# Lecture 6 - Elliptic Curves & Pairings

Gidon Rosalki
2026-06-10

Notice: If you find any mistakes, please open an issue in the github repository

## 1. Introduction

Last semester, we discussed all manner of parts of asymmetric encryption, and always began with "Let $\mathbb{G}$ be a prime order group", and from here continued onto assumptions such as DLOG, DDH, CDH, and so on. However, we never discussed how we made these groups.

If we consider the group $(\mathbb{Z}_p, +)$, we will note that these assumptions are not true for this group. If we instead consider the group $(\mathbb{Z}_p^*, \times)$, then it would appear that these assumptions hold true over this group. This group has $p-1$ elements, which is an even amount. The complexity of the assumptions is dependent on the smallest subset of the group, for which they hold true. Since the size of our group is even, we can relatively trivially break it into smaller groups, and then it is very easy to solve these problems on top of these smaller groups.
Instead of using this group, what we do is we take a subgroup of $(\mathbb{Z}_p^*, \times)$, of prime order, and work on top of that. So, we will pick $p$ such that it is a safe prime, where $p = 2q + 1$.

Our security parameter for DLOG is now $2^{O(\sqrt[3]{\log p})}$. So, if we want a security parameter of 128 bits, then we need it to hold that every value in our group will have about 3072 bits, since we have required $\log p = 3072$.

This is quite a lot, so performing these computations is now very expensive. To resolve this, we begin considering elliptic curves.

## 2. Elliptic Curves

*Definition*: Pythagorean pairs are pairs of numbers where $x^2 + y^2 = 1$.

*Definition*: Fermat triples are triples of numbers where $x^3 + y^3 = z^3$.
We will note that there are no such triples in $\mathbb{Z}$, nor for any power greater than 2.

*Definition*: Diophantine pairs are pairs where $y^2 = x^3 + Ax + B$

Diophantine pairs define elliptic curves. Currently, we do not know of any trivial method to break this type of encryption. Currently, the best known method costs $O(2^{\frac{\log p}{2}}$. So, if you want 128 bit security, then you only need $\log p = 256$, which is much more efficient than the traditional method. As a result, this is the method that is used, but which group should we use? We have standardised on a few specific elliptic curves:

- secp256r1:

$$p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$$
$$y^2 = x^3 - 3x + b : b = \text{magic}$$

- secp256r1:

$$p = 2^{255} - 19y^2 = x^3 + 486662x^2 + x$$

These appeared because some people chose them, and they appear good. What is particularly screwy is that there are good $b$'s, and bad $b$'s, and the ones that we use are simply ones that were published by the NSA as "probably good" at some point. Are they secretly an NSA backdoor into modern cryptography? Maybe.

# 3. Pairings

A pairing is

$$e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$$

such that it enables

1. Bilinearity : $e(g^a, g^b) = e(g, g)^{ab}$
2. Non-Degeneracy : If $g$ generates $\mathbb{G} \implies e(g, g)$ generates $\mathbb{G}$
3. Efficiency : $e$ may be computed in polynomial time

So, why is this interesting? It potentially helps us break DDH (1993). If we can convert 2 groups to another group, which is easier to break, then we can use the same ideas to break DDH. That is to say, DDH is that it is hard given $g^a, g^b$ to compute $g^{ab}$. Since this concept directly seems to imply we can use it to break DDH, we can either be exceedingly careful with our groups, or just not assume DDH in the original groups.

## 3.1. Joux 2000

Recall Diffie Hellman from the 70s, Alice selects $a$, Bob $b$, they send each other $g^x : x \in \{a, b\}$, and then may compute the entire key $g^{ab}$ by raising what they received to the power of their own number.

How do we do this for 3 people, Alice, Bob, and Claire? Each will do the standard exchange with each other. Let us consider Claire. Claire finishes the exchanges with $g^a, g^b, c$. She may now compute $e(g, g)^{ab}$, and raise that to $c$, resulting in the shared key $e(g, g)^{abc}$.

This is DBDH: Decisional Bilinear Diffie Hellman, where we say that

$$\left(g^a, g^b, g^c, e(g, g)^{abc}\right) \approx \left(g^a, g^b, g^c, e(g, g)^r\right)$$

## 3.2. BLS 01

Boneh - Lynn - Shacham signatures. Let us briefly consider some other signatures.

| Scheme | Group | Security | Group | Size of signature | bits |
|---|---|---|---|---|---|
| RSA | $\mathbb{Z}_n^*$ | $2^{\sqrt[3]{\log n}}$ | $\approx 3000$ | One group element | $\approx 3000$ |
| ECDSA | Elliptic curve | $\sqrt{p}$ | 256 | 2 group elements | 512 |
| Schnorr | Elliptic curve | $\sqrt{p}$ | 256 | 1 group element, + hash | 384 |
| BLS | Elliptic curve | $\sqrt{p}$ | 256 | 1 group element | 256 |

Table 1: Signatures

How do we sign? We need:

- KeyGen: $sk = a, vk = (g^a, g)$
- Sign(a, m): $\sigma = H(m)^a$ for a hash function $H : \{0,1\}^n - \}\mathbb{G}$
- Verify$(g^a, m, \sigma)$: $e(H(m), g^a) == e(\sigma, g)$ which should both return $e(g, g)^{aH(m)}$

**Correctness**:

$$e(\sigma, g) = e(H(m)^a, g) = e(g^{za}, g) = e(g, g)^{za}$$
$$e(H(m), g^a) = e(g^z, g^a) = e(g, g)^{za}$$

Assuming CDH in $\mathbb{G}$, show that BLS is **secure**: Note that in $\mathbb{G}$, DDH is false, since DDH is just to differentiate, where CDH requires computation, and we may differentiate quite easily, since this group is bilinear.
Let us consider the game:

$$\begin{aligned} A &\leftarrow vk = g^a \\ A &\to m \\ &\leftarrow H(m)^a \\ &\to m^*, \sigma^* \end{aligned}$$

Let us state that $H$ is some public function, which receives queries from either side, and deterministically returns the results. We may assume w.l.o.g:

1. $A$ queries $H$ on $m^*$. This is true, since $A$ need not use the result of $H$
2. $A$ never queries $H$ twice with the same $m$. This is true since $H$ is deterministic, so there would be no point in calling it twice on the same input, we may simply store the previous input, or add a wrapper, or something.

