In recent years, tremendous advances have been made in factoring integers and computing finite field discrete logarithms [5, p. 131]. Public key cryptography systems based around these problems have become less favorable and less frequently used, with elliptic curve cryptosystems filling this demand. This new cryptosystem's strength comes from its unusual group structure, which is much more difficult to understand than generic integer groups. As a result, it can provide the same degree of security as those traditional cryptosystems with only a fraction of the key size. For example, the minimum security level required of RSA has a key size of 1024 bits, which is 6.4 times larger than an equally-secure key size (160 bits) for an elliptic curve cryptosystem. The difference in equally-secure key sizes grows exponentially as bits increase [2, p. 19]. Hence, elliptic curve cryptosystems are more efficient than traditional RSA and discrete logarithm systems with respect to storing encrypted data and decrypting data, although they are somewhat slower at encrypting data (due to the complicated group structure mentioned earlier). However, compared to RSA and general discrete logarithm systems, very little research has been done into solving the underlying and unproven problem on which elliptic curve cryptosystems are based, the elliptic curve discrete logarithm problem (ECDLP).

In this chapter of the book, we will be examining algorithms to solve the ECDLP in the most optimal way. Our methodology will be similar to prior chapters, where we examined algorithms to find the local minimum of a function in the most optimal way.

Structure of Elliptic Curves

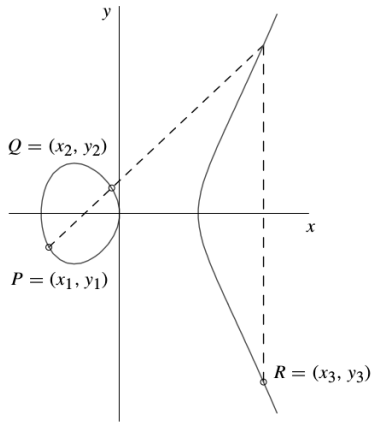
Before we describe the problem, we must formally define an elliptic curve. A curve over the finite field of integers[1] modulo q, $\mathbb{Z}_q$, with $q = p^r$ and p a prime not equal to 2 or 3, is the group of all points on the curve, satisfying the following two equations for $a, b \in \mathbb{F}_q$:

[1]Other fields (such as the rational and complex) are applicable, however, curves over them will always contain a torsion subgroup which is isomorphic to a subgroup of a curve over the field of integers. Thus, we will constrain our study to $\mathbb{Z}_q$. [5, p. 137].

$$y^2 = x^3 + ax + b$$

$$-(4a^3 + 27b^2) \not\equiv 0 \pmod{p}$$

In cases where $p = 2, 3$, the first equation accounts for an additional x or y term. As these are uncommon cases, they are omitted here, but can be found in [4, p. 168].The points satisfying the equations above are in the form (x, y), and are members of the Cartesian product $\mathbb{Z}_q \times \mathbb{Z}_q \cup \mathcal{O}$, where $\mathcal{O}$ is the group's identity. The group operator, denoted by "+", describes a complex permutation of the input points P and Q:



Addition: $P + Q = R$.

[2]

$$P + Q = (x_1, y_1) + (x_2, y_2) = (x_3, y_3) = \quad R$$

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$$

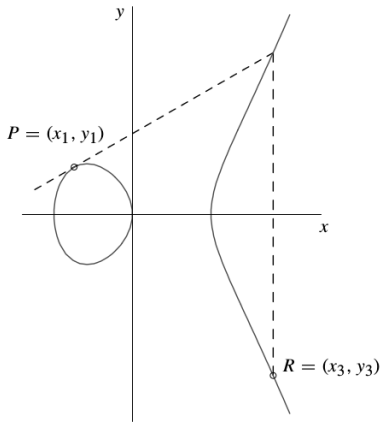
$$y_3 = -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3)$$

[5, p. 120]

This operation utilizes the slope of the chord between the two added points in order to derive their sum. Addition of a point with itself requires a slightly different formula that uses the tangent line to the curve at P (where a is the same a from the original equation $y^2 = x^3 + ax + b$):

Furthermore, we define the inverse of a point $P = (x, y)$ to be $(-P) = (x, -y)$. Observe that:

$$\begin{aligned} P + (-P) &= (x, y) + (x, -y) \\ &= \left(\left(\frac{-y - y}{x - x}\right)^2 - x - x, -y + \left(\frac{-y - y}{x - x}\right)(x - x)\right) \\ &= \left(\left(\frac{-2y}{0}\right)^2 - 2x, -y + \left(\frac{-2y}{0}\right)(0)\right) \\ &= \mathcal{O} \end{aligned}$$



Doubling: $P + P = R$. [2]

$$P + P = 2P = (x_1, y_1) + (x_1, y_1) = (x_3, y_3)$$

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1$$

$$y_3 = -y_1 + \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) \quad \text{[5, p. 120]}$$

Finally, we have defined our group of points on an elliptic curve with a well-defined operator, a closed set of elements and their inverses, and an identity element. Moreover, we know that the group is Abelian [7], and that the group is associative [12, p. 250]. Yet, with all this in mind, we cannot know the order of the group without performing a maximum of $O((\log_2 q)^8)$ additional bit operations [12, p. 254]. Thus, it will suffice to only know the lower and upper bounds on the order of the group (which bounds are derived from the elliptic curve zeta-function): $[q+1-2\sqrt{q},\ q+1+2\sqrt{q}]$ [5, p. 127]. Fortunately, the exact order of the group will not be absolutely necessary. Now we can describe the seemingly-intractable elliptic curve discrete logarithm problem:

> Let E be an elliptic curve defined over a finite field, and let P be a point of order n. Given $Q \in \langle P \rangle$, the elliptic curve discrete logarithm problem (ECDLP) is to find the integer α, $0 \leq \alpha \leq n$, such that $Q = \alpha P$.

But before we dive into the various algorithms for determining α in an optimal way, we must first qualify the measure by which we are optimizing, which involves a brief background in complexity theory.

Complexity of Algorithms

An algorithm's efficiency can be described by the number of bit operations it executes in terms of the size of its input. We will use efficiency as our criterion for how optimal an algorithm is at solving the ECDLP. Hence, we must first give some notation and background information on how we will measure efficiency.

The number of bit operations required to execute an algorithm, whose input is an integer n or smaller, where $log_2 n = r$ is the number of bits in n, and C is some (unimportant) constant, is:

$$\begin{aligned} L_n[\alpha] &= O(e^{C(ln(n))^{\alpha}(lnln(n))^{1-\alpha}}) \\ &\approx O(e^{C(r)^{\alpha}(ln(r))^{1-\alpha}}) \end{aligned}$$

In this notation, α is a real number between 0 and 1 denoting how exponential our algorithm is. This number is derived from the maximum number of steps an algorithm will take to execute. When α is 0, the maximum number of steps the algorithm will take is logarithmicly proportional to n. Such an algorithm runs in *polynomial-time*, which is computationally feasible:

$$\begin{aligned} L_n[0] &\approx O(e^{C \cdot ln(r)}) \\ &= O(r^C) \end{aligned}$$

However, when α is greater than 0, the number of steps is subexponentially proportional to n; when α reaches 1, the proportion becomes exponential. For our purposes, we will consider subexponential and exponential algorithms both as exponential, as their difference is not noticeable with the input sizes our algorithms are working with. When an algorithm runs in *exponential-time*, it is not computationally feasible:

$$L_n[1] \approx O(e^{C \cdot r})$$

Computationally feasible algorithms

The addition operator in the elliptic curve group, when used to add a point to itself k times, requires $O((ln\ k)\cdot(ln^3\ q))$ bit operations [4, p. 178]. In real-world applications, q would usually be no larger than a 384 bit integer [1], thus, the curve could have up to $e^{384}+1+2\sqrt{e^{384}} \approx 10^{129}$ different points on it, making the amount of time to compute all the points clearly infeasible. However, under certain conditions, a curve's points could demonstrate a pattern such than an algorithm can limit its search space to a significantly smaller number of points, often to the point that the algorithm runs in a computationally feasible amount of time.

To break these conditions into different cases, consider what data is already known when solving the ECDLP: the two points P and Q, the field $\mathbb{Z}_q$ over which the curve is defined, the equation describing points on the curve (in the form $y^2 = x^3 + ax + b$), and the approximate number of points on the curve (in the range $[q+1-2\sqrt{q},\ q+1+2\sqrt{q}]$).

Supersingular curves

As proven in [6], it is possible to imbed an elliptic curve over $\mathbb{Z}_q$ into the multiplicative group $\mathbb{Z}_{q^m}$ for some m. With this in mind, Menezes and Vanstone broke down all supersingular elliptic curves (where superginularity is a quality easily determined based on [3, p. 258]) into six classes (where $E(\mathbb{Z}_q)$ is the elliptic curve of order $q+1-t$ for some t):

(I) $t = 0$ and $E(\mathbb{Z}_q) \simeq \mathbb{Z}_q$
(II) $t = 0$ and $E(\mathbb{Z}_q) \simeq \mathbb{Z}_{(q+1)/2} \oplus \mathbb{Z}_2$ (and $q \equiv 3 \pmod 4$)
(III) $t^2 = q$ (and m is even)
(IV) $t^2 = 2q$ (and $p = 2$ and m is odd)
(V) $t^2 = 3q$ (and $p = 3$ and m is odd)
(IV) $t^2 = 4q$ (and m is even)

Using these six classifications of supersingular curves, the Menezes-Okamoto-Vanstone (MOV) reduction algorithm (with code found at the end of this chapter) can find a solution to the ECDLP in probabilistic-polynomial time, such that $\alpha = 0$. Next, we will describe one more class of curve whose group structure can be easily determined based on its order.

Curves which map to the additive group of integers

Suppose a curve $E(\mathbb{Z}_q)$ had an order of q. Then, define a mapping $\psi : E(\mathbb{Z}_q) \to \mathbb{Z}_q$ by [10, ch. VII]:

$$\psi_q(x, y) \equiv \frac{-x}{y} \pmod{q^2}$$

For some $P, Q \in E(\mathbb{Z}_q)$, this mapping can be used to easily calculate a discrete logarithm [11]:

$$\frac{\psi_q(qQ)}{\psi_q(qP)} \equiv \alpha \pmod{q} \qquad \text{such that } Q = \alpha P$$

Hence, the bottleneck computation for this method is computing the order of the elliptic curve, which has already been shown to be feasible. Therefore, we have shown that elliptic curves with prime power order or supersingularity should not be used for ECDLP applications, such as cryptographic algorithms. We will now move on to the most efficient known methods for solving the ECDLP in other cases. These methods are feasible for input sizes up to a certain point, with some begin feasible for up to 112 bits of input.

Computationally infeasible algorithms

For more general cases of the ECDLP, no computationally feasible algorithms have been discovered. However, it has not been proven than none exist [13]. In this section, we will survey the most efficient general-case algorithms used today, as well as analyze exactly how efficient they are.

Baby-step giant-step

The baby-step giant-step method was devised in [9] to find certain multiples of element orders that were known to lie in certain residue classes [13]. The concept of the general algorithms is simply as follows:

$$\text{To find } \alpha = \log_P Q \quad \text{for } |\langle P \rangle| = n, \text{ let } s = \left\lfloor \sqrt{n} \right\rfloor$$

$$\text{Find } \alpha = st + r \quad \text{where } 0 \leq t \leq s \text{ and } 0 \leq r \leq s$$

In our analogy, t represents our "giant-step", and r our "baby-step". Here is the algorithm:

1) For $0 \leq j \leq s-1$, compute the tuple $(j, jP+Q)$, storing the results sorted by second component in ascending order.

2) For $1 \leq i \leq s$, compute the tuple $(i, (s \cdot i)P)$, storing the results in the same way.

3) Iterate the two lists simultaneously, searching for two second components which are equal (such that $jP + Q = (s \cdot i)P$). If a match is found, return $i \cdot s + j$ as α.

Rather than relying on approximate results (as many number-theoretic algorithms do), this algorithm will definitely find an answer in $2 \cdot \sqrt{n} = \sqrt{4n}$ computations. However, because of how expensive the elliptic curve operator is, the complexity of baby-step giant-step becomes much larger when applied to the ECDLP. Using our estimated cost of k elliptic curve addition operations prior, the cost of baby-step giant-step would be $O(2 \cdot (ln\ (s+1)) \cdot (ln^3\ q))$ for the step generation and $O(s)$ comparisons, summing to $O(s + (2 \cdot (ln\ (s+1)) \cdot (ln^3\ q))) = O(\lfloor \sqrt{n}\ \rfloor + (2 \cdot (ln\ (\lfloor \sqrt{n}\ \rfloor + 1)) \cdot (ln^3\ q)))$. Since n is in the range $[q+1-2\sqrt{q},\ q+1+2\sqrt{q}]$, we will approximate using its max, $q+1+2\sqrt{q}$ (as is convention in big-O notation). Hence, baby-step giant-step requires:

$$
\begin{aligned}
& O(\lfloor \sqrt{q+1+\sqrt{q}}\ \rfloor + (2 \cdot (ln\ (\lfloor \sqrt{q+1+\sqrt{q}}\ \rfloor + 1)) \cdot (ln^3\ q))) \\
=& O(\lfloor \sqrt{e^r+1+\sqrt{e^r}}\ \rfloor + (2 \cdot (ln\ (\lfloor \sqrt{e^r+1+\sqrt{e^r}}\ \rfloor + 1)) \cdot (ln^3\ e^r))) \\
\approx& O(e^{\frac{3 \cdot r}{4}} + (2 \cdot \frac{3 \cdot r}{2} \cdot r^3)) \\
=& O(e^{\frac{3 \cdot r}{4}} + 3r^4)
\end{aligned}
$$

The complexity is clearly exponential. For a look at the code to this algorithm, see the end of this chapter. We will now move onto a more efficient algorithm for solving the ECDLP.

Pollard's rho

Pollard's Rho method was devised in [8]. It details an algorithm for the general discrete logarithm problem (applicable beyond the ECDLP) which runs in a number of steps proportional

to the square-root of the number of items. Hence, for an elliptic curve with order n, the algorithm will run in $O(\sqrt{n})$ steps.

The rho algorithm attempts to find two pairs of integers, (c, d) and (c', d'), such that $cP+dQ = c'P + d'Q$. Since the goal of the ECDLP is to find α in $Q = \alpha P$, one can rearrange the pairs of integers c and d such that [2, p. 157]:

$$\begin{aligned} cP + dQ &= c'P + d'Q \\ \leftrightarrow cP - c'P + dQ - dQ &= c'P - c'P + d'Q - dQ \\ \leftrightarrow (c - c')P &= (d' - d)Q \\ &= (d' - d)(\alpha P) \qquad \text{Since } Q = \alpha P \end{aligned}$$

$$\begin{aligned} \text{Thus} \qquad (c - c') &\equiv (d' - d)\alpha \pmod{n} \\ \leftrightarrow (c - c')(d' - d)^{-1} &\equiv \alpha \pmod{n} \end{aligned}$$

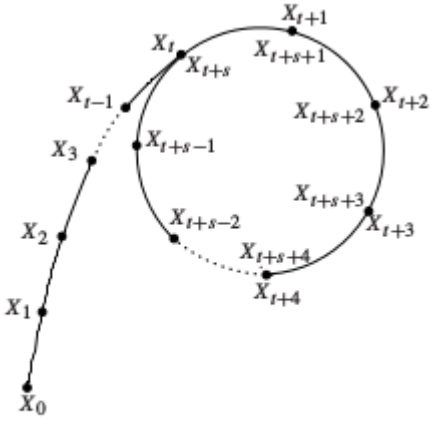
The methodology for finding these pairs of integers is as follows:

1) Define a partition of the points in $\langle P \rangle$ by some number of the least significant bits (e.g. the last five would lead to 32 partitions) such that $\langle P \rangle = \{S_1, S_2, S_3, \ldots, S_L\}$ where L is the number of partitions. Define a function $H(X) = j \ \leftrightarrow X \in S_j$.

2) Let two sequences of integers $\in [0, n]$ of length L be denoted as a_j and b_j.

3) Define a function $f(X_0) = X_1 = X_0 + a_{H(X_0)}P + b_{H(X_0)}Q$.

4) This sequence of Xs will eventually cycle such that there is some X_t for which $X_t = X_{t+s}$ because $\langle P \rangle$ is finite.



ρ-like shape of the sequence $\{X_i\}$ in Pollard's rho algorithm

[2]

t and s are each expected to be $\approx \sqrt{\frac{\pi \cdot n}{8}}$, culminating in the total number of steps being $\approx 2\sqrt{\frac{\pi \cdot n}{8}} = \sqrt{4 \cdot \frac{\pi \cdot n}{8}} = \sqrt{\frac{\pi \cdot n}{2}}$. This means that the total run-time of the algorithm (in terms of bit operations) is $L[\alpha = 0.5] = O(\sqrt{r}^{C \cdot \sqrt{\frac{\pi \cdot r}{2}}})$. Included at the end of this chapter is code for the rho method utilizing Floyd's cycle-finding algorithm.

Conclusion

In this section, we learned about elliptic curves and their group structure. With this knowledge, we were able to understand the elliptic curve discrete logarithm problem and task of optimizing solution-finding algorithms for it. We looked at the basic brute-force solution, several tricks for solving the problem under certain conditions, and the best-known general case algorithms. However, the question still remains: can the ECDLP be solved in polynomial time?

Code

```
# For all functions:
# Given: at least two points P and Q.
# Points are objects with standard elliptic curve operators represented
# by scalar multiplication and group operator addition

# Bruteforce search
# Given: two points P and Q.
# Assume that Q is in the subgroup generated by P
def brute_force(P, Q):
    alpha=1
    while Q != P_temp:
        P_temp+=P
        alpha+=1
    return alpha

# Baby-step Giant-step
# Given: two points P, Q, and the order of P, n
def bstep_gstep(P, Q, n):
```

```
    s=floor(n**(0.5))
    baby_steps=[ (j,((j*P)+Q)) for j in range(s)]
    giant_steps=[ (i,((s*i)*P)) for i in range(1,s+1)]
    bindex,gindex=0,0
    while bindex != len(baby_steps) and gindex != len(giant_steps):
        if baby_steps[bindex][1] > giant_steps[gindex][1]:
            gindex+=1
        elif baby_steps[bindex][1] < giant_steps[gindex][1]:
            bindex+=1
        else:
            return giant_steps[gindex][0]*s + baby_steps[bindex][0]
    return -1

# Pollard's Rho
# Given: two points P,Q, the order of P, n, and the number of branches L
# rho algorithm implements Floyd's cycle finding algorithm
from random import randint
def rho(P, Q, n, L):
    H = lambda x: mult_int(x) % L
    Rs = [(point[0],point[1], (point[0]*P)+(point[1]*Q)) for point in
            [(randint(0,n),randint(0,n),(0,0)) for j in range(L)]]
    start = randint(0,len(Rs))
    c_0, d_0, X_0 = Rs[start][0], Rs[start][1], Rs[start][2]
    c_1, d_1, X_1 = c_0, d_0, X_0
    while X_0 != X_1 or (c_0 == c_1 and d_0 == d_1):
        j = H(X_0)
        c_0, d_0 = (c_0 + Rs[j][0]) % n,(d_0 + Rs[j][1]) % n
        X_0 += Rs[j][2]
        for i in range(2):
```

```
        j = H(X_1)
        c_1, d_1 = (c_1 + Rs[j][0]) % n, (d_1 + Rs[j][1]) % n
        X_1 += Rs[j][2]
return ((c_0-c_1)((d_1-d_0)**(-1))) % n
```

References

[1] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Nist special publication 800-57. *NIST Special Publication*, 800(57):1–142, 2007.

[2] Darrel Hankerson, Scott Vanstone, and Alfred J Menezes. *Guide to elliptic curve cryptography*. Springer, 2004.

[3] Dale Husemöller. *Elliptic curves*. Graduate texts in mathematics. Springer-Verlag, 1987.

[4] Neal Koblitz. *A course in number theory and cryptography*, volume 114. Springer, 1994.

[5] Neal Koblitz. *Algebraic aspects of cryptography*, volume 3 of *Algorithms and computation in mathematics*. Springer, 1998.

[6] Alfred J Menezes, Tatsuaki Okamoto, and Scott A Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *Information Theory, IEEE Transactions on*, 39(5):1639–1646, 1993.

[7] Louis Joel Mordell. On the rational solutions of the indeterminate equations of the third and fourth degrees. In *Proc. Cambridge Philos. Soc*, volume 21, pages 179–192, 1922.

[8] John M Pollard. Monte carlo methods for index computation (mod p). *Mathematics of computation*, 32(143):918–924, 1978.

[9] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. Symp. Pure Math*, volume 20, pages 415–440, 1971.

[10] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

[11] Nigel P Smart. The discrete logarithm problem on elliptic curves of trace one. *Journal of cryptology*, 12(3):193–196, 1999.

[12] Douglas R Stinson. *Cryptography: theory and practice*. CRC press, second edition, 2002.

[13] Edlyn Teske. Square-root algorithms for the discrete logarithm problem (a survey). In *In Public Key Cryptography and Computational Number Theory, Walter de Gruyter*. Citeseer, 2001.