Elliptic Curve Cryptography

Christine Alar, Kyle Hansen, Cooper Young

March 2021

Contents

1 | Overview of Cryptography

1.1 | Introduction

When sending a message online, how do we make sure that the message is not intercepted and read by a third party? Cryptography is the study and practice of techniques to keep communication secure from being accessed by third parties.

In this paper, we discuss how elliptic curves can be used to ensure communication is secure over a public domain using the Elliptic-curve Diffie-Hellman (ECDH) key exchange, as well as its vulnerabilities and comparisons with other algorithms.

We first introduce the Diffie-Hellman key exchange, and several relevant definitions.

1.2 | Diffie-Hellman Key Exchange

A few quick definitions:

- *cryptographic key*: a string of characters used to convert a message to cipher text, or vice versa
- *cipher text*: the encrypted form of a message
- *insecure channel*: a way of transferring data that is subject to eavesdropping

The Diffie Hellman (DH) key exchange is a method of securely exchanging cryptographic keys over a public channel. Originally, two correspondents needed to exchange keys via some secure physical method, and DH was one of the first methods to allow two correspondents without prior contact to create a shared secret key over an insecure channel.

We begin with an analogy using paints to introduce the concept of Diffie-Hellman.

Alice and Bob publicly agree on an arbitrary starting color (yellow). Alice adds her secret color (red) to her own paint can to make an orange mixture, and Bob adds his secret color (teal) to his can to make a blue mixture. The two exchange their paint cans publicly, and then add their secret color to the delivered paint can. They each end up with a common secret paint color (brown).

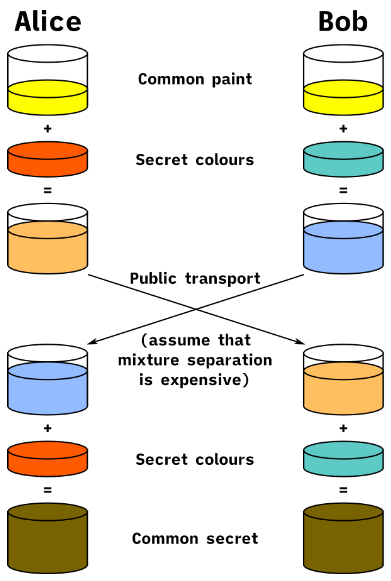

Figure 1.1: Diffie-Hellman paint analogy.

A third party witnessing this exchange, although witnessing the common starting color and transported mixtures, would have a difficult time determining the final secret color.

We next introduce the original implementation of Diffie-Hellman which uses multiplication of integers modulo a prime p, and a primitive root modulo p denoted by g. [2]

Say Alice and Bob publicly agree on a prime p and primitive root g.

Alice then chooses a secret integer a and calculates $A = (g^a \bmod p)$, while Bob chooses a secret integer b and calculates $B = (g^b \bmod p)$.

Over a public channel, Alice sends Bob A and Bob sends Alice B.

Alice now calculates $B^a = (g^b)^a = g^{ba} \bmod p$ while Bob calculates $A^b = (g^a)^b = g^{ab} \bmod p$, and since $g^{ab} = g^{ba} \bmod p$, the two end up with a common secret number.

Example 1.1. Let $p = 29$ and $g = 8$, a primitive root modulo 29.

Say Alice chooses $a = 4$ and Bob chooses $b = 3$.

Then Alice calculates $g^a \bmod p$: $8^4 \equiv 7 \bmod 29$.

Bob calculates $g^b \bmod p$: $8^3 \equiv 19 \bmod 29$.

Publicly, Alice and Bob exchange the values $A = 7$ and $B = 19$.

Now Alice calculates $B^a \bmod p$: $19^4 \equiv 24 \bmod 29$.

Bob calculates $A^b \bmod p$: $7^3 \equiv 24 \bmod 29$.

As shown above, the two end with the common secret number 24. △

The publicly known values are p, g, g^a, and g^b. Just knowing these values, it takes extremely long times to compute $g^{ab} \bmod p = g^{ba} \bmod p$ by any known algorithms, which is what gives strength to this method.

It should be noted that the above example is not particularly secure since there are only 29 possible solutions to $n \bmod 29$. To make this method secure we need to choose large values for a, b, and p. However, it is known that for p a prime that is at least 600 digits then even "the fastest modern computers using the fastest known algorithm cannot find a given only g, p, and $g^a \bmod p$." [3] This problem, known as the *discrete logarithm problem*, will be discussed further in Chapter 3.

2 | Elliptic Curves in Cryptography

In this chapter we'll explore two ways that elliptic curves are used in cryptography. The first is the Elliptic Curve Diffie-Hellman, which is a protocol that establishes a shared key over an insecure channel. The second is the Elliptic Curve Digital Signature Algorithm, which is used to authenticate digital content.

