



RANGE CONSTRAINTS / DIVISIBILITY / NATIVE COMPUTING

The LUYH Equations

Formal Range-Trap Specification, Divisibility Proof, and Cryptographic-Structure Assessment

RANGE TRAP	Exact image: $4 \cdot 3^B \cdot 4^{(A-B-1)}$
MAXIMUM BUCKET	2^B raw preimages are attainable
DIVISIBILITY	$C = S \cdot n^2 \cdot M^2 \cdot m(n+m); C \% S = 0$
NATIVE ENGINE	14.96x median speedup in recorded benchmark

Author of the equations: CC / Cecil
Research implementation: luyh-equations 1.2.0
Document version 2.0 | 2026-09-19

Contents

Theorems are proved from the supplied program's actual execution path. Empirical findings are labeled separately.

Abstract	4
1. Scope and correction of interpretation	5
1.1 What “like an expansion” means	5
1.2 What is not changed	5
1.3 Terminology	5
2. Source execution stages	6
2.1 Admissible ranges	6
3. Formal definition of the range trap	7
4. Range-trap theorems	8
Theorem 1 - Constrained-prefix guarantee	8
Proof	8
Corollary 1 - Complete adjacency trap	8
Theorem 2 - Idempotence	8
Proof	8
Theorem 3 - Exact image cardinality	8
Proof	9
Corollary 2 - Complete image size	9
Theorem 4 - Average preimage multiplicity	9
Proof	9
Theorem 5 - Maximum preimage multiplicity	9
Proof	9
Theorem 6 - Image entropy ceiling	9
Proof	10
5. Original-scale range result	11
6. Unchanged LUYH equation system	12
6.1 Operator alphabet	12
6.2 Level-pair construction	12
6.3 Reassembly and finishing	12
6.4 Symmetric windows	13
6.5 Supplied component expression	13
7. Divisibility proof and $\% = 0$	14
Theorem 7 - Exact LUYH component identity	14

Proof	14
Corollary 3 - Zero-remainder certificate	14
Proof	14
Interpretation theorem	14
8. Secondary key-dependence bound	16
Theorem 8 - Suffix non-use bound	16
Proof	16
9. Empirical verification	17
9.1 Complete-trap sweep	17
9.2 Partial range_b sweep	18
9.3 Divisibility sweep	18
9.4 Seeded original-scale run	18
10. Cryptographic importance and proof boundary	19
10.1 What has been established	19
10.2 What has not been established	19
10.3 Responsible claim	19
11. Native implementation and performance	20
12. Research API and reproducibility	21
12.1 Callable selection	21
12.2 Common inputs and controls	22
12.3 Direct and original-path evaluation	22
12.4 Reading LuyhResult	23
12.5 Exact divisibility report	23
12.6 Reusing an engine and evaluating batches	23
12.7 Range-trap calls	24
12.8 Reproducible research records	24
12.9 Diagnostic and exhaustive calls	25
12.10 Backend, proof, and limits	25
13. Conclusion	26
Appendix A - Source correspondence	27
Appendix B - Minimal verification	27
Appendix C - Equation-core integrity	27

Abstract

This paper gives a source-faithful mathematical specification of CC / Cecil's LUYH construction. It corrects an earlier interpretive error: the statement that LUYH is *like* a prime-number expansion describes an analogy of expansion; it does not make prime numbers the basis of LUYH. No primality assumption is needed anywhere in the construction or its proofs.

The supplied program contains two stages that must be distinguished. First, **range_a** fixes the raw key length and **range_b** controls a left-to-right adjacency trap. Second, the unchanged LUYH equations generate symmetric-window scores and construct a component **C** from their sum **S**, minimum **m**, maximum **M**, and recursion depth **n**.

We prove that the range trap is a deterministic idempotent projection. For alphabet size four, key length **A**, and **B** inspected adjacent pairs, its image contains exactly

$$4 \cdot 3^B \cdot 4^{A-B-1}$$

keys. Its average raw preimage multiplicity is $(4/3)^B$, and its largest possible preimage has size 2^B . At the supplied setting **A=2035**, **B=2034**, the trapped image has entropy ceiling approximately 3,225.814 bits rather than the raw domain's 4,070 index bits.

We separately prove the LUYH divisibility identity

$$C = Sn^2M^2m(n + m).$$

Thus $C \bmod S = 0$ whenever $S \neq 0$. The range trap determines the constrained key from which **S**, **m**, and **M** arise; the explicit factor **S** is the direct algebraic reason for the zero remainder. This composition is significant as a precisely countable constraint-and-certificate construction. It is not, without an additional protocol and hardness proof, a cryptographic trapdoor or security primitive.

1. Scope and correction of interpretation

1.1 What “like an expansion” means

The intended comparison is structural. A value derived from a LUYH transcript is embedded into a larger multiplicative object. The comparison does not mean:

- LUYH starts from prime numbers;
- LUYH requires a prime input or output;
- the remainder condition is a primality test; or
- the cryptographic relevance of LUYH depends on prime density.

