



CRYPTOGRAPHY | OCTOBER 17, 2026

The Missing Recipe Behind Bitcoin's Starting Point

SEC GENERATOR PROVENANCE

Brian Mingus

A numerical and archival investigation of how SEC elliptic-curve generators were selected. The calculations can be reproduced; the original seed and program remain unknown.

Collection edition. The complete 12-page white paper follows.

AI-generated conceptual artwork; separate from scientific figures.

brianmingus.com

The Missing Recipe Behind Bitcoin's Starting Point

Brian Mingus

SEC generator provenance: numerical reconstruction, archival evidence, and exact exclusions

Abstract

Bitcoin's elliptic curve has a public starting point whose coordinates can be checked exactly, but the historical procedure that selected it remains unexplained in the sources examined here. We investigate that procedure by following numerical relationships across related Standards for Efficient Cryptography (SEC) curves, comparing early parameter documents, and testing candidate constructions against original software and published controls. The familiar prime-curve fingerprint is prior work. Our analysis builds on it with explicit exclusions for shared-input models, a reproducible scan-and-double construction for six binary curves, and a source trail reaching the 1999 drafts. Two larger starting integers remain unexplained. A resemblance to a Certicom challenge coordinate fails exact field-trace tests; several direct random-number constructions also fail within their stated models. Period documentation identifies plausible random-number APIs and a persistent seed-mixing mechanism, but does not connect either to the SEC generation program. The result is an evidence-backed account of what can be reconstructed, what has been excluded, and which missing records could settle the question. We have not recovered the original seed or program.

Keywords: *secp256k1; Certicom; SEC 2; generator provenance; reproducible research*

1 Why this particular point?

Bitcoin uses the elliptic curve secp256k1. Its parameters specify a public point, conventionally called G , from which public keys are derived by scalar multiplication. A private scalar d gives a public point dG . The coordinates of G are therefore part of the common starting material used by everyone, rather than a secret held by a particular participant. The SEC standard publishes those coordinates, along with the field, curve equation, subgroup order, and cofactor [1].

The coordinates look like large arbitrary numbers. That leaves a historical question: what did the person or program selecting them actually do? The standard is sufficient to implement the curve, but implementing the curve does not recover the process that selected its generator. A complete answer would identify an input and a procedure that produce the published point without inserting that point into the procedure in advance.

This investigation began after Anthropic's October 2026 report reconstructed the origins of several NIST curve seeds and base points using historical random-number generators [2]. That work supplies useful methods and positive controls. It does not, by itself, supply the missing construction for Bitcoin's generator. NIST P-256 and secp256k1 are different curves, and a curve-coefficient seed is a different object from a generator-selection seed.

There is already a substantial numerical clue. When the generators of four related SEC prime curves are halved, their x coordinates share a long hexadecimal interior. The observation predates this investigation: it appears in public discussion, John Zheng's reproducible script, and Nadia Heninger's ECC 2024 slides [3–5]. We treat it as prior work. We also treat earlier suggestions about hashing and reused cofactor-clearing code as hypotheses with a history, rather than discoveries made here.

Here we follow that clue through the related binary curves, which provide additional examples of how a point could have been selected. The distinction between the two families matters. Bitcoin's curve is defined over a prime field. The six binary curves studied below use fields of characteristic two. A shared implementation would help explain their relationship, but no recovered record establishes that implementation. The binary results are evidence about those curves in their own right.

2 What would count as recovering the recipe?

There are several levels of evidence in this problem. A numerical identity can be established with exact arithmetic. A candidate recipe can reproduce a published point. A source document can establish that an algorithm existed at a particular time. These facts can support each other, but they answer different questions.

For example, computing a point P such that $2P = G$ proves an arithmetic relationship. Starting a program at $x(P)$ and obtaining G proves that the program is compatible with the result. It does not explain where $x(P)$ came from. Likewise, finding a 1998 API named for a random-number generator establishes that a product exposed that interface; it does not establish that the standards authors called it.

We use the term *reconstruction* for a reproducible procedure consistent with the published constants. We reserve *historical recovery* for a construction supported by an independently specified input or generation record. A compact procedure that predicts another constant left out of its development would also provide much stronger evidence than a formula fitted to every observed value.

Negative results require the same discipline. If all initial states are impossible under a stated output format, that format has been excluded. If a finite list of public seeds produces no match, only that list and the tested transformations have been checked. Neither result licenses a claim that an entire library, hash function, or historical software ecosystem has been eliminated. Throughout this paper, bounds and assumptions are part of the result.

3 The prime-curve fingerprint

Let n be the published odd order of the subgroup generated by G . The point

$$H = \left(\frac{n+1}{2} \right) G \quad (1)$$

is its unique half within that subgroup, since $2H = (n+1)G = G$. Exact arithmetic reproduces the four coordinates in Table 1. Every G and H lies on its stated curve, nG is the point at infinity, and $2H = G$. All four H points have even integer y coordinates.

Table 1: Reproduced prime-curve halves. Coordinates are hexadecimal. The shared pattern is prior work; the table is independently recomputed from the public parameters.

Curve	$x(H)$	Bits
secp160k1	48ce563f89a0ed9414f5aa28ad0d96d6795f9c62	159
secp192k1	554123b78ce563f89a0ed9414f5aa28ad0d96d6795f9c66	187
secp224k1	3b78ce563f89a0ed9414f5aa28ad0d96d6795f9c63	166
secp256k1	3b78ce563f89a0ed9414f5aa28ad0d96d6795f9c63	166

The common interior has 152 bits:

8ce563f89a0ed9414f5aa28ad0d96d6795f9c6.

The two largest curves have exactly the same half-generator x coordinate, despite having different fields. The smaller curves retain the interior but have different surrounding digits. Those differences are useful: a proposed construction must account for them as well as the repeated part.

3.1 A common integer reduced into each field

Consider a simple construction. Choose one nonnegative integer T , set $x = T \bmod p$ on each curve, increment x by one until a point exists, choose a compatible root, and double the point. This model permits no extra rejection test or curve-specific transformation.

The previous usable coordinate constrains the starting value. If x_i is the observed coordinate and $v_i < x_i$ is the immediately preceding coordinate that lifts to a point, the start must lie in

$$I_i = \{v_i + 1, \dots, x_i\}. \quad (2)$$

A start at or below v_i would stop too early. The verified possible increments $x_i - (T \bmod p_i)$ are respectively $\{0\}$, $\{0, 1, 2, 3, 4\}$, $\{0\}$, and $\{0, 1\}$ for the 160-, 192-, 224-, and 256-bit curves. There are only ten combinations to check.

For each combination, the Chinese remainder theorem gives the least nonnegative integer having those four residues. The smallest of the ten solutions has 829 bits. Consequently, no integer of at most 828 bits fits this particular reduce-and-increment construction. A single SHA-1, SHA-256, or SHA-512 digest interpreted directly as T cannot fit it. If the reduction is instead performed modulo the subgroup orders and the residues are used as coordinates, the corresponding minimum is 827 bits.

These are bounds on an intermediate integer. They are not bounds on seed entropy, and they do not measure the difficulty of recovering the historical recipe. A short seed expanded by another procedure is a different hypothesis. The order-modulus calculation also says nothing about generating G by scalar multiplication of another point. The supplied certificates contain all ten residue combinations and their least solutions; two different CRT implementations agreed.

3.2 Truncation and the published challenge expansion

A shared integer followed by low-bit truncation fails more directly. The 224-bit curve forces one 166-bit starting value. Truncating that value to 192 bits leaves it unchanged, whereas every allowable 192-bit start has 187 bits. Thus no integer, however large, works under this nested low-bit model. Clearing the highest permitted bit does not repair the conflict.

Certicom's published challenge recipe provides a second nested construction [6]. Write $A = \text{SHA1}(s)$ and $B = \text{SHA1}(s + 1)$ for a common 160-bit seed, with the documented seed encoding and wraparound. The relevant prime-field expansions are

$$X_{192} = (A \bmod 2^{31}) 2^{160} + B, \quad (3)$$

$$X_{224} = (A \bmod 2^{63}) 2^{160} + B. \quad (4)$$

They necessarily satisfy $X_{192} = X_{224} \bmod 2^{191}$. None of the allowable starting pairs satisfies that identity. This excludes the common-seed expansion *when followed by the separately assumed first-valid-coordinate increment rule*. The published challenge procedure retries seed selection; we do not attribute the added coordinate-increment rule to it.

The expansion implementation has a positive control: it reproduces all 48 published point x coordinates available in the challenge table [7]. For binary challenges these are the coordinates before cofactor clearing. This control checks expansion and encoding, not the subsequent root selection or complete point-generation procedure. It also exposes a documentary inconsistency: the inspected prime algorithms name the curve seed inside the point expansion, while using the point seed reproduces the published table. We make no claim to have first noticed that inconsistency.

4 Six binary curves and two unexplained integers

The binary curves have cofactor two. Their equations take the form

$$y^2 + xy = x^3 + ax^2 + b \quad \text{over } \mathbb{F}_{2^m}. \quad (5)$$

Field addition here is XOR. The integers printed as coordinates encode polynomial-basis bitstrings; incrementing that integer is not the same operation as adding the field element one.

There are two doubling preimages of each published generator. Equation 1 gives the preimage H in the odd-order subgroup. The other is $H + T_2$, where $T_2 = (0, \sqrt{b})$ is the nonidentity point of order two. Both double to G . We compute both and select the one with the smaller integer x coordinate, calling it P . Only the selected P for sect113r1 belongs to the odd-order subgroup; the other five satisfy $nP = T_2$. Calling all six selected points the subgroup half would therefore be incorrect.

The selected points admit a common procedure within each field-size pair. Begin at an integer S_m , scan upward one integer at a time, and accept the first coordinate that lifts to a point whose double is nonzero. Choose a compatible y , then double. Define the two larger starts as

$$S_{131} = 0x10134db223421eb623422888a7879682, \quad (6)$$

$$S_{193} = 0x23daf8a23206f9c4f299d7b21a9c369137f2c84ae1aa04. \quad (7)$$

They contain 125 and 182 significant bits, respectively. Table 2 shows the resulting coordinates.

Table 2: A compatible scan-and-double construction for the binary pairs. Uniqueness refers only to the stated shared forward-scan model.

Pair	Common positive start	First accepted x , r1	First accepted x , r2
113 bits	1 or 2	3	2
131 bits	S_{131} only	S_{131}	$S_{131} + 2$
193 bits	S_{193} only	S_{193}	S_{193}

For all six curves, choosing the numerically smaller root reproduces the published G . Using the unadjusted half-trace solution also reproduces it. These observations do not distinguish which rule, if either, was used historically. If starts at zero are permitted and the order-two point is rejected because it doubles to infinity, zero also works for the 113-bit pair. The table states the positive-start convention used by the verifier.

The checks go beyond comparing the final coordinates. They verify curve membership, $nG = \mathcal{O}$, both doubling preimages, the order-two point, each failed coordinate between the start and acceptance, and the previous acceptable coordinate. A separate doubling formula,

$$x(2P) = x(P)^2 + \frac{b}{x(P)^2}, \quad (8)$$

agrees with the full point calculation. It shares the same field-arithmetic implementation, so it is an additional algebraic check rather than a completely independent implementation.

This gives an exact reconstruction from the two starting integers to the published generators. The step before those integers is still missing. The standard's published curve seeds do not automatically fill that gap: a seed used to define a coefficient cannot be relabeled as a seed for the generator without evidence.

5 How far back the constants can be followed

The complete SEC 2 Version 0.6 working draft, dated October 1999, contains the same ten relevant curves as Version 1.0: the four prime curves and the six binary curves [1, 8]. Two extraction and

parsing paths agree on 82 numerical fields, including the applicable field parameters, coefficients, point encodings, orders, cofactors, and binary curve seeds. The same point arithmetic therefore reconstructs the same intermediate values from that draft.

Cached numerical pages from GEC 1 Version 0.5, dated September 1999, also agree on these curves [9]. This is weaker evidence than the complete October PDF: the September file itself was not recovered. SECG meeting minutes from 16 September 1999 independently record the transition from GEC 1 to SEC 2 [10]. A printed date, PDF metadata, and an archive capture date each describe a different event. We use the printed dates to identify the documents, without treating a later capture as proof of an exact original publication day.

There is an earlier control for one curve. The 11 February 1999 WTLS specification assigns curve 4 the same checked parameters as sect113r1; its smaller doubling preimage has $x = 3$ [11]. The April 1998 WTLS table lists only curves 1–3 [12]. This establishes a change in that document series. It does not date the larger binary starts, or Bitcoin’s generator, to February 1999.

The documentary record narrows the search to software and records that could have produced the constants by the 1999 drafts. It does not identify the author of those constants. Names appearing as document contacts or standards contributors are insufficient for that attribution.

6 A close numerical resemblance that does not survive an exact test

Certicom’s ECC2-131 challenge supplies a published point seed:

b55555555554d696e67687561517586aa17bf3f4.

SHA-1 of its 20 decoded bytes is

33255631db221deb61dc208ada4d88885a787969.

Taking the low 131 bits reproduces the challenge coordinate

$$U_x = 0x1db221deb61dc208ada4d88885a787969. \quad (9)$$

Several fragments resemble S_{131} . That made the value worth investigating. It does not make the values equal, and selecting a resemblance after inspecting many numbers does not supply a probability that they share an origin.

One possible explanation was a change of field representation. The challenge and SEC curves use different irreducible polynomials. If the same field element had merely been printed in another basis, an isomorphism could change its bitstring while preserving its arithmetic meaning. Absolute field trace gives a short test:

$$\text{Tr}(u) = \sum_{j=0}^{m-1} u^{2^j} \in \mathbb{F}_2. \quad (10)$$

The trace of U_x in the published challenge field is one. The traces of both S_{131} and $S_{131} + 2$ in the SEC field are zero. Field isomorphisms preserve trace. Consequently, no field isomorphism maps that exact challenge coordinate to either selected SEC precursor.

Interpreting the same bits as normal-basis coordinates fails as well. In a normal basis $\beta, \beta^2, \dots, \beta^{2^{m-1}}$, the trace of β must be one; otherwise the sum of all basis vectors would give a nontrivial linear dependence. By linearity, the trace of a normal-basis coordinate string is its Hamming parity. The published U_x has 61 set bits and hence odd parity. Permuting, reversing, or rotating those bits does not change it. It cannot represent either trace-zero target in any normal basis.

Two separately written squaring implementations confirmed the trace calculations. These proofs concern the exact published value. They do not exclude edited bits, another hash input, or an additional arithmetic transformation. The resemblance remains an unexplained observation, with its simplest representation-only explanations eliminated.

7 Historical software: what the exclusions actually cover

We examined historical arithmetic libraries because they can leave recognizable structure in the large integers they produce. The difficult part is matching the historical API, word width, output conversion, and handling of rejected draws. A modern implementation with the same function name can behave differently.

FreeLIP illustrates the distinction. A LiDIA 1.1 archive contains FreeLIP V1.0 with a README dated 1 May 1995 [13]. The recovered implementation supports 26-, 30-, and 62-bit arithmetic configurations. For the direct calls `zrandomb(2131)` and `zrandomb(2193)`, exact recurrence and packing constraints exclude the two respective shared starts for every initial state in those configurations, including the modeled internal rejected draws. The source-order cases and an additional reversed-order hypothesis are recorded separately.

Original C source was compiled during the investigation to check the arithmetic model. These were modern builds of historical code, not emulated 1995 machines. The 26-bit control used the source's generic arithmetic branch; a successful native control for its specialized 32-bit multiplication branch was not obtained. The portable replay checks the supplied certificates and recorded control outputs, rather than recompiling that library.

Other results have different strengths. Declared direct-output layouts for `lrand48`, historical PARI recurrence/packing models, and specified `mrnd48` mappings yield all-state exclusions. Default-stream checks for MIRACL, NTL, and GMP are finite tests. The tested NTL source is dated February 2000, so its identity with the implementation available before the September 1999 draft remains unverified. Maple-style assembly checks concern the modeled recurrence and assembly documented in the inspected sources, not every implementation of historical Maple [14–16].

Some apparently minor differences changed the models substantially. The inspected 1997 GNU implementation of `mrnd48` and the May 1999 implementation do not implement the same output mapping. Neither was silently replaced with ordinary top-32-bit extraction. Likewise, GMP's inspected generic branch combines two `random()` outputs per limb, and the evaluation order must be accounted for. A failed comparison against the wrong mapping would tell us very little.

The larger collection of checks also tests published seeds under ordinary hashes, byte and word layouts, polynomial reduction, and selected normal-basis conversions. Those tests do not identify a recipe. Their numerical counts are useful for reproducing the experiment, but many comparisons overlap or reuse the same stream. We do not add them together and present the sum as independent evidence against an unknown input. Appendix A records representative boundaries.

8 The missing initialization step

A Certicom Security Builder SDK 2.0 page captured in December 1998 lists `sb_rngSeedFIPS186`, `sb_rngFIPS186Private`, and `sb_rngFIPS186Session`, alongside ordinary SHA-1 functions and an elliptic-curve Blum–Blum–Shub option [17]. The page provides API names, not implementation bodies or the parameter-generation program.

There is a more concrete initialization mechanism in Certicom's VeriSmart card security policy, retained by NIST. Printed page 144 describes a session seed XORed with a persistent initial seed before initializing the FIPS 186 generator. The persistent seed is subsequently mixed with

the generator state at session end. Printed page 162 cites the *Security Builder Programmer's Reference Manual*, Release 1.1, document SBPR-9706-0001, dated 1997 [18].

This identifies a real seed-mixing mechanism in a Certicom product. Its use in the SEC parameter-generation software is unproven. We have neither the relevant persistent value nor a record that the standards-generation program used this API. An arbitrary XOR mask chosen to force one desired output would merely move the unexplained constant into the mask.

The distinction between ordinary SHA-1 and the FIPS 186 construction also matters. The inspected FIPS 186-1 procedure uses a SHA-1 compression operation on a seed block padded with zeroes, with a specified initial state. Ordinary SHA-1 adds its own message padding and length encoding [19, 20]. A search using only an ordinary hash function would not test that construction.

We reconstructed the compression-based procedure and checked the existing corpus of 102 distinct public seeds. The experiment fixes the seed width at 160 bits, optional input at zero, two stated initial-state variants, the first one or two output blocks, and eight named coordinate conversions per degree. Its 3,264 candidate equalities give no exact match to either larger shared start. Both FIPS 186-1 Appendix 5 examples and the complete 160-byte RFC 4186 example pass as controls [21]. The coordinate adaptations and two-block session variant are hypotheses, rather than recovered Security Builder behavior. Unknown seed mixing, later calls, other seed widths, DES-based generation, and EC-BBS remain outside this test.

The archive search found attributed sample code in a later Qpopper distribution calling `EZInitRandom`, `EZSeedRandom`, and `EZGenerateRandom`; the underlying implementations were absent [22]. A period Security Builder brochure distinguishes compiled libraries from supplied sample code [23]. The missing manual itself was not recovered. Internet Archive searches that appeared to return it led to software collections whose catalogues matched the title's words separately. Their inspected catalogues and available directory listings supplied no matching manual or SDK filename. This is a search result with limited coverage, not proof that no copy survives.

9 One shared random stream is still a restrictive hypothesis

An unknown initial seed does not make every construction possible. Suppose both field sizes reset to the *same* unknown output stream and apply the same specified conversion. Treat its first two 160-bit blocks, R_0 and R_1 , as completely arbitrary. This grants more freedom than a particular generator or seed can provide.

If the 131-bit result is the low 131 bits of R_0 , and the 193-bit result is the low 193 bits of $R_0 \parallel R_1$, their overlapping 33 bits must agree. The observed values do not:

$$\text{low}_{33}(S_{131}) = 0\text{xa}7879682, \quad (11)$$

$$S_{193} \gg 160 = 0\text{x}23\text{daf}8. \quad (12)$$

No seed can fix that contradiction while leaving those assumptions intact.

The supplied linear-algebra check treats all 320 stream bits as unknowns. It finds explicit inconsistent-equation certificates for five conversions: low complete-block bits, high complete-block bits, polynomial reduction of complete blocks, and masked byte prefixes in either byte order. All 170 constructed positive controls are consistent and yield valid witnesses. Appendix B gives a separate elementary check for the polynomial case.

This result does not rule out width-dependent initialization, a different call offset, output consumed between calls, modular-order reduction, or nonlinear processing. A persistent seed XOR would leave the contradiction intact only if both sizes still began with the identical resulting stream. The point is to state precisely what an initialization explanation must change, rather than treating an unknown seed as an explanation by itself.

10 What would resolve the question

The investigation has reached a concrete boundary. The ten public generators can be checked against early numerical documents. The six binary generators can be reconstructed from their shared starts. Several simple explanations for those starts are inconsistent with the data. None of these steps produces the two larger integers from an independently justified historical input.

The most useful next evidence is therefore specific: the 1997 Security Builder manual identified by document number, the implementations behind the period RNG interfaces, the original parameter-generation program, or a contemporaneous seed and generation log. A recovered source fragment would need to connect the seed, any preprocessing, the integer conversion, the retry rule, and the final point. Reproducing a second constant without fitting it would be especially valuable.

A larger arbitrary seed range would answer a narrower question: whether that range contains a match under the chosen model. It would not settle how a different program operated. The same applies to attractive stories about manual editing or intentional design. The surviving numerical evidence does not establish either story.

10.1 Limits and research record

This is a technical white paper, not a claim that the historical mystery has been solved or that the observations are all novel. The prime fingerprint and earlier cofactor suggestions are credited to prior work. Priority for the additional reconstructions and exclusions has not been established by an exhaustive literature review. A subsequent original record could explain the constants using a model that the present tests never considered.

The arithmetic verifiers trust the standard's subgroup orders and reduction polynomials. They check the stated point identities; they do not independently prove every order prime, every field polynomial irreducible, or every full curve order. Historical text extraction has its own boundaries, including cached pages for the September draft and custom-font normalization for the October PDF. Both the assumptions and the retained controls are part of the evidence package.

OpenAI Codex assisted with source retrieval, programming, hypothesis testing, and preparation of this paper. The numerical claims are supported by executable checks, published control examples, and the independent calculations identified in the text. The paper-preparation replay reran the Python checks in a separate copy of the evidence bundle. It did not recompile every historical C library, and comparison with recorded C outputs should be read on that basis. AI assistance is not a substitute for those verification boundaries.

The original recipe remains unknown. What we can now state precisely is where the published points lead, which proposed routes fail, and which source records would allow the investigation to move from a compatible reconstruction to a historical explanation.

A Scope of the reproducible checks

Table 3 separates exact structural results from finite searches. The full bundle records the source-specific layouts and control vectors. A software name in this table refers to those stated models, not to all uses or versions of the software.

Table 3. Representative checks and their limits.

Check	Result and coverage	Outside the result
Prime shared input	Ten CRT cases; minimum 829 bits under field reduction, 827 under order reduction used as x . Nested low-bit and stated common-seed expansion models are inconsistent.	Curve-specific inputs, extra rejection rules, other expansion, scalar-generation models.
Binary reconstruction	Six curves; both doubling preimages, scan intervals, two compatible root rules.	Historical identity of the scan, root rule, or initial seed.
Challenge resemblance	Exact polynomial-basis trace mismatch and odd normal-basis parity.	Changed input bits or subsequent arithmetic.
FreeLIP	All-state direct-output exclusions for the two starts at 26-, 30-, and 62-bit limb widths with the specified bounds.	Other bounds, hashing, field conversions, interleaved calls, broken machine arithmetic.
1rand48	1,148 declared direct-output layouts excluded for all 48-bit states.	Hashing, field reduction, arbitrary interleaving.
PARI 2.0.16	Three recurrence/packing models excluded; an additional prefix check covers the prime scan intervals.	Different transformations or routines.
NTL 3.8b	No match in 50 distinct API configurations at raw-word offsets 0–4,095; original-source controls and an independent replay agree.	Other seeds, later offsets, or identity with pre-September-1999 code.
GMP 2.0.2	896 default whole-number draw positions across the stated GNU C library models; no match.	Explicit seeding, arbitrary prior calls, other platforms and routines.
Public-seed hashes	102 distinct seeds, stated offsets and encodings; SHA-1, SHA-0, MD5 and RIPEMD-160 tests give no match.	Unknown seeds, arbitrary text, untested transforms.
FIPS 186 experiment	3,264 specified candidate equalities; controls pass, no exact start match.	Unknown seed mixing, other widths, later calls, DES and EC-BBS.
Shared reset stream	Five named linear conversions inconsistent for any two 160-bit blocks.	Different streams, call offsets, or nonlinear processing.

B An elementary check of the polynomial-stream case

The shared-stream polynomial model uses the SEC reduction polynomials

$$f_{131}(z) = z^{131} + z^8 + z^3 + z^2 + 1, \quad f_{193}(z) = z^{193} + z^{15} + 1. \quad (13)$$

Represent the first block as $R_0 = U + Qz^{131}$, where $\deg U < 131$ and $\deg Q < 29$. Its 131-bit remainder is

$$U + Q(1 + z^2 + z^3 + z^8). \quad (14)$$

For the second output, reduce $R_0 z^{160} + R_1$ modulo f_{193} . Its coefficients at degrees 160 through 192 are exactly the low 33 coefficients of R_0 . The folded high part has degree at most 141, and R_1 has degree at most 159, so neither can alter those 33 coefficients.

