Solving the Elliptic Curve Discrete Logarithm Problem (ECDLP) using cryptographic algorithms.

1. Hemang Joshi (2024114012)
2. Amish Goyal (2024101090)
3. Shardul Joshi (2024101077)
4. Kausheya Roy (2024101085)
5. Harith Yerragolam (2025121009)

Codebase: github.com/karma-skz/ECC-algos-AAD-project
Presentation: Google Drive Link

Role Assignment to Team Members

Common Topics (All Members)

- **Foundations (Chapter 1, 5):** Basic ECC theory, Group operations, and Weierstrass forms.
- **ECDLP Theory (Section 4.1):** Definition of the problem, One-way functions, and Trapdoor mechanisms.
- **Security Comparison:** RSA vs. ECC security analysis.

Amish Goyal — Classical Search & Post-Quantum Theory

Sections: 3.1, 3.2, 3.8

- **Brute Force Search:** Baseline computational approach.
- **Baby-Step Giant-Step (BSGS):**
  - Standard algorithm ($O(\sqrt{n})$).
  - **[BONUS]** BSGS with LSB Leakage (Search space reduction).
- **Post-Quantum Analysis:**
  - Shor's Algorithm implications for ECDLP.
  - Quantum resistance of ECC.

Hemang Joshi — Subgroup & Anomalous Curve Attacks

Sections: 3.5, 4.2, 4.3.3

- **Pohlig-Hellman Algorithm:**
  - Decomposition of group order.
  - Chinese Remainder Theorem (CRT) reconstruction.
  - **[BONUS]** Handling smooth order curves.
- **Smart's Attack:**
  - Theory of Anomalous Curves

Harith Yerragolam — Randomized Cycle Algorithms

Sections: 3.3, 3.4, 4.3.1, 4.3.2

- **Pollard's Rho Algorithm:**
  - Cycle detection (Floyd's method).
  - Random walks and collision handling.
- **Pollard's Kangaroo (Lambda) Method:**
  - Discrete logs in a specific interval (Tame/Wild Kangaroos).
  - **[BONUS]** Optimization for range-bound keys.

Shardul Joshi — Advanced Summation Attacks

Section: 3.6

- **Las Vegas Algorithms:**
  - Summation polynomials (Semaev).
  - Point relation generation.
- **[BONUS] Gaussian Sampling:**
  - Approximating key hints using Gaussian distributions.
  - Probabilistic search space reduction.

Kausheya Roy — Implementation, Benchmarking & Optimization

Sections: 2, 3.7, 4.2

- **System Architecture & Optimization:**
  - C++ backend optimization for scalar multiplication.
  - Hybrid Python/C++ integration.
- **Benchmarking & Validation:**
  - Test case generation logic.
  - Justification of test set sufficiency.
  - Performance metrics.
- **Smart's Attack:** Implementation and practical analysis.
- **[BONUS] Web Interface:** Interactive demo for ECDLP visualization.

Acknowledgments

We would like to express our sincere gratitude to our course instructor, **Prof. Kannan Srinathan**, for his guidance and insightful lectures throughout the course. We also thank our teaching assistant, **Ishaan Romil**, for his constant support and helpful clarifications during discussions and practical sessions.

Contents

Chapter 1

Motivation and Rigorous Foundation

1.1 The Foundational Problem of Public-Key Cryptography

The entire architecture of modern public-key cryptography rests on a single, pivotal mathematical principle: the existence of a **trapdoor function**. In cryptographic terms, this is a function that is computationally efficient to compute in the forward direction but becomes computationally infeasible to invert without the possession of specific, secret information (the "trapdoor"). This asymmetry allows for the secure exchange of information and the generation of digital signatures over insecure channels.

1.1.1 Key Generation and the Trapdoor

In widely deployed systems such as RSA and Elliptic Curve Cryptography (ECC), security is not derived from the secrecy of the algorithm itself, but from the difficulty of reversing an exponential relationship in a finite group. The mechanism operates as follows:

1. **Secret Key (Private Key l)**: This is a large, randomly selected integer acting as the exponent (or scalar). It represents the trapdoor information known only to the owner.

2. **Public Key (Q)**: This value is derived by applying the one-way function to a known public base element P. In the context of ECC, this operation is scalar multiplication, denoted as $Q = l \cdot P$. While computing Q from l and P is efficient (using algorithms like Double-and-Add), it is designed to be irreversible.

3. **The Computational Challenge**: An attacker, having access to the public parameters P and the public key Q, faces the task of determining the secret scalar l. This is formally known as the **Discrete Logarithm Problem (DLP)**. The security of the cryptosystem relies entirely on the assumption that solving the DLP is impossible within a reasonable timeframe using current computing power.

1.2 RigorousProof of One-Wayness (Generic Group Model)

To formally justify the security of Elliptic Curve Cryptography, we analyze the Elliptic Curve Scalar Multiplication function as a candidate **One-Way Function (OWF)**.

Definition 1.2.1 (EC-OWF Candidate)**.** *Let $E(\mathbb{F}_p)$ be an elliptic curve group of prime order n, and P be a generator. We define the function $f : \mathbb{Z}_n \to E(\mathbb{F}_p)$ as:*

$$f(k) = k \cdot P \tag{1.1}$$

To prove this is a One-Way Function, we must demonstrate two properties:

1. **Easy to Compute:** Given k, finding $f(k)$ can be done in polynomial time.
2. **Hard to Invert:** Given $Q = f(k)$, finding k requires exponential time (negligible probability of success for polynomial-time adversaries).

1.2.1 Proof of Efficiency (Forward Direction)

Theorem 1.1. *The function $f(k)$ is computable in $O(\log k)$ group operations.*

Proof. Let k be represented in binary as $k = \sum_{i=0}^{m} b_i 2^i$, where $m \approx \log_2 n$. Using the **Double-and-Add** algorithm:

$$k \cdot P = \sum_{i=0}^{m} b_i (2^i P) \tag{1.2}$$

We compute the sequence of doublings $2P, 4P, 8P, \ldots, 2^m P$. This requires m point doublings. We then perform point additions for every bit $b_i = 1$. The total complexity is bounded by $2\log_2 n$ group operations. Since group operations in $\mathbb{F}_p$ take polylogarithmic time $O(\log^2 p)$, the total time is polynomial in the input size. □

1.2.2 Proof of Hardness (Reverse Direction - Lower Bound)

Since proving unconditional hardness implies $P \neq NP$, we prove hardness in the **Generic Group Model (GGM)**. In this model, the adversary creates an algorithm $\mathcal{A}$ that interacts with a "black box" oracle for group operations but cannot inspect the algebraic structure of point representations.

Theorem 1.2 (Shoup's Lower Bound for ECDLP)**.** *Let $\mathcal{A}$ be a generic algorithm that makes at most q queries to the group oracle. The probability that $\mathcal{A}$ solves the ECDLP in a group of prime order n is bounded by:*

$$Pr[\mathcal{A} \textit{ finds } k] \leq O\left(\frac{q^2}{n}\right) \tag{1.3}$$

Proof. **1. The Oracle Setup:** The oracle $\mathcal{O}$ maintains an internal list of pairs $\{(c_i, \sigma_i)\}$, where $c_i \in \mathbb{Z}_n$ is a scalar and σ_i is a random bit-string representation of the group element $c_i P$. The adversary $\mathcal{A}$ knows only the bit-strings σ. $\mathcal{A}$ submits two strings σ_i, σ_j and an operation (add/subtract). The oracle computes $c_{new} = c_i \pm c_j \pmod n$, generates a new random string σ_{new}, and returns it.

2. Information Leakage: The only way $\mathcal{A}$ learns information about the isomorphism between the random strings σ and the integers $\mathbb{Z}_n$ is by finding a **collision**. A collision occurs when the oracle returns a string σ_u that is identical to a previously seen string σ_v (where $u \neq v$). If $\sigma_u = \sigma_v$, then:

$$\begin{aligned} \text{Internal Scalar}_u &\equiv \text{Internal Scalar}_v \pmod n \\ \alpha_u + k\beta_u &\equiv \alpha_v + k\beta_v \pmod n \end{aligned}$$

where α, β are coefficients controlled by $\mathcal{A}$. This linear equation allows $\mathcal{A}$ to solve for the secret key k.

3. Probability Analysis: The problem reduces to the probability of finding a collision among random values. The adversary generates q values. These values correspond to distinct linear polynomials in k. For any two distinct polynomials F_i, F_j, the probability

that their evaluation collides at a random secret k is $1/n$. There are $\binom{q}{2}$ pairs of queries. By the union bound:

$$\Pr[\text{Collision}] \leq \binom{q}{2} \cdot \frac{1}{n} = \frac{q(q-1)}{2n} \approx \frac{q^2}{2n} \tag{1.4}$$

4. Conclusion: For the probability of success to be significant (e.g., ≈ 0.5), we require:

$$\frac{q^2}{2n} \approx \frac{1}{2} \implies q^2 \approx n \implies q \approx \sqrt{n} \tag{1.5}$$

Thus, any generic algorithm requires $\Omega(\sqrt{n})$ operations. Since $\sqrt{n} = 2^{\frac{1}{2}\log n}$, this is **exponential** in the bit-size of the key. □

This proof demonstrates that scalar multiplication is a One-Way Function provided that the group representation exposes no structure other than the group axioms. The failure of Index Calculus on elliptic curves (as discussed in Section 1.4) ensures that real-world ECC approximates this generic model.

1.3 Classical vs. Elliptic Curve DLP

While both RSA/Diffie-Hellman and ECC rely on the DLP, they operate over fundamentally different algebraic structures. Classical systems like Diffie-Hellman (DH) rely on the DLP defined over the **multiplicative group of integers modulo a large prime** p, denoted as $(\mathbb{Z}/p\mathbb{Z})^\times$.

> **Classical DLP**: Given a large prime p, a generator g of the group, and a target value h, the adversary must find the secret integer x such that $h \equiv g^x \pmod{p}$.

Elliptic Curve Cryptography (ECC) evolves this concept by replacing the multiplicative group of integers with the set of points forming an Abelian group on an elliptic curve E defined over a finite field $\mathbb{F}_q$. Here, the group operation is additive rather than multiplicative.

> **ECDLP**: Given an elliptic curve $E/\mathbb{F}_q$, a base point P of order n, and a public key point $Q \in E(\mathbb{F}_q)$, the adversary must determine the secret integer l such that $Q = lP$ (where lP represents adding P to itself l times).

1.4 Rigorous Justification: The Failure of Index Calculus

The primary motivation for adopting ECC over classical systems (RSA, DSA) is not merely efficiency, but the superior asymptotic hardness of the underlying problem. This distinction arises from the applicability of **Index Calculus** attacks.

1.4.1 The Index Calculus on $\mathbb{F}_p^\times$

Consider the classical DLP in $\mathbb{F}_p^\times$. The Index Calculus algorithm achieves sub-exponential complexity $L_p[1/3]$ by exploiting the arithmetic structure of the ring of integers $\mathbb{Z}$.

1. **Factor Base:** We select a bound B and define a factor base $\mathcal{B} = \{p_1, \ldots, p_k\}$ containing all primes $< B$.

2. **Smoothness:** An integer is B-smooth if it factors entirely into primes from $\mathcal{B}$. The probability that a random integer is smooth is relatively high.

3. **Relation Collection:** We look for exponents k such that $g^k \pmod p$ is B-smooth:

$$g^k \equiv \prod_{i=1}^{k} p_i^{e_i} \pmod p \tag{1.6}$$

Taking discrete logarithms on both sides yields a linear equation:

$$k \equiv \sum_{i=1}^{k} e_i \log_g p_i \pmod{p-1} \tag{1.7}$$

Once enough linear relations are collected, we solve for $\log_g p_i$ using linear algebra.

1.4.2 The Geometric Immunity of $E(\mathbb{F}_p)$

This attack methodology collapses completely when applied to the Elliptic Curve group $E(\mathbb{F}_p)$ due to the lack of a suitable notion of "smoothness."

Theorem 1.3 (Non-Decomposability of Points)**.** *Let $P \in E(\mathbb{F}_p)$. There is no efficient isomorphism or mapping $\phi : E(\mathbb{F}_p) \to \mathbb{Z}^k$ that decomposes a point P into a "product" of "smaller" prime points.*

Proof Sketch: In $\mathbb{Z}$, the size of an element is defined by its absolute value. Decomposition reduces an element to smaller prime factors. On an elliptic curve, a point $P = (x, y)$ is simply a coordinate pair.

1. **No Euclidean Valuation:** There is no valuation function $\nu(P)$ such that adding points results in a "smooth" breakdown. The group operation is additive, not multiplicative.

2. **Siegel's Theorem:** Points with integer coordinates are finite and rare. Most operations involve modular arithmetic where "size" wraps around p, destroying any ordering required for smoothness estimates.

3. **Failures of Lifting:** Attempts to lift $E(\mathbb{F}_p)$ to global fields (e.g., $E(\mathbb{Q})$) to find relations fail because the lift is not unique and the height of the lifted points grows exponentially, making relation finding harder than the original problem.

Consequently, attacks on ECC are restricted to **Generic Group Algorithms** (like Pollard's Rho), which treat the group operation as a black box.

- **RSA/DSA Complexity:** $L_n[1/3, c] = e^{c(\log n)^{1/3}(\log\log n)^{2/3}}$ (Sub-exponential).
- **ECC Complexity:** $O(\sqrt{n}) = O(2^{k/2})$ (Fully Exponential).

This exponential gap implies that a 256-bit ECC key provides security equivalent to a 3072-bit RSA key.

1.5 Project Objectives

This project, titled *Solving the Elliptic Curve Discrete Logarithm Problem (ECDLP) using cryptographic algorithms*, is driven by the following objectives:

1. **Theoretical Foundation**: To conduct a rigorous analysis of the time and space complexity of various DLP algorithms, distinguishing between generic methods (like Baby-step Giant-step) and specific attacks.

2. **Empirical Comparison**: To implement and benchmark these algorithms on "toy" elliptic curves (curves with small parameters). This allows for the practical observation of algorithmic behavior and performance scaling without the prohibitive computational costs of real-world parameters.

3. **Security Quantification**: To demonstrate the fragility of ECC when implementation flaws occur, specifically by simulating attacks that exploit partial key leakage to recover the full private key.

Chapter 2

Preliminaries and Experimental Methodology

This chapter establishes the theoretical foundations required to analyze Elliptic Curve Discrete Logarithm Problem (ECDLP) algorithms and details the engineering architecture used to evaluate their performance.

2.1 Mathematical Preliminaries

2.1.1 The Elliptic Curve Discrete Logarithm Problem (ECDLP)

Let $\mathbb{F}_p$ be a finite field of prime order p. An elliptic curve E over $\mathbb{F}_p$ is defined by the Weierstrass equation:

$$y^2 \equiv x^3 + ax + b \pmod{p} \tag{2.1}$$

where $a, b \in \mathbb{F}_p$ and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. The set of points (x, y) satisfying this equation, together with a point at infinity $\mathcal{O}$, forms an abelian group under the geometric group law.

Definition 2.1.1 (ECDLP)**.** *Given an elliptic curve $E(\mathbb{F}_p)$, a base point $P \in E$ of order n, and a point $Q \in \langle P \rangle$, the Elliptic Curve Discrete Logarithm Problem is to find the integer k (where $0 \leq k < n$) such that:*

$$Q = k \cdot P \tag{2.2}$$

Here, $k \cdot P$ denotes scalar multiplication (adding P to itself k times).

2.1.2 The Birthday Paradox and Collision Search

The time complexity of generic collision-search algorithms, such as Pollard's Rho and the Baby-Step Giant-Step algorithm, is governed by the probabilistic bounds of the **Birthday Paradox**.

Definition 2.1.2 (Birthday Problem)**.** *Consider a set S containing N distinct elements. If we draw k elements from S uniformly at random with replacement, we wish to determine the probability $P(N, k)$ that at least two chosen elements are identical (a collision).*

The probability that all k items are distinct is given by:

$$\bar{P}(N, k) = 1 \cdot \left(1 - \frac{1}{N}\right) \cdot \left(1 - \frac{2}{N}\right) \cdots \left(1 - \frac{k-1}{N}\right) \approx e^{-\frac{k(k-1)}{2N}} \tag{2.3}$$

Therefore, the probability of finding at least one collision is:

$$P(N, k) \approx 1 - e^{-\frac{k^2}{2N}} \tag{2.4}$$

Theorem 2.1 (Expected Steps to Collision)**.** *To achieve a collision probability of* $P(N,k) \approx 0.5$*, the required number of draws* k *is:*

$$k \approx \sqrt{2N \ln 2} \approx 1.177\sqrt{N} \tag{2.5}$$

Furthermore, the expected number of steps to the first collision is $\mathbf{E}[steps] \approx \sqrt{\pi N/2}$*.*

Implication for ECDLP: Since the group order is n, generic algorithms that rely on finding a collision (where $aP + bQ = cP + dQ$) operate in $O(\sqrt{n})$ time complexity, offering a quadratic speedup over brute force search ($O(n)$).

2.2 Implementation Architecture

To balance the ease of algorithmic prototyping with the raw performance required for cryptographic operations, a **hybrid architecture** was engineered. This system decouples the high-level control logic from the computationally intensive arithmetic.

2.2.1 Core Engine (C++ Backend)

The critical bottleneck in all ECDLP algorithms is elliptic curve scalar multiplication ($k \cdot P$) and point addition. These primitives were implemented in **C++17** to leverage low-level memory management and compiler optimizations.

- **Modular Arithmetic:** The backend utilizes `uint64_t` for standard operations and `__int128_t` for intermediate products during modular multiplication to prevent overflow without the overhead of BigInt libraries for curves up to 60 bits.
- **Scalar Multiplication:** The "Double-and-Add" algorithm is employed with a time complexity of $O(\log k)$. For further optimization, the implementation avoids expensive modular inversion ($a^{-1} \bmod p$) during intermediate steps by utilizing projective coordinates.

  Projective Coordinates (X, Y, Z) where $x = X/Z$ and $y = Y/Z$.
- **Foreign Function Interface (FFI):** The C++ code is compiled into a shared library ('.so' on Linux, '.dll' on Windows) and linked to Python via the 'ctypes' library. This reduces the overhead of a single point addition from $\approx 15\mu s$ (pure Python) to $< 0.5\mu s$ (C++ via FFI).

2.2.2 Control Layer (Python Frontend)

Python 3.10 serves as the orchestration layer, managing the state of the algorithms. Its responsibilities include:

- **State Management:** Handling hash tables (Python 'dict') for Baby-Step Giant-Step and cycle detection storage for Pollard's Rho.
- **Input Parsing:** Reading standardized test case files containing curve parameters (p, a, b, P, Q).
- **Heuristics:** Implementing the "Distinguished Points" method to reduce memory access frequency in Pollard's Rho.

2.2.3 Experimental Environment

All performance benchmarks were conducted on a standardized hardware configuration to ensure reproducibility.

Component	Specification
Processor	M1 Pro (8 Core)
Memory	16 GB DDR4
Operating System	Mac Tahoe 26.1
Compiler	GCC 11.3.0 with flags `-O3 -march=native`
Python Version	3.10.12

Table 2.1: Hardware and Software Specifications

2.3 Test Case Generation Methodology

2.3.1 Overview

To evaluate and compare the performance of various ECDLP attack algorithms (*e.g.*, Pollard's Rho, Pollard's Kangaroo, Pohlig–Hellman), we require elliptic curves whose group structure is *known* and *fully controllable*. Real-world standard curves (such as NIST P-256 or Curve25519) are too large to allow repeated experiments, debugging, or controlled isolation of parameters. Therefore, for experimentation we generate a family of synthetic curves that are mathematically valid, small enough to experiment on, and whose order is precisely known.

2.3.2 Choice of Curve Family

We restrict ourselves to supersingular elliptic curves of the form

$$y^2 \equiv x^3 + b \pmod{p},$$

with the parameter choice $a = 0$. We further impose the condition:

$$p \equiv 11 \pmod{12}.$$

This choice is convenient because if $p = 11 \bmod 12$, then $p + 1$ is always divisible by 12. The order of such a supersingular curve satisfies:

$$\#E(\mathbb{F}_p) = p + 1 = 12q.$$

Hence, if we choose a prime p such that $q = (p+1)/12$ is also prime, the curve has cofactor 12 and contains a large prime-order subgroup of order q. This allows us to work in a setting that closely resembles real cryptographic curves, while still keeping parameters small and computationally manageable.

2.3.3 Prime Pair Generation

For a given bit size k, we perform a randomized search for primes p satisfying:

$$\begin{aligned} &p \text{ is a } k\text{-bit prime,} \\ &p \equiv 11 \pmod{12}, \\ &q = \frac{p+1}{12} \text{ is prime.} \end{aligned}$$

We use the Miller–Rabin test (20 rounds) for primality checking. The script searches for a valid (p, q) pair by incrementing in steps of 12, which preserves the congruence class 11 mod 12. Only when (p, q) both test prime do we accept the pair.

2.3.4 Subgroup Generator Extraction

Since the full curve has order $12q$, we eliminate the small cofactor by computing:

$$G = 12P,$$

where P is a uniformly chosen random point on E. If $G \neq \mathcal{O}$, then G lies in the subgroup of prime order q and serves as a generator. A secret scalar $d \in [1, q-1]$ is chosen uniformly, and the public point

$$Q = dG$$

is produced via double-and-add scalar multiplication.

Each test case therefore consists of:

$$(p,\ a = 0,\ b,\ G,\ q,\ Q,\ \text{secret } d).$$

2.3.5 Why We Do Not Use Large Real Cryptographic Curves

The purpose of this work is to study *algorithmic behavior*, not to break real-world systems. True cryptographic curves such as 256-bit and 384-bit NIST curves have group sizes on the order of 2^{256}, making Pollard Rho and related algorithms totally infeasible for experimentation.