2.1 | ECDH Protocol

Elliptic curve cryptography seeks to exploit the group structure of elliptic curves over finite fields to generate keys for asymmetric cryptography. One of the most prominent ways that this is done is through the Elliptic Curve Diffie-Hellman (ECDH) protocol; two parties, Alice and Bob, start with a public elliptic curve $E : y^2 = x^3 + ax + b$, a prime p, and a generating point G on E which has a large order. Alice and Bob both have a private keys which take the form $n_A, n_B \in \mathbb{Z}_{>0}$, respectively. The protocol then proceeds as follows

Alice: generates her public key by computing $n_A G$ (adding the generator to itself n_A times).

Bob: computes his public key, $n_B G$.

[The two exchange public keys.]

Alice: determines the shared key by computing n_A(Bob's secret key).

Bob: determines the shared key by computing n_B(Alices's secret key).

By the end of this procedure, Alice and Bob have the same shared key since $n_A(n_B G) = (n_A n_B)G = n_B(n_A G)$. The pair can now utilize symmetric key algorithms to encrypt and decrypt text or other forms of information that they wish to send to one another.

The security of ECDH comes from the fact that given the public information and a point nG, it is difficult to determine the integer n. This is explored more fully in Chapter 3, but for intuition, let's consider the elliptic curve $E : y^2 = x^3 + 7$. This curve is referred to as the secp256k1 curve and become popular when Bitcoin started using it for their Elliptic Curve Digital Signature Algorithm (ECDSA). Over the finite field $\mathbb{F}_{17}$, our curve looks like this

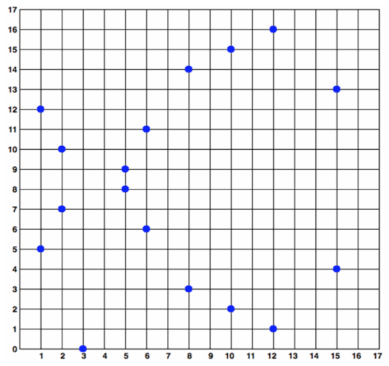

Figure 2.1: secp256k1 over $\mathbb{F}_{17}$

When we start with the generating point $G = (15, 13) \in E$, we can label the points as follows

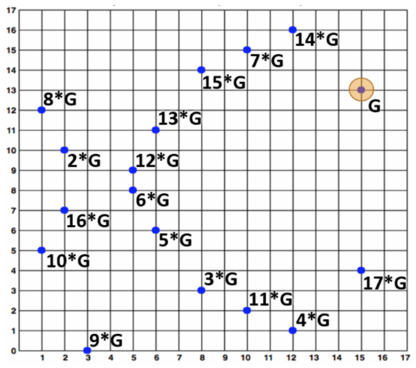

Figure 2.2: G generates all points on E

As we can see, given some point $nG \in E$, it's not easy to determine which n led to that point. Furthermore, the security of ECDH scales with the size of our prime p, and typically our prime is chosen to be 256 bits long.

The algorithm mirrors the typical Diffie-Hellman key exchange (DH), except it uses elliptic curve point multiplication instead of modular exponentiation. One benefit of using ECDH is that it is much less computationally expensive than DH. For example, according to the WebTrust-certified certificate authority provider GlobalSign, to match the security of a DH protocal with a 3000 bit key, one would need to use a prime of only 256 bits when using ECDH [9]. The security of ECDH will be explored in Chapter 3, but this fact alone gives credence to the benefit of elliptic curves in cryptography.

2.2 | ECDH Example

Now that we have a working understanding of ECDH, let's outline an implementation of it using python. We start by importing the python library *tinyec* and preparing the curve "secp256r1."

One of the main differences between secp256k1 and secp256r1 is that the former is a Koblitz curve while the latter is not. That is, secp256k1 takes the form $y^3 = x^3 + ax + b$ with $a = 0$, while for secp256r1, neither a nor b are trivial. In general, the etymology for an elliptic curve identifies useful characteristics of the curve.

```
import tinyec.ec as ec
import tinyec.registry as reg

curve = reg.get_curve("secp256r1")
print('curve:', curve)
```

```
curve: "secp256r1" => y^2 = x^3 + 115792089210356248762697446949407573530086143415290314195533631
308867097853948x + 41058363725152142129326129780047268409114441015993725554835256314039467401291
(mod 115792089210356248762697446949407573530086143415290314195533631308867097853951)
```

For example, "sec" in an elliptic curve's names stands for "Standards for Efficient Cryptography," and when a curve starts with "secp," it means that it's taken over a prime field $\mathbb{F}_p$. Having a t in the name denotes the base field takes the form $\mathbb{F}_{2^n}$, having a k denotes a Koblitz curve, and an r denotes verifiably random parameters.