The two observed outputs therefore require

$$Q(1 + z^2 + z^3 + z^8) \equiv D \pmod{z^{33}}, \quad (15)$$

where the coefficient bitstring for D is

$$D = 0xa7879682 \oplus 0x23daf8 = 0xa7a44c7a. \quad (16)$$

The constant coefficient of the multiplier is one. Its coefficients determine Q successively, with $q_i = d_i + q_{i-2} + q_{i-3} + q_{i-8}$ and negative-index coefficients zero. The forced result modulo z^{33} is $0x1d2aaf2e2$. Its bits above index 28 are nonzero, contradicting $\deg Q < 29$. This independently confirms the linear-algebra exclusion for this one conversion. It is a necessary-condition argument over arbitrary blocks, not an inversion of a particular generator.

C Reproduction and document provenance

The companion archive contains this LaTeX project, the investigation bundle, and a paper-specific validation receipt. The bundle preserves dated checkpoints, including statements later superseded by recovered documents; the paper and `LATEST-FINDINGS.txt` provide the current interpretation. Paths below are relative to `evidence/secp-generator-investigation/` in the companion archive.

```
python -E -s -S -B verify.py
python -E -s -S -B hypotheses.py
python -E -s -S -B certicom_controls.py
python -E -s -S -B followup/binary_pairs/verify_binary.py
python -E -s -S -B shared-starts/run_checks.py
python -E -s -S -B continuation/run_checks.py
python -E -s -S -B origin-followup/run_checks.py
python -E -s -S -B certicom-source/run_checks.py
python -E -s -S -B archive-and-stream/check_shared_stream.py
```

These commands use Python's standard library and require no network access. The checks use assertions; Python optimization must remain disabled. The input fixtures contain public parameter values and exported source evidence. Replaying a fixture is not equivalent to fetching and authenticating every original document again.

The paper was typeset using an adaptation of Guillermo Jiménez's Rho class, Version 3.0.1, released 15 April 2026, with contribution credit retained for Eduardo Gracidas [24]. The source package records the upstream commit, license, and changes for a single-column white paper. Template attribution is separate from the authorship of this analysis.

References

- [1] Certicom Research. *Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters*, September 2000. URL <https://www.secg.org/SEC2-Ver-1.0.pdf>. Version 1.0, 20 September.
- [2] Anthropic. Claude reveals ECDSA curve origins, October 2026. URL <https://www.anthropic.com/research/ecdsa-curve-origins>. 16 October.
- [3] Gregory Maxwell. Answer to: What does the special form of the base point of secp256k1 allow?, 2019. URL <https://crypto.stackexchange.com/a/76010>. Crypto Stack Exchange, answer 76010; includes discussion of earlier cofactor-generation hypotheses.

