Vietnam National University - Ho Chi Minh City
University of Information Technology
Faculty of Computer Networks and Communications

FINAL REPORT

Cryptanalysis of ECC-based Algorithms

Authors: Phan Nguyen Viet Bac - 23520087
Tran Gia Bao - 23520139
Chau Hoang Phuc - 23521191

Class: NT219.P22.ANTTT
Lecturer: Nguyen Ngoc Tu
Course: Cryptography

Ho Chi Minh City, 2025

Contents

I Overview

Elliptic Curve Cryptography (ECC) is a public-key cryptographic scheme that relies on the algebraic structure of elliptic curves over finite fields. Compared to traditional algorithms such as RSA, ECC offers equivalent levels of security while requiring significantly shorter key lengths [1]. This characteristic enables faster computation, reduced storage requirements, and lower power consumption, making ECC particularly well-suited for resource-constrained environments such as mobile devices and embedded systems. As a result, ECC has been widely adopted in secure communication protocols, digital signatures, and blockchain-based technologies [2, 3].

Recognizing the growing importance of ECC, the U.S. National Institute of Standards and Technology (NIST) has standardized its use for digital signatures in FIPS 186 [4] and for key establishment schemes in SP 800-56A [5]. FIPS 186-4 recommends fifteen elliptic curves with varying security levels. However, since these curves were introduced more than fifteen years ago, advancements in cryptographic research have prompted the development of newer elliptic curves that offer improved security margins, performance benefits, and ease of secure implementation [6].

In response to these developments, NIST organized the 2015 Workshop on Elliptic Curve Cryptography Standards, where stakeholders expressed strong interest in modernizing ECC standards [7]. Subsequently, NIST solicited public feedback on potential revisions to FIPS 186-4, including the inclusion of new elliptic curves and digital signature schemes. Throughout 2016, efforts were made to review these inputs and revise the standards accordingly.

II Mathematical Background

An elliptic curve in Weierstrass form is given by the equation:

$$y^2 = x^3 + Ax + B, \quad \text{where } 4A^3 + 27B^2 \neq 0$$

An elliptic curve (EC) is defined over a field F when $A, B \in F$ [8]. The condition $4A^3 + 27B^2 \neq 0$ ensures the curve is non-singular.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on an elliptic curve E. Then their sum $P_3 = (x_3, y_3) = P_1 + P_2$ is calculated as follows:

$$P_3 = \begin{cases} (m^2 - x_1 - x_2,\, m(x_1 - x_3) - y_1), & \text{if } P_1 \neq P_2, \text{ where } m = \frac{y_2 - y_1}{x_2 - x_1} \\ (m^2 - 2x_1,\, m(x_1 - x_3) - y_1), & \text{if } P_1 = P_2 \text{ and } y_1 \neq 0, \text{ where } m = \frac{3x_1^2 + A}{2y_1} \\ \mathcal{O}, & \text{if } x_1 = x_2 \text{ and } y_1 = -y_2 \pmod{p} \\ \mathcal{O}, & \text{if } P_1 = P_2 \text{ and } y_1 = 0 \\ P_1, & \text{if } P_2 = \mathcal{O} \\ P_2, & \text{if } P_1 = \mathcal{O} \end{cases}$$

Point scalar multiplication is defined as repeated addition:

$$nP = \underbrace{P + P + \cdots + P}_{n \text{ times}}$$

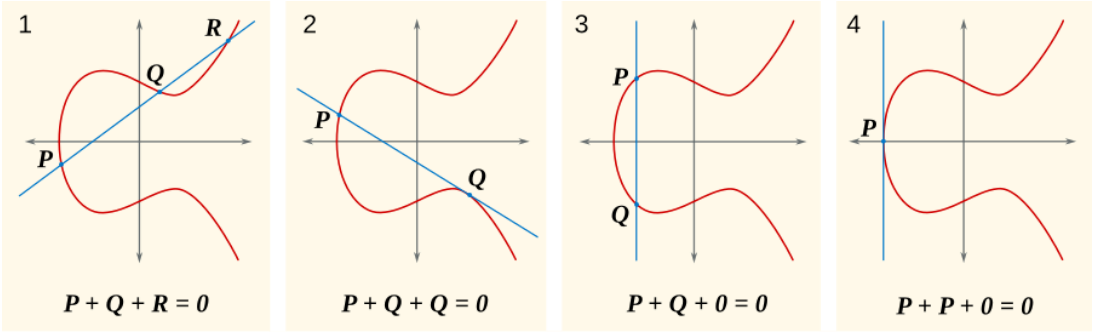

Figure 1: Addition and Multiplication on an Elliptic Curve [1]

II.1 Order

In elliptic curve theory, the **order** of an elliptic curve E over a finite field $\mathbb{F}_p$ (where p is a prime number) is the total number of points on the curve, including the *point at infinity* $\mathcal{O}$. It is denoted as:

$$\#E(\mathbb{F}_p)$$

Computing the order of an elliptic curve is essential in cryptographic applications. The **Hasse Theorem** provides a bound for this value:

$$|\#E(\mathbb{F}_p) - (p+1)| \leq 2\sqrt{p}$$

Therefore, we have:

$$\#E(\mathbb{F}_p) \in [p+1-2\sqrt{p},\ p+1+2\sqrt{p}]$$

Advanced algorithms such as **Schoof's algorithm** [9] and its variants (e.g., Schoof–Elkies–Atkin algorithm) are used to compute the exact number of points efficiently.

II.2 Subgroup Order

A **subgroup** of an elliptic curve group over $\mathbb{F}_p$ is a set of points that is closed under the elliptic curve group operation (addition and scalar multiplication).

Given a point P on the curve, the subgroup generated by P is defined as:

$$\langle P \rangle = \{\mathcal{O}, P, 2P, 3P, \ldots, (n-1)P\}$$

where n is the smallest positive integer such that:

$$nP = \mathcal{O}$$

Then, n is called the **order of the point** P. By **Lagrange's Theorem**, the order of any subgroup divides the order of the full elliptic curve group:

$$\text{ord}(P) \mid \#E(\mathbb{F}_p)$$

If $\#E(\mathbb{F}_p)$ is a prime number (or n is a large prime factor of $\#E(\mathbb{F}_p)$ and P generates a subgroup of this prime order), then this forms a secure basis for cryptographic applications [8].

II.3 Finding a Base Point

In **Elliptic Curve Cryptography (ECC)**, a **base point** P (also known as a generator point) is used to generate public-private key pairs. A good base point must satisfy [10]:

- $P \in E(\mathbb{F}_p)$ — the point lies on the curve.
- $\text{ord}(P)$ is a large prime number.
- $kP \neq \mathcal{O}$ for any small positive integer k (this is implied if $\text{ord}(P)$ is prime and large).

This ensures resistance against small subgroup attacks and improves the overall security of ECC systems. Curves used in practice are selected such that their order is either a large prime or has a large prime factor [11].

II.4 Discrete Logarithm Problem

The security of ECC relies on the hardness of the **Elliptic Curve Discrete Logarithm Problem (ECDLP)** [12, 13]. Given two points P and Q on the curve such that:

$$Q = kP$$

for some unknown scalar k, the ECDLP is the problem of finding k, given only P and Q.

This problem is believed to be computationally infeasible to solve efficiently for properly chosen curves and parameters. It is the foundation of widely used cryptographic protocols, such as:

- ECDSA — Elliptic Curve Digital Signature Algorithm [4, 14]
- ECDH — Elliptic Curve Diffie–Hellman Key Exchange [5]
- ECIES — Elliptic Curve Integrated Encryption Scheme

III Elliptic Curves In The Context Of Cryptography

Elliptic Curve Cryptography (ECC) translates the mathematical properties of **elliptic curves** into cryptographic applications, yielding powerful public-key encryption schemes. The security foundation of ECC lies in the computational **difficulty of the Elliptic Curve Discrete Logarithm Problem (ECDLP)**. Given a point P on the curve and a point $Q = kP$ (where k is an integer), finding k (the discrete logarithm of Q to the base P) is considered computationally infeasible when the curve parameters are chosen correctly. This inherent difficulty forms the basis for the security of ECC algorithms.

In the context of cryptography, the primary applications of elliptic curves include:

1. **Key Exchange**: Enabling two parties to establish a shared secret key over an insecure public channel.
2. **Digital Signatures**: Providing a means to verify the integrity and non-repudiation of data.

Key components of an ECC system typically involve:

- **Elliptic Curve (E)**: Defined over a **finite field** $\mathbb{F}_p$ (or $\mathbb{F}_{2^m}$), along with parameters A and B.
- **Base Point (G)**: A point on the curve with a large order, serving as the "generator" for cryptographic operations.
- **Group Order (n)**: The total number of points on the curve (including the point at infinity), or the order of the subgroup generated by the base point G. This n is often a large prime number to ensure security.
- **Private Key (d)**: A randomly chosen secret integer d from the range $[1, n-1]$.
- **Public Key (Q)**: Computed by multiplying the base point G by the private key d, i.e., $Q = dG$.

III.1 Elliptic Curve Diffie-Hellman (ECDH)

ECDH is a variant of the Diffie-Hellman key exchange protocol, allowing two parties (Alice and Bob) to establish a shared secret key over an insecure channel. The security of ECDH relies on the hardness of the ECDLP.

How it works:

- **Parameter Setup:** Alice and Bob agree on an **elliptic curve** E, a **finite field** $\mathbb{F}_p$, and a **base point** G with order n.
- **Alice:**
  - Chooses a random private key $d_A \in [1, n-1]$.
  - Computes her public key $Q_A = d_A G$.
  - Sends Q_A to Bob.
- **Bob:**
  - Chooses a random private key $d_B \in [1, n-1]$.
  - Computes his public key $Q_B = d_B G$.
  - Sends Q_B to Alice.
- **Shared Secret Computation:**
  - Alice computes $K = d_A Q_B = d_A(d_B G) = (d_A d_B)G$.
  - Bob computes $K = d_B Q_A = d_B(d_A G) = (d_B d_A)G$.

