very close to the field size.

The rough intuition for this is that about half of $\mathbb{F}$ will be quadratic residues[1], and we expect that $x^3 + ax + b$ should "evenly distribute" points over $\mathbb{F}$, and so for roughly half of the elements x, $x^3 + ax + b$ will be a quadratic residue. Each quadratic residue has two roots, so for a fixed x, if $y^2 = x^3 + ax + b$ is a quadratic residue, then it has two solutions (x, y). Therefore, the size of E should be about the size of $|\mathbb{F}|$.

The takeaway of this result is that *picking a large field will yield a elliptic curve that is a large group*, and so we will aim to use large fields for security. Additionally, if we can generate a group of prime order, then the group will also be cyclic, and we can build cryptography off of discrete log on that group. In order to utilize an elliptic curve for discrete log or other similar problems, we need to determine $|E|$ efficiently (e.g. to check if it is prime), and we also need to generate elements of E efficiently (e.g. to find a generator, if $|E|$ is prime and hence E is cyclic).

- *Schoof's algorithm* can compute $|E|$ in polynomial time (that is, in time polynomial to $\log q$, as the input is the field size).
- To generate a point in E, we can make use of the quadratic residue intuition from before: generate a random $x \in \mathbb{F}$, and compute $x^3 + ax + b$, which will be a quadratic residue with probability $\approx 1/2$; if $x^3 + ax + b$ is a quadratic residue y^2, then compute one of its square roots y. Since this succeeds with probability $\approx 1/2$, we will need to repeat about 2 times in expectation to successfully find a point in E. All of these steps can be done efficiently for a finite field.

2.1 Security of elliptic curves

Why use elliptic curves in cryptography? The original motivation is that elliptic curves seem to provide the same level of security as other cryptographic groups while requiring less resources. For instance, the best attacks on discrete log instances over finite fields $\mathbb{F}$ run in subexponential time:

$$\exp(O((\log|\mathbb{F}|)^{1/3} \cdot (\log\log|\mathbb{F}|)^{2/3}))$$

Therefore, in practice, we need $|\mathbb{F}|$ to be quite large in order to maintain a good level of security. On the other hand, the best known discrete log attacks on many elliptic curves run in time $|\mathbb{F}|^{1/2}$, which is exponential in $O(\log|\mathbb{F}|)$. These attacks, such as the *baby-step giant-step* algorithm, can be applied to any group: for any group of order p, discrete log can be solved in $O(p^{1/2})$ group operations[2]. Hence, assuming there are no faster attacks, we can use a smaller (i.e. more efficient) elliptic curve group to get the same level of security as a larger finite field.

[1] k is a quadratic residue if $\exists m.\, m^2 = k$ in $\mathbb{F}$

[2] Interestingly, while baby-step giant-step requires $O(p^{1/2})$ space, there exist other attacks (Pollard's rho algorithm) with similar runtime but that requires $O(1)$ space.

3 Pairings

A notable non-generic attack on elliptic curves (i.e. an attack that exploits the structure of elliptic curves, rather than applying to any group) is the *MOV attack*[3], which constructs a pairing for an elliptic curve.

Definition 3 *A* pairing *(also called a* bilinear map*) is a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_2$ that satisfies the following:*

- non-degenerate*: for a fixed g, $e(g, h)$ always outputs the identity element 1 for all h, only if g is the identity element (and vice versa for a fixed h). In other words, e should not always output the identity, unless its input is the identity.*
- bilinear*: for $g, g_i, h, h_i \in \mathbb{G}$,*

$$e(g_1 \times g_2, h) = e(g_1, h) \times e(g_2, h)$$
$$e(g, h_1 \times h_2) = e(g, h_1) \times e(g, h_2)$$

  Note that the multiplication within the parameters of e denotes the group law of $\mathbb{G}$, while the multiplication of the images of e denotes the group law of $\mathbb{G}_2$.

