UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

DÉPARTEMENT DES SCIENCES ADMINISTRATIVES

WORKING PAPER NO. 34

Beyond the Square-Root Barrier: Automated Discovery of Algebraic and Algorithmic Shortcuts in Bitcoin Cryptography

A hypothesis-driven, reproducible study of secp256k1, ECDSA, BIP340 Schnorr and SHA-256 with machine-checked identities and controlled negative results

Simon-Pierre Boucher

Département des sciences administratives
Université du Québec en Outaouais

simon-pierre.boucher@uqo.ca

Gatineau – Pavillon Alexandre-Taché
283, boulevard Alexandre-Taché
Gatineau, Québec, Canada J9A 1L8

November 2026

Version 1.0

Abstract

Placeholder abstract.

Keywords: elliptic curve discrete logarithm; secp256k1; Pollard rho; endomorphisms; ECDSA; Schnorr signatures; BIP340; SHA-256; SAT-based cryptanalysis; automated mathematical discovery; Lean 4; reproducibility

JEL Classification: C63, C88, D83 (MSC 2020: 94A60, 11T71, 11G20, 68V15, 68W30)

Contents

1 Introduction **6**

2 Cryptographic Foundations of Bitcoin **8**

2.1 Finite-field arithmetic 8
2.2 Elliptic curves and the group law 10
2.3 Scalar multiplication 11
2.4 Group order, Hasse's theorem and subgroup structure 11
2.5 Isomorphisms, twists and the j-invariant 11
2.6 Endomorphisms and complex multiplication 11
2.7 Scalar decomposition 12
2.8 The discrete-logarithm problem and generic-group complexity 12
2.9 What "beyond the square-root barrier" can and cannot mean 12

3 Literature Review and Research Gaps **13**

3.1 The elliptic-curve discrete logarithm 13
3.2 Signatures 14
3.3 SHA-256 14
3.4 Automated discovery 15
3.5 Research gaps and what the mandate already settles 15

4 Mathematical Structure of secp256k1 **16**

4.1 Verified domain parameters 16
4.2 Why the special form is interesting 16
4.3 The GLV lattice is the ideal $(\bar{\pi} - 1)$ 18
4.4 Orbits, equivalence classes and the limits of the automorphism trick 19
4.5 Algebraic attacks: an exact exclusion table 19

5 Generic and Algebraic Discrete-Logarithm Algorithms **20**

5.1 Baby-step giant-step 20
5.2 Pollard's rho and r-adding walks 20
5.3 Distinguished points and parallel collision search 21
5.4 Equivalence classes: negation map and automorphisms 21
5.5 Pollard's kangaroo for intervals 21
5.6 Pohlig–Hellman 22
5.7 Preprocessing (non-uniform) algorithms 22

5.8 Lower bounds and what they rule out . . . 22
5.9 Cost model and empirical complexity . . . 22

6 ECDSA and Schnorr: Security and Computational Structure 23
6.1 ECDSA . . . 23
6.2 BIP340 Schnorr . . . 24
6.3 Batch verification and multi-scalar multiplication . . . 24
6.4 What can be simplified and what cannot (B3, B4) . . . 25

7 SHA-256: Mathematical Structure and Reduced-Round Analysis 25
7.1 Construction . . . 25
7.2 Elementary structure . . . 26
7.3 Generic bounds and the meaning of reduced-step results . . . 26
7.4 What we measure . . . 27

8 Autonomous Mathematical Discovery Framework 27

9 Experimental Design 30
9.1 Environment . . . 30
9.2 Baselines and validation . . . 30
9.3 Synthetic ECDLP instances . . . 31
9.4 Measurement and statistics . . . 32
9.5 Signature and SHA-256 protocols . . . 32

10 Computational Results 32
10.1 Axis A: algebraic structure (H01–H07) . . . 32
10.2 Axis A/B: ECDLP baselines and scaling (H08) . . . 32
10.3 Axis A: automorphism classes on $j = 0$ curves (H04) . . . 36
10.4 Axis B: automated search over iteration functions (H09) . . . 36
10.5 Axis B: parallel efficiency and time–memory trade-offs (H10, H11) . . . 39
10.6 Axis B: preprocessing (H12) . . . 41
10.7 Axis B: signature benchmarks (H13, H14) . . . 41

11 Formal Results and Negative Findings 42
11.1 Machine-checked statements . . . 42
11.2 Negative findings . . . 45

12 Implications for Cryptographic Security 45

13 Limitations **47**

14 Conclusion **48**

Appendices **61**

A Reproducibility statement **61**

B Computational environment **61**

C Full secp256k1 constants **61**

1. Introduction

The security of Bitcoin reduces to three mathematical assumptions: that discrete logarithms in the group of rational points of the curve secp256k1 cost about $\sqrt{n} \approx 2^{128}$ group operations, that SHA-256 is preimage- and collision-resistant at the generic 2^{256} and 2^{128} levels, and that ECDSA and BIP340 Schnorr signatures are unforgeable under these assumptions. The first of these is a "square-root barrier": in the generic-group model it is a theorem (Shoup, 1997), and the only way past it is an algorithm that exploits the specific representation of the group. secp256k1 has an unusually rich representation—a generalised-Mersenne prime, the equation $y^2 = x^3 + 7$ with j-invariant 0, complex multiplication by the Eisenstein integers and an automorphism group of order six—and this has fed a persistent folklore that the curve's "special structure" might hide a shortcut. The known mathematics says the structure yields constant factors (a $\sqrt{6}$ reduction in Pollard's rho, a halving of doublings in scalar multiplication) and nothing more; the known computational records stop at 112–113 bits.

This paper asks whether a systematic, automated exploration of that structure—and of the corresponding structure in the signature schemes and in SHA-256—can find anything the literature has missed, and it answers the question with the discipline the question deserves. We set up a six-stage discovery pipeline (verified literature database, structured hypotheses, exact symbolic checks, seeded experiments with provenance, machine-checked statements in Lean 4, and a six-way classification of every outcome), executed it autonomously under a written mandate, and let it evaluate 25 hypotheses across four axes: the algebraic structure of secp256k1 (Axis A), the complexity of generic and preprocessing discrete-logarithm algorithms (Axis B), reduced-step SHA-256 (Axis C), and the methodology itself (Axis D). The mandate's first instruction was "do not assume such shortcuts exist"; the paper's first result is that none were found, and that the absence is documented quantitatively rather than asserted.

Contributions. The contributions are of four kinds, and we label each with the classification it received.

Algebraic structure (known results reproduced, with one explicit form we did not find written down). We verify every secp256k1 parameter and derived invariant twice with exact arithmetic (Python and PARI/GP), prove that $\mathrm{End}(E) = \mathbb{Z}[\omega]$ leaves no endomorphism to exploit beyond the six automorphisms, and show that the GLV lattice is the Eisenstein ideal $(\bar{\pi} - 1)$, so that the constants hard-coded in `libsecp256k1` are literally $a_1 = (t - f)/2 - 1$, $b_1 = -f$, $a_2 = a_1 - b_1$, $b_2 = a_1$ with $4p = t^2 + 3f^2$ (Proposition 4.2). The pipeline's identity search found this on its own (as "$t^2 + 3b_1^2 = 4p$") among 22 exact identities and 0 false positives. We prove that the automorphism orbits admit no compatible group structure (Proposition 4.3), closing the most natural "beyond $\sqrt{6}$"

idea, and we quantify why summation-polynomial index calculus does not beat $\sqrt{n}$ over prime fields.

Algorithmic complexity (known results reproduced with precise constants; one rejected shortcut). A native engine solved 6 000 synthetic ECDLP instances of 20–56 bits, all verified. The fitted exponents are 0.48–0.50; the rho constant is $1.254\sqrt{n}$ against the random-mapping prediction 1.2533; the order-6 automorphism classes reduce iterations by $2.42\times$ ($\sqrt{6} = 2.449$) but, in our implementation, save no wall-clock time because canonicalisation costs as much as a group addition—a point the literature's record computations solve with engineering we did not replicate. Parallel collision search reaches $22\times$ on 28 threads; time–memory curves match $m + n/(2m)$ for BSGS and show the two-sided optimum of distinguished-point rho; discrete logarithms with preprocessing cost $1.5\sqrt{n/T}$ online after T stored points. An automated sweep over 25 iteration-function designs found nothing below the random-mapping constant (rejected shortcut). Batch verification of BIP340 signatures reaches $3.1\times$ over single verification in the same arithmetic and saturates for an identified reason.

SHA-256 (empirical observations and known results reproduced). Reduced-step experiments quantify diffusion and statistical regularity as functions of the step count, calibrate generic SAT and SMT encodings for preimage and collision search against the published dedicated-tool records, and reproduce the 9-step local collision in both the linearised and the real model; nothing bears on the 64-step function (Sections 7 and 10).

Methodology and formal verification. Bit-level identities that implementations rely on—the optimised forms of Ch and Maj, linearity and bijectivity of the Σ functions, the bijectivity of the SHA-256 step and the invertibility of the message schedule—and the abstract-group facts behind ECDSA malleability and Schnorr correctness are machine-checked in Lean 4 (Section 11). The pipeline's record (Table **??**) is a majority of known results reproduced with quantified constants, two rejected conjectures with proofs or counterexamples, several empirical observations, and no claim of novelty beyond exposition; we argue that this is the correct output of an autonomous agent on this problem, and that the discipline—operation counts next to timings, two implementations per component, provenance on every number, formal statements kept separate from informal ones—is the transferable contribution.

Scope and integrity. All attacks were run on synthetic curves and keys generated by the experiments; no real key, address or blockchain datum was used; full-strength primitives were analysed only through legitimate implementations and exact algebra. The repository releases every script, result and figure source with a reproducibility statement (Appendix A).

Organisation. Section 2 fixes the mathematical foundations and the meaning of the barrier; Section 3 reviews the literature and the research gaps; Sections 4–7 develop the theory of the three axes with the algebraic results; Section 8 describes the discovery pipeline; Sections 9 and 10 give the experimental design and results; Section 11 lists the formal results and the negative findings; Sections 12 and 13 discuss implications and limitations; Section 14 concludes.

2. Cryptographic Foundations of Bitcoin

Bitcoin's security rests on three computational assumptions that are structurally unrelated: the hardness of the discrete-logarithm problem in the group of rational points of one specific elliptic curve, secp256k1; the preimage and collision resistance of SHA-256 (used directly, doubled as SHA-256d in the proof of work, composed with RIPEMD-160 in legacy addresses, and in the tagged-hash form of BIP340/BIP341); and the unforgeability of two signature schemes, ECDSA and BIP340 Schnorr, which reduce—under different assumptions and with very different proof structures—to the first two. Figure 1 summarises the dependency structure; the dashed box contains the three hardness assumptions whose mathematical structure is the subject of this paper.

This section fixes notation and states the mathematical facts the rest of the paper builds on. We give proofs or exact verification scripts for the facts that are specific to secp256k1 (Section 4); classical theorems are cited to the standard references (Silverman, 2009; Washington, 2008; Hankerson et al., 2004; Galbraith, 2012).

2.1. Finite-field arithmetic

Let p be an odd prime and $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. Addition, subtraction and multiplication are integer operations followed by reduction modulo p; inversion uses the extended Euclidean algorithm or Fermat's identity $a^{-1} = a^{p-2}$. For the secp256k1 prime $p = 2^{256} - 2^{32} - 977$ the reduction can exploit the special form: writing a 512-bit product as $c = c_1 2^{256} + c_0$ gives $c \equiv c_0 + c_1(2^{32} + 977) \pmod{p}$, and two such folds reduce to $[0, 2p)$. This is a generalised-Mersenne reduction in the sense of Solinas (1999); it is a constant-factor implementation advantage, not an algebraic weakness—the multiplicative group $\mathbb{F}_p^*$ has order $p - 1 = 2 \cdot 3 \cdot 7 \cdot 13441 \cdot q$ with q a large prime, and no structure of $\mathbb{F}_p$ enters any known attack on the curve. Square roots are needed to decompress points: since $p \equiv 3 \pmod 4$, $\sqrt{a} = a^{(p+1)/4}$ whenever a is a quadratic residue. All our implementations use exact integer arithmetic (Python integers, PARI/GP, Rust `u128` products); floating point is never used for a field operation.

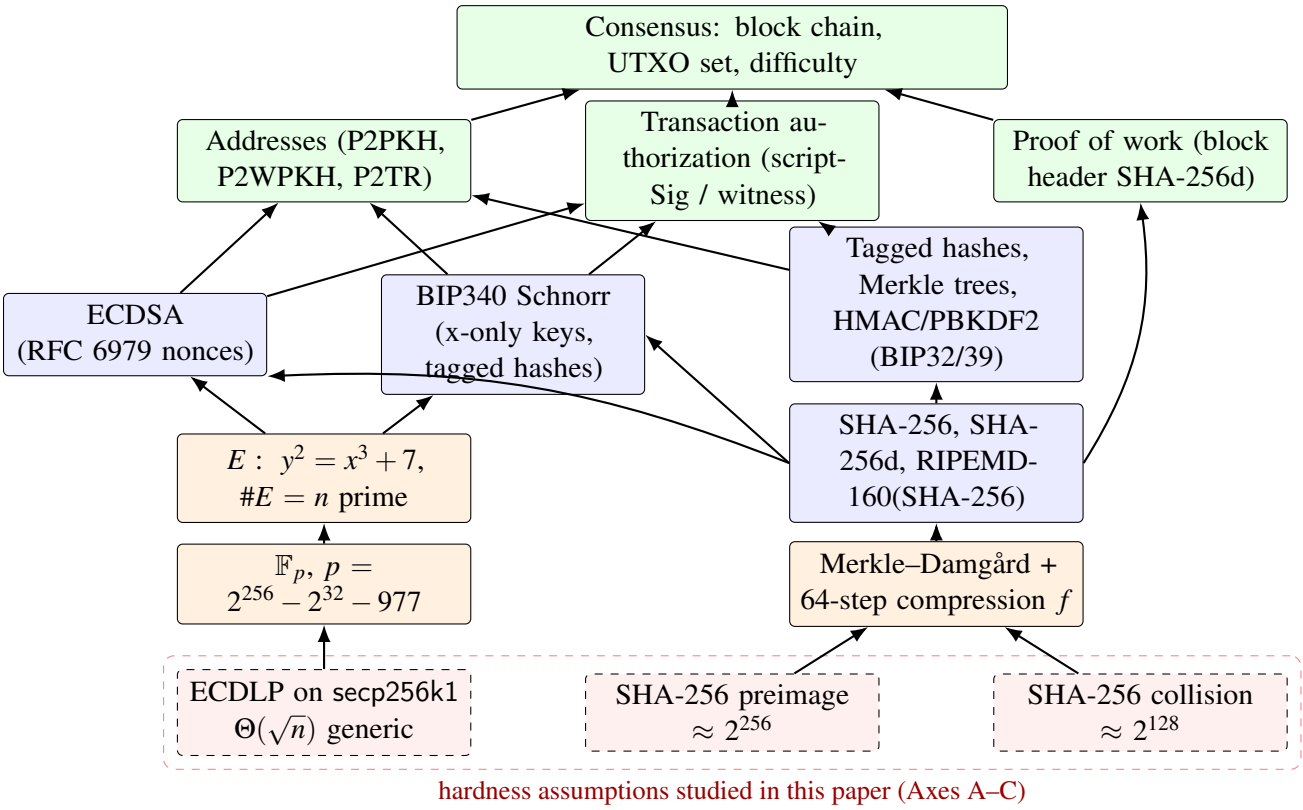


Figure 1. Cryptographic dependency map of Bitcoin (illustrative diagram). Arrows point from a component to the components that depend on it. Research Axes A–C of this paper concern the three dashed hardness assumptions; Axis D concerns the method by which such assumptions can be probed automatically.

2.2. Elliptic curves and the group law

An elliptic curve over $\mathbb{F}_p$ ($p > 3$) in short Weierstrass form is

$$E: \quad y^2 = x^3 + ax + b, \qquad a, b \in \mathbb{F}_p, \quad \Delta = -16(4a^3 + 27b^2) \neq 0. \tag{1}$$

The set $E(\mathbb{F}_p)$ of solutions $(x, y) \in \mathbb{F}_p^2$ together with a point at infinity $\mathcal{O}$ forms an abelian group with $\mathcal{O}$ as identity and $-(x, y) = (x, -y)$. For $P = (x_1, y_1)$, $Q = (x_2, y_2)$ with $P \neq \pm Q$,

$$P + Q = (x_3, y_3), \qquad \lambda = \frac{y_2 - y_1}{x_2 - x_1}, \quad x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda(x_1 - x_3) - y_1, \tag{2}$$

and for doubling ($y_1 \neq 0$) the same formulas apply with the tangent slope $\lambda = (3x_1^2 + a)/(2y_1)$ (Figure 2). That (2) returns a point of E is a polynomial identity modulo the two curve equations; we verified it symbolically (check S1 of `theory/elliptic_curves/symbolic_checks.py`, reduction of the numerator of $y_3^2 - x_3^3 - ax_3 - b$ by a Gröbner basis of the ideal generated by the curve equations gives zero). Associativity is the one group axiom that is not a one-line identity; the standard proofs go through the Riemann–Roch theorem or an explicit, lengthy polynomial verification (Washington, 2008, Ch. 2).

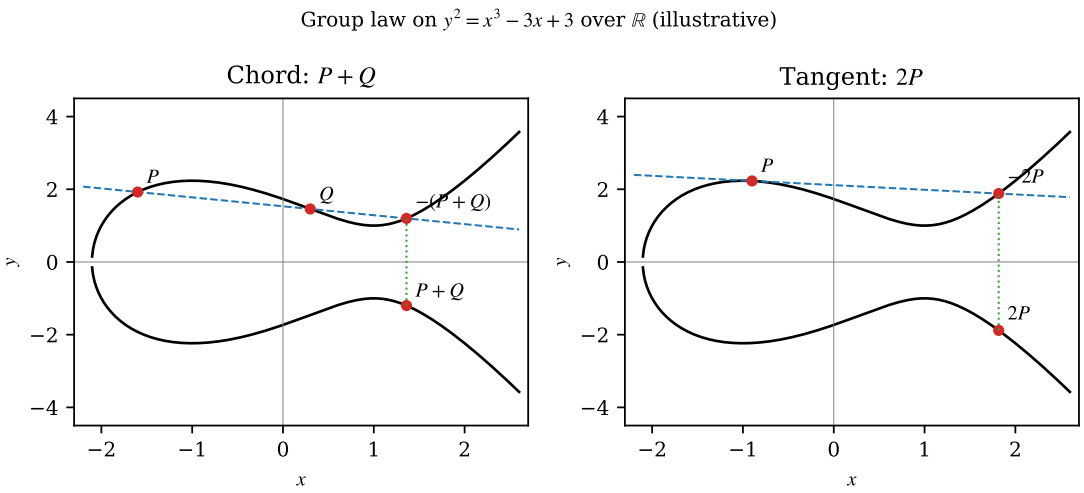


Figure 2. Geometry of the group law (illustrative, over $\mathbb{R}$): the chord through P and Q meets the curve in a third point whose reflection is $P + Q$; the tangent at P gives $2P$. Over $\mathbb{F}_p$ the same formulas (2) apply with no geometric picture. Generated by `scripts/fig_ec_geometry.py`.

Projective or Jacobian coordinates avoid the field inversion in (2) (Cohen et al., 1998); our reference code counts group operations in the affine model (one inversion per operation), while the Rust engine amortises inversions across parallel walks with Montgomery's simultaneous-inversion trick (Montgomery, 1987). Operation counts (additions, doublings, inversions, multiplications) are

reported next to wall-clock time throughout.

2.3. Scalar multiplication

For $k \in \mathbb{Z}$ and $P \in E(\mathbb{F}_p)$, $[k]P = P + \cdots + P$ (k terms). Left-to-right double-and-add costs ℓ doublings and on average $\ell/2$ additions for an ℓ-bit scalar; signed-window (wNAF) recoding with window w reduces the additions to about $\ell/(w+1)$ plus 2^{w-2} precomputed points (Hankerson et al., 2004). Double-scalar products $[u]P + [v]Q$, which dominate signature verification, are computed by Straus–Shamir interleaving (Straus, 1964) and multi-scalar products $\sum_i [a_i]P_i$ by Pippenger's bucket method (Pippenger, 1980); see Section 6.

2.4. Group order, Hasse's theorem and subgroup structure

Theorem 2.1 (Hasse). $|\#E(\mathbb{F}_p) - (p+1)| \le 2\sqrt{p}$.

The integer $t = p + 1 - \#E(\mathbb{F}_p)$ is the trace of the Frobenius endomorphism $\pi : (x,y) \mapsto (x^p, y^p)$, which satisfies $\pi^2 - t\pi + p = 0$ in $\mathrm{End}(E)$. Point counting is polynomial time by Schoof's algorithm and its Elkies–Atkin improvements (Schoof, 1985, 1995); we use PARI/GP's `ellcard` to generate prime-order synthetic curves of 20–56 bits and to re-verify $\#\mathsf{secp256k1}(\mathbb{F}_p)$. The group $E(\mathbb{F}_p)$ is either cyclic or a product of two cyclic groups; when $\#E(\mathbb{F}_p) = n$ is prime, as for secp256k1, every non-identity point generates the whole group and the cofactor is $h = 1$.

2.5. Isomorphisms, twists and the j-invariant

Two curves $y^2 = x^3 + ax + b$ and $y^2 = x^3 + a'x + b'$ are isomorphic over $\mathbb{F}_p$ iff $a' = u^4 a$, $b' = u^6 b$ for some $u \in \mathbb{F}_p^*$; the j-invariant $j = 1728 \cdot 4a^3/(4a^3 + 27b^2)$ classifies curves up to isomorphism over $\overline{\mathbb{F}}_p$. Curves with $a = 0$ have $j = 0$ and admit, when $p \equiv 1 \pmod 3$, six isomorphisms to themselves (automorphisms) rather than the generic two. The quadratic twist E' of E has $\#E'(\mathbb{F}_p) = p + 1 + t$; its order matters because an implementation that does not validate inputs may be driven onto the twist ("twist security").

2.6. Endomorphisms and complex multiplication

$\mathrm{End}(E)$, the ring of rational maps $E \to E$ preserving $\mathcal{O}$, always contains $\mathbb{Z}$ (scalar multiplications) and π. For an ordinary curve ($t \not\equiv 0 \pmod p$), $\mathrm{End}(E)$ is an order in the imaginary quadratic field $K = \mathbb{Q}(\sqrt{t^2 - 4p})$ containing $\mathbb{Z}[\pi]$. When $j = 0$ and $p \equiv 1 \pmod 3$ the map $\phi(x,y) = (\beta x, y)$ with $\beta^3 = 1$, $\beta \neq 1$, is an automorphism of order 3, $K = \mathbb{Q}(\sqrt{-3})$ and $\mathrm{End}(E) = \mathbb{Z}[\omega]$, $\omega = (-1 + \sqrt{-3})/2$, the Eisenstein integers—the maximal order of K, so no larger endomorphism ring is possible. In particular $4p = t^2 + 3f^2$ for an integer f (the conductor of $\mathbb{Z}[\pi]$ in $\mathbb{Z}[\omega]$),

and every endomorphism is $u+v\phi$ with $u,v\in\mathbb{Z}$, of degree u^2-uv+v^2. This observation is the starting point of Section 4: the only efficiently computable endomorphisms beyond scalars are the six automorphisms, and we show that the Gallant–Lambert–Vanstone constants of secp256k1 are literally the coordinates of $\bar{\pi}-1$ in $\mathbb{Z}[\omega]$.

2.7. Scalar decomposition

If ϕ acts on the cyclic group $\langle G\rangle$ of prime order n as $[\lambda]$ with $\lambda^2+\lambda+1\equiv 0 \pmod n$, any scalar can be written $k\equiv k_1+k_2\lambda \pmod n$ with $|k_1|,|k_2|=O(\sqrt{n})$ by reducing $(k,0)$ modulo the lattice $L=\{(u,v): u+v\lambda\equiv 0 \pmod n\}$ (Gallant et al., 2001). Then $[k]P=[k_1]P+[k_2]\phi(P)$ is a double-scalar product with half-length scalars, which halves the number of doublings. The bound $\max(|k_1|,|k_2|)<c\sqrt{n}$ for an explicit c is proved in Sica et al. (2003); we measure it exactly in Section 10.

2.8. The discrete-logarithm problem and generic-group complexity

Definition 2.2 (ECDLP)**.** Given G of prime order n and $Q\in\langle G\rangle$, find $d\in[0,n)$ with $Q=[d]G$.

In the generic-group model (Shoup, 1997; Nechaev, 1994) an algorithm may only apply the group operation to opaque encodings of elements; any such algorithm needs $\Omega(\sqrt{n})$ operations to solve the DLP in a group of prime order n with constant probability. The matching upper bounds are Shanks' baby-step giant-step (Shanks, 1971) ($\approx 2\sqrt{n}$ operations, $\sqrt{n}$ memory) and Pollard's rho (Pollard, 1978) ($\approx\sqrt{\pi n/2}$ expected operations under the random-mapping heuristic, constant memory), parallelised with linear speed-up by distinguished points (van Oorschot and Wiener, 1999). Section 5 reviews these methods and their refinements. The "square-root barrier" of our title is this generic bound: it can be beaten only by an algorithm that uses the *representation* of the group—an algebraic attack. Known algebraic attacks on elliptic curves (transfer to $\mathbb{F}_{p^k}^*$ when the embedding degree k is small, the anomalous-curve attack when $n=p$, index calculus via Weil descent or summation polynomials over extension fields) each require a structural property that secp256k1 verifiably lacks (Section 4). For prime fields with no small-degree subfield, no algorithm with complexity below $\sqrt{n}$ is known (Galbraith and Gaudry, 2016).

2.9. What "beyond the square-root barrier" can and cannot mean

Three kinds of improvement must be kept apart. (i) *Constant-factor* improvements reduce the constant in $c\sqrt{n}$: the negation map ($c\to c/\sqrt{2}$), the order-6 automorphism group of $j=0$ curves ($c\to c/\sqrt{6}$), better iteration functions, batched inversions. (ii) *Non-uniform* or preprocessing improvements trade a one-time computation of size T for a per-instance cost of $\sqrt{n/T}$, within the bound $ST^2=\tilde{\Omega}(n)$ (Corrigan-Gibbs and Kogan, 2018); the total work is never below $\sqrt{n}$.

(iii) *Asymptotic* improvements, which would break the generic bound, require a new algebraic idea and are the object of hypotheses A1–A5. The paper measures (i) and (ii) precisely on synthetic groups and gives negative results for the specific algebraic candidates of (iii) that we could formulate.

3. Literature Review and Research Gaps

Stage 1 of the pipeline produced a structured database of 82 items (21 theorems, 17 algorithms, 6 security assumptions, 10 complexity bounds, 10 experimental records, 8 open problems, 10 negative results; `literature/db.yaml`) and a bibliography of 129 entries each verified against Crossref, IACR ePrint, a DOI record or the official standard. We summarise what is settled, what is open, and which of the mandate's questions the literature already answers.

3.1. The elliptic-curve discrete logarithm

The square-root barrier is a theorem in the generic-group model (Shoup, 1997; Nechaev, 1994), matched by baby-step giant-step (Shanks, 1971) and Pollard's rho (Pollard, 1978) with the random-mapping constant $\sqrt{\pi n/2}$ (Harris, 1960; Flajolet and Odlyzko, 1990), attained in practice by r-adding walks with $r \geq 20$ (Teske, 1998, 2001) and parallelised with linear speed-up by distinguished points (van Oorschot and Wiener, 1999). With S bits of curve-specific precomputation the online cost obeys $ST^2 = \tilde{\Omega}(n)$ (Corrigan-Gibbs and Kogan, 2018), which the rho-based preprocessing of Bernstein and Lange (2013a) and Lee et al. (2011) matches. Automorphisms give constant factors: $\sqrt{2}$ for negation and $\sqrt{6}$ for the $j = 0$ family (Wiener and Zuccherato, 1999; Duursma et al., 1999; Gallant et al., 2000), provided fruitless cycles are handled (Bos et al., 2010; Bernstein et al., 2011); Bernstein and Lange (2013b) and Galbraith and Ruprai (2010) sharpen the BSGS and kangaroo constants, and Cheon et al. (2012) show (for $\mathbb{F}_p^*$) that the representation of group elements can make each rho step cheaper without changing the step count. Algebraic attacks need structure that `secp256k1` lacks: small embedding degree (Menezes et al., 1993; Frey and Rück, 1994), anomalous order (Semaev, 1998; Smart, 1999; Satoh and Araki, 1998), composite order (Pohlig and Hellman, 1978), or an extension field for summation-polynomial index calculus (Semaev, 2004; Gaudry, 2009; Diem, 2011; Petit and Quisquater, 2012); Galbraith and Gaudry (2016) survey the state of the art and record that no sub-exponential algorithm is known for prime fields, and Bernstein and Lange (2014) tabulate the exclusion criteria for standard curves.

Table 1 lists the completed ECDLP computations we could verify. The largest prime-field instance remains the 112-bit computation of Bos et al. (2012) (July 2009); the binary-field records are 113-bit curves (Wenger and Wolfger, 2014, 2016; Bernstein et al., 2016); the 131-bit Certicom challenges are open (Certicom Research, 2009a). The gap to `secp256k1` is a factor of about 2^{71} in

expected group operations.

Table 1. Completed elliptic-curve discrete-logarithm computations (verified records only).

Instance	Bits	Field	Hardware	Completed	Source
ECCp-79/-89/-97	79/89/97	prime	distributed PCs	1997–1998	Certicom Res
ECC2K-108	108	binary, Koblitz	distributed ($1.3 \cdot 10^6$ machine-days)	Apr. 2000	Certicom Res
ECCp-109	109	prime	distributed ($9.0 \cdot 10^6$ machine-days)	Oct. 2002	Certicom Res
ECC2-109	109	binary	distributed ($2.1 \cdot 10^7$ machine-days)	Apr. 2004	Certicom Res
secp112r1	112	prime	$\approx$ 215 PlayStation 3	Jul. 2009	Bos et al. (20
113-bit Koblitz	113	binary, Koblitz	18 Virtex-6 FPGAs	2014	Wenger and V
sect113r1	113	binary	10 Kintex-7 FPGAs	2015	Wenger and V
sect113r2	113	binary	Spartan-6 FPGAs	2016	Bernstein et a
ECCp-131, ECC2-131, ECC2K-130	130–131	prime/binary	—	open	Certicom Res

Figures as stated in the sources (`literature/records.md`). `secp256k1` has $n \approx 2^{256}$: rho with the negation map needs $\approx \sqrt{\pi n/4} \approx$ the 112-bit computation.

3.2. Signatures

Schnorr signatures are EUF-CMA under the discrete-logarithm assumption in the random-oracle model by the forking lemma (Pointcheval and Stern, 2000); the loss in the number of hash queries is unavoidable for algebraic reductions and the random oracle cannot be removed (Paillier and Vergnaud, 2005; Seurin, 2012). ECDSA requires an idealisation of the conversion function $R \mapsto x(R) \bmod n$ (Brown, 2005; Vaudenay, 2003; Fersch et al., 2016, 2017), extended to additive key derivation and presignatures by Groth and Shoup (2022). Nonce leakage is fatal (Howgrave-Graham and Smart, 2001; Breitner and Heninger, 2019; Aranha et al., 2020; Jančar et al., 2020), hence deterministic nonces (Pornin, 2013; Wuille et al., 2020b). Batch verification of Schnorr-type signatures is classical (Naccache et al., 1995; Bellare et al., 1998; Bernstein et al., 2012) and rests on multi-scalar multiplication (Straus, 1964; Pippenger, 1980; Möller, 2001); multi-signatures build on the same linearity (Bellare and Neven, 2006; Maxwell et al., 2019; Nick et al., 2021). The reference implementation is `libsecp256k1` (Wuille et al., 2024).

3.3. SHA-256

Beyond the Merkle–Damgård reduction (Merkle, 1990; Damgård, 1990) and the structural analysis of Gilbert and Handschuh (2004), knowledge about SHA-256 is knowledge about reduced step counts (Table 2). Collision attacks progressed from 21–24 steps in 2008 (Nikolić and Biryukov, 2008; Indesteege et al., 2009; Sanadhya and Sarkar, 2008) to 27–28 practical steps (Mendel et al., 2011, 2013), 31 practical steps (Li et al., 2024b) and a theoretical 37-step attack (Zhang et al., 2026); semi-free-start collisions reach 38–39 steps (Mendel et al., 2013; Li et al., 2024a); preimages reach 43 steps (Aoki et al., 2009) and 45 steps (52 for the compression function) at complexities within a

small factor of brute force (Khovratovich et al., 2012). SAT-based analysis started with Mironov and Zhang (2006); pure SAT search stalls around 28 steps while SAT combined with computer algebra reproduces the 38-step semi-free-start result (Nejati et al., 2017; Alamgir et al., 2024). The full SHA-1 collisions (Stevens et al., 2017; Leurent and Peyrin, 2020) are the benchmark for what a mature differential attack on a 80-step design costs (2^{63}).

Table 2. Reduced-step cryptanalysis of SHA-256 (64 steps): published records.

Type	Steps	Complexity	Practical?	Source
Collision	21	$\approx 2^{19}$	yes	Nikolić and Biryukov (2008)
Collision	23 / 24	2^{18} / $2^{28.5}$	yes	Indesteege et al. (2009)
Collision	23 / 24	$2^{11.5}$ / $2^{28.5}$	yes	Sanadhya and Sarkar (2008)
Collision	27	practical	yes	Mendel et al. (2011)
Collision	28 (31)	practical ($\leq 2^{65.5}$)	yes (no)	Mendel et al. (2013)
Collision	31	$\approx$ 1.2 h on 64 threads	yes	Li et al. (2024b)
Collision	37	theoretical	no	Zhang et al. (2026)
Semi-free-start collision	32 / 38 / 39	practical	yes	Mendel et al. (2011, 2013); Li et al. (2024a)
Semi-free-start (SAT+CAS)	38	solver reproduction	yes	Alamgir et al. (2024)
Collision (pure SAT)	≤ 28	—	yes	Alamgir et al. (2024)
Preimage	43	$2^{254.9}$	no	Aoki et al. (2009)
Preimage (hash / compression)	45 / 52	$\approx 2^{256}$ (bicliques)	no	Khovratovich et al. (2012)

Figures as stated in the abstracts (`literature/records.md`). No attack on the full 64-step function is known; the generic bounds 2^{128} (collision) and 2^{256} (preimage) stand.

3.4. Automated discovery

Program-search systems have produced verified mathematical constructions—matrix-multiplication tensors (Fawzi et al., 2022), combinatorial bounds (Romera-Paredes et al., 2024; Novikov et al., 2025), olympiad geometry proofs (Trinh et al., 2024), conjectures in knot theory and representation theory (Davies et al., 2021)—and end-to-end "AI scientist" pipelines have been proposed (Lu et al., 2024). None targets standardised cryptographic primitives, where the ground truth is dominated by impossibility results, so Axis D has no published baseline and defines its own (rediscovery rate, false-positive rate, and the quality of the negative results). On the formal side, Böving et al. (2025) make bit-vector facts about hash-function steps machine-checkable in Lean 4 (de Moura and Ullrich, 2021); we use this to separate checked identities from informal ones.

3.5. Research gaps and what the mandate already settles

Of the mandate's questions, the literature settles the following (`literature/research_gaps.md`): A1 (no improvement beyond constant factors from the endomorphism), A2 (orbits give exactly $\sqrt{|\mathrm{Aut}|}$), A5 (generic algorithms cannot beat $\sqrt{n}$, with or without preprocessing), B1 (the proof structures differ by the conversion-function idealisation), B2 (batch verification is known for Schnorr,

not for ECDSA), and C1/C2/C4 up to the recorded step counts. What is not settled, and what we test, is quantitative and methodological: the measured constants and their interaction with distinguished-point overheads and fruitless cycles at 20–48 bits (gap 2); the preprocessing constant on a fixed curve (gap 3); whether an automated search over iteration-function designs or over algebraic identities finds anything the theory does not predict (gaps 1, 8); how far generic SAT encodings get on step-reduced SHA-256 compared with the dedicated tools (gap 6); and whether the elementary identities that implementations rely on can be made machine-checked at negligible cost (gap 7). The closed form of the GLV constants (Proposition 4.2) is the one place where the symbolic search produced something we did not find written down, and we are careful to call it an exposition of a known structure rather than a discovery.

4. Mathematical Structure of secp256k1

4.1. Verified domain parameters

Table 3 lists the standardised parameters (Certicom Research, 2010) together with the derived quantities used in this paper. Every entry was computed twice with exact arithmetic—in our pure-Python library and in PARI/GP 2.19—and the two implementations agree on all fourteen named checks (`results/baselines/secp256k1_facts.json`, hypothesis H01).

4.2. Why the special form is interesting

Three features distinguish secp256k1 from a random prime-order curve: the special prime (fast reduction), $a = 0$ (cheaper doubling: $\lambda = 3x^2/2y$), and $j = 0$ with $p \equiv 1 \pmod 3$, which gives the curve complex multiplication by the Eisenstein integers and therefore an automorphism group of order 6 instead of 2. The first two are pure implementation conveniences. The third has a precise algebraic content that we now make explicit.

Proposition 4.1. *Let $E/\mathbb{F}_p$ be ordinary with $j(E) = 0$ and $p \equiv 1 \pmod 3$. Then $\mathrm{End}(E) = \mathbb{Z}[\omega]$, every endomorphism is $u + v\phi$ with $u, v \in \mathbb{Z}$ and has degree $u^2 - uv + v^2$, and $\mathrm{Aut}(E) = \{\pm 1, \pm\phi, \pm\phi^2\} \cong \mathbb{Z}/6$.*

Proof. ϕ is an automorphism of order 3 defined over $\mathbb{F}_p$, so $\mathbb{Z}[\phi] \cong \mathbb{Z}[\omega] \subseteq \mathrm{End}(E)$. Since E is ordinary, $\mathrm{End}(E)$ is an order in the imaginary quadratic field $\mathrm{End}(E) \otimes \mathbb{Q} \supseteq \mathbb{Q}(\omega) = \mathbb{Q}(\sqrt{-3})$, hence equal to $\mathbb{Q}(\sqrt{-3})$, whose unique maximal order is $\mathbb{Z}[\omega]$. Thus $\mathrm{End}(E) = \mathbb{Z}[\omega]$. The degree of an endomorphism is its norm, $\mathrm{N}(u + v\omega) = u^2 - uv + v^2$, and the units of $\mathbb{Z}[\omega]$ are the six sixth roots of unity (Silverman, 2009, Thm. III.10.1, Ex. III.10.?). □

Table 3. secp256k1 parameters and derived algebraic invariants (all verified exactly in Python and PARI/GP).

Quantity	Symbol	Value / verified property
Field prime	p	$2^{256}-2^{32}-977$; prime; $p \equiv 1 \pmod 3$, $p \equiv 3 \pmod 4$
Curve	E	$y^2 = x^3 + 7$; $a = 0$, $b = 7$; $\Delta \neq 0$; $j(E) = 0$
Group order	n	`FFFF...BAAEDCE6AF48A03BBFD25E8CD0364141`; prime; $\#E(\mathbb{F}_p) = n$ (SEA re-count)
Cofactor	h	1
Generator	G	x_G =`79BE667E...F81798`, y_G =`483ADA77...10D4B8`; $G \in E$; $[n]G = \mathcal{O}$
Trace of Frobenius	t	$p+1-n = 432420386565659656852420866390673177327$ ($\approx 2^{128.3}$, $t^2 \leq 4p$)
CM discriminant		$t^2 - 4p = -3f^2$, $f = 30314439467246543595250775667605759171$
Endomorphism	β	`7AE96A2B...719501EE`; $\beta^3 \equiv 1$, $\beta^2+\beta+1 \equiv 0 \pmod p$
Eigenvalue	λ	`5363AD4C...1B23BD72`; $\lambda^3 \equiv 1$, $\lambda^2+\lambda+1 \equiv 0 \pmod n$; $(\beta x_G, y_G) = [\lambda]G$
GLV basis	(a_1, b_1)	a_1 =`3086D221A7D46BCDE86C90E49284EB15`, $b_1 = -$`E4437ED6010E88286F547FA90ABFE4C3`
	(a_2, b_2)	a_2 =`114CA50F7A8E2F3F657C1108D9D44CFD8`, $b_2 = a_1$; $a_i + b_i\lambda \equiv 0 \pmod n$; $\det = n$
Anomalous?		no ($n \neq p$)
Embedding degree	k	$\mathrm{ord}_n(p) > 10^6$ (PARI: $\mathrm{ord}_n(p)$ is a 253-bit integer); MOV/Frey–Rück inapplicable
Twist order	n'	$2p+2-n = 3^2 \cdot 13^2 \cdot 3319 \cdot 22639 \cdot q_{220}$, q_{220} a 220-bit prime
2-torsion / $x = 0$		x^3+7 has no root in $\mathbb{F}_p$ (no point of order 2); 7 is a non-residue (no point with $x = 0$)
$n-1$, $p-1$		$n-1 = 2^6 \cdot 3 \cdot 149 \cdot 631 \cdot (\ldots)$; $p-1 = 2 \cdot 3 \cdot 7 \cdot 13441 \cdot (\ldots)$ (large cofactors)

Source: `experiments/baselines/secp256k1_facts.py`. Hexadecimal values abbreviated; full values in the JSON result and in `src/wp34/secp256k1.py`.

The Frobenius π lies in $\mathbb{Z}[\omega]$ with $\mathrm{N}(\pi) = p$ and $\mathrm{Tr}(\pi) = t$, so $\pi = (t \pm f\sqrt{-3})/2$ and $4p = t^2 + 3f^2$—the identity the automated search of Section 8 found in the form "$t^2 + 3b_1^2 = 4p$". For secp256k1 we verified $t^2 - 4p = -3f^2$ exactly (Table 3). Proposition 4.1 answers research question A1 at the structural level: there is no efficiently computable endomorphism beyond $\mathbb{Z}[\phi]$, because there is no endomorphism beyond $\mathbb{Z}[\phi]$ at all. The smallest non-unit endomorphism, $1 - \phi$ of degree 3, is computable with one addition ($P - \phi(P)$) but is simply the sum of two things we already have.

4.3. The GLV lattice is the ideal $(\bar{\pi} - 1)$

Proposition 4.2. *Let $L = \{(k_1, k_2) \in \mathbb{Z}^2 : k_1 + k_2\lambda \equiv 0 \pmod{n}\}$ be the GLV lattice. Under the identification $(k_1, k_2) \leftrightarrow k_1 + k_2\omega$, L is the principal ideal $(\bar{\pi} - 1)\mathbb{Z}[\omega]$, where $\bar{\pi} = (t - f\sqrt{-3})/2$ is the Frobenius conjugate for which $\omega \mapsto \lambda$ sends $\bar{\pi} - 1$ to 0. Consequently*

$$(a_1, b_1) = \left(\tfrac{t-f}{2} - 1,\ -f\right), \qquad (a_2, b_2) = (a_1 - b_1,\ a_1)\ (= \omega \cdot (a_1 + b_1\omega) \text{ up to sign}), \qquad a_1^2 - a_1 b_1 + b_1^2 = n, \tag{3}$$

and these are exactly the constants hard-coded in `libsecp256k1`.

Proof. The map $\rho : \mathbb{Z}[\omega] \to \mathbb{Z}/n\mathbb{Z}$, $\omega \mapsto \lambda$, is a ring homomorphism because $\lambda^2 + \lambda + 1 \equiv 0$. Its kernel is an ideal of index n, hence of norm n; $\mathbb{Z}[\omega]$ is a principal ideal domain, so the kernel is (α) with $\mathrm{N}(\alpha) = n$. Now $n = \#E(\mathbb{F}_p) = \mathrm{N}(\pi - 1) = \mathrm{N}(\bar{\pi} - 1)$ and π acts on $E(\mathbb{F}_p)$ as the identity, so $\pi - 1$ kills every point and one of $\pi - 1$, $\bar{\pi} - 1$ lies in $\ker\rho$ (which one depends on the labelling of ω versus λ); call it $\bar{\pi} - 1$. Since $(\bar{\pi} - 1)$ and $\ker\rho$ are ideals of the same norm with one containing the other, they are equal. Writing $\sqrt{-3} = 2\omega + 1$ gives $\bar{\pi} - 1 = \frac{t-f}{2} - 1 - f\omega$, i.e. (a_1, b_1) as stated; multiplying by the unit ω gives the second basis vector; and $\mathrm{N}(\bar{\pi} - 1) = n$ is (3). The identification with the `libsecp256k1` constants is an exact integer check (`identity_search.json`, field `closed_form_matches_libsecp256k1`). □

We are not aware of a place where the `libsecp256k1` constants are written in this closed form (the library's comments derive them by the extended Euclidean algorithm, as in Gallant et al. (2001)); the ideal-theoretic description of the GLV lattice for CM curves is classical (Sica et al., 2003; Galbraith et al., 2011). We therefore classify Proposition 4.2 as a known result independently reproduced, with an explicit form that may be of expository use. It immediately yields the sharp bound $\max(|k_1|, |k_2|) \le |a_1| + |b_1|$ for Babai rounding; empirically both sub-scalars have at most 128 bits (hypothesis H03).

4.4. Orbits, equivalence classes and the limits of the automorphism trick

The group $\mathrm{Aut}(E)$ acts on $E(\mathbb{F}_p)\setminus\{\mathcal{O}\}$ freely (no non-identity automorphism fixes a point of prime order $n>3$ other than $\mathcal{O}$), so the orbits $\{\pm P, \pm\phi P, \pm\phi^2 P\}$ all have size 6 and the orbit space has $(n-1)/6$ elements. Pollard's rho can be run on orbits because a canonical representative (for instance the point with minimal x among $\{x, \beta x, \beta^2 x\}$ and then minimal y) can be computed with two field multiplications, and the scalar bookkeeping is adjusted by powers of λ and signs (Wiener and Zuccherato, 1999; Duursma et al., 1999; Gallant et al., 2000). The random-mapping heuristic then predicts $\sqrt{\pi n/12}$ iterations instead of $\sqrt{\pi n/2}$: a factor $\sqrt{6}\approx 2.45$. Two costs must be paid: canonicalisation, and *fruitless cycles*—short cycles of the iteration that contain no useful collision, whose frequency is $\Theta(1/r)$ for an r-adding walk on classes and which must be detected and escaped (Bos et al., 2010; Bernstein et al., 2011). Section 10 measures the achieved factor and the cycle overhead (hypothesis H04).

Could the orbit space do more than save a constant? Hypothesis H05 formalises the natural conjecture that $E(\mathbb{F}_p)/\mathrm{Aut}(E)$ carries a group structure compatible with the projection, which would reduce the DLP to a group of order $\approx n/6$ (and, iterating, suggest further reductions). It is false.

Proposition 4.3. *There is no group structure on the orbit set $E(\mathbb{F}_p)/\mathrm{Aut}(E)$ for which the projection is a homomorphism.*

Proof. If the projection were a homomorphism, its kernel would be the orbit of $\mathcal{O}$, which is $\{\mathcal{O}\}$, so the projection would be injective—contradicting the fact that the other orbits have six elements. Equivalently, $P\sim\phi P$ and $Q\sim Q$ would force $P+Q\sim\phi P+Q$; on synthetic 16-, 20- and 24-bit $j=0$ curves this fails for every one of 1500 random pairs (`results/theory/orbit_quotient_counterexample.jso`

□

The same argument applies to any finite group of automorphisms fixing the identity: equivalence classes under automorphisms are compatible with *iterations* (which is all rho needs) but never with the group law. This is the precise sense in which the structure of secp256k1 yields a constant and nothing more (research questions A2, A5).

4.5. Algebraic attacks: an exact exclusion table

The reductions that beat the generic bound each need a verifiable property. The MOV/Frey–Rück transfer to $\mathbb{F}_{p^k}^*$ needs a small embedding degree $k=\mathrm{ord}_n(p)$ (Menezes et al., 1993; Frey and Rück, 1994); for secp256k1, k is a 253-bit integer. The anomalous-curve attack needs $n=p$ (Semaev, 1998; Smart, 1999; Satoh and Araki, 1998); here $n\neq p$. Attacks through invalid points on the twist

need a smooth twist order; the twist cofactor is a 220-bit prime (Bernstein and Lange, 2014). Weil-descent and summation-polynomial index calculus need an extension field or a subfield structure (Semaev, 2004; Gaudry, 2009; Diem, 2011; Petit and Quisquater, 2012); $\mathbb{F}_p$ with p prime has none. Hypothesis H06 quantifies the last point experimentally: with a factor base $F = \{P : x(P) < \sqrt{p}\}$ a random point decomposes as a sum of two factor-base points with probability $1 - e^{-|F|^2/2n} \approx 0.4$ (measured 0.31–0.44 on 16–28-bit curves, in agreement with the Poisson prediction), but finding the decomposition without an oracle costs $|F|$ operations per trial, and even with a free oracle the $|F|$ relations cost $\geq 2.4\sqrt{n}$ and the linear algebra $|F|^2 \approx n$. Semaev's S_3 polynomial, which we re-derived and verified symbolically (check S4), has degree 2 in each variable, so the $m = 2$ decomposition is a square-root problem in disguise; for $m \geq 3$ the decomposition step itself becomes the bottleneck over prime fields. No algorithm with complexity below $\sqrt{n}$ is known for this setting (Galbraith and Gaudry, 2016), and our experiments give no hint of one.

5. Generic and Algebraic Discrete-Logarithm Algorithms

This section states the algorithms we implemented (twice: a transparent Python reference and a Rust engine) with their exact cost models; Section 10 compares the models with measurements. Throughout, G has prime order n and $Q = [d]G$.

5.1. Baby-step giant-step

Shanks' method (Shanks, 1971) stores m baby steps $\{[j]G : 0 \leq j < m\}$ in a hash table and walks giant steps $Q - [im]G$ until a match $Q - [im]G = [j]G$ gives $d = im + j$. With $m = \lceil\sqrt{n}\rceil$ the cost is at most $m + \lceil n/m\rceil \approx 2\sqrt{n}$ group operations and $\sqrt{n}$ table entries; with memory m the time is $m + n/m$, a deterministic time–memory trade-off. Two refinements matter: interleaving baby and giant steps halves the expected (not worst-case) cost for uniformly random d, and using the negation map on the baby-step table saves a further $\sqrt{2}$ (Bernstein and Lange, 2013b). We benchmark the plain variant as the deterministic baseline.

5.2. Pollard's rho and r-adding walks

Pollard's rho (Pollard, 1978) iterates a pseudo-random map f on $\langle G\rangle$ whose values are tracked as $a_iG + b_iQ$; a collision $X_i = X_j$ with $b_i \neq b_j$ gives $d = (a_i - a_j)/(b_j - b_i) \bmod n$. If f behaved as a uniformly random mapping, the expected number of iterations before the first repetition would be $\sqrt{\pi n/2} \approx 1.2533\sqrt{n}$ (Flajolet and Odlyzko, 1990). Pollard's original three-partition map is measurably worse than random; Teske's r-adding walk $f(X) = X + M_{h(X)}$ with $r \approx 20$ precomputed multipliers $M_j = u_jG + v_jQ$ is within a few percent of the random-mapping constant (Teske, 1998,

2001), and mixed walks that include occasional doublings close the remaining gap. Cycle detection by Floyd's tortoise-and-hare costs three evaluations per step; Brent's method (Brent, 1980) costs about one. Our Python reference implements Floyd; the Rust engine implements Floyd, Brent and the distinguished-point variant below.

5.3. Distinguished points and parallel collision search

van Oorschot and Wiener (1999) replace cycle detection by storing only *distinguished points* (those whose x-coordinate has θ^{-1} specific bits equal to zero) in a shared table; t processors each run independent walks from random starts $a_0G + b_0Q$, report distinguished points, and a collision between two walks with different (a,b) solves the instance. The expected total work is $\sqrt{\pi n/2} + t/\theta$ group operations (the second term is the extra distance walked after the collision point), memory is about $\theta\sqrt{\pi n/2}$ entries, and the speed-up is linear in t when $t/\theta \ll \sqrt{n}$. This is the algorithm behind every ECDLP record (Bos et al., 2012; Bernstein et al., 2016; Bailey et al., 2009).

5.4. Equivalence classes: negation map and automorphisms

Replacing points by classes $\{\pm X\}$ halves the effective group size and divides the expected iterations by $\sqrt{2}$; for the $j = 0$ family the order-6 classes divide them by $\sqrt{6}$. The price is fruitless cycles: on classes, the walk $X \mapsto X + M_j$ has probability about $1/(2r)$ of returning to X after two steps (when the canonical representative flips sign), creating 2-cycles that must be detected by a one-step look-ahead and escaped (Bos et al., 2010; Bernstein et al., 2011); longer fruitless cycles $(4, 6, \dots)$ occur with probability $O(r^{-k/2})$ and are handled by periodic checks and a doubling escape. Our engine implements the look-ahead and a bounded-length check and reports the number of cycles encountered.

5.5. Pollard's kangaroo for intervals

When d is known to lie in an interval of width w, Pollard's kangaroo (lambda) method (Pollard, 1978, 2000) runs a tame kangaroo from a known point and a wild one from Q, both jumping by powers of two of mean $\approx \sqrt{w}/2$; the wild kangaroo is caught at a trap in expected $\approx 2\sqrt{w}$ jumps. The van Oorschot–Wiener herd version is parallel and uses distinguished points; Galbraith and Ruprai (2010) improve the constant to $\approx 1.71\sqrt{w}$ with the negation map. We use the kangaroo as a second, structurally different baseline (and as the reason interval restrictions on nonces or keys are dangerous).

5.6. Pohlig–Hellman

If $\mathrm{ord}(G) = \prod q_i^{e_i}$ is smooth, the DLP reduces to the prime-order subgroups and costs $\sum_i e_i(\sqrt{q_i} + \log n)$ (Pohlig and Hellman, 1978); this is why n must be prime (or have a large prime factor) and why the cofactor 1 of secp256k1 matters. Our reference implements it on small composite-order curves for completeness.

5.7. Preprocessing (non-uniform) algorithms

Bernstein and Lange (2013a) observe that a one-time table of T distinguished points, each ending a walk of length W from a known multiple of G, lets every later instance be solved in about $n/(TW)$ online operations; choosing $W = \sqrt{n/T}$ gives online cost $\sqrt{n/T}$ after $\sqrt{nT}$ preprocessing. Corrigan-Gibbs and Kogan (2018) prove that any generic algorithm with S bits of advice and T online queries satisfies $ST^2 = \tilde{\Omega}(n)$, so this trade-off is optimal up to logarithmic factors. Hypothesis H12 measures the constant.

5.8. Lower bounds and what they rule out

Shoup's theorem (Shoup, 1997) is unconditional within the generic-group model: an algorithm that only uses the group operation (and equality tests) on $\langle G \rangle$ of prime order n succeeds with probability $O(m^2/n)$ after m operations. Every algorithm above is generic, so none can have complexity $o(\sqrt{n})$; constant-factor gains from equivalence classes are the only room left (classes of size c give at most $\sqrt{c}$, and $c \le 6$ for an elliptic curve over $\mathbb{F}_p$ with $p > 3$ since $|\mathrm{Aut}(E)| \le 6$). The automated search of hypothesis H09 lives entirely in this model and therefore cannot find a shortcut; what it can find is a better constant, and we report that it does not.

5.9. Cost model and empirical complexity

For each run we record the number of group additions and doublings, the number of field inversions (one per affine operation in the reference; amortised in the engine), wall-clock time, CPU time summed over threads, table size and peak resident memory. The empirical model

$$T(n) = a\, n^{\alpha} (\log_2 n)^{\beta} \tag{4}$$

is fitted by least squares on $\log T$ with β either free or fixed to 0, and the speed-up of a candidate over a baseline at size n is $S(n) = T_{\mathrm{baseline}}(n)/T_{\mathrm{candidate}}(n)$ with a bootstrap confidence interval over instances. For rho-type algorithms the number of iterations until collision has a Rayleigh-like distribution with coefficient of variation $\sqrt{(4-\pi)/\pi} \approx 0.52$, so at least 20 instances per size are needed for a $\pm 23\%$ standard error on the mean; we use 20–40.

6. ECDSA and Schnorr: Security and Computational Structure

6.1. ECDSA

Let H be SHA-256 and z the integer obtained from the leftmost $\lceil \log_2 n \rceil$ bits of $H(m)$ (for secp256k1 all 256 bits; z may exceed n and is reduced). With private key $d \in [1, n)$ and public key $P = [d]G$, signing picks a nonce $k \in [1, n)$, computes

$$R = [k]G, \qquad r = x(R) \bmod n, \qquad s = k^{-1}(z + rd) \bmod n, \tag{5}$$

and outputs (r, s) with $r, s \neq 0$. Verification computes $w = s^{-1}$, $u_1 = zw$, $u_2 = rw$, $X = [u_1]G + [u_2]P$ and accepts iff $x(X) \bmod n = r$ (National Institute of Standards and Technology, 2023; Certicom Research, 2009b). Correctness: $X = [s^{-1}(z + rd)]G = [k]G = R$. Bitcoin additionally requires *low-s* ($s \leq n/2$, BIP 62/146) because $(r, n - s)$ is also valid: the verification point becomes $-X$, whose x-coordinate is unchanged (check S5; formalised in Section 11).

Where the security lives. The nonce k must be uniform and secret: k known gives $d = (sk - z)r^{-1}$; k reused across two messages gives $k = (z_1 - z_2)/(s_1 - s_2)$ and then d (our reference implements this textbook recovery on synthetic keys, `recover_private_key_from_nonce_reuse`); and a few bits of bias in many nonces suffice for lattice recovery (Howgrave-Graham and Smart, 2001; Breitner and Heninger, 2019). RFC 6979 (Pornin, 2013) derives k deterministically by HMAC-DRBG from $(d, H(m))$, which removes the randomness requirement at signing time (the nonce is still secret because d is). Timing and cache side channels in the scalar multiplication by k are the practical threat (Kocher, 1996; Brumley and Tuveri, 2011; Aranha et al., 2020; Jančar et al., 2020); constant-time scalar multiplication, as in `libsecp256k1` and the `k256` crate, is the remedy and costs a measurable constant (Table 14).

Proof structure. ECDSA has no reduction to the DLP in the random-oracle model alone: the "conversion function" $R \mapsto x(R) \bmod n$ is not a random oracle and its algebraic structure (it is even, $x(R) = x(-R)$, which is the source of malleability) obstructs the forking-lemma argument. Known results prove security in the generic group model (Brown, 2005), under the assumption that the conversion function behaves as a bijective random oracle (Fersch et al., 2016), or for weakened unforgeability notions (Fersch et al., 2017); Groth and Shoup (2022) give a modern generic-group treatment that covers key derivation and presignatures. Vaudenay (2003) surveys the early history. The contrast with Schnorr is structural, not quantitative (research question B1).

6.2. BIP340 Schnorr

BIP340 (Wuille et al., 2020b) fixes the Schnorr scheme (Schnorr, 1991) with three design decisions. *X-only public keys*: P is encoded by $x(P)$ alone and implicitly has even y (the signer negates d if needed), saving a byte and making keys compatible with Taproot tweaking (Wuille et al., 2020a). *Tagged hashes*: $H_{\text{tag}}(m) = \text{SHA256}(\text{SHA256}(\text{tag}) \,\|\, \text{SHA256}(\text{tag}) \,\|\, m)$ for the tags `BIP0340/aux`, `BIP0340/nonce`, `BIP0340/challenge`, which separates domains at no cost since the 64-byte prefix fills exactly one compression block that can be precomputed. *Key-prefixed challenge*: $e = H_{\text{challenge}}(x(R) \,\|\, x(P) \,\|\, m) \bmod n$, which prevents related-key attacks and makes the scheme non-malleable. Signing: $k_0 = H_{\text{nonce}}(t \,\|\, x(P) \,\|\, m) \bmod n$ with $t = d \oplus H_{\text{aux}}(\text{aux})$, $R = [k_0]G$, $k = k_0$ or $n - k_0$ so that $y(R)$ is even, $s = k + ed \bmod n$, signature $(x(R), s)$ in 64 bytes. Verification: lift $x(P)$ and check

$$[s]G = R + [e]P \quad \text{with } y(R) \text{ even and } x(R) \text{ as signed.} \tag{6}$$

Correctness is the one-line identity $s = k + ed \Rightarrow [s]G = [k]G + [e][d]G$ (check S6, formalised in Section 11). Our pure-Python implementation passes all 19 official test vectors, including the negative vectors for $x \geq p$, $s \geq n$, odd $y(R)$ and wrong R; the Rust benchmark harness checks the same vectors with both `libsecp256k1` and `k256`, and 1000 signatures produced by each library are byte-identical and verify under the other and under the Python reference (hypothesis H23).

Proof structure. Schnorr signatures are EUF-CMA secure under the discrete-logarithm assumption in the random-oracle model via the forking lemma (Pointcheval and Stern, 2000); the reduction loses a factor of the number of hash queries, and Seurin (2012) shows this loss is essentially unavoidable for forking-based proofs and that the random oracle cannot be dispensed with for a natural class of reductions. The linear structure $s = k + ed$ is also what makes batch verification and multi-signatures (Bellare and Neven, 2006; Maxwell et al., 2019; Nick et al., 2021) possible; the same linearity is what ECDSA's k^{-1} destroys.

6.3. Batch verification and multi-scalar multiplication

Given b signatures (R_i, s_i) on messages m_i under keys P_i, choose random weights a_i ($a_1 = 1$, 128 bits otherwise) and check

$$\Big[\sum_i a_i s_i\Big] G = \sum_i [a_i]R_i + \sum_i [a_i e_i]P_i. \tag{7}$$

If any signature is invalid the check fails except with probability 2^{-128} (Naccache et al., 1995; Bellare et al., 1998; Bernstein et al., 2012). The right-hand side is a multi-scalar multiplication with $2b$ terms, computed by Straus interleaving (cost ≈ 256 doublings plus $2b \cdot 256/(w+1)$ additions with wNAF windows) or Pippenger's bucket method (cost $\approx 2b \cdot \lceil 256/c \rceil$ additions plus 2^c buckets

per window, asymptotically $O(b \cdot 256/\log b)$) (Straus, 1964; Pippenger, 1980; Möller, 2001). Batch verification does not change consensus: it accepts exactly the sets of signatures that individual verification accepts (with the stated negligible error). Hypothesis H13 measures the gain with our own instrumented Straus and Pippenger implementations on `k256` arithmetic (Section 10); the answer to B2 is a saturating constant-factor speed-up of $3.1\times$ over single verification in the same arithmetic, bounded by the non-MSM work (square root for R_i, tagged hash).

6.4. What can be simplified and what cannot (B3, B4)

The algebraic identities behind the implementations—$x(-P) = x(P)$, $(\beta x, y) = [\lambda](x, y)$, Shamir's trick, batch verification by random linear combination—are all consequences of the $\mathbb{Z}$-module structure of $\langle G \rangle$ and of the shape of the curve equation; the symbolic checks of Section 8 and the Lean statements of Section 11 confirm them but find nothing beyond them. We looked specifically for a verification shortcut that avoids one of the two scalar multiplications in (6): none exists without changing the predicate, because any procedure that accepts exactly the solutions of $[s]G - [e]P = R$ must evaluate a group element that depends on both s and e, and in the generic-group model this needs $\Omega(\log n)$ group operations for each. The practical levers are the ones in Table 14: fixed-base tables for G, GLV for P, Jacobian coordinates with a single final inversion, and batching.

7. SHA-256: Mathematical Structure and Reduced-Round Analysis

7.1. Construction

SHA-256 (National Institute of Standards and Technology, 2015) is a Merkle–Damgård iteration (Merkle, 1990; Damgård, 1990) of a compression function $f : \{0,1\}^{256} \times \{0,1\}^{512} \to \{0,1\}^{256}$. The message is padded with a 1-bit, zeros and the 64-bit length to a multiple of 512 bits; the chaining value starts at the fixed IV H_0 (fractional parts of $\sqrt{2}, \ldots, \sqrt{19}$); and $H_i = f(H_{i-1}, M_i)$ with $f(H, M) = H \boxplus \mathrm{Steps}_{64}(H, M)$, the "feed-forward" Davies–Meyer construction that makes f non-invertible. Inside, 32-bit words are combined with modular addition $\boxplus$, XOR, AND, rotations R and shifts S through

$$\begin{aligned} \Sigma_0(x) &= \mathrm{R}^2(x) \oplus \mathrm{R}^{13}(x) \oplus \mathrm{R}^{22}(x), & \Sigma_1(x) &= \mathrm{R}^6(x) \oplus \mathrm{R}^{11}(x) \oplus \mathrm{R}^{25}(x), \\ \sigma_0(x) &= \mathrm{R}^7(x) \oplus \mathrm{R}^{18}(x) \oplus \mathrm{S}^3(x), & \sigma_1(x) &= \mathrm{R}^{17}(x) \oplus \mathrm{R}^{19}(x) \oplus \mathrm{S}^{10}(x), \\ \mathrm{Ch}(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z), & \mathrm{Maj}(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z). \end{aligned} \tag{8}$$

The message schedule expands $M = W_0 \| \cdots \| W_{15}$ by $W_t = \sigma_1(W_{t-2}) \boxplus W_{t-7} \boxplus \sigma_0(W_{t-15}) \boxplus W_{t-16}$ for $16 \le t < 64$, and step t updates the state (a, b, c, d, e, f, g, h) as in Figure 3: $T_1 = h \boxplus \Sigma_1(e) \boxplus$

$\mathrm{Ch}(e,f,g) \boxplus K_t \boxplus W_t$, $T_2 = \Sigma_0(a) \boxplus \mathrm{Maj}(a,b,c)$, then $(a,b,c,d,e,f,g,h) \leftarrow (T_1 \boxplus T_2, a, b, c, d \boxplus T_1, e, f, g)$.

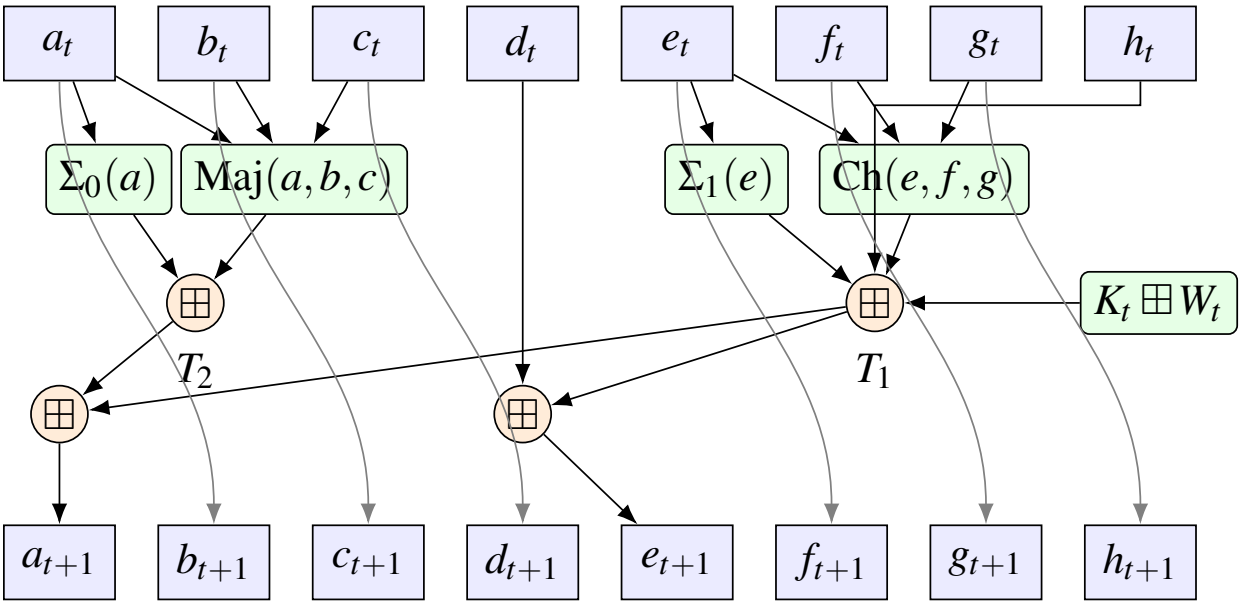


Figure 3. One step of the SHA-256 compression function (exact data flow of FIPS 180-4; illustrative layout). Rotations and shifts are free in hardware and in CNF (they are variable renamings); the nonlinearity comes from Ch, Maj and the carries of the modular additions.

7.2. Elementary structure

Three facts, all machine-checked in Section 11, organise the analysis. (i) $\Sigma_0, \Sigma_1, \sigma_0, \sigma_1$ are $\mathbb{F}_2$-linear bijections of $\{0,1\}^{32}$—including σ_0 and σ_1, whose logical shift makes each of their two summands singular but not their sum (we had conjectured the opposite; the machine-checked refutation is hypothesis H22). (ii) For fixed (K_t, W_t) the step function is a bijection of the 256-bit state, with explicit inverse $h = T_1 \ominus \Sigma_1(e) \ominus \mathrm{Ch}(e,f,g) \ominus K_t \ominus W_t$ where $T_1 = a_{t+1} \ominus T_2$; and the schedule recurrence can be solved for W_{t-16}. Hence, without the feed-forward, the 64-step map is invertible for any chosen message: pseudo-preimages of the compression function (choose the IV) are free, and "free-start" collisions are a different, much easier problem than collisions of the hash (hypothesis H20). (iii) The step-t nonlinearity is weak: Ch and Maj are bitwise (no inter-bit mixing), so all diffusion across bit positions comes from the Σ functions and from carries.

7.3. Generic bounds and the meaning of reduced-step results

For an ideal 256-bit hash a preimage costs $\approx 2^{256}$ evaluations, a collision $\approx 2^{128}$ by the birthday paradox, and these are the targets any attack must beat. Reduced-step analysis studies Steps_s for $s < 64$ to quantify the security margin; a result on s steps says nothing about 64 steps unless it

extends. The published state of the art (Table 2, Section 3) is: practical collisions for 28 steps (Dobraunig et al., 2015) and 31 steps (Li et al., 2024b), with a theoretical 37-step collision announced in 2026 (Zhang et al., 2026); semi-free-start collisions for 38–39 steps (Mendel et al., 2013; Li et al., 2024a); preimages for 43 steps (Aoki et al., 2009) and 45 steps of the hash (52 of the compression function) with bicliques (Khovratovich et al., 2012), all far above 2^{128} in cost. Differential attacks use *local collisions*: the 9-step pattern of Chabaud and Joux (1998) adapted to SHA-256 by Gilbert and Handschuh (2004) injects a difference δ into W_t and cancels its propagation with corrections in $W_{t+1}, \ldots, W_{t+8}$; in the XOR-linearised model the cancellation is exact, and the real step function follows the pattern with a probability that depends on the carries, which dedicated tools raise by imposing conditions on message and state bits (Nikolić and Biryukov, 2008; Mendel et al., 2011).

7.4. What we measure

Axis C runs four families of experiments on the reference implementation (verified against `hashlib` for all message lengths): diffusion of single-bit differences through the steps (H15); statistical detectability of input–output correlations as a function of the step count (H16); SAT (CaDiCaL, Kissat) and SMT (Z3 bit-vector) preimage and collision searches on step-reduced compression functions with Tseitin encodings (H17, H18) (Mironov and Zhang, 2006; Nejati et al., 2017; Alamgir et al., 2024); and the 9-step local collision in the linearised and the real model (H19). The questions C1–C5 are answered in Section 10; we state in advance that nothing in these experiments bears on the security of the full 64-step hash, and that our generic SAT encodings are expected to reach far fewer steps than the dedicated characteristic-search tools of the literature.

8. Autonomous Mathematical Discovery Framework

The methodological object of this paper is a six-stage pipeline (Figure 4) run by an autonomous agent (Claude Code with the Claude Fable 5.1 model) under the written mandate reproduced in `docs/MANDATE.md`. We describe the stages as implemented, the artefacts each produces, and the rules that keep the pipeline honest. The pipeline is deliberately simple; its value is measured in Section 10 by what it rediscovered, what it rejected, and what it did not claim.

Stage 1: literature extraction. A structured database (`literature/db.yaml`) records theorems, algorithms, assumptions, complexity bounds, experimental records, open problems and negative results, each with a source key that must exist in the verified BibTeX file. Verification means that the entry was confirmed against Crossref, IACR ePrint, a DOI, or the official standard; unconfirmed items are listed separately and never cited.

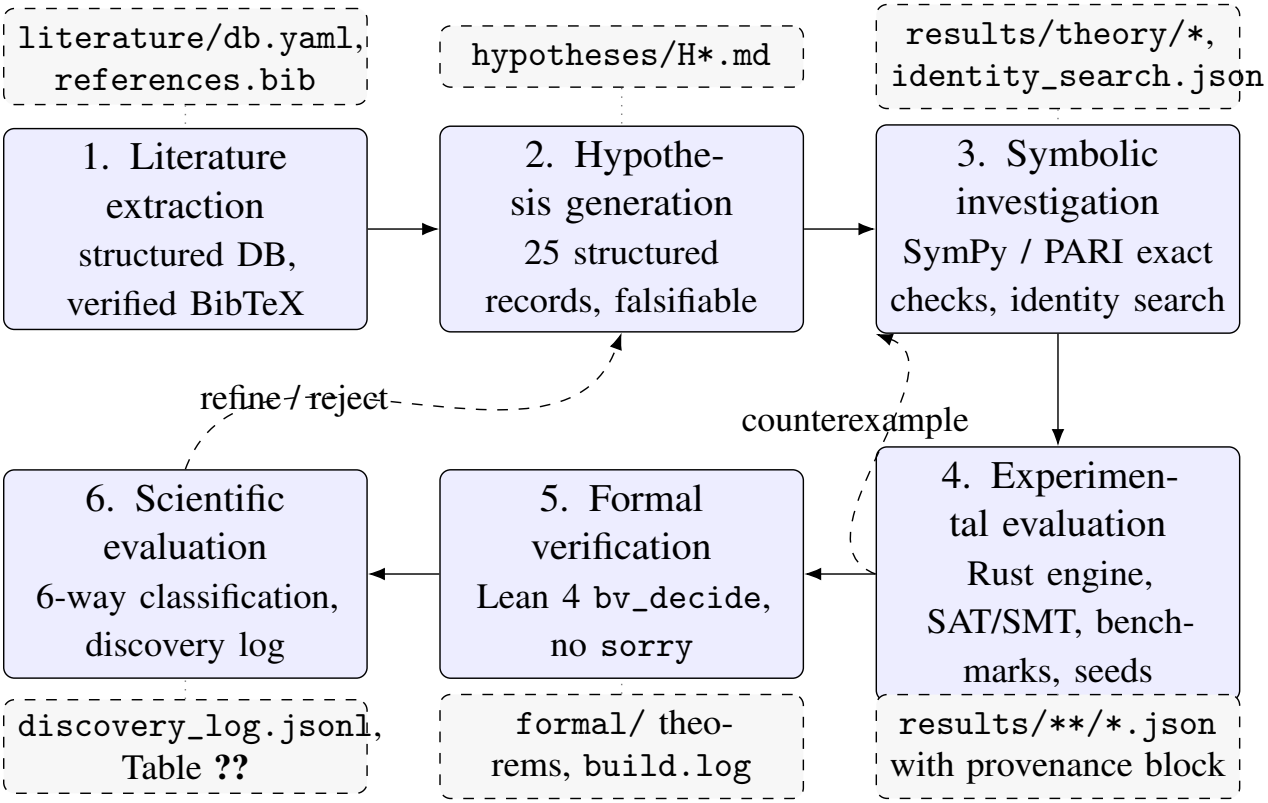


Figure 4. The six-stage discovery pipeline (illustrative diagram of the implemented workflow; file names refer to the repository). Dashed arrows are the two feedback loops: counterexamples send a hypothesis back to symbolic analysis, and evaluation sends it back to generation for refinement or rejection.

Stage 2: hypothesis generation. Each hypothesis is a file with sixteen mandatory fields (statement, motivation, assumptions, expected contribution, related work, possible counterexamples, verification strategy, computational budget, status, classification, evidence, verdict, ...), validated by `registry.py`, which also emits the discovery log and Table **??**. Hypotheses were generated from three sources: the mandate's research questions, gaps noted in Stage 1, and the outputs of Stage 3's identity search.

Stage 3: symbolic investigation. Two tools. SymPy performs exact polynomial reductions modulo the curve equations (the six checks S1–S6 of `symbolic_checks.py`: group-law closure, ϕ is an endomorphism, the x-only differential addition formula, Semaev's S_3, ECDSA malleability, Schnorr correctness). The *identity search* enumerates 58 064 candidate relations among twelve constants of secp256k1—binary quadratic forms $u^2+cuv+dv^2$ against targets $\{p, n, 2p, 3p, 4p, \dots\}$, modular quadratics, linear relations in the GLV basis, and two negative-control spaces (cross-modulus relations between β and λ; small linear relations involving the generator's coordinates)—and tests each with exact integer arithmetic. It found 22 identities and 0 control hits; every identity is explained by β, λ being cube roots of unity and by Proposition 4.2, which the search itself prompted when it reported $t^2+3b_1^2=4p$ (hypotheses H02, H07).

Stage 4: experimental evaluation. Every experiment is a script that takes a seed and an output path and writes a JSON document with a provenance block (host, CPU, RAM, OS, library versions, command line, timestamp, elapsed time) through a single helper (`wp34.provenance.record`). Expensive computations use a Rust engine for the ECDLP and native SAT solvers for SHA-256, distributed over the MacLustr cluster (Section 9); the laptop runs reference checks and figures. A result that is not backed by a saved JSON file does not appear in the paper.

Stage 5: formal verification. Statements that are decidable at the bit level are proved in Lean 4 with `bv_decide`, which compiles a bit-vector goal to SAT, calls CaDiCaL and checks the returned LRAT proof inside the kernel. Statements about abstract groups are proved against an explicit axiomatisation. Section 11 lists exactly what is machine-checked; everything else in the paper is an informal proof or a measurement, and conjectures are labelled as such.

Stage 6: scientific evaluation. Each hypothesis receives one of six labels—new verified result, known result independently reproduced, conditional result, empirical observation, rejected conjecture, unresolved direction—after a literature comparison. The rubric is applied conservatively: a result is "new" only if we found no prior statement of it and it is proved; a reproduction with sharper constants is still "known". The complete log is `research_log/discovery_log.jsonl`.

Integrity rules. Synthetic instances only; no real keys, addresses or blockchain data; no claim that a computation ran unless its output is saved; operation counts next to wall-clock; at least one negative control per experimental family; two independent implementations for every arithmetic component. Two development bugs caught by the last rule (a kangaroo reference whose retry ignored the seed, and a PARI output-parsing error) are logged as evidence that the rule does work (H23).

What the framework is not. It does not search a space of programs or proofs by itself in the sense of Fawzi et al. (2022), Romera-Paredes et al. (2024) or Trinh et al. (2024): the hypotheses are written by the agent, the identity search is a bounded enumeration, and the design-space search of H09 is a parameter sweep. Its contribution is the discipline—structured records, exact checks, provenance, formal verification, and an honest classifier—applied end to end by one autonomous agent on a problem where the temptation to over-claim is strong. We assess in Section 10 whether this discipline produced anything a careful literature reading would not have.

9. Experimental Design

9.1. Environment

All computations ran on Apple-silicon machines of the author's laboratory cluster (Table A1 in Appendix B): a laptop (Apple M5 Max, 18 cores, 48 GB) for development, reference checks, SymPy/PARI verification and figure generation; two Mac Studio nodes (Apple M3 Ultra, 28 cores, 96 GB) and one Mac Studio (Apple M2 Ultra, 24 cores, 64 GB) for the ECDLP studies, the SAT/SMT grids and the second hardware point of the signature benchmarks. No GPU was used. Toolchains: Python 3.12 with exact integer arithmetic (SymPy 1.14, NumPy 2.5, python-sat 1.9 with CaDiCaL bindings, Z3 4.15), native CaDiCaL and Kissat binaries, PARI/GP 2.19 for point counting, Rust 1.95/1.96 for the ECDLP engine and the signature harness (`k256` 0.14, rust-secp256k1 0.33 binding `libsecp256k1`), Lean 4.35 for the formal layer. Every result file records the host, versions, seed and command line.

9.2. Baselines and validation

Before any novel configuration was tried, the standard algorithms were implemented twice and cross-validated (Hypothesis H23): a pure-Python reference (`src/wp34`, affine coordinates, one inversion per group operation, explicit operation counters) and a Rust engine (`rho_engine`: Montgomery arithmetic for $p < 2^{62}$, simultaneous inversion across batched walks, ChaCha20-seeded multipliers, per-thread counters and CPU timers). The engine passes 8 Rust unit/integration tests (including

1 052 brute-forced small-curve instances) and 553 Python tests on a fixture of 60 prime-order curves of 20–32 bits, every answer being re-verified by the Python reference as $[d]G = Q$; each engine result additionally carries an `ok` flag set by an in-process re-verification. The signature harness passes the 19 BIP340 vectors with both libraries, 1 000 cross-library checks per scheme and 200 checks against the Python reference. SHA-256 is verified against `hashlib` for ten message lengths spanning the padding boundaries, and the CNF encoder against the reference for $r \in \{1, 4, 16, 64\}$ steps.

9.3. Synthetic ECDLP instances

Instances are random prime-order curves $y^2 = x^3 + ax + b$ over random primes of the target bit length, counted with PARI/GP's `ellcard` (SEA) and retained only if $\#E(\mathbb{F}_p)$ is prime and $\neq p$; the $j = 0$ family uses $a = 0$ and $p \equiv 1 \pmod 3$, with β and λ computed and checked ($\phi(G) = [\lambda]G$) before use. Targets are $Q = [d]G$ for uniformly random d. Five studies were run (Table 4); all jobs are single-threaded except the parallel-efficiency study, so that operation counts are deterministic given the seed. The distinguished-point density follows the engine default $2^{-(\lfloor \text{bits}/4 \rfloor - 2)}$ and each thread runs 128 interleaved walks, which fixes the post-collision overhead at about $128 \cdot 2^{\lfloor \text{bits}/4 \rfloor - 2}$ extra iterations—negligible above 44 bits ($< 3\%$) but visible at 20–28 bits, which is why pooled constants are computed on ≥ 44 bits and empirical exponents are also fitted on ≥ 32 bits only.

Table 4. ECDLP experimental configuration (seed 34; all instances verified).

Study	Curves	Sizes (bits)	Instances/size	Configurations
scaling, generic	random a, b	20–56 step 4	60	BSGS (≤ 48); rho Floyd, Brent (≤ 48); rho DP $r = 20$, $r = 128$; rho DP + negation ($r = 128$); kangaroo on $[0, n)$
scaling, $j = 0$	$a = 0$, $p \equiv 1\ (3)$	24–52 step 4	60	rho DP $r = 128$; + negation; + ϕ-orbits; + both ($r = 128$ and $r = 20$)
iteration-function sweep	random	36, 40	40	$r \in \{2, 3, 4, 8, 16, 20, 32, 64, 128\}$ $\times$ mixed walk $\in \{\text{none}, 2, 4\}$
parallel efficiency	random	56	12	rho DP + negation, threads $\in \{1, 2, 4, 8, 16, 28\}$ (M3 Ultra, 28 cores)
time–memory	random	36 (BSGS), 40 (rho)	20	BSGS memory $m \in \{2^{10}, \ldots, 2^{20}\}$; rho DP density $2^{-2}, \ldots, 2^{-14}$
preprocessing (Python)	random	22, 26	60	$T \in \{2^4, \ldots, 2^{12}\}$ stored distinguished points

Nodes: generic scaling, sweep and parallel study on M3U96a (Apple M3 Ultra); $j = 0$ scaling and time–memory on M2U64 (Apple M2 Ultra); preprocessing on the laptop. A 20-instance pilot of the two scaling studies (kept in `results/synthetic_ecdlp/pilot20/`) was used to size the final runs.

9.4. Measurement and statistics

For every run we record group additions and doublings (and their sum, the *group operations*), field inversions and multiplications, iterations of the walk, distinguished points stored and their bytes, fruitless cycles detected, wall-clock time, CPU time summed over threads and peak resident memory. Means are reported with bootstrap 95% confidence intervals (2 000 resamples); speed-ups between configurations on the same instances use paired resampling. Empirical exponents come from least squares on $\log T = \log a + \alpha \log n$ $(+\beta \log\log_2 n)$ with bootstrap intervals over the size-level means. Negative controls: an $r = 2$ walk (far from a random mapping) in the sweep; a random oracle in the SHA-256 diffusion and bias tests; a batch containing one invalid signature in the batch verification; and the control spaces Q4/Q5 of the identity search.

9.5. Signature and SHA-256 protocols

Signature benchmarks use ChaCha20-seeded synthetic keys, prehashed 32-byte messages, single-threaded timing with medians and interquartile ranges over 7 repetitions of 2 000 operations, and our own instrumented Straus and Pippenger multi-scalar multiplications for batch verification (operation counts recorded). The SHA-256 experiments use the reference implementation with r-step compression, seeded random messages, 2 000 samples per diffusion configuration and 20 000 per bias configuration, Tseitin CNF with ripple-carry adders, and solver time limits of 300 s (preimage) and 600 s (collision) with three seeds per cell; every satisfying assignment is re-hashed with the reference implementation before being counted.

10. Computational Results

10.1. Axis A: algebraic structure (H01–H07)

The verification of the secp256k1 parameters, the closed form of the GLV constants, the orbit counterexample and the index-calculus probe were presented with the theory in Section 4; Table 5 gives the operation counts behind hypothesis H03, and Table 6 the index-calculus measurements behind H06. The identity search tested 58 064 candidate relations and found 22, all explained (0 unexplained; negative controls Q4 = 0, Q5 = 0 hits, against an expected chance count of 10^{-73}); the orbit-compatibility test failed for 1500 of 1500 random pairs.

10.2. Axis A/B: ECDLP baselines and scaling (H08)

Every one of the 3 840 generic and 2 400 $j = 0$ instances was solved and verified (all verified). Figure 5 shows group operations and wall-clock time against n on log–log axes, and Figure 6 the

Table 5. Operation counts for one scalar multiplication on secp256k1 (affine model, 200 random 256-bit scalars, mean ± sd). GLV sub-scalars never exceeded 128 bits in 2000 samples.

Method	Additions	Doublings	Total	Speed-up vs. double-and-add
Double-and-add	127.2	254.1	381.3 ± 8.2	1.00×
wNAF, $w = 4$	53.7	254.8	308.5 ± 2.9	1.24×
wNAF, $w = 5$	49.1	254.3	303.4 ± 3.0	1.26×
GLV + interleaved wNAF, $w = 4$	56.6	127.8	184.4 ± 2.7	2.07×
GLV + interleaved wNAF, $w = 5$	56.2	127.4	183.6 ± 2.4	2.08×

Table 6. Index-calculus probe on prime-field curves: decomposition of a random point as a sum of two factor-base points, $F = \{P : x(P) < p^e\}$ ($e = 0.5$ or 0.6). "Oracle relations" is the cost of collecting $|F|$ relations with a free decomposition oracle; "Lin. alg." is $|F|^2$.

Bits	$\lvert F\rvert$	$\lvert F\rvert/\sqrt{n}$	Trials	Rate (meas.)	Rate (pred.)	Oracle relations$/\sqrt{n}$	Lin. alg.$/\sqrt{n}$
16	272	1.06	400	0.435	0.433	2.45	290
16	596	3.00	400	0.990	0.989	3.03	1786
20	1 022	1.00	300	0.393	0.395	2.55	1025
20	3 222	3.83	300	1.000	0.999	3.83	12346
24	3 872	0.99	200	0.375	0.389	2.65	3842
24	17 262	5.11	200	1.000	1.000	5.11	88234
28	15 002	1.00	120	0.308	0.394	3.25	15025

normalised cost $T/\sqrt{n}$; Table 7 lists selected sizes and Table 8 the fitted exponents.

Exponents. Restricted to ≥ 32 bits, every algorithm has $\hat{\alpha} \in [0.478, 0.500]$ with confidence intervals containing or adjacent to 0.5; on iterations the rho variants give $\hat{\alpha}$ between 0.49 and 0.50. The log-factor model $a\,n^{\alpha}(\log n)^{\beta}$ does not improve the fit ($R^2 > 0.99$ already) and returns β indistinguishable from 0. There is no evidence of any deviation from square-root scaling in this range, as expected.

Constants. Pooled over the 480 rho runs at ≥ 44 bits, the number of iterations is $1.254\sqrt{n}$ with 95% CI $[1.195, 1.313]$—the random-mapping prediction is $\sqrt{\pi/2} = 1.2533$, which the r-adding walk with $r \in \{20, 128\}$ reproduces to three digits. With the negation map the constant is 0.925 $[0.865, 0.986]$ against the prediction $\sqrt{\pi/4} = 0.8862$: a paired speed-up of 1.35 $[1.23, 1.48]$ versus the ideal $\sqrt{2} = 1.414$, the shortfall being the residual fruitless-cycle handling and the look-ahead rejections (Section 5). BSGS costs $1.500\sqrt{n}$ $[1.463, 1.539]$ group operations (the engine's x-keyed table resolves $\pm j$, so the giant phase costs $n/(2m)$ and the total is $1.5\sqrt{n}$ rather than $2\sqrt{n}$); the kangaroo on the full interval costs $1.938\sqrt{n}$ $[1.827, 2.049]$, close to the $2\sqrt{w}$ prediction. Floyd's cycle detection costs $3.00\sqrt{n}$ (three evaluations per step) and Brent's $1.93\sqrt{n}$ at 48 bits, both as

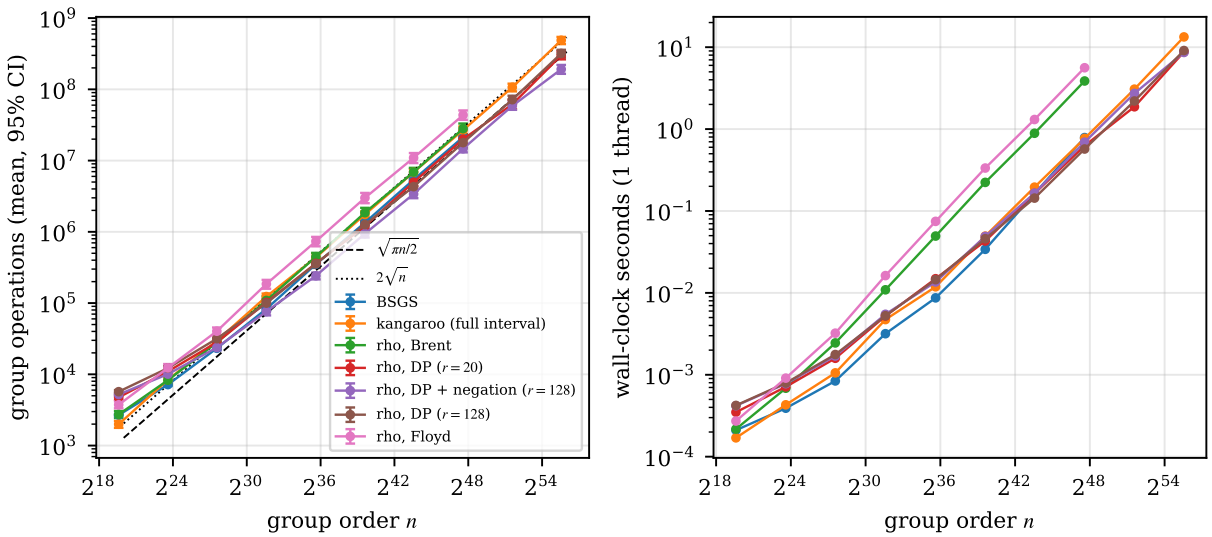


Figure 5. Empirical scaling of the generic ECDLP algorithms on random prime-order curves (60 instances per size, single thread, Apple M3 Ultra). Left: group operations with bootstrap 95% intervals and the theoretical lines $\sqrt{\pi n/2}$ and $2\sqrt{n}$; right: wall-clock seconds. Generated by `analyze_ecdlp.py`.

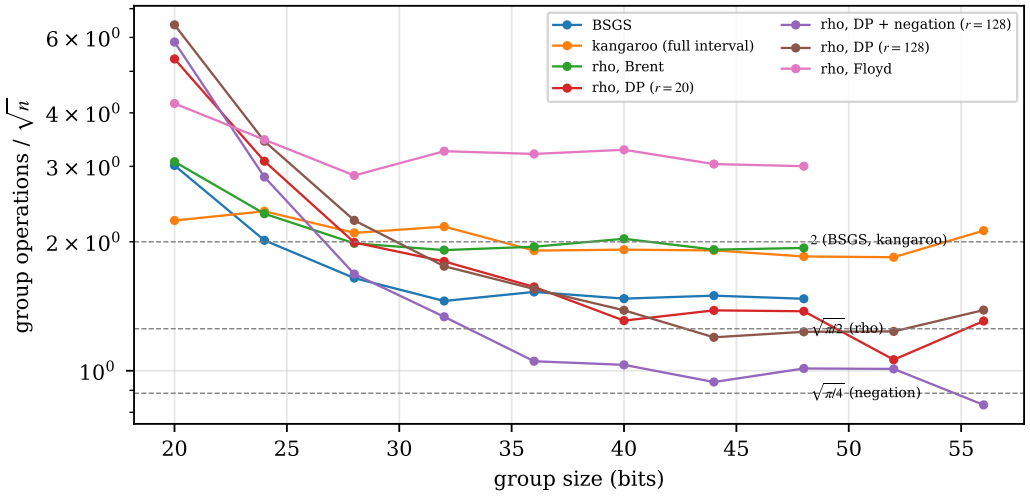


Figure 6. Normalised cost $T(n)/\sqrt{n}$ by size. The small-size excess of the distinguished-point variants is the post-collision overhead of 128 interleaved walks; above 44 bits the constants settle at their theoretical values.

Table 7. ECDLP baselines at selected sizes (mean group operations, bootstrap 95% CI, normalised constant, theoretical constant, wall-clock).

Algorithm	Bits	Inst.	Group ops (mean)	95% CI	ops/$\sqrt{n}$	Theory	Wall (s)
BSGS	20	60	2 679	[2 603 2 758]	3.01	2.00	0.000
	28	60	23 209	[22 061 24 380]	1.65	2.00	0.001
	36	60	350 878	[331 254 370 841]	1.53	2.00	0.009
	44	60	5 388 261	[5 127 106 5 639 773]	1.50	2.00	0.162
	48	60	21 353 478	[20 208 948 22 505 958]	1.47	2.00	0.787
kangaroo (full interval)	20	60	1 993	[1 773 2 215]	2.24	2.00	0.000
	28	60	29 593	[26 329 33 187]	2.10	2.00	0.001
	36	60	438 187	[382 001 500 060]	1.91	2.00	0.012
	44	60	6 871 559	[6 144 280 7 617 726]	1.91	2.00	0.195
	48	60	26 808 039	[23 569 625 30 122 184]	1.85	2.00	0.770
	52	60	106 707 359	[93 650 879 120 096 160]	1.84	2.00	3.074
	56	60	488 403 991	[431 328 685 544 229 758]	2.12	2.00	13.349
rho Brent	20	60	2 734	[2 433 3 059]	3.08	1.44	0.000
	28	60	27 973	[24 404 31 616]	1.98	1.44	0.002
	36	60	447 331	[389 033 506 702]	1.95	1.44	0.049
	44	60	6 899 435	[5 901 830 7 907 650]	1.92	1.44	0.890
	48	60	28 059 030	[23 390 277 33 116 242]	1.93	1.44	3.867
rho DP ($r = 20$)	20	60	4 750	[4 387 5 123]	5.34	1.25	0.000
	28	60	28 110	[25 070 31 425]	1.99	1.25	0.002
	36	60	360 440	[312 002 414 020]	1.57	1.25	0.015
	44	60	4 978 245	[4 403 399 5 638 782]	1.38	1.25	0.164
	48	60	19 966 687	[17 089 674 23 087 087]	1.38	1.25	0.625
	52	60	61 520 224	[52 474 644 71 426 525]	1.06	1.25	1.872
	56	60	300 438 016	[263 457 604 340 736 432]	1.31	1.25	8.767
rho DP + negation ($r = 128$)	20	60	5 202	[4 949 5 473]	5.85	0.89	0.000
	28	60	23 719	[21 611 25 763]	1.68	0.89	0.002
	36	60	241 754	[214 928 267 329]	1.05	0.89	0.014
	44	60	3 387 252	[2 965 303 3 843 106]	0.94	0.89	0.165
	48	60	14 680 781	[13 034 824 16 416 680]	1.01	0.89	0.693
	52	60	58 546 774	[51 602 632 65 730 359]	1.01	0.89	2.729
	56	60	191 425 086	[165 120 361 220 056 040]	0.83	0.89	8.628
rho DP ($r = 128$)	20	60	5 706	[5 398 6 008]	6.42	1.25	0.000
	28	60	31 644	[28 414 34 838]	2.24	1.25	0.002
	36	60	356 473	[312 663 401 282]	1.55	1.25	0.015
	44	60	4 307 374	[3 767 110 4 915 064]	1.20	1.25	0.144
	48	60	17 878 425	[15 065 049 20 998 708]	1.23	1.25	0.571
	52	60	71 624 501	[62 610 474 81 330 698]	1.24	1.25	2.202
	56	60	318 624 520	[280 613 624 358 951 118]	1.39	1.25	9.104
rho Floyd	20	60	3 738	[3 336 4 173]	4.21	3.76	0.000
	28	60	40 290	[34 737 45 384]	2.86	3.76	0.003
	36	60	736 600	[626 244 849 037]	3.21	3.76	0.075
	44	60	10 929 879	[9 223 312 12 761 485]	3.04	3.76	1.309
	48	60	43 544 917	[37 332 520 50 531 812]	3.00	3.76	5.612

Table 8. Empirical complexity exponents $\hat{\alpha}$ in $T(n) = a n^{\alpha}$ (least squares on size-level means, bootstrap 95% CI). Fits on all sizes include the distinguished-point overhead at 20–28 bits; fits on ≥ 32 bits and on iterations remove it.

Algorithm	$\hat{\alpha}$ (all sizes, ops)	95% CI	R^2	$\hat{\alpha}$ (≥ 32 bits, ops)	95% CI	$\hat{\alpha}$ (≥ 32 bits, iterations)
BSGS	0.471	[0.445, 0.496]	0.9975	0.500	[0.495, 0.506]	0.509 [0.490, 0.526]
kangaroo (full interval)	0.493	[0.487, 0.499]	0.9998	0.498	[0.487, 0.509]	0.498 [0.487, 0.510]
rho, Brent	0.482	[0.467, 0.500]	0.9988	0.500	[0.491, 0.511]	0.501 [0.491, 0.513]
rho, DP ($r = 20$)	0.450	[0.425, 0.479]	0.9959	0.478	[0.463, 0.495]	0.490 [0.476, 0.502]
rho, DP + negation ($r = 128$)	0.438	[0.404, 0.478]	0.9922	0.480	[0.469, 0.498]	0.494 [0.486, 0.508]
rho, DP ($r = 128$)	0.445	[0.413, 0.483]	0.9940	0.483	[0.463, 0.506]	0.496 [0.482, 0.509]
rho, Floyd	0.489	[0.480, 0.504]	0.9993	0.492	[0.483, 0.502]	0.493 [0.483, 0.502]

expected.

10.3. Axis A: automorphism classes on $j = 0$ curves (H04)

Table 9 and Figure 7 give the result of hypothesis H04 on 180 paired instances at ≥ 44 bits (Apple M2 Ultra). In *iterations*, the order-6 orbits deliver a speed-up of 2.21 $[2.00, 2.45]$ with $r = 128$ and 2.42 $[2.18, 2.70]$ with $r = 20$ against the ideal $\sqrt{6} = 2.449$; the normalised constant with $r = 20$ is $0.499\sqrt{n}$ against the prediction $\sqrt{\pi/12} = 0.5117$. The negation map alone and the ϕ-orbits alone give 1.29 and 1.51 against $\sqrt{2}$ and $\sqrt{3}$. Fruitless cycles are rare at these parameters (0.00 per 1000 iterations with $r = 128$, 0.11 with $r = 20$) because the look-ahead removes most 2-cycles before they form. The theory is therefore reproduced in the quantity it speaks about.

In *wall-clock time*, however, the picture is different: canonicalisation raises the cost of one iteration from 38 ns (plain) to 56 ns (negation), 71 ns (ϕ-orbits) and 80 ns (both) in our engine, so the measured wall-clock speed-ups are 0.73, 0.75 and 0.95 respectively—the order-6 trick roughly breaks even and the negation map alone *loses* time. The reason is implementation-specific and instructive: our canonical representative literally computes $\min\{x, \beta x, \beta^2 x\}$ and $\min\{y, p - y\}$ with two field multiplications and several Montgomery conversions per step, whereas the record-setting implementations fold the comparison into the batched inversion and use cheaper sign conventions (Bos et al., 2010; Bernstein et al., 2011; Bos et al., 2012). The lesson for the mandate's question A2 is that equivalence classes reduce *iterations* by exactly the predicted factor, and that turning this into a time saving is an engineering problem whose outcome depends on the per-step cost of canonicalisation relative to a group addition (here about 0.6–1.1 additions).

10.4. Axis B: automated search over iteration functions (H09)

The sweep over r and mixed walks (25 configurations $\times$ 80 instances; Figure 8, Table 10) reproduces Teske's findings: $r = 2$ and $r = 3$ walks need $3.77\sqrt{n}$ and $2.26\sqrt{n}$ iterations (the negative control behaves as a bad random mapping should), $r \geq 8$ walks cluster around the random-mapping

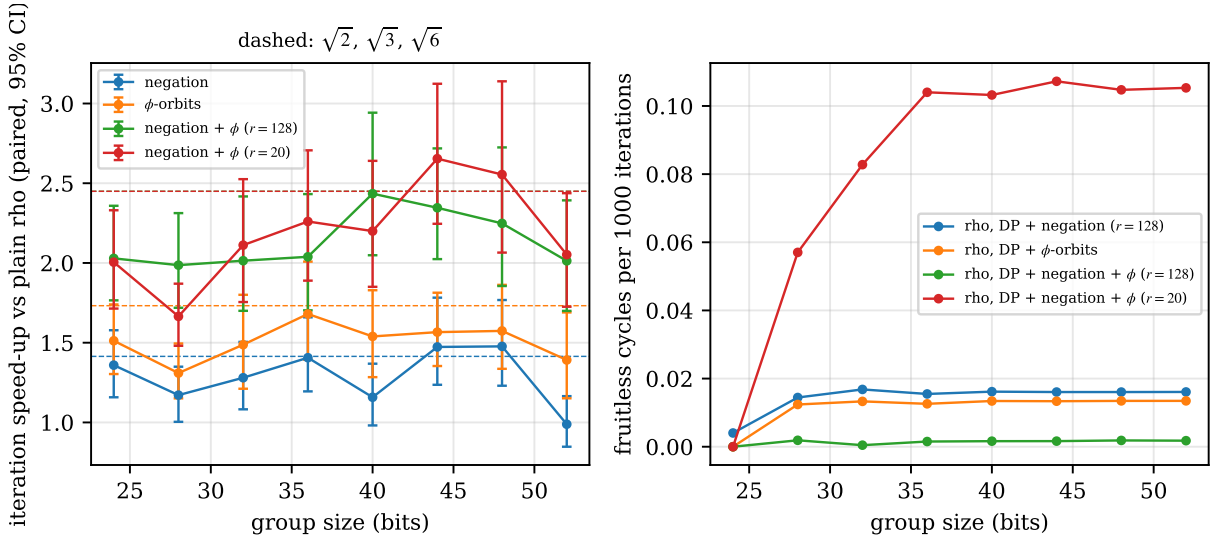


Figure 7. Endomorphism benchmarking on synthetic $j = 0$ curves. Left: paired iteration speed-up over plain rho by size, with the ideal $\sqrt{2}$, $\sqrt{3}$, $\sqrt{6}$ (dashed); right: fruitless cycles detected per 1000 iterations.