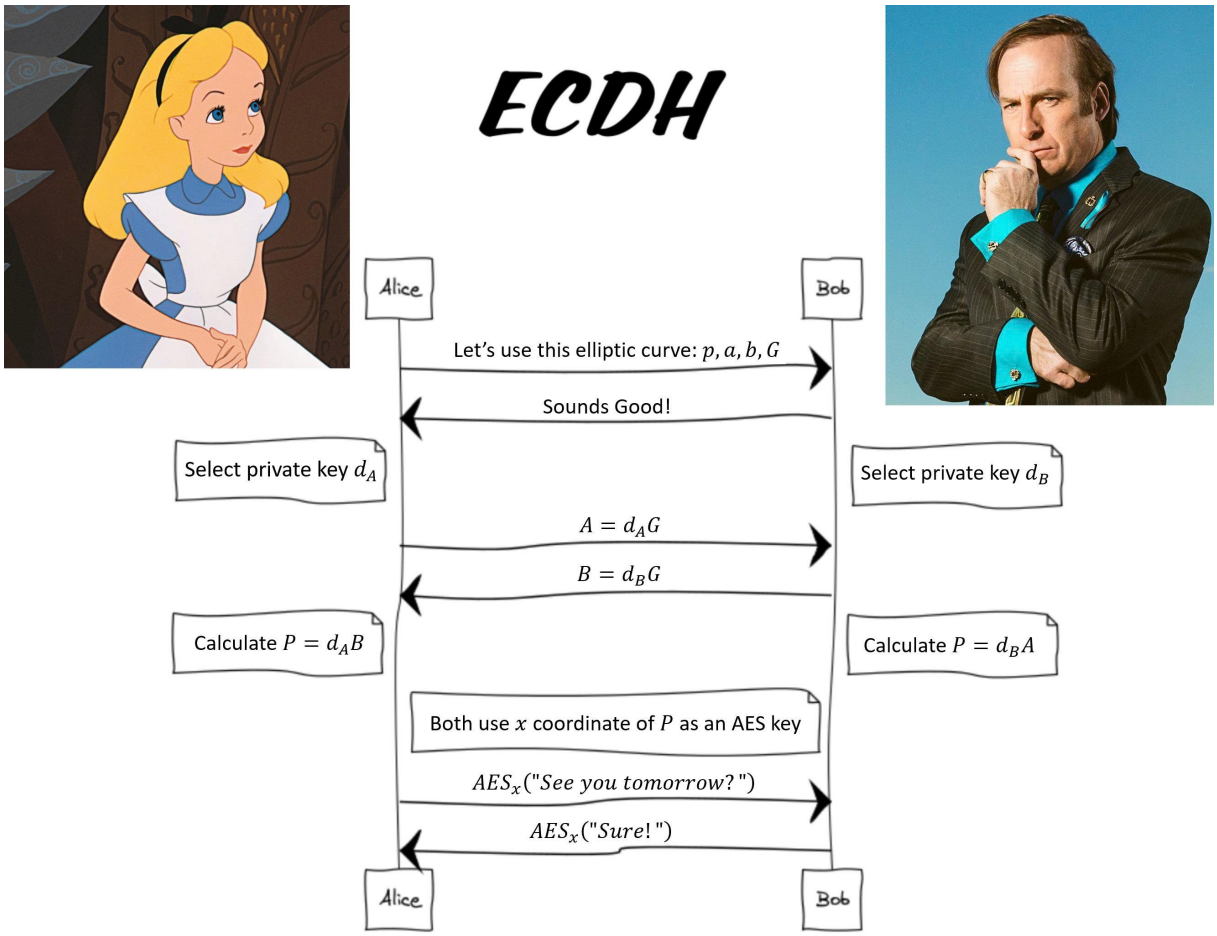

Figure 2: ECDH Flowchart

III.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is a widely used digital signature algorithm that verifies the authenticity and integrity of messages. It's a variant of the Digital Signature Algorithm (DSA) that employs elliptic curve operations.

How it works:

- **Parameter Setup:** The signer and verifier agree on an **elliptic curve** E, a **finite field** $\mathbb{F}_p$, and a **base point** G with order n. The signer holds a private key (d_A) and a corresponding public key ($Q_A = d_A G$).
- **Signature Generation** (r, s) **for message** m**:**
  1. Compute $e = \text{HASH}(m)$ (the message hash).
  2. Choose a cryptographically secure random integer $k \in [1, n-1]$ (the **nonce**). This value is critically important and *must be unique and random* for each signature.
  3. Compute the elliptic curve point $P = kG = (x_1, y_1)$.
  4. Compute $r = x_1 \bmod n$. If $r = 0$, go back to step 2.
  5. Compute $s = k^{-1}(e + r \cdot d_A) \bmod n$. If $s = 0$, go back to step 2.
  6. The signature is the pair (r, s).
- **Signature Verification** (r, s) **for message** m**:**
  1. Compute $e = \text{HASH}(m)$.
  2. Check if r and s are within the range $[1, n-1]$. If not, the signature is invalid.
  3. Compute $w = s^{-1} \bmod n$.
  4. Compute $u_1 = e \cdot w \bmod n$.
  5. Compute $u_2 = r \cdot w \bmod n$.
  6. Compute the point $P' = u_1 G + u_2 Q_A$.
  7. If P' is the point at infinity ($\mathcal{O}$), the signature is invalid.
  8. Compute $r' = x_{P'} \bmod n$ (the x-coordinate of point P').
  9. The signature is valid if $r' = r$.

Both ECDH and ECDSA effectively leverage the high security offered by ECC with significantly smaller key sizes compared to algorithms based on integer factorization (like RSA) or discrete logarithm problems in finite fields. However, as Part IV will detail, careless implementation or the use of non-standard parameters can lead to severe vulnerabilities.

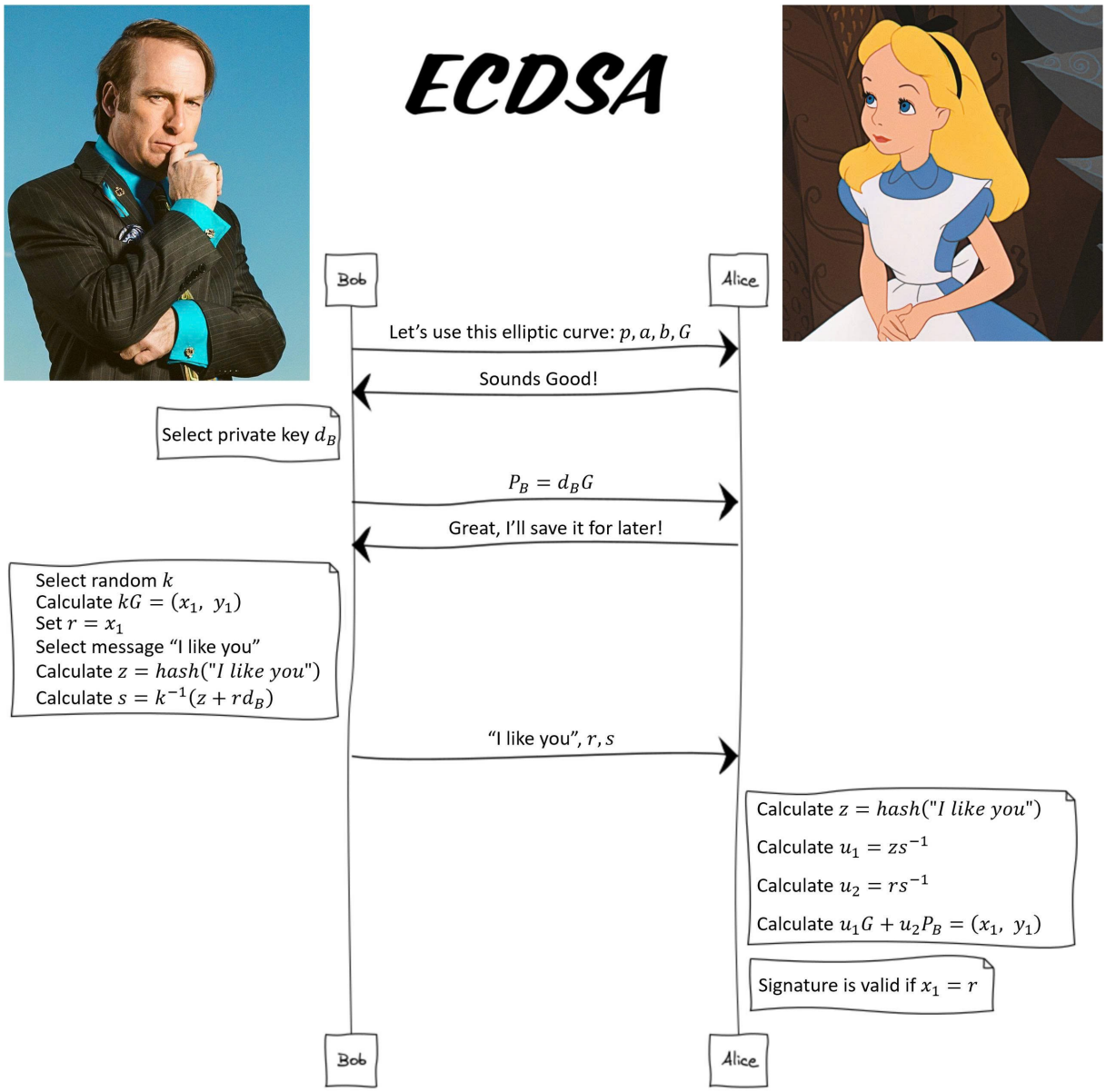

Figure 3: ECDSA Flowchart

IV Elliptic Curves Attack and Threat Analysis

IV.1 Requirements

- SageMath: open-source mathematics software useful for ECC operations.
- Python 3.x: a high-level programming language.
- PyCryptodome: a Python library for cryptographic algorithms.

IV.2 ECDH Attacks

IV.2.1 Small Order Curve

When the generator point P has a small order $n = \text{ord}(P)$, the ECDLP can be solved relatively easily. If an attacker can force the key exchange to use a point P of small order n, they can recover the secret key with complexity related to $O(\sqrt{n})$ using algorithms like Baby-Step Giant-Step [15], or even faster for very small n.

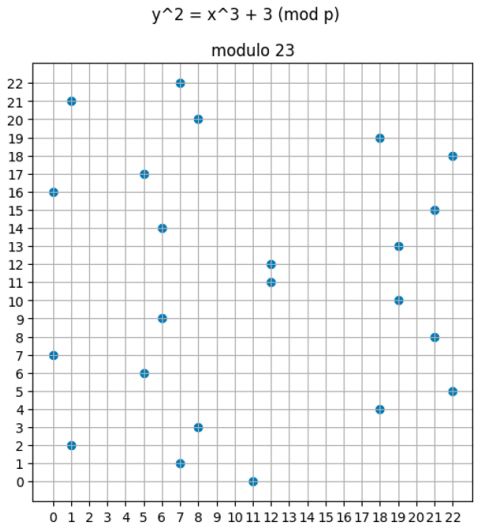

Figure 4: Small order curve illustration

Baby-step Giant-step (BSGS): [15]

1. Let $n = \text{ord}(P)$, and $m = \lceil \sqrt{n} \rceil$.
2. **Baby steps**: Compute and store jP for $j = 0$ to $m - 1$.
3. **Giant steps**: Compute $Q - imP$ for $i = 0$ to $m - 1$.
4. Find i, j such that $Q - imP = jP$; then $k = (im + j) \pmod{n}$.

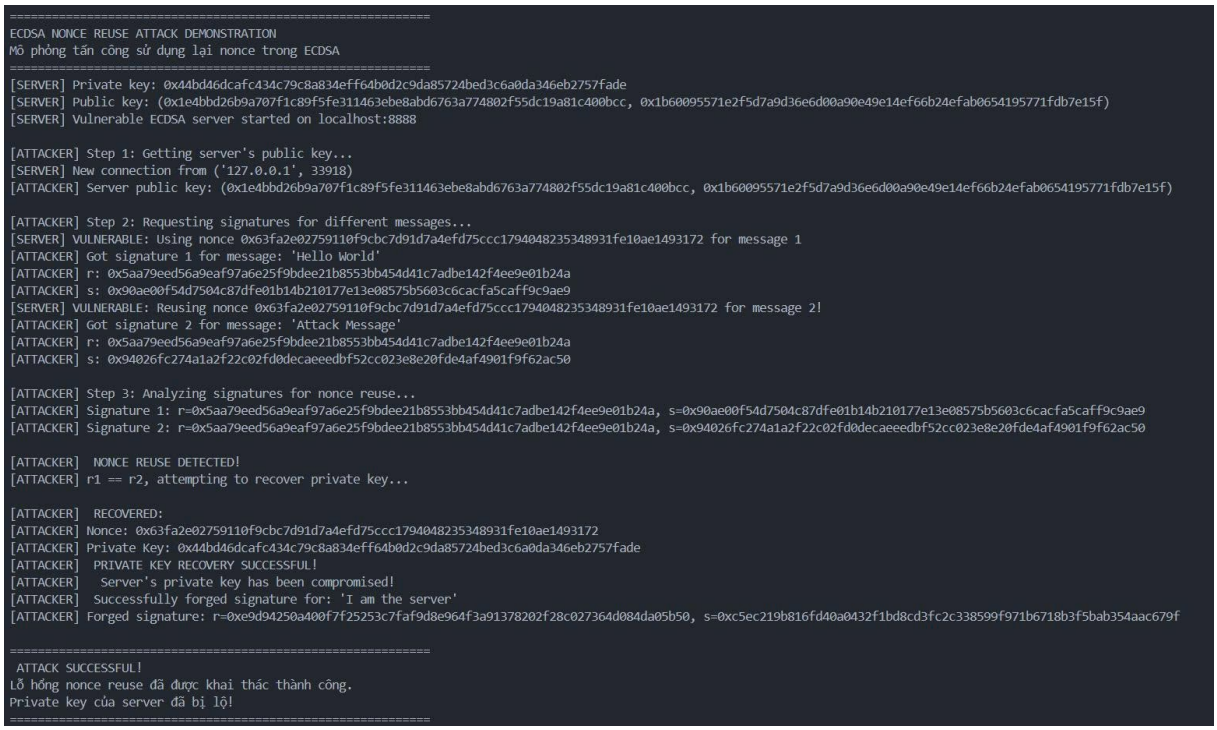

Figure 5: Demo Baby-step Giant-step (BSGS)

Pollard's Rho: [16]

1. Initialize a random point $R_0 = a_0P + b_0Q$.
2. Generate a pseudorandom sequence $R_{i+1} = f(R_i)$, where f is a function that maps points to points, and keep track of coefficients a_i, b_i such that $R_i = a_iP + b_iQ$.
3. Detect a collision $R_i = R_j$ for $i \neq j$.
4. This implies $(a_i - a_j)P + (b_i - b_j)Q = \mathcal{O}$. If $b_i - b_j \not\equiv 0 \pmod{n}$, then $Q = -\frac{a_i - a_j}{b_i - b_j}P$. So $k \equiv -\frac{a_i - a_j}{b_i - b_j} \pmod{n}$.

Smooth Order Curve

If the order n of the generator point P is a composite number whose prime factors are all small (i.e., n is smooth), then the ECDLP can be solved efficiently using the Pohlig-Hellman algorithm [17].

Pohlig-Hellman Algorithm [17]

Let $Q = kP$, $n = \text{ord}(P)$, and the prime factorization of n be:

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

The algorithm works by finding $k \pmod{p_i^{e_i}}$ for each i, and then combining these results using the Chinese Remainder Theorem (CRT) to find $k \pmod{n}$.

1. For each prime power factor $p_i^{e_i}$ of n:

(a) Let $n' = n/p_i^{e_i}$. Compute $P' = n'P$ and $Q' = n'Q$.

(b) The order of P' is $p_i^{e_i}$. We need to solve $k_i P' = Q'$, where $k \equiv k_i \pmod{p_i^{e_i}}$.

(c) To find $k_i \pmod{p_i^{e_i}}$, we can write $k_i = c_0 + c_1 p_i + \cdots + c_{e_i-1} p_i^{e_i-1}$, where $0 \le c_j < p_i$.

(d) Solve for c_j iteratively. For c_0, solve $c_0(p_i^{e_i-1}P') = (p_i^{e_i-1}Q')$. This is a DLP in a group of order p_i, solvable by BSGS or Pollard's Rho if p_i is small.

(e) Repeat for $c_1, c_2, \ldots$ by adjusting Q' at each step.

2. Use CRT to combine the congruences $k \equiv k_i \pmod{p_i^{e_i}}$ to find $k \pmod{n}$.

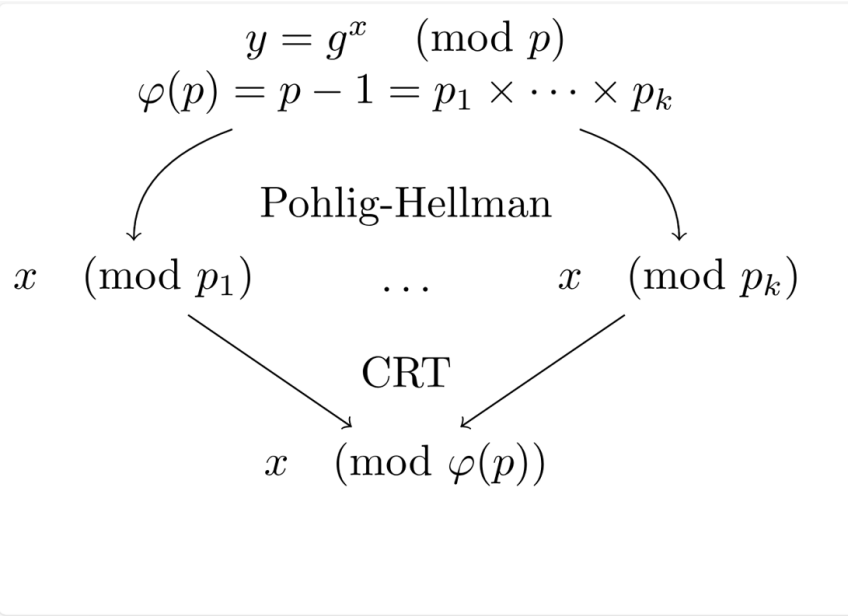

Figure 6: Pohlig-Hellman strategy on a smooth-order group

The complexity is dominated by solving the DLPs in subgroups of order p_i, which is roughly $O(\sum \sqrt{p_i})$ if BSGS is used for each subproblem, or $O(\sqrt{p_{\max}})$, where $p_{\max}$ is the largest prime factor of n. For example, if n is a 128-bit number with its largest prime factor being 30 bits, the problem's complexity can drop from roughly 2^{64} (for general ECDLP) to around 2^{15}, making the attack feasible.

```
================================================
Curve defined on GF(115792089210356248762697446949407573530086143415290314195533631308867097853951)
Generator G: (58739589111611962715835544993606157139975695246024583862682820878769866632269, 868570398378907381588006562837391004190836985747239187551070566336206338977772)
Order of G: 115792089210356248762697446949407573529834065819575831761743834748271926193 40
Factorization of order: 2^2 * 5 * 7 * 13 * 617 * 4759 * 8260807 * 144722299 * 928070153503 * 2590206143359 * 13389517011401 * 56307812771039
================================================
The order is a 'smooth number', making Pohlig-Hellman attack feasible.
================================================
```

Figure 7: Server exposes a weak curve with a smooth order

```
================================================
Curve defined on GF(115792089210356248762697446949407573530086143415290314195533631308867097853951)
Generator G: (58739589111611962715835544993606157139975695246024583862682820878769866632269, 868570398378907381588006562837391004190836985747239187551070566336206338977772)
Order of G: 115792089210356248762697446949407573529834065819575831761743834748271926193 40
Factorization of order: 2^2 * 5 * 7 * 13 * 617 * 4759 * 8260807 * 144722299 * 928070153503 * 2590206143359 * 13389517011401 * 56307812771039
================================================
The order is a 'smooth number', making Pohlig-Hellman attack feasible.
================================================
Server listening on 127.0.0.1:65432
Connected by ('127.0.0.1', 55342)
Generated secret: 39447804672313090851699735024383166242060310590129185117016291747475196939 53
Public key Q = n*G: (113086381761596750094230910105668447133470152323125613287928113076833359010 07, 446884547833584273636156042101311036618875096176556935841060677806703940908 35)
```

Figure 8: The server accepts the client's connection and generates a new secret/public key pair (n, Q) for the challenge

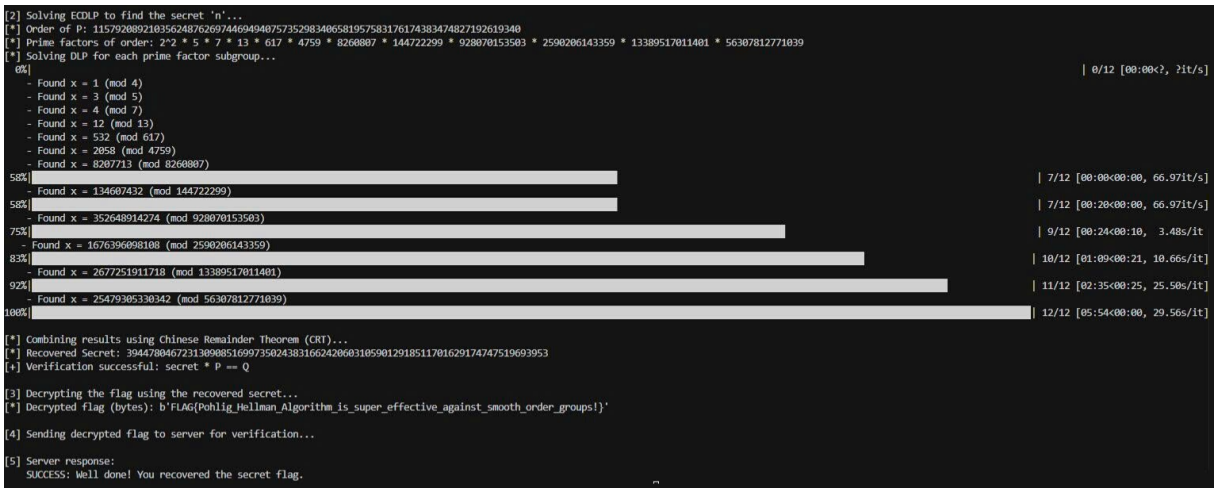

Figure 9: The client executes the Pohlig-Hellman attack: it solves the DLP for each small prime factor of the order, then uses the Chinese Remainder Theorem (CRT) to recover the full secret key, decrypts the flag, and receives a success confirmation from the server

IV.2.2 Not Smooth Order & Small Private Key

Suppose the generator's order n factors into primes, but the largest prime factor is too large to solve the ECDLP practically using Pohlig-Hellman—for example, a 256-bit order n with a 128-bit largest prime factor $p_{\max}$. In this case, Pohlig-Hellman would require about $O(\sqrt{p_{\max}}) \approx O(2^{64})$ operations, which is infeasible.

However, if the private key k is known to be small—say, $k < K_{\text{bound}}$ where K_{bound} is significantly smaller than n (e.g., k is 64 bits instead of 256 bits)—the attack can still succeed. Even though the group has approximately 2^{256} points, the public key $Q = kP$ lies within the first K_{bound} multiples of P.

An attacker can try to find $k \pmod m$ where m is a product of small prime factors of n such that $m > K_{\text{bound}}$. Then, one can simply iterate through possible values of k or use BSGS/Pollard's Rho restricted to the range $[0, K_{\text{bound}} - 1]$ [16, 15]. More effectively, if k is small, one can run BSGS or Pollard's Rho directly to find k, with complexity $O(\sqrt{K_{\text{bound}}})$. This does not directly use the Pohlig-Hellman structure but attacks the small key assumption.

The text seems to imply a hybrid approach: run Pohlig-Hellman on the smooth part of n. If $k \pmod{m_{\text{smooth}}} = k_0$ is found, and m_{smooth} covers enough bits of k (e.g., if $K_{\text{bound}} < m_{\text{smooth}}$), then $k = k_0$. If K_{bound} is larger, the search space for k is reduced.

IV.2.3 Invalid Curve Attack

An invalid curve attack occurs when a party (e.g., a server) accepts a point from another party (e.g., a client or an attacker) and performs ECC operations without verifying that the point actually lies on the agreed-upon elliptic curve $E : y^2 = x^3 + Ax + B$ over $\mathbb{F}_p$. The formulas for point addition and doubling involve the curve parameter A, but not B directly (unless specific conditions like $y_1 = 0$ for doubling are met, or in the verification $y^2 = x^3 + Ax + B$).

Doubling	**Addition** ($P_1 \neq P_2$)
If $y_1 = 0$ then $2P_1 = \mathcal{O}$, else $\lambda = \frac{3x_1^2+A}{2y_1}$ $x_3 = \lambda^2 - 2x_1$ $y_3 = \lambda(x_1 - x_3) - y_1$	If $x_1 = x_2$ (so $y_1 = -y_2$), $P_1 + P_2 = \mathcal{O}$. Else, $\lambda = \frac{y_2-y_1}{x_2-x_1}$ $x_3 = \lambda^2 - x_1 - x_2$ $y_3 = \lambda(x_1 - x_3) - y_1$

Table 1. Doubling and Addition laws over $E(\mathbb{F}_p) : y^2 = x^3 + Ax + B$

An attacker can send a point (x_A, y_A) that is not on E but on a different curve $\tilde{E} : y^2 = x^3 + Ax + \tilde{B}$, where $\tilde{B} = y_A^2 - x_A^3 - Ax_A \pmod{p}$. If the server computes $d_S \cdot (x_A, y_A)$ using its private key d_S, the computation effectively happens on $\tilde{E}$. The attacker can choose $\tilde{E}$ such that its order $\#\tilde{E}(\mathbb{F}_p)$ is smooth or has small subgroups [18]. By observing the result $d_S(x_A, y_A)$ on $\tilde{E}$, the attacker learns information about $d_S \pmod{n_i}$, where n_i is the order of (x_A, y_A) on $\tilde{E}$. If the attacker sends points from several such maliciously chosen curves $\tilde{E}_1, \tilde{E}_2, \ldots, \tilde{E}_t$:

$$\begin{cases} y^2 = x^3 + Ax + \tilde{B}_1 \pmod{p} & (\text{leaks } d_S \pmod{n_1}, \text{ where } n_1 \text{ is small or smooth}) \\ y^2 = x^3 + Ax + \tilde{B}_2 \pmod{p} & (\text{leaks } d_S \pmod{n_2}) \\ \vdots & \\ y^2 = x^3 + Ax + \tilde{B}_t \pmod{p} & (\text{leaks } d_S \pmod{n_t}) \end{cases}$$

If $\text{lcm}(n_1, n_2, \ldots, n_t)$ is greater than or equal to the order of the generator on the original curve (or at least large enough to determine d_S), then the secret key d_S can be recovered using the Chinese Remainder Theorem (CRT) [19]. This is particularly effective if the n_i are small prime powers.

IV.2.4 Singular Elliptic Curves

A cryptographically secure elliptic curve must be non-singular, meaning its discriminant $\Delta = -(4A^3 + 27B^2) \neq 0 \pmod{p}$. If $\Delta \equiv 0 \pmod{p}$, the curve is singular, and the ECDLP can be reduced to a simpler problem [8].

Cusp If $4A^3 + 27B^2 \equiv 0 \pmod{p}$ and the singular point is a cusp (e.g., for the curve $y^2 = x^3 \pmod{p}$, we have $A = 0, B = 0$), the group of non-singular points $E_{ns}(\mathbb{F}_p)$ is isomorphic to the additive group $(\mathbb{F}_p, +)$. The isomorphism can be given by:

$$\psi : E_{ns}(\mathbb{F}_p) \to (\mathbb{F}_p, +), \quad (x, y) \mapsto \frac{x}{y} \text{ (if } y \neq 0), \quad \mathcal{O} \mapsto 0$$

The DLP $Q = kP$ on $E_{ns}(\mathbb{F}_p)$ becomes $q_{\text{iso}} = k \cdot p_{\text{iso}}$ in $\mathbb{F}_p$ (under the isomorphism, p_{iso} is the image of P, and q_{iso} is the image of Q). So $k = q_{\text{iso}} \cdot p_{\text{iso}}^{-1} \pmod{p}$, which is trivial to solve if $p_{\text{iso}} \neq 0$.

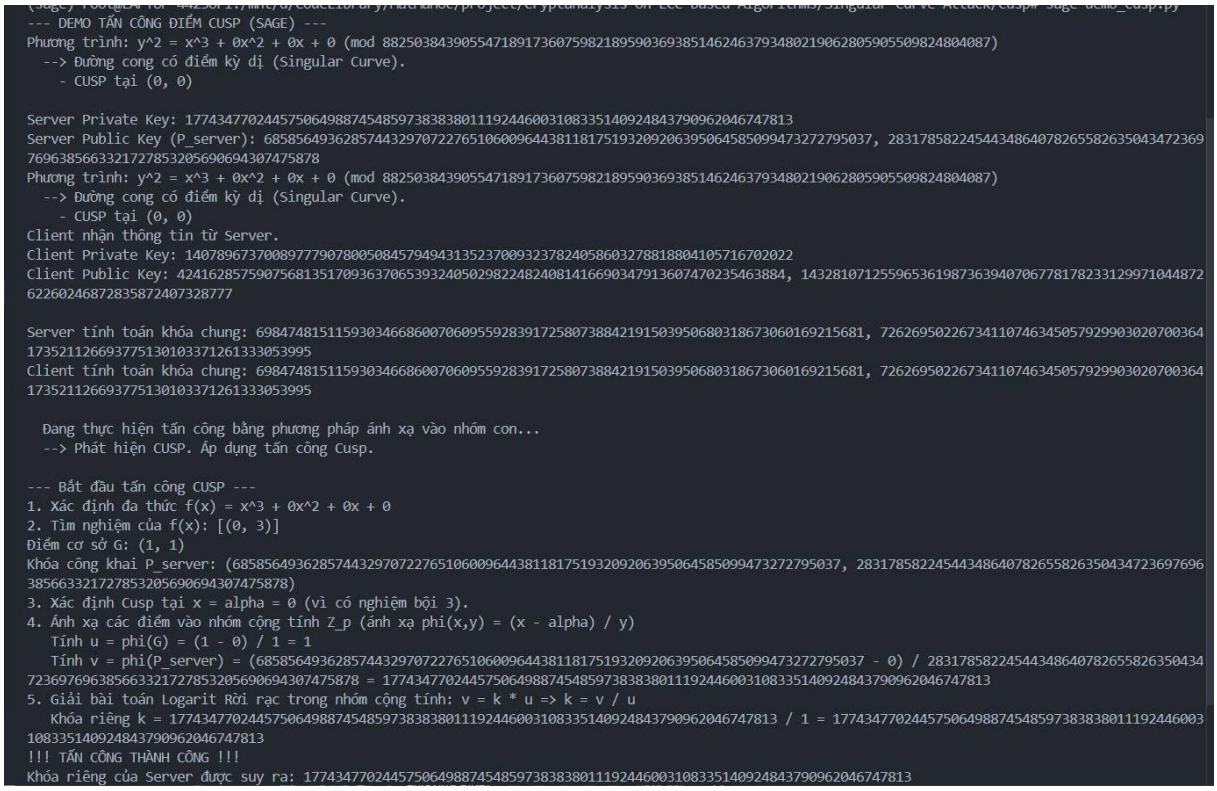

Figure 10: Demo Cusp

Node If $4A^3 + 27B^2 \equiv 0 \pmod{p}$ and the singular point is a node (e.g., for $y^2 = x^3 + Ax^2 = x^2(x + A) \pmod{p}$, we have $B = 0$ and $A \neq 0$), the group of non-singular points $E_{ns}(\mathbb{F}_p)$ is isomorphic to the multiplicative group $(\mathbb{F}_p^*, \times)$ if the slopes at the node are in $\mathbb{F}_p$, or to the multiplicative group of a quadratic extension field otherwise. Assuming it's isomorphic to $\mathbb{F}_p^*$: For a curve like $y^2 = x^2(x + t)$, an isomorphism is:

$$\psi : E_{ns}(\mathbb{F}_p) \to (\mathbb{F}_p^*, \times), \quad (x, y) \mapsto \frac{y + \alpha x}{y - \alpha x} \text{ (for some } \alpha \text{ related to the slopes at the node)}, \quad \mathcal{O} \mapsto 1$$

The DLP $Q = kP$ on $E_{ns}(\mathbb{F}_p)$ becomes $q_{\text{iso}} = p_{\text{iso}}^k \pmod{p}$ in $\mathbb{F}_p^*$. This is the standard DLP in $\mathbb{F}_p^*$, which is solvable using algorithms like Index Calculus if $p - 1$ is smooth.