Our curve secp256r1 comes equipped with a generating point

```
print('generator: (', curve.g.x, ', ', curve.g.y, ')')
```

```
generator: ( 48439561293906451759052585252797914202762949526041747995844080717082404635286 ,
36134250956749795798585127919587881956611106672985015071877198253568414405109 )
```

and the 78-digit prime as seen when printing secp256r1 above. The SubGroup class in tinyec takes arguments (p, g, n, h) where p is the prime we are working over, g is the generating point, n is the order of g, and h is the cofactor (the cardinality of $E(\mathbb{F}_p)$ divided by n) and the Curve class takes in arguments $(a, b, field, name)$ where $E : y^2 = x^3 + ax + b$, $field$ is the field we are working over, and $name$ is the nickname we give the curve. Hence, we could have also defined our curve by hand

```
field = SubGroup(p=115792089210356248762697446949407573530086143415290314195533631308867097853951,
                 g=(curve.g.x, curve.g.y), n=0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551,
                 h=0x1)
curve_by_hand = Curve(curve.a, curve.b, field=field, name='secp256r1')

print('curve:', curve)
```

```
curve: "secp256r1" => y^2 = x^3 + 115792089210356248762697446949407573530086143415290314195533631308867097853948x
+ 41058363725152142129326129780047268409114441015993725554835256314039467401291 (mod 115792089210356248762697446694
94075735300861434152903141955336313088670978539511)
```

Next, we import secrets, which is used to generate random numbers for Alice and Bob's private keys.

```
import secrets

Alice_PrivKey = secrets.randbelow(curve.field.n)
Alice_PubKey = Alice_PrivKey * curve.g
print("Alice's public key:", Alice_PubKey.x, ', ', Alice_PubKey.y, ')' '\n')

Bob_PrivKey = secrets.randbelow(curve.field.n)
Bob_PubKey = Bob_PrivKey * curve.g
print("Bob's public key:", Bob_PubKey.x, ', ', Bob_PubKey.y, ')')
```

```
Alice's public key: 38852616408081238969091097157024996768655800675473046599015926981508625638023 ,
47437174069289959752830002256600570177722402914385873874490204603627962660765 )

Bob's public key: 78053943024279867629462808037552191907997009238385059144584818386608286214374 ,
50001034204974712706689521837605606445396512744707062929940701875829831666986 )
```

Finally, these public keys are exchanged, allowing Alice and Bob to have a shared key. Often times, it is secure enough to just use the x-coordinate of the computed point:

```python
Alice_SharedKey = Alice_PrivKey * Bob_PubKey
print("x-coord of Alice's shared key:", Alice_SharedKey.x, '\n')

Bob_SharedKey = Bob_PrivKey * Alice_PubKey
print("x-coord of Bob's shared key:", Bob_SharedKey.x, '\n')

print("The shared keys are equal:", Alice_SharedKey == Bob_SharedKey)
```

```
x-coord of Alice's shared key: 104519385082665032655866195472312894077549247918698979605461995514792496010481

x-coord of Bob's shared key: 104519385082665032655866195472312894077549247918698979605461995514792496010481

The shared keys are equal: True
```

This shared key can now be used in various symmetric-key algorithms to securely transfer information between Alice and Bob.

2.3 | ECDSA Protocol

Another common use of elliptic curves in cryptography is through the Elliptic Curve Digital Signature Algorithm (ECDSA). The protocol has two parts: the signing and the verifying. Together, they show that the signer was the one who sent the message and that the message wasn't altered in transit.

Suppose Alice has some message, M, that she wants to send to Bob. She wants to show that she is the one that sent M and make sure the message wasn't altered before it got to Bob. Beforehand, the pair agree on an elliptic curve E, a generating point G, and the large prime order n for G. Alice has a private key in the form of a random integer $d_A \in [1, n-1]$, and a public key, d_AG. The signing algorithm carried out by Alice is as follows:

`Signing Algorithm`

1.) Compute $h := HASH(M)$, where $HASH$ is a hash function like SHA-256 whose output is an integer

2.) Generate a random integer $k \in [1, n-1]$

3.) Calculate the point $(x_1, y_1) = kG$

4.) Calculate $r \equiv x_1 \pmod{n}$ and $s \equiv k^{-1}(z + rd_A) \pmod{n}$

5.) Return the signature: (r, s)

If either r or s is zero, we return to step 2 and compute a new random integer. Note that the reason we require n to be prime is so that we are guaranteed the existence of k^{-1} in step 4. Once Alice has computed the pair of integers (r, s), she sends the message M to Bob first and then the signature (r, s). Bob then uses the verifying algorithm below, which outputs a boolean value representing if the signature is valid or not

`Verifying Algorithm`

1.) Check the validity of d_AG by verifying it's on E and $n(d_AG) = O$.

