Worm Army: A Difference-of-Squares Recurrence, Multiplier Transformation, and Parallel Search Architecture

White Paper v1.11

TargetDeLab - E3 Prime Number Factorization Research

5 October 2026

Abstract

This initial white paper documents the first stage of the TargetDeLab E3 factorization study. The starting point is a 2017 MATLAB algorithm that represents an odd composite integer as a difference of two squares, but replaces the classical repeated perfect-square test of Fermat factorization with a two-coordinate recurrence driven by a signed residual. The resulting process is called the *Worm*: its "head" and "tail" advance by unit steps while the residual is updated using odd-square increments. The paper then introduces a multiplier transformation, applying independent Worm searches to kN, and finally a Worm Army architecture in which many multiplier instances are evaluated concurrently or in reusable waves. The multiplier idea itself is not claimed as new: multiplication of N before Fermat-style factorization is established prior art, including Lawrence/Lehman-type methods. The research contribution investigated here is the particular recurrence representation, multiplier-population organization, and parallel implementation architecture. This work extends the architectural definition with a formal theoretical-properties section, a controlled benchmark family, a detailed 24-digit case study, and a separately disclosed large-integer campaign extending to 100 decimal digits. The experimental sections distinguish the baseline 8–32 digit campaign from the 40–100 digit campaign, for which $q \approx 3p$ is an explicit construction assumption. Machine cost is reported as timing-derived equivalent CPU clock cycles rather than absolute milliseconds.

1 Scope and Research Position

Let $N = pq$ be an odd semiprime, with $p \leq q$. The work addresses three linked questions: (i) can a difference-of-squares search be expressed as a low-cost recurrence rather than a sequence of square tests; (ii) can multiplication by selected integers k transform an

unfavorable factor geometry into a more favorable one; and (iii) can many such transformed searches be organized as a parallel population of simple, independent engines?

The terminology *Worm* and *Worm Army* is used for the recurrence and its parallel population, respectively. This terminology is project-specific. The underlying difference-of-squares identity is classical, and the use of multipliers kN in Fermat-related factorization predates this work. Accordingly, novelty is not asserted for those mathematical foundations. The purpose of this paper is to define the TargetDeLab formulation precisely enough that its behavior can subsequently be measured and compared reproducibly.

2 Classical Fermat Factorization

Fermat factorization uses the classical difference-of-squares identity [1]:

$$N = x^2 - y^2 = (x + y)(x - y). \tag{1}$$

For $N = pq$ with odd p, q, the target point is

$$x_\star = \frac{p+q}{2}, \qquad y_\star = \frac{q-p}{2}. \tag{2}$$

The classical procedure starts at $x = \lceil \sqrt{N} \rceil$ and repeatedly evaluates $x^2 - N$ until it is a perfect square [1]. Its major strength is therefore also its limitation: when p and q are close, $x_\star$ lies close to $\sqrt{N}$ and the method can terminate rapidly; when the factors are strongly unbalanced, the required search interval grows substantially.

A useful implementation detail of classical Fermat is that $x^2 - N$ itself can be updated incrementally as x advances. Nevertheless, a classical implementation still needs to determine whether the current residual is a perfect square. The Worm formulation changes this search geometry by maintaining both square coordinates and allowing either one to advance.

3 The Worm Recurrence

3.1 Initialization

The original 2017 MATLAB implementation initializes

$$x_0 = \lfloor \sqrt{N} \rfloor + 1, \tag{3}$$

$$y_0 = \left\lfloor \sqrt{x_0^2 - N} \right\rfloor, \tag{4}$$

and defines the signed residual

$$D = x^2 - y^2 - N. \tag{5}$$

A solution is obtained when $D = 0$, at which point

$$p = x - y, \qquad q = x + y \tag{6}$$

(up to ordering).

3.2 TargetDeLab Worm Formulation and Attribution

The name *Worm Recurrence* refers in this paper to the specific TargetDeLab organization of the search as two coordinated integer states, a Head x and a Tail y, controlled by the signed residual

$$D = x^2 - y^2 - N. \tag{7}$$

The Head moves when $D < 0$, the Tail moves when $D > 0$, and $D = 0$ is the score condition. This named Head/Tail state organization is the computational primitive that TargetDeLab subsequently replicates in the Worm Army.

The historical boundary of the claim is important. The difference-of-squares identity and Fermat factorization are classical [1]. Incremental updating of successive square expressions by odd-number additions is also known in optimized descriptions of Fermat's method [2]. TargetDeLab therefore does not claim invention of those underlying identities or of incremental square arithmetic. The contribution claimed here is the particular Worm formulation and organization defined above, together with its later multiplier-population and Worm Army architecture.

3.3 Elimination of Repeated Roots and Divisions in the Worm Recurrence

Square-root operations are used to initialize the Worm state, for example

$$x_0 = \lfloor \sqrt{N} \rfloor + 1, \qquad y_0 = \left\lfloor \sqrt{x_0^2 - N} \right\rfloor. \tag{8}$$

After x_0, y_0, and $D_0 = x_0^2 - y_0^2 - N$ have been formed, the steady-state recurrence does not need another square-root extraction or division to generate its successive square entities. This follows directly from the difference between consecutive squares:

$$(x + 1)^2 - x^2 = 2x + 1, \tag{9}$$

$$(y+1)^2 - y^2 = 2y + 1. \tag{10}$$

Consequently, a Head movement updates the residual by

$$D_{\text{new}} = D + (2x + 1), \tag{11}$$

whereas a Tail movement updates it by

$$D_{\text{new}} = D - (2y + 1). \tag{12}$$

Thus, after initialization, the Worm evolves the equivalent square quantities through integer increments and additions/subtractions while a sign comparison selects the moving coordinate. The score test is simply $D = 0$. No division and no repeated root extraction are required in this recurrence loop.

This is an implementation distinction from the straightforward textbook Fermat presentation, which repeatedly asks whether $x^2 - N$ is a perfect square [1]. It is not a claim that every optimized Fermat implementation performs a hardware square root on every iteration; incremental square updates are established prior art [2]. The significance for the Worm Army is that the TargetDeLab engine can be represented by a compact integer state and replicated without a root or divider in its steady-state recurrence datapath.

3.4 Head and Tail Motion

The recurrence follows directly from the difference between consecutive squares. If $D < 0$, the x coordinate, the Worm's *head*, is incremented:

$$x \leftarrow x + 1, \tag{13}$$

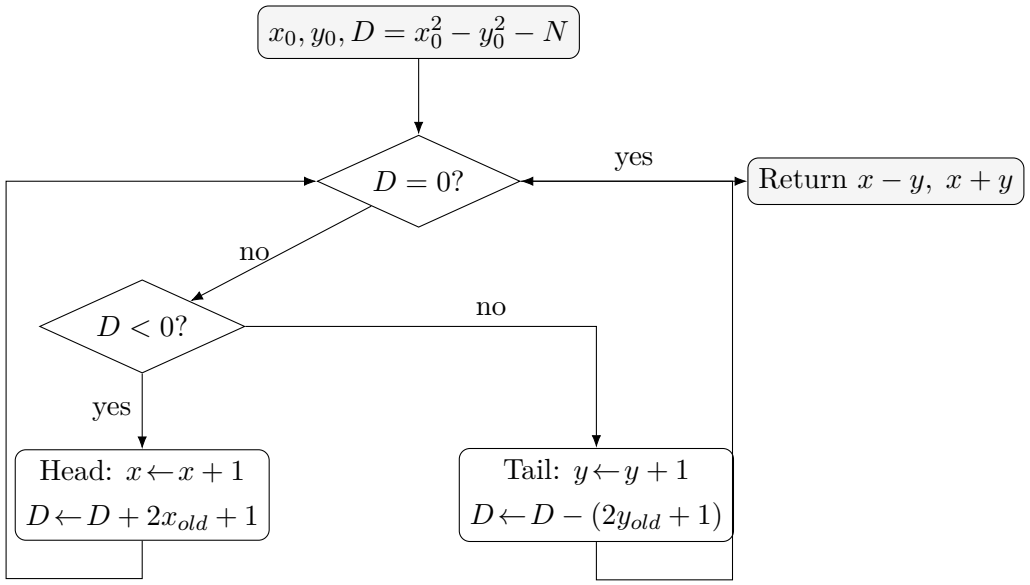
$$D \leftarrow D + 2x_{\text{old}} + 1. \tag{14}$$

If $D > 0$, the y coordinate, the *tail*, is incremented:

$$y \leftarrow y + 1, \tag{15}$$

$$D \leftarrow D - (2y_{\text{old}} + 1). \tag{16}$$

Thus, after initialization, the inner recurrence requires comparisons, integer increments, additions, and subtractions. It does not require a square root, division, or perfect-square test at every iteration.



3.5 Relation to Fermat and Practical Benefits

The Worm does *not* remove the fundamental dependence on factor separation. It changes the per-step arithmetic and the path through the integer lattice. Three implementation benefits motivate its study:

1. **Cheap steady-state recurrence.** Expensive initialization operations are separated from a loop dominated by integer add/subtract/compare operations.
2. **Deterministic state.** Each engine has a compact state (x, y, D, k, T), convenient for software threads or replicated hardware datapaths.
3. **Natural independence.** Once a multiplier k is assigned, one Worm can advance without communication with other Worms until a factor is reported.

These are architectural advantages, not by themselves a proof of asymptotically superior factorization complexity.

3.6 Exact Iteration Interpretation for $k = 1$

For the original Worm applied directly to $N = pq$, define

$$x_0 = \lfloor \sqrt{N} \rfloor + 1, \qquad y_0 = \left\lfloor \sqrt{x_0^2 - N} \right\rfloor. \tag{17}$$

Because exactly one coordinate advances per recurrence step, the number of steps to the target point is

$$T = (x_\star - x_0) + (y_\star - y_0) = q - x_0 - y_0. \tag{18}$$

This identity is useful because it makes the principal weakness explicit: for an unfavorable factor ratio, the Worm may still require a very large number of individually cheap steps. The original MATLAB source recorded, for example, 4,001 iterations for 2657×8971 and progressively larger iteration counts for its historical benchmark set. These historical measurements motivate, rather than prove, the multiplier strategy described next.

Iteration complexity. The exact $k = 1$ iteration count is $T = q - x_0 - y_0$, so the recurrence requires $O(T)$ Worm steps. For a semiprime family with fixed factor ratio $r = q/p$, one has $p = \sqrt{N/r}$ and $q = \sqrt{Nr}$; asymptotically this gives

$$T \sim \sqrt{N}\,(\sqrt{r} - 1), \tag{19}$$

and therefore $O(\sqrt{N})$ recurrence iterations when r is fixed. This is not a universal $O(\sqrt{N})$ bound. For increasingly asymmetric factor pairs, q can be of order N, and the $k = 1$ recurrence count can correspondingly approach $O(N)$. These bounds count recurrence iterations; bit-operation cost also grows with operand length.

4 Theoretical Properties of the Worm Recurrence and Worm Army

This section separates properties that follow rigorously from the recurrence from hypotheses that still require experimental study. The results below do not establish a new global asymptotic factorization bound. In particular, they do not bound the number of multiplier candidates or Army waves required to encounter a successful k.

4.1 Recurrence Invariance and Score Correctness

Theorem 1 (Exact residual invariant)**.** *Let the initialized Worm state satisfy*

$$D_n = x_n^2 - y_n^2 - N. \tag{20}$$

If $D_n < 0$ and the Head update is applied, or if $D_n > 0$ and the Tail update is applied, then the updated state satisfies the same identity exactly. Consequently, after initialization no repeated square-root extraction, division, or explicit recomputation of x_n^2 and y_n^2 is required to maintain the residual.

Proof. For a Head move, $x_{n+1} = x_n + 1$ and $y_{n+1} = y_n$. Hence

$$x_{n+1}^2 - y_{n+1}^2 - N = (x_n + 1)^2 - y_n^2 - N \tag{21}$$
$$= D_n + 2x_n + 1, \tag{22}$$

which is exactly the stated Head recurrence. For a Tail move, $x_{n+1} = x_n$ and $y_{n+1} = y_n + 1$, so

$$x_{n+1}^2 - y_{n+1}^2 - N = x_n^2 - (y_n + 1)^2 - N \tag{23}$$
$$= D_n - (2y_n + 1), \tag{24}$$

which is exactly the Tail recurrence. Induction over the recurrence index proves the invariant for every subsequent state. □

Corollary 2 (Correctness of the score condition)**.** *If a Worm reaches $D_n = 0$, then*

$$N = (x_n - y_n)(x_n + y_n). \tag{25}$$

Therefore, when $1 < x_n - y_n < N$, the two quantities $x_n - y_n$ and $x_n + y_n$ give a nontrivial factorization of N up to ordering.