In the case that $\mathbb{G}$ is cyclic (which will be true in most cryptography-relevant cases), then it is also true that

$$e(g^a, h^b) = e(g, h)^{ab}$$

If we have a pairing e that can be efficiently evaluated, we can reduce certain problem instances in $\mathbb{G}$ to instances in $\mathbb{G}_2$. This is a problem because for a prime-order elliptic curve E on $\mathbb{F}$ with $|\mathbb{F}| = q$, there exists[4] an efficiently-computable *Weil pairing* $e : E \times E \to \mathbb{F}_{q^k}$, where k is the smallest integer such that $|E|$ divides $q^k - 1$. Intuitively, the multiplicative subgroup of $\mathbb{F}_{q^k}$ has $q^k - 1$ elements, and using the nondegeneracy and bilinearity properties of e, we have that the image of e in $\mathbb{F}_{q^k}$ is a multiplicative subgroup of $\mathbb{F}_{q^k}$. Therefore, the cardinality of the image of e, which is $|E|$, must divide $q^k - 1$. If we pick E such that k is e.g. a small constant, then we will be able to solve discrete log and decisional Diffie-Hellman instances on E by reducing them to instances on $\mathbb{F}_{q^k}$, where we can use the subexponential attacks on finite fields. So, we should be careful to pick E such that k is large enough that this reduction to finite fields does not provide any speedup over the generic $|E|^{1/2}$ time attacks.

3.1 Attacking discrete log

If a pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_2$ exists, then discrete log in prime order $\mathbb{G}$ can be reduced to discrete log in $\mathbb{G}_2$. Suppose that we are given $(g, h = g^a) \in \mathbb{G}^2$, and we would

[3]named for Menezes, Okamoto, and Vanstone

[4]ignoring a few corner cases

like to recover a. Then, compute $g' = e(g, g)$ and $h' = e(h, g)$; by bilinearity, $h' = e(g^a, g) = e(g, g)^a = (g')^a$. Therefore, solving the discrete log instance $(g', h') \in \mathbb{G}_2^2$ will yield an answer for the original instance $(g, h) \in \mathbb{G}^2$, but potentially modulo a different number. But because $e(\cdot, g)$ for a fixed g is a group homomorphism, and by nondegeneracy and the prime order of $\mathbb{G}$, the image of $e(\cdot, g)$ is a subgroup of $\mathbb{G}_2$ with the same prime order as $\mathbb{G}$, and so the answer to (g', h') is precisely the answer to (g, h).

3.2 Attacking decisional Diffie-Hellman

A pairing can also be used to reduce decisional Diffie-Hellman (DDH) instances. Suppose we are provided with $(g, h = g^a, u = g^b, v = g^c)$, where $c = ab$ or c is uniformly random and independent of a and b; we would like to determine whether $c = ab$ or c is random. We can compute $e(g, v)$ and $e(h, u)$:

- If $c = ab$, then $e(g, v) = e(g, g)^{ab} = e(h, u)$.
- If c is random, then $e(g, v) = e(g, g)^c$, but $e(h, u) = e(g, g)^{ab}$.

Hence, we can test whether $e(g, v) = e(h, u)$; if $c = ab$ then they will always be equal, but if c is random, then the two values will almost always be different.

3.3 Application: three-party non-interactive key agreement

While we have seen uses of pairings for attacks, we can also use pairings to build cryptography. In particular, a *three-party non-interactive key agreement* protocol is where three parties A, B, C each broadcast public messages, and can compute a secret that is known only to the three parties. Note that we are interested in a non-interactive protocol, that is a party cannot read others' messages in order to decide what to send out[5].

We have already seen the ElGamal public key encryption scheme, which also works as a two-party non-interactive protocol: A picks some $a \leftarrow \mathbb{Z}_p$ and broadcasts g^a, while B picks some $b \in \mathbb{Z}_p$ and broadcasts g^b; each party can compute $g^{ab} = (g^a)^b = (g^b)^a$ as the shared secret, and using the DDH hardness assumption, the shared secret cannot be distinguished from a random group element, even though g^a and g^b are public.

To extend this to three parties, have party A pick a private secret $a \leftarrow \mathbb{Z}_p$, compute g^a, and broadcasts g^a; similarly, have B and C each compute and broadcast g^b, g^c. Now,

[5]A multiparty interactive key agreement protocol for arbitrarily many parties can be as follows: each party generates a public key and a secret key, and sends their public key to a designated party A; then, A generates the shared key, encrypts it with each public key, and broadcasts the ciphertexts so that each party can use their own secret key to recover the shared key.

A can compute the shared secret $k = e(g,g)^{abc}$, as $e(g^b, g^c)^a = e(g,g)^{abc}$. Similarly, B can compute $e(g^a, g^c)^b = e(g,g)^{abc}$ and C can compute $e(g^a, g^b)^c = e(g,g)^{abc}$, so everyone has the same shared secret. In contrast with ElGamal, where the pairing helps to break DDH, it is unclear whether the pairing can help us break this scheme. The security of this scheme is based on the *bilinear DDH* assumption: given $(g, h = g^a, u = g^b, v = g^c, w = e(g,g)^d)$, no PPT adversary distinguishes (with non-negligible advantage) whether $d = abc$ or d is uniformly random[6].

It is a big challenge to find (or disprove the existence of) useful *multilinear maps*, the natural generalization to bilinear maps, which for e.g. would be useful for multiparty non-interactive protocols. It seems that although elliptic curves have efficiently computable bilinear maps, but it is unclear whether they have useful multilinear maps. Later in the course, we may see "noisy" multilinear maps which would also be useful for multiparty non-interactive protocols.

3.4 Application: signature scheme

We can also briefly describe how to use a pairing to build a signature scheme for messages in $\mathcal{M}$, assuming we have a hash function $H : \mathcal{M} \to \mathbb{G}$ that behaves like a random oracle[7]. Then, let $\mathsf{sk} = a \leftarrow \mathbb{Z}_p$, and let $\mathsf{pk} = (g, g^a)$. To sign, return $\mathsf{Sign}(\mathsf{sk}, m) = (H(m))^a$, and for $\mathsf{Ver}(\mathsf{pk}, m, \sigma)$, test if $e(g,\sigma) \overset{?}{=} e(h, H(m))$. By bilinearity, if the signature is valid, then $e(g,\sigma) = e(g, H(m))^a$, which equals $e(h, H(m)) = e(g, H(m))^a$.

Intuitively, an adversary gets to see g, g^a, and $H(m)$ which is equal to g^b for some b; forging a signature requires computing $(H(m))^a = g^{ab}$, which is hard using the computational Diffie-Hellman (CDH) assumption. Note that while a pairing lets us solve DDH, it does not allow us to solve CDH, and indeed we rely on CDH still being hard to prove that this signature scheme is hard. This property of pairings is sometimes referred to as *gap Diffie-Hellman*. If we build this scheme using an elliptic curve E on a field of size, say, 2^{256}, and we are careful to pick E such that discrete log is not easy (using Section 3.1), then we can have short (i.e. 256-bit) signatures where the best-known attacks require 2^{128} time, which is infeasible. This is essentially the best practical signature scheme known, and a major challenge in cryptography research is to find a scheme that yields shorter signatures. We cannot get shorter signatures with the same security using pairings or elliptic groups, because of the generic baby-step giant-step attack, but there are other advanced (although impractical) techniques such as obfuscation, which we may see later in the course, that can yield shorter signatures.

[6]There are no known attacks for bilinear DDH other than solving discrete log, and it's unclear if we can base bilinear DDH security off of other security assumptions, since a pairing helps us to break DDH.

[7]i.e. H looks like a uniformly random function