Elliptic Curve Cryptography

Sebastian Pallais-Aks

November 19th, 2025

Outline

Connection to Modular Forms: Modularity Theorem

Elliptic Curve Discrete Log Problem

ECIES

CCA Security

Elliptic Curves

- An elliptic curve E is given by $y^2 = x^3 + Ax + B$.
- We can reduce $E \pmod{p}$ and count points $N_p = |E(\mathbb{F}_p)|$.
- The "error terms" $a_p = p + 1 - N_p$ encode deep arithmetic.
- From these, we build the Hasse-Weil L-function: $L(E, s)$.

Modular Forms

- A (newform) cusp form f of weight 2 has a Fourier expansion:

$$f(\tau) = \sum_{n=1}^{\infty} b_n q^n \quad (q = e^{2\pi i \tau})$$

- From its coefficients b_n, we also build an L-function: $L(f, s)$.

The Modularity Theorem

Theorem (Taniyama-Shimura-Weil, Wiles, et al.)

*Every elliptic curve E over $\mathbb{Q}$ is **modular**.*

What This Means

For every $E/\mathbb{Q}$, there exists a modular form f (of weight 2, for some $\Gamma_0(N)$) such that their L-functions are identical:

$$L(E, s) = L(f, s)$$

This implies their coefficients match: $\boldsymbol{a_p = b_p}$ for all (good) primes p.

The Playground: $E(\mathbb{F}_p)$

The Group

- We fix a large prime p and work with an elliptic curve E over $\mathbb{F}_p$.
- The set of points forms a finite abelian group (over addition):

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p^2 \mid y^2 \equiv x^3 + Ax + B \pmod{p}\} \cup \{\mathcal{O}\}$$

- We pick a base point P that generates a large subgroup of prime order n.

The Playground: $E(\mathbb{F}_p)$

The Operation

We define $R = P + Q$ as follows:

- Draw a straight line that passes through both P and Q.
- By the definition of an Elliptic Curve, we know that this line will intersect the elliptic curve at exactly one other point, S.
- R is the reflection of S across the x-axis.

The Playground: $E(\mathbb{F}_p)$

The "Easy" Problem: Scalar Multiplication

- **Given:** $k \in \mathbb{Z}$ and P (a point on $E(\mathbb{F}_p)$).
- **Compute:** $Q = kP = P + P + ... + P$ (k times).
- **How:** Fast, using the "double-and-add" algorithm (analog of repeated squaring).
- **Runtime:** $O(\log k)$.

The "Hard" Problem: ECDLP

- **Given:** P and $Q = kP$.
- **Find:** The integer k.
- This is the **Elliptic Curve Discrete Logarithm Problem (ECDLP)**.
- The security of all ECC rests on the hardness of this problem.

Proof Sketch: Why is ECDLP "Harder" than Factoring?

Classical DLP (in $\mathbb{Z}_p^*$)

- **Problem:** Find k where $h \equiv g^k \pmod{p}$.
- **Attack:** The sub-exponential **Index Calculus** algorithm.
- **Why it works:** It relies on the "structure" of $\mathbb{Z}$. We can "factor" numbers into a factor base of small primes.
- **Runtime: Sub-exponential**.

Proof Sketch: Why is ECDLP "Harder" than Factoring?

ECDLP (in $E(\mathbb{F}_p)$)

- **Problem:** Find k where $Q = kP$.
- **Attack:** No known "Index Calculus" analog.
- **Why?**: There is no known "factor base" of points. Thus, we can't exploit smoothness in the same way as with $\mathbb{Z}$. This is due to the
- **Best Attacks:** Generic group algorithms (Pollard's Rho, Baby-Step Giant-Step).
- **Runtime:** $O(\sqrt{n})$. **This is exponential in the bit-length of n.**

ECC vs. RSA

RSA Attack (General Number Field Sieve - GNFS)

For an input key of k bits, the runtime is **sub-exponential**:

$$O\left(\exp\left(c \cdot k^{1/3} \cdot (\log k)^{2/3}\right)\right)$$

The exponent ($k^{1/3}$) grows *slower* than k.

ECC Attack (Pollard's Rho)

For an input key of k bits, the runtime is **exponential**:

$$O(2^{k/2})$$

The exponent ($k/2$) grows *linearly* with k.

ECC vs. RSA

Conclusion

To get 2^{128} security:

- **ECC**: We need $k/2 = 128 \implies \boldsymbol{k = 256}$ **bits**.
- **RSA**: We need $k^{1/3}(\dots) \approx 128 \implies \boldsymbol{k = 3072}$ **bits**.

Elliptic Curve Diffie-Hellman

Public:
- Elliptic curve E
- Point P on E
- $n \in \mathbb{Z}$

Alice
- Picks private key $a \in \{1, \ldots, n-1\}$
- Computes public key $A = aP = P + P + \cdots + P$ (a times)

Bob
- Picks private key $b \in \{1, \ldots, n-1\}$
- Computes public key $B = bP = P + P \cdots + P$ (b times)

Key exchange:

Alice computes $S = aB = a(bP) = (ab)P$

Bob computes $S = bA = b(aP) = (ba)P$

Now, they share a secret point S

Eve cannot find S without solving a hard problem

Elliptic Curve Integrated Encryption Scheme

Setup

Alice has Bob's public key B and a message m.

1. **Key Generation (Asymmetric):**
   - Alice generates a new, *ephemeral* private key r.
   - She computes the ephemeral public key $R = rP$.
   - She computes the shared secret: $\boldsymbol{S = rB}$.
2. **Key Derivation (KDF):**
   - Use the x-coordinate of S to derive symmetric keys:

$$K_{\text{enc}} \| K_{\text{mac}} = \text{KDF}(S_x)$$

3. **Encryption & Authentication (Symmetric):**
   - **Encrypt:** $c = \text{Encrypt}(K_{\text{enc}}, m)$.
   - **Authenticate:** $t = \text{MAC}(K_{\text{mac}}, c)$.
4. **Output:** Alice sends the ciphertext $\boldsymbol{(R, c, t)}$.

Elliptic Curve Integrated Encryption Scheme

Setup

Bob has his private key b and receives $(\boldsymbol{R}, \boldsymbol{c}, \boldsymbol{t})$.

1. **Key Generation (Asymmetric):**
   - Bob computes the *same* shared secret: $\boldsymbol{S} = \boldsymbol{bR}$.
   - (Since $bR = b(rP) = (br)P = r(bP) = rB$).
2. **Key Derivation (KDF):**
   - Bob derives the *exact same* keys:

$$K_{\text{enc}} \| K_{\text{mac}} = \text{KDF}(S_x)$$

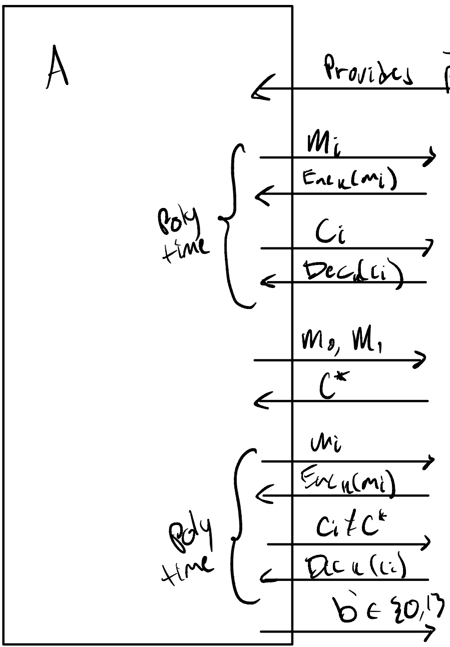
3. **Verify & Decrypt (Symmetric):**
   - **Verify FIRST:** Check if $t \stackrel{?}{=} \text{Verify}(K_{\text{mac}}, c)$.
   - If check fails $\implies$ **ABORT!**
   - If check passes $\implies$ **Decrypt:** $m = \text{Decrypt}(K_{\text{enc}}, c)$.

The Security Proof (sketch)

Our Security Goal: IND-CCA2

- **IND: Indistinguishability**. An attacker cannot distinguish between an encryption of m_0 and m_1.
- **CCA: Chosen Ciphertext Attack**. The scheme remains secure even if the attacker has access to a **decryption oracle**.

The CCA Security Game



challenger: public key to attacker

Attacker sends two messages, challenger flips a coin and sends one back encrypted $b = \{0,1\}$

A succeeds if $b' = b$ with non-negligible advantage

Why ECIES (with a MAC) is CCA-Secure

Why "ECIES-without-MAC" Fails

- A scheme without a MAC is often "malleable."
- An attacker could intercept $C = (R, c)$, modify it to $C' = (R, c')$, and send C' to the oracle.
- The oracle would decrypt c' (using the same key K_{enc}) and return m'.
- This m' might leak information about the original m.

Why ECIES (with a MAC) is CCA-Secure

Why ECIES (with a MAC) Succeeds

- This is an **Encrypt-then-MAC** construction.
- Attacker tries to forge a new ciphertext $C' = (R, c', t')$.
- They don't know K_{mac}, so they **cannot forge a valid tag** t' that matches their new c'.
- The decryption oracle (Bob) computes the *correct* tag $t_{correct} = \text{MAC}(K_{\text{mac}}, c')$.
- It sees $t' \neq t_{correct}$ and just returns **ABORT**.
- **The attacker learns nothing.** The oracle is useless to them.