Proof. Setting the invariant to zero gives $x_n^2 - y_n^2 = N$. Factoring the difference of two squares yields the result. □

4.2 Coordinate Transformation and Exact Step Accounting

Proposition 3 (Unit advance of the factor coordinate)**.** *Define*

$$u_n = x_n + y_n, \qquad v_n = x_n - y_n. \tag{26}$$

Then

$$D_n = u_n v_n - N. \tag{27}$$

Every Worm recurrence cycle increments u by exactly one. A Head move gives $(u, v) \mapsto (u + 1, v + 1)$, whereas a Tail move gives $(u, v) \mapsto (u + 1, v - 1)$.

Proof. The identity $x^2 - y^2 = (x + y)(x - y)$ gives $D = uv - N$. If x alone is incremented, both $x + y$ and $x - y$ increase by one. If y alone is incremented, $x + y$ increases by one while $x - y$ decreases by one. □

For an odd semiprime $N = pq$, $p \le q$, the desired factor point has

$$u_\star = q, \qquad v_\star = p. \tag{28}$$

Therefore, whenever the Worm reaches this target from its prescribed initialization, its recurrence count is not an empirical estimate but the exact quantity

$$T = q - u_0 = q - x_0 - y_0, \tag{29}$$

with

$$x_0 = \lfloor \sqrt{N} \rfloor + 1, \qquad y_0 = \left\lfloor \sqrt{x_0^2 - N} \right\rfloor. \tag{30}$$

Equivalently,

$$\boxed{T = q - (\lfloor \sqrt{N} \rfloor + 1) - \left\lfloor \sqrt{(\lfloor \sqrt{N} \rfloor + 1)^2 - N} \right\rfloor.} \tag{31}$$

This result explains a central limitation of the unmultiplied Worm: the recurrence cycle itself is cheap, but the target factor coordinate can still be far from u_0.

4.3 Multiplier Geometry and Zero-Iteration Condition

Proposition 4 (Multiplier balancing geometry)**.** *Let $N = pq$ and factor a multiplier as $k = ab$. Then*

$$kN = (ap)(bq). \tag{32}$$

When ap and bq have compatible parity, the associated difference-of-squares coordinates are

$$x_k = \frac{ap + bq}{2}, \qquad y_k = \frac{|bq - ap|}{2}. \tag{33}$$

Thus the transformed difference-of-squares geometry becomes favorable as ap approaches bq, equivalently as a/b approaches q/p up to exchanging the factor roles.

Proof. The product identity is immediate from $kN = abpq$. For two same-parity factors $A = ap$ and $B = bq$, the standard identities $x = (A + B)/2$ and $y = |B - A|/2$ satisfy $x^2 - y^2 = AB = kN$. □

The use of multipliers in Fermat-related factorization is established prior art [4, 3]; the proposition is included to state precisely the geometry used by the Worm Army rather than to claim the multiplier principle as new.

Theorem 5 (Zero-iteration criterion for the simple multiplier case)**.** *Let $N = pq$ with odd p, q, and let an admissible multiplier k produce the same-parity factor pair (kp, q) of kN. Assume $kp \neq q$, so that the transformed product is not the square associated with this pair.*

Define

$$x_\star = \frac{kp+q}{2}, \qquad y_\star = \frac{|q-kp|}{2} > 0. \tag{34}$$

The prescribed Worm initialization lands directly on $(x_\star, y_\star)$*, and hence scores with* $T = 0$*, if and only if*

$$\boxed{(q-kp)^2 \le 4(kp+q-1).} \tag{35}$$

Proof. Because $kN = x_\star^2 - y_\star^2$ and $y_\star > 0$, the initialization has $x_0 = x_\star$ exactly when

$$(x_\star - 1)^2 \le kN < x_\star^2. \tag{36}$$

Substituting $kN = x_\star^2 - y_\star^2$, the left inequality becomes

$$y_\star^2 \le 2x_\star - 1. \tag{37}$$

When this holds, the second initialization gives

$$y_0 = \left\lfloor \sqrt{x_\star^2 - kN} \right\rfloor = y_\star, \tag{38}$$

so $D_0 = 0$. Substituting $x_\star = (kp+q)/2$ and $y_\star = |q-kp|/2$ and multiplying by four yields the boxed condition. The reverse implications follow by the same identities. □

Writing $k = q/p + \delta$ gives $q - kp = -p\delta$. Near the balanced point $kp \approx q$, the exact zero-iteration inequality therefore suggests the local approximation

$$|\delta| \lesssim \frac{\sqrt{8q}}{p}, \tag{39}$$

and an approximate full width

$$\Delta k_0 \approx \frac{4\sqrt{2q}}{p}. \tag{40}$$

This approximation is an interpretation of the exact criterion, not a separate guarantee. It predicts that some strongly asymmetric semiprimes can possess wider low- or zero-iteration multiplier windows.

4.4 Bounded Recurrence Work of a Failed Full Wave

Proposition 6 (Failed-wave recurrence bound)**.** *For a full Army wave of* W_P *physical Worms using the research budget*

$$B = \left\lfloor \frac{\sqrt{N}}{W_P} \right\rfloor, \tag{41}$$

if every Worm exhausts its budget without scoring, then the total number of Worm recurrence cycles in that wave satisfies

$$\boxed{C_{\text{wave}} = W_P B \leq \sqrt{N}.} \tag{42}$$

Proof. For any real z, $\lfloor z \rfloor \leq z$. Therefore

$$W_P B = W_P \left\lfloor \frac{\sqrt{N}}{W_P} \right\rfloor \leq W_P \frac{\sqrt{N}}{W_P} = \sqrt{N}. \tag{43}$$

□

This bound concerns recurrence cycles in one failed full wave only. It excludes initialization, multiplier formation, GCD, scheduling, synchronization, and arbitrary-precision costs. More importantly, it does not bound the number of failed waves before a useful multiplier is encountered. A global complexity result would require a factor-independent bound on that multiplier-search quantity; no such result is claimed here.

5 Multiplier Transformation: From One Worm to a Family of Problems

5.1 Definition

Instead of searching only

$$N = x^2 - y^2, \tag{44}$$

one may search a family

$$kN = x_k^2 - y_k^2, \tag{45}$$

where k is a positive integer satisfying the necessary parity conditions. If a nontrivial difference-of-squares representation of kN is found, factors of the original N can be recovered by greatest-common-divisor tests such as

$$\gcd(x_k - y_k, N), \qquad \gcd(x_k + y_k, N). \tag{46}$$

For odd N, a multiplier $k \equiv 2 \pmod 4$ is unsuitable because $kN \equiv 2 \pmod 4$, and integers congruent to 2 modulo 4 cannot be represented as a difference of two integer squares.

5.2 Why k Can Change the Geometry

Suppose $N = pq$ and write $k = ab$. One possible factor pairing of kN is

$$kN = (ap)(bq). \tag{47}$$

A Fermat-style search is favorable when these transformed factors are close. Hence a useful multiplier decomposition approximately satisfies

$$ap \approx bq, \tag{48}$$

or, equivalently,

$$\frac{a}{b} \approx \frac{q}{p} \tag{49}$$

(up to exchanging the roles of the factors). The multiplier therefore changes the geometry of the difference-of-squares problem: it does not change the unknown factorization of N, but can produce a multiple kN whose relevant factor pair is much better balanced.

This principle is established prior art. Lawrence generalized Fermat-style searches to multiples kN, and Lehman later provided a systematic multiplier method [4, 3]. Modern analyses of Lehman's method retain this multiplier-assisted balancing framework [4]. The Worm Army therefore does not claim invention of multiplier-assisted Fermat factorization; it investigates a specific recurrence engine and population architecture built around that principle.

5.3 Digit-Scale k Zones

The hidden ratio q/p is unavailable in a blind factorization. The current Worm Army design therefore uses only information available from N itself to define broad multiplier zones. If N has D decimal digits and a possible factor-digit split is (d_p, d_q) with $d_p + d_q \approx D$, then the digit imbalance

$$\Delta = d_q - d_p \tag{50}$$

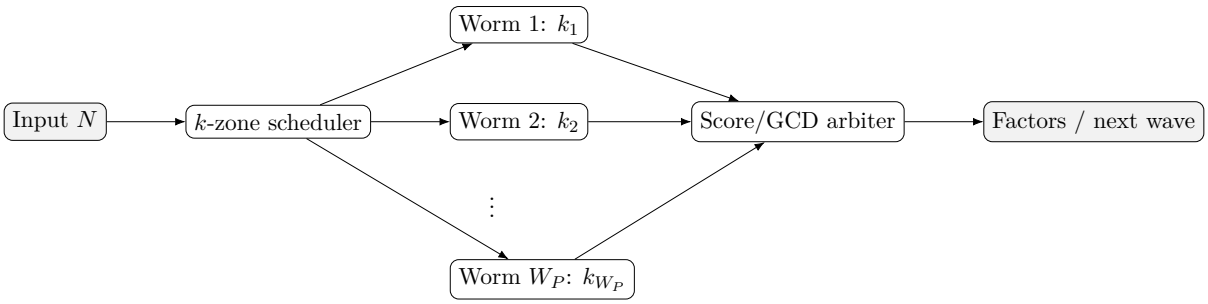
suggests that balancing multipliers may occur on scales near 10^{Δ}, with neighboring decades retained because leading digits and the decomposition $k = ab$ matter. For a 12-digit semiprime, illustrative factor-digit hypotheses include $(6, 6), (5, 7), (4, 8), (3, 9), (2, 10), (1, 11)$, corresponding to increasingly large multiplier scales.

This is presently a *search heuristic*, not a theorem that the correct multiplier must occur in a specified decimal band. The zone definitions and traversal order must be fixed before benchmark execution to prevent hidden knowledge of p or q from leaking into the experiment. Prime-only multiplier selection is also rejected: useful multipliers can be composite.

6 The Worm Army Parallel Architecture

6.1 Independent Engines

For a set of multiplier candidates $\mathcal{K} = \{k_1, k_2, \ldots\}$, each Worm initializes and evolves its own state for $k_r N$. There is no need for state exchange between active Worms. A successful engine performs the GCD recovery against N and signals termination to the population.



6.2 Physical Worms Versus Logical Multiplier Population

The logical k population may be extremely large; it is neither necessary nor desirable to instantiate one physical engine per candidate. Let W_P denote the number of simultaneously active physical Worms. The current implementation concept uses a reusable bank (for example, $W_P = 65{,}536$ as an engineering target) and streams candidate k values through it in waves:

$$\mathcal{K}_1, \mathcal{K}_2, \ldots, \qquad |\mathcal{K}_j| \le W_P. \tag{51}$$

A physical slot executes *load* $k \to$ *run* $\to$ *score or timeout* $\to$ *reset* $\to$ *load next* k. This separates mathematical search breadth from physical resource count and maps naturally to multicore software, GPU-like execution models, or replicated FPGA engines.

6.3 Per-Wave Iteration Budget

The present research rule assigns a full-wave per-Worm budget

$$B = \left\lfloor \frac{\sqrt{N}}{W_P} \right\rfloor. \tag{52}$$

If every Worm in a failed full wave consumes the complete budget, the aggregate recurrence work is approximately

$$W_P B \lesssim \sqrt{N}. \tag{53}$$

If a winning Worm terminates after $T_{\text{win}} < B$ steps and all engines are stopped at that point, the recurrence work of the successful wave is bounded by approximately $W_P T_{\text{win}} <$

$\sqrt{N}$, excluding initialization, scheduling, GCD, and synchronization costs. Consequently, increasing W_P trades search depth per multiplier for greater simultaneous breadth.

This budget is a design rule under investigation, not an established complexity theorem for the complete algorithm. A fair implementation must also count the cost of forming kN, arbitrary-precision square-root initialization, resetting engines, and traversing unsuccessful waves.

6.4 Success Metric and Accounting

For the active population $\mathcal{K}_j$, define

$$T_{\text{win}} = \min_{k \in \mathcal{K}_j} T(k). \tag{54}$$

A wave succeeds if at least one candidate reaches a valid factor within its budget. Useful experimental outputs include the winning multiplier k_{win}, T_{win}, wave index, number of k candidates tested, wall-clock time, and aggregate recurrence work. Parallel latency and total algorithmic work must be reported separately: adding processors can reduce elapsed time without reducing total arithmetic work.

7 Worm Army Search Mechanics: Detailed 24-Digit Example

Consider the strict semiprime

$$N = 120000000159200000052761 = 200000000129 \times 600000000409. \tag{55}$$

The factors are both prime. They are displayed here for verification, but the Army scheduler uses only N, the predefined population size, and the predefined ascending multiplier stream.

For a 24-digit N, possible factor-digit hypotheses range from nearly balanced $(12, 12)$ through increasingly asymmetric splits. The general digit-zone heuristic associates an imbalance $\Delta = d_q - d_p$ with multiplier scales around 10^{Δ}, retaining neighboring decades. The present case actually has a $(12, 12)$ split, so the smallest multiplier zone is immediately relevant; the blind scheduler nevertheless does not know that fact in advance. Its first wave contains 32,768 admissible k values in ascending order, skipping only $k \equiv 2 \pmod 4$. The largest k reached by that first wave is 43,691 and

$$B = 10,571,599. \tag{56}$$

The original $k = 1$ Worm requires

$$T(1) = 253,589,287,085 \tag{57}$$

recurrence cycles for this case, while classical Fermat requires 53,589,838,525 successive x candidates. By contrast, the first scoring Army multiplier is

$$\boxed{k = 3, \qquad T(3) = 0.} \tag{58}$$

At initialization, the difference-of-squares representation of $3N$ already yields a factor pair whose GCDs with N recover the two prime factors.

The full first-wave scan contains 120 distinct valid zero-cycle multipliers. The first values are

$$3, 12, 27, 48, 75, 108, 147, 192, 243, 300, 363, 432, 507, 588, 675, \ldots \tag{59}$$

and the last five within the wave are

$$40368,\ 41067,\ 41772,\ 42483,\ 43200. \tag{60}$$

For this constructed family these values follow the visible sequence $k = 3m^2$ over the portion admitted by the first-wave traversal. Multiplication by a square preserves the same favorable transformed factor ratio, explaining why the zero-cycle result is not an isolated jackpot. This observation is useful architecturally, but it must not be generalized to unknown semiprimes without measurement: the benchmark family was deliberately generated with $q \approx 3p$.

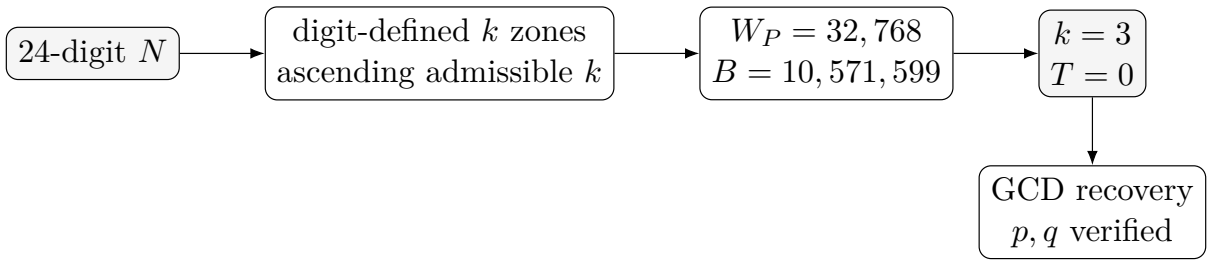


Figure 1: 24-digit controlled case-study path. The scheduler is factor-blind; disclosure of p, q is used only for verification and interpretation.

8 Experimental Campaigns

8.1 Measurement Basis and Reporting Discipline

Two experimental campaigns are reported separately so that their assumptions cannot be mixed. The development benchmark used a single C++ process compiled with GNU g++ 14.2.0 at `-O3` on an AMD EPYC 9V74 virtualized host. Five runs were made and the median elapsed time was retained internally. To avoid presenting host-specific milliseconds as an absolute performance claim, the figures and result tables report *estimated equivalent clock cycles*, computed as

$$C_{\mathrm{eq}} = t_{\mathrm{measured}} f_{\mathrm{ref}}, \tag{61}$$

using the observed reference clock $f_{\mathrm{ref}} = 2.596124$ GHz of the development host during this campaign. These are timing-derived equivalent cycles, not hardware performance-counter measurements. They permit only a rough cross-machine estimate because IPC, microarchitecture, memory behavior, compiler optimization, virtualization, and BigInt implementation also affect elapsed time.

The Fermat and Single-Worm counts reported in the result tables are *algorithmic iteration counts*, not CPU clock-cycle counts. A Worm recurrence iteration and a Fermat candidate iteration do not necessarily carry the same computational workload and therefore their numerical ratio must not be interpreted directly as a runtime or CPU-cost ratio. The timing-derived quantity C_{eq} is reported separately and must not be confused with either algorithmic iteration count.

8.2 Experiment I: Baseline Campaign, 8–32 Digits

8.2.1 Experimental setup

Experiment I contains only the 8, 12, 16, 20, 24, 28, and 32 digit cases. The multiplier scheduler is factor-blind: it receives N and traverses positive k values in ascending order, rejecting only $k \equiv 2 \pmod 4$; neither p, q, nor the ratio q/p is supplied to the scheduler. The factors are shown only for reproducibility and verification.

A necessary disclosure is that the present baseline dataset itself was constructed with factors close to $q \approx 3p$. Thus the *algorithm* is not given a $q \approx 3p$ pre-assumption, but the *benchmark family* has that geometry. Importantly, this favorable geometry does *not* reduce, focus, or otherwise guide the generated multiplier population in Experiment I: the scheduler still generates the complete predefined ascending admissible k population without factor-ratio knowledge. The construction affects the outcome—in particular, it makes the naturally encountered $k = 3$ highly favorable—but not the rule by which the k values

are produced. Experiment I therefore must not be interpreted as a random-semiprime campaign. This distinction is stated explicitly here to prevent the favorable early $k = 3$ result from being mistaken for general blind-factorization behavior.

Each active Worm receives

$$B = \left\lfloor \frac{\sqrt{N}}{W_P} \right\rfloor \tag{62}$$

recurrence iterations. A score at initialization is recorded as $T = 0$. The first score terminates the logical attack; for benchmarking, the initialization of the complete first wave was also scanned so that the density of zero-cycle candidates could be measured.

Table 1: Experiment I configuration, 8–32 digits. The k stream is ascending and factor-blind; $k \equiv 2 \pmod 4$ is skipped.

Digits	W_P	B	First-wave active k	Arithmetic
8	2,048	1	2,048	BigInt
12	4,096	84	4,096	BigInt
16	8,192	4,228	8,192	BigInt
20	16,384	211,431	16,384	BigInt
24	32,768	10,571,599	32,768	BigInt
28	65,536	528,579,958	65,536	BigInt
32	65,536	52,857,995,836	65,536	BigInt

Table 2: Experiment I strict prime-by-prime inputs, 8–32 digits. Factors are disclosed for reproducibility and were not supplied to the scheduler.

Digits	p	q	$N = pq$
8	2,129	6,389	13,602,181
12	200,131	600,401	120,158,852,531
16	20,000,147	60,000,511	1,200,019,040,075,117
20	2,000,000,137	6,000,000,421	12,000,001,664,000,057,677
24	200,000,000,129	600,000,000,409	120,000,000,159,200,000,052,761
28	20,000,000,000,129	60,000,000,000,433	1,200,000,000,016,400,000,000,055,857
32	2,000,000,000,000,143	6,000,000,000,000,431	12,000,000,000,001,720,000,000,000,061,633

Table 3: Experiment I results. C_{eq} is the timing-derived equivalent CPU clock-cycle count at the 2.596124 GHz reference clock, not a hardware-counter measurement.

Digits	Fermat iterations	Single-Worm iterations	k_{win}	T_{win}	Zero-T k's	C_{eq}
8	570	2,620	3	0	30	1.887×10^{6}
12	53,626	253,100	3	0	42	3.224×10^{6}
16	5,359,038	25,357,781	3	0	60	7.308×10^{6}
20	535,898,423	2,535,829,699	3	0	85	1.682×10^{7}
24	53,589,838,525	253,589,287,085	3	0	120	3.376×10^{7}
28	5,358,983,848,666	25,358,976,686,091	3	0	170	7.158×10^{7}
32	535,898,384,862,284	2,535,898,352,375,989	3	0	170	7.430×10^{7}

Iteration-count interpretation. The Fermat and Single-Worm columns above count algorithmic iterations, not equivalent CPU work. After initialization, a Worm recurrence step is organized around a sign comparison, one coordinate increment, and an additive/subtractive residual update. Classical Fermat implementations have a different per-iteration workload; optimized implementations may also use incremental square arithmetic. Consequently, the ratio between these two iteration columns is not a measured speed or CPU-cost ratio. C_{eq} is the separate timing-derived cost quantity.

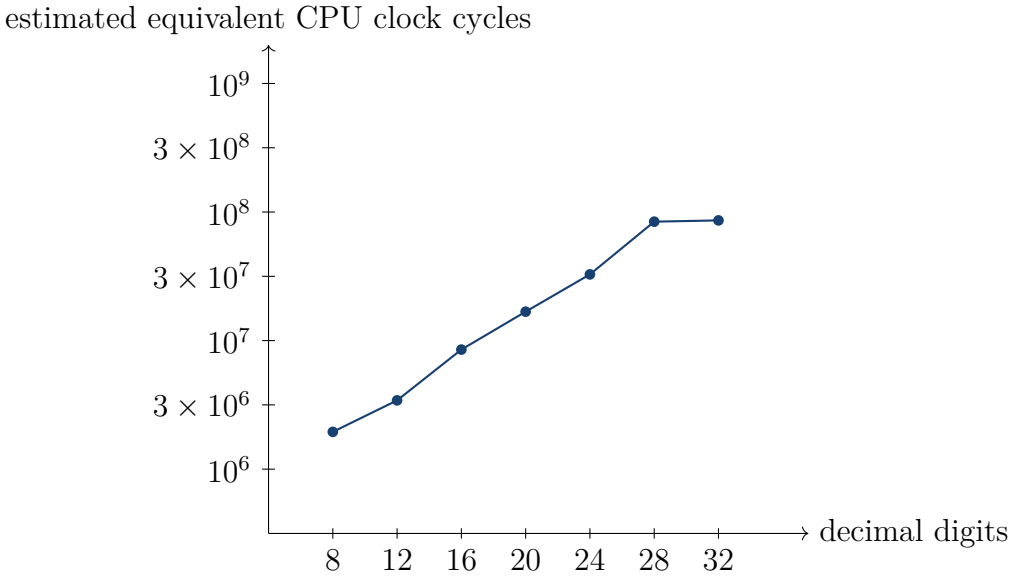


Figure 1(a): Experiment I, 8–32 digits. Timing cost is reported as estimated equivalent CPU clock cycles using the 2.596124 GHz development-host reference clock. The scheduler receives no factor-ratio input; however, the disclosed benchmark dataset itself has $q \approx 3p$, so the curve is not an average-case or random-semiprime result.

8.3 Experiment II: Controlled Large-Integer Campaign, 40–100 Digits

8.3.1 Pre-assumption and purpose

Experiment II is deliberately separated from Experiment I. Its 40, 60, 80, and 100 digit semiprimes are explicitly constructed under the condition

$$\boxed{q \approx 3p}. \tag{63}$$

This is a benchmark pre-assumption. It makes the small multiplier $k = 3$ geometrically favorable because $3p$ and q are nearly balanced. Unlike Experiment I, whose predefined multiplier population is generated without using the favorable ratio, Experiment II deliberately constructs the operands so that the relevant multiplier region occurs immediately at small

k. The experiment still evaluates a population of multipliers and is not simply a test of $k = 3$; however, it intentionally avoids the potentially enormous blind search that could be required to locate a useful multiplier region for an arbitrary large semiprime. Consequently, Experiment II tests large-integer initialization, multiplier traversal, GCD recovery, and implementation scaling under favorable disclosed geometry. It does *not* measure the general cost of discovering k_{win}, and it does *not* demonstrate blind efficient factorization of arbitrary 40–100 digit semiprimes.

Table 4: Experiment II configuration, 40–100 digits. All cases use the disclosed $q \approx 3p$ construction.

Digits	W_P	B	Arithmetic
40	65,536	5.286×10^{14}	BigInt
60	65,536	5.286×10^{24}	BigInt
80	65,536	5.286×10^{34}	BigInt
100	65,536	5.286×10^{44}	BigInt

Table 5: Experiment II strict prime-by-prime inputs, 40–100 digits. Long values are wrapped vertically; every digit is retained.

Digits	Quantity	Value
40	p	20000000000000000153
	q	60000000000000000581
		12000000000000000208
	N	00000000000000008893
60	p	200000000000000000000000000141
	q	600000000000000000000000000487
		120000000000000000000000000182
	N	000000000000000000000000000068667
		20000000000000000000
80	p	00000000000000000351
		60000000000000000000
	q	00000000000000001103
		1200000000000000000000000000000000000431
	N	2000000000000000000000000000000000387153
		200000000000000000000000
100	p	000000000000000000000333
		600000000000000000000000
	q	000000000000000000001069
		120000000000000000000000000000000000
		000000000000041360000000000000000
	N	00000000000000000000000000355977

Table 6: Experiment II results under the disclosed $q \approx 3p$ construction. C_{eq} is timing-derived at the 2.596124 GHz reference clock.

Digits	Fermat iterations	Single-Worm iterations	k_{win}	T_{win}	Zero-T k's	C_{eq}
40	5.359×10^{18}	2.536×10^{19}	3	0	170	2.178×10^{8}
60	5.359×10^{28}	2.536×10^{29}	3	0	170	2.448×10^{8}
80	5.359×10^{38}	2.536×10^{39}	3	0	170	4.927×10^{8}
100	5.359×10^{48}	2.536×10^{49}	3	0	170	5.500×10^{8}

Iteration-count interpretation. As in Experiment I, the Fermat and Single-Worm columns are algorithmic iteration counts, not CPU clock cycles and not directly comparable measures of computational cost. The Worm recurrence is deliberately lightweight after initialization, whereas Fermat implementations have a different per-iteration workload and may themselves use incremental arithmetic. C_{eq} remains the separate timing-derived experimental cost metric.

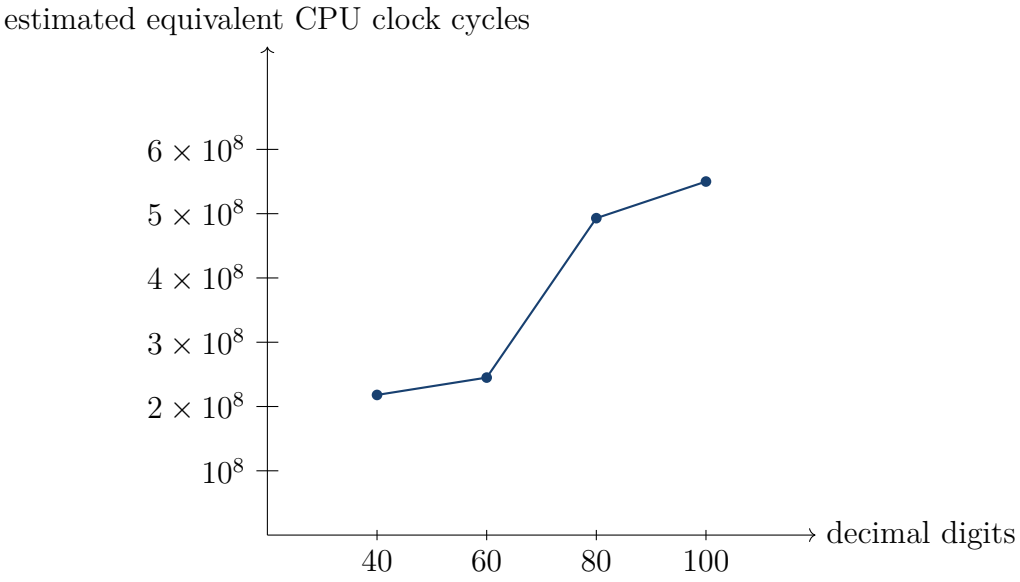


Figure 1(b): Experiment II, 40–100 digits, under the explicit benchmark construction $q \approx 3p$. Timing cost is shown as estimated equivalent CPU clock cycles using the 2.596124 GHz development-host reference clock. The favorable $k = 3$ geometry is a disclosed property of this campaign; this figure is not evidence of general blind 40–100 digit semiprime factorization.

8.3.2 Interpretation

In all four Experiment II cases, $k = 3$ scores at initialization, so $T_{\text{win}} = 0$. The 100-digit case verifies that the current Boost.Multiprecision representation and initialization path execute correctly at this scale for the controlled family. The reported cost primarily measures first-wave initialization and BigInt arithmetic rather than a difficult recurrence search.

The classical Fermat and Single-Worm values reported in Table 6 are *algorithmic iteration counts*, not equivalent CPU clock-cycle counts. They therefore must not be compared directly as measures of computational cost. For this controlled $q \approx 3p$ geometry, the Single-Worm recurrence requires approximately 4.73 times as many algorithmic iterations as the classical Fermat candidate progression. However, after initialization a Worm recurrence iteration is deliberately reduced to a sign comparison, a coordinate increment, and an additive/subtractive residual update. Classical Fermat implementations have a different per-iteration workload, although optimized implementations can likewise exploit incremental square arithmetic. Consequently, the 4.73-fold iteration-count ratio does not imply that the Worm requires 4.73 times the execution cost. The timing-derived C_{eq} quantity is reported separately and must not be confused with either algorithmic iteration count.

The observed Army efficiency in Experiment II comes primarily from the favorable multiplier geometry and parallel breadth, not from a universally shorter $k = 1$ Worm trajectory. Moreover, because the $q \approx 3p$ construction deliberately places a useful multiplier region at small k, this campaign does not measure the cost of locating that region when the factor ratio is unknown.

9 Reference C++ BigInt Implementation

A working C++ implementation of the Worm Army using Boost.Multiprecision `cpp_int` arbitrary-precision integer arithmetic is provided as a companion to this paper. Its purpose is to allow the reader to reproduce and verify the reported behavior and to perform additional experiments with other semiprimes, multiplier zones, Worm populations, and recurrence budgets. The availability of arbitrary-precision arithmetic is an implementation capability and must not be interpreted as evidence that arbitrary semiprimes of a corresponding digit length are practically factorable by the present method.

9.1 User Configuration

The supplied program is configured through `parameters_inp.txt`. The principal user parameters are summarized in Table 7.

Table 7: User-configurable parameters of the companion C++ BigInt implementation.

Parameter	Purpose
`N`	Decimal integer to be factored.
`MAX_WORMS`	Maximum Worm population processed in one software batch.
`K_START`	First multiplier k in the requested search zone.
`K_END`	Last multiplier k in the requested search zone.
`MAX_BATCHES`	Maximum number of batches; zero processes batches until the configured k zone is exhausted.
`ODD_K_ONLY`	When true, restricts the candidate stream to odd multipliers. Independently, $k \equiv 2 \pmod 4$ is skipped automatically.
`MAX_ITERATIONS_PER_WORM`	Operational safety cap on recurrence iterations assigned to an individual Worm.