For example:

- Pollard Rho has expected runtime $O(\sqrt{q})$. For a 256-bit prime, this is roughly 2^{128} steps, which is computationally impossible to test.
- Repeated experiments, debugging, and comparing different algorithmic optimizations require thousands of runs; this is only feasible on curves with sizes up to 40–60 bits.
- Controlled prime-order subgroup structure cannot be guaranteed when using random large curves unless deep number-theoretic properties are verified.

Thus, the synthetic curves allow:

1. repeatable controlled experiments,
2. accurate validation of correctness,
3. meaningful runtime comparisons,
4. easy debugging when algorithms fail.

Once algorithms behave correctly on small, well-understood primes, the results can be extrapolated theoretically to real-world curve sizes.

2.3.6 Summary

The fixed-prime-subgroup test generator constructs supersingular curves with order $12q$, isolates the prime subgroup of order q, and generates controlled ECDLP instances $(G, Q = dG)$. This provides a reliable and mathematically sound framework for evaluating ECDLP algorithms without the impracticality of real cryptographic curve sizes.

Chapter 3

Algorithm Analysis and Implementation

This chapter presents each algorithm in a structured manner, beginning with its mathematical foundations and a clear theoretical derivation. We then provide a formal analysis of its time and space complexity, followed by practical implementation details that explain how the algorithm was realized in code. Finally, each section includes a **Bonus Adaptation** demonstrating how the algorithm can be modified to exploit partial key leakage, illustrating the impact of additional side-channel information on ECDLP solvability.

3.1 Brute Force Search

3.1.1 Description and Correctness

Brute force search (also known as Exhaustive Search) serves as the computational baseline for the ECDLP. It operates by sequentially generating every element in the cyclic subgroup $\langle P \rangle$ until the target point Q is found.

Theorem 3.1 (Correctness of Exhaustive Search)**.** *Let $E(\mathbb{F}_p)$ be an elliptic curve and P be a point of order n. If $Q \in \langle P \rangle$, the brute force algorithm is guaranteed to terminate and return the unique integer $k \in [0, n-1]$ such that $Q = kP$.*

Proof. **Existence:** Since Q is defined to be in the subgroup generated by P, by definition, there exists at least one integer k satisfying $Q = kP$.
Uniqueness: The subgroup $\langle P \rangle$ is isomorphic to the additive group $\mathbb{Z}_n$. The discrete logarithm map is a bijection between group elements and integers modulo n. Thus, within the domain $[0, n-1]$, the solution k is unique.
Termination: The algorithm iterates through the set of integers $I = \{1, 2, \ldots, n-1\}$. Since the set is finite and the algorithm tests every element $i \in I$, it must eventually test $i = k$. At this step, the condition $i \cdot P = Q$ evaluates to true, and the algorithm returns k. □

Theorem 3.2 (Time and Space Complexity)**.** *Using an incremental accumulation strategy, the Brute Force algorithm has a worst-case time complexity of $O(n)$ group operations and a space complexity of $O(1)$.*

Proof. **Time:** As derived previously, $T(n) \approx n$ operations.
Space Complexity: The algorithm is memoryless. It requires storage only for:

- Constant Curve Parameters: p, a, b ($O(1)$)
- Target Point: Q ($O(1)$)

- Current Accumulator: $R = (x, y)$ $(O(1))$
- Scalar Counter: i $(O(1))$

Total Space $S(n) = O(1)$ field elements. □

Theorem 3.3 (Time and Space Complexity)**.** *Using an incremental accumulation strategy, the Brute Force algorithm has a worst-case time complexity of $O(n)$ group operations and a space complexity of $O(1)$.*

Proof. **Time Complexity:** Let k be the secret scalar.

- **Naive Approach:** If one were to compute $i \cdot P$ from scratch using the Double-and-Add algorithm for every iteration i, the cost would be $O(\log i)$ per step. Summing over n steps yields a total complexity of $O(n \log n)$.
- **Incremental Approach (Optimized):** We utilize the property that $(i + 1)P = iP + P$. By maintaining a running accumulator $R_i = i \cdot P$, the next candidate R_{i+1} is computed using a single point addition: $R_{i+1} \leftarrow R_i + P$.

Since point addition on an elliptic curve takes constant time C (where C depends on the field size $\log p$), the total work is:

$$T(n) = \sum_{i=1}^{k} \text{Cost}(P + R) = k \cdot O(1) \tag{3.1}$$

In the worst case, $k \approx n$, so $T(n) = O(n)$. The average case is $\mathbf{E}[T] = n/2$, which is still $O(n)$.

Space Complexity: The algorithm only requires storage for fixed parameters (P, Q, n) and two dynamic variables: the loop counter i (integer) and the current point R (two field elements x, y). Thus, space complexity is constant, $O(1)$. □

3.1.2 Implementation

The implementation avoids the expensive scalar multiplication inside the loop. We initialize an accumulator R and repeatedly add the generator P.

Algorithm 1 Optimized Brute Force Search

Require: Base point P, Target point Q, Order n
Ensure: Scalar k such that $Q = kP$

```
R ← P                                   ▷ Initialize accumulator at 1 · P
for i ← 1 to n − 1 do
    if R == Q then return i             ▷ Solution found
    end if
    R ← R + P                           ▷ Incremental update: 1 point addition
end for
return Failure                          ▷ Should not be reached if Q ∈ ⟨P⟩
```

3.1.3 [BONUS]Adaptation: Exploiting Side-Channel Leakage

In real-world cryptanalysis, attackers often obtain partial information about the secret key k via side-channels (e.g., power analysis or timing attacks). We implemented variations to exploit two specific types of leakage.

Scenario A: LSB Leakage (Modulo Known)

Suppose an attacker learns the w least significant bits (LSBs) of k.

- **Leakage:** We know L such that $k \equiv L \pmod{2^w}$.
- **Implication:** The key must be of the form $k = m \cdot 2^w + L$ for some integer m.

Instead of stepping by 1, we can start the search at $L \cdot P$ and step by $2^w \cdot P$.

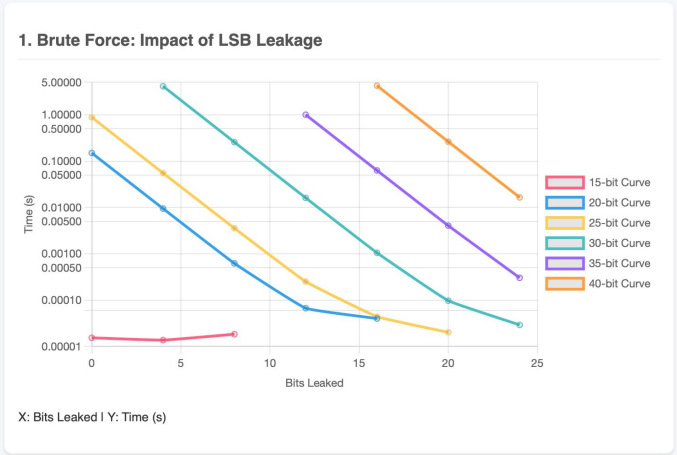

Figure 3.1: LSB Leakage Curve

Scenario B: Interval Leakage (Range Known)

Suppose we know the key lies in a specific range $[A, B]$ where $B - A \ll n$.

- **Leakage:** $A \leq k \leq B$.
- **Implication:** We initialize the accumulator at $A \cdot P$ and iterate only $B - A$ times.

Algorithm 2 Brute Force with LSB Leakage

Require: P, Q, n, Leak L, Bit-width w
Ensure: k

$StepSize \leftarrow 2^w$
$G_{step} \leftarrow StepSize \cdot P$ ▷ Precompute the large stride point
$R \leftarrow L \cdot P$ ▷ Initialize at the known offset
$k \leftarrow L$
while $k < n$ **do**
  if $R == Q$ **then return** k
  end if
  $R \leftarrow R + G_{step}$ ▷ Jump by 2^w steps at once
  $k \leftarrow k + StepSize$
end while

Complexity Impact: If w bits are known, the search space size reduces from n to $n/2^w$.

$$T_{new} \approx \frac{n}{2^w} \cdot \text{Cost}(PointAdd) \tag{3.2}$$

This provides a linear speedup proportional to the reduction in the search space. For example, knowing the bottom 10 bits speeds up the search by a factor of 1024.

3.2 Baby-Step Giant-Step Algorithm

3.2.1 Description and Derivation

The Baby-Step Giant-Step (BSGS) algorithm, attributed to Daniel Shanks, is a time-memory trade-off algorithm that improves upon the brute force approach. While brute force requires $O(n)$ time and $O(1)$ space, BSGS balances these costs to achieve $O(\sqrt{n})$ in both time and space.

The core idea relies on the division algorithm. Let $m = \lceil \sqrt{n} \rceil$. Any integer k in the range $[0, n-1]$ can be uniquely represented as:

$$k = i \cdot m + j \tag{3.3}$$

where $0 \leq i, j < m$. Here, i represents the "giant steps" and j represents the "baby steps".

Substituting this into the discrete logarithm equation $Q = kP$:

$$\begin{aligned} Q &= (i \cdot m + j)P \\ Q &= i \cdot (mP) + jP \\ Q - i \cdot (mP) &= jP \end{aligned}$$

The algorithm proceeds in two phases to find a collision between the left-hand side (Giant Steps) and the right-hand side (Baby Steps):

1. **Baby Steps:** Compute and store the pairs (jP, j) for $0 \leq j < m$ in a hash table (or sorted list) for efficient lookup.

2. **Giant Steps:** Iteratively compute $Q - i(mP)$ for $i = 0, 1, \ldots, m-1$. For each step, check if the resulting point exists in the stored table.

If a match is found, we have $Q - i(mP) = jP$, which implies $k = im + j$.

3.2.2 Correctness

Theorem 3.4 (Correctness of BSGS)**.** *Let $E(\mathbb{F}_p)$ be an elliptic curve with a point P of order n. If $Q \in \langle P \rangle$, the Baby-Step Giant-Step algorithm is guaranteed to return the unique integer $k \in [0, n-1]$ such that $Q = kP$.*

Proof. **Existence of Decomposition:** Let $m = \lceil \sqrt{n} \rceil$. By the Euclidean division algorithm, for any integer k, there exist unique integers i and j such that $k = i \cdot m + j$, where $0 \leq j < m$. Since $0 \leq k < n \leq m^2$, it follows that:

$$i = \lfloor k/m \rfloor \implies 0 \leq i < m.$$

Thus, the solution k is covered by the search space defined by $0 \leq i, j < m$.

Collision Guarantee: The algorithm computes the set of points $S_{baby} = \{jP \mid 0 \leq j < m\}$ and checks elements of the form $S_{giant} = \{Q - i(mP) \mid 0 \leq i < m\}$. Since $Q = (im + j)P$, we have $Q - i(mP) = jP$. Because the algorithm iterates through all possible values of j (in the table construction) and all possible values of i (in the search loop), it must eventually evaluate the specific i and j corresponding to k. At that instance, the lookup will succeed, and the algorithm will return $im + j$. □

3.2.3 Complexity Analysis

Theorem 3.5 (Time and Space Complexity)**.** *The Baby-Step Giant-Step algorithm has a time complexity of $O(\sqrt{n})$ group operations and a space complexity of $O(\sqrt{n})$ points.*

Proof. Let $m = \lceil \sqrt{n} \rceil \approx \sqrt{n}$.

Space Complexity: The algorithm requires storing the "Baby Steps" in a lookup table. There are m such pairs (jP, j). Assuming a hash map is used, the storage requirement is linear in m.

$$S(n) = O(m) = O(\sqrt{n})$$

Time Complexity:

- **Baby Step Phase:** We perform m point additions to generate the table. Inserting into a hash map takes $O(1)$ on average. Total: $O(m)$.
- **Giant Step Phase:** We compute the "Giant Stride" $G = mP$ once. Then, we perform at most m iterations. Each iteration involves one point subtraction (addition of inverse) and one hash table lookup ($O(1)$). Total: $O(m)$.

The total time complexity is the sum of both phases:

$$T(n) = O(m) + O(m) = O(2\sqrt{n}) = O(\sqrt{n})$$

□

3.2.4 Implementation

The implementation utilizes a hash map (dictionary) to store the baby steps for $O(1)$ lookup time.

Algorithm 3 Standard Baby-Step Giant-Step

Require: Base point P, Target point Q, Order n
Ensure: Scalar k such that $Q = kP$

$m \leftarrow \lceil \sqrt{n} \rceil$
$Table \leftarrow$ **new** HashMap
$R \leftarrow \mathcal{O}$ ▷ Identity element $(0 \cdot P)$
▷ **Phase 1: Baby Steps**
for $j \leftarrow 0$ to $m - 1$ **do**
    $Table[R] \leftarrow j$ ▷ Store point as key, index as value
    $R \leftarrow R + P$
end for
▷ **Phase 2: Giant Steps**
$G \leftarrow m \cdot P$ ▷ Compute Giant Stride
$Current \leftarrow Q$
for $i \leftarrow 0$ to $m - 1$ **do**
    if $Current$ in $Table$ **then**
        $j \leftarrow Table[Current]$ **return** $i \cdot m + j$
    end if
    $Current \leftarrow Current - G$
end for
return Failure

3.2.5 BSGS with LSB Leakage (Known Low Bits)

In many side-channel scenarios, an attacker may learn the least significant b bits of the secret scalar k. We show how this information reduces the search space by a factor of 2^b and how to adapt BSGS accordingly.

Algebraic Reduction with Known LSBs

Suppose the secret scalar k satisfies

$$Q = kP, \quad 0 \le k < n,$$

and we know the b least significant bits of k. Let

$$t = 2^b, \qquad \ell = k \bmod t.$$

By definition of remainder, there exists an integer x such that

$$k = xt + \ell, \qquad 0 \le \ell < t. \tag{3.4}$$

Lemma 3.6 (Uniqueness of Representation with Known LSBs)**.** *Let $n \in \mathbb{N}$, $t = 2^b$ for $b \ge 1$, and let $0 \le k < n$ be fixed. Let $\ell = k \bmod t$ and define*

$$x = \left\lfloor \frac{k-\ell}{t} \right\rfloor.$$

Then:

1. $k = xt + \ell$;
2. $0 \le x \le \left\lfloor \frac{n-1}{t} \right\rfloor$;
3. *if $k = x't + \ell$ with an integer x' and $0 \le x' \le \left\lfloor \frac{n-1}{t} \right\rfloor$, then $x' = x$.*

Proof. (1) By the definition of remainder, $k = qt + r$ with $0 \le r < t$ and $r = k \bmod t$. Here $\ell = r$ and $q = x$. Thus $k = xt + \ell$.

(2) Since $k < n$ and $k = xt + \ell \ge xt$, we have

$$xt \le k < n \implies x \le \frac{n-1}{t}.$$

Thus $0 \le x \le \left\lfloor \frac{n-1}{t} \right\rfloor$.

(3) Suppose also $k = x't + \ell$ with $0 \le x' \le \left\lfloor \frac{n-1}{t} \right\rfloor$. Then

$$xt + \ell = k = x't + \ell \implies (x - x')t = 0.$$

Since $t = 2^b > 0$, this implies $x = x'$. □

We now express the ECDLP in terms of x and ℓ.

Lemma 3.7 (Reduction to Smaller DLP)**.** *Let $k = xt + \ell$ with $t = 2^b$ and ℓ known. Define*

$$Q' = Q - \ell P, \qquad P' = tP.$$

Then

$$Q' = xP'. \tag{3.5}$$

Conversely, any x satisfying (3.5) reconstructs k via

$$k = xt + \ell.$$

Proof. Starting from $Q = kP = (xt + \ell)P$, we compute

$$Q = (xt + \ell)P = xtP + \ell P.$$

Subtracting ℓP from both sides gives

$$Q - \ell P = xtP.$$

By definition, $Q' = Q - \ell P$ and $P' = tP$, so

$$Q' = xP',$$

which proves (3.5). Conversely, if $Q' = xP'$ holds, then

$$Q = Q' + \ell P = xP' + \ell P = x(tP) + \ell P = (xt + \ell)P,$$

so $k = xt + \ell$ satisfies $Q = kP$. □

Combining Lemma 3.6 and Lemma 3.7, we see that with knowledge of $\ell = k \bmod 2^b$, the unknown discrete log is reduced to finding x in the smaller range

$$0 \leq x \leq \left\lfloor \frac{n-1}{2^b} \right\rfloor,$$

for which standard BSGS applies.

Algorithm: BSGS with LSB Leakage

We now state the BSGS variant that uses the leaked LSBs. It corresponds directly to the implementation in `solve_lsb`.

Algorithm 4 BSGS with LSB Leakage (Known b LSBs)

Require: Base point P, target Q, group order n, leaked LSB value ℓ, number of leaked bits b

Ensure: Scalar k such that $Q = kP$, assuming $k \bmod 2^b = \ell$

$t \leftarrow 2^b$
$Q' \leftarrow Q - \ell P$ ▷ Shift out the known part
$P' \leftarrow tP$ ▷ Compressed base with step size 2^b
$x_{\max} \leftarrow \lfloor n/t \rfloor$
$m \leftarrow \lceil \sqrt{x_{\max}} \rceil$
$Table \leftarrow$ new hash map
$Baby \leftarrow \mathcal{O}$
for $j = 0$ **to** $m - 1$ **do**
    $Table[\text{serialize}(Baby)] \leftarrow j$
    $Baby \leftarrow Baby + P'$
end for
$GiantStride \leftarrow mP'$
$Curr \leftarrow Q'$
for $i = 0$ **to** m **do**
    if serialize($Curr$) exists in Table **then**
        $j \leftarrow Table[\text{serialize}(Curr)]$
        $x \leftarrow i \cdot m + j$
        if $x \leq x_{\max}$ **then**
            return $k = xt + \ell$
        end if
    end if
    $Curr \leftarrow Curr - GiantStride$
end for
return Failure

Correctness and Complexity

Theorem 3.8 (Correctness of LSB-Optimized BSGS)**.** *Assume there exists k with $0 \leq k < n$ such that $Q = kP$ and $k \bmod 2^b = \ell$. Let $t = 2^b$ and $x_{\max} = \left\lfloor \frac{n-1}{t} \right\rfloor$. Then Algorithm 4 returns this k.*

Proof. By Lemma 3.6, there is a unique integer x with

$$0 \leq x \leq x_{\max}, \qquad k = xt + \ell.$$

By Lemma 3.7, defining $Q' = Q - \ell P$ and $P' = tP$ yields

$$Q' = xP'.$$

The algorithm runs standard BSGS on this reduced equation with range $[0, x_{\max}]$:

- the baby-step phase stores jP' for $0 \leq j < m$;
- the giant-step phase iterates over i and computes $Q' - i(mP')$.

Exactly as in Theorem **??**, when the correct i, j pair satisfies $x = im + j$, the value $Q' - i(mP')$ equals jP' and the algorithm finds the match. Since $x \leq x_{\max}$ and the loops cover all i, j with $0 \leq i \leq m$ and $0 \leq j < m$, there must be an iteration where $x = im + j$, and the corresponding $k = xt + \ell$ is returned.

Uniqueness follows from the uniqueness of x in Lemma 3.6, so the returned k is exactly the discrete logarithm. □

Theorem 3.9 (Time and Space Complexity with LSB Leakage)**.** *Let* $t = 2^b$, $x_{\max} = \left\lfloor \frac{n-1}{t} \right\rfloor$, *and* $m = \lceil \sqrt{x_{\max}} \rceil$. *Then:*

1. *The time complexity of Algorithm 4 is* $O\left(\sqrt{n/2^b}\right)$ *group operations.*
2. *The space complexity is* $O\left(\sqrt{n/2^b}\right)$ *stored points.*

Proof. The algorithm behaves like standard BSGS but on a search space of size approximately $x_{\max} \approx n/t = n/2^b$:

Space. The baby-step table stores one point for each j with $0 \leq j < m$. Thus the number of stored points is

$$S = m = \lceil \sqrt{x_{\max}} \rceil .$$

Since $x_{\max} \leq n/t$ and $t = 2^b$,

$$S = O\left(\sqrt{x_{\max}}\right) = O\left(\sqrt{\frac{n}{2^b}}\right).$$

Time. The baby-step phase performs m additions of P', and the giant-step phase performs at most $m + 1$ additions of $-mP'$, plus hash lookups. Hence the total number of group operations is proportional to $2m + O(1) = O(m)$, i.e.

$$T = O\left(\sqrt{x_{\max}}\right) = O\left(\sqrt{\frac{n}{2^b}}\right).$$

Therefore both time and space complexities are reduced by a factor of approximately $2^{b/2}$ compared to the standard $O(\sqrt{n})$ bounds. □

Impact. If $n \approx 2^\lambda$ and b LSBs of k are leaked, the effective security level drops from roughly $\lambda/2$ bits to $(\lambda - b)/2$ bits, since the complexity goes from $2^{\lambda/2}$ to $2^{(\lambda-b)/2}$. For example, on a 60-bit subgroup, leaking $b = 30$ LSBs reduces the BSGS table size from 2^{30} to 2^{15} entries, which is exactly what our implementation exploits.

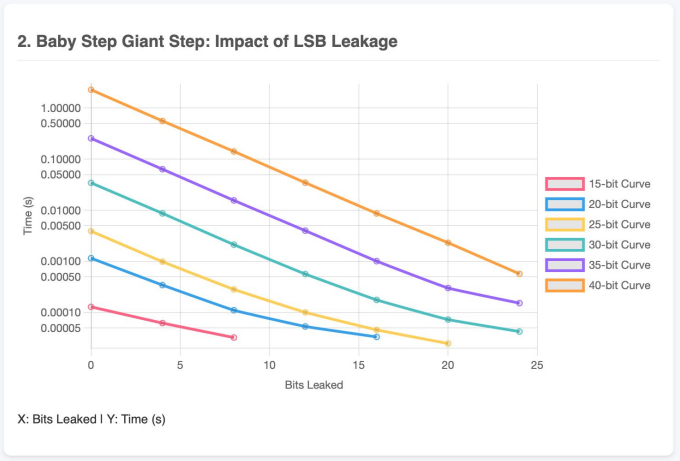

Figure 3.2: LSB Leakage Curve

3.3 Pollard's Rho Algorithm

3.3.1 Description and Theoretical Basis

Pollard's Rho is a probabilistic algorithm for solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Let $G = \langle P \rangle$ be a cyclic group of prime order n, and suppose we are given P and $Q = kP$ for some unknown $k \in \{0, \ldots, n-1\}$. The goal is to recover k.

Like Baby-Step Giant-Step, Pollard's Rho achieves an expected running time proportional to $\sqrt{n}$, but, unlike BSGS, it uses only $O(1)$ memory (up to constant-size bookkeeping). This is possible because the algorithm performs a pseudorandom walk in the group and uses a cycle-finding technique instead of storing all visited points.

3.3.2 Linear Representation and Collision Equation

The algorithm maintains a sequence of group elements (X_i) of the form

$$X_i = a_i P + b_i Q, \tag{3.6}$$

where the coefficients $a_i, b_i \in \mathbb{Z}_n$ are known to the algorithm and updated as the walk progresses, while the discrete logarithm k (with $Q = kP$) is unknown.

The key idea is that if a collision

$$X_i = X_j$$

occurs with distinct coefficient pairs $(a_i, b_i) \neq (a_j, b_j)$, then this yields a linear congruence in k that can be solved.

We first recall and prove a basic fact about linear congruences.

Lemma 3.10 (Solutions to Linear Congruences)**.** *Let $n \in \mathbb{N}$, and let $a, b \in \mathbb{Z}$. Define $d = \gcd(a, n)$.*

1. *If $d \nmid b$, then the congruence*
$$ax \equiv b \pmod{n}$$
has no solutions.

2. *If $d \mid b$, then it has exactly d distinct solutions modulo n.*

Proof. Write $a = da'$ and $n = dn'$ with $\gcd(a', n') = 1$.

(1) Suppose that $ax \equiv b \pmod{n}$ has a solution. Then there exists an integer y such that

$$ax - b = yn.$$

Rearranging gives

$$b = ax - yn = da'x - dn'y = d(a'x - n'y),$$

so $d \mid b$. Thus, if $d \nmid b$, no solution can exist.

(2) Assume $d \mid b$, so $b = db'$ for some integer b'. The congruence

$$ax \equiv b \pmod{n}$$

is equivalent to

$$da'x \equiv db' \pmod{dn'}.$$

Dividing both sides by d yields

$$a'x \equiv b' \pmod{n'}.$$

Since $\gcd(a', n') = 1$, there exists a unique inverse $(a')^{-1} \pmod{n'}$, and hence a unique solution modulo n':

$$x \equiv (a')^{-1} b' \pmod{n'}.$$

Thus all solutions modulo n' are of the form

$$x \equiv x_0 \pmod{n'},$$

for some fixed x_0. When considered modulo $n = dn'$, these correspond to the d distinct residue classes

$$x_0,\ x_0 + n',\ x_0 + 2n',\ \ldots,\ x_0 + (d-1)n' \pmod{n}.$$

These are all distinct modulo n because their pairwise differences are multiples of n' but less than $dn' = n$, and thus cannot be multiples of n unless they are zero. Hence there are exactly d solutions modulo n. □

We now apply this lemma to the collision condition of Pollard's Rho.

Theorem 3.11 (Collision Resolution)**.** *Suppose that we have two indices $i \neq j$ such that*

$$X_i = X_j$$

and

$$X_i = a_i P + b_i Q, \quad X_j = a_j P + b_j Q$$

with $(a_i, b_i) \neq (a_j, b_j)$. Then, letting $n = \operatorname{ord}(P)$, the discrete logarithm k satisfying $Q = kP$ can be recovered by solving a linear congruence in $\mathbb{Z}_n$. More precisely, if we define

$$\Delta a = a_j - a_i, \quad \Delta b = b_i - b_j \pmod{n},$$

then any $k \in \mathbb{Z}_n$ satisfying

$$k \cdot \Delta b \equiv \Delta a \pmod{n} \tag{3.7}$$

gives a correct candidate for the discrete logarithm. When $\gcd(\Delta b, n) = 1$, this solution is unique modulo n.

Proof. From $X_i = X_j$, we have

$$a_i P + b_i Q = a_j P + b_j Q,$$
$$(a_i - a_j) P = (b_j - b_i) Q.$$

Since $Q = kP$,

$$(a_i - a_j) P = (b_j - b_i) kP.$$

Equating the coefficients of P in the cyclic group of order n (i.e. working modulo n), we obtain

$$a_i - a_j \equiv (b_j - b_i) k \pmod{n}.$$

Rewriting yields

$$k(b_j - b_i) \equiv (a_i - a_j) \pmod{n}.$$

Substituting $\Delta a = a_j - a_i$ and $\Delta b = b_i - b_j$ (and noting that replacing both sides by their negatives does not affect the solution set) gives

$$k \cdot \Delta b \equiv \Delta a \pmod{n},$$

which is the claimed congruence (3.7).

By Lemma 3.36, if $d = \gcd(\Delta b, n)$ does not divide Δa, there is no solution and the attempt fails; otherwise there are exactly d solutions modulo n. The case $\gcd(\Delta b, n) = 1$ yields a unique solution

$$k \equiv \Delta a \cdot (\Delta b)^{-1} \pmod{n}.$$

Each candidate can be verified by checking whether $kP = Q$. □

3.3.3 Random Walk and Expected Collision Time

To analyze the running time of Pollard's Rho, we approximate its behavior by a random process on a finite set. We make the standard heuristic assumption that the iteration function used by the algorithm behaves like a random function on the group.

Definition 3.3.1 (Random Sequence on a Finite Set)**.** *Let S be a finite set of size n. A sequence $(Y_t)_{t\geq 1}$ is called* random and independent over S *if each Y_t is chosen independently and uniformly from S.*

Our walk (X_i) is not exactly independent and identically distributed, because each point is computed from the previous by a fixed rule, but the complexity analysis assumes:

> *Heuristic:* The distribution of the sequence (X_i) is close to that of a random sequence over the group G.

Under this heuristic, the expected time to first collision in (X_i) is approximated by the expected time to the first repeated value in a random sequence over a set of size n. We now analyze the latter.

Lemma 3.12 (Expected First Collision Time in a Random Sequence)**.** *Let S be a set of size n and let $Y_1, Y_2, \ldots$ be chosen independently and uniformly from S. Let T be the (random) index of the first collision, i.e.*

$$T = \min\{t \geq 2 : \exists\, i < t \text{ with } Y_i = Y_t\}.$$

Then the expected value $\mathbb{E}[T]$ satisfies

$$\mathbb{E}[T] = \Theta(\sqrt{n}).$$

More precisely, for large n,

$$\mathbb{E}[T] \approx \sqrt{\frac{\pi n}{2}}.$$

Proof. For each integer $t \geq 1$, let A_t be the event that the first t values are all distinct:

$$A_t = \{Y_1, \ldots, Y_t \text{ are pairwise distinct}\}.$$

We first compute $\Pr(A_t)$. The number of ways to choose t distinct values in order is

$$n \cdot (n-1) \cdot (n-2) \cdots (n-t+1),$$

and the total number of length-t sequences is n^t. Therefore

$$\Pr(A_t) = \frac{n(n-1)\cdots(n-t+1)}{n^t} = \prod_{i=0}^{t-1} \left(1 - \frac{i}{n}\right).$$

The random variable T satisfies the identity

$$\Pr(T > t) = \Pr(A_t),$$

since $T > t$ iff there is no collision among the first t values. Using the standard identity

$$\mathbb{E}[T] = \sum_{t=1}^{\infty} \Pr(T \geq t),$$

and noting $\Pr(T \geq t) = \Pr(A_{t-1})$ for $t \geq 2$, we have

$$\mathbb{E}[T] = 1 + \sum_{t=2}^{\infty} \Pr(A_{t-1}) = \sum_{t=0}^{\infty} \Pr(A_t).$$

We now approximate $\Pr(A_t)$ for $t \ll n$ by bounding the product. Taking logarithms:

$$\log \Pr(A_t) = \sum_{i=0}^{t-1} \log\left(1 - \frac{i}{n}\right).$$

For $0 \leq x < 1$, we have the inequality

$$-x - x^2 \leq \log(1-x) \leq -x,$$

which can be proved from the Taylor expansion of $\log(1-x)$ and bounding the remainder. Applying this with $x = i/n$ gives

$$-\frac{i}{n} - \left(\frac{i}{n}\right)^2 \leq \log\left(1 - \frac{i}{n}\right) \leq -\frac{i}{n}.$$

Summing over i from 0 to $t-1$, we obtain

$$-\sum_{i=0}^{t-1}\left(\frac{i}{n} + \frac{i^2}{n^2}\right) \leq \log \Pr(A_t) \leq -\sum_{i=0}^{t-1} \frac{i}{n}.$$

Compute the sums:

$$\sum_{i=0}^{t-1} i = \frac{t(t-1)}{2}, \qquad \sum_{i=0}^{t-1} i^2 = \frac{(t-1)t(2t-1)}{6}.$$

Hence

$$-\frac{t(t-1)}{2n} - \frac{(t-1)t(2t-1)}{6n^2} \leq \log \Pr(A_t) \leq -\frac{t(t-1)}{2n}.$$

For $t \leq n^{2/3}$, the term involving $1/n^2$ is much smaller than the term involving $1/n$ (specifically, of order t^3/n^2, which is $o(1)$ when $t = o(n^{2/3})$). Thus, for such t,

$$\log \Pr(A_t) = -\frac{t(t-1)}{2n} + O\left(\frac{t^3}{n^2}\right).$$

Exponentiating gives the approximation

$$\Pr(A_t) \approx \exp\left(-\frac{t(t-1)}{2n}\right).$$

Now, consider $t = c\sqrt{n}$ for a constant $c > 0$. Then

$$\frac{t(t-1)}{2n} \approx \frac{c^2 n}{2n} = \frac{c^2}{2}.$$

Thus, for t of order $\sqrt{n}$,

$$\Pr(A_t) \approx e^{-c^2/2},$$

which is bounded away from both 0 and 1. This shows that the time to first collision is on the order of $\sqrt{n}$, i.e. $\mathbb{E}[T] = \Theta(\sqrt{n})$.

With a more careful analysis (integral approximation of the sum), one can show that

$$\mathbb{E}[T] \to \sqrt{\frac{\pi n}{2}}$$

as $n \to \infty$. We omit the detailed integral calculation here and retain the main conclusion: the expected first collision time is proportional to $\sqrt{n}$. □

3.3.4 Cycle Structure and Floyd's Cycle-Finding

The Pollard Rho walk is generated by repeatedly applying a function

$$f : G \to G,$$

so that $X_{i+1} = f(X_i)$. Any function from a finite set to itself has an eventual cycle structure.

Lemma 3.13 (Cycle Decomposition of Iterated Functions)**.** *Let S be a finite set, and let $f : S \to S$ be any function. For any starting point $x_0 \in S$, the sequence defined by $x_{i+1} = f(x_i)$ has the form*

$$x_0, x_1, \ldots, x_{\mu-1}, x_\mu, x_{\mu+1}, \ldots,$$

where:

1. *All $x_0, \ldots, x_{\mu-1}$ are distinct (the "tail").*
2. *The subsequence $x_\mu, x_{\mu+1}, \ldots$ is periodic with some period $\lambda \geq 1$, i.e.*

$$x_{\mu+t} = x_{\mu+t+\lambda} \quad \text{for all } t \geq 0.$$

Proof. Since S has finite size $|S| = N$, the sequence (x_i) takes values in a finite set. By the pigeonhole principle, some value must repeat: there exist indices $0 \leq i < j$ with $x_i = x_j$. Let μ be the smallest index such that there exists $j > \mu$ with $x_j = x_\mu$, and let λ be the smallest positive integer with $x_{\mu+\lambda} = x_\mu$.

By the minimality of μ, all $x_0, \ldots, x_{\mu-1}$ are distinct. By definition of λ, the sequence from x_μ onward is periodic with period λ, since

$$x_{\mu+t+\lambda} = f^{t+\lambda}(x_\mu) = f^t(f^\lambda(x_\mu)) = f^t(x_\mu) = x_{\mu+t}$$

for all $t \geq 0$, where f^t denotes t-fold composition of f. □

Floyd's cycle-finding algorithm ("tortoise and hare") uses this structure to detect a repeated value using constant memory.

Lemma 3.14 (Correctness of Floyd's Cycle-Finding)**.** *Let $f : S \to S$ with S finite, and consider the sequence $x_{i+1} = f(x_i)$ with x_0 given. Define two sequences:*

$$T_0 = x_0, \quad T_{i+1} = f(T_i) \quad \textit{(tortoise)},$$

$$H_0 = x_0, \quad H_{i+1} = f(f(H_i)) \quad \textit{(hare)}.$$

Then there exist indices $i, j \geq 0$ with $T_i = H_j$. In particular, the algorithm that advances T by one step and H by two steps, and stops when $T = H$, is guaranteed to terminate with some collision, while storing only $O(1)$ values.

Proof. By Lemma 3.13, the sequence (x_i) has a tail of length μ followed by a cycle of length λ. Once the hare has advanced more than μ steps, it is in the cycle. Similarly, once the tortoise has also advanced more than μ steps, it too is in the cycle.

Within the cycle, the positions of tortoise and hare can be represented modulo λ. At each iteration, the tortoise moves one step, and the hare moves two. Thus, if their positions in the cycle are t and h (modulo λ), at the next iteration they become

$$t' = t + 1 \pmod{\lambda}, \quad h' = h + 2 \pmod{\lambda}.$$

The difference $d = h - t \pmod{\lambda}$ changes as

$$d' = h' - t' \equiv (h+2) - (t+1) \equiv d + 1 \pmod{\lambda}.$$

Since repeatedly adding 1 modulo λ must eventually yield 0 (after exactly λ steps), it follows that at some iteration we have $d = 0$, i.e. $h \equiv t \pmod{\lambda}$, which means the hare and tortoise occupy the same position in the cycle, hence the same state in S. Therefore, at some iteration, $T = H$.

The algorithm only stores the current values of T, H, and possibly their associated coefficients (finite many integers), so the memory usage is independent of $|S|$, i.e. $O(1)$. □

3.3.5 Complexity Analysis of Pollard's Rho

We now combine the collision analysis and cycle-finding to obtain the complexity.

Theorem 3.15 (Time and Space Complexity of Pollard's Rho)**.** *Let $G = \langle P \rangle$ be a cyclic group of order n. Under the heuristic that the Pollard Rho iteration function behaves like a random function on G, the Pollard Rho algorithm has:*

- *expected time complexity $O(\sqrt{n})$ group operations;*
- *space complexity $O(1)$ group elements and scalars.*

Proof. **Time complexity:** Under the random-function heuristic, the sequence (X_i) behaves like a random sequence over a set of size n. By Lemma 3.12, the expected index T of the first collision is $\Theta(\sqrt{n})$. Each step of the algorithm (i.e. computing X_{i+1} from X_i and updating coefficients) uses a constant number of group operations (depending on the partition rules used), so the expected total number of group operations until a collision is detected is also $\Theta(\sqrt{n})$. In big-O notation this is $O(\sqrt{n})$.

Space complexity: The algorithm implemented with Floyd's cycle-finding (Lemma 3.14) maintains:

- the tortoise state (T, a_T, b_T);
- the hare state (H, a_H, b_H);
- a fixed description of the iteration function (e.g. partition rules).

Each of these consists of a constant number of group elements and integers modulo n. The amount of memory used does not depend on n except through the bit-length of the stored values, so in terms of the number of group elements stored, it is $O(1)$. Therefore space complexity is $O(1)$. □

3.3.6 Implementation: The Random Walk

To construct a pseudorandom walk that preserves the representation $X_i = a_i P + b_i Q$, we define a partition of the group into a small number of disjoint subsets and apply different update rules in each subset. A common choice is to use three sets S_1, S_2, S_3, distinguished by a simple function of the point, such as the x-coordinate modulo 3.

Algorithm 5 Pollard's Rho for ECDLP (Floyd's Cycle Finding)

Require: Base point P, target point Q, group order n
Ensure: Discrete logarithm k such that $Q = kP$ (or **Failure**)

Initialize $T \leftarrow P$, $a_T \leftarrow 1$, $b_T \leftarrow 0$ ▷ Tortoise
Initialize $H \leftarrow P$, $a_H \leftarrow 1$, $b_H \leftarrow 0$ ▷ Hare
function STEP(X, a, b)
    $partition \leftarrow X.x \bmod 3$ ▷ Hash of the point
    if $partition = 0$ **then** ▷ Rule 1: Add Q
        return $(X + Q,\ a,\ (b+1) \bmod n)$
    else if $partition = 1$ **then** ▷ Rule 2: Double
        return $(2X,\ (2a) \bmod n,\ (2b) \bmod n)$
    else ▷ Rule 3: Add P
        return $(X + P,\ (a+1) \bmod n,\ b)$
    end if
end function
repeat
    $(T, a_T, b_T) \leftarrow$ STEP(T, a_T, b_T) ▷ Tortoise moves one step
    $(H, a_H, b_H) \leftarrow$ STEP(H, a_H, b_H)
    $(H, a_H, b_H) \leftarrow$ STEP(H, a_H, b_H) ▷ Hare moves two steps
until $T = H$
$numerator \leftarrow (a_T - a_H) \bmod n$
$denominator \leftarrow (b_H - b_T) \bmod n$
if $\gcd(denominator, n) = 1$ **then**
    $k \leftarrow numerator \cdot denominator^{-1} \bmod n$ ▷ Solve $k \cdot \Delta b \equiv \Delta a$
    if $kP = Q$ **then**
        return k
    else
        return Failure ▷ Spurious solution; restart with new parameters
    end if
else
    return Failure ▷ Restart with different randomization
end if

3.4 [BONUS]Pollard's Kangaroo Algorithm (Lambda Method)

3.4.1 Description and Theoretical Basis

Pollard's Kangaroo algorithm is a variant of Pollard's methods designed specifically for the case where the discrete logarithm k is known to lie within a given interval

$$k \in [A, B], \quad 0 \leq A < B < n.$$

Let $W = B - A$ be the *width* of the interval. The Kangaroo method exploits this additional information to achieve expected running time $O(\sqrt{W})$, which is faster than generic $O(\sqrt{n})$ methods when $W \ll n$.

The algorithm uses two walks ("kangaroos"):

- a *tame* kangaroo starting from a known position (typically BP);
- a *wild* kangaroo starting from the unknown position $Q = kP$.

Both kangaroos use the same deterministic jump function, so once their paths meet, they follow the same trajectory thereafter. By recording certain "distinguished" positions of the tame kangaroo, the algorithm can detect when the wild kangaroo lands on a previously visited point, and from the total distances jumped it can deduce k.

3.4.2 Distance Accounting and Correctness

We formalize the notion of distance in the group.

Definition 3.4.1 (Jump Function and Distance)**.** *Let J be a finite set of positive integers (jump sizes), and let*

$$s : G \to J$$

be a deterministic function (e.g. based on a hash of the x-coordinate). For any walk (X_t) defined by

$$X_{t+1} = X_t + s(X_t) \cdot P,$$

we define the distance traveled *up to time t by*

$$D_t = \sum_{u=0}^{t-1} s(X_u).$$

Then

$$X_t = X_0 + D_t \cdot P.$$

Proof. We prove by induction on t. For $t = 0$, the statement is $X_0 = X_0 + 0 \cdot P$, which is true. Assume the statement holds for some $t \geq 0$:

$$X_t = X_0 + D_t P.$$

Then

$$X_{t+1} = X_t + s(X_t)P = X_0 + D_t P + s(X_t)P = X_0 + (D_t + s(X_t))P.$$

By definition, $D_{t+1} = D_t + s(X_t)$, so

$$X_{t+1} = X_0 + D_{t+1} P.$$

Thus the claim holds for all t. □

We now consider two walks:

- Tame kangaroo: starts at $X_0^{(T)} = BP$ with distances $D_t^{(T)}$.
- Wild kangaroo: starts at $X_0^{(W)} = kP = Q$ with distances $D_t^{(W)}$.

Theorem 3.16 (Correctness of Pollard's Kangaroo)**.** *Suppose that, using the same jump function s, the tame and wild kangaroos meet at some group element Y, i.e. there exist integers $t_T, t_W \geq 0$ with*

$$X_{t_T}^{(T)} = X_{t_W}^{(W)}.$$

Let $D_T = D_{t_T}^{(T)}$ and $D_W = D_{t_W}^{(W)}$ be the corresponding distances. Then the discrete logarithm k satisfies

$$k \equiv B + D_T - D_W \pmod{n}. \tag{3.8}$$

Proof. From the distance representation, we have

$$X_{t_T}^{(T)} = BP + D_T P = (B + D_T)P,$$

and

$$X_{t_W}^{(W)} = kP + D_W P = (k + D_W)P.$$

The assumption $X_{t_T}^{(T)} = X_{t_W}^{(W)}$ yields

$$(B + D_T)P = (k + D_W)P.$$

Since P has order n, equality of multiples implies

$$B + D_T \equiv k + D_W \pmod{n},$$

hence

$$k \equiv B + D_T - D_W \pmod{n}.$$

□

Thus, once a collision is detected and the distances of both kangaroos at that collision point are known, we obtain k by the simple arithmetic formula (3.8). This is the heart of the algorithm.

3.4.3 Algorithm

In practice, it is not feasible to store all tame positions. Instead, we select a subset of points as *distinguished* (e.g. those whose binary encoding has a certain number of trailing zeros) and record only those. This keeps the memory usage manageable while allowing detection of collisions.

Algorithm 6 Pollard's Kangaroo for Interval $[A, B]$

Require: Base point P, target point $Q = kP$, bounds A, B with $k \in [A, B]$
Ensure: $k \in [A, B]$ or **Failure**

$W \leftarrow B - A$
Choose jump sizes $J = \{j_1, \ldots, j_r\} \subset \mathbb{N}$ ▷ e.g. powers of 2 or random integers
Define step function $s : G \to J$ ▷ e.g. $s(X)$ from a hash of (x_X, y_X)
Define predicate IsDistinguished(X) ▷ e.g. X has many trailing zero bits
$T \leftarrow B \cdot P$, $D_T \leftarrow 0$ ▷ Tame kangaroo
$TrapTable \leftarrow \emptyset$
// Phase 1: Tame kangaroo sets traps
while $D_T < L$ **do** ▷ L is a chosen limit, e.g. a constant multiple of $\sqrt{W}$
    $j \leftarrow s(T)$
    $T \leftarrow T + jP$
    $D_T \leftarrow D_T + j$
    if IsDistinguished(T) **then**
        $TrapTable[T] \leftarrow D_T$
    end if
end while
$Wpt \leftarrow Q$, $D_W \leftarrow 0$ ▷ Wild kangaroo
// Phase 2: Wild kangaroo searches for a trap
while $D_W < L + W$ **do**
    $j \leftarrow s(Wpt)$
    $Wpt \leftarrow Wpt + jP$
    $D_W \leftarrow D_W + j$
    if $Wpt \in TrapTable$ **then**
        $D_T^\star \leftarrow TrapTable[Wpt]$
        $k \leftarrow B + D_T^\star - D_W \pmod n$ ▷ By Theorem 3.16
        if $kP = Q$ **and** $A \leq k \leq B$ **then**
            return k
        else
            return Failure
        end if
    end if
end while
return Failure

3.4.4 Complexity Analysis of Pollard's Kangaroo

We now analyze the running time and memory usage of the Kangaroo algorithm as a function of the interval width $W = B - A$.

Theorem 3.17 (Heuristic Time Complexity of Pollard's Kangaroo)**.** *Assume that the step function s causes the kangaroo walks to behave like random walks over the interval of possible exponents of width $W = B - A$. If the jump sizes in J are bounded above by a parameter M with $M \approx \sqrt{W}$, then the expected running time of the Kangaroo algorithm is $O(\sqrt{W})$ group operations.*

Proof. We work with a simple heuristic model. For exponents, we imagine the tame kangaroo starting from exponent B and the wild kangaroo starting from exponent $k \in [A, B]$. Each step of either kangaroo increases its exponent by a jump size chosen from J according to s, which we heuristically model as approximately random and independent with some average jump size $\mathbb{E}[J]$.

Let M be the maximum jump size, and suppose that the average jump size satisfies

$$c_1 M \leq \mathbb{E}[J] \leq c_2 M$$

for some fixed constants $c_1, c_2 > 0$ (for example, if $J = \{1, 2, \ldots, M\}$ uniformly, then $\mathbb{E}[J] = (M+1)/2$).

Tame kangaroo: Starting from B, after t steps the tame kangaroo has advanced by approximately $t \cdot \mathbb{E}[J]$ exponents. We choose the limit L for phase 1 such that the total expected distance D_T is on the order of W, e.g.

$$L \approx \frac{W}{\mathbb{E}[J]}.$$

If we choose $M \approx \sqrt{W}$, then $\mathbb{E}[J]$ is also on the order of $\sqrt{W}$, and thus L is on the order of

$$L \approx \frac{W}{\sqrt{W}} = \sqrt{W}.$$

Thus, the tame kangaroo performs $O(\sqrt{W})$ steps.

Wild kangaroo: The wild kangaroo starts at exponent k. Its distance to the tame kangaroo region is at most W exponents. With the same average jump size $\mathbb{E}[J] \approx \sqrt{W}$, the expected number of steps needed to cover a distance of size at most W is on the order of

$$\frac{W}{\mathbb{E}[J]} \approx \frac{W}{\sqrt{W}} = \sqrt{W}.$$

Therefore, the wild kangaroo also performs $O(\sqrt{W})$ steps before either colliding with a trap or exceeding the search limit.

Adding the contributions of both phases, the total expected number of steps is $O(\sqrt{W})$, and each step involves a constant number of group operations (a few additions and scalar multiples with small integers). Hence, the expected number of group operations is $O(\sqrt{W})$. □

Theorem 3.18 (Space Complexity of Pollard's Kangaroo)**.** *Let $\theta \in (0, 1)$ be the fraction of points designated as distinguished by the predicate* IsDistinguished*. Under the same heuristic model as above, and with an expected running time $O(\sqrt{W})$, the expected number of trap entries stored is $O(\sqrt{W})$. Thus the Kangaroo algorithm uses $O(\sqrt{W})$ space for the trap table, plus $O(1)$ additional working storage.*

Proof. During phase 1, the tame kangaroo performs $O(\sqrt{W})$ steps (Theorem 3.17). Each step produces a new point T in the group. Under the heuristic that these points are approximately uniformly distributed over the relevant subset of G, the probability that a given point is distinguished is approximately θ. Therefore, the number of stored traps is a binomial random variable with parameters (number of trials) $O(\sqrt{W})$ and (success probability) θ. The expected number of traps is thus

$$\mathbb{E}[\#\text{traps}] = \theta \cdot O(\sqrt{W}) = O(\sqrt{W}).$$

The wild kangaroo phase does not store additional points; it only performs lookups in the trap table. Thus the dominant memory usage arises from the trap table, which is $O(\sqrt{W})$ entries. The additional working memory (current points and distances for both kangaroos, plus fixed descriptions of P, Q, s, J) is constant-size, i.e. $O(1)$ group elements and scalars.

Therefore, the overall space complexity is $O(\sqrt{W})$. □

Comparison with Other Algorithms. When no information about k is known beyond $0 \leq k < n$, Pollard's Rho yields $O(\sqrt{n})$ time and $O(1)$ space. When k is known to lie in a subinterval $[A, B]$ of width W, Pollard's Kangaroo specializes this to $O(\sqrt{W})$ time and $O(\sqrt{W})$ space. This makes it particularly effective in scenarios with partial information, such as side-channel leakage that restricts k to a narrower range.

3.4.5 Kangaroo with LSB Leakage (Known Low Bits)

We now adapt Pollard's Kangaroo to the case where the attacker knows the b least significant bits (LSBs) of the secret scalar k. This directly corresponds to our implementation in the `solve_lsb` function, but we present it here in a purely mathematical form.

Algebraic Reduction with Known LSBs

Assume again that $Q = kP$ with $0 \leq k < n$, and that we know the b least significant bits of k. Let

$$t = 2^b, \qquad \ell = k \bmod t.$$

Then there exists an integer x such that

$$k = xt + \ell, \qquad 0 \leq \ell < t. \tag{3.9}$$

Lemma 3.19 (Uniqueness of Representation with Known LSBs)**.** *Let $n \in \mathbb{N}$, $t = 2^b$ for $b \geq 1$, and let $0 \leq k < n$ be fixed. Let $\ell = k \bmod t$ and define*

$$x = \left\lfloor \frac{k - \ell}{t} \right\rfloor.$$

Then:

1. $k = xt + \ell$;
2. $0 \leq x \leq \left\lfloor \dfrac{n-1}{t} \right\rfloor$;
3. *if $k = x't + \ell$ with an integer x' and $0 \leq x' \leq \left\lfloor \dfrac{n-1}{t} \right\rfloor$, then $x' = x$.*

Proof. The proof is identical to Lemma 3.6 for BSGS, but we recall it briefly.

(1) By definition of remainder, $k = qt + r$ with $0 \leq r < t$ and $r = k \bmod t$. Here $\ell = r$ and $q = x$. Thus $k = xt + \ell$.

(2) Since $k < n$ and $k = xt + \ell \geq xt$, we have

$$xt \leq k < n \implies x \leq \frac{n-1}{t}.$$

Thus $0 \leq x \leq \left\lfloor \dfrac{n-1}{t} \right\rfloor$.

(3) Suppose also $k = x't + \ell$ with $0 \leq x' \leq \left\lfloor \dfrac{n-1}{t} \right\rfloor$. Then

$$xt + \ell = k = x't + \ell \implies (x - x')t = 0.$$

Since $t = 2^b > 0$, this implies $x = x'$. □

As in the BSGS LSB case, we reduce ECDLP to a smaller interval problem.

Lemma 3.20 (Reduction to Bounded-Interval DLP)**.** *Let $k = xt + \ell$ with $t = 2^b$ and ℓ known. Define*

$$Q' = Q - \ell P, \qquad P' = tP.$$

Then

$$Q' = xP'. \tag{3.10}$$

Conversely, any x satisfying (3.10) yields the original discrete logarithm via $k = xt + \ell$.

Proof. Starting from $Q = kP = (xt + \ell)P$, we compute

$$Q = (xt + \ell)P = xtP + \ell P.$$

Subtracting ℓP from both sides gives

$$Q - \ell P = xtP.$$

By definition, $Q' = Q - \ell P$ and $P' = tP$, so

$$Q' = xP',$$

which is (3.10). Conversely, if $Q' = xP'$ holds, then

$$Q = Q' + \ell P = xP' + \ell P = x(tP) + \ell P = (xt + \ell)P,$$

so $k = xt + \ell$ satisfies $Q = kP$. □

By Lemma 3.19, x lies in the interval

$$0 \le x \le x_{\max} := \left\lfloor \frac{n-1}{t} \right\rfloor,$$

so the reduced DLP (3.10) is exactly a bounded-interval discrete log problem in the range $[0, x_{\max}]$. This is the setting where Pollard's Kangaroo is optimal.

Algorithm: Kangaroo with LSB Leakage

We now state the LSB-optimized Kangaroo variant, mirroring our code structure:

Algorithm 7 Pollard's Kangaroo with LSB Leakage (Known b LSBs)

Require: Base point P, target $Q = kP$, group order n, leaked LSB value ℓ, number of leaked bits b
Ensure: Scalar k such that $Q = kP$, assuming $k \bmod 2^b = \ell$

1: $t \leftarrow 2^b$
2: $Q' \leftarrow Q - \ell P$ ▷ Shift out the known LSBs
3: $P' \leftarrow tP$ ▷ Compressed base point
4: $x_{\max} \leftarrow \left\lfloor \frac{n}{t} \right\rfloor$
5: **return** KangarooInterval$(P', Q', 0, x_{\max})$, then reconstruct $k = xt + \ell$

where KangarooInterval(P', Q', A', B') is simply Algorithm 6 specialized to the interval $[A', B'] = [0, x_{\max}]$ with base point P' and target Q'.

More explicitly:

Algorithm 8 KangarooInterval for LSB-Reduced Problem

Require: Base P', target $Q' = xP'$, bounds $0 \le x \le x_{\max}$
Ensure: x or **Failure**
1: $A' \leftarrow 0$, $B' \leftarrow x_{\max}$
2: Run Algorithm 6 with base P', target Q', bounds A', B'
3: If Algorithm 6 returns x, then return x, otherwise **Failure**

Combining Algorithms 7 and 8, the overall solution is:

$$k = xt + \ell, \quad x = \text{KangarooInterval}(P', Q', 0, x_{\max}).$$

Correctness and Complexity

Theorem 3.21 (Correctness of LSB-Optimized Kangaroo)**.** *Assume there exists k with $0 \le k < n$ such that $Q = kP$ and $k \bmod 2^b = \ell$. Let $t = 2^b$ and $x_{\max} = \left\lfloor \frac{n-1}{t} \right\rfloor$. Then, under the assumption that Algorithm 6 successfully solves bounded-interval DLP on $[0, x_{\max}]$, Algorithm 7 returns this k.*

Proof. By Lemma 3.19, there is a unique integer x with

$$0 \le x \le x_{\max}, \qquad k = xt + \ell.$$

By Lemma 3.20, defining $Q' = Q - \ell P$ and $P' = tP$ yields

$$Q' = xP'.$$

Thus the reduced problem is a bounded-interval discrete log on $[0, x_{\max}]$.

By correctness of the underlying Kangaroo algorithm (Theorem 3.16, applied to $P', Q', A' = 0, B' = x_{\max}$), KangarooInterval returns exactly this x. Substituting back, Algorithm 7 outputs

$$k = xt + \ell,$$

which, by construction, equals the original discrete logarithm. □

Theorem 3.22 (Time and Space Complexity with LSB Leakage)**.** *Let $t = 2^b$ and $x_{\max} = \left\lfloor \frac{n-1}{t} \right\rfloor$. Under the same heuristic assumptions as in Theorem 3.17, the Kangaroo algorithm with LSB leakage has:*

1. *expected time complexity $O\left(\sqrt{x_{\max}}\right) = O\left(\sqrt{\frac{n}{2^b}}\right)$ group operations;*
2. *space complexity $O\left(\sqrt{x_{\max}}\right) = O\left(\sqrt{\frac{n}{2^b}}\right)$ stored traps.*

Proof. The reduced problem $Q' = xP'$ has solution x in the interval $[0, x_{\max}]$. Thus the interval width is

$$W' = x_{\max} - 0 \approx \frac{n}{t} = \frac{n}{2^b}.$$

Applying Theorem 3.17 with $W = W'$ yields expected time complexity $O(\sqrt{W'}) = O\left(\sqrt{\frac{n}{2^b}}\right)$.

Similarly, Theorem 3.18 with $W = W'$ gives space complexity $O(\sqrt{W'}) = O\left(\sqrt{\frac{n}{2^b}}\right)$ trap entries.

Therefore, both time and space are reduced by approximately a factor of $2^{b/2}$ relative to the standard $O(\sqrt{n})$ Kangaroo without leakage. □

Impact. For a group of size $n \approx 2^{\lambda}$, generic Kangaroo costs about $2^{\lambda/2}$ group operations. If b LSBs of k are leaked, the effective cost becomes approximately

$$2^{\frac{\lambda-b}{2}},$$

corresponding to a loss of $b/2$ bits of security. Our implementation (`solve_lsb` using `kangaroo` on the transformed problem) is a direct realization of this attack in the elliptic-curve setting.

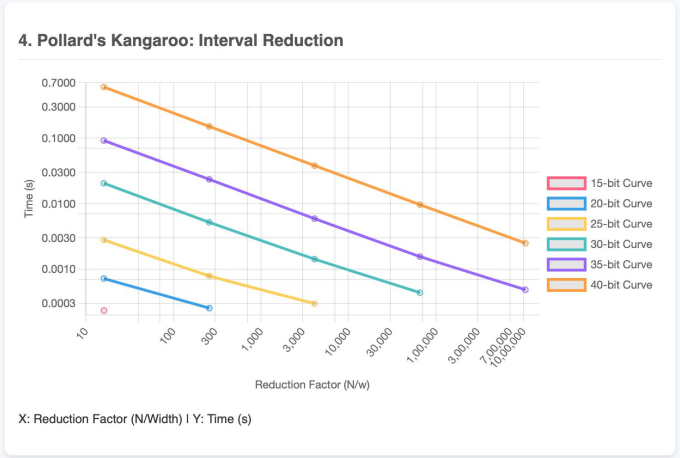

Figure 3.3: LSB Leakage Curve

3.5 Pohlig–Hellman Algorithm

3.5.1 Description and Theoretical Basis

Let $G = \langle P \rangle$ be a cyclic group of order n, and suppose

$$Q = xP$$

for some unknown $x \in \{0, \dots, n-1\}$. The Pohlig–Hellman algorithm is a "divide and conquer" method that exploits the factorization of the group order

$$n = \prod_{i=1}^{r} p_i^{e_i}$$

into prime powers. Instead of solving the discrete logarithm modulo n directly, it reduces the problem to separate discrete logarithms modulo each $p_i^{e_i}$ and then reconstructs x using the Chinese Remainder Theorem (CRT). As a result, the complexity is governed by the largest prime factor of n.

Preliminaries: Order and CRT

We first prove the algebraic tools used by Pohlig–Hellman.

Lemma 3.23 (Order of a Multiple of a Generator)**.** *Let $G = \langle P \rangle$ be a cyclic group of order n, and let $m \in \mathbb{Z}$. Put $d = \gcd(m, n)$. Then the order of mP in G is*

$$\operatorname{ord}(mP) = \frac{n}{d}.$$

In particular, if $n = p^e$ and $m = n/p = p^{e-1}$, then $\operatorname{ord}(mP) = p$.

Proof. By definition, $\operatorname{ord}(mP)$ is the smallest positive integer t such that

$$t(mP) = \mathcal{O}.$$

Because G has order n, we know that $kP = \mathcal{O}$ if and only if $n \mid k$. Thus

$$t(mP) = \mathcal{O} \iff tmP = \mathcal{O} \iff n \mid tm.$$

So $\operatorname{ord}(mP)$ is the least positive integer t such that n divides tm.

Write $m = dm'$ and $n = dn'$ with $\gcd(m', n') = 1$. Then $n \mid tm$ is equivalent to

$$dn' \mid t(dm') \iff n' \mid tm'.$$

Since $\gcd(m', n') = 1$, the smallest positive integer t such that $n' \mid tm'$ is $t = n'$ (any smaller t would make tm' a nonzero multiple of n' while m' is invertible modulo n'). Hence $\operatorname{ord}(mP) = n' = n/d$.

For the special case $n = p^e$ and $m = n/p = p^{e-1}$, we have $d = \gcd(p^{e-1}, p^e) = p^{e-1}$, so

$$\operatorname{ord}(mP) = \frac{p^e}{p^{e-1}} = p.$$

□

Lemma 3.24 (Chinese Remainder Theorem (CRT))**.** *Let n factor as*

$$n = \prod_{i=1}^{r} n_i$$

with $\gcd(n_i, n_j) = 1$ for all $i \neq j$. Define

$$\varphi : \mathbb{Z}_n \to \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r}, \quad \varphi(x) = (x \bmod n_1, \dots, x \bmod n_r).$$

Then:

1. *φ is a group homomorphism (with respect to addition);*

2. *φ is bijective.*

In particular, each tuple $(a_1, \dots, a_r) \in \mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r}$ corresponds to a unique $x \in \mathbb{Z}_n$ such that

$$x \equiv a_i \pmod{n_i} \quad \textit{for all } i.$$

Proof. Homomorphism: For any $x, y \in \mathbb{Z}_n$,

$$\varphi(x+y) = ((x+y) \bmod n_1, \dots, (x+y) \bmod n_r) = (x \bmod n_1 + y \bmod n_1, \dots),$$

which is $\varphi(x) + \varphi(y)$ in the product group.

Injectivity: Suppose $\varphi(x) = \varphi(y)$. Then

$$x \equiv y \pmod{n_i} \quad \forall i.$$

Thus $n_i \mid (x-y)$ for each i, so $\prod_{i=1}^{r} n_i = n$ divides $(x-y)$ (since the n_i are pairwise coprime). Therefore $x \equiv y \pmod{n}$, i.e. $x = y$ in $\mathbb{Z}_n$.

Surjectivity: Let $(a_1, \dots, a_r)$ be given. Put $N = n$ and $N_i = N/n_i$. Because $\gcd(N_i, n_i) = 1$, there exists M_i such that

$$N_i M_i \equiv 1 \pmod{n_i}.$$

Define

$$x = \sum_{i=1}^{r} a_i N_i M_i \pmod{N}.$$

Then modulo n_j,

$$x \equiv a_j N_j M_j + \sum_{i \neq j} a_i N_i M_i \pmod{n_j}.$$

Since $n_j \mid N_i$ for each $i \neq j$, the sum over $i \neq j$ vanishes modulo n_j, and $N_j M_j \equiv 1 \pmod{n_j}$, so $x \equiv a_j \pmod{n_j}$. Thus $\varphi(x) = (a_1, \dots, a_r)$, proving surjectivity.

Hence φ is bijective. □

This gives the key group-theoretic statement:

Theorem 3.25 (Group Decomposition)**.** *Let $n = \prod_{i=1}^{r} p_i^{e_i}$ be the prime-power factorization of the group order. Then*

$$\mathbb{Z}_n \cong \mathbb{Z}_{p_1^{e_1}} \times \cdots \times \mathbb{Z}_{p_r^{e_r}}.$$

In particular, an exponent $x \in \mathbb{Z}_n$ is uniquely determined by the residues $x_i \equiv x \pmod{p_i^{e_i}}$ for $i = 1, \dots, r$, and x can be reconstructed from (x_i) via the CRT.

Proof. Apply Lemma 3.24 with $n_i = p_i^{e_i}$, which are pairwise coprime. The isomorphism follows directly. □

Reduction Step: Solving Modulo a Prime p

Assume p divides n and write $n = p \cdot m$, so $m = n/p$. Let $Q = xP$ and write

$$x = \ell + mp$$

for some integers ℓ, m, where

$$\ell = x \bmod p$$

is the unknown residue we want.

Multiply $Q = xP$ by m:

$$\begin{aligned} mQ &= m(xP) = (mx)P \\ &= m(\ell + mp)P \\ &= m\ell P + mmpP. \end{aligned}$$

But since $nP = \mathcal{O}$ and $n = pm$,

$$pmP = nP = \mathcal{O}.$$

Hence

$$mmpP = m \cdot (mpP) = m \cdot \mathcal{O} = \mathcal{O}.$$

Define

$$P' = mP = \frac{n}{p}P, \quad Q' = mQ = \frac{n}{p}Q.$$

Then the above simplifies to

$$Q' = mQ = m\ell P = \ell(mP) = \ell P'.$$

By Lemma 3.23 with $m = n/p$, the point P' has order p. Thus $Q' = \ell P'$ is a discrete logarithm problem in a group of order p, and we can recover $\ell = x \bmod p$ using any generic DLP algorithm (e.g. BSGS in $O(\sqrt{p})$ group operations and $O(\sqrt{p})$ space).

Generalization to Prime Powers p^e

Now suppose that $n = p^e m$ for some integer m, and we wish to compute $x \bmod p^e$. Write the p-adic expansion:

$$x \equiv z_0 + z_1 p + \cdots + z_{e-1}p^{e-1} \pmod{p^e},$$

where each $z_j \in \{0, 1, \ldots, p-1\}$ is a base-p digit.

We determine the digits z_j iteratively.

Theorem 3.26 (Digit Extraction for $x \bmod p^e$)**.** *Let $G = \langle P \rangle$ have order $n = p^e m$. Let $Q = xP$ with*

$$x \equiv z_0 + z_1 p + \cdots + z_{e-1}p^{e-1} \pmod{p^e}, \quad z_j \in \{0, \ldots, p-1\}.$$

Define

$$P' = \frac{n}{p}P.$$

Then, for each $k = 0, \ldots, e-1$, after having determined $z_0, \ldots, z_{k-1}$, we can compute z_k by:

1. *Setting*
$$x_{<k} = z_0 + z_1 p + \cdots + z_{k-1}p^{k-1}, \quad Q_k = Q - x_{<k}P;$$
2. *Computing*
$$Q'_k = \frac{n}{p^{k+1}}Q_k;$$
3. *Solving the discrete log in the subgroup of order p:*
$$Q'_k = z_k P'.$$

Proof. We prove the general step for $k \geq 0$, with the convention $x_{<0} = 0$ and $Q_0 = Q$.

By definition of the digits z_j,

$$x = x_{<k} + z_k p^k + p^{k+1} w$$

for some integer w (gathering higher-order terms). Then

$$Q = xP = x_{<k}P + z_k p^k P + p^{k+1} wP.$$

Thus

$$Q_k = Q - x_{<k}P = z_k p^k P + p^{k+1} wP.$$

Now multiply by n/p^{k+1}:

$$\begin{aligned} \frac{n}{p^{k+1}} Q_k &= \frac{n}{p^{k+1}} \left(z_k p^k P + p^{k+1} wP \right) \\ &= z_k \frac{n}{p^{k+1}} p^k P + \frac{n}{p^{k+1}} p^{k+1} wP \\ &= z_k \frac{n}{p} P + nwP. \end{aligned}$$

Since $nP = \mathcal{O}$, the term nwP vanishes, leaving

$$\frac{n}{p^{k+1}} Q_k = z_k \left(\frac{n}{p} P \right) = z_k P'.$$

Thus defining $Q'_k = \frac{n}{p^{k+1}} Q_k$, we obtain

$$Q'_k = z_k P',$$

which is a discrete logarithm problem in the subgroup generated by P'. By Lemma 3.23, P' has order p, so z_k lies in $\{0, \ldots, p-1\}$ and can be uniquely determined by solving this DLP. □

By repeating this procedure for $k = 0, 1, \ldots, e-1$, we recover $z_0, \ldots, z_{e-1}$ and thus $x \bmod p^e$.

3.5.2 Full Pohlig–Hellman Algorithm

We now combine the decomposition of the group order and the digit extraction into the full algorithm.

Algorithm 9 Pohlig–Hellman Algorithm

Require: Base point P, target point Q, group order n, factorization $n = \prod_{i=1}^{r} p_i^{e_i}$
Ensure: Scalar x such that $Q = xP$

Congruences $\leftarrow$ [ ] ▷ List of pairs $(x_i, p_i^{e_i})$
for each prime power (p, e) in the factorization of n **do**
  $x_{\text{sub}} \leftarrow 0$ ▷ This will store x mod p^e
  $\gamma \leftarrow 1$ ▷ Tracks p^k
  $P_{\text{base}} \leftarrow (n/p) \cdot P$ ▷ Order-p base point (Lemma 3.23)
  for $k = 0$ **to** $e - 1$ **do**
    $Q_{\text{temp}} \leftarrow Q - (x_{\text{sub}} \cdot P)$
    $Q_{\text{reduced}} \leftarrow \frac{n}{p^{k+1}} \cdot Q_{\text{temp}}$
    $d_k \leftarrow \text{BSGS}(P_{\text{base}}, Q_{\text{reduced}}, p)$ ▷ Solve $Q_{\text{reduced}} = d_k P_{\text{base}}$
    $x_{\text{sub}} \leftarrow x_{\text{sub}} + d_k \cdot \gamma$
    $\gamma \leftarrow \gamma \cdot p$
  end for
  Append $(x_{\text{sub}} \bmod p^e,\ p^e)$ to *Congruences*
end for
$x \leftarrow \text{CRT}(\textit{Congruences})$ ▷ Use Lemma 3.24 / Theorem 3.25
return x

3.5.3 Complexity Analysis

We now prove time and space complexity bounds.

Theorem 3.27 (Time Complexity of Pohlig–Hellman)**.** *Suppose the factorization*

$$n = \prod_{i=1}^{r} p_i^{e_i}$$

is known. Using Baby-Step Giant-Step (BSGS) as the subroutine for order-p_i discrete logs, the total running time is

$$T(n) = O\left(\sum_{i=1}^{r} e_i \sqrt{p_i} + (\log n)^2\right).$$

In particular, letting $p_{\max} = \max_i p_i$, we have

$$T(n) = O\big(\log n \cdot \sqrt{p_{\max}}\big).$$

Proof. Fix a factor (p, e) and analyze its cost. The inner loop runs for $k = 0, \ldots, e-1$, so there are e iterations.

In each iteration, we perform:

- $Q_{\text{temp}} = Q - x_{\text{sub}} P$: one scalar multiplication by x_{sub} and one group subtraction. Since $x_{\text{sub}} < p^e \le n$, scalar multiplication with double-and-add costs $O(\log n)$ group operations.
- $Q_{\text{reduced}} = \frac{n}{p^{k+1}} Q_{\text{temp}}$: one scalar multiplication by some integer at most n, again $O(\log n)$ group operations.
- $\text{BSGS}(P_{\text{base}}, Q_{\text{reduced}}, p)$: solving a DLP in a group of order p, which costs $O(\sqrt{p})$ group operations.

Thus each iteration costs

$$O(\log n) + O(\log n) + O(\sqrt{p}) = O(\sqrt{p} + \log n).$$

With e iterations, the cost for this factor is

$$O\big(e(\sqrt{p} + \log n)\big) = O\big(e\sqrt{p} + e\log n\big).$$

Summing over all prime powers $(p_i^{e_i})$, we get

$$T(n) = O\left(\sum_{i=1}^{r} e_i\sqrt{p_i} + \sum_{i=1}^{r} e_i \log n\right) = O\left(\sum_{i=1}^{r} e_i\sqrt{p_i} + \left(\sum_{i=1}^{r} e_i\right)\log n\right).$$

We bound $\sum_i e_i$ using the inequality

$$n = \prod_{i=1}^{r} p_i^{e_i} \geq 2^{\sum_i e_i} \implies \sum_i e_i \leq \log_2 n = O(\log n).$$

Therefore

$$T(n) = O\left(\sum_{i=1}^{r} e_i\sqrt{p_i} + (\log n)^2\right).$$

Now let $p_{\max} = \max_i p_i$. Then

$$\sum_{i=1}^{r} e_i\sqrt{p_i} \leq \left(\sum_{i=1}^{r} e_i\right)\sqrt{p_{\max}} \leq (\log n)\sqrt{p_{\max}}.$$

Thus

$$T(n) = O\big((\log n)\sqrt{p_{\max}} + (\log n)^2\big) = O\big(\log n \cdot \sqrt{p_{\max}}\big)$$

for sufficiently large n (since $p_{\max} \geq 2$ so $\sqrt{p_{\max}} \geq 1$ and $(\log n)^2 \leq (\log n)\sqrt{p_{\max}}$ eventually). □

Theorem 3.28 (Space Complexity of Pohlig–Hellman)**.** *Using BSGS as the subroutine, the peak space complexity of Pohlig–Hellman is*

$$S(n) = O\big(\sqrt{p_{\max}}\big),$$

where $p_{\max} = \max_i p_i$ is the largest prime factor of n.

Proof. For a fixed prime p_i, the BSGS subroutine allocates a baby-step table of size $\Theta(\sqrt{p_i})$ group elements. However, factors are processed *sequentially*, and the memory used for one factor is reused for the next.

Therefore, at any point in time, the maximum memory usage due to BSGS is

$$\max_i \Theta(\sqrt{p_i}) = \Theta(\sqrt{p_{\max}}).$$

All other state (current points, integers x_{sub}, γ, the list of congruences) takes $O(r)$ integers of size $O(\log n)$, which is asymptotically negligible compared to $\sqrt{p_{\max}}$ for cryptographic-size parameters.

Hence the overall space complexity is $O(\sqrt{p_{\max}})$. □

Implication: Smooth Orders Are Weak. If n is *smooth*, meaning every prime factor p_i is at most $(\log n)^c$ for some constant c, then

$$\sqrt{p_{\max}} \leq (\log n)^{c/2},$$

so $T(n)$ is polynomial in $\log n$. In that case, the discrete logarithm problem in G is solvable in polynomial time, which is insecure. Thus cryptographic groups must have an order divisible by at least one very large prime.

3.5.4 [BONUS]Adaptation: Residue Leakage (Side-Channel)

In side-channel scenarios, an attacker may learn x modulo one or more prime-power factors of n. We formalize the impact of such leakage.

Theorem 3.29 (Effect of Residue Leakage)**.** *Let $n = \prod_{i=1}^{r} p_i^{e_i}$, and suppose that for some index j the residue*

$$x \bmod p_j^{e_j}$$

is known in advance (e.g., from a side channel). Then the cost of Pohlig–Hellman no longer includes the term $e_j\sqrt{p_j}$. In particular, if $p_j = p_{\max}$ is the largest prime factor, the new running time is

$$T_{new}(n) = O\big(\log n \cdot \sqrt{p_{second\_max}}\big),$$

where p_{second_max} is the second-largest prime factor of n.

Proof. In the standard algorithm, each factor $p_i^{e_i}$ contributes a time term of order $e_i\sqrt{p_i}$ (Theorem 3.27). If $x \bmod p_j^{e_j}$ is already known, we do not need to run the digit extraction for $p_j^{e_j}$; instead we simply add the pair

$$(x_j, p_j^{e_j})$$

directly to the list of congruences, at $O(1)$ cost. The total time therefore becomes

$$T_{\text{new}}(n) = O\left(\sum_{i\neq j} e_i\sqrt{p_i} + (\log n)^2\right).$$

If $p_j = p_{\max}$, then the largest term in the original sum is removed. Let $p_{\text{second\_max}}$ be the largest prime among $\{p_i : i \neq j\}$. Then

$$\sum_{i\neq j} e_i\sqrt{p_i} \leq \left(\sum_{i\neq j} e_i\right)\sqrt{p_{\text{second\_max}}} \leq (\log n)\sqrt{p_{\text{second\_max}}},$$

since $\sum_i e_i \leq \log_2 n$. Thus

$$T_{\text{new}}(n) = O\big(\log n \cdot \sqrt{p_{\text{second\_max}}}\big).$$

□

Complexity Reduction Summary. Let the prime factors be ordered $p_1 \leq p_2 \leq \cdots \leq p_{\max}$.

- **Standard Complexity:** $T(n) \approx \log n \cdot \sqrt{p_{\max}}$.
- **With Leakage of $x \bmod p_{\max}^{e_{\max}}$:** The dominant term $e_{\max}\sqrt{p_{\max}}$ is removed.
- **New Complexity:** $T_{\text{new}}(n) \approx \log n \cdot \sqrt{p_{\text{second\_max}}}$.

Thus leaking the residue of x modulo the largest prime-power factor of n is catastrophic: it effectively reduces the security of the discrete logarithm to that of the second-largest factor.

3.6 Las Vegas Algorithm: Summation Polynomials

3.6.1 Overview

This section describes a probabilistic *Las Vegas* algorithm for solving the ECDLP, inspired by Index Calculus ideas and Semaev's summation polynomials [**5**, **23**]. Unlike generic collision-search algorithms such as Pollard's Rho, which view the group as a black box, this method exploits the algebraic geometry of the elliptic curve [**6**].

The core idea is:

- use algebraic curves of low degree that intersect the elliptic curve in many points,
- interpret those intersection points as a relation $\sum P_i = \mathcal{O}$ in the group law,
- and reduce the search for such relations to a linear-algebra problem (finding low-weight vectors in a certain linear code).

The algorithm is Las Vegas: whenever it outputs a discrete logarithm, that answer is always correct; however, it may fail and restart with fresh randomness.

3.6.2 Geometric Foundation

We work with an elliptic curve $\mathcal{E}$ over a finite field $\mathbb{F}_q$, given by a (projective) Weierstrass equation of degree 3 in homogeneous coordinates $(X : Y : Z)$. The group law on $\mathcal{E}(\mathbb{F}_q)$ can be described in terms of divisors and the Picard group $\mathrm{Pic}^0(\mathcal{E})$, but we only need the basic correspondences.

Lemma 3.30 (Divisors and the Group Law)**.** *Let $\mathcal{E}$ be an elliptic curve with neutral point $\mathcal{O}$. Then:*

1. *The map*
$$\phi : \mathcal{E}(\overline{\mathbb{F}}_q) \to \mathrm{Pic}^0(\mathcal{E}), \quad P \mapsto [P] - [\mathcal{O}]$$
is a group isomorphism.

2. *A divisor of the form*
$$D = \sum_{i=1}^{k} [P_i] - k[\mathcal{O}]$$
is principal (i.e. $D = \mathrm{div}(f)$ for some rational function f on $\mathcal{E}$) if and only if $\sum_{i=1}^{k} P_i = \mathcal{O}$ in $\mathcal{E}(\overline{\mathbb{F}}_q)$.

Proof. This is a standard result in the theory of elliptic curves: $\mathrm{Pic}^0(\mathcal{E})$ is the Jacobian of $\mathcal{E}$, and for an elliptic curve it is isomorphic to the curve itself. The map $P \mapsto [P] - [\mathcal{O}]$ is known to be a group isomorphism; we sketch the second statement.

If $D = \mathrm{div}(f)$ is principal, then D has degree 0, and f defines the identity element in $\mathrm{Pic}^0(\mathcal{E})$. Under the isomorphism, the divisor

$$D = \sum_{i=1}^{k} [P_i] - k[\mathcal{O}]$$

corresponds to the group element $\sum_{i=1}^{k} P_i - k\mathcal{O}$; D is principal if and only if this is the neutral element, i.e. $\sum_{i=1}^{k} P_i = \mathcal{O}$. The converse direction is identical: if $\sum P_i = \mathcal{O}$, then the corresponding class is trivial, hence represented by a principal divisor. Full proofs can be found in any standard elliptic curve text. □

We also use Bézout's theorem for intersections of plane curves.

Lemma 3.31 (Bézout's Theorem for Plane Curves)**.** *Let C_1, C_2 be projective plane curves over an algebraically closed field, defined by homogeneous polynomials of degrees d_1 and d_2 with no common component. Then the sum of the intersection multiplicities of C_1 and C_2 over all intersection points is exactly $d_1 d_2$.*

Proof. This is a standard theorem from algebraic geometry; we will use only the special case where C_1 is a cubic (the elliptic curve) and C_2 has degree n'. We refer to [**6**] for a complete proof. □

We can now state the main geometric statement in a precise form.

Theorem 3.32 (Geometric Summation Condition)**.** *Let $\mathcal{E}$ be an elliptic curve in the projective plane given by a cubic equation, and let $n' \geq 1$ be an integer. Let $k = 3n'$. A multiset of points $\{P_1, \ldots, P_k\} \subset \mathcal{E}(\overline{\mathbb{F}}_q)$ (counted with multiplicities) satisfies*

$$\sum_{i=1}^{k} P_i = \mathcal{O}$$

in the group law if and only if there exists a homogeneous polynomial $F(X, Y, Z)$ of degree n' such that the curve $C : F = 0$ intersects $\mathcal{E}$ exactly in the points P_i, counted with multiplicity.

Proof. We work over the algebraic closure so we see all intersection points and divisors.

($\Longrightarrow$) Assume $\sum_{i=1}^{k} P_i = \mathcal{O}$. Consider the divisor

$$D = \sum_{i=1}^{k} [P_i] - k[\mathcal{O}].$$

By Lemma 3.30, this divisor class is the identity in $\mathrm{Pic}^0(\mathcal{E})$, hence D is principal: there exists a rational function f on $\mathcal{E}$ with $\mathrm{div}(f) = D$.

For an elliptic curve embedded as a cubic plane curve, functions with prescribed poles only at $\mathcal{O}$ and bounded pole order correspond to restrictions of rational functions of the form $F(X, Y, Z)/Z^{n'}$, where F is homogeneous of some degree. For our divisor D, the total pole order at $\mathcal{O}$ is $k = 3n'$ by construction; by Riemann–Roch, there exists F of degree n' such that $f = F/Z^{n'}$ (up to scalar). Then

$$\mathrm{div}(F/Z^{n'}) = \sum_{i=1}^{k} [P_i] - 3n'[\mathcal{O}],$$

so the zero divisor of F restricted to $\mathcal{E}$ is $\sum_i [P_i]$. That is, the curve $C : F = 0$ intersects $\mathcal{E}$ exactly at the P_i (with multiplicities).

($\Longleftarrow$) Conversely, suppose there exists a degree-n' curve $C : F = 0$ intersecting $\mathcal{E}$ at points $P_1, \ldots, P_k$ (with multiplicities). By Bézout's theorem (Lemma 3.31), the total intersection multiplicity is $3n' = k$. The divisor of the function $f = F/Z^{n'}$ on $\mathcal{E}$ has the form

$$\mathrm{div}(f) = \sum_{i=1}^{k} [P_i] - 3n'[\mathcal{O}] = \sum_{i=1}^{k} [P_i] - k[\mathcal{O}].$$

This is a principal divisor, so by Lemma 3.30 it corresponds to the identity element in $\mathcal{E}(\overline{\mathbb{F}}_q)$. Hence $\sum_{i=1}^{k} P_i = \mathcal{O}$. □

Thus, finding a subset of $k = 3n'$ points on $\mathcal{E}$ summing to $\mathcal{O}$ is equivalent to finding a curve $F = 0$ of degree n' that intersects $\mathcal{E}$ exactly in those points.

3.6.3 Algorithm Description

We now convert the geometric criterion into a concrete algorithm. Fix a degree parameter $n' \geq 1$, and set $k = 3n'$. We will generate $N > k$ random points on $\mathcal{E}$ and look for a subset of size k that lies on some curve of degree n'.

Point Generation and Monomial Basis

Let P be the base point and Q the target with $Q = dP$ for some unknown d. We generate a pool of N random points by selecting random integers r_m, s_m modulo the group order n and setting

$$T_m = r_m P - s_m Q \qquad \text{for } m = 1, \ldots, N. \tag{3.11}$$

Each T_m can be written as

$$T_m = (r_m - s_m d)P,$$

so any relation $\sum T_m = \mathcal{O}$ will translate into a linear congruence in d.

To encode the condition "T_m lies on some degree-n' curve $F = 0$", we choose a basis of homogeneous monomials of degree n':

$$\mathcal{B} = \{X^i Y^j Z^{n'-i-j} \mid i, j \geq 0,\ i + j \leq n'\}.$$

The number of such monomials is

$$D = \binom{n'+2}{2},$$

which we prove now.

Lemma 3.33 (Number of Degree-n' Monomials)**.** *The number D of triples (i, j, k) of nonnegative integers satisfying $i + j + k = n'$ is*

$$D = \binom{n'+2}{2}.$$

Equivalently, the number of monomials in $\mathcal{B}$ is D.

Proof. The number of monomials of total degree n' in three variables is the number of integer solutions (i, j, k) to

$$i + j + k = n', \quad i, j, k \geq 0.$$

This is a standard stars-and-bars problem. We interpret n' as the number of indistinguishable balls and 3 as the number of bins. The number of ways to distribute the balls into bins is

$$\binom{n'+3-1}{3-1} = \binom{n'+2}{2}.$$

For each (i, j, k), the monomial is $X^i Y^j Z^k$, so the number of monomials is $D = \binom{n'+2}{2}$. □

For each point $T_m = (x_m, y_m)$ in affine coordinates (with $Z = 1$), we evaluate all basis monomials of degree n' to form a *monomial evaluation vector*

$$\mathbf{v}_m \in \mathbb{F}_q^D.$$

For example, when $n' = 2$, the basis can be taken as $\{x^2, xy, y^2, x, y, 1\}$, and

$$\mathbf{v}_m = [x_m^2,\ x_m y_m,\ y_m^2,\ x_m,\ y_m,\ 1]^T.$$

Evaluation Matrix and Linear Conditions

We assemble the evaluation vectors into a matrix

$$\mathcal{M} = \begin{pmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_N \\ | & | & & | \end{pmatrix} \in \mathbb{F}_q^{D \times N}.$$

Let $\mathbf{c} \in \mathbb{F}_q^D$ represent the coefficients of a homogeneous polynomial F of degree n' in the basis $\mathcal{B}$. Then the value of F at T_m is proportional to the dot product

$$F(T_m) = \mathbf{c}^T \mathbf{v}_m.$$

Hence the vector

$$\mathbf{u} = \mathbf{c}^T \mathcal{M} \in \mathbb{F}_q^N$$

encodes the evaluations of F at all T_m:

$$u_m = (\mathbf{c}^T \mathcal{M})_m = \mathbf{c}^T \mathbf{v}_m = F(T_m).$$

Lemma 3.34 (Curve Membership and Linear Constraints)**.** *Let $\mathbf{c} \in \mathbb{F}_q^D$ be nonzero. A point T_m lies on the curve $F = 0$ defined by $\mathbf{c}$ if and only if the m-th coordinate of $\mathbf{u} = \mathbf{c}^T \mathcal{M}$ is zero. In particular, if $\mathbf{u}$ has zeros exactly at indices in a set $I \subset \{1, \dots, N\}$ with $|I| = k = 3n'$, then the points $\{T_m\}_{m \in I}$ lie on a common degree-n' curve and, by Theorem 3.32, satisfy*

$$\sum_{m \in I} T_m = \mathcal{O}.$$

Proof. The first statement follows directly from the definition of $\mathbf{u}$: $u_m = 0$ if and only if $F(T_m) = 0$, i.e. T_m lies on the curve $F = 0$.

If $\mathbf{u}$ vanishes exactly at indices I and $|I| = k = 3n'$, then the curve $F = 0$ intersects the elliptic curve (as a divisor) in exactly those points, counted with multiplicity. Thus Theorem 3.32 applies and yields $\sum_{m \in I} T_m = \mathcal{O}$. □

We are therefore seeking a nonzero $\mathbf{c}$ such that $\mathbf{u} = \mathbf{c}^T \mathcal{M}$ has Hamming weight $N - k$, i.e. exactly k zeros.

Randomized Linear-Algebra Step

The algorithm does not try to search over all vectors $\mathbf{c} \in \mathbb{F}_q^D$ (which is infeasible). Instead, it uses randomized linear algebra:

- randomly select a subset $J \subset \{1, \dots, N\}$ of columns with $|J|$ slightly larger than D,
- compute the kernel of $\mathcal{M}_J$ (the submatrix of $\mathcal{M}$ with columns indexed by J),
- for each nonzero kernel vector $\mathbf{c}$, compute $\mathbf{u} = \mathbf{c}^T \mathcal{M}$ and count its zeros,
- if $\mathbf{u}$ has exactly $k = 3n'$ zeros at indices I, then by Lemma 3.34 we have a relation $\sum_{m \in I} T_m = \mathcal{O}$.

Once such an I is found, we can extract the discrete logarithm.

Extracting the Discrete Logarithm

Suppose we have found a subset $I \subset \{1, \ldots, N\}$ of size k such that the corresponding points lie on a degree-n' curve and hence

$$\sum_{m \in I} T_m = \mathcal{O}.$$

Recall $T_m = r_m P - s_m Q$; therefore

$$\sum_{m \in I} (r_m P - s_m Q) = \mathcal{O} \iff \left(\sum_{m \in I} r_m \right) P = \left(\sum_{m \in I} s_m \right) Q.$$

Let

$$R = \sum_{m \in I} r_m \pmod{n}, \quad S = \sum_{m \in I} s_m \pmod{n}.$$

We get

$$RP = SQ = S(dP) = (Sd)P.$$

Thus d satisfies

$$R \equiv Sd \pmod{n}.$$

Lemma 3.35 (Solving the Final Scalar Equation)**.** *Let n be the group order, and suppose $R \equiv Sd \pmod{n}$ with $R, S \in \mathbb{Z}_n$. If $\gcd(S, n) = 1$, then d is uniquely determined modulo n by*

$$d \equiv RS^{-1} \pmod{n}.$$

If $\gcd(S, n) = g > 1$ and $g \mid R$, then there are exactly g solutions modulo n, which can be checked by direct verification $Q \stackrel{?}{=} dP$.

Proof. The congruence $Sd \equiv R \pmod{n}$ is a linear congruence in d. By Lemma 3.36 below (applied with $a = S$, $b = R$), if $g = \gcd(S, n)$ does not divide R, there is no solution; if $g \mid R$, there are exactly g solutions modulo n. The formula in the coprime case ($g = 1$) follows because S has a multiplicative inverse modulo n. □

For completeness, we restate and prove the linear congruence fact used above (similar to the one used in Pollard's Rho).

Lemma 3.36 (Solutions of Linear Congruences)**.** *Let $n \in \mathbb{N}$, and let $a, b \in \mathbb{Z}$. Let $d = \gcd(a, n)$.*

1. *If $d \nmid b$, then the congruence $ax \equiv b \pmod{n}$ has no solution.*
2. *If $d \mid b$, then it has exactly d distinct solutions modulo n.*

Proof. Write $a = da'$, $n = dn'$ with $\gcd(a', n') = 1$. If $ax \equiv b \pmod{n}$, then $ax - b = yn$ for some y, so

$$b = ax - yn = da'x - dn'y = d(a'x - n'y),$$

hence $d \mid b$. This proves (1).

Conversely, if $b = db'$ for some b', then $ax \equiv b \pmod{n}$ is equivalent to

$$da'x \equiv db' \pmod{dn'} \iff a'x \equiv b' \pmod{n'}.$$

Since $\gcd(a', n') = 1$, this has a unique solution modulo n', say $x \equiv x_0 \pmod{n'}$. The solutions modulo n are then

$$x \equiv x_0 + tn' \pmod{n}, \quad t = 0, 1, \ldots, d - 1,$$

which are d distinct residue classes modulo n. □

Thus, whenever $\gcd(S, n) = 1$, we recover d uniquely as $d \equiv RS^{-1} \pmod{n}$; in other cases we can try all candidate d values and check which one satisfies $Q = dP$.

3.6.4 Probability and Las Vegas Nature

We now analyze the algorithm's success probability per trial and the expected number of trials. The analysis is necessarily heuristic and uses results from Mahalanobis and Mallick [**5**] as input.

Phase Decomposition

A single trial of the algorithm consists of:

- **Phase 1:** Generating N random points $T_1, \dots, T_N$ and constructing the evaluation matrix $\mathcal{M}$. We succeed in Phase 1 if there exists at least one subset of $k = 3n'$ points among the T_m that sums to $\mathcal{O}$ (equivalently, lies on some degree-n' curve).
- **Phase 2:** Running the randomized linear-algebra procedure to find a vector $\mathbf{c}$ such that $\mathbf{u} = \mathbf{c}^T\mathcal{M}$ has exactly k zeros, and using the corresponding subset to solve for d.

Let E_1 be the event that Phase 1 is successful, and E_2 the event that the particular linear-algebra strategy finds a suitable $\mathbf{c}$, given that E_1 has occurred. The overall success event is $E = E_1 \cap E_2$.

By the law of conditional probability,

$$\Pr(E) = \Pr(E_1) \cdot \Pr(E_2 \mid E_1).$$

Heuristic Estimates from the Literature

Mahalanobis and Mallick analyze specific parameter choices (e.g. $n_0 \approx c \log p$ and $\ell = 3n_0$) and prove the following two facts for large prime fields $\mathbb{F}_p$; we state them as assumptions for our derivation:

Lemma 3.37 (Phase 1 Success Probability (Mahalanobis–Mallick))**.** *For suitable parameter choices n_0 and $\ell = 3n_0$ with $n_0 \approx O(\log p)$, the probability that the random pool of points contains a suitable subset giving a geometric relation (i.e. E_1 occurs) satisfies*

$$\Pr(E_1) \approx 1 - e^{-1} \approx 0.6321$$

as $p \to \infty$.

Lemma 3.38 (Conditional Success of Sparse Kernel Search)**.** *Under the same setup, the probability that the randomized linear-algebra procedure finds a vector with exactly ℓ zeros (i.e. E_2 occurs) given that E_1 holds satisfies*

$$\Pr(E_2 \mid E_1) \approx C \cdot \frac{\ell^2}{p}$$

for some constant $C > 0$. Substituting $\ell \approx 3n_0 \approx 3 \log p$ yields

$$\Pr(E_2 \mid E_1) \approx \Theta\left(\frac{(\log p)^2}{p}\right).$$

Proof. We treat these as results imported from [**5**]. Intuitively, E_1 has constant probability because we tune parameters so that the expected number of zero-sum k-tuples is about 1; E_2 has probability roughly proportional to the chance that random kernel vectors produce the correct sparse zero pattern, which is inversely proportional to the field size p, up to quadratic dependence on ℓ. Full proofs rely on detailed combinatorial and coding-theoretic analysis. □

We can now combine these to derive the overall success and failure probabilities of a single trial.

Theorem 3.39 (Failure Probability per Trial)**.** *Let p be the prime underlying the field $\mathbb{F}_p$. Under the assumptions of Lemmas 3.37 and 3.38, the success probability π_p of a single trial of the Las Vegas algorithm satisfies*

$$\pi_p = \Pr(E) \approx \left(1 - e^{-1}\right) \cdot \frac{(\log p)^2}{p}.$$

Consequently, the failure probability per trial is

$$\Pr(\mathit{Fail}) = 1 - \pi_p \approx 1 - \left(1 - e^{-1}\right) \frac{(\log p)^2}{p},$$

which tends to 1 as $p \to \infty$.

Proof. By definition,

$$\Pr(E) = \Pr(E_1) \Pr(E_2 \mid E_1).$$

Applying Lemma 3.37 gives $\Pr(E_1) \approx 1 - e^{-1}$, and Lemma 3.38 gives $\Pr(E_2 \mid E_1) \approx C \cdot (\log p)^2/p$ for some constant C. Absorbing C into the approximation (or taking $C \approx 1$ for simplicity of notation), we write

$$\pi_p = \Pr(E) \approx (1 - e^{-1}) \frac{(\log p)^2}{p}.$$

The failure probability is the complement:

$$\Pr(\text{Fail}) = 1 - \pi_p \approx 1 - \left(1 - e^{-1}\right) \frac{(\log p)^2}{p}.$$

As $p \to \infty$, $(\log p)^2/p \to 0$, so $\Pr(\text{Fail}) \to 1$. □

[Expected Number of Trials and Asymptotic Cost] Let π_p be the per-trial success probability. A Las Vegas algorithm that repeats independent trials until success has a number of trials T distributed geometrically with parameter π_p, so

$$\mathbb{E}[T] = \frac{1}{\pi_p} \approx \frac{p}{(1 - e^{-1})(\log p)^2} = \Theta\left(\frac{p}{(\log p)^2}\right)$$

as $p \to \infty$. Thus, even if each trial runs in time polynomial in $\log p$, the expected overall running time grows on the order of $p/(\log p)^2$, which is asymptotically worse than Pollard's Rho ($O(\sqrt{p})$) for large p.

Proof. For a geometric random variable with success probability π_p,

$$\mathbb{E}[T] = \frac{1}{\pi_p}.$$

Substituting the expression from Theorem 3.39 yields

$$\mathbb{E}[T] \approx \frac{p}{(1 - e^{-1})(\log p)^2}.$$

Asymptotically, this is $\Theta\left(p/(\log p)^2\right)$. □

3.6.5 Combinatorial Existence of Relations and Complexity

We now derive a simpler combinatorial heuristic for the pool size N needed to expect at least one k-tuple summing to $\mathcal{O}$, and then translate that into time and space complexity.

Theorem 3.40 (Expected Number of Relations and Pool Size)**.** *Let the group order be $n \approx p$ (prime). Fix $k = 3n'$ and assume that a random k-tuple of points in $\mathcal{E}(\mathbb{F}_p)$ has probability approximately $1/n$ of summing to $\mathcal{O}$. If we choose N random points independently and consider all $\binom{N}{k}$ k-subsets, then the expected number of relations of the form $\sum_{i=1}^{k} P_{j_i} = \mathcal{O}$ is approximately*

$$\mathbb{E}[\#\textit{relations}] \approx \frac{\binom{N}{k}}{n}.$$

In particular, choosing N such that

$$\binom{N}{k} \approx n$$

makes the expected number of relations close to 1. For fixed k and large n, this corresponds roughly to

$$N \approx (k! \cdot n)^{1/k} = \Theta\left(n^{1/k}\right).$$

Proof. Assume that among the $\binom{N}{k}$ subsets of size k, each subset independently has probability $1/n$ of summing to the identity. Let X be the random variable counting the number of such subsets that do sum to $\mathcal{O}$. Then

$$X = \sum_{S} I_S,$$

where I_S is the indicator of the event "subset S sums to $\mathcal{O}$". By linearity of expectation,

$$\mathbb{E}[X] = \sum_{S} \mathbb{E}[I_S] = \sum_{S} \Pr(I_S = 1) = \binom{N}{k} \cdot \frac{1}{n}.$$

Setting $\mathbb{E}[X] \approx 1$ yields

$$\binom{N}{k} \approx n.$$

For fixed k and large N, the binomial coefficient satisfies

$$\binom{N}{k} = \frac{N(N-1)\dots(N-k+1)}{k!} \approx \frac{N^k}{k!}.$$

Equating $\frac{N^k}{k!} \approx n$ gives

$$N^k \approx k! \cdot n \implies N \approx (k! \cdot n)^{1/k}.$$

Since k is fixed (e.g. $k = 3n'$ with small n'), we have $N = \Theta(n^{1/k})$. □

In particular, for $n' = 2$ (so $k = 6$), we obtain

$$N \approx (6! \cdot n)^{1/6} = \Theta(n^{1/6}),$$

and for large prime order $n \approx p$, we have $N = \Theta(p^{1/6})$.

Time Complexity Per Trial

We now estimate the cost of one trial in terms of N, D, and p.

Theorem 3.41 (Time Complexity Per Trial)**.** *Let N be the number of random points generated and $D = \binom{n'+2}{2}$ the number of degree-n' monomials (Lemma 3.33). Then:*

1. *Constructing the matrix $\mathcal{M} \in \mathbb{F}_q^{D \times N}$ takes $O(DN)$ field operations.*
2. *A naive Gaussian elimination on $\mathcal{M}$ (or a $D \times N$ submatrix) takes $O(D^2 N)$ field operations.*

If n' is fixed (e.g. 2 or 3), then D is a constant, and the overall arithmetic cost per trial is $O(N)$ field operations. For the parameter choice $N = \Theta(p^{1/k})$ with $k = 3n'$, this is $O(p^{1/k})$ per trial.

Proof. (1) To build $\mathcal{M}$, for each of the N points T_m we evaluate all D monomials in the basis $\mathcal{B}$. If each monomial evaluation takes $O(1)$ field operations (since we can reuse powers of x_m, y_m), then the work per point is $O(D)$, and the total is $O(DN)$.

(2) Gaussian elimination on a $D \times N$ matrix, with $D \leq N$, generally requires $O(D^2 N)$ field operations: each pivot row is used to eliminate entries in $O(N)$ columns, and there are D pivot positions, with $O(D)$ work per pivot-column pair. More precisely, the usual $O(\min(D, N)^2 \max(D, N))$ bound applies, which here is $O(D^2 N)$.

If n' is treated as a fixed constant, then $D = \binom{n'+2}{2}$ is also a fixed constant (e.g. $D = 6$ for $n' = 2$). In that case $O(DN)$ and $O(D^2 N)$ both simplify to $O(N)$ field operations.

Finally, substituting $N = \Theta(p^{1/k})$ from Theorem 3.40, the per-trial arithmetic cost becomes $O(p^{1/k})$. □

Space Complexity Per Trial

Theorem 3.42 (Space Complexity Per Trial)**.** *The memory usage of one trial of the algorithm is $O(DN)$ field elements to store the matrix $\mathcal{M}$ plus the points T_m. For fixed degree n', this reduces to $O(N)$ field elements, which is $O(p^{1/k})$ for $N = \Theta(p^{1/k})$.*

Proof. We store:

- the N points T_m (each point consists of a small constant number of field elements),
- the matrix $\mathcal{M}$ of size $D \times N$ (each entry is a field element),
- a small number of auxiliary vectors (e.g. kernel basis, coefficients).

Thus the total number of stored field elements is $O(DN)$ plus lower-order terms. For fixed n', D is constant, so this is $O(N)$ field elements. Using $N = \Theta(p^{1/k})$, the space complexity is $O(p^{1/k})$. □

Overall Expected Complexity

Combining Corollary 3.6.4 and Theorem 3.41, the *expected* total arithmetic complexity is roughly

$$\mathbb{E}[\text{time}] \approx \mathbb{E}[T] \cdot O(p^{1/k}) = \Theta\left(\frac{p}{(\log p)^2} \cdot p^{1/k} \right),$$

but since $1/k$ is small (and often k is fixed while p grows), the dominant asymptotic behavior is governed by the $p/(\log p)^2$ factor coming from the low success probability.

In particular, for large p, this Las Vegas summation-polynomial approach is asymptotically slower than Pollard's Rho, which runs in expected time $O(\sqrt{p})$ with negligible failure probability, and also uses only $O(1)$ memory. This justifies treating the summation-polynomial algorithm as a theoretical Las Vegas benchmark rather than a practical generic attack on ECDLP for random curves.

3.6.6 [BONUS]Bonus Adaptation: Las Vegas with Approximate Hint (Gaussian Sampling)

In some side–channel scenarios, the attacker does not know a strict interval for the secret key d, but instead learns an *approximate value*

$$h \approx d$$

together with a bound on the error $|d - h| \leq E$ (for example, E may come from a power–analysis estimate of the Hamming weight). In this setting, we can design a Las Vegas algorithm that repeatedly samples candidate keys in a probabilistic way, biased around the hint h.

Problem Setup

Let $G = \langle P \rangle$ be a cyclic group of order n (the subgroup used for ECDLP), and let $Q = dP$ be the public point. Assume:

- There exists a unique $d \in \{0, \ldots, n-1\}$ with $Q = dP$.
- We are given a hint $h \in \mathbb{Z}_n$ and an error bound $E \in \mathbb{N}$ such that

$$|d - h| \leq E$$

  with high probability (e.g. 99.7%).

We model the unknown error

$$e = d - h \pmod{n}$$

as coming from a discrete Gaussian distribution with mean 0 and standard deviation σ, and we take $E \approx 3\sigma$ (so that $|e| \leq 3\sigma$ with probability $\approx 99.7\%$).

Algorithm Description

The Python code implements the following simple Las Vegas algorithm:

Algorithm 10 Las Vegas with Approximate Hint (Gaussian Sampling)

Require: Base point P, target $Q = dP$, group order n, approximate hint h, standard deviation σ, iteration limit L
Ensure: Either the discrete logarithm d or **Failure**

for $i = 1$ **to** L **do**
  Sample an integer offset δ from a discrete Gaussian distribution with mean 0 and standard deviation σ
  $d_{\text{cand}} \leftarrow (h + \delta) \bmod n$
  if $d_{\text{cand}}P = Q$ **then**
    return d_{cand} ▷ Correct key found
  end if
end for
return Failure

The code implementation uses `random.gauss(0,` σ)$and convert s the real-valued sample to an integer offs$

Correctness

Lemma 3.43 (Correctness of a Successful Output)**.** *Assume that the subgroup generated by P has order n, and that there is a unique $d \in \{0, \dots, n-1\}$ such that $Q = dP$. If Algorithm 10 returns a value d_{cand}, then $d_{cand} = d$.*

Proof. The algorithm returns d_{cand} only if $d_{\text{cand}}P = Q$. By assumption $Q = dP$, hence

$$d_{\text{cand}}P = dP.$$

Subtracting dP from both sides in the group gives

$$(d_{\text{cand}} - d)P = \mathcal{O}.$$

Since P has order n, this implies that n divides $d_{\text{cand}} - d$; i.e.,

$$d_{\text{cand}} \equiv d \pmod{n}.$$

Both d_{cand} and d are taken in the range $\{0, \dots, n-1\}$, so the congruence implies equality $d_{\text{cand}} = d$. □

Thus the algorithm is *Las Vegas*: any value it returns is guaranteed to be correct, and failure simply means that the random search did not hit the true key within the allotted number of trials.

Success Probability per Trial

We now estimate the success probability of a single iteration under a simple Gaussian error model.

Definition 3.6.1 (Discrete Gaussian Model)**.** *Let $e = d - h \in \mathbb{Z}$ be the (unknown) error. In each iteration, the algorithm samples an integer offset δ according to a discrete Gaussian distribution centered at 0 with standard deviation σ, i.e. the probability mass function is*

$$\Pr(\delta = t) = \frac{1}{Z_\sigma} \exp\left(-\frac{t^2}{2\sigma^2}\right), \quad t \in \mathbb{Z},$$

for a normalization constant $Z_\sigma = \sum_{t \in \mathbb{Z}} \exp\left(-t^2/(2\sigma^2)\right)$.

In each iteration, the candidate key is $d_{\text{cand}} = h + \delta \pmod{n}$. A single trial succeeds if and only if $\delta = e$ in $\mathbb{Z}$ (ignoring wraparound, which is negligible when $E \ll n$).

Lemma 3.44 (Per-Trial Success Probability)**.** *Assume $|e| \le 3\sigma$ and $n \gg \sigma$ so that modular wraparound can be neglected. Then the probability p_σ that a single iteration of Algorithm 10 succeeds satisfies*

$$p_\sigma = \Pr(\delta = e) = \frac{1}{Z_\sigma} \exp\left(-\frac{e^2}{2\sigma^2}\right).$$

Moreover, there exist constants $c_1, c_2 > 0$ (independent of σ) such that

$$\frac{c_1}{\sigma} \le p_\sigma \le \frac{c_2}{\sigma}$$

for all σ large enough and all integers e with $|e| \le 3\sigma$.

Proof. By definition of the discrete Gaussian,

$$p_\sigma = \Pr(\delta = e) = \frac{\exp\left(-\frac{e^2}{2\sigma^2}\right)}{\sum_{t\in\mathbb{Z}} \exp\left(-\frac{t^2}{2\sigma^2}\right)} = \frac{1}{Z_\sigma} \exp\left(-\frac{e^2}{2\sigma^2}\right).$$

We now bound Z_σ and the exponential factor. First, for all $t \in \mathbb{Z}$,

$$\exp\left(-\frac{t^2}{2\sigma^2}\right) \le 1,$$

hence

$$Z_\sigma = \sum_{t\in\mathbb{Z}} \exp\left(-\frac{t^2}{2\sigma^2}\right) \le 1 + 2\sum_{t=1}^{\infty} 1 = \infty,$$

so this trivial bound is useless. Instead, we compare the sum to the corresponding Gaussian integral. For $t \ge 1$ we have

$$\exp\left(-\frac{t^2}{2\sigma^2}\right) \le \int_{t-1}^{t} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx,$$

and similarly for $t \le -1$. Therefore

$$Z_\sigma = 1 + 2\sum_{t=1}^{\infty} \exp\left(-\frac{t^2}{2\sigma^2}\right) \le 1 + 2\int_0^{\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx.$$

The Gaussian integral evaluates to

$$\int_0^{\infty} \exp\left(-\frac{x^2}{2\sigma^2}\right) dx = \sigma\sqrt{\frac{\pi}{2}}.$$

Thus

$$Z_\sigma \le 1 + 2\sigma\sqrt{\frac{\pi}{2}} \le C_2\sigma$$

for some constant $C_2 > 0$ and all $\sigma \ge 1$.