2.) Compute $h = HASH(M)$ with the same has function as in the signing algorithm

3.) Calculate $u_1 \equiv hs^{-1} \pmod{n}$ and $u_2 \equiv rs^{-1} \pmod{n}$

4.) Recover the point $(x_1, y_1) = u_1G + u_2Q_A$

5.) $r \equiv x_1 \pmod{n}$, the signature is valid, and it is invalid otherwise

It's not obvious why this verification works, but the key observation is that the point recovered in step 4 of the verifying algorithm is the same point computed in step 3 of the signing algorithm. To see this, notice that

$$\begin{aligned} u_1G + u_2Q_A &= u_1G + u_2d_AG \\ &= (u_1 + u_2d_A)G \\ &= (hs^{-1} + rs^{-1}d_A)G \\ &= (h + rd_A)s^{-1}G \\ &= (h + rd_A)(z + rd_A)^{-1}kG \\ &= kG \end{aligned}$$

It's worth noting that for the signing algorithm, a new random integer k must be used every time or else someone could compute the private key d_A; suppose that Eve knows two signatures (r, s) and (r, s') which used the same k value to make certificates for the messages M and M'. First, Eve can compute the integer k by recalling that $s \equiv k^{-1}(h + rd_A) \pmod{n}$ and $s' \equiv k^{-1}(h' + rd_A) \pmod{n}$. These equations give us

$$s - s' \equiv k^{-1}(h - h') \pmod{n} \qquad \Longrightarrow \qquad k \equiv (h - h')(s - s')^{-1} \pmod{n}$$

and from this we can recover

$$d_A \equiv r'(sk - h) \pmod{n}$$

In 2010, a group noticed that Sony used a static k for their ECDSA and used the technique above to recover the private key that Sony used to sign software for their PlayStation 3 console. If different random integers k are generated for each signature, the technique is secure, and ECDSA is widely used today in the TLS protocol by popular sites such as Wikipedia, YouTube, and Facebook. In fact, Facebook uses the secp256r1 curve from the previous section for their ECDSA to generate a certificate during TLS.

3 | Security

It is a common occurrence that cryptographic systems find their security in mathematical problems which are hard to solve—that is, for which there is either no known deterministic algorithm which solves the problem, or for which any current deterministic algorithms are intractable, meaning that they require too much time or space in order to operate efficiently and effectively. For example, there is currently no deterministic polynomial-time algorithm which solves the problem of factoring (large) composite numbers [10]. Another such problem which is hard to solve, and which is related to Elliptic Curve Cryptography, is known as the "Discrete Log Problem", or DLP.

3.1 | DLP and ECDLP

The classical version of DLP can be described as follows:

DLP:

Fix a prime p. Given $a, b \in \mathbb{F}_p$ subject to the condition that $b \equiv a^x \pmod{p}$ for some $x \in \mathbb{Z}$, determine the value of x. That is, compute the discrete logarithm $L_a(b) = x$.

This problem is hard, provided p is large enough. When p is small, one can perform a brute force search to determine x. For example, given $p = 11$, if $a = 2$ and $b = 9$, one can compute by brute force that $2^x \equiv 9 \pmod{11}$ is solved by $x = 6$. However, as long as the factorization of $p - 1$ contains a large enough prime, this problem is resistant to the Pohlig-Hellman attack, and if p itself is large enough (say, $p >> 10^{20}$), the problem is resistant to so-called "Baby Step, Giant Step" attacks [10, 202,207]. We will encounter variations of these attacks for Elliptic Curves in Section 4.1 and 4.2 respectively.

There is an elliptic curve variation of this problem, appropriately named the "Elliptic Curve Discrete Log Problem", or ECDLP, which is described as follows:

ECDLP:

Fix a finite field $\mathbb{F}_q$ for $q = p^k$ with p prime, and an elliptic curve $E/\mathbb{F}_q$. Given $G, P \in E(\mathbb{F}_q)$ subject to the condition that $P = nG$ for some $n \in \mathbb{Z}$, determine the value of n. That is, compute the "elliptic curve discrete log" $EL_G(P) = n$.

The similarities with DLP are obvious, though the difficulty of solving ECDLP does not now depend only on the choice of prime p, but rather on the finite field $\mathbb{F}_q$, as well as on the choice of curve $E/\mathbb{F}_q$. As will be seen in Section 3.3 and Chapter 4, these choices must be made carefully to ensure the hardness of ECDLP. In

Section 5.1, we will see that, even though there is no proof that ECDLP is a computationally hard problem, such a proof would have tremendous implications for the theory of computation. Hence, ECDLP is considered a hard enough problem to provide cryptographic security.

3.2 | ECDHP: Applying ECDLP to ECDH

Before considering the difficulty of solving ECDLP, let us see how it is relevant in the context of ECDH. Recall the description of ECDH in Section 2.1, and consider things now from the perspective of the malicious eavesdropper Eve. By assumption, Eve knows that Alice and Bob are using ECDH to create a shared key K. Throughout the entire exchange, Eve is assumed to gain access to any information that is transmitted or made public, and so in particular, Eve is assumed to have access to the following information:

1.) The curve and finite field $E/\mathbb{F}_q$,

2.) The generating point $G \in E(\mathbb{F}_q)$,

3.) The points computed by each of Alice and Bob, $Q_A = n_A G$ and $Q_B = n_B G$ respectively (which are transmitted publicly during the key exchange)

Alice and Bob each independently compute the shared key $K = n_A n_B G = n_B n_A G$ from the information given, as well as from the fact that the values n_A and n_B remain secret. If Eve could obtain a copy of this key K, then Eve to be able to eavesdrop on their correspondences which use that key. Hence, it is Eve's goal to solve the following problem, known as the "Elliptic Curve Diffie-Hellman Problem", or "ECDHP" for short:

ECDHP:

Fix a finite field $\mathbb{F}_q$ for $q = p^k$ with prime p, and an elliptic curve $E/\mathbb{F}_q$. Given $G, Q_A, Q_B \in E(\mathbb{F}_q)$ subject to the condition that $Q_A = n_A G$ and $Q_B = n_B G$ for some $n_A, n_B \in \mathbb{Z}$, determine the value of $K = n_A n_B G$.

With this setup in mind, we can make the following, straightforward claim:

Proposition 3.1. *ECDHP reduces to ECDLP.*

Proof. Suppose that Eve has solved ECDLP, and is listening in as above. Since she has access to the points G and $Q_A = n_A G$, for example, she will be able to compute n_A, since she has solved ECDPL. Then, because she also has access to $Q_B = n_B G$, she can compute $n_A Q_B = n_A n_B G = K$. Thus, ECDLP reduces to ECDLP. □

Let's return our attention to the curve sepc256k1 from Section 2.2, and for the sake of tractability, let's consider an example over a small finite field.

Example 3.2. Alice and Bob have chosen to work with the curve sepc256k1 (i.e. $E : y^2 = x^3 + 7$) over the finite field $\mathbb{F}_q = \mathbb{F}_{17}$, with the generator $G = (15, 13) \in E(\mathbb{F}_{17})$. During the key exchange, Eve sees that Alice and Bob send the points $Q_A = (8, 3)$ and $Q_B = (10, 16)$ respectively. If she could solve ECDLP, Eve could compute $K = (12, 1) = 4G$ using this information alone, and could then listen in on their future correspondences. △

3.3 | Hardness of ECDLP, and $\#E(\mathbb{F})$

In selecting an elliptic curve $E/\mathbb{F}_q$, along with a generating point $G \in E(\mathbb{F}_q)$, over which to perform computations, one must be careful. The *domain parameters* of an elliptic curve scheme is a tuple of information which allows one to test the security of the scheme in question. In particular, in addition to the curve and generating point, this tuple contains:

1.) The order n of G,

2.) The *cofactor* $h := \#E(\mathbb{F}_q)/n$.

The hardness of ECDLP, and certain vulnerabilities of the scheme, depend on wise choices of curves that guarantee the cofactor h is sufficiently small (typically $h \leq 4$). However, one should also ask that $n = o(G)$ be fairly large (e.g., at least $n > 2^{160}$ according to [4, 172]) to avoid certain attacks such as the Pohlig-Hellman attack in Section 4.1, as will be seen later.

Because of this, we see that choosing an elliptic curve scheme with the right domain parameters will depend on being able to count the number of points on the curve, in order to effectively compute h. Moreover, as will be seen in Chapter 4, knowing $\#E(\mathbb{F}_q)$ will help determine the runtime of certain algorithms which attack ECDLP, and hence this parameter in particular is crucial for determining the security of the scheme in question.

Recall the following theorem of Hasse:

Theorem 3.3 (Hasse, 1936)**.** *An elliptic curve $E/\mathbb{F}_q$ has $\#E(\mathbb{F}_q) = q + 1 - t$ points where $|t| \leq 2\sqrt{q}$.*

In general counting the number of points on a curve by direct methods (i.e. counting them one at a time) would be infeasible, and so it was a significant problem to find a polynomial-time algorithm which counts the points on the curve in an efficient way. Hasse's Theorem gives an *estimate* for the number of points on E, but does not give a way to determine exactly the number of points.

However, more recently Schoof's Algorithm (1985) provides a polynomial-time, deterministic algorithm for computing t given any elliptic curve $E/\mathbb{F}_q$. Though the exact description of this algorithm and its runtime is beyond the scope of this paper, the main ideas make use of the the Frobenius Endomorphism $\varphi\colon E/\mathbb{F}_q \to E/\mathbb{F}_q$ defined by $\varphi(x : y : z) = (x^q : y^q : z^q)$. In fact, the brunt of the proof relies on showing one can compute the trace $\mathrm{tr}\varphi \pmod{l}$ efficiently for enough primes l. Combining these results with the Chinese Remainder Theorem (in a manner similar to that seen later in Section 4.1) one can then compute the value of $t = \#E(\mathbb{F}_q)$ efficiently. For an in-depth description of Schoof's Algorithm and its applications, we direct readers to [6] and [1, 109 ff].

The main benefit to Schoof's Algorithm (and the various improvements to the algorithm which have been developed since Schoof's breakthrough) is that one has a feasible method for computing the number of points on a curve, and hence one can compute domain parameters for curves efficiently as well. This allows us to more efficiently classify curves according their cryptographic security. For a list of some such acceptable curves, we refer the reader to FIPS recommended curves [8, 89 ff].

4 | Vulnerabilities

In this section, we discuss various attacks that might be made against cryptosystems which employ elliptic curve cryptography. Each of these attacks have analogues in standard (non-elliptic curve based) cryptosystems. However, elliptic curve based cryptography has an added advantage of being resistant to what are known as "index-calculus" attacks, a well-known attack in standard cryptosystems (see [4, 165 ff] for details). The point of this section is not to dissuade use of elliptic curve cryptography, but rather to highlight the importance of using curves with "good" domain parameters, and following conventions such as those laid out the ANSI Standard [11, 14-18].

In what follows, we assume the following conventions, as in the description of ECDLP:

- $E : y^2 = x^3 + ax + b$ is an elliptic curve over $\mathbb{F}_q$
- $G \in E(\mathbb{F}_q)$ is a point of order $n = o(G)$
- $P = \alpha G$ for some $\alpha \in \mathbb{Z}$
- $\#E(\mathbb{F}_q)$ is the number of points on the curve
- $h := \#E(\mathbb{F}_q)/o(G)$ is called the *cofactor*

For pseudo-code algorithms implementing following attacks, and others, we direct readers to [4, 155 ff]. There are also attacks (e.g. "Person-in-the-Middle Attacks") which deal more with implementation rather than mathematical security, and as such their discussion is beyond the scope of this paper.

4.1 | Pohlig-Hellman Attack

When $n = o(G)$ is composed only of "small" prime factors, there is an attack known as the Pohlig-Hellman attack which exploits this weakness. A classical version of the Pohlig-Hellman attack on DLP is outlined in [10, 203], the ideas behind which have a natural analogue in ECDLP (as seen in [4, 155]). We explain this Elliptic Curve variation here.

The main idea is that, given the generator G and a point $Q \in \langle G \rangle$, one can compute the value of $\alpha \pmod{p^m}$ efficiently for small primes p and powers $m \in \mathbb{N}$, and then recombine these results using the Chinese Remainder Theorem. We outline this procedure for $p = 2$, followed by the more general situation.

First, suppose that $2 \mid n$, so that $2^m G = \mathcal{O}$ (the additive identity on E) for some $m \in \mathbb{N}$, and that $2^{m-1}G \neq \mathcal{O}$. Without loss of generality, we may assume $0 \leq \alpha < 2^m$. Writing α as its binary expansion

$$\alpha = \sum_{i=0}^{m-1} 2^i x_i \qquad (x_i \in \{0,1\}, i = 0,1,\ldots,m-1)$$

one can see that $x_0 = 0$ if and only if $2^{m-1}Q = \mathcal{O}$, since $o(G) = 2^m$. Inductively, suppose that the values of $x_0, \ldots, x_{r-1}$ have been determined for some $r < m$. Define

$$Q_r := Q - \left(\sum_{i=0}^{r-1} 2^i x_i\right) G = \left(\sum_{i=r}^{m-1} 2^i x_i\right) G = 2^{r-1} \left(\sum_{i=0}^{m-r-1} 2^i x_{i+r}\right) G.$$

Then since $x_r = 0$ if and only if $2^{m-r-1}Q_r = \mathcal{O}$, one can compute x_r iteratively by testing if $2^{m-r-1}Q_r = \mathcal{O}$, and hence one can compute $\alpha \pmod{2^m}$.

More generally, given a (small) prime p for which $p \mid n$, we may write $p^m G = nG = \mathcal{O}$ for some $m \in \mathbb{N}$. Writing α in its base-p expansion

$$\alpha = \sum_{i=0}^{m-1} p^i x_i \qquad (x_i \in \{0,\ldots,p-1\}, i = 0,1,\ldots,m-1)$$

define $Q_0^{(p)} := Q$, and for each $0 < r \leq m-1$ define

$$Q_r^{(p)} := Q - \left(\sum_{i=0}^{r-1} p^i x_i\right) G = \left(\sum_{i=r}^{m-1} p^i x_i\right) G = p^r \left(\sum_{i=0}^{m-r-1} p^i x_{i+r}\right) G.$$

Note then by construction that $p^{m-r-1}Q_r^{(p)} = x_r\left(p^{m-1}G\right)$, since $p^m G = \mathcal{O}$. Hence, our attack reduces to determining which value $x_r \in \{0,\ldots,p-1\}$ is such that $x_r\left(p^{m-1}G\right) = p^{m-r-1}Q_r^{(p)}$. For small enough p, brute force computations and searches make this appraoch feasible. Thus one can compute x_r using $Q_r^{(p)}$ and then inductively compute $Q_{r+1}^{(p)}$ using $x_0, \ldots, x_r$. We can therefore determine the value of $\alpha \pmod{p^m}$ in this iterative fashion.