- [4] John Zweng. Reproducible calculations of the SEC prime-curve half-generators, 2022. URL <https://gist.github.com/johnzweng/863f412689ee383cc41ac7c709ca662c>. Public code, April 2022.
- [5] Nadia Heninger. The state of the art of ECC in real-world protocols, 2024. URL <https://troll.iis.sinica.edu.tw/ecc24/slides/4-05-ecc.pdf>. ECC 2024 presentation, slide 44.
- [6] Certicom. *The Certicom ECC Challenge*, n.d.. URL https://www.certicom.com/content/dam/certicom/images/pdfs/cert_ecc_challenge.pdf. Older downloadable revision. Inspected PDF metadata dates to 2003; not treated as an authenticated 1997 edition.
- [7] Certicom. Curves list: ECC challenge parameters, n.d.. URL <https://www.certicom.com/content/certicom/en/curves-list.html>. Public parameter table; accessed 17 October 2026.
- [8] Certicom Research. *Standards for Efficient Cryptography (SEC) 2: Recommended Elliptic Curve Domain Parameters*, October 1999. URL https://web.archive.org/web/20240414044829if_/https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2840bfaf7cf8f97deaddcd4274e82863ec522ee. Version 0.6, complete archived working draft.
- [9] Certicom Research. *GEC 1: Recommended Elliptic Curve Domain Parameters*, September 1999. URL <https://citeseerx.ist.psu.edu/document?doi=fe42ff586c6b5bd314cee6577221a2675319ed5e&repid=rep1&type=pdf>. Version 0.5. Comparison uses retained cached numerical pages; complete PDF not recovered.
- [10] Standards for Efficient Cryptography Group. Meeting minutes, 16 september 1999, 1999. URL https://web.archive.org/web/20001102094425id_/http://www.secg.org:80/collateral/Minutes19990916Mtg.pdf.
- [11] Wireless Application Protocol Forum. *Wireless Transport Layer Security Specification*, 1999. URL <https://www.c3.hu/docs/wml1.1/SPEC-WTLS-19990211.pdf>. Version 11-Feb-1999; Table 8, printed page 80.
- [12] Wireless Application Protocol Forum. *Wireless Transport Layer Security Specification*, 1998. URL <https://www.c3.hu/docs/wml1.0/WTLS-30-Apr-98.pdf>. Version 30-Apr-1998.
- [13] LiDIA Project. LiDIA 1.1 source archive, including FreeLIP V1.0, n.d. URL <https://ftp.mizar.org/pub/security/replay/crypto/LIBS/math/LiDIA-1.1.tar.gz>. Bundled FreeLIP README dated 1 May 1995; archive hash and controls in the companion evidence.
- [14] PARI/GP Project. PARI 2.0.16 beta source archive, 1999. URL <https://pari.math.u-bordeaux.fr/pub/pari/OLD/2.0/pari-2.0.16.beta.tgz>. Historical release, June 1999.
- [15] MIRACL. Historical MIRACL source archive, n.d. URL <https://www.mirror-service.org/sites/ftp.wiretapped.net/pub/security/cryptography/libraries/math/miracl/miracl.zip>. Mirror listing dated April 1996; source carries earlier dates.
- [16] Maplesoft. RandomTools/LinearCongruence/NewGenerator, n.d. URL <https://www.maplesoft.com/support/help/maple/view.aspx?path=RandomTools/LinearCongruence/NewGenerator>. Current compatibility documentation used to define the modeled assembly; not proof of every historical implementation.
- [17] Certicom. Security builder SDK 2.0, 1998. URL https://web.archive.org/web/19981207011453id_/http://www.certicom.com:80/cSecure/sdk.htm. Archived capture, 7 December 1998.
- [18] Certicom. *VeriSmart elliptic-curve card security policy*, n.d.. URL <https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp55.pdf>. Retained by NIST for certificate 55. Printed pages 144 and 162; PDF creation metadata 8 July 1999. Exact revision/publication date not inferred from metadata alone.

- [19] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*, 1998. URL <https://csrc.nist.gov/files/pubs/fips/186-1/final/docs/fips186-1.pdf>. FIPS PUB 186-1, Appendices 3 and 5.
- [20] ANSI X9.62 Working Group. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 1998. URL <https://safecurves.cr.yp.to/grouper.ieee.org/groups/1363/private/x9-62-09-20-98.pdf>. Working draft dated 20 September 1998; Appendix A.4 and point-arithmetic appendices.
- [21] Henry Haverinen and Joseph Salowey. Extensible authentication protocol method for GSM subscriber identity modules (EAP-SIM). Technical Report RFC 4186, IETF, January 2006. URL <https://www.rfc-editor.org/rfc/rfc4186>. Appendix A.5 is used as a later primitive/feedback control.
- [22] QUALCOMM. Qpopper 4.0.5 source distribution, n.d. URL https://archive.debian.org/debian/pool/main/q/qpopper/qpopper_4.0.5.orig.tar.gz. The sslplus_utils.c wrapper carries SSL Plus 3.0.2 / July 1999 headers; archive member modified in 2003.
- [23] Certicom. Security builder: cryptographic toolkit data sheet, 1999. URL https://web.archive.org/web/20000819094825id_/http://www2.certicom.com:80/pdf/datasheets/securitybuilder.pdf. Printed copyright 1999; archived 19 August 2000.
- [24] Guillermo Jiménez. Rho class, version 3.0.1, 2026. URL <https://github.com/MemoJimenez/Rho-class>. 15 April 2026; contributor Eduardo Gracidas. CC BY 4.0. Upstream commit 70808a3e3e1c780129613573bb1ce64c8e436bb5.