For each run the program still computes and logs the research budget

$$B_{\text{theory}} = \left\lfloor \frac{\sqrt{N}}{W} \right\rfloor . \tag{64}$$

When `MAX_ITERATIONS_PER_WORM` is nonzero, the operational budget is

$$B_{\text{run}} = \min\left(B_{\text{theory}},\ \texttt{MAX_ITERATIONS_PER_WORM}\right). \tag{65}$$

Both values are reported by the program, allowing the reader to distinguish the mathematical research rule from a finite experimental safety cap. The parameters therefore permit direct reproduction of a stated k zone and Worm population while also allowing the user to explore alternative configurations. In this software reference implementation, `MAX_WORMS` denotes the Worm population processed by the program; it must not be interpreted as a claim that the executable instantiates that number of physically parallel hardware engines.

9.2 Search-Limit Termination

If all Worms in a software batch exhaust their assigned B_{run} recurrence budget without scoring, that batch is recorded as a failure and the executable proceeds to the next batch of admissible multipliers in the configured k zone. This process continues until a factor is recovered, the interval `K_START` through `K_END` is exhausted, or a nonzero `MAX_BATCHES` limit is reached.

When the configured search limits are exhausted without a successful Worm, the program reports `NO_FACTOR_IN_SEARCH_LIMIT`. This status means only that no factor was found within the selected multiplier range, batch limit, and per-Worm iteration budget. It must not be interpreted as a statement that N is unfactorable by the Worm Army. The user may extend or move the k zone, alter the recurrence cap, or perform additional experiments as

appropriate.

10 Limitations and Claims Deliberately Not Made

At this stage, the following distinctions are essential. The Worm reduces the arithmetic complexity of an individual steady-state recurrence step, but that alone does not establish a superior asymptotic factorization algorithm. The multiplier concept has substantial prior art and is not presented as a TargetDeLab invention. The decimal k-zone strategy is presently heuristic. The parallel architecture provides an obvious route to lower wall-clock latency when useful k values are covered, but speedup must be measured against aggregate work, initialization overhead, and hardware resources. Finally, support for 500+ digit integer representation is an implementation property, not evidence of successful factorization at that scale.

11 Planned Validation for Subsequent Revisions

This work continues that campaign with a deliberately controlled family and extends the controlled BigInt initialization tests to 100 decimal digits. Subsequent revisions should broaden it to random and adversarial strict semiprimes, multiple factor-digit splits, fixed digit-zone schedules, unsuccessful cases, and multiple physical population sizes. Comparisons should include classical Fermat and established multiplier-based methods. Operating-system-specific measurements should be reported separately from the development-host timings in this revision.

12 Conclusion

The Worm formulation recasts a difference-of-squares factorization search as the motion of two integer coordinates controlled by a signed residual. The formal results prove exact residual preservation and score correctness, establish unit advance of the transformed factor coordinate, derive an exact zero-iteration multiplier condition for the simple multiplier case, and bound recurrence work in one failed full Army wave. Its primary engineering attraction is a cheap recurrence and compact independent state. Multipliers create a family of transformed problems in which some factor pairings may be substantially better balanced than the original pq geometry, and the Worm Army evaluates that family through independent, recyclable search engines.

The experimental evidence is deliberately separated into two campaigns. Experiment I

covers 8–32 digits and uses a factor-blind ascending multiplier scheduler: p, q, and q/p are not supplied to the algorithm, and the favorable $q \approx 3p$ geometry does not restrict or focus the generated k population. For full transparency, the present Experiment I dataset itself has factors near $q \approx 3p$; it is therefore a baseline controlled family rather than a random-semiprime sample. Experiment II independently covers 40–100 digits and explicitly declares $q \approx 3p$ as a benchmark construction assumption, deliberately placing the useful multiplier region at small k so that large-integer implementation scaling can be studied without claiming a general blind multiplier search. In that second campaign the favorable geometry makes $k = 3$ an initialization score, so the large-digit results demonstrate BigInt operation and the implementation path under disclosed favorable geometry, not arbitrary 100-digit factorization.

An important architectural property of the Worm Army is the independence of its individual Worm engines. Once a multiplier k, an initialized state, and an iteration budget have been assigned, a Worm evolves without exchanging recurrence state with other Worms. This independence makes the architecture naturally scalable across multicore and many-core CPU systems, where Worm populations can be distributed among processing threads or processors. It is also well suited as a candidate for FPGA realization: multiple Worm engines can in principle be instantiated concurrently with compact local state and a steady-state recurrence dominated by integer comparison, increment, addition, and subtraction. Inter-engine synchronization is principally required for work allocation and successful-factor reporting rather than for recurrence-state communication. This architectural parallelism does not by itself establish algorithmic speedup; achievable acceleration still depends on multiplier coverage, initialization cost, scheduling and memory overhead, available hardware resources, and the probability of assigning useful k regions.

Machine-dependent milliseconds have been removed from the experimental presentation. Timing cost is reported as estimated equivalent CPU clock cycles using the observed 2.596124 GHz reference clock of the AMD EPYC 9V74 virtualized development host. These values are derived from measured elapsed time and are not hardware performance-counter cycle measurements; they provide only a rough basis for cross-machine estimation. The central open questions remain multiplier selection without factor knowledge, performance on random and adversarial factor ratios, unsuccessful-wave cost, and the tradeoff between physical population, iteration budget, and aggregate work. Those questions define the next validation stage and prevent the controlled 100-digit result from being interpreted as a general large-semiprime factorization claim.

References

[1] K. K. H. Cheung, *Number Theory: In Context and Interactive*, "Introduction to Factorization," Algorithm 12.5.9, Gordon College, modern description of Fermat factorization as a difference-of-squares search.

[2] E. W. Weisstein, "Fermat's Factorization Method," *MathWorld: A Wolfram Web Resource*. Includes incremental updating of successive differences by additions.

[3] R. S. Lehman, "Factoring Large Integers," *Mathematics of Computation*, vol. 28, 1974, pp. 637-646, doi:10.1090/S0025-5718-1974-0340163-2.

[4] M. Hittmeir, "A time-space tradeoff for Lehman's deterministic integer factorization method," arXiv:2006.16729, 2020. The historical discussion notes Lawrence's 1895 generalization to multiples kN and Lehman's systematic multiplier selection.

[5] TargetDeLab, *prime_fact2.m*, Algorithm #1 MATLAB research source, initial version 30 July 2017; and *Worm_Army_Log.txt*, C++ implementation history through v1_1_01, 3 October 2026.