This process demonstrates that if $n = p_1^{m_1} p_2^{m_2} \cdots p_s^{m_s}$ is the prime factorization of n, one can determine the values $\alpha \equiv \alpha_r \pmod{p_r^{m_r}}$ for each $r \leq s$ efficiently, provided the primes p_r are small enough. By the Chinese Remainder Theorem, there is a unique solution to this system of relations, and there is an efficient, deterministic algorithm for computing this solution. So one can reconstruct α using these relations, and thus determine $EL_G(P) = \alpha$.

In the following example, we use the notation $\frac{n}{p^r}$ in place of p^{m-r} as above.

Example 4.1. Let $E/\mathbb{F}_{37}$ be the curve $E : y^2 = x^3 + x + 4$. Let $G = (25,15)$, and consider $Q = (6,2) \in \langle G \rangle$. Note that $n = o(G) = 28 = 2^2 \cdot 7$, and write $Q = \alpha G$. Our goal is to determine the value of α.

$\underline{\alpha \pmod 4}$: First write $\alpha = x_0 + 2x_1 \pmod 4$. We see that $x_0 = 1$, since $\frac{n}{2}Q_0^{(2)} = 14Q = (19,0) \neq \mathcal{O}$. Next, we write $Q_1^{(2)} = Q - x_0 G = Q - G = (6,2) + (25,22) = (15,8)$. Since $7(15,8) = \frac{n}{2^2}Q_1^{(2)} = \mathcal{O}$, we have $x_1 = 0$. Therefore we conclude that $\alpha \equiv x_0 + 2x_1 \equiv 1 \pmod 4$.

$\underline{\alpha \pmod 7}$: Now, write $\alpha = y_0 \pmod 7$, its base-7 expansion. Since $\frac{n}{p}Q_0^{(7)} = 4Q = (15,8)$, we seek to find y_0 such that $y_0(\frac{n}{p}G) = \frac{n}{p}Q_0^{(7)}$, which is to say, such that $y_0(4G) = (15,8)$. Since $4G = (3,21)$ we want a value $0 \leq y_0 < 7$ for which $y_0(3, 21) = (15, 8)$. One can compute that $y_0 = 2$ gives the desired result, and hence $\alpha \equiv 2 \pmod 7$.

Therefore, $\alpha \equiv 1 \pmod 4$ and $\alpha \equiv 2 \pmod 7$, which implies by the Chinese Remainder Theorem that we have $\alpha \equiv 9 \pmod{28}$. One can check that indeed $(6, 2) = 9(25, 15)$. $\triangle$

The effectiveness of this attack depends entirely on the fact that n is divisible only by small primes. If n has even one prime factor that is large enough, the computations become exceedingly difficult to perform efficiently, and the attack becomes unusable.

4.2 | Baby Step, Giant Step Attack

Another deterministic algorithm, dependent on the order n of G, is that known as the "Baby Step, Giant Step" (or BSGS) attack. Again, the idea stems from a classical attack on DLP, which has an analogue in ECDLP. We examine this analogue below.

The idea behind this attack is that, given the points G and $P = \alpha G$, create two lists of size N for which $N^2 \geq n = o(G)$ as follows:

1.) A list of values βG for $0 \leq \beta < N$, the "baby steps",

2.) A list of values $P - N\gamma G$ for $0 \leq \gamma < N$, the "giant steps".

If there is a match between the lists, we have some values $\beta, \gamma \in \mathbb{N}$ such that $\beta G = P - N\gamma G$, and hence $P = (\beta + N\gamma)G$ so that $\alpha = \beta + N\gamma$. Note that such a match will always exist by the fact that $N^2 \geq n$. Hence this algorithm is guaranteed to terminate. The most obvious failing on the part of this algorithm is the fact that it requires tremendous amounts of memory (on the order of $\sqrt{n}$ for each list), which is intractable for large enough n. Though the attack is guaranteed to eventually terminate, it can be thwarted simply by choosing good parameters (e.g., such that n is large enough).

4.3 | Birthday Attacks

An attack similar to that of the BSGS attack is known as a "Birthday" Attack. The method of attack is essentially the same as that of the BSGS attack in Section 4.2 above, but with the significant difference that instead of computing both lists for *all* values of $0 \leq \beta < N$ and $0 \leq \gamma < N$, one computes two lists of random values for some "large enough" N. Namely, one computes the following two lists, where N is a large value:

1.) A list of values βG for N randomly chosen values of β

2.) A list of values $P - n\gamma G$ for N randomly chosen values of γ

By the same token as in the BSGS attack, if a match occurs in these two lists, one can successfully compute $\alpha = \beta + n\gamma$. The advantage to this attack over BSGS, is that by nature, it uses less memory, when N is selected so that $N^2 < n$.

On the other hand, there is no guarantee that the process of the Birthday Attack terminates. Instead, this algorithm is *probabilistic*, in that there is a high probability a collision between the two lists occurs. This probability is computed using the same methods as the well-known "birthday paradox" (hence the name "birthday attack") which explains, for example, the phenomenon that given a random selection of 23 people, there is about a 50% probability two of them have the same birthday.

Another advantage of using BSGS attacks over a birthday attack is that typically computations of points of the form βG or $P - n\gamma G$ can be simplified by processes such as a point doubling formula, or using previously computed points in some other way. Because of this, Birthday Attacks tend to be slower, and, though it may seem that the space-time trade-off could balance out, BSGS attacks end up being somewhat superior to Birthday attacks [10, 232].

4.4 | Other Attacks

Other attacks (such as Weil and Tate Pairing Attacks, attacks exploiting Weil Descent, and attacks on prime-field-anomalous curves) make use of the fact that points of the form $P = \alpha G$ are elements of the cyclic subgroup $\langle G \rangle \subseteq E(\mathbb{F}_q)$ generated by G. Such attacks are known as "Isomorphism Attacks" since they utilize the isomorphism $\langle G \rangle \approx \mathbb{Z}_n$. For more on these, we direct readers to [4, 168 ff]. Furthermore, as mentioned in Section 2.3, one must implement the right practices for ECDSA processes, such as avoiding repeated uses of certain parameters. This highlights the fact that there may be subtle attacks specific to the protocol being used, which do not require solving ECDHP or ECDLP. However, by careful adherence to standards laid out by experts in cryptography, such attacks can be easily prevented.

5 | Comparison with other Algorithms

Here we compare ECDH with other relevant key exchange algorithms.

5.1 | ECDH and DH

In the ECDH key exchange, one cannot find Alice's secret key unless they can solve the elliptic curve discrete logarithm problem discussed Chapter 3. According to [4], given a point of order n, one can circumvent an exhaustive search by choosing n large enough ($n > 2^{80}$) such that the search would be impracticable. There is no proof that ECDLP cannot be solved, but solving it is believed to be infeasible given we choose our parameters such that the attacks mentioned in the previous chapter are avoided. In particular, we circumvent the Pohlig-Hellman attack by choosing n with large enough prime divisor p ($p > 2^{160}$) so that attack becomes effectively useless. Note that it has not been proven that there is no algorithm that can efficiently solve ECDLP, however the non-existence of a polynomial time algorithm than can solve ECDLP would imply $P \neq NP$. One also cannot find Alice and Bob's shared secret key without solving the Elliptic Curve Diffie-Hellman problem (ECDHP). ECDH is stronger than DH in the sense that we can choose much smaller values for our parameters (DH needing primes of over 600 digits against ECDH needing primes of about 50 digits as exemplified above).

5.2 | ECDH and RSA

Another public key exchange algorithm is the RSA (Rivest-Shamir-Adleman) algorithm [4]. Breaking RSA can be done by solving the RSA problem, which relies upon the difficulty in integer factorization. Similar to ECDH, choosing sufficiently large parameters makes solving the RSA problem infeasible. However it has been proven that a quantum computer, if ever constructed, would break RSA, as it the computer would be able to compute factorizations in polynomial time via Shor's algorithm. Notably, ECDH would be susceptible to quantum computing as well.

5.3 | ECDH and SIDH

The Supersingular Isogeny Diffie-Hellman (SIDH) key exchange is another algorithm based upon DH which utilizes walks in a supersingular isogeny graph, and is designed to be resistant to quantum computer attacks. The nodes of an isogeny graph are given by all elliptic curves in a finite field $\mathbb{F}_q$ belonging to a fixed isogeny class, up to $\mathbb{F}_q-$isomorphism. A supersingular isogeny graph is an isogeny graph with nodes given by supersingular elliptic curves, which are elliptic curves with endomorphism rings isomorphic to an order in a quaternion algebra. The reader can explore this more in depth in [7].

References

[1] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

[2] W. Diffie and M. Hellman. New directions in cryptography, 1976.

[3] Adrian et al. Imperfect forward secrecy: How diffie-hellman fails in practice, 2015.

[4] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, New York, 2004.

[5] D. Hunter. Lecture Notes on Cryptography and Coding Theory: Elliptic Curve Cryptography. https://djhunter.github.io/cryptography/slides/16elliptic_curves3.html#/, 2020.

[6] T. Izu, J. Kogure, M. Noro, and K. Yokoyama. Efficient implementation of schoof's algorithm. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology — ASIACRYPT'98*, pages 66–79, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[7] De Feo L. Jao, D. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. in: Yang by. (eds) post-quantum cryptography. pqcrypto 2011. lecture notes in computer science, vol 7071. springer, 2011.

[8] National Institute of Standards and Technology. Federal Information Processing Standards: Digital Signature Standard (DSS), 2013.

[9] Julie Olenski. "what is ECC and why would I want to use it?". https://www.globalsign.com/en/blog/elliptic-curve-cryptography, 2015.

[10] W. Trappe and L. Washington. *Introduction to Cryptography with Coding Theory*. Pearson Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 2006.

[11] Accredited Standards Committee X9. Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1998.