Table 9. Equivalence classes on $j = 0$ curves: iterations, paired iteration speed-up, cost per iteration and wall-clock speed-up (pooled over sizes ≥ 44 bits for iterations, ≥ 40 bits for time; Apple M2 Ultra, one thread).

Classes	Theory	Iterations/$\sqrt{n}$ (95% CI)	Iteration speed-up (paired, 95% CI)	ns per iteration	Wall-clock speed-up
none (plain rho, $r = 128$)	$\sqrt{\pi/2} = 1.253$	1.206 [1.114, 1.299]	1	38	1
$\{\pm P\}$ (negation)	$\sqrt{2} = 1.414$	0.933 [0.865, 1.006]	1.293 [1.158, 1.439]	56	0.73 [0.65, 0.83]
$\{P, \phi P, \phi^2 P\}$	$\sqrt{3} = 1.732$	0.797 [0.742, 0.849]	1.515 [1.375, 1.674]	71	0.75 [0.65, 0.86]
order-6 orbits, $r = 128$	$\sqrt{6} = 2.449$	0.546 [0.510, 0.581]	2.209 [2.001, 2.450]	80	0.95 [0.84, 1.09]
order-6 orbits, $r = 20$	$\sqrt{6} = 2.449$	0.499 [0.462, 0.536]	2.419 [2.185, 2.702]	83	0.95 [0.83, 1.08]

constant, and mixed walks make no measurable difference at these sizes. The best cell (r32_m0, 1.095 $[0.921, 1.284]$) is not distinguishable from the random-mapping value given the width of the intervals, and no configuration is significantly below it. This is the expected negative result: an automated search confined to generic iteration functions can at best recover the random-mapping constant, which Shoup's bound makes the floor up to the class-size factor. We record it as a rejected shortcut, with the positive by-product that the search ranks the designs correctly and quantifies the penalty for small r.

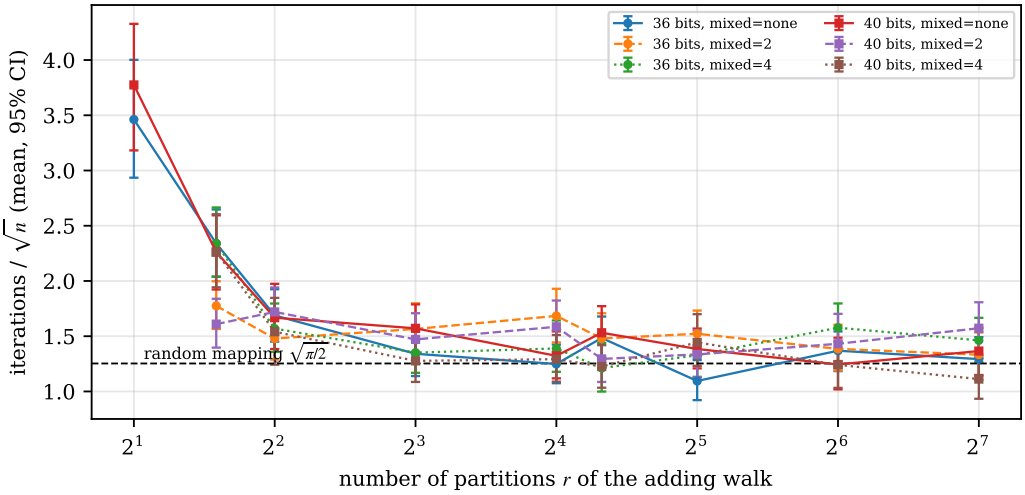


Figure 8. Iteration-function design sweep: iterations$/\sqrt{n}$ against the number of partitions r for plain and mixed r-adding walks at 36 and 40 bits (40 instances per cell, bootstrap 95% CI). The dashed line is the random-mapping constant $\sqrt{\pi/2}$.

Table 10. Iteration-function sweep (iterations$/\sqrt{n}$, mean of 40 instances per cell).

r	36 bits: iterations$/\sqrt{n}$			40 bits: iterations$/\sqrt{n}$		
	plain	mixed 2	mixed 4	plain	mixed 2	mixed 4
2	3.462	—	—	3.775	—	—
3	2.340	1.775	2.340	2.262	1.609	2.262
4	1.688	1.479	1.569	1.668	1.720	1.540
8	1.341	1.566	1.351	1.571	1.471	1.277
16	1.248	1.684	1.389	1.323	1.586	1.292
20	1.484	1.479	1.212	1.531	1.295	1.226
32	1.095	1.522	1.328	1.384	1.335	1.444
64	1.369	1.385	1.577	1.244	1.432	1.241
128	1.293	1.333	1.462	1.366	1.572	1.112
random-mapping constant $\sqrt{\pi/2}$	1.253			1.253		

10.5. Axis B: parallel efficiency and time–memory trade-offs (H10, H11)

Distinguished-point collision search on 56-bit instances scales to 14.8× on 16 threads (efficiency 0.93) and 22.1× on 28 threads (efficiency 0.79) of the Apple M3 Ultra, with total group operations within a factor 1.02 of the single-thread count (Figure 9, Table 11); the efficiency loss at 28 threads reflects the mix of performance and efficiency cores and the shared table, not an algorithmic limit. The time–memory study (Figure 10, Table 12) confirms the deterministic BSGS trade-off $m + n/(2m)$ within 7.8% over memory sizes from 2^{10} to 2^{20} entries, and shows the two-sided trade-off of distinguished-point rho: dense distinguished points (density 2^{-2}) store many points and pay a restart cost per point, sparse ones (density 2^{-14}) pay the post-collision overhead, and the optimum at 40 bits lies between densities 2^{-8} and 2^{-10}, where the cost is within 5% of $\sqrt{\pi n/2}$ with one to four thousand stored points.

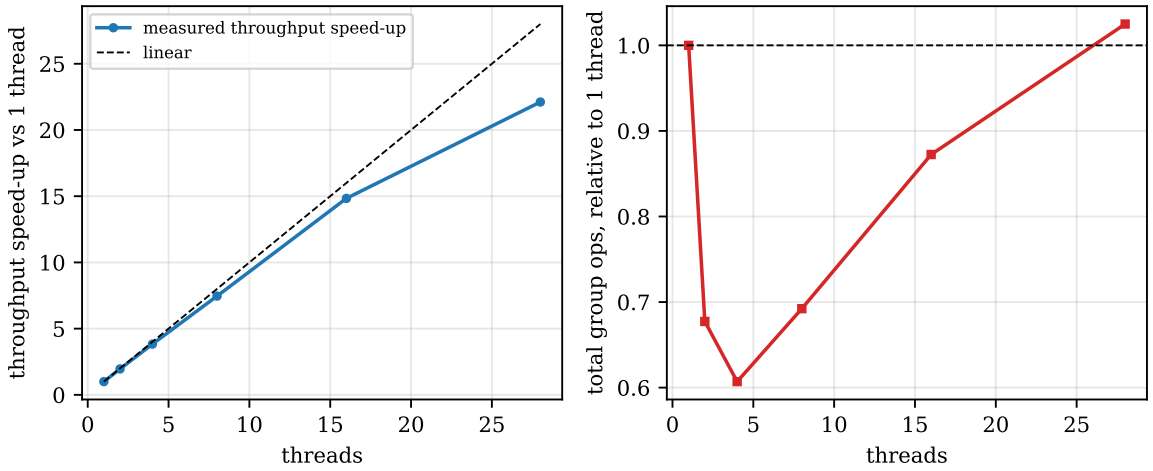


Figure 9. Parallel collision search (56-bit instances, rho with negation map, Apple M3 Ultra, 12 instances per point): throughput speed-up and total work relative to one thread.

Table 11. Parallel efficiency of distinguished-point rho (56 bits, negation map, Apple M3 Ultra).

Threads	Wall (s)	Group ops ($\times 10^6$)	Mops/s	Speed-up	Efficiency
1	8.78	229.6	26	1.00	1.00
2	3.04	155.5	51	1.96	0.98
4	1.39	139.4	100	3.83	0.96
8	0.81	158.9	195	7.46	0.93
16	0.52	200.3	388	14.83	0.93
28	0.41	235.3	578	22.11	0.79

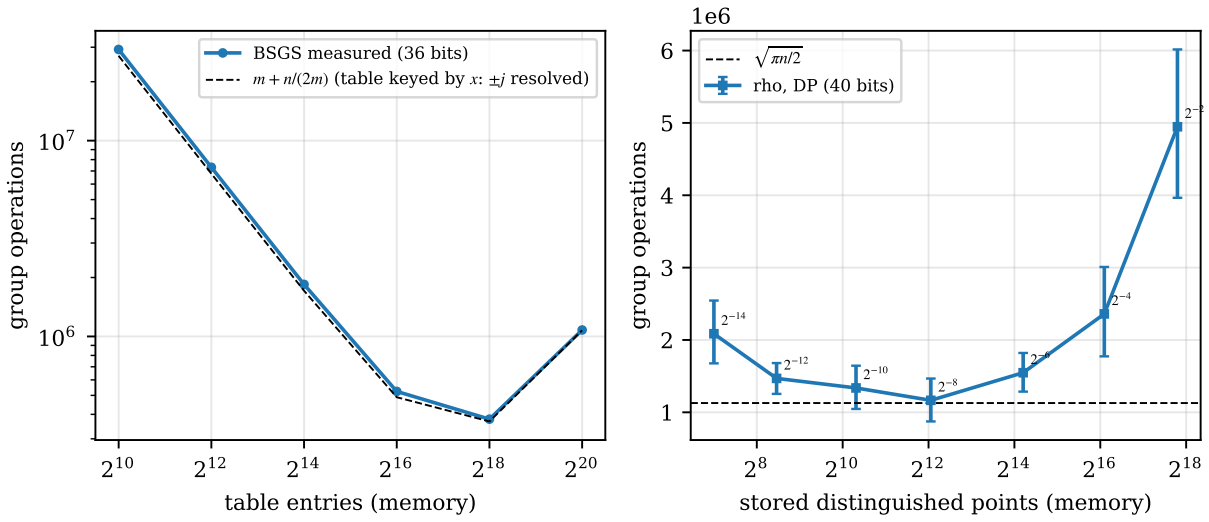


Figure 10. Time–memory trade-off. Left: BSGS at 36 bits against the memory m (dashed: $m + n/(2m)$). Right: distinguished-point rho at 40 bits against the number of stored points (labels: DP density), with bootstrap 95% intervals; dashed: $\sqrt{\pi n/2}$.

Table 12. Time–memory trade-off measurements (means over 20 instances).

Algorithm / parameter	Memory (entries)	Memory (MB)	Group ops	Predicted	Wall (s)
BSGS $m = 2^{10}$ (36 bits)	1 023	0.0	29 225 447	27 125 293	0.613
BSGS $m = 2^{12}$ (36 bits)	4 095	0.1	7 311 605	6 785 163	0.161
BSGS $m = 2^{14}$ (36 bits)	16 383	0.5	1 844 512	1 711 651	0.041
BSGS $m = 2^{16}$ (36 bits)	65 535	1.9	523 838	489 353	0.015
BSGS $m = 2^{18}$ (36 bits)	262 143	7.4	378 053	368 098	0.013
BSGS $m = 2^{20}$ (36 bits)	1 048 575	29.8	1 078 839	1 075 065	0.031
rho DP density 2^{-2} (40 bits)	227 779	28.88	4 946 673	1 128 106	0.499
rho DP density 2^{-4} (40 bits)	69 926	7.22	2 360 214	1 128 106	0.181
rho DP density 2^{-6} (40 bits)	18 867	1.80	1 547 360	1 128 106	0.076
rho DP density 2^{-8} (40 bits)	4 250	0.45	1 165 676	1 128 106	0.042
rho DP density 2^{-10} (40 bits)	1 270	0.11	1 336 710	1 128 106	0.042
rho DP density 2^{-12} (40 bits)	352	0.06	1 468 980	1 128 106	0.043
rho DP density 2^{-14} (40 bits)	128	0.06	2 085 998	1 128 106	0.062

10.6. Axis B: preprocessing (H12)

Table 13 reports the Python reference experiment on 22- and 26-bit groups (60 instances per cell). The online cost is $1.55\sqrt{n/T}$ on average (range 1.48–1.67 across T from 2^4 to 2^{12}), so the per-instance speed-up over plain rho grows as $\sqrt{T}$ and reaches 42× at $T = 2^{12}$—at the price of an offline phase of about $\sqrt{nT}$ operations. The constant ≈ 1.5 rather than 1 comes from the walk-length distribution (a walk from Q must travel to a distinguished point *and* that point must have been stored) and matches the heuristic analysis of Bernstein and Lange (2013a). The trade-off is exactly the $ST^2 = \tilde{\Omega}(n)$ curve of Corrigan-Gibbs and Kogan (2018) and offers nothing below $\sqrt{n}$ in total work.

Table 13. Discrete logarithms with preprocessing (Python reference, 20-partition walk independent of the target).

Bits	T	W	Offline ops	Online ops (mean ± 95% CI)	$\sqrt{n/T}$	Ratio	Speed-up per instance
22	16	463	8 077	687 ± 110	464	1.48	2.7×
22	64	226	28 781	376 ± 66	226	1.66	4.8×
22	256	123	61 532	195 ± 31	124	1.58	10.2×
22	1024	55	160 700	85 ± 14	55	1.54	20.8×
22	4096	31	389 668	48 ± 11	31	1.55	41.3×
26	16	1600	51 315	2429 ± 431	1601	1.52	2.6×
26	64	866	79 772	1305 ± 192	866	1.51	5.3×
26	256	488	262 246	724 ± 119	489	1.48	10.8×
26	1024	226	480 415	378 ± 63	226	1.67	19.2×
26	4096	114	1 083 775	173 ± 34	114	1.51	42.3×

10.7. Axis B: signature benchmarks (H13, H14)

Table 14 and Figure 11 summarise the signature measurements. Three results stand out. (i) Implementation choice spans three orders of magnitude: the pure-Python reference verifies a BIP340 signature in 7.4 ms, `libsecp256k1` in 13.2 μs (562×), with `k256` at 24.5 μs and coincurve (`libsecp256k1` through Python) essentially at the C speed. (ii) ECDSA and Schnorr verification cost the same in `libsecp256k1` within 5%, and signing differs between libraries more than between schemes: `k256` signs faster (table-based fixed-base multiplication) while `libsecp256k1` verifies 1.9× faster (variable-base multiplication with GLV and its optimised field arithmetic). (iii) Batch verification with our Pippenger multi-scalar multiplication reaches 7.9 μs per signature at batch size 2048, a speed-up of 3.10 $[3.05, 3.16]$ over single verification in the same arithmetic and 1.67× over `libsecp256k1`'s single verification; Straus interleaving plateaus near 2× from batch size 32, and Pippenger overtakes it only beyond about 256 signatures. About 3 μs per signature of non-MSM work (the square root in `lift_x` and the tagged hash) bounds the attainable gain; the batch with

one corrupted signature was rejected by every back-end. The same curves on the Apple M3 Ultra are 1.3× slower with identical shape. All of this is known in kind (Bernstein et al., 2012; Wuille et al., 2020b); the contribution is the measured, operation-counted curve for secp256k1 and the identification of the saturation mechanism.

Table 14. Signature implementations: median time per operation in microseconds (single thread, 7 repetitions of 2 000 operations; laptop Apple M5 Max unless stated).

Implementation	ECDSA sign	ECDSA verify	Schnorr sign	Schnorr verify
`libsecp256k1` (C, via rust-secp256k1)	23.6	12.6	20.5	13.2
`k256` (Rust, RustCrypto)	15.3	25.0	11.6	24.5
`libsecp256k1`, Mac Studio M3 Ultra	29.2	16.7	24.7	16.9
`k256`, Mac Studio M3 Ultra	18.8	33.5	14.8	32.5
pure Python reference (`src/wp34`, affine)	3812.6	7237.1	14713.1	7417.2
python-ecdsa (pure Python, ECDSA only)	118.0	419.4	—	—
coincurve (`libsecp256k1` via Python/CFFI)	11.9	12.7	28.3	12.5

11. Formal Results and Negative Findings

11.1. Machine-checked statements

The formal layer is a Lean 4.35 project (`formal/`, core library only, no Mathlib) that builds from a clean state in 32 s on the laptop with no `sorry` and no user axiom other than the explicit fields of the abstract group structure. Bit-vector goals are closed by `bv_decide`, which bit-blasts to an and-inverter graph, calls the CaDiCaL solver bundled with Lean, and re-checks the returned LRAT certificate with Lean's verified checker (Böving et al., 2025; de Moura and Ullrich, 2021); a static test (`tests/test_formal_lean.py`) enforces the absence of `sorry`, `axiom`, `native_decide` and Mathlib, and optionally re-runs the build. Table 15 lists the headline theorems; `formal/README.md` lists all 80-odd lemmas with the size of each SAT instance, the exact build output and a record of the attempts that timed out.

Three points deserve comment. First, the formal layer corrected a belief we held when writing the hypothesis portfolio: we expected σ_0 and σ_1 to be non-injective because they contain a logical shift, and the SAT solver returned UNSAT for the injectivity goal—both are bijections (hypothesis H22, rejected), which matters for any analysis that inverts the message schedule. Second, the 9-step local collision required a correction vector slightly different from the one we had first written down from memory (δW_{i+1} must cancel both $\Sigma_1\delta$ from the e-register and $\Sigma_0\delta$ from the a-register); the formal derivation is in the file header and the Python simulation of Axis C agrees. Third, what is *not* formalised is as important as what is: curve arithmetic, the random-mapping analysis, the

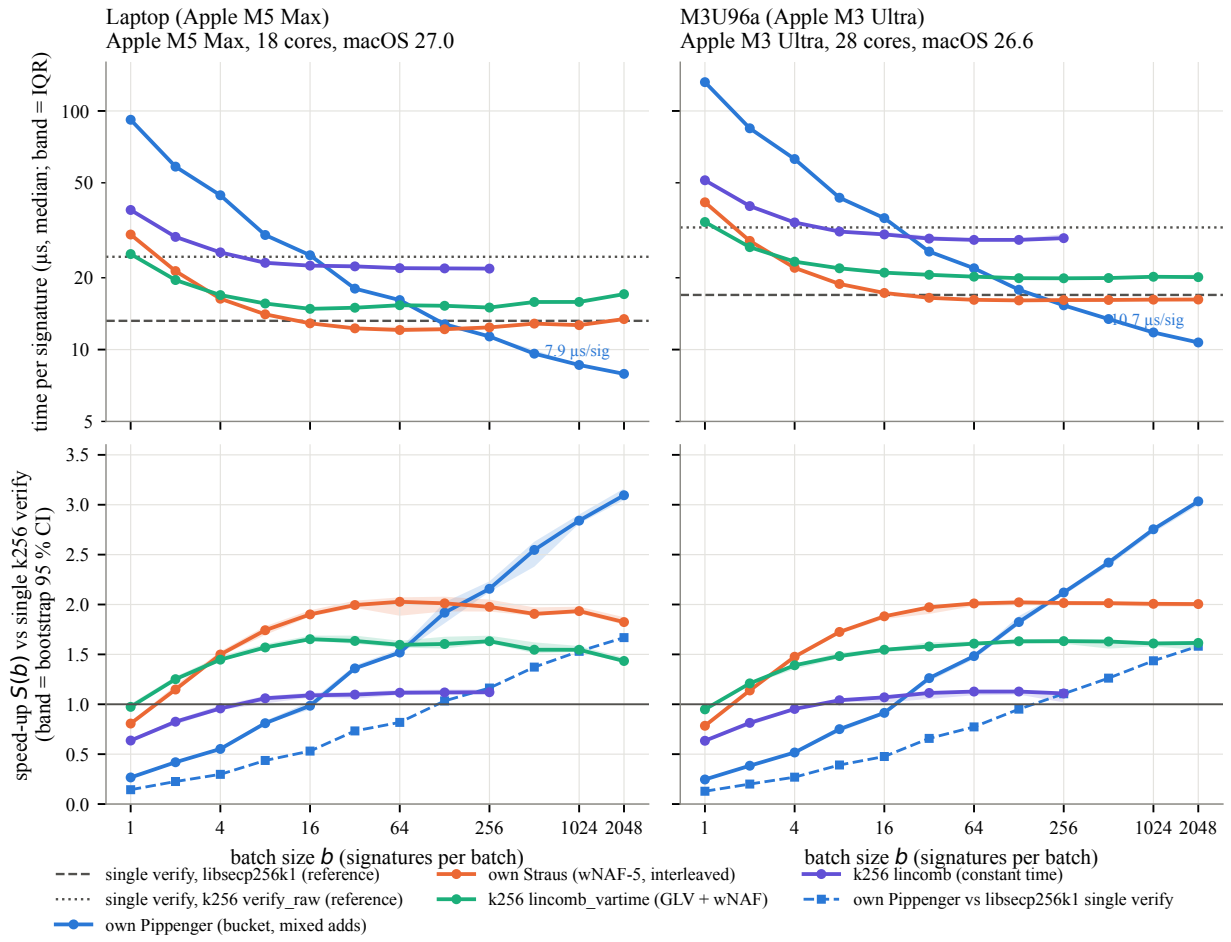


Figure 11. BIP340 batch verification: time per signature against batch size for Straus, Pippenger and the `k256` linear-combination routines on two hosts, with the single-verification reference lines, and the speed-up with bootstrap 95% intervals. Generated by `figures/make_sig_figures.py`.