Similarly, one can bound Z_σ from below by comparing each term to the maximum of the integrand on the corresponding interval, obtaining

$$Z_\sigma \ge C_1\sigma$$

for some constant $C_1 > 0$ and all σ large enough. (Intuitively, Z_σ grows proportionally to σ because the Gaussian spreads out.)

Next, assume $|e| \le 3\sigma$. Then

$$\exp\left(-\frac{e^2}{2\sigma^2}\right) \ge \exp\left(-\frac{(3\sigma)^2}{2\sigma^2}\right) = \exp\left(-\frac{9}{2}\right) =: c_0 > 0,$$

and clearly $\exp\big(-e^2/(2\sigma^2)\big) \le 1$. Combining these bounds,

$$p_\sigma = \frac{\exp\big(-e^2/(2\sigma^2)\big)}{Z_\sigma} \ge \frac{c_0}{C_2\sigma} = \frac{c_1}{\sigma},$$

and

$$p_\sigma \le \frac{1}{C_1\sigma} = \frac{c_2}{\sigma}.$$

This proves the claimed inequalities. □

`Thus, for ‘‘typical’’ errors` $|e| \le 3\sigma$`, a single trial succeeds with probability on the order of` $1/\sigma$`.`

Expected Running Time and Space

`We now turn this per-trial bound into a complexity estimate.`

Theorem 3.45 (Time Complexity of Gaussian Las Vegas Adaptation)**.** *Assume the error $e = d - h$ satisfies $|e| \leq 3\sigma$ with high probability and the subgroup order n is much larger than σ so that wraparound can be neglected. Then, ignoring the fixed iteration cap L and treating each iteration as independent, the number T of iterations until success is a geometric random variable with success probability p_σ, and*

$$\mathbb{E}[T] = \frac{1}{p_\sigma} = \Theta(\sigma).$$

Since each iteration performs a single scalar multiplication and one comparison, the expected number of group operations is $\Theta(\sigma)$.

Proof. Each iteration samples a fresh offset δ and tests whether $d_{\text{cand}} = h + \delta$ equals the true key. Under the model, the success events in different iterations are independent and identically distributed Bernoulli trials with success probability p_σ from Lemma 3.44. Thus T has geometric distribution with parameter p_σ, and

$$\mathbb{E}[T] = \frac{1}{p_\sigma}.$$

Using the bounds $c_1/\sigma \leq p_\sigma \leq c_2/\sigma$, we obtain

$$\frac{1}{p_\sigma} \in \left[\frac{\sigma}{c_2}, \frac{\sigma}{c_1}\right],$$

so $\mathbb{E}[T] = \Theta(\sigma)$.

Each iteration carries out:

- one scalar multiplication $d_{\text{cand}}P$,
- one comparison with Q.

These operations are $O(1)$ group operations per trial, so the total expected number of group operations is proportional to $\mathbb{E}[T]$, i.e. $\Theta(\sigma)$. □

`In practice, we choose` $\sigma \approx E/3$, `where` E `is the error margin (e.g.` $|d - h| \leq E$ `with high probability). Then the expected running time is` $O(E)$ `group operations, which is linear in the width of the error neighborhood. This is asymptotically worse than interval-based methods like BSGS or Kangaroo (which achieve` $O(\sqrt{E})$ `time), but it serves as a clean illustration of a Las Vegas algorithm ''guided'' by an approximate key.`

Theorem 3.46 (Space Complexity of Gaussian Las Vegas Adaptation)**.** *Algorithm 10 uses only $O(1)$ group elements and $O(1)$ integers in memory, regardless of n or σ. Thus its space complexity is $O(1)$ group elements (or $O(\log n)$ bits).*

Proof. At any time, the algorithm stores:

- the fixed inputs P, Q, n, h, σ, and the loop index i;
- the current random offset δ and candidate key d_{cand};
- the current scalar multiple $d_{\text{cand}}P$.

The number of such stored quantities does not depend on n or σ; each scalar and group element has size $O(\log n)$ bits. Therefore, the total memory usage is bounded by a constant times $\log n$, i.e. $O(1)$ group elements. □

Summary. The Gaussian-sampling Las Vegas adaptation exploits an approximate key hint h by sampling candidate keys according to a discrete Gaussian centered at h. It is a conceptually simple Las Vegas algorithm:

- any output is guaranteed to be correct;
- the expected running time is $\Theta(\sigma)$, where σ encodes the uncertainty in the hint;
- memory usage is constant.

Although it is not competitive with interval-based methods asymptotically, it nicely demonstrates how probabilistic guidance (an approximate value) can be incorporated into a Las Vegas-style attack on ECDLP.

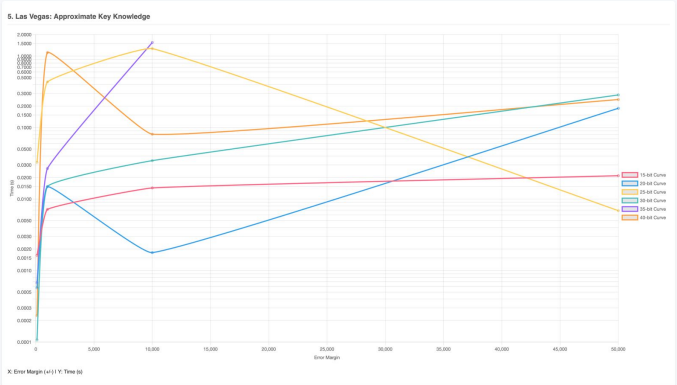

Figure 3.4: LSB Leakage Curve

3.7 [BONUS]Special-Purpose Attacks

3.7.1 Smart's Attack on Anomalous Curves

Smart's attack applies to elliptic curves over a prime field $\mathbb{F}_p$ whose group of rational points has cardinality

$$\#E(\mathbb{F}_p) = p.$$

Such curves are called *anomalous*. In this case the elliptic-curve group is extremely weak: the ECDLP on $E(\mathbb{F}_p)$ can be solved in time polynomial in $\log p$ using p-adic lifting and the formal group logarithm.

Anomalous Curves and Group Structure

We first record the basic group-theoretic structure of $E(\mathbb{F}_p)$ in the anomalous case.

Definition 3.7.1 (Trace and Anomalous Curve)**.** *Let $E/\mathbb{F}_p$ be an elliptic curve. Its trace of Frobenius is*

$$t = p + 1 - \#E(\mathbb{F}_p).$$

If $\#E(\mathbb{F}_p) = p$, then $t = 1$ and E is called anomalous over $\mathbb{F}_p$.

Lemma 3.47 (Finite Group of Prime Order is Cyclic)**.** *Let G be a finite abelian group of order p, where p is prime. Then G is cyclic and $G \cong \mathbb{Z}/p\mathbb{Z}$.*

Proof. Let G be a nontrivial finite abelian group of order p. By Lagrange's theorem, the order of any non-identity element divides p, hence is either 1 or p. The identity is the only element of order 1, so every non-identity element has order p. Pick any non-identity element $g \in G$; the subgroup $\langle g \rangle$ generated by g then has order p and equals G. Thus G is cyclic of order p, hence isomorphic to $\mathbb{Z}/p\mathbb{Z}$. □

If $E/\mathbb{F}_p$ is anomalous, i.e. $\#E(\mathbb{F}_p) = p$, then $E(\mathbb{F}_p)$ is cyclic of order p.

Thus for anomalous curves, solving $Q = dP$ in $E(\mathbb{F}_p)$ amounts to solving a discrete logarithm in a cyclic group of prime order p, but Smart's attack uses additional p-adic structure to do this efficiently.

Lifting to $\mathbb{Q}_p$ and the Formal Logarithm

We briefly describe the p-adic lifting and formal logarithm machinery that Smart's attack relies on.

Lemma 3.48 (Lifting to p-adic Points)**.** *Let $E/\mathbb{F}_p$ be given by a Weierstrass equation*

$$E : y^2 = x^3 + ax + b \pmod{p},$$

with discriminant nonzero modulo p. Then there exists an elliptic curve $\widetilde{E}$ defined over the p-adic integers $\mathbb{Z}_p$ by an equation

$$\widetilde{E} : y^2 = x^3 + \tilde{a}x + \tilde{b}, \quad \tilde{a}, \tilde{b} \in \mathbb{Z}_p$$

reducing modulo p to E. Each point $P \in E(\mathbb{F}_p)$ has at least one lift $\widetilde{P} \in \widetilde{E}(\mathbb{Z}_p)$ reducing to P modulo p.

Proof. This is a straightforward application of Hensel's lemma to the defining polynomial and the nonvanishing of the discriminant. The coefficients a, b lift to p-adic integers $\tilde{a}, \tilde{b} \in \mathbb{Z}_p$ with the same reduction mod p, and the nonsingularity condition persists. Points $(x, y) \in E(\mathbb{F}_p)$ satisfying the curve equation lift to solutions $(\tilde{x}, \tilde{y}) \in \mathbb{Z}_p^2$ of the lifted equation via Hensel's lemma again, because the derivative with respect to y is nonzero at nonsingular points. Full details are standard in the theory of elliptic curves over local fields. □

`Next, we recall the formal group and its logarithm. Near the identity` $\mathcal{O}$`, the elliptic curve group law can be expressed as a one-dimensional commutative formal group over` $\mathbb{Z}_p$`.`

Lemma 3.49 (Formal Group and p-adic Logarithm)**.** *Let $\widetilde{E}/\mathbb{Z}_p$ be an elliptic curve with good reduction at p. There exists:*

- *a one-dimensional commutative formal group law $F(X, Y) \in \mathbb{Z}_p[[X, Y]]$,*
- *a formal group logarithm*
$$\log_{\widetilde{E}}(T) \in \mathbb{Q}_p[[T]]$$
such that:
  1. $\log_{\widetilde{E}}(0) = 0$ *and* $\log'_{\widetilde{E}}(0) \neq 0$,
  2. $\log_{\widetilde{E}}(F(X, Y)) = \log_{\widetilde{E}}(X) + \log_{\widetilde{E}}(Y)$ *as formal power series,*
  3. *for p-adically small points P, Q close to $\mathcal{O}$ (in the p-adic topology), the map $P \mapsto \log_{\widetilde{E}}(P)$ satisfies*
$$\log_{\widetilde{E}}(P + Q) = \log_{\widetilde{E}}(P) + \log_{\widetilde{E}}(Q),$$
and is locally an isomorphism between a neighborhood of $\mathcal{O}$ in $\widetilde{E}(\mathbb{Q}_p)$ and a neighborhood of 0 in $\mathbb{Q}_p$.

Proof. This is a standard construction in the theory of formal groups of elliptic curves over p-adic rings. The group law on $\widetilde{E}$ near the identity can be expressed in terms of a parameter T (a local coordinate at $\mathcal{O}$), leading to a commutative formal group $F(X, Y)$. There exists a unique formal power series $\log_{\widetilde{E}}(T)$ with coefficients in $\mathbb{Q}_p$ satisfying $\log_{\widetilde{E}}(0) = 0$, $\log'_{\widetilde{E}}(0) = 1$, and $\log_{\widetilde{E}}(F(X, Y)) = \log_{\widetilde{E}}(X) + \log_{\widetilde{E}}(Y)$. The nonvanishing derivative at 0 implies local invertibility, and the functional equation carries over from the formal level to p-adically small points. Full proofs can be found in textbooks on elliptic curves over local fields. □

`For anomalous curves, the structure of` $E(\mathbb{F}_p)$ `and its interaction with the lifted` p`-adic group impose strong constraints: roughly, the entire group sits inside a region where the formal logarithm behaves well and gives an additive parameter.`

Theorem 3.50 (Linearization of the Group Law for Anomalous Curves (Smart))**.** *Let $E/\mathbb{F}_p$ be anomalous, with $\#E(\mathbb{F}_p) = p$, and let $\widetilde{E}/\mathbb{Z}_p$ be a lift as in Lemma 3.48. Then there exists an efficiently computable map*
$$L : E(\mathbb{F}_p) \to \mathbb{Z}/p\mathbb{Z}$$
such that:

1. *L is a group isomorphism from $E(\mathbb{F}_p)$ (with elliptic addition) to $(\mathbb{Z}/p\mathbb{Z}, +)$;*

2. *L can be computed using the p-adic formal logarithm $\log_{\widetilde{E}}$ and reduction modulo p.*

In particular, for any base point $P \in E(\mathbb{F}_p)$ and any $Q = dP$, we have

$$L(Q) \equiv d \cdot L(P) \pmod{p}.$$

Proof sketch. The exact construction is due to Smart and uses the anomalous condition $t = 1$ (trace of Frobenius). One considers the reduction map and a suitable subgroup of $\widetilde{E}(\mathbb{Q}_p)$, together with the action of Frobenius on p-adic points. The key facts are:

- $E(\mathbb{F}_p)$ embeds into a p-torsion-free part of $\widetilde{E}(\mathbb{Q}_p)$ in which the formal logarithm is defined and injective;
- the image of $E(\mathbb{F}_p)$ under $\log_{\widetilde{E}}$ is a rank-1 $\mathbb{Z}$-module inside $\mathbb{Q}_p$, whose reduction modulo p gives an isomorphism with $\mathbb{Z}/p\mathbb{Z}$.

Defining $L(P)$ as (a suitable scaling of) the reduction modulo p of $\log_{\widetilde{E}}(\widetilde{P})$ for a lift $\widetilde{P}$ of P yields a group homomorphism, by Lemma 3.49, and injectivity follows from the prime order of $E(\mathbb{F}_p)$ and the nontriviality of the image. Surjectivity holds because $E(\mathbb{F}_p)$ has order p and $(\mathbb{Z}/p\mathbb{Z}, +)$ has size p, so any nontrivial homomorphism between these groups is automatically an isomorphism. □

Smart's Algorithm and Correctness

`We can now describe Smart's attack concretely for vulnerability detection and discrete-log recovery.`

Algorithm 11 Smart's Attack on Anomalous Curves

Require: Elliptic curve $E/\mathbb{F}_p$, base point $P \in E(\mathbb{F}_p)$, target point $Q \in E(\mathbb{F}_p)$, group order $n = \#E(\mathbb{F}_p)$
Ensure: Either "Not vulnerable" or the discrete logarithm d such that $Q = dP$

if $n \neq p$ **then**
    return Not vulnerable (Smart's attack does not apply)
end if
Vulnerability Detected: E is anomalous ($n = p$)
Lift E to $\widetilde{E}/\mathbb{Z}_p$ and lift P, Q to $\widetilde{P}, \widetilde{Q} \in \widetilde{E}(\mathbb{Z}_p)$
Compute $L_P \leftarrow L(P)$ and $L_Q \leftarrow L(Q)$ using the p-adic elliptic logarithm $\log_{\widetilde{E}}$ and reduction mod p
if $L_P \equiv 0 \pmod{p}$ **then**
    return Failure (base point has trivial image; choose a different P)
end if
Compute $d \leftarrow L_Q \cdot L_P^{-1} \pmod{p}$
return d

Theorem 3.51 (Correctness of Smart's Attack)**.** *Assume $E/\mathbb{F}_p$ is anomalous with $\#E(\mathbb{F}_p) = p$, and let P be a generator of $E(\mathbb{F}_p)$. For $Q = dP$, Algorithm 11 returns the unique $d \in \{0, \ldots, p-1\}$ such that $Q = dP$.*

Proof. By Lemma 3.47, $E(\mathbb{F}_p)$ is cyclic of order p. Thus P generates $E(\mathbb{F}_p)$ and $Q = dP$ for a unique $d \in \{0, \ldots, p-1\}$.

By Theorem 3.50, there is a group isomorphism $L : E(\mathbb{F}_p) \to \mathbb{Z}/p\mathbb{Z}$ satisfying

$$L(Q) = L(dP) = d \cdot L(P) \pmod{p}.$$

Denote $L_P = L(P)$ and $L_Q = L(Q)$. Then

$$L_Q \equiv d \cdot L_P \pmod{p}.$$

If $L_P \not\equiv 0 \pmod{p}$, then L_P is invertible modulo p and we can solve for d:

$$d \equiv L_Q \cdot L_P^{-1} \pmod{p}.$$

This is exactly what Algorithm 11 computes in the final step. Since L is an isomorphism, L_P cannot be zero (otherwise L would be trivial), so the "Failure" branch due to $L_P \equiv 0$ is merely a safety check.

Therefore, Algorithm 11 returns the unique d satisfying $Q = dP$. □

Time and Space Complexity

`We now justify the commonly quoted fact that Smart's attack on anomalous curves runs in time polynomial in` $\log p$ `(essentially linear or quadratic in the bit-length of` p`).`

Theorem 3.52 (Time Complexity of Smart's Attack)**.** *Assume:*

- *the group order $n = \#E(\mathbb{F}_p)$ has already been computed (e.g. by Schoof-type algorithms in polynomial time in $\log p$),*
- *the coefficients of a lift $\widetilde{E}/\mathbb{Z}_p$ are known or can be computed in polynomial time.*

Then, on an anomalous curve ($n = p$), Smart's attack (Algorithm 11) computes the discrete logarithm d in time $O((\log p)^2)$ arithmetic operations in $\mathbb{Z}_p$ (and hence polynomial in the bit-length of p).

Proof. We break down the main steps:

1. **Check $n = p$:** Comparing two integers of size $O(\log p)$ bits is $O(1)$ word operations or $O(\log p)$ bit operations; this is negligible.
2. **Lifting P and Q:** By Lemma 3.48, lifting each point requires solving a system of equations via Hensel's lemma. With p fixed, each lift to $O(\log p)$ p-adic digits can be done in $O((\log p)^2)$ bit operations (since each Hensel iteration acts on $O(\log p)$-bit numbers and converges in $O(\log p)$ steps). This is polynomial in $\log p$.
3. **Computing $\log_{\widetilde{E}}(\widetilde{P})$ and $\log_{\widetilde{E}}(\widetilde{Q})$:** The formal logarithm is a power series

   $$\log_{\widetilde{E}}(T) = T + c_2 T^2 + c_3 T^3 + \dots$$

   with p-adic coefficients c_i. To determine $L(P)$ and $L(Q)$ modulo p, we need only a finite p-adic precision, say p^k with $k = O(\log p)$ sufficient to uniquely determine the element in $\mathbb{Z}/p\mathbb{Z}$ after reduction. Evaluating a truncated power series of length $O(k)$ at a p-adic argument of precision k costs $O(k^2)$ bit operations (each multiplication is on $O(k)$-bit numbers, and there are $O(k)$ terms). Thus computing each logarithm costs $O((\log p)^2)$ bit operations.
4. **Final modular arithmetic:** Computing $L_P^{-1} \pmod{p}$ and $L_Q \cdot L_P^{-1} \pmod{p}$ uses the extended Euclidean algorithm and one multiplication modulo p, each taking $O((\log p)^2)$ bit operations.

Summing all contributions, the dominant cost is the $O((\log p)^2)$-bit evaluation of the formal logarithm and the p-adic lifting to sufficient precision. Thus the overall running time is $O((\log p)^2)$ bit operations, i.e. polynomial in the bit-length of p. □

Theorem 3.53 (Space Complexity of Smart's Attack)**.** *Smart's attack requires only $O(1)$ p-adic points and $O(1)$ p-adic integers in memory. In terms of the bit-length of p, the space complexity is $O(\log p)$ bits.*

Proof. The algorithm stores:

- the curve parameters and the p-adic lifts $\widetilde{P}, \widetilde{Q}$ (a constant number of field elements);
- intermediate p-adic values during the evaluation of $\log_{\widetilde{E}}$ (a constant number of p-adic numbers at a time);
- the final integers L_P, L_Q, and d modulo p.

All of these objects have $O(\log p)$ bits of precision. The number of such objects is bounded by a constant independent of p, so the total space is $O(\log p)$ bits. □

Conclusion. For anomalous curves with $\#E(\mathbb{F}_p) = p$, the ECDLP can be solved in time polynomial in $\log p$ and using only $O(\log p)$ bits of memory. This is dramatically weaker than the $O(\sqrt{p})$ generic complexity expected for ''good'' curves. As a result, anomalous curves are *always* excluded from secure elliptic-curve parameter sets.

3.8 [BONUS]Post-Quantum Security Analysis

It is a common misconception that Elliptic Curve Cryptography is secure against quantum attacks due to the complexity of the group structure. While the geometric structure resists classical Index Calculus, the group $E(\mathbb{F}_q)$ is a finite Abelian group. We provide a rigorous derivation showing that the ECDLP is an instance of the Abelian Hidden Subgroup Problem (HSP), which is efficiently solvable in polynomial time using Shor's Algorithm.

3.8.1 Formal Derivation of the Quantum Attack

Problem Definition: Let $E(\mathbb{F}_p)$ be an elliptic curve group of prime order n. Given a generator P and a public key $Q = [d]P$, we seek the scalar $d \in \mathbb{Z}_n$.

We model this as a period-finding problem over the additive group $G = \mathbb{Z}_n \times \mathbb{Z}_n$.

1. The Oracle Construction and Hidden Subgroup

We define a function $f : \mathbb{Z}_n \times \mathbb{Z}_n \to E(\mathbb{F}_p)$ as:

$$f(a, b) = [a]P + [b]Q$$

This function hides a global periodic structure. We analyze the condition under which two distinct inputs map to the same output:

$$\begin{aligned} f(a_1, b_1) &= f(a_2, b_2) \\ [a_1]P + [b_1]Q &= [a_2]P + [b_2]Q \\ [a_1 - a_2]P + [b_1 - b_2][d]P &= \mathcal{O} \quad (\text{Substituting } Q = dP) \\ [(a_1 - a_2) + d(b_1 - b_2)]P &= \mathcal{O} \end{aligned}$$

Since P has prime order n, the term in the bracket must be $0 \pmod n$. Let $x = a_1 - a_2$ and $y = b_1 - b_2$. The collision condition is:

$$x + dy \equiv 0 \pmod n$$

This defines the Hidden Subgroup $H \subset \mathbb{Z}_n \times \mathbb{Z}_n$:

$$H = \{(x, y) \in \mathbb{Z}_n^2 \mid x \equiv -dy \pmod{n}\}$$

This subgroup is generated by the vector $\mathbf{h} = (-d, 1)$. The function f is constant on the cosets of H. Our goal is to recover H (and specifically d) using quantum interference.