Figure 11: Demo Node

IV.2.5 Supersingular Curves

MOV Attack [20]

The MOV attack (Menezes-Okamoto-Vanstone) reduces the ECDLP on certain elliptic curves $E(\mathbb{F}_p)$ to the standard DLP in a small extension field $\mathbb{F}_{p^k}$. The integer k is called the **embedding degree**, which is the smallest positive integer such that $n \mid (p^k - 1)$, where $n = \#E(\mathbb{F}_p)$ or $n = \text{ord}(P)$. If k is small (e.g., $k \leq 6$, or generally small enough that DLP in $\mathbb{F}_{p^k}$ is feasible), the curve is vulnerable. Curves for which k is small are often (but not always) **supersingular** curves. The attack uses a non-degenerate bilinear pairing, such as the Weil pairing $e_n : E[n] \times E[n] \to \mu_n(\mathbb{F}_{p^k})$, where $E[n]$ are points of order n and $\mu_n(\mathbb{F}_{p^k})$ is the group of n-th roots of unity in $\mathbb{F}_{p^k}$. The key property is $e_n(aP, bQ) = e_n(P, Q)^{ab}$. Given P (generator of order n) and $Q = mP$, the goal is to find m.

1. Choose a point $R \in E(\mathbb{F}_{p^k})$ such that $R \notin \langle P \rangle$ and $e_n(P, R) \neq 1$.
2. Compute $\alpha = e_n(P, R) \in \mathbb{F}_{p^k}^*$.
3. Compute $\beta = e_n(Q, R) = e_n(mP, R) = e_n(P, R)^m = \alpha^m \in \mathbb{F}_{p^k}^*$.
4. Solve the DLP $\beta = \alpha^m$ for m in the field $\mathbb{F}_{p^k}^*$.

If k is small, this DLP can be much easier to solve than the original ECDLP. For instance, if p is a 160-bit prime and $k = 2$, the DLP is in $\mathbb{F}_{p^2}$ (a 320-bit field), which might be significantly easier than ECDLP over a 160-bit group. Cryptographically chosen curves usually have very large embedding degrees [1].

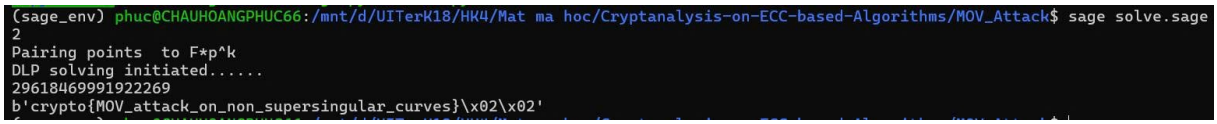

Figure 12: MOV Attack Demo

IV.2.6 Anomalous Elliptic Curves

Smart Attack [21]

This attack, proposed by Nigel Smart, applies to elliptic curves where the number of points over $\mathbb{F}_p$ is equal to p, i.e., $\#E(\mathbb{F}_p) = p$. Such curves are called anomalous curves. This condition implies that the trace of Frobenius $t = p + 1 - \#E(\mathbb{F}_p) = p + 1 - p = 1$.

The Smart attack provides a linear-time algorithm (in terms of $\log p$) for solving the ECDLP on these curves. The attack involves lifting the points $P, Q \in E(\mathbb{F}_p)$ to points P', Q' on the same elliptic curve but defined over the p-adic numbers $\mathbb{Q}_p$. This is done using Hensel's Lemma to find p-adic y-coordinates corresponding to the x-coordinates from $\mathbb{F}_p$.

If $Q = kP$ in $E(\mathbb{F}_p)$, then $Q' - kP'$ reduces to the point at infinity O modulo p. This means $Q' - kP'$ is in the kernel of the reduction map $E(\mathbb{Q}_p) \to E(\mathbb{F}_p)$. At a crucial step uses the p-adic elliptic logarithm $\psi : E_1(\mathbb{Q}_p) \to p\mathbb{Z}_p$, which is an isomorphism under certain conditions. The relationship becomes:

$$\psi(pQ') - k\psi(pP') \equiv 0 \pmod{p^N\mathbb{Z}_p} \text{ (for sufficiently large } N) \tag{1}$$

This allows solving for k:

$$k \equiv \frac{\psi(pQ')}{\psi(pP')} \pmod{p} \tag{2}$$

The p-adic logarithms can be computed efficiently. Such curves are generally avoided in practice

Countermeasure: Ensure that $\#E(\mathbb{F}_p) \neq p$ during curve parameter generation and validation.

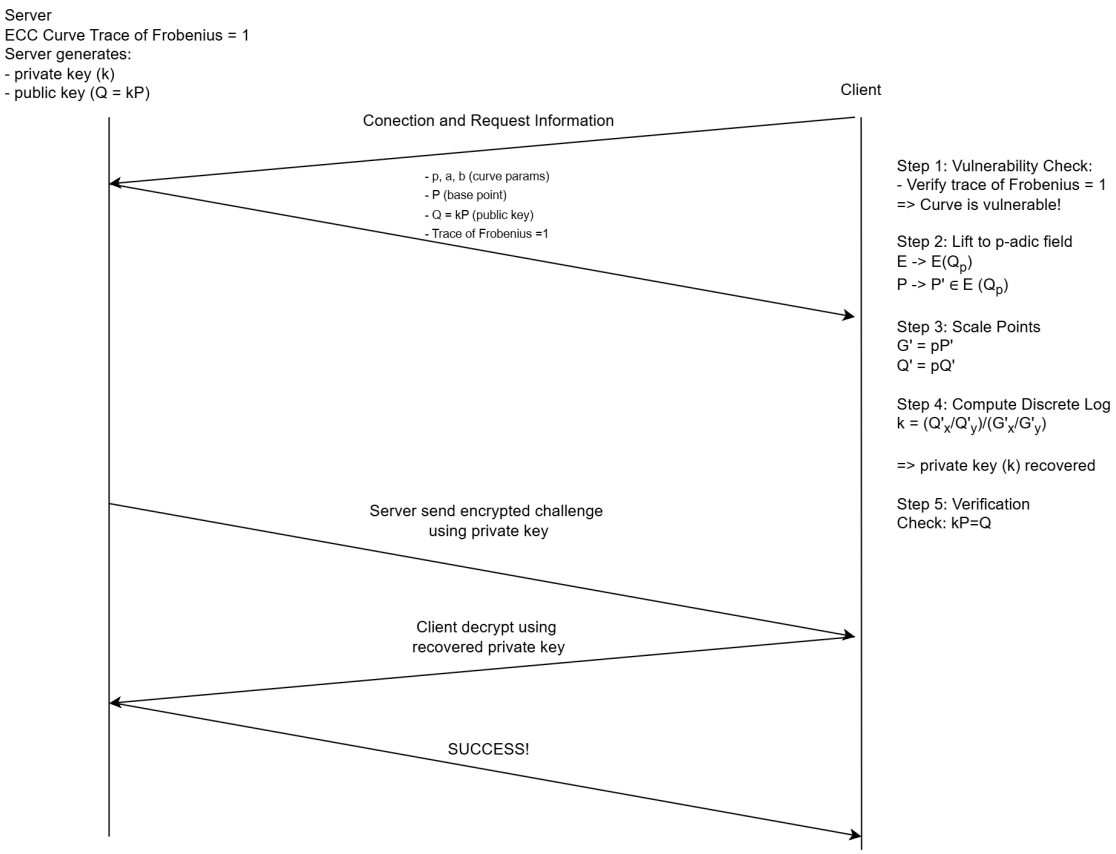

Figure 13: Smart Attack Diagram (Conceptual Illustration)

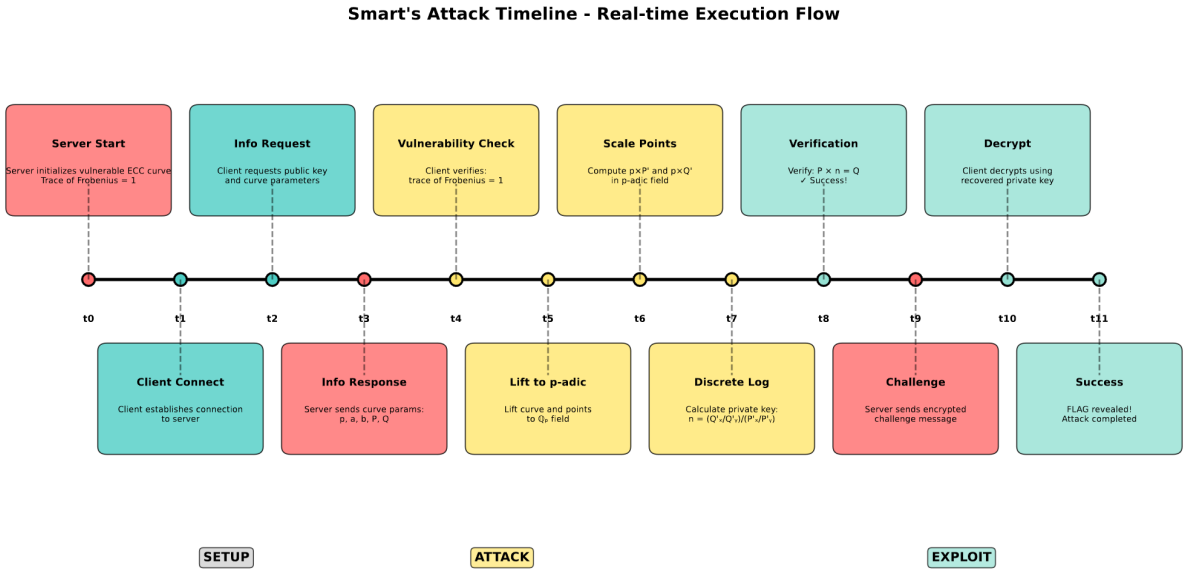

Figure 14: Smart Attack Timeline (Conceptual Illustration)

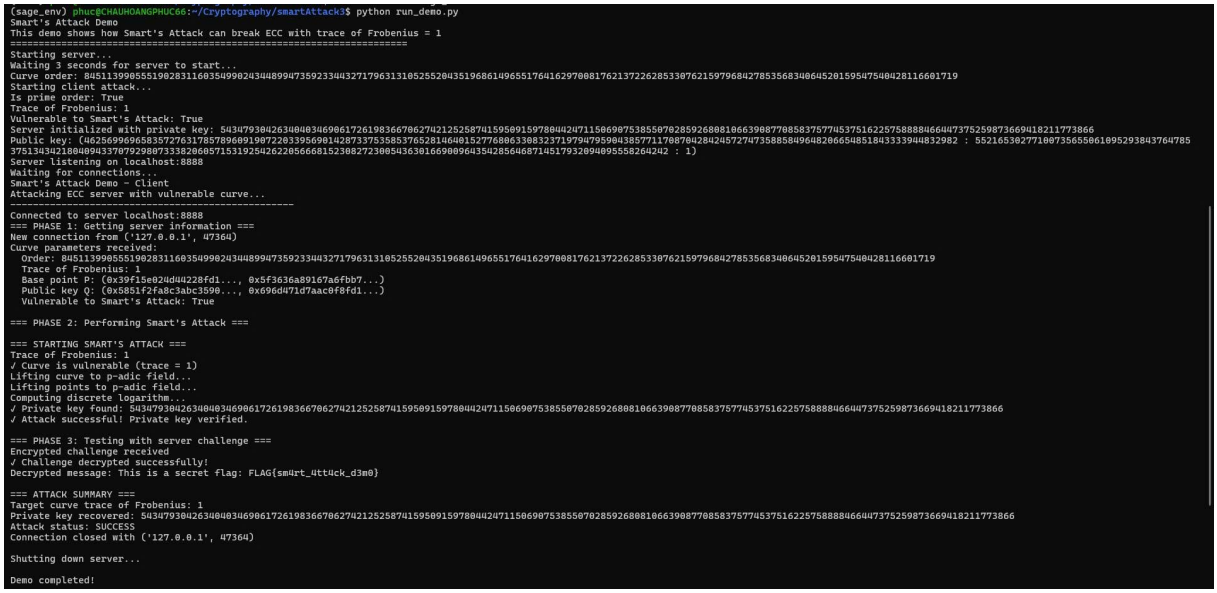

Figure 15: Smart Attack Demo (Conceptual Illustration of SageMath use)

IV.3 ECDSA Signature Scheme

The Elliptic Curve Digital Signature Algorithm (ECDSA) [4] generation process is as follows: Let n be the order of the base point G. Let d_A be the private key and $Q_A = d_A G$ be the public key. To sign a message m:

1. Compute: $e = \text{HASH}(m)$ using a cryptographic hash function (e.g., SHA-256).
2. Let z be the leftmost L_n bits of e, where L_n is the bit length of n. (Often, z is e taken as an integer modulo n).
3. Choose a cryptographically secure random integer $k \in [1, n-1]$ (the nonce).

4. Compute the curve point $(x_1, y_1) = kG$.

5. Compute $r = x_1 \pmod{n}$. If $r = 0$, go back to step 3.

6. Compute $s = k^{-1}(z + rd_A) \pmod{n}$. If $s = 0$, go back to step 3.

7. The signature is the pair (r, s).

The security of ECDSA is analyzed in depth in works like [14].

IV.3.1 Not Hashing the Message

If the full message m is not hashed, but instead, z is derived directly from a (possibly truncated) part of m, vulnerabilities can arise. For example, if z is taken from the first L_n bits of m, an attacker could append arbitrary data to m without changing z, and thus the signature (r, s) would remain valid for the modified message.

Example:

Original signed message m_1 leading to z:

> "Please authorize a payment of $500 to Alice for consulting services."

Attacker's altered message m_2 (if only the prefix is used for z):

> "Please authorize a payment of $500 to Alice for consulting services. Also, transfer $100,000 to Mallory's offshore account."

If z is derived only from the first part, both messages might yield the same z, and thus the same signature would validate m_2. This highlights the importance of hashing the *entire* message that is to be signed.

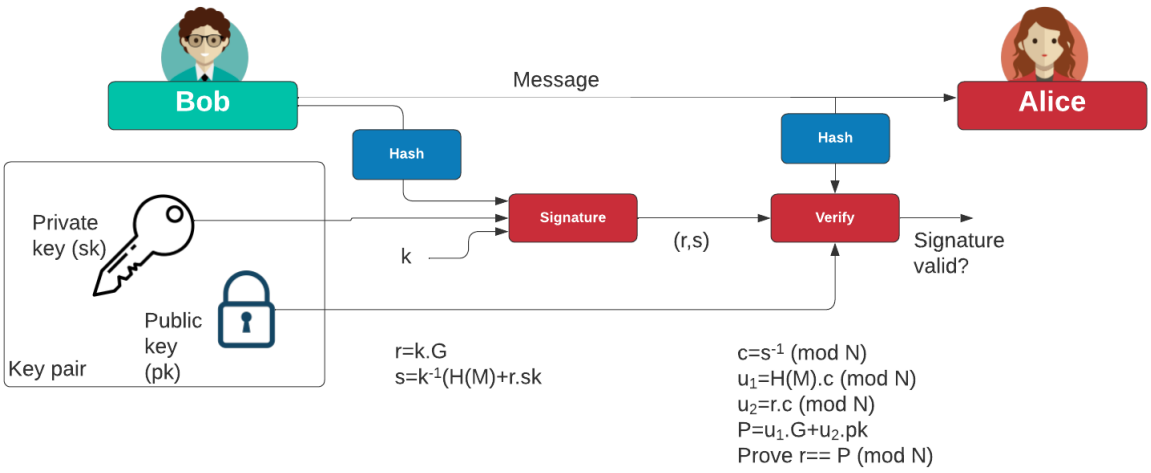

Figure 16: Outline of ECDSA

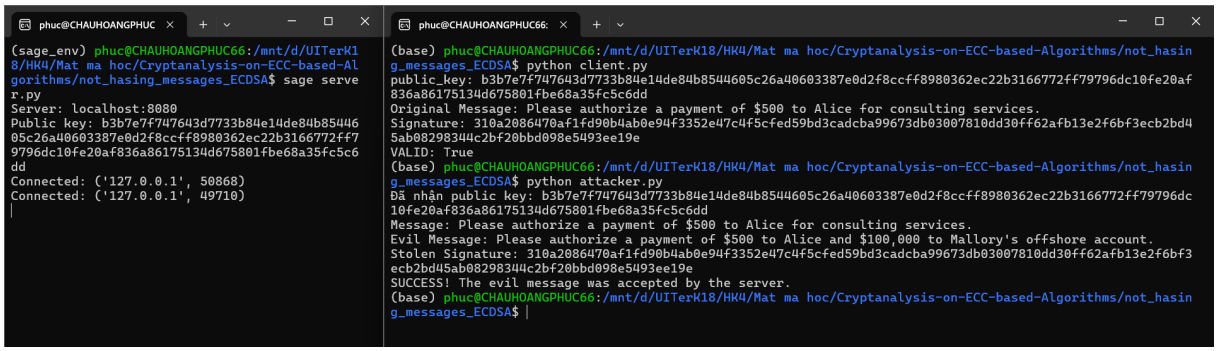

Figure 17: Not Hashing Message Demo

IV.3.2 Nonce Reuse

If the same random nonce k is used to sign two different messages m_1 and m_2 (with corresponding hash values z_1 and z_2), an attacker who obtains both signatures (r, s_1) and (r, s_2) can recover the private key d_A. Since $kG = (x_1, y_1)$ is the same for both, $r = x_1 \pmod{n}$ will be the same. We have:

$$s_1 = k^{-1}(z_1 + rd_A) \pmod{n}$$

$$s_2 = k^{-1}(z_2 + rd_A) \pmod{n}$$

This implies:

$$ks_1 \equiv z_1 + rd_A \pmod{n}$$

$$ks_2 \equiv z_2 + rd_A \pmod{n}$$

Subtracting the two congruences:

$$k(s_1 - s_2) \equiv z_1 - z_2 \pmod{n}$$

If $s_1 \not\equiv s_2 \pmod{n}$ (which happens if $z_1 \not\equiv z_2 \pmod{n}$), then k can be recovered:

$$k \equiv (z_1 - z_2)(s_1 - s_2)^{-1} \pmod{n}$$

Once k is known, the private key d_A can be recovered from either signature equation:

$$d_A \equiv r^{-1}(ks_1 - z_1) \pmod{n}$$

(Requires $r \not\equiv 0 \pmod{n}$, which is a condition for valid signatures). This was famously exploited in the Sony PlayStation 3 hack [22].

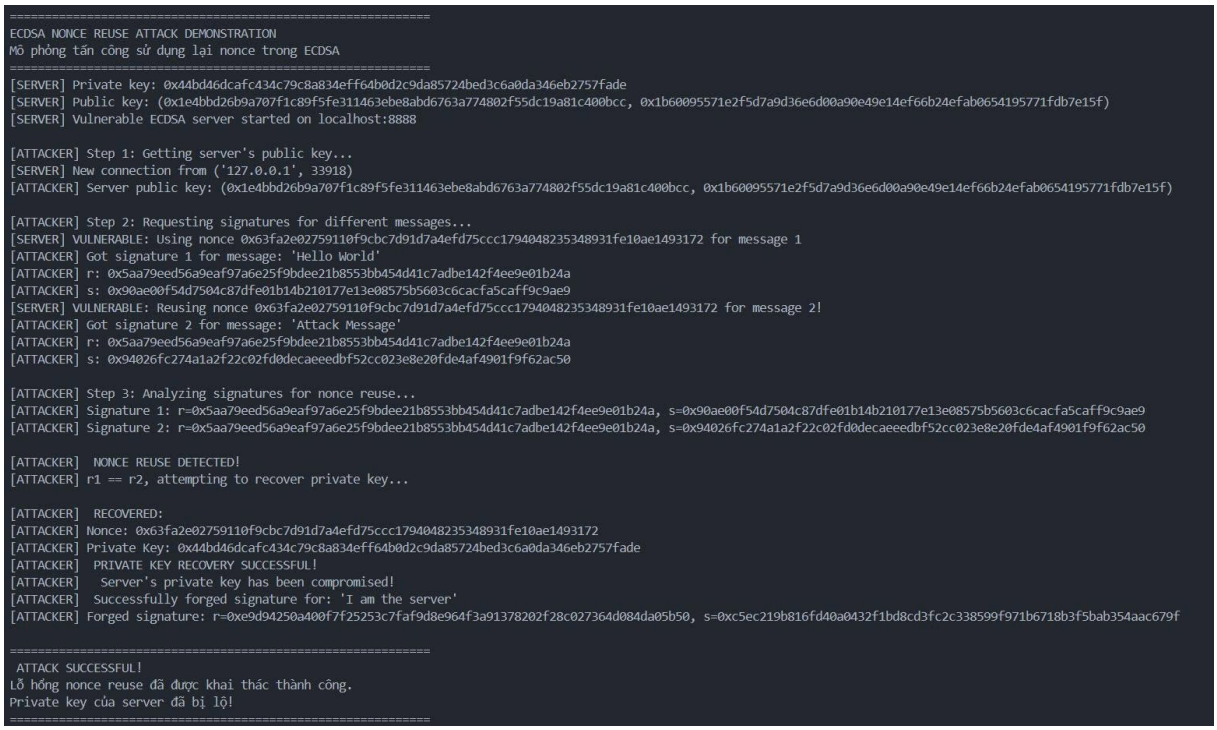

Figure 18: Demo Nonce Reuse

IV.3.3 Biased Nonce and Lattice Attacks

One of the most critical requirements for the security of ECDSA is that the nonce k (a random number used once) must be chosen uniformly at random from the interval $[1, n-1]$ and must be unique for each signature. If the nonce k is not truly random but is "biased" – for example, some bits of k are predictable, k is chosen from a range significantly smaller than n, or k has a special structure due to flaws in the random number generator (RNG) – then an attacker can recover the private key d_A by using lattice-based techniques, provided enough signatures are collected [23].

Principle of Biased Nonce Attacks From the ECDSA signature equation:

$$s = k^{-1}(z + rd_A) \pmod{n}$$

We can rearrange this to express k:

$$k \equiv s^{-1}(z + rd_A) \pmod{n}$$

Alternatively:

$$k \equiv s^{-1}z + s^{-1}rd_A \pmod{n}$$

Suppose an attacker collects m signatures (r_i, s_i) for messages with corresponding hash values z_i (for $i = 1, \ldots, m$). For each signature, we have an equation relating the nonce k_i and the private key d_A:

$$k_i \equiv s_i^{-1}z_i + (s_i^{-1}r_i)d_A \pmod{n}$$

Let $u_i = s_i^{-1}z_i \pmod{n}$ and $v_i = s_i^{-1}r_i \pmod{n}$. Then:

$$k_i \equiv u_i + v_i d_A \pmod{n}$$

This means that $k_i - v_i d_A - u_i$ is a multiple of n. In other words:

$$k_i - v_i d_A - u_i = t_i n \quad \text{for some integer } t_i.$$

Or, approximately:

$$k_i \approx (v_i d_A + u_i) \pmod{n}$$

If the nonces k_i are biased in some way (e.g., they are smaller than a certain threshold X, or certain bits of k_i are 0 or predictable), an attacker can exploit this information.

Using Lattices to Solve the Hidden Number Problem (HNP)

The problem of finding d_A from the above equations, given that the k_i's have a certain bias, can be modeled as a form of the Hidden Number Problem (HNP) or the Closest Vector Problem (CVP) in a lattice.

One way to construct a lattice is as follows. We are looking for a value d_A (the private key) and "small" values $k_1, \ldots, k_m$ (the biased nonces) such that the following equations hold modulo n:

$$k_i - v_i d_A \equiv u_i \pmod{n} \quad \text{for } i = 1, \ldots, m$$

This can be rewritten as:

$$v_i d_A - k_i + u_i \equiv 0 \pmod{n}$$

Or $v_i d_A - k_i + u_i = \lambda_i n$ for some integer λ_i.

If we assume that $|k_i| < K_{max}$ (e.g., K_{max} is 2^ℓ if the top ℓ bits of k_i are 0), we can set up a lattice. Consider the lattice L generated by the columns (or rows, depending on convention) of a matrix. The precise construction of the lattice can be complex and depends on the exact assumption about the nonce bias. Prominent approaches are based on the work of Boneh and Durfee, or Howgrave-Graham for HNP.

For instance, if we know that $k_i < K_{max}$ and typically $d_A < n$. A common lattice construction for attacking biased ECDSA nonces involves finding a short vector in a lattice whose basis vectors are derived from the signature equations. Given m signatures, we have $k_i s_i \equiv z_i + r_i d_A \pmod{n}$. If the top L bits of each k_i are known (e.g., they are zero, so $k_i < 2^{\text{bitlength}(n)-L}$), we can construct a lattice. Consider the set of linear equations modulo n:

$$k_i \equiv u_i + v_i d_A \pmod{n}$$

We are searching for d_A and small integers k_i. This is a classic setup for HNP. A lattice can be formed by vectors related to these congruences. For example, a lattice L could be generated by the rows of the following $(m+1) \times (m+1)$ matrix:

$$\begin{pmatrix} n & 0 & \ldots & 0 & 0 \\ 0 & n & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & n & 0 \\ v_1 & v_2 & \ldots & v_m & C \end{pmatrix}$$

where C is a carefully chosen constant (e.g., 1 or a small integer to give weight to d_A). We are looking for a vector $x = (x_0, x_1, \ldots, x_m)$ in this lattice such that if $x = (d_A, \lambda_1, \ldots, \lambda_m) \cdot M_{basis}$, then the resulting $k_i = u_i + v_i d_A - \lambda_i n$ are small. More practically, the problem

is often framed as finding a vector $(d_A, k_1, \ldots, k_m)$ that satisfies the equations and where k_i are small. A common matrix used in lattice attacks on ECDSA (e.g., when L most significant bits of k_i are known for $i = 1, \ldots, d$) might look like:

$$M = \begin{pmatrix} n & 0 & \ldots & 0 & 0 \\ 0 & n & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & n & 0 \\ s_1^{-1}r_1 & s_2^{-1}r_2 & \ldots & s_d^{-1}r_d & X \end{pmatrix}_{(d+1)\times(d+1)}$$

where X is a scaling factor, e.g., $X \approx 1/2^{\text{bitlength}(n)-L}$. The target is to find a short vector in the lattice spanned by the rows of this matrix. Such a vector corresponds to a linear combination of the rows which, when its last component is appropriately scaled, reveals d_A. The other components relate to the nonces k_i.

The LLL algorithm (Lenstra–Lenstra–Lovász) is a powerful tool used to find a basis of "short" vectors for a given lattice. If a vector in this basis (or a specific linear combination thereof) is sufficiently short, it can reveal information about d_A.

Conditions for a Successful Attack The number of signatures m required and the probability of success depend on:

1. **The bias of the nonce** k: The greater the bias (e.g., the more bits of k are known, or the smaller k is compared to n), the fewer signatures are needed. If only a few bits of k are leaked per signature (e.g., 2-3 bits), tens to hundreds of signatures might be required for a successful attack.

2. **The size of** n: The larger the modulus n, the harder the attack becomes, as the search space is larger.

3. **The dimension of the lattice**: The dimension of the lattice (usually related to the number of signatures m) affects the running time of the LLL algorithm. The LLL algorithm has a polynomial time complexity in the dimension of the lattice and the logarithm of the size of its entries.

If an attacker can collect enough signatures with biased nonces, they can construct an appropriate lattice and use the LLL algorithm (or stronger variants like BKZ) to find a short vector. This short vector often corresponds to a linear (or approximately linear) relationship involving the private key d_A and the known values (r_i, s_i, z_i), allowing the attacker to solve for d_A.

These attacks underscore the importance of using a cryptographically strong pseudo-random number generator (CSPRNG) to generate the nonce k, or, preferably, using secure deterministic nonce generation methods as described in RFC 6979. RFC 6979 generates the nonce k deterministically from the private key and the message hash, completely eliminating reliance on an RNG at signing time and thereby avoiding issues of biased or repeated nonces.

Figure 19: Demo Biased Nonce

V Proposed Solution

V.1 ECDH (Elliptic Curve Diffie-Hellman) Vulnerabilities

V.1.1 Curve Agreement and Trust: Critical Security Considerations in ECDH

In Elliptic Curve Diffie-Hellman (ECDH), both parties must agree on the elliptic curve parameters (p, A, B, G, n, h) at the beginning of the key exchange protocol. This agreement is a fundamental security requirement detailed in standards like [5].

The Main Risk: Untrusted Curve Parameters

If one party (Alice) allows the other party (Bob, potentially an attacker or a Man-in-the-Middle) to supply the curve parameters, Bob can choose parameters that weaken the security of the exchange for Alice. This can lead to:

1. **Reduced ECDLP difficulty:** Bob can choose a curve where the ECDLP is easy (e.g., a singular curve [8], a supersingular curve vulnerable to the MOV attack [20], an anomalous curve vulnerable to Smart's attack [21], or a curve whose order is small or smooth [17]). Alice would compute her shared secret using her private key on this weak curve, allowing Bob to determine Alice's private key (or the shared secret directly).

2. **Invalid Curve Attacks:** As described in Section IV.B.4, Bob can provide a point that is not on the intended curve but on a different curve $\tilde{E}$ where computations are done. Alice, using her private key d_A, would compute $d_A P_B$ on $\tilde{E}$. If $\tilde{E}$ has a small order subgroup containing P_B, Bob learns $d_A \pmod{\text{ord}_{\tilde{E}}(P_B)}$ [19].

3. **Small Subgroup Attacks:** Even if the main curve E is secure, Bob might provide a point P_B that belongs to a small subgroup of E. Alice computes $d_A P_B$. The resulting shared secret $d_A d_B G$ will then also be in this small subgroup. Bob can then test possible shared secrets (since there are few of them).

To mitigate this, Alice must either:

- Only use standardized, well-vetted curve parameters (e.g., from NIST [4], SECG [10], or widely trusted curves like Curve25519 [6]).
- If Bob provides curve parameters, Alice must rigorously validate them (check for non-singularity, estimate order, check for known weaknesses). This is complex and generally discouraged.
- If Bob provides a public key (a point P_B), Alice must validate that P_B is on the agreed curve and is not the point at infinity. She should also verify that P_B has the correct order (i.e., $nP_B = \mathcal{O}$ and P_B is not in any small subgroups, often checked by $hP_B \neq \mathcal{O}$ if h is small and n is prime). Point validation is a key defense [19].

V.1.2 Solution Architecture

To prevent the classes of attacks associated with ECDH (e.g., invalid curve attacks [18], malicious parameter injection, weak key reuse), the architecture must be built on strong cryptographic primitives, strict validation mechanisms, and properly modularized components.

Cryptographic Algorithms and Protocols

- **Use Standardized Curves:** Exclusively use well-known and vetted elliptic curves such as those specified in NIST FIPS 186-4 [4], SECG SEC 2 [10], or modern high-security curves like Curve25519 [6] and Ed25519. Avoid custom or non-standard curves unless thoroughly analyzed by experts [11].
- **Point Validation:** Always validate incoming public keys (points). This includes:
  1. Checking if the coordinates are in the correct range (e.g., $0 \leq x, y < p$).
  2. Verifying that the point satisfies the curve equation $y^2 \equiv x^3 + Ax + B \pmod{p}$.
  3. Ensuring the point is not the point at infinity $\mathcal{O}$.
  4. Verifying that the point lies in the correct subgroup of prime order n. This is typically done by checking $nP = \mathcal{O}$. For curves with cofactor $h > 1$, also check that P is not in any small subgroup. This can be done by verifying $hP \neq \mathcal{O}$ if P is supposed to generate the full group of order hn, or by working with hP_B if the key derivation only needs a point in the prime order subgroup.
- **Secure Key Generation:** Private keys must be generated using a cryptographically secure pseudo-random number generator (CSPRNG) and be of appropriate length (e.g., close to the bit length of n).
- **Ephemeral Keys (ECDHE):** For key exchange, prefer Elliptic Curve Diffie-Hellman Ephemeral (ECDHE). In ECDHE, both parties generate a new key pair for each session, providing forward secrecy. This prevents compromise of past session keys if a long-term private key is compromised.

V.2 ECDSA (Elliptic Curve Digital Signature Algorithm) Vulnerabilities

V.2.1 Nonce Management is Critical

The security of ECDSA heavily relies on the uniqueness and unpredictability of the per-signature nonce k.

- **Prevent Nonce Reuse:** As shown in Section IV.C.2, reusing k for different messages with the same private key leads to private key recovery [22]. Implementations must ensure k is never repeated.
- **Prevent Biased Nonces:** Nonces must be uniformly random or generated deterministically in a way that prevents bias (e.g., as per RFC 6979). Biased nonces can be exploited by lattice attacks [23]. Using a CSPRNG is essential if nonces are generated randomly.
- **Deterministic ECDSA (RFC 6979):** To avoid issues with faulty RNGs leading to nonce reuse or bias, consider implementing deterministic ECDSA as specified in RFC 6979. This standard defines a way to generate k deterministically from the private key and the message hash, ensuring uniqueness and good distribution properties while removing reliance on an external RNG at signing time.

V.2.2 Signature and Message Integrity

- **Hash the Entire Message:** Always hash the entire message being signed, as discussed in Section IV.C.1. The hash input should unambiguously represent the data being protected.
- **Proper Signature Verification:** When verifying a signature (r, s):
  1. Check that $r, s \in [1, n-1]$.
  2. Compute $e = \text{HASH}(m)$ and let z be the integer derived from e.
  3. Compute $w = s^{-1} \pmod{n}$.
  4. Compute $u_1 = zw \pmod{n}$ and $u_2 = rw \pmod{n}$.
  5. Compute the curve point $(x_1, y_1) = u_1 G + u_2 Q_A$.
  6. If $(x_1, y_1) = \mathcal{O}$, the signature is invalid.
  7. The signature is valid if and only if $r \equiv x_1 \pmod{n}$.
- **Protect Against Fault Attacks:** Implementations on physical devices should consider countermeasures against fault injection attacks (e.g., Bellcore attack variants) that might corrupt computations to leak key material [24, 18].

VI Deployment

To ensure the secure deployment of our ECC-based cryptosystem, we propose the following strategies to mitigate risks and enhance security:

VI.1 Use Recommended Curves and Strong Keys

To prevent attacks exploiting weaknesses in elliptic curves, carefully select curves that are verified and standardized. We strongly recommend using curves such as:

- **Curve25519** (and its signature counterpart Ed25519): Known for high security, efficiency, and resistance to many side-channel and invalid curve attacks due to its Montgomery/Edwards form and careful parameter selection [6]. The use of specific

curve forms like Edwards or Hessian can also improve resistance to side-channel attacks [25].

- **NIST P-curves (P-256, P-384, P-521):** Standardized by NIST [4] and widely supported, though they require more careful implementation to avoid certain pitfalls compared to newer curves.
- Other curves from SECG [10] or those vetted by projects like SafeCurves [11]. For example, Curve448 (Ed448).

These curves are endorsed by standards like NIST and SECG, and modern recommendations often favor curves like Curve25519 for new deployments. For further details, refer to:

SECG Guidelines: `https://www.secg.org/sec2-v2.pdf`
Standard Curve Specifications: `https://neuromancer.sk/std/` (This is a useful compilation, but primary sources like NIST/SECG should be consulted.)
SafeCurves: `https://safecurves.cr.yp.to`

Ensure private keys are generated using a cryptographically secure random number generator and have a bit length comparable to the order of the curve's base point.

VI.2 Prevent Invalid Curve Attacks

To counter invalid curve attacks (Section IV.B.4), the server (or any party receiving an ECC point) must validate all untrusted user inputs. Specifically:

- **Full Point Validation:** Before using any externally supplied point $P = (x, y)$ in cryptographic operations:
  1. Verify that x and y coordinates are in the field $\mathbb{F}_p$ (i.e., $0 \leq x, y < p$).
  2. Verify that the point satisfies the curve equation: $y^2 \equiv x^3 + Ax + B \pmod{p}$.
  3. Verify that the point is not the point at infinity $\mathcal{O}$.
  4. Verify that the point is in the correct (large prime order n) subgroup. This is done by checking $nP = \mathcal{O}$. For curves with a small cofactor $h > 1$, it is also common to check that $P \neq \mathcal{O}$ after multiplying by h (i.e., $hP \neq \mathcal{O}$) to ensure it is not in a small cofactor subgroup, or alternatively ensure that the key derivation works with hP. Most modern curve choices (like Curve25519) simplify this by having small cofactors (e.g., $h = 8$ for Curve25519, $h = 4$ for Ed25519) and specific protocols for their use. Validating points is crucial [19].
- Implement strict input validation for all parameters to reject malformed or malicious data.

Using curves in Montgomery or Edwards form (like Curve25519/Ed25519) can intrinsically avoid some of these checks or make them simpler [6].

VI.3 Regular Security Updates and Testing

To maintain the integrity of the cryptosystem:

- Regularly update cryptographic libraries (e.g., OpenSSL, libsodium, PyCryptodome) and dependencies to patch known vulnerabilities.
- Conduct periodic security audits, including code reviews and penetration testing, focusing on the cryptographic implementation and protocol logic.
- Stay informed about new research in ECC cryptanalysis and adjust security measures accordingly.
- Upgrade to the latest secure versions of ECC implementations as new standards emerge or older ones are found to have weaknesses [7].

VI.4 Recommended Curves for Specific Applications

For applications such as key exchange (e.g., ECDH) or digital signatures (e.g., ECDSA), we recommend using standardized and well-vetted curves like:

- **Curve25519 (for ECDH, often called X25519) and Ed25519 (for signatures):** Ideal for high-speed cryptography, widely adopted in modern protocols (TLS 1.3, SSH, Signal), and designed with several safety features [6].
- **NIST P-256 (secp256r1):** Widely deployed but has more "special-case" handling requirements in implementation than Curve25519. Still considered secure if implemented correctly, as per [4].

The choice may also depend on interoperability requirements with existing systems. Careful selection of curves is paramount for security [11].

References

[1] Darrel Hankerson, Alfred J. Menezes, and Scott A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer, 2004.

[2] Standards for Efficient Cryptography Group (SECG). Sec 1: Elliptic curve cryptography. Technical report, Standards for Efficient Cryptography Group (SECG), May 2009. Often cited as ref1 in the document.

[3] Alfred J. Menezes. Elliptic curve public key cryptosystems. In *The Kluwer International Series in Engineering and Computer Science*, pages 1–136. Springer US, 1993.

[4] National Institute of Standards and Technology (NIST). Fips pub 186-4: Digital signature standard (dss). Technical report, U.S. Department of Commerce, July 2013.

[5] National Institute of Standards and Technology (NIST). Sp 800-56a revision 3: Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography. Technical report, U.S. Department of Commerce, April 2018.

[6] Daniel J. Bernstein. Curve25519: new diffie-hellman speed records. *Public Key Cryptography - PKC 2006*, 3958:207–228, 2006.

[7] National Institute of Standards and Technology (NIST). Workshop on elliptic curve cryptography standards. Technical report, U.S. Department of Commerce, 2015. Often cited as ref2 in the document.

[8] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman and Hall/CRC, second edition, 2008.

[9] René Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44(170):483–494, 1985.

[10] Standards for Efficient Cryptography Group (SECG). Sec 2: Recommended elliptic curve domain parameters. Technical report, Standards for Efficient Cryptography Group (SECG), January 2010.

[11] Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe elliptic curves for cryptography, 2013. Ongoing project and website. Version from 2013-09-24. `https://safecurves.cr.yp.to`.

[12] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[13] Victor S. Miller. Use of elliptic curves in cryptography. *Advances in Cryptology — CRYPTO '85 Proceedings*, 218:417–426, 1986.

[14] Daniel R. L. Brown. Generic groups, collision resistance, and ecdsa. *Designs, Codes and Cryptography*, 35(1):119–152, 2005.

[15] Daniel Shanks. Class number, a theory of factorization, and genera. *Proceedings of Symposia in Pure Mathematics*, 20:415–440, 1971.

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

[17] Stephen C. Pohlig and Martin E. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24(1):106–110, 1978.

[18] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer Berlin Heidelberg, 2000.

[19] Adrian Antipa, Daniel Brown, Robert Gallant, Robert Lambert, Rene Struik, and Scott Vanstone. Validated ecc on the aardvark (tm) microcontroller. In Mitsuru Matsui and Robert Zuccherato, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 355–367. Springer Berlin Heidelberg, 2003.

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

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

[22] fail0verflow. PS3 Epic Fail. `https://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html`, December 2010. Presentation at 27th Chaos Communication Congress (27C3). Details the ECDSA nonce reuse vulnerability in Sony PlayStation 3.

[23] Nick Howgrave-Graham and Nigel P. Smart. Lattice attacks on digital signature schemes. *Designs, Codes and Cryptography*, 23(3):283–290, 2001.

[24] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer Berlin Heidelberg, 1997.

[25] Marc Joye and Jean-Jacques Quisquater. Hessian elliptic curves and side-channel attacks. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 402–410. Springer Berlin Heidelberg, 2001.