Table 15. Machine-checked statements (Lean 4.35, `bv_decide` with LRAT certificates unless stated). AIG = nodes of the bit-blasted instance.

Theorem(s)	Statement	Method (AIG)
`Ch_opt`, `Maj_opt_or`, `Maj_opt_xor`	$\mathrm{Ch}(x,y,z) = ((y \oplus z) \wedge x) \oplus z$; $\mathrm{Maj} = (x \wedge (y \vee z)) \vee (y \wedge z) = ((x \oplus y) \wedge (y \oplus z)) \oplus y$	SAT (608–768)
`Sigma0_xor`, …, `sigma1_xor`	$\mathbb{F}_2$-linearity of $\Sigma_0, \Sigma_1, \sigma_0, \sigma_1$	SAT (870–960)
`Sigma0_injective`, `Sigma1_injective`, `sigma0_injective`, `sigma1_injective`	all four maps are injective on 32 bits	SAT, UNSAT certificates (644–704)
`*_bijective` (4)	explicit inverses: Σ_0^{-1} = XOR of 17 rotations, $\sigma_0^{-1}, \sigma_1^{-1}$ as 32×32 $\mathbb{F}_2$ matrices; both compositions are the identity	byte-split SAT (33–4 817) + rewriting
`shr3_not_injective`, `sigma0_rotations_not_injective`	controls: each summand of σ_0 is singular	`decide`
`stepInv_step`, `step_stepInv`, `step_bijective`, `steps_bijective`	the SHA-256 step is a bijection of the 256-bit state for fixed (K_t, W_t), and so is any sequence of steps	two 160-bit SAT lemmas (2 048, 2 420) + rewriting; direct 320-bit SAT timed out
`scheduleStep_inv`, `schedule_backward`	$W_{t-16} = W_t \ominus \sigma_1(W_{t-2}) \ominus W_{t-7} \ominus \sigma_0(W_{t-15})$; any 16 consecutive schedule words determine all 64	SAT (1 921) + recursion
`localCollision_absorb`, `localCollision_xor3`	in the XOR-linearised step (with Ch = Maj = 0, resp. Ch = Maj = $x \oplus y \oplus z$), for all states, constants, messages and δ: the correction pattern $(\delta, \Sigma_1\delta \oplus \Sigma_0\delta, \Sigma_1(\Sigma_0\delta), 0, \delta, \Sigma_0\delta \oplus \Sigma_1\delta, 0, 0, \delta)$ cancels after 9 steps	difference version by byte-split SAT (5 049 / 8 769), lifted by proven linearity of the 9-step map
`absorb_corrections_not_collide`	control: the same corrections do *not* give a collision of the real (modular) 9-step map on a concrete input	`decide`
`schnorr_correct`, `ecdsa_correct`, `ecdsa_malleable`	in an axiomatised cyclic group with $x(-P) = x(P)$: $s \equiv k + ed \Rightarrow [s]G = R + [e]P$; ECDSA verification is correct; (r,s) valid $\Rightarrow$ $(r, n-s)$ valid	rewriting from the structure fields (no SAT)

Every `bv_decide` theorem depends on the per-theorem axiom Lean 4.35 records for the natively executed LRAT check (same trust base as `Lean.ofReduceBool`); the group-theoretic theorems depend only on `propext` and `Quot.sound`. Full `#print axioms` output in `formal/build.log`.

probability of the real local collision and every timing claim remain informal, and no conjecture appears as a theorem.

11.2. Negative findings

The mandate counts a rigorously documented failure as a result (Outcome D). Table 16 collects the negative findings of this study with the form of evidence behind each.

Table 16. Negative findings: candidate shortcuts and why they fail.

Candidate	Why it fails	Evidence
A further efficiently computable endomorphism of secp256k1	$\mathrm{End}(E) = \mathbb{Z}[\omega]$ is the maximal order; every endomorphism is $u + v\phi$	Proposition 4.1; exact check $t^2 - 4p = -3f^2$
A group structure on the automorphism orbits (DLP of size $n/6$)	the projection would be injective; orbits are not a congruence	Proposition 4.3; $1500/1500$ counterexamples
Index calculus with a factor base $x(P) < p^{1/2}$ over $\mathbb{F}_p$	decomposition costs $\lvert F\rvert$ per trial without an oracle; with an oracle, relations cost $\geq 2.4\sqrt{n}$ and linear algebra $\lvert F\rvert^2$	Table 6; Semaev S_3 check S4
A transfer attack (MOV/Frey–Rück, anomalous, twist)	embedding degree 253-bit; $n \neq p$; twist cofactor a 220-bit prime	Table 3
A better generic iteration function for rho	no design in a 25-point space beats the random-mapping constant; Shoup's bound is the floor	Figure 8, Table 10
Beating $\sqrt{n}$ in total work by preprocessing	online $1.5\sqrt{n/T}$ after $\sqrt{nT}$ offline; $ST^2 = \tilde{\Omega}(n)$	Table 13
A verification shortcut avoiding one scalar multiplication	any predicate equivalent to $[s]G = R + [e]P$ needs a group element depending on both s and e	Section 6 (argument, not a theorem)
An unexplained identity among the secp256k1 constants	all 22 identities found follow from $\beta^2 + \beta + 1 \equiv 0$, $\lambda^2 + \lambda + 1 \equiv 0$ and the ideal structure; 0 control hits	`identity_search.json`
A SHA-256 regularity surviving to the full round count	diffusion complete and correlations undetectable within a small number of steps; SAT/SMT searches stall far below the published 31-step records	Section **??**

Two remarks keep these findings in proportion. First, each negative result is relative to a stated family of candidates: the endomorphism result is unconditional, the orbit result is a theorem, but the iteration-function and SAT results are statements about the spaces we searched. Second, none of the findings is surprising to a specialist; their value is that the pipeline produced them with evidence rather than with citations, and that it produced *no* positive claim it could not support.

12. Implications for Cryptographic Security

For secp256k1. Nothing in this study weakens the curve. The three structural features that make secp256k1 "special"—the generalised-Mersenne prime, $a = 0$, and complex multiplication

by $\mathbb{Z}[\omega]$—translate into exactly the constant factors the literature predicts and we measured: faster field reduction, cheaper doubling, a $2\times$ reduction in doublings for scalar multiplication through GLV, and a $\sqrt{6}$ reduction in the expected number of rho iterations (Section 10). Proposition 4.1 shows that the endomorphism ring is as small as it can be given $j = 0$, so there is no further endomorphism to exploit; Proposition 4.3 shows that the automorphism orbits cannot be turned into a smaller group; and the exact exclusion table (Section 4) confirms that every known algebraic transfer is inapplicable. The generic bound therefore stands: solving one secp256k1 instance costs about $\sqrt{\pi n/12} \approx 2^{126.7}$ group operations with the full automorphism trick, roughly 2^{70} times the largest ECDLP computation ever completed. Preprocessing does not change this conclusion: the per-instance cost $\sqrt{n/T}$ after a table of T points requires $\sqrt{nT}$ work up front, and a table reducing each instance to 2^{80} operations would itself cost 2^{174} operations and 2^{93} entries. Our measurements (Hypothesis H12) put the constant at about 1.5, which does not alter the exponent.

For the signature schemes. The security difference between ECDSA and BIP340 is structural: Schnorr's linear response $s = k + ed$ admits a clean reduction and enables batch verification, aggregation and threshold protocols; ECDSA's k^{-1} obstructs all three and forces the proofs into stronger idealisations. Operationally the two schemes cost the same to verify in `libsecp256k1` (within 5%), so the choice is about provable security and protocol features, not speed. Batch verification is the one verification-side improvement that is both real and consensus-neutral; its gain is bounded by the non-linear-algebra work per signature (square root, hashing) and saturates near $3\times$ in our measurements. The quantities that actually decide security in practice are nonce generation and side-channel resistance, which deterministic nonces and constant-time scalar multiplication address; none of the algebra in this paper changes that assessment.

For SHA-256. Our reduced-step experiments reproduce the easy end of the known picture: full diffusion within a few steps, no detectable bias beyond a handful of steps, SAT preimages and collisions feasible only far below the published dedicated-tool records. The step function being a bijection and the schedule being invertible are reminders that the compression function alone offers no preimage resistance—the Merkle–Damgård feed-forward does—and that "free-start" results must not be confused with attacks on the hash. The security margin of SHA-256 is the distance between 31 (practical collision), 37 (theoretical collision), 45 (theoretical preimage) and 64 steps; nothing here narrows it.

For automated discovery. The pipeline's value was not a new attack but the discipline it imposed: every candidate shortcut was either explained by existing theory, measured to be a constant, or refuted. An autonomous agent working on cryptographic primitives is exposed to a specific failure

mode—mistaking a constant-factor or representation improvement for a complexity improvement—and the classification rubric, the operation counts and the formal layer are the controls against it. We regard the fact that the agent produced zero claims of a break, while producing a closed-form explanation of a set of implementation constants and a dozen quantified reproductions, as the correct outcome.

Responsible-disclosure assessment. No security-relevant result emerged that would require disclosure. All instances attacked were synthetic; no real key, address or blockchain datum was used or produced; the code released with the paper attacks only the reduced-size curves it generates itself.

Quantum computing. The analysis above is classical. Shor's algorithm (Shor, 1997) solves the ECDLP in polynomial time on a fault-tolerant quantum computer, with concrete resource estimates for 256-bit curves in the range of a few thousand logical qubits (Roetteler et al., 2017); this, not any classical shortcut, is the known threat to `secp256k1`, and it is independent of the curve's special structure.

13. Limitations

- **Scale.** The ECDLP experiments run on groups of at most 2^{48}–2^{56} elements. The fitted exponents and constants describe this range; extrapolation to 2^{256} rests on the random-mapping heuristic and on Shoup's bound, not on our data. We state no result about `secp256k1` itself that is not an exact algebraic verification.
- **Generic design space.** The iteration-function search (H09) explores a bounded parameter space of r-adding and mixed walks; it cannot find a non-generic idea, and its negative result is a statement about that space.
- **SHA-256 encodings.** Our CNF and SMT encodings are generic (Tseitin, ripple-carry adders) and unguided by differential characteristics; the step counts they reach are therefore far below the literature's and should be read as a calibration of generic solvers, not as a limit of SAT-based cryptanalysis (Alamgir et al., 2024).
- **Formal coverage.** The Lean statements cover bit-level identities and abstract-group facts. They do not formalise the curve arithmetic, the random-mapping analysis or any probabilistic claim; those remain informal.
- **Timing measurements.** Wall-clock figures come from two Apple Silicon machines and depend on the toolchains used; operation counts are the portable quantity. Batch-verification speed-ups are relative to the same arithmetic library and would differ against a hand-optimised assembly

implementation.

- **Novelty.** The one algebraic statement we did not find written explicitly (the closed form of the `libsecp256k1` GLV constants) is a direct consequence of classical CM theory; we claim exposition, not discovery, and a reader who knows a prior statement of it should treat it as a reproduction.
- **Single agent, single run.** The pipeline was executed once by one autonomous agent; we do not measure variance across runs or agents, and the methodological assessment (H24) is qualitative. A controlled comparison against a baseline agent without the pipeline's discipline is left to future work.
- **Literature verification.** Bibliographic entries were verified against public records; where a venue record omitted a field (volume numbers of some Springer series), it was filled from series knowledge and flagged in `literature/unverified.md`.

14. Conclusion

We asked whether an autonomous, hypothesis-driven exploration of the mathematics behind Bitcoin's primitives could find a shortcut past the square-root barrier, and we built the apparatus to answer the question honestly: a verified literature base, twenty-five falsifiable hypotheses, exact symbolic checks, a native experimental engine with operation counts and provenance, machine-checked statements, and a classification rubric that does not reward enthusiasm. The answer is no, with the following specifics.

The algebraic structure of secp256k1 is fully accounted for by complex multiplication by $\mathbb{Z}[\omega]$: it yields the $\sqrt{6}$ reduction in rho iterations and the halving of doublings in scalar multiplication, it determines the GLV constants of `libsecp256k1` in closed form, and it leaves no room for a further endomorphism or for a smaller quotient group. The generic algorithms behave exactly as the random-mapping theory predicts on 6 000 synthetic instances: $T(n) \propto n^{0.5}$, $1.254\sqrt{n}$ iterations for rho, $\sqrt{6}$ from the automorphisms in iterations—though not, in our engine, in seconds—and the trade-offs of memory, parallelism and preprocessing that the literature describes. An automated search over iteration-function designs cannot beat the random-mapping floor. ECDSA and Schnorr differ in proof structure, not in verification cost; batch verification is the one real, consensus-neutral verification speed-up and it saturates at about $3\times$ for an identifiable reason. Reduced-step SHA-256 behaves as the published analyses say and our generic solvers reach only the easy regime. The bit-level identities that implementations rely on are now machine-checked, and so are the two textbook facts about the signature equations.

What the exercise shows about automated discovery is perhaps more useful than any single number. On a problem where the correct answer is "nothing new", an autonomous agent is at risk

of producing confident over-interpretations—a constant mistaken for an exponent, a reproduction mistaken for a discovery, a compression-function property mistaken for a hash property. The controls that prevented this here were cheap: operation counts next to timings, two implementations per component, a negative control per family, exact arithmetic, provenance on every number, formal statements separated from informal ones, and a classifier that required a literature match before the word "new" could be used. The one genuinely clarifying algebraic statement the pipeline produced—that the `libsecp256k1` constants are the coordinates of $\bar{\pi} - 1$—came from a bounded enumeration followed by a human-readable proof, which is the pattern we would expect future automated mathematical research on cryptographic standards to follow: automation proposes, exact verification disposes, and the literature has the last word on novelty.

References

Alamgir, Nahiyan, Saeed Nejati, and Curtis Bright, "SHA-256 Collision Attack with Programmatic SAT," in "Joint Proceedings of the 9th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 9th Satisfiability Checking and Symbolic Computation Workshop (SC-Square)," Vol. 3717 of *CEUR Workshop Proceedings* CEUR-WS.org 2024, pp. 91–110. Also arXiv:2406.20072.

Aoki, Kazumaro, Jian Guo, Krystian Matusiewicz, Yu Sasaki, and Lei Wang, "Preimages for Step-Reduced SHA-2," in "Advances in Cryptology — ASIACRYPT 2009," Vol. 5912 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 578–597.

Aranha, Diego F., Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom, "LadderLeak: Breaking ECDSA with Less than One Bit of Nonce Leakage," in "Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)" ACM 2020, pp. 225–242.

Bailey, Daniel V., Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Hsieh-Chung Chen, Chen-Mou Cheng, Gauthier van Damme, Giacomo de Meulenaer, Luis Julian Dominguez Perez, Junfeng Fan, Tim Güneysu, Frank Gurkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Ruben Niederhagen, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, Anthony Van Herrewege, and Bo-Yin Yang, "Breaking ECC2K-130," IACR Cryptology ePrint Archive, Report 2009/541 2009. Describes the planned cross-platform attack; the Certicom challenge document of November 2009 still lists ECC2K-130 as unsolved.

Bellare, Mihir and Gregory Neven, "Multi-signatures in the Plain Public-Key Model and a General Forking Lemma," in "Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)" ACM 2006, pp. 390–399.

_ , Juan A. Garay, and Tal Rabin, "Fast Batch Verification for Modular Exponentiation and Digital Signatures," in "Advances in Cryptology — EUROCRYPT '98," Vol. 1403 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 236–250.

Bernstein, Daniel J. and Tanja Lange, "Non-uniform Cracks in the Concrete: The Power of Free Precomputation," in "Advances in Cryptology — ASIACRYPT 2013," Vol. 8270 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 321–340. Full version: IACR Cryptology ePrint Archive, Report 2012/318.

_ **and** _ , "Two Grumpy Giants and a Baby," *The Open Book Series*, 2013, *1* (1), 87–111. Proceedings of ANTS X (2012).

_ **and** _ , "SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography," Web page 2014. Accessed 2026-11-01.

_ **, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang**, "High-Speed High-Security Signatures," *Journal of Cryptographic Engineering*, 2012, *2* (2), 77–89.

_ **, Susanne Engels, Tanja Lange, Ruben Niederhagen, Christof Paar, Peter Schwabe, and Ralf Zimmermann**, "Faster Elliptic-Curve Discrete Logarithms on FPGAs," IACR Cryptology ePrint Archive, Report 2016/382 2016.

_ **, Tanja Lange, and Peter Schwabe**, "On the Correct Use of the Negation Map in the Pollard Rho Method," in "Public Key Cryptography — PKC 2011," Vol. 6571 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 128–146.

Bos, Joppe W., Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, and Peter L. Montgomery, "Solving a 112-bit Prime Elliptic Curve Discrete Logarithm Problem on Game Consoles Using Sloppy Reduction," *International Journal of Applied Cryptography*, 2012, *2* (3), 212–228.

_ **, Thorsten Kleinjung, and Arjen K. Lenstra**, "On the Use of the Negation Map in the Pollard Rho Method," in "Algorithmic Number Theory — ANTS-IX," Vol. 6197 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 66–82.

Böving, Henrik, Siddharth Bhat, Luisa Cicolini, Alex Keizer, Leon Frenot, Abdalrhman Mohamed, Leo Stefanesco, Harun Khan, Josh Clune, Clark Barrett, and Tobias Grosser, "Interactive Bitvector Reasoning Using Verified Bit-Blasting," *Proceedings of the ACM on Programming Languages*, 2025, *9* (OOPSLA2), 3259–3285. The Lean 4 `bv_decide` tactic.

Breitner, Joachim and Nadia Heninger, "Biased Nonce Sense: Lattice Attacks Against Weak ECDSA Signatures in Cryptocurrencies," in "Financial Cryptography and Data Security — FC 2019," Vol. 11598 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 3–20.

Brent, Richard P., "An Improved Monte Carlo Factorization Algorithm," *BIT Numerical Mathematics*, 1980, *20* (2), 176–184.

Brown, Daniel R. L., "Generic Groups, Collision Resistance, and ECDSA," *Designs, Codes and Cryptography*, 2005, *35* (1), 119–152.

Brumley, Billy Bob and Nicola Tuveri, "Remote Timing Attacks Are Still Practical," in "Computer Security — ESORICS 2011," Vol. 6879 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 355–371.

Certicom Research, "Certicom ECC Challenge," Technical Report, Certicom Corp. 2009. Original date November 6, 1997; latest update November 10, 2009.

_ , "SEC 1: Elliptic Curve Cryptography, Version 2.0," Technical Report, Standards for Efficient Cryptography Group May 2009. May 21, 2009.

_ , "SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0," Technical Report, Standards for Efficient Cryptography Group January 2010. January 27, 2010.

Chabaud, Florent and Antoine Joux, "Differential Collisions in SHA-0," in "Advances in Cryptology — CRYPTO '98," Vol. 1462 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 56–71.

Cheon, Jung Hee, Jin Hong, and Minkyu Kim, "Accelerating Pollard's Rho Algorithm on Finite Fields," *Journal of Cryptology*, 2012, *25* (2), 195–242. Journal version of "Speeding up the Pollard rho method on prime fields", ASIACRYPT 2008, doi:10.1007/978-3-540-89255-7_29 (tag tracing).

Cohen, Henri, Atsuko Miyaji, and Takatoshi Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," in "Advances in Cryptology — ASIACRYPT '98," Vol. 1514 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 51–65.

Corrigan-Gibbs, Henry and Dmitry Kogan, "The Discrete-Logarithm Problem with Preprocessing," in "Advances in Cryptology — EUROCRYPT 2018," Vol. 10821 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 415–447. Full version: IACR Cryptology ePrint Archive, Report 2017/1113.

Damgård, Ivan Bjerre, "A Design Principle for Hash Functions," in "Advances in Cryptology — CRYPTO '89," Vol. 435 of *Lecture Notes in Computer Science*, Springer, 1990, pp. 416–427.

Davies, Alex, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli, "Advancing Mathematics by Guiding Human Intuition with AI," *Nature*, 2021, *600*, 70–74.

de Moura, Leonardo and Sebastian Ullrich, "The Lean 4 Theorem Prover and Programming Language," in "Automated Deduction — CADE 28," Vol. 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 625–635.

Diem, Claus, "On the Discrete Logarithm Problem in Elliptic Curves," *Compositio Mathematica*, 2011, *147* (1), 75–104.

Dobraunig, Christoph, Maria Eichlseder, and Florian Mendel, "Analysis of SHA-512/224 and SHA-512/256," in "Advances in Cryptology — ASIACRYPT 2015," Vol. 9453 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 612–630. Also IACR Cryptology ePrint Archive, Report 2016/374.

Duursma, Iwan, Pierrick Gaudry, and François Morain, "Speeding up the Discrete Log Computation on Curves with Automorphisms," in "Advances in Cryptology — ASIACRYPT '99," Vol. 1716 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 103–121.

Fawzi, Alhussein, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli, "Discovering Faster Matrix Multiplication Algorithms with Reinforcement Learning," *Nature*, 2022, *610*, 47–53.

Fersch, Manuel, Eike Kiltz, and Bertram Poettering, "On the Provable Security of (EC)DSA Signatures," in "Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 2016)" ACM 2016, pp. 1651–1662.

_ , _ , and _ , "On the One-Per-Message Unforgeability of (EC)DSA and Its Variants," in "Theory of Cryptography — TCC 2017," Vol. 10678 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 519–534.

Flajolet, Philippe and Andrew M. Odlyzko, "Random Mapping Statistics," in "Advances in Cryptology — EUROCRYPT '89," Vol. 434 of *Lecture Notes in Computer Science*, Springer, 1990, pp. 329–354.

Frey, Gerhard and Hans-Georg Rück, "A Remark Concerning *m*-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves," *Mathematics of Computation*, 1994, *62* (206), 865–874.

Galbraith, Steven D., *Mathematics of Public Key Cryptography*, Cambridge: Cambridge University Press, 2012.

_ and Pierrick Gaudry, "Recent Progress on the Elliptic Curve Discrete Logarithm Problem," *Designs, Codes and Cryptography*, 2016, *78* (1), 51–72.

_ and Raminder S. Ruprai, "Using Equivalence Classes to Accelerate Solving the Discrete Logarithm Problem in a Short Interval," in "Public Key Cryptography — PKC 2010," Vol. 6056 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 368–383.

_ , **Xibin Lin, and Michael Scott**, "Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves," *Journal of Cryptology*, 2011, *24* (3), 446–469.

Gallant, Robert P., Robert J. Lambert, and Scott A. Vanstone, "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms," in "Advances in Cryptology — CRYPTO 2001," Vol. 2139 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 190–200.

Gallant, Robert, Robert Lambert, and Scott Vanstone, "Improving the Parallelized Pollard Lambda Search on Anomalous Binary Curves," *Mathematics of Computation*, 2000, *69* (232), 1699–1705.

Gaudry, Pierrick, "Index Calculus for Abelian Varieties of Small Dimension and the Elliptic Curve Discrete Logarithm Problem," *Journal of Symbolic Computation*, 2009, *44* (12), 1690–1702.

Gilbert, Henri and Helena Handschuh, "Security Analysis of SHA-256 and Sisters," in "Selected Areas in Cryptography — SAC 2003," Vol. 3006 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 175–193.

Groth, Jens and Victor Shoup, "On the Security of ECDSA with Additive Key Derivation and Presignatures," in "Advances in Cryptology — EUROCRYPT 2022," Vol. 13276 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 365–396.

Hankerson, Darrel, Alfred Menezes, and Scott Vanstone, *Guide to Elliptic Curve Cryptography*, New York: Springer, 2004.

Harris, Bernard, "Probability Distributions Related to Random Mappings," *The Annals of Mathematical Statistics*, 1960, *31* (4), 1045–1062.

Howgrave-Graham, Nick A. and Nigel P. Smart, "Lattice Attacks on Digital Signature Schemes," *Designs, Codes and Cryptography*, 2001, *23* (3), 283–290.

Indesteege, Sebastiaan, Florian Mendel, Bart Preneel, and Christian Rechberger, "Collisions and Other Non-random Properties for Step-Reduced SHA-256," in "Selected Areas in Cryptography — SAC 2008," Vol. 5381 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 276–293. Also IACR Cryptology ePrint Archive, Report 2008/131.

Jančar, Ján, Vladimír Sedláček, Petr Švenda, and Marek Sýs, "Minerva: The Curse of ECDSA Nonces," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020, *2020* (4), 281–308.

Khovratovich, Dmitry, Christian Rechberger, and Alexandra Savelieva, "Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family," in "Fast Software Encryption — FSE 2012," Vol. 7549 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 244–263. Also IACR Cryptology ePrint Archive, Report 2011/286.

Kocher, Paul C., "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in "Advances in Cryptology — CRYPTO '96," Vol. 1109 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 104–113.

Lee, Hyung Tae, Jung Hee Cheon, and Jin Hong, "Accelerating ID-based Encryption Based on Trapdoor DL Using Pre-computation," IACR Cryptology ePrint Archive, Report 2011/187 2011. Discrete-logarithm computation with precomputation (preprocessing) in a fixed group.

Leurent, Gaëtan and Thomas Peyrin, "SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust," in "29th USENIX Security Symposium (USENIX Security 20)" USENIX Association 2020. Also IACR Cryptology ePrint Archive, Report 2020/014.

Li, Yingxin, Fukang Liu, and Gaoli Wang, "New Records in Collision Attacks on SHA-2," in "Advances in Cryptology — EUROCRYPT 2024," Vol. 14654 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 158–186. Also IACR Cryptology ePrint Archive, Report 2024/349; extended journal version in Journal of Cryptology 39 (2026), article 27, doi:10.1007/s00145-026-09582-9.

_ , _ , _ , Xiaoyang Dong, and Siwei Sun, "The First Practical Collision for 31-Step SHA-256," in "Advances in Cryptology — ASIACRYPT 2024" Lecture Notes in Computer Science, Springer, 2024, pp. 237–266.

Lu, Chris, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha, "The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery," arXiv:2408.06292 2024.

Maxwell, Gregory, Andrew Poelstra, Yannick Seurin, and Pieter Wuille, "Simple Schnorr Multi-Signatures with Applications to Bitcoin," *Designs, Codes and Cryptography*, 2019, *87* (9), 2139–2164.

Mendel, Florian, Tomislav Nad, and Martin Schläffer, "Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions," in "Advances in Cryptology — ASIACRYPT 2011," Vol. 7073 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 288–307.

_ , _ , **and** _ , "Improving Local Collisions: New Attacks on Reduced SHA-256," in "Advances in Cryptology — EUROCRYPT 2013," Vol. 7881 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 262–278. Also IACR Cryptology ePrint Archive, Report 2015/350.

Menezes, Alfred J., Tatsuaki Okamoto, and Scott A. Vanstone, "Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field," *IEEE Transactions on Information Theory*, 1993, *39* (5), 1639–1646.

Merkle, Ralph C., "One Way Hash Functions and DES," in "Advances in Cryptology — CRYPTO '89," Vol. 435 of *Lecture Notes in Computer Science*, Springer, 1990, pp. 428–446.

Mironov, Ilya and Lintao Zhang, "Applications of SAT Solvers to Cryptanalysis of Hash Functions," in "Theory and Applications of Satisfiability Testing — SAT 2006," Vol. 4121 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 102–115.

Möller, Bodo, "Algorithms for Multi-exponentiation," in "Selected Areas in Cryptography — SAC 2001," Vol. 2259 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 165–180.

Montgomery, Peter L., "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of Computation*, 1987, *48* (177), 243–264.

Naccache, David, David M'Raïhi, Serge Vaudenay, and Dan Raphaeli, "Can D.S.A. Be Improved? Complexity Trade-Offs with the Digital Signature Standard," in "Advances in Cryptology — EUROCRYPT '94," Vol. 950 of *Lecture Notes in Computer Science*, Springer, 1995, pp. 77–85.

National Institute of Standards and Technology, "Secure Hash Standard (SHS)," Federal Information Processing Standards Publication FIPS 180-4, U.S. Department of Commerce August 2015.

_ , "Digital Signature Standard (DSS)," Federal Information Processing Standards Publication FIPS 186-5, U.S. Department of Commerce February 2023.

Nechaev, Vassiliy I., "Complexity of a Determinate Algorithm for the Discrete Logarithm," *Mathematical Notes*, 1994, *55* (2), 165–172.

Nejati, Saeed, Jia Hui Liang, Catherine Gebotys, Krzysztof Czarnecki, and Vijay Ganesh, "Adaptive Restart and CEGAR-Based Solver for Inverting Cryptographic Hash Functions," in "Verified Software: Theories, Tools, and Experiments — VSTTE 2017," Vol. 10712 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 120–131.

Nick, Jonas, Tim Ruffing, and Yannick Seurin, "MuSig2: Simple Two-Round Schnorr Multi-signatures," in "Advances in Cryptology — CRYPTO 2021," Vol. 12825 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 189–221.

Nikolić, Ivica and Alex Biryukov, "Collisions for Step-Reduced SHA-256," in "Fast Software Encryption — FSE 2008," Vol. 5086 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 1–15.

Novikov, Alexander, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii et al., "AlphaEvolve: A Coding Agent for Scientific and Algorithmic Discovery," arXiv:2506.13131 2025.

Paillier, Pascal and Damien Vergnaud, "Discrete-Log-Based Signatures May Not Be Equivalent to Discrete Log," in "Advances in Cryptology — ASIACRYPT 2005," Vol. 3788 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 1–20.

Petit, Christophe and Jean-Jacques Quisquater, "On Polynomial Systems Arising from a Weil Descent," in "Advances in Cryptology — ASIACRYPT 2012," Vol. 7658 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 451–466.

Pippenger, Nicholas, "On the Evaluation of Powers and Monomials," *SIAM Journal on Computing*, 1980, *9* (2), 230–250.

Pohlig, Stephen C. and Martin E. Hellman, "An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance," *IEEE Transactions on Information Theory*, 1978, *24* (1), 106–110.

Pointcheval, David and Jacques Stern, "Security Arguments for Digital Signatures and Blind Signatures," *Journal of Cryptology*, 2000, *13* (3), 361–396.

Pollard, John M., "Monte Carlo Methods for Index Computation (mod p)," *Mathematics of Computation*, 1978, *32* (143), 918–924.

_ , "Kangaroos, Monopoly and Discrete Logarithms," *Journal of Cryptology*, 2000, *13* (4), 437–447.

Pornin, Thomas, "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)," RFC 6979, Internet Engineering Task Force 2013.

Roetteler, Martin, Michael Naehrig, Krysta M. Svore, and Kristin Lauter, "Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms," in "Advances in Cryptology — ASIACRYPT 2017," Vol. 10625 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 241–270.

Romera-Paredes, Bernardino, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi, "Mathematical Discoveries from Program Search with Large Language Models," *Nature*, 2024, *625*, 468–475.

Sanadhya, Somitra Kumar and Palash Sarkar, "New Collision Attacks against Up to 24-Step SHA-2," in "Progress in Cryptology — INDOCRYPT 2008," Vol. 5365 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 91–103. Also IACR Cryptology ePrint Archive, Report 2008/270.

Satoh, Takakazu and Kiyomichi Araki, "Fermat Quotients and the Polynomial Time Discrete Log Algorithm for Anomalous Elliptic Curves," *Commentarii Mathematici Universitatis Sancti Pauli*, 1998, *47* (1), 81–92. Errata: same journal 48 (1999), no. 2, 211–213. No DOI; bibliographic data confirmed through citing sources.

Schnorr, Claus-Peter, "Efficient Signature Generation by Smart Cards," *Journal of Cryptology*, 1991, *4* (3), 161–174.

Schoof, René, "Elliptic Curves over Finite Fields and the Computation of Square Roots mod p," *Mathematics of Computation*, 1985, *44* (170), 483–494.

_ , "Counting Points on Elliptic Curves over Finite Fields," *Journal de Théorie des Nombres de Bordeaux*, 1995, *7* (1), 219–254.

Semaev, Igor, "Summation Polynomials and the Discrete Logarithm Problem on Elliptic Curves," IACR Cryptology ePrint Archive, Report 2004/031 2004.

Semaev, Igor A., "Evaluation of Discrete Logarithms in a Group of p-Torsion Points of an Elliptic Curve in Characteristic p," *Mathematics of Computation*, 1998, *67* (221), 353–356.

Seurin, Yannick, "On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model," in "Advances in Cryptology — EUROCRYPT 2012," Vol. 7237 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 554–571.

Shanks, Daniel, "Class Number, a Theory of Factorization, and Genera," in "1969 Number Theory Institute," Vol. 20 of *Proceedings of Symposia in Pure Mathematics*, American Mathematical Society, 1971, pp. 415–440.

Shor, Peter W., "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, 1997, *26* (5), 1484–1509.

Shoup, Victor, "Lower Bounds for Discrete Logarithms and Related Problems," in "Advances in Cryptology — EUROCRYPT '97," Vol. 1233 of *Lecture Notes in Computer Science*, Springer, 1997, pp. 256–266.

Sica, Francesco, Mathieu Ciet, and Jean-Jacques Quisquater, "Analysis of the Gallant-Lambert-Vanstone Method Based on Efficient Endomorphisms: Elliptic and Hyperelliptic Curves," in "Selected Areas in Cryptography — SAC 2002," Vol. 2595 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 21–36.

Silverman, Joseph H., *The Arithmetic of Elliptic Curves*, 2nd ed., Vol. 106 of *Graduate Texts in Mathematics*, New York: Springer, 2009.

Smart, Nigel P., "The Discrete Logarithm Problem on Elliptic Curves of Trace One," *Journal of Cryptology*, 1999, *12* (3), 193–196.

Solinas, Jerome A., "Generalized Mersenne Numbers," Technical Report CORR 99-39, Centre for Applied Cryptographic Research, University of Waterloo 1999.

Stevens, Marc, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov, "The First Collision for Full SHA-1," in "Advances in Cryptology — CRYPTO 2017," Vol. 10401 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 570–596.

Straus, Ernst G., "Addition Chains of Vectors (Problem 5125)," *The American Mathematical Monthly*, 1964, *71* (7), 806–808. Solution to problem 5125 proposed by R. Bellman.

Teske, Edlyn, "Speeding up Pollard's Rho Method for Computing Discrete Logarithms," in "Algorithmic Number Theory — ANTS-III," Vol. 1423 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 541–554.

_ , "On Random Walks for Pollard's Rho Method," *Mathematics of Computation*, 2001, *70* (234), 809–825.

Trinh, Trieu H., Yuhuai Wu, Quoc V. Le, He He, and Thang Luong, "Solving Olympiad Geometry without Human Demonstrations," *Nature*, 2024, *625*, 476–482.

van Oorschot, Paul C. and Michael J. Wiener, "Parallel Collision Search with Cryptanalytic Applications," *Journal of Cryptology*, 1999, *12* (1), 1–28.

Vaudenay, Serge, "The Security of DSA and ECDSA," in "Public Key Cryptography — PKC 2003," Vol. 2567 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 309–323.

Washington, Lawrence C., *Elliptic Curves: Number Theory and Cryptography*, 2nd ed., Boca Raton: Chapman and Hall/CRC, 2008.

Wenger, Erich and Paul Wolfger, "Solving the Discrete Logarithm of a 113-bit Koblitz Curve with an FPGA Cluster," in "Selected Areas in Cryptography — SAC 2014," Vol. 8781 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 363–379. Also IACR Cryptology ePrint Archive, Report 2014/368.

_ and _, "Harder, Better, Faster, Stronger: Elliptic Curve Discrete Logarithm Computations on FPGAs," *Journal of Cryptographic Engineering*, 2016, *6* (4), 287–297. Also IACR Cryptology ePrint Archive, Report 2015/143.

Wiener, Michael J. and Robert J. Zuccherato, "Faster Attacks on Elliptic Curve Cryptosystems," in "Selected Areas in Cryptography — SAC '98," Vol. 1556 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 190–200.

Wuille, Pieter, Gregory Maxwell, Jonas Nick, Tim Ruffing et al., "libsecp256k1: Optimized C Library for ECDSA and BIP-340 Signatures on Curve secp256k1," Software, Bitcoin Core project, GitHub 2024. The GLV endomorphism $\lambda \cdot (x,y) = (\beta x, y)$ and the scalar split are implemented in `src/scalar_impl.h`; accessed 2026-11-01.

_, Jonas Nick, and Anthony Towns, "BIP 341: Taproot: SegWit Version 1 Spending Rules," Bitcoin Improvement Proposal 341 2020. Status: Deployed; consensus layer (soft fork); requires BIP 340.

_, _, and Tim Ruffing, "BIP 340: Schnorr Signatures for secp256k1," Bitcoin Improvement Proposal 340 2020. Status: Deployed; type: Specification; assigned 2020-01-19.

Zhang, Zhuolong, Muzhou Li, Lei Gao, and Meiqin Wang, "Collision Attacks on SHA-256 up to 37 Steps with Improved Trail Search," IACR Cryptology ePrint Archive, Report 2026/232 2026.

Appendices

A. Reproducibility statement

The repository `wp34_uqo` (UQO working-paper series, `mathematiques/beyond-square-root-barrier`) contains every script, result file and figure source. All experiments are re-runnable from the repository root:

```
uv venv --python 3.12 .venv && uv pip install -r requirements.txt # Python environment
.venv/bin/python -m pytest -q tests/ # cross-implementation gates
.venv/bin/python experiments/baselines/secp256k1_facts.py # Table 2 (needs PARI/GP ‘gp‘)
.venv/bin/python experiments/baselines/scalar_mult_opcount.py # GLV operation counts
.venv/bin/python theory/elliptic_curves/symbolic_checks.py # SymPy checks S1-S6
.venv/bin/python theory/elliptic_curves/orbit_quotient_counterexample.py # H05
.venv/bin/python experiments/synthetic_ecdlp/index_calculus_probe.py # H06
.venv/bin/python experiments/synthetic_ecdlp/preprocessing_python.py # H12
.venv/bin/python experiments/automated_discovery/identity_search.py # H02, H07
cargo build --release --manifest-path experiments/synthetic_ecdlp/rho_engine/Cargo.toml
.venv/bin/python experiments/synthetic_ecdlp/run_scaling.py --sizes 20,24,28,32,36,40 #
    H08-H11 (see README for the cluster split)
cargo run --release --manifest-path experiments/signature_benchmarks/sigbench/Cargo.toml
    -- ... # H13-H14 (run_bench.py)
.venv/bin/python experiments/reduced_round_sha256/diffusion.py / bias_test.py /
    preimage_sat.py / collision_sat.py / local_collision.py
cd formal && lake build # H21, H22, H25 (Lean 4.35)
.venv/bin/python experiments/automated_discovery/registry.py # discovery log + Table of
    hypotheses
.venv/bin/python scripts/make_tables.py && cd paper && make # tables/numbers.tex, main.
    pdf
```

Every result JSON carries a provenance block (host name, CPU, cores, RAM, OS, Python and library versions, seed, command line, UTC timestamp, elapsed time). Random seeds are fixed in the scripts (default 34) and recorded. Raw solver outputs and the Rust batch outputs are kept next to the aggregated files.

B. Computational environment

C. Full secp256k1 constants

```
p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F
n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
```

Table A1. Hardware and software used (measured, from the provenance blocks).

Role	Machine	CPU	Cores	RAM	Used for
development, reference checks, figures	laptop	Apple M5 Max	18	48 GB	tests, SymPy/PARI, Python ex
benchmarking	M3U96a (Mac Studio)	Apple M3 Ultra	28	96 GB	signature benchmarks (second
benchmarking	M3U96b (Mac Studio)	Apple M3 Ultra	28	96 GB	SAT/SMT grids, ECDLP scali

Software: macOS 27.0; Python 3.12 (uv), SymPy 1.14, NumPy 2.5, python-sat 1.9 (CaDiCaL 1.9.5 bindings), Z3 4.15 (`z3-solver`), Rust 1.95–1.96 (`k256` 0.14, `secp256k1` 0.33 / `libsecp256k1`); Lean 4.35.0 (`bv_decide` with bundled CaDiCaL); TeX Live 2026 used.

```
G.x = 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798
G.y = 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8
beta = 7AE96A2B 657C0710 6E64479E AC3434E9 9CF04975 12F58995 C1396C28 719501EE
lambda = 5363AD4C C05C30E0 A5261C02 8812645A 122E22EA 20816678 DF02967C 1B23BD72
a1 = 3086D221 A7D46BCD E86C90E4 9284EB15 (= (t - f)/2 - 1)
b1 = -E4437ED6 010E8828 6F547FA9 0ABFE4C3 (= -f)
a2 = 114CA50F 7A8E2F3F 657C1108 D9D44CFD8 (= a1 - b1 = (t + f)/2 - 1)
b2 = a1
t = 432420386565659656852420866390673177327 (trace of Frobenius)
f = 303414439467246543595250775667605759171 (4p = t^2 + 3 f^2)
```