AMERICAN UNIVERSITY OF BEIRUT

Applied Cryptography

CMPS 297AD/396AI

Fall 2025

Part 1: Provable Security

1.8: Elliptic Curves & Digital Signatures

Nadim Kobeissi

https://appliedcryptography.page

Section 1

Elliptic Curves: Theory

Elliptic-curve cryptography

- **Revolutionary introduction (1985):** Elliptic Curve Cryptography (ECC) transformed public-key cryptography.
- **Superior efficiency:** More powerful than RSA and classical Diffie-Hellman.
  - ECC with 256-bit key $\approx$ RSA with 4,096-bit key (security).
  - Significantly smaller key sizes for equivalent security.
- **Mathematical foundation:** Operations on points of elliptic curves.
  - Many curve types: simple/sophisticated, efficient/inefficient, secure/insecure.
- **Adoption timeline:**
  - Early 2000s: Standardization bodies.
  - 2005: OpenSSL support.
  - 2011: OpenSSH support.
- **Current applications:** HTTPS, mobile phones, blockchain (Bitcoin, Ethereum).
- **Based on ECDLP:** Elliptic Curve Discrete Logarithm Problem.

Why elliptic curve cryptography matters

- **Key size efficiency:** ECC provides equivalent security with much smaller keys.
  - 256-bit ECC key $\simeq$ 4,096-bit RSA key $\simeq$ 15,360-bit finite field DH.
  - Exponential security advantage as key sizes increase.
- **Performance benefits:**
  - Faster key generation, signing, and verification.
  - Lower computational overhead.
  - Reduced memory usage.
- **Bandwidth efficiency:** Smaller certificates, signatures, and key exchanges.
- **Mobile and IoT devices:** Critical for resource-constrained environments.
  - Limited battery life.
  - Constrained processing power.
  - Minimal storage capacity.

ECDH vs. finite field Diffie-Hellman

Traditional Finite Field DH:

- Works in multiplicative group $\mathbb{Z}_p^*$
- Security based on discrete log in $\mathbb{Z}_p^*$
- Requires large primes (2048+ bits)
- Key exchange: $g^{ab} \bmod p$

Elliptic Curve DH (ECDH):

- Works on elliptic curve group
- Security based on ECDLP[a]
- Requires smaller keys (256 bits)
- Key exchange: $a \cdot (b \cdot G)$

[a] Elliptic-curve discrete logarithm problem

ECDH advantages:

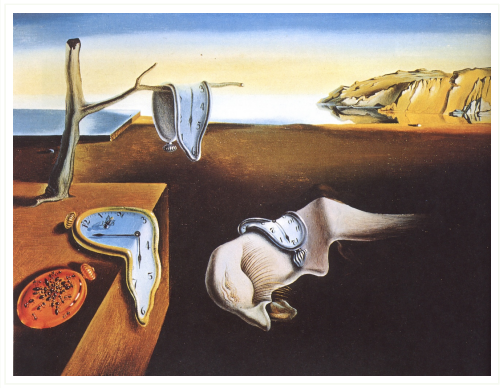
- **Efficiency:** 10-40x faster than finite field DH for equivalent security.
- **Scalability:** Performance gap widens with higher security levels.
- **Standards compliance:** Widely adopted (TLS 1.3, Signal Protocol, etc.)

Why finite field DH attacks don't work on ECDH

- **Different mathematical structures:**
  - Finite field DH: multiplicative group $\mathbb{Z}_p^*$ with multiplication.
  - ECDH: elliptic curve group with point addition (geometrically defined).
- **Index calculus attack limitation:**
  - Works on finite fields: factorize g^x using small primes.
  - Fails on elliptic curves: no equivalent of "small primes" for points.
  - Elliptic curve points cannot be "factorized" in the same way.
- **Subexponential vs. exponential algorithms:**
  - Finite field DL: subexponential algorithms exist (index calculus variants).
  - ECDLP: only exponential algorithms known (Pollard's rho, brute force).
- **Algebraic structure protection:**
  - Elliptic curve addition is more "rigid" than modular multiplication.
  - Geometric constraints prevent many algebraic manipulation attacks.
- **Result:** ECDH requires exponentially more work to break → smaller key sizes.

Very intuitively

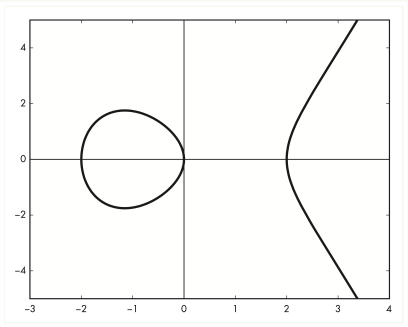
Finite-field Diffie-Hellman's structure allows for certain mathematically efficient attacks.



Let's make that structure "weirder" using elliptic curves and avoid these attacks.

What is an elliptic curve?

- **Definition:** An elliptic curve is a curve on a plane—a set of points with x- and y-coordinates.
- **Curve equations:** A curve's equation defines all the points that belong to that curve.
- **Examples of curves:**
  - $y = 3$: horizontal line with vertical coordinate 3
  - $y = ax + b$: straight lines (with fixed a, b)
  - $x^2 + y^2 = 1$: circle of radius 1 centered on origin
- **Key concept:** Points on a curve are (x, y) pairs that satisfy the curve's equation.



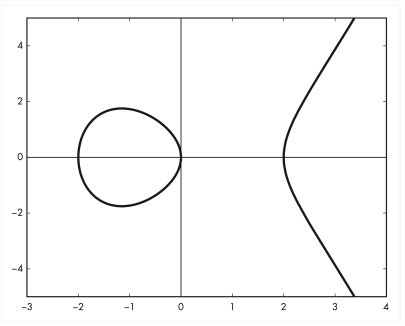
An elliptic curve with the equation $y^2 = x^3 - 4x$.
Source: Serious Cryptography

What is an elliptic curve?

- **Weierstrass form:** In cryptography, elliptic curves typically have equation:

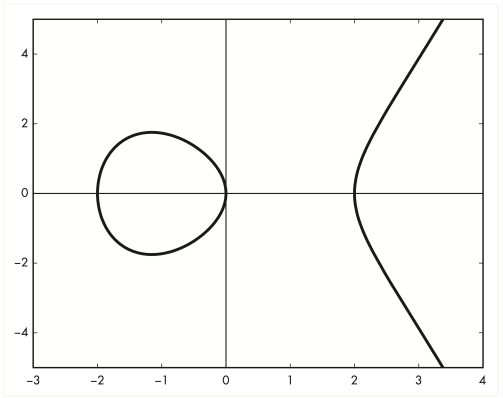
$$y^2 = x^3 + ax + b$$

- **Shape parameters:** Constants a and b define the shape of the curve.
- **Example:** The elliptic curve $y^2 = x^3 - 4x$
  - Here: $a = 0$ and $b = -4$
  - Creates a characteristic symmetric curve
- **Geometric properties:** Elliptic curves have special addition properties that make them useful for cryptography.

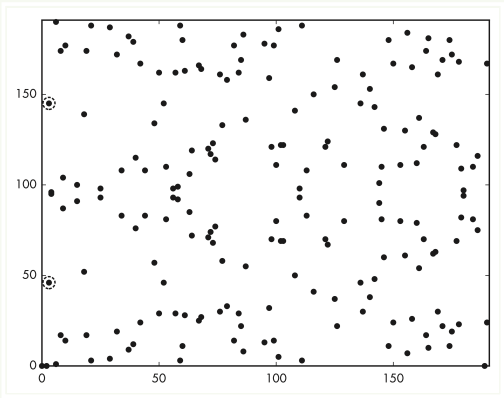


An elliptic curve with the equation $y^2 = x^3 - 4x$.
Source: Serious Cryptography

Elliptic curves over real numbers vs. integers



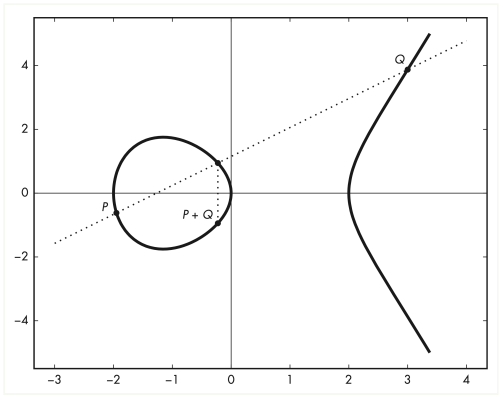
Elliptic curve over the real numbers (includes negative numbers, decimals)...



Same elliptic curve over the integers (only whole positive numbers)

Adding two points on an elliptic curve

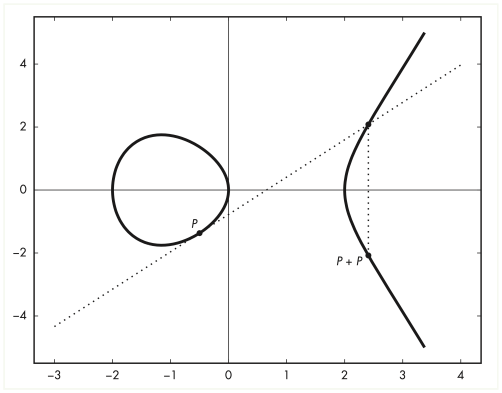
- Point addition follows a simple geometric process:
  - Draw the line that connects points P and Q.
  - Find the other point where this line intersects the curve.
  - R is the reflection of this intersection point with respect to the x-axis.
- **Result:** Point $P + Q$ has the same x-coordinate as the intersection but the inverse y-coordinate.



Adding two points on an elliptic curve.
Source: Serious Cryptography

Doubling a point on an elliptic curve

- **Point doubling:** When $P = Q$, adding P and Q is equivalent to computing $P + P = 2P$.
- **Geometric process:**
  - Can't draw a line between P and itself.
  - Instead, draw the line tangent to the curve at point P.
  - Find where this tangent line intersects the curve.
  - $2P$ is the reflection of this intersection point with respect to the x-axis.



Doubling a point on an elliptic curve.
Source: Serious Cryptography

Remember this?

We need an equivalent for elliptic curves

- The discrete logarithm problem:
  - Given a finite cyclic group G, a generator $g \in G$, and an element $h \in G$, find the integer x such that $g^x = h$
- In more concrete terms:
  - Let p be a large prime and let g be a generator of the multiplicative group $\mathbb{Z}_p^*$ (all nonzero integers modulo p).
  - Given:
    - $g \in \mathbb{Z}_p^*$, $h \in \mathbb{Z}_p^*$
    - Find $x \in \{0, 1, \ldots, p-2\}$ such that $g^x \equiv h \pmod{p}$
  - This problem is believed to be computationally hard when p is large and g is a primitive root modulo p.
    - "Believed to be" = we don't know of any way to do it that doesn't take forever, unless we have a strong, stable quantum computer (Shor's algorithm)

Group structure of elliptic curves

We need to define all these operations so that our elliptic curve have a group structure, allowing us to then use them as a new basis for Diffie-Hellman, and then do DH using point addition instead of modular multiplication.

- **Closure property:** If points P and Q belong to a curve, then $P + Q$ also belongs to the curve.
- **Associativity:** $(P + Q) + R = P + (Q + R)$ for any points P, Q, and R.
- **Identity element:** The *point at infinity* $\mathcal{O}$ such that $P + \mathcal{O} = P$ for any P.
- **Inverse elements:** Every point $P = (x_P, y_P)$ has an inverse $-P = (x_P, -y_P)$ such that $P + (-P) = \mathcal{O}$.
- Great! We have a group structure!

Elliptic curves over finite fields

- **Practical implementation:** Most elliptic curve cryptosystems work with coordinates modulo a prime p.
  - Coordinates are numbers in the finite field $\mathbb{Z}_p$.
  - Same geometric operations, but computed modulo p.
- **Security foundation:** Security depends on the *cardinality* (number of points) on the curve.
  - Analogous to how RSA security depends on the size of numbers used.
  - More points $\rightarrow$ harder discrete logarithm problem.
- **Curve cardinality:** The number of points depends on:
  - The specific curve equation (parameters a and b).
  - The prime modulus p.
  - Can be computed using specialized algorithms.
- **Why finite fields?** Infinite precision real numbers are impractical for computers.
  - Finite field arithmetic is exact and efficient.
  - Discrete structure enables cryptographic security.

The elliptic curve discrete logarithm problem (ECDLP)

- **Remember the original DLP:** Given g, h, and prime p, find x such that $g^x \equiv h \pmod{p}$.
- **ECDLP is the elliptic curve version:**
  - Given an elliptic curve and a base point G on that curve,
  - Given another point H on the same curve,
  - Find the integer k such that $k \cdot G = H$.
- **Why is this hard?**
  - Easy direction: Given k and G, computing $k \cdot G$ is efficient.
  - Hard direction: Given G and $H = k \cdot G$, finding k is very difficult.
  - No known efficient algorithms (except with quantum computers).

Diffie-Hellman key agreement over elliptic curves

- **Classical Diffie-Hellman recap:**
  - Alice picks secret a, computes $A = g^a$, sends A to Bob
  - Bob picks secret b, computes $B = g^b$, sends B to Alice
  - Both compute shared secret: $A^b = B^a = g^{ab}$
- **Elliptic Curve Diffie-Hellman (ECDH):**
  - Alice picks secret a, computes $A = a \cdot G$, sends A to Bob
  - Bob picks secret b, computes $B = b \cdot G$, sends B to Alice
  - Both compute shared secret: $a \cdot B = b \cdot A = ab \cdot G$
- **Key differences:**
  - Exponentiation $g^x \rightarrow$ Scalar multiplication $x \cdot G$
  - Modular arithmetic $\rightarrow$ Elliptic curve point operations
  - Generator $g \rightarrow$ Base point G

Section 2

Digital Signatures

Digital signatures with elliptic curves

- **Why elliptic curve signatures?** Same advantages as ECDH:
  - Smaller signatures for equivalent security.
  - Faster generation and verification.
  - Better performance on mobile/IoT devices.
- **Two main approaches:**
  - **ECDSA:** Elliptic Curve Digital Signature Algorithm (1990s).
  - **EdDSA:** Edwards-curve Digital Signature Algorithm (2011).
- **Real-world adoption:**
  - ECDSA: Bitcoin, Ethereum, TLS, SSH.
  - Ed25519: OpenSSH, Signal Protocol, many modern systems.
- **Key insight:** Replace RSA's modular exponentiation with elliptic curve point multiplication.

ECDSA: The established standard

- **Elliptic Curve Digital Signature Algorithm (ECDSA):**
  - NIST standardized in the early 1990s.
  - Elliptic curve version of the Digital Signature Algorithm (DSA).
  - Widely adopted in blockchain and web security.
- **Key components:**
  - Private key: secret number d
  - Public key: elliptic curve point $P = d \cdot G$
  - Base point G on agreed elliptic curve
- **Security foundation:** Based on ECDLP hardness.
- **Signature format:** Two numbers (r, s)
  - For 256-bit curves: 512-bit total signature size.
  - Much smaller than equivalent RSA signatures.

ECDSA signature generation

Input: Message M, private key d

1. **Hash the message:** $h = \text{Hash}(M)$
   - Use SHA-256, SHA-3, or similar
   - Interpret hash as number $h \in [0, n-1]$
2. **Generate random nonce:** Pick random $k \in [1, n-1]$
3. **Compute signature point:** $k \cdot G = (x, y)$
4. **Calculate r:** $r = x \bmod n$
5. **Calculate s:** $s = \frac{h + rd}{k} \bmod n$
6. **Output signature:** (r, s)

Critical requirement:

- Random k must be:
  - Cryptographically random
  - Different for every signature
  - Never reused
- **Reusing k = private key exposure!**

ECDSA signature verification

Input: Message M, signature (r, s), public key P

1. **Hash the message:** $h = \text{Hash}(M)$
2. **Compute modular inverse:** $w = \frac{1}{s} \bmod n$
3. **Calculate verification values:**
   - $u = h \cdot w \bmod n$
   - $v = r \cdot w \bmod n$
4. **Compute verification point:** $Q = u \cdot G + v \cdot P$
5. **Check signature:** Accept if $Q_x = r$

Why this works:

- Mathematical relationship:

$$\begin{aligned} Q &= u \cdot G + v \cdot P \\ &= u \cdot G + v \cdot d \cdot G \\ &= (u + vd) \cdot G \end{aligned}$$

- When signature is valid:

$$u + vd = k \bmod n$$

- So $Q = k \cdot G$, giving $Q_x = r$

EdDSA: The modern alternative

- **Background:** Built on Schnorr signatures (1989).
  - Schnorr's patent prevented adoption until 2008.
  - Edwards-curve DSA developed by Bernstein et al. (2011).
- **Key advantages over ECDSA:**
  - **Deterministic:** No random number generation during signing.
  - **Faster:** Both signing and verification.
  - **Simpler:** Cleaner mathematical structure.
  - **Safer:** Eliminates randomness-related vulnerabilities.
- **Design philosophy:** Avoid the pitfalls that plague ECDSA.
- **Most popular instance:** Ed25519 (based on Curve25519).

EdDSA signature generation

Key insight: Derive everything deterministically from private key and message.

Input: Message M, private key k (byte string)

1. **Expand private key:** $a \parallel h = \text{Hash}(k)$
   - a: actual signing scalar (first 256 bits)
   - h: randomness source (last 256 bits)
2. **Compute public key:** $A = a \cdot B$ (precomputed)
3. **Generate nonce deterministically:** $r = \text{Hash}(h \parallel M)$
4. **Compute signature point:** $R = r \cdot B$
5. **Compute signature scalar:** $S = r + \text{Hash}(R, A, M) \times a$
6. **Output signature:** (R, S)

Benefits:

- No randomness needed
- Same message = same signature
- Immune to bad RNG
- Faster (no modular inverse)

EdDSA signature verification

Input: Message M, signature (R, S), public key A

1. **Verify equation:** Check if:

$$S \cdot B = R + \text{HASH}(R, A, M) \cdot A$$

2. **Accept signature if equation holds**

Why this works:

- From signing: $S = r + \text{HASH}(R, A, M) \times a$
- So: $S \cdot B = (r + \text{HASH}(R, A, M) \times a) \cdot B$
- $= r \cdot B + \text{HASH}(R, A, M) \times a \cdot B$
- $= R + \text{HASH}(R, A, M) \times A$

Performance benefits:

- No modular inverse computation
- Two scalar multiplications (like ECDSA)
- Simpler arithmetic
- Better constant-time implementation

Ed25519: The practical implementation

- **Ed25519 = EdDSA + specific parameters:**
  - Twisted Edwards curve based on Curve25519
  - SHA-512 as hash function
  - Optimized base point for efficiency
- **Performance characteristics:**
  - Signing: 40-90 microseconds (modern CPUs)
  - Verification: 100-200 microseconds
  - 64-byte signatures (512 bits)
- **Security level:** 128 bits (equivalent to 3072-bit RSA)
- **Adoption milestones:**
  - 2011: Initial specification
  - 2017: RFC 8032 standardization
  - 2023: Added to NIST FIPS 186-5

ECDSA vs. Ed25519: The comparison

ECDSA:

- **Pros:**
  - Established standard (1990s)
  - Wide library support
  - Blockchain industry standard
- **Cons:**
  - Requires secure randomness
  - Slower verification
  - Complex implementation
  - Vulnerable to bad RNG

Ed25519:

- **Pros:**
  - Deterministic signing
  - Faster performance
  - Simpler implementation
  - Better security properties
- **Cons:**
  - Newer standard
  - Some validation inconsistencies
  - Less blockchain adoption

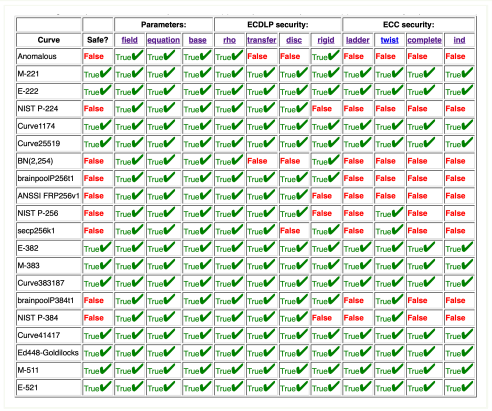
Recommendation: Use Ed25519 for new projects unless ECDSA is specifically required.

Section 3

Elliptic Curves: Practice

Choosing the right elliptic curve

- **Not all elliptic curves are created equal:** The mathematical structure of the curve directly impacts cryptographic security.
- **Security implications:** Poor curve choice can make ECDLP much easier to solve.
- **In practice:** You'll use established curves, but understanding what makes a curve safe helps you:
  - Choose among available options.
  - Better understand associated risks.
  - Evaluate new curve proposals.



		Parameters:			ECDLP security:				ECC security:			
Curve	Safe?	field	equation	base	rho	transfer	disc	rigid	ladder	twist	complete	ind
Anomalous	False	True✔	True✔	True✔	True✔	False	False	True✔	False	False	False	False
M-221	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
E-222	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
NIST P-224	False	True✔	True✔	True✔	True✔	True✔	True✔	False	False	False	False	False
Curve1174	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
Curve25519	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
BN(2,254)	False	True✔	True✔	True✔	True✔	False	False	True✔	False	False	False	False
brainpoolP256t1	False	True✔	True✔	True✔	True✔	True✔	True✔	True✔	False	False	False	False
ANSSI FRP256v1	False	True✔	True✔	True✔	True✔	True✔	True✔	False	False	False	False	False
NIST P-256	False	True✔	True✔	True✔	True✔	True✔	True✔	False	False	False	False	False
secp256k1	False	True✔	True✔	True✔	True✔	True✔	False	True✔	False	True✔	False	False
E-382	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
M-383	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
Curve383187	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
brainpoolP384t1	False	True✔	True✔	True✔	True✔	True✔	True✔	True✔	False	True✔	False	False
NIST P-384	False	True✔	True✔	True✔	True✔	True✔	True✔	False	False	True✔	False	False
Curve41417	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
Ed448-Goldilocks	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
M-511	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔
E-521	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔	True✔

Elliptic curves have many distinct and complex security criteria.
Source: `https://safecurves.cr.yp.to`

Criteria for safe elliptic curves

- **Group order security:** The number of points on the curve shouldn't factor into small numbers.
  - If the order has small factors, ECDLP becomes much easier.
  - Attackers can use algorithms like Pohlig-Hellman to exploit small factors.
- **Addition formula consistency:** Unified addition laws are preferred.
  - Some curves require different formulas for $P + Q$ vs. $P + P$ (doubling).
  - Timing differences between these operations can leak information.
  - Secure curves use the same formula for all additions.
- **Parameter transparency:** The origin of curve parameters should be clearly explained.
  - Unknown parameter origins raise suspicion of backdoors.
  - "Nothing up my sleeve" numbers increase trust.

NIST curves: The establishment standard

- **Official standardization:** NIST standardized several curves in FIPS 186 (2000).
  - "Recommended Elliptic Curves for Federal Government Use"
  - Five prime curves working modulo prime numbers.
  - Ten binary polynomial curves (rarely used today).
- **Most popular: P-256**
  - Works modulo $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
  - Equation: $y^2 = x^3 - 3x + b$ (256-bit b parameter)
  - Other sizes: P-192, P-224, P-384, P-521 (yes, 521 not 512!)
- **Wide adoption:** Used in TLS, government systems, many commercial applications.

The NIST controversy: Suspicious constants

- **The problem:** Only the NSA knows the true origin of the b coefficient in NIST curves.
- **NSA's explanation:** b results from hashing a "random-looking" constant with SHA-1.
  - P-256's b comes from: `c49d3608 86e70493 6a6678e1 139d26b7 819f7e90`
  - But why this particular constant? Nobody knows.
- **Community response:**
  - Most experts don't believe the curves hide backdoors.
  - But the lack of transparency creates suspicion.
  - Led to development of alternative curves with transparent parameters.
- **Post-Snowden era:** Increased scrutiny of NSA-designed cryptographic standards.

Curve25519: The performance revolution

- **Created by Daniel J. Bernstein (2006):** Motivated by performance and security concerns.
- **Performance advantages:**
  - Faster than NIST curves.
  - Shorter keys for equivalent security.
  - Optimized for software implementation.
- **Security improvements:**
  - No suspicious constants—all parameters have clear origins.
  - Unified addition formula (same for $P + Q$ and $P + P$).
  - Resistant to timing attacks.
- **Mathematical form:** $y^2 = x^3 + 486662x^2 + x$
  - Works modulo $2^{255} - 19$ (closest prime to 2^{255}).
  - Coefficient 486662 is the smallest integer satisfying security criteria.

Curve25519: From rebel to standard

- **Widespread adoption:**
  - WhatsApp end-to-end encryption
  - TLS 1.3 key exchange
  - OpenSSH connections
  - Signal Protocol
  - Many cryptocurrency systems
- **Official recognition:** Added to NIST-approved curves in February 2023.
  - SP 800-186: “Recommendations for Discrete Logarithm-based Cryptography”
  - Took 17 years for official government approval!
- **Trust through transparency:** Clear parameter origins make Curve25519 more trustworthy than NIST curves.
- **Related: Ed25519** for digital signatures using the same curve.

Other curves in the ecosystem

- **Legacy national standards:**
  - **ANSSI curves (France):** Constants of unknown origin, no unified addition.
  - **Brainpool curves (Germany):** Similar issues to ANSSI curves.
- **Modern alternatives:**
  - **Curve41417:** Variant of Curve25519 with higher security (200 bits).
  - **Ed448-Goldilocks:** 448-bit curve (RFC 8032, 2014).
  - **Aranha et al. curves:** Six high-security curves (rarely used).
- **Ristretto initiative:** Technique for safe point representation.
  - Constructs prime-order groups from non-prime-order curves.
  - Eliminates certain structural risks.

Practical curve selection guidance

- **For new projects:** Use Curve25519/Ed25519
  - Excellent performance and security.
  - Transparent parameter generation.
  - Wide library support.
  - Now NIST-approved for government use.
- **For government/compliance:** NIST P-256 is still widely accepted
  - Required by some standards and regulations.
  - Well-audited implementations available.
  - Despite parameter concerns, no known weaknesses.
- **Avoid:** Legacy curves with unknown parameter origins
  - ANSSI, Brainpool curves.
  - Curves without unified addition laws.
- **Future-proofing:** Consider post-quantum alternatives for long-term security.

How things can go wrong

- **ECC complexity brings risks:** More parameters than RSA create a larger attack surface.
- **Implementation vulnerabilities:**
  - Side-channel attacks on big-number arithmetic.
  - Timing attacks when computation time depends on secret values.
  - Point validation failures.
- **Design-level vulnerabilities:**
  - Bad randomness in signature generation.
  - Invalid curve attacks on key exchange.
  - Inconsistent validation rules across implementations.
- Let's examine three major categories of ECC vulnerabilities.

ECDSA with bad randomness

- **ECDSA signing requires randomness:** Each signature uses a secret random number k.

$$s = \frac{h + rd}{k} \bmod n$$

- **The catastrophic mistake:** Reusing the same k for two different messages.
- **Attack scenario:** If k is reused:
  - Attacker gets: $s_1 = \frac{h_1 + rd}{k}$ and $s_2 = \frac{h_2 + rd}{k}$
  - Compute: $s_1 - s_2 = \frac{h_1 - h_2}{k}$
  - Recover randomness: $k = \frac{h_1 - h_2}{s_1 - s_2}$
  - Recover private key: $d = \frac{sk - h}{r}$
- **Why this is devastating:** Complete private key recovery from just two signatures.
- **Prevention:** Always use cryptographically secure random number generation.

Case study: PlayStation 3 hack (2010)

- **The vulnerability:** Sony's PlayStation 3 reused the same k value to sign different games.
- **Discovery:** fail0verflow team at 27th Chaos Communication Congress.
- **Attack process:**
  - Collected ECDSA signatures from multiple PS3 games.
  - Noticed identical r values (indicating same k).
  - Applied the mathematical attack to recover Sony's signing key.
- **Consequences:**
  - Attackers could sign any program to run on PS3.
  - Homebrew software and piracy became possible.
  - Sony had to revoke and update their entire signing infrastructure.
- **Lesson:** Even major companies can make fundamental cryptographic mistakes.

Invalid curve attacks

- **The vulnerability:** ECDH implementations that don't validate input points.
- **Mathematical insight:** Point addition formulas don't use the b coefficient:

$$P + Q \text{ only depends on coordinates of } P, Q \text{ and coefficient } a$$

- **Attack scenario:**
  - Alice and Bob agree on curve and base point G.
  - Bob sends legitimate public key bG.
  - Alice sends point P from a *different, weaker* curve.
  - Bob computes "shared secret" bP on the wrong curve.
- **Why this works:** Addition formulas work the same way on the wrong curve.
- **Attacker's advantage:** Choose P from a curve with weak discrete logarithm.

Invalid curve attack: The mathematics

- **Attacker's strategy:** Choose point P with small order on a weak curve.
  - Small order means $kP = \mathcal{O}$ for relatively small k.
  - Bob computes bP, which also has small order.
- **Attack execution:**
  - Bob believes he computed shared secret bP.
  - He hashes bP and uses result as encryption key.
  - Since bP belongs to small subgroup, attacker can brute-force it.
- **Real-world example:** Found in TLS-ECDH implementations (2015).
  - Paper: "Practical Invalid Curve Attacks on TLS-ECDH"[a]
  - Jager, Schwenk, and Somorovsky
- **Prevention:** Always validate that points satisfy the correct curve equation.

[a] https://appliedcryptography.page/papers/invalid-curve.pdf

Invalid curve attack prevention

- **Point validation:** Before using any received point $P = (x, y)$:

$$\text{Check: } y^2 \stackrel{?}{=} x^3 + ax + b \pmod{p}$$

- **Additional checks:**
  - Verify point is not the point at infinity.
  - Ensure coordinates are in valid range $[0, p-1]$.
  - Check point has correct order (belongs to right subgroup).
- **Implementation note:** Many libraries now perform validation automatically.
- **Defense in depth:** Use curves with prime order (like Curve25519).
  - Eliminates small subgroup attacks entirely.
  - Even invalid points can't exploit subgroup structure.

Ed25519 validation inconsistencies

- **Expectation:** One standard should mean identical behavior across implementations.
- **Reality:** Ed25519 implementations have different validation criteria.
- **The problem:** RFC 8032 doesn't fully specify validation requirements.
  - How to validate signature point R.
  - How to validate public key point A.
  - How to verify the signature equation.
- **Research findings:** Henry de Valence analyzed 15 Ed25519 implementations.[a]
  - Each had different validation criteria.
  - Same signature could be valid in one implementation, invalid in another.

[a] https://hdevalence.ca/blog/2020-10-04-its-25519am/

Real-world impact of validation differences

- **Consensus failures:** In blockchain networks:
  - Some nodes accept a signature, others reject it.
  - Breaks consensus protocol assumptions.
  - Can lead to network splits or transaction inconsistencies.
- **Interoperability issues:** Systems using different libraries may disagree.
- **Security implications:** Inconsistent validation can enable attacks.
  - Malleability attacks.
  - Signature forgery in edge cases.
- **The solution:** Standardization efforts like ZIP-215 (Zcash) aim to:
  - Specify exact validation rules.
  - Ensure all implementations behave identically.
  - Prevent consensus failures.
- **Lesson:** Cryptographic standards must be completely unambiguous.

Library selection: The ecosystem landscape

- **Don't implement ECC from scratch:** Cryptographic implementations require years of hardening.
- **Rust ecosystem:**
  - `ring`: Fast, audited, used by major companies.
  - `p256`, `k256`: RustCrypto pure-Rust implementations.
  - `curve25519-dalek`: Ed25519/X25519 with extensive validation.
- **Go ecosystem:**
  - `crypto/elliptic`: Standard library (NIST curves).
  - `golang.org/x/crypto/curve25519`: Official X25519 implementation.
  - `filippo.io/edwards25519`: Modern Ed25519 with clear APIs.
- **Selection criteria:**
  - Active maintenance and security updates.
  - Independent security audits.
  - Constant-time guarantees.
  - Clear documentation and examples.

Performance considerations in practice

- **Scalar multiplication is the bottleneck:** Operations like $k \cdot G$ dominate runtime.
- **Precomputation strategies:**
  - Store multiples of base point: $G, 2G, 4G, 8G, \ldots$
  - Sliding window methods for arbitrary points.
  - Trade memory for speed.
- **Coordinate systems matter:**
  - Affine coordinates: Simple but require expensive modular inverse.
  - Jacobian coordinates: Avoid inverse, faster for repeated operations.
  - Montgomery ladders: Optimal for X25519-style protocols.
- **Real-world impact:**
  - TLS handshake time directly affects user experience.
  - Mobile devices: battery life and thermal constraints.
  - IoT devices: limited computational resources.

Constant-time implementation: Why it matters

- **The threat:** Attackers can measure timing differences to extract secrets.
- **Vulnerable patterns in ECC:**
  - Conditional branches based on secret bits
  - Variable-time modular arithmetic
  - Memory access patterns that depend on secret data
- **Example: Scalar multiplication timing**
  - Binary method: `if (bit == 1) result += point`
  - Timing reveals which bits are 1 vs 0
  - After enough measurements, attacker recovers private key
- **Defense:** Always perform the same operations regardless of secret values.
- **Modern libraries handle this:** But you need to choose libraries that guarantee constant-time behavior.

Memory management and sensitive data

- **The problem:** Private keys in memory can be extracted by attackers.
- **Attack vectors:**
  - Memory dumps during crashes.
  - Swap files writing secrets to disk.
  - Cold boot attacks on RAM.
  - Process memory scanning.
- **Defense strategies:**
  - Zero memory immediately after use.
  - Use protected memory (mlock/VirtualLock).
  - Hardware security modules (HSMs) for high-value keys.
  - Minimize lifetime of secrets in memory.
- **Language-specific considerations:**
  - Rust: `zeroize` crate for secure memory clearing.

Testing elliptic curve implementations

- **Standard test vectors:** Use RFC and NIST test cases to verify correctness.
- **Cross-implementation testing:**
  - Generate signatures with one library, verify with another.
  - Perform ECDH with different implementations.
  - Ensure interoperability across programming languages.
- **Edge case testing:**
  - Point at infinity handling.
  - Invalid curve points.
  - Malformed signature formats.
  - Zero and maximum values.
- **Property-based testing:**
  - Verify mathematical properties: $P + Q = Q + P$
  - Test with random inputs within valid ranges.
  - Ensure operations always produce valid outputs.

Real-world case study: WhatsApp's implementation

- **Challenge:** Secure messaging for 2+ billion users across diverse devices.
- **Solution:** Signal Protocol with Curve25519 and Ed25519.
- **Implementation details:**
  - X25519 for key agreement (ECDH).
  - Ed25519 for identity key signatures.
  - Custom optimizations for mobile platforms.
  - Cross-platform C library for consistency.
- **Engineering considerations:**
  - Battery life optimization on mobile devices.
  - Constant-time implementation to prevent side-channel attacks.
  - Extensive testing across iOS, Android, and desktop platforms.
  - Regular security audits by external firms.
- **Lessons:** Real world requires balancing security, performance, and compatibility.

Real-world case study: TLS 1.3 performance

- **Challenge:** Replace RSA key exchange with elliptic curve alternatives.
- **Implementation impact:**
  - X25519 ECDH: 40-100x faster than 2048-bit RSA key exchange.
  - Smaller certificates reduce network overhead.
  - Enables features like 0-RTT handshakes.
- **Engineering challenges solved:**
  - Constant-time implementation in BoringSSL.
  - Optimized assembly for common architectures.
  - Fallback implementations for edge cases.

Common implementation pitfalls

- **Pitfall 1: Poor random number generation**
  - Using `rand()` instead of cryptographic RNG.
  - Not seeding random generators properly.
  - Reusing random values (PlayStation 3 scenario).
- **Pitfall 2: Skipping input validation**
  - Not checking if points are on the correct curve.
  - Accepting points at infinity without proper handling.
  - Missing range checks on coordinates.
- **Pitfall 3: Side-channel vulnerabilities**
  - Conditional operations based on secret data.
  - Variable memory access patterns.
  - Timing differences in error handling.
- **Prevention:** Use audited libraries, follow security guidelines, test extensively.

Best practices for ECC implementation

- **Library selection:**
  - Choose libraries with security audit history.
  - Prefer constant-time implementations.
  - Ensure active maintenance and updates.
- **Development practices:**
  - Use standard curves (avoid custom parameters).
  - Implement comprehensive input validation.
  - Clear sensitive data from memory.
  - Use secure random number generation.
- **Testing and deployment:**
  - Test with standard vectors and edge cases.
  - Perform interoperability testing.
  - Monitor for timing analysis vulnerabilities.
  - Plan for cryptographic agility (algorithm migration).

Digital signatures: real-world adoption patterns

- **ECDSA dominance:**
  - Bitcoin, Ethereum, most cryptocurrencies.
  - TLS certificates (still common).
  - Legacy enterprise systems.
- **Ed25519 growth:**
  - OpenSSH default since 2014.
  - Signal Protocol messaging.
  - Modern certificate authorities.
  - New blockchain projects (Solana, etc.)
- **Migration considerations:**
  - Interoperability with existing systems.
  - Library availability in your ecosystem.
  - Compliance requirements.
  - Performance requirements.

Digital signatures: practical implementation guidelines

- **For ECDSA implementations:**
  - Use cryptographically secure random number generator.
  - Never reuse nonce values.
  - Implement constant-time operations.
  - Validate all input points.
- **For Ed25519 implementations:**
  - Follow RFC 8032 specification carefully.
  - Handle validation edge cases consistently.
  - Use established libraries (libsodium, etc.)
- **General best practices:**
  - Don’t implement from scratch.
  - Use constant-time libraries.
  - Test with standard vectors.

Looking forward: Implementation challenges

- **Post-quantum transition:** ECC implementations need migration paths.
  - Hybrid classical/post-quantum systems.
  - Algorithm negotiation mechanisms.
  - Backward compatibility requirements.
  - Discussed in a future class topic!
- **Formal verification:** Mathematical proofs of implementation correctness.
  - Projects like Cryspen's HAX and Libcrux generate verified code.
  - Higher assurance for critical applications.
  - Trade-off between verification effort and deployment flexibility.
  - Discussed in a future class topic!

AMERICAN UNIVERSITY OF BEIRUT

Applied Cryptography

CMPS 297AD/396AI

Fall 2025

Part 1: Provable Security

1.8: Elliptic Curves & Digital Signatures

Nadim Kobeissi

https://appliedcryptography.page