So, if CDH holds then BLS is secure: We shall create a reduction to a CDH adversary $B$, which accepts $g^a, g^b$, and returns $g^{ab}$.
$A$ may request 2 things, request a signature, and request a hash. $A$ expects to receive a verification key.

1. $B$ sends $A$ $(g, g^a)$, so $g^a$ is the $vk$
2. $\forall H$ query on $m_i$, $g^{r_i} : r_i \leftarrow \mathbb{Z}_p$
3. Guess index $i^*$ on which $A$ will request $H(m^*)$, and on the query $i^*$ to $H$, return $g^b$
4. $\forall$ sign query on $m_i$, return $(g^a)^{r_i}$
5. If we were right, then we get $m^*, \sigma^*$, and we may return the signature. Otherwise, fail

We still need to show that the distributions of the signatures, and of what $B$ returns are statistically equivalent, but this is quite easily done. We have also assumed the programmable random model, where $B$ controls $H$, which is not necessarily trivial, but the alternative is *much* more complex.

# 4. Identity Based Encryption

In traditional public key encryption, every person has a secret key, and a public key, someone maintains the public key database, and through this, we may all communicate. The idea here is to simplify this such that we encrypt with respect to someone's identity, as in their email address, or name, or some such.

So, we have some certificate authority, which distributes the secret key according to your email. This idea was first introduced by Shamir in 1984. In around 2001, Boneh Franklin (as in BLS) proposed the most popular form of identity based encryption.

$$\text{Setup}(1) = msk, mpk$$
$$\text{KeyGen}(msk, id) = sk_{id}$$
$$\text{Enc}(mpk, id, m) = ct_{id,m}$$
$$\text{Dec}\big(sk_{id}, id, ct_{id,m}\big)$$

Yes, this does mean that our CA knows your secret key. We are working under the assumption that we are all working together in a company, that is our CA, and we have no secrets from them, so by allowing this, we have saved $O(n^2)$ space.

**Correctness**:

$$\forall msk, mpk \leftarrow \text{Setup}$$
$$\forall id, \forall sk_{id} \leftarrow \text{KeyGen}(msk, id)$$
$$\forall m, \text{Enc}(mpk, id, m) \rightarrow ct$$
$$\text{Dec}(sk_{id}, id, ct) = m$$

**Security**: The game is as follows.

1. We send the adversary $mpk$, having sampled $mpk, msk$
2. Adversary requests the secret key for some $id$
3. The adversary sends $id^*, m_0, m_1$, where $id^*$ is the $id$ it is trying to break, and someone for whom it has not received the $sk$
4. We sample $b \leftarrow \{0, 1\}$, and send $\text{Enc}(m_b)$
5. $A$ returns $b'$, and wins if $b' == b$

We may further strengthen this by having another round of id requesting, and even CCA style with multiple encryption requests, and so on.

## 4.1. Boneh Franklin IBE Scheme

1. Setup: $s \leftarrow \mathbb{Z}_p$, and let $msk = s, mpk = g^s = h$
2. KeyGen$(msk, id)$: $sk_{id} = H(id)^s$
3. Enc$(mpk, id, m)$: $r \leftarrow \mathbb{Z}_p, u = g^r, v = e(H(id), h^r) \cdot m$, output $c = (u, v)$
4. Dec$(sk_{id}, id, ct = (u, v))$: $z = e(sk_{id}, u)$ output $m = \frac{v}{z}$

**Correctness**:

$$e(sk_{id}, u) = e(H(id)^s, g^r) = e(g^{\alpha s}, g^r) = e(g, g)^{\alpha sr} = e(H(id), h^r)$$

So when we divide, we are left with $m$.

**Security**: We will use a reduction to the game in order to break DBDH, where we distinguish between $e(g, g)^{abc}$ and $e(g, g)^r$, given $g^a, g^b, g^c$. So, we will construct $B(h_1 = g^x, h_2 = g^y, h_3 = g^z, h_T)$
We will begin with the same assumptions, that $A$ never queries $H$ twice on the same input, and that $A$ always queries $H(id)$ before sending it.

1. Send $A$ the $mpk = h_1$. Sample $j \leftarrow \{1, ..., q\}$ as the index of $id^*$ from all the $q$ queries that $A$ will do
2. $H$ query ($i$-th such query)
   - $i = j \implies H(id_i) = h_2$
   - $i \neq j \implies H(id_i) = g^{r_i} : r_i$ is randomly sampled
3. KeyGen$(id)$: $h_1^{r_i}$
4. $h_3, h_T \cdot m_b$

If $h_T$ is random, then there no chance. However, if $h_T = e(g, g)^{xyz}$, and we have correctly chosen $j$, then it has some minor advantage. The pk seen by $A$ is random, and all the random oracle queries appear completely random. So, everything seen by the adversary appears completely random, as it should. We just need to show that the encryption appears correct, which is to say, $(u, v)$ has the correct distribution. $u = g^z$, so does, and

$$\begin{aligned} v &= e(h_2, h_1^z) \\ &= e(g^{y, g^{xz}}) \\ &= e(g, g)^{x, y, z} \end{aligned}$$

as required.