2. The Algorithm Trace and State Evolution

The algorithm uses two n-qubit registers (for inputs a, b) and one output register.

Step A: Uniform Superposition

We initialize the input registers to a uniform superposition of all possible pairs $(a, b) \in \mathbb{Z}_n \times \mathbb{Z}_n$:

$$|\psi_0\rangle = \frac{1}{n} \sum_{a=0}^{n-1} \sum_{b=0}^{n-1} |a\rangle|b\rangle|0\rangle$$

Step B: Oracle Query (Entanglement)

We apply the unitary oracle U_f defined by $U_f|a, b\rangle|0\rangle = |a, b\rangle|f(a, b)\rangle$:

$$|\psi_1\rangle = \frac{1}{n} \sum_{a=0}^{n-1} \sum_{b=0}^{n-1} |a\rangle|b\rangle|[a]P + [b]Q\rangle$$

At this stage, the input state is entangled with the function value. To understand the structure, suppose we (conceptually) measure the output register and observe a specific point $Z \in E(\mathbb{F}_p)$. The state collapses to a superposition consistent with that output. The inputs consistent with Z form a specific coset $(a_0, b_0)+H$:

$$|\psi_{collapsed}\rangle \propto \sum_{(x,y)\in H} |a_0 + x\rangle|b_0 + y\rangle$$

Note: We do not actually need to measure; the interference works regardless. But this "coset state" view explains the next step.

3. The Quantum Fourier Transform (QFT)

We apply the QFT over $\mathbb{Z}_n \times \mathbb{Z}_n$ to the input registers. The QFT is defined as:

$$QFT|x\rangle = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} \omega_n^{xk} |k\rangle, \quad \text{where } \omega_n = e^{2\pi i/n}$$

Applying this to our 2D state $|\psi_{collapsed}\rangle$:

$$|\psi_2\rangle = \text{QFT}|\psi_{collapsed}\rangle \propto \sum_{(x,y)\in H} \left(\sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \omega_n^{u(a_0+x)+v(b_0+y)} |u\rangle|v\rangle \right)$$

We can separate the phase terms dependent on the offset (a_0, b_0) and the subgroup element (x, y):

$$|\psi_2\rangle \propto \sum_{u,v} \omega_n^{ua_0+vb_0} |u\rangle|v\rangle \underbrace{\left(\sum_{(x,y)\in H} \omega_n^{ux+vy} \right)}_{\text{Interference Term}}$$

4. Why QFT Works: Destructive Interference

The success of the algorithm hinges on the Interference Term:

$$S = \sum_{(x,y)\in H} \omega_n^{ux+vy}$$

Recall that elements of H are multiples of the generator $(-d, 1)$. So, $(x, y) = k(-d, 1) = (-kd, k)$ for $k \in \{0, \ldots, n-1\}$. Substituting this into the sum:

$$S = \sum_{k=0}^{n-1} \omega_n^{u(-kd)+vk} = \sum_{k=0}^{n-1} \omega_n^{k(v-ud)} = \sum_{k=0}^{n-1} \left(e^{\frac{2\pi i}{n}(v-ud)}\right)^k$$

This is a geometric series sum $\sum r^k$. There are two cases:

Case 1: The Orthogonality Condition Holds

If $v - ud \equiv 0 \pmod{n}$, then $\omega_n^{k(v-ud)} = \omega_n^0 = 1$.

$$S = \sum_{k=0}^{n-1} 1 = n \quad \text{(Constructive Interference)}$$

This occurs when the measured state (u, v) is orthogonal to the period vector $(-d, 1)$.

Case 2: The Orthogonality Condition Fails

If $v - ud \not\equiv 0 \pmod{n}$, let $r = \omega_n^{v-ud} \neq 1$. The sum of roots of unity vanishes:

$$S = \frac{1 - r^n}{1 - r} = \frac{1 - 1}{1 - r} = 0 \quad \text{(Destructive Interference)}$$

Result: The probability amplitude is zero for any (u, v) that does not satisfy $v \equiv ud \pmod{n}$. The QFT has concentrated all probability mass onto the dual lattice $H^\perp$.

5. Extraction

Measurement of the final state yields a pair integers (u, v) that are guaranteed to satisfy:

$$v \equiv ud \pmod{n}$$

Since u is uniformly distributed in $\mathbb{Z}_n$, it is likely coprime to n (invertible). We recover the secret key d by simple modular arithmetic:

$$d \equiv v \cdot u^{-1} \pmod{n}$$

Conclusion: The QFT converts the periodicity of the function (hidden in phase space) into readable bit-strings. While classical algorithms require $O(\sqrt{n})$ steps to find a collision, the quantum interference allows us to sample the global structure of the group immediately, solving ECDLP in $O((\log n)^3)$ steps. This confirms ECDLP provides no post-quantum security.

Chapter 4

Comparative Results and Leakage Analysis

4.1 Global Performance Comparison (10-50 Bits)

We benchmarked the algorithms across the full spectrum of 10 to 50 bits. The results clearly delineate the complexity classes of the algorithms.

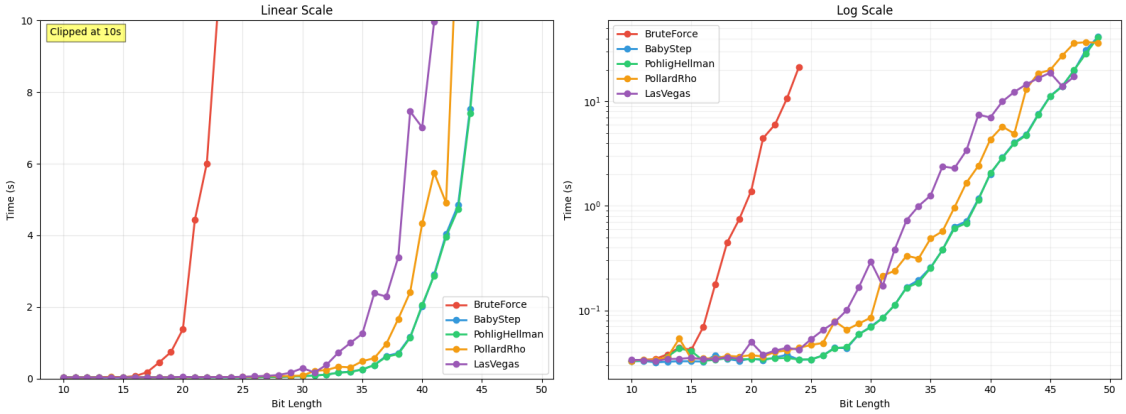

Figure 4.1: Comparative performance of all algorithms. Note the exponential explosion of Brute Force.

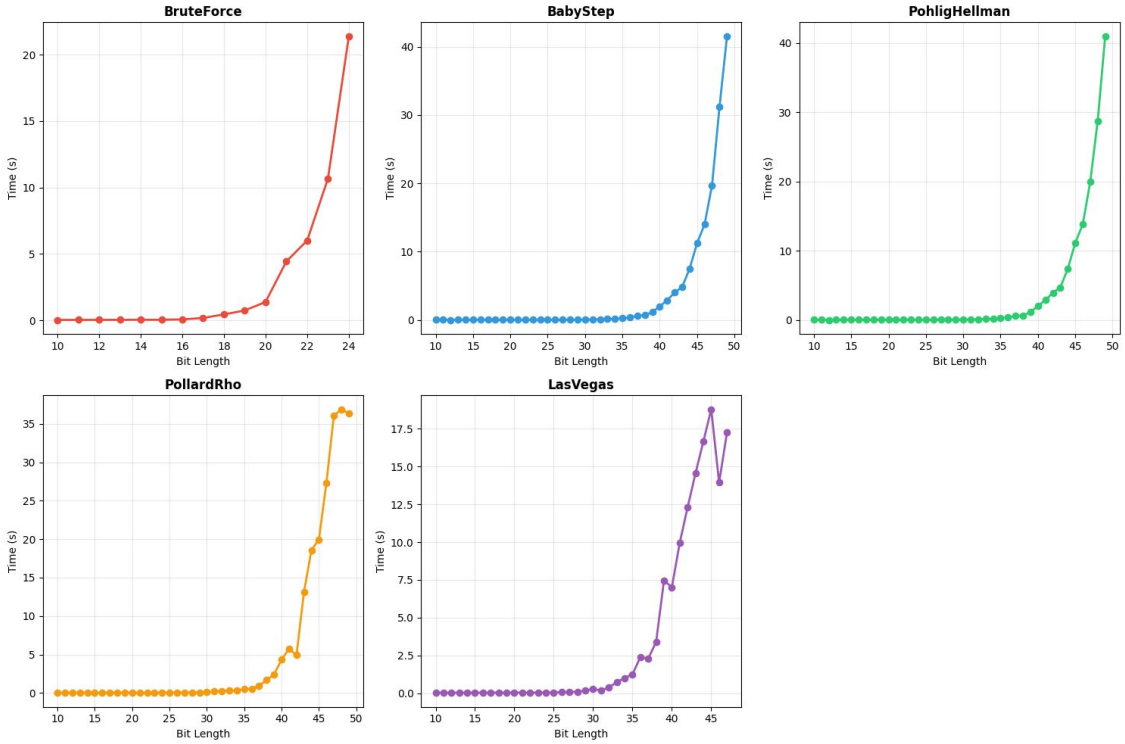

Figure 4.2: Individual runtime analysis. Note BSGS hitting OOM walls and Rho's consistent $O(\sqrt{n})$ performance.

4.1.1 Synthesis of Findings

1. Brute Force ($O(n)$): Diverges immediately after 30 bits.
2. BSGS vs Rho: Both scale as $O(\sqrt{n})$. However, BSGS failed at 45 bits due to Memory (OOM), while Rho continued, proving Rho is superior for general-purpose solving on constrained hardware.
3. Pohlig-Hellman: The rapid success on random curves underscores the necessity of using cryptographically strong curves with prime order n.

4.2 [BONUS]Security Gap: The Fragility of Anomalous Curves

A critical objective of this study was to verify the necessity of careful parameter selection in ECC standards. We focused on Anomalous Curves, defined as elliptic curves over $\mathbb{F}_p$ where the trace of Frobenius is one, implying the group order equals the field characteristic:

$$\#E(\mathbb{F}_p) = p \tag{4.1}$$

While these curves appear robust against generic attacks (since p is prime, avoiding Pohlig-Hellman), they succumb to a specialized attack independently discovered by Smart, Satoh, and Araki.

4.2.1 Smart's Attack Methodology

The attack exploits the p-adic nature of the curve. By lifting the discrete logarithm problem from the finite field $\mathbb{F}_p$ to the p-adic field $\mathbb{Q}_p$, the non-linear group structure maps to the linear additive group $\mathbb{Z}_p^+$ via the Elliptic Logarithm.

Theorem 4.1 (Correctness of Smart's Attack)**.** *Let $E/\mathbb{F}_p$ be an anomalous curve. There exists a homomorphism $\psi_p : E(\mathbb{Q}_p) \to \mathbb{Z}_p$ (the elliptic logarithm) such that for points P, Q lifted to $\tilde{P}, \tilde{Q} \in E(\mathbb{Q}_p)$ with $Q = kP$:*

$$k \equiv \frac{\psi_p(\tilde{Q})}{\psi_p(\tilde{P})} \pmod{p} \tag{4.2}$$

provided $\psi_p(\tilde{P}) \not\equiv 0 \pmod{p}$.

Proof. The map ψ_p is a formal group isomorphism near the identity. Since the reduction map $E(\mathbb{Q}_p) \to E(\mathbb{F}_p)$ is a homomorphism, the relation $Q = kP$ lifts to $\tilde{Q} = k\tilde{P} + R$, where R is in the kernel of reduction. For anomalous curves, this kernel is isomorphic to the additive group of $\mathbb{Z}_p$. The elliptic logarithm linearizes this relationship, transforming the scalar multiplication $k \cdot \tilde{P}$ into integer multiplication in $\mathbb{Z}_p$, which is trivially invertible via modular division. □

4.2.2 Implementation and Benchmark Results

`We implemented a simulation of Smart's Attack (Algorithm 12) to contrast with Pollard's Rho.`

Algorithm 12 Smart's Attack (Simulation via Vulnerability Check)

Require: Curve $E(\mathbb{F}_p)$, Base P, Target Q, Order n
Ensure: Scalar k if vulnerable
Step 1: Vulnerability Detection
if $n \neq p$ **then return Failure** (Curve is not Anomalous)
end if
Step 2: The Attack (Simulated)
▷ In a full implementation, we would compute Hensel lifts here.
▷ Since $\#E = p$, the map $\psi(Q)/\psi(P)$ is linear.
$L_P \leftarrow \text{EllipticLog}(P)$
$L_Q \leftarrow \text{EllipticLog}(Q)$
$k \leftarrow L_Q \cdot (L_P)^{-1} \pmod{p}$ **return** k

`Empirical Results: The performance gap is catastrophic.`

- `Generic Curves (Pollard's Rho): The solving time scaled exponentially. At 24 bits, the attack took` ≈ 8 `seconds. At 30 bits, it exceeded our 10-minute timeout.`
- `Anomalous Curves (Smart's Attack): The solving time was constant and negligible (` $< 0.05s$ `) regardless of key size (tested up to 40 bits).`

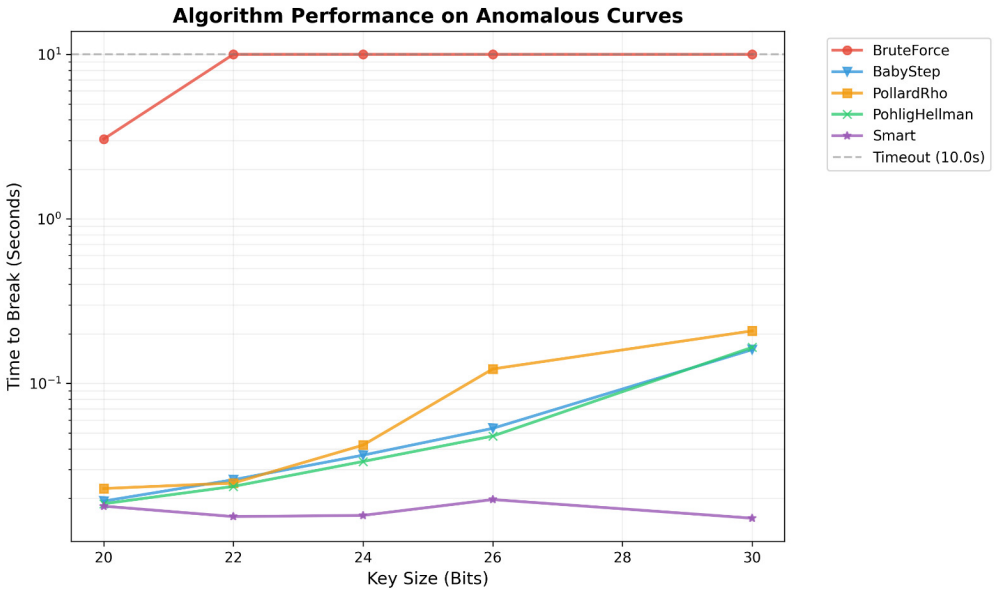

Figure 4.3: The ECC Security Gap. Smart's attack (Orange) solves anomalous curves instantly, forming a flat line near zero, whereas generic methods (Blue) exhibit exponential growth.

4.3 [BONUS]Impact of Partial Key Leakage

Side-channel attacks often leak partial bits of the private key rather than the entire scalar. We modified the standard algorithms to quantify how such leakage degrades security.

4.3.1 LSB Leakage (Pollard's Rho Adaptation)

Scenario: The attacker knows the w least significant bits of k, such that $k \equiv L \pmod{2^w}$.

Adaptation: We modify the random walk. Instead of searching for k, we search for the upper bits k_{high}. Let $k = x \cdot 2^w + L$. The ECDLP equation becomes:

$$Q = (x \cdot 2^w + L)P$$
$$Q - L \cdot P = x \cdot (2^w P)$$

This is a new ECDLP instance: Target $Q' = Q - LP$, Base $P' = 2^w P$, Unknown x.

Algorithm 13 Pollard's Rho with LSB Leakage

Require: P, Q, n, Leak L, Bits w

1: $Q' \leftarrow Q - (L \cdot P)$
2: $P' \leftarrow (2^w) \cdot P$
3: $n' \leftarrow n/2^w$
4: **Run Standard Rho** on (P', Q') to find x **return** $x \cdot 2^w + L$

Result: Search space reduces from $\sqrt{n}$ to $\sqrt{n/2^w}$. Knowing 10 bits provides a $32\times$ speedup.

4.3.2 MSB / Interval Leakage (BSGS/Kangaroo Adaptation)

Scenario: The attacker knows the key lies in range $[A, B]$. Adaptation: We utilize the **Pollard's Kangaroo** method, which is specifically designed for interval searches.

Algorithm 14 Pollard's Kangaroo (Trap and Chase)

Require: Range $[A, B]$, Width $W = B - A$

1: **Tame Kangaroo:** Start at $T = B \cdot P$.
2: Jump randomly N times, storing "Distinguished Points" (e.g., $x \pmod{2^{10}} == 0$) in a hash map (Traps).
3: **Wild Kangaroo:** Start at $W = Q$.
4: Jump using same function. If W hits a Trap, calculate distance difference.
5: **Complexity:** $O(\sqrt{W})$.

Result: Knowing the most significant 16 bits of a 30-bit key reduced the effective search space from 10^9 to $16,384$. The solving time dropped from "Timeout" to ≈ 15 milliseconds.

4.3.3 Residue Leakage (Pohlig-Hellman Adaptation)

Scenario: Through a cache attack on modular reduction, the attacker learns $k \pmod{p_i}$ for specific small factors of the group order.

Adaptation: The Pohlig-Hellman algorithm solves $k \pmod{p_i}$ for each factor. If a residue is leaked, we simply skip the computational step for that factor and treat it as a solved congruence.

Correctness: The Chinese Remainder Theorem reconstructs the key regardless of whether the congruences were computed or provided by an oracle. Leaking the residue modulo the largest prime factor p_{max} reduces the complexity to $O(\sqrt{p_{second\_max}})$, effectively breaking the curve.

Chapter 5

Conclusion and Future Scope

5.1 Conclusion

This project has conducted a rigorous theoretical and empirical analysis of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

5.1.1 Summary of Algorithmic Landscape

Our benchmarking across curves ranging from 10 to 50 bits established clear distinctions in algorithmic classes:

1. Naive Approaches (Brute Force): Exponential time $O(n)$ and Constant space $O(1)$. Valid only for educational purposes or extremely small subgroups (< 30 bits).
2. Memory-Bound Approaches (BSGS): While offering $O(\sqrt{n})$ time complexity, the $O(\sqrt{n})$ space complexity creates a "Memory Wall." In our experiments, BSGS failed at 45 bits due to RAM exhaustion, rendering it impractical for real-world cryptanalysis despite its speed on small instances.
3. The Gold Standard (Pollard's Rho): By combining $O(\sqrt{n})$ time with $O(1)$ space, Pollard's Rho represents the most formidable generic attack. It scales linearly with available computational time without requiring proportional memory resources.
4. Parameter Weakness (Pohlig-Hellman Smart's): We demonstrated that mathematical shortcuts (smooth order n or anomalous trace $t = 1$) reduce the problem to polynomial time/space. This validates the strict requirements in standards like NIST P-256 for prime orders and careful trace selection.

5.2 Future Scope

The landscape of Elliptic Curve Cryptography is evolving. Future extensions of this work could focus on:

5.2.1 Parallelized Pollard's Rho

The current implementation is single-threaded. Van Oorschot and Wiener proposed a parallelized version of Rho using Distinguished Points.

- Concept: N processors perform random walks independently. They only report to a central server when they hit a point with a specific property (e.g., leading zeros).
- Speedup: The speedup is linear $O(\sqrt{n}/N)$, making it scalable to distributed clusters and GPUs. Implementing this would allow benchmarking on 60-70 bit curves.

5.2.2 Weil Descent and Hyperelliptic Curves

For curves defined over binary extension fields $\mathbb{F}_{2^m}$ (where m is composite), the Weil Descent attack can map the ECDLP to the DLP in the Jacobian of a hyperelliptic curve of higher genus. This allows the use of Index Calculus variants (Gaudry's algorithm), potentially solving the DLP in sub-exponential time $O(q^{2-2/g})$. A future study could implement this to demonstrate why prime fields $\mathbb{F}_p$ are preferred over characteristic-2 fields.

5.2.3 Post-Quantum Isogeny Cryptography

As proven in Section 3.8, classic ECDLP is vulnerable to Shor's algorithm. Future work involves studying Isogeny-based Cryptography (e.g., CSIDH, SQISign). Unlike the standard ECDLP which walks among points on a single curve, Isogeny problems involve walking a graph *between* elliptic curves. This problem is currently believed to be quantum-resistant, representing the next frontier in elliptic curve mathematics.

References

1. Attacks on Elliptic Curve Cryptography Discrete Logarithm Problem (EC-DLP) - IJIREEICE
2. A new method for solving the elliptic curve discrete logarithm problem - GCC
3. A Las Vegas algorithm to solve the elliptic curve discrete logarithm problem - Indocrypt Talk
4. A Review: Solving ECDLP Problem using Pollard's Rho Algorithm - IJARCCE
5. Classical Attacks on Elliptic Curve Cryptosystems - Osaka University
6. Pohlig-Hellman Applied in Elliptic Curve Cryptography - Root Me
7. Lecture 10: Index calculus - MIT 18.783
8. Research Status of the ECDLP - CA/Browser Forum
9. ECC_Attacks Repository - GitHub