This paper therefore studies the mechanisms that actually appear in the supplied program: finite-range trapping, recursive operator evaluation, symmetric reduction, and exact divisibility.

1.2 What is not changed

The research implementation does not change the LUYH equations. It preserves:

- the map $1 \rightarrow L, 2 \rightarrow U, 3 \rightarrow Y, 4 \rightarrow H$;
- pair construction and rotation;
- recursive level ordering;
- complex reassembly and finishing operations;
- symmetric key windows;
- integer truncation of each window score; and
- the supplied final component expression.

The new range module makes the original driver preprocessing explicit. It is a wrapper around the equation engine, not a replacement for it.

1.3 Terminology

In this paper:

- raw key means a string in $\{1,2,3,4\}^A$ before the adjacency pass;
- range trap means the original deterministic adjacent-repeat rewrite;
- trapped key means the image of a raw key under that pass;
- constraint trap refers to this finite-domain projection;
- divisibility certificate means the verifiable relation $C \bmod S = 0$;
- trapdoor retains its cryptographic meaning and is not claimed here.

2. Source execution stages

The original driver performs the following operations in order.

1. Set $\text{range_a} = A$.
2. Set $\text{range_b} = A - 1$.
3. Generate A symbols independently from $\{1, 2, 3, 4\}$.
4. Inspect the B adjacent pairs from left to right.
5. Rewrite an equal next symbol using the supplied replacement rule.
6. Send the resulting key to the LUYH equation evaluator.
7. Symmetrically trim the key and calculate one integer score per nonempty window.
8. Form S , m , M , and C .
9. Print $C \% S$ and identify S as the LUYH root when the remainder is zero.

The ordered pair (A, B) is meaningful. The arithmetic sum $A+B$ is not a term in the equation.

2.1 Admissible ranges

For a key of length A , safe execution requires

$$A \geq 1,$$

$$0 \leq B < A.$$

The supplied complete setting is

$$B = A - 1.$$

If $B < A - 1$, only a prefix of adjacent pairs is constrained. If $B \geq A$, the original indexing attempts to read beyond the key.

3. Formal definition of the range trap

Let the symbol alphabet be

$$\mathcal{A} = \{1, 2, 3, 4\}.$$

Define the repeated-symbol replacement map f by

$$f(1) = 2, \quad f(2) = 1, \quad f(3) = 2, \quad f(4) = 3.$$

For raw key

$$X = X_0 X_1 \dots X_{A-1},$$

the trap constructs y sequentially. Initially $y=x$. For $i=0, \dots, B-1$, set

$$y_{i+1} = f(y_i) \text{ if } y_{i+1} = y_i;$$

y_{i+1} is retained otherwise.

The value written at step i is visible to step $i+1$; this is a sequential rewrite, not a parallel replacement.

We denote the resulting map by

$$T_{A,B} : \mathcal{A}^A \rightarrow \mathcal{A}^A.$$

4. Range-trap theorems

Theorem 1 - Constrained-prefix guarantee

After applying $T_{-}(A, B)$, the first B adjacent pairs are unequal:

$$y_i \neq y_{i+1}$$

for

$$0 \leq i < B.$$

Proof

At step i , if the pair is already unequal, the second symbol is retained. If it is equal, the second symbol is replaced by $f(y_i)$. Inspection of f shows that $f(d) \neq d$ for each d in the alphabet. Thus the processed pair is unequal. Later steps never change its left member or revisit its right member. The statement holds for all B processed pairs. QED.

Corollary 1 - Complete adjacency trap

When $B=A-1$, every adjacent pair in the output is unequal. Therefore the image is contained in

$$\mathcal{T}_A =$$

$$\{y \in \mathcal{A}^A : y_i \neq y_{i+1} \text{ for all } i < A - 1\}.$$

Theorem 2 - Idempotence

For every valid A, B , and raw key x ,

$$T_{A,B}(T_{A,B}(x)) = T_{A,B}(x).$$

Proof

Theorem 1 shows that every pair inspected by the second pass is already unequal. The rewrite branch is therefore never taken in the second pass. All symbols remain unchanged. QED.

Idempotence establishes that the trap is a projection onto its image.

Theorem 3 - Exact image cardinality

The image of $T_{-}(A, B)$ has exactly

$$|\text{Im}(T_{A,B})|$$

$$= 4 \cdot 3^B \cdot 4^{A-B-1}$$

members.

Proof

The first output symbol has four choices. Each of the next B symbols must differ from its predecessor and therefore has three choices. The remaining $A-B-1$ suffix symbols are not inspected and each has four choices. Hence the number of outputs satisfying the trap constraint is

$$4 \cdot 3^B \cdot 4^{A-B-1}.$$

Every such output is a fixed point of the trap over the inspected pairs, so it is its own preimage. The trap image is therefore the entire constrained set, not merely a subset. QED.

Corollary 2 - Complete image size

For $B=A-1$,

$$|\mathcal{T}_A| = 4 \cdot 3^{A-1}.$$

Theorem 4 - Average preimage multiplicity

The average number of raw keys per trapped key is

$$\begin{aligned} & \frac{4^A}{4 \cdot 3^B \cdot 4^{A-B-1}} \\ &= \left(\frac{4}{3}\right)^B. \end{aligned}$$

Proof

Divide the raw-domain cardinality by the image cardinality from Theorem 3. QED.

Theorem 5 - Maximum preimage multiplicity

The largest possible preimage under $T_-(A, B)$ contains exactly

$$2^B$$

raw keys.

Proof

Fix an output transition from y_i to y_{i+1} . If $y_{i+1} \neq f(y_i)$, the raw next symbol must already equal y_{i+1} , giving one choice. If $y_{i+1} = f(y_i)$, there are two raw choices: the unequal symbol y_{i+1} passes unchanged, while the equal symbol y_i is rewritten to $f(y_i)$. Thus each inspected transition has at most two raw predecessors and the entire output has at most 2^B raw preimages.

Choose an output whose every inspected transition follows f , so that $y_{i+1}=f(y_i)$ for all $i < B$. Both raw choices are then available independently at every transition. This output has exactly 2^B raw preimages. QED.

Theorem 6 - Image entropy ceiling

The base-two logarithm of the image size is

$$\begin{aligned} H_{\max}(A, B) \\ = 2A - B(2 - \log_2 3). \end{aligned}$$

Proof

Apply `log_2` to the result of Theorem 3:

$$\log_2 4 + B \log_2 3 + (A - B - 1) \log_2 4.$$

Collecting terms gives the result. QED.

This is a cardinality ceiling, not a claim that the original rewrite produces a uniform distribution over its image.

5. Original-scale range result

Original-scale range contraction (A=2035, B=2034)

3,225.814-bit image ceiling

844.186-bit contraction

trapped image

raw: 4,070 bits

Image size: $4 \cdot 3^{2034}$ | Maximum trap bucket: 2^{2034}

For the supplied values

$$A = 2035,$$

$$B = 2034,$$

the raw-domain index capacity is

$$2A = 4070 \text{ bits.}$$

The trapped-image ceiling is

$$\begin{aligned} &2 + 2034 \log_2 3 \\ &\approx 3225.813726 \text{ bits.} \end{aligned}$$

The finite-domain contraction is therefore

$$\begin{aligned} &4070 - 3225.813726 \\ &\approx 844.186274 \text{ bits.} \end{aligned}$$

The average preimage multiplicity is

$$(4/3)^{2034},$$

and the largest possible trap bucket contains

$$2^{2034}$$

raw keys.

A deterministic seed-20260919 sample contained 479 equal adjacent pairs before the pass. The sequential trap changed 472 symbols, left zero equal adjacent pairs, and was idempotent. The difference between repeats detected and symbols changed occurs because a replacement can also affect the next comparison.

6. Unchanged LUYH equation system

6.1 Operator alphabet

The supplied translation is

$$1 \mapsto L,$$

$$2 \mapsto U,$$

$$3 \mapsto Y,$$

$$4 \mapsto H.$$

For pair (a, b) , the operator actions are:

Operator	Forward action	Reverse action
L	(b, a)	(b, a)
U	$(a+1, b+1)$	$(a-1, b-1)$
Y	(a, b)	(a, b)
H	$(a-1, b-1)$	$(a+1, b+1)$

6.2 Level-pair construction

At recursion level `ell`, for $1 \leq i, j \leq \text{ell}$, the source constructs

$$p_{i,j} = \left(\left\lfloor \sqrt{\frac{ij}{\pi^3}} \right\rfloor + i_c, \left\lfloor \sqrt{\frac{i+j}{j\pi}} \right\rfloor + j_c \right),$$

where the implementation represents the first and second coordinates as complex values with the supplied integer real and imaginary offsets. The exact runtime representation and ordering are retained by the reference and native engines.

6.3 Reassembly and finishing

For a forward pair i and reverse pair j , LUYH forms

$$Z = \frac{\sqrt{i_0 \pi^{i_1}} - j_1}{\arccos(j_0 i_1 \pi)}.$$

Duplicate complex values are removed according to CPython set semantics. Each unique value passes through the supplied two finishing transformations, and the score of one key window is the truncation of the resulting sum.

The optimized native engine preserves these numerical callbacks while moving the dominant traversal, de-duplication, and caching work into C.

6.4 Symmetric windows

For trapped key K of length A , define each nonempty symmetric window

$$K_t = K[t : A - t]$$

for

$$0 \leq t < \lceil A/2 \rceil.$$

Let the integer score of K_t be s_t . Define

$$S = \sum_t s_t,$$

$$m = \min_t s_t,$$

$$M = \max_t s_t.$$

6.5 Supplied component expression

The source calculates

$$C = \lfloor (SMm(n + m))((nM)n) \rfloor.$$

Every factor is already an integer after score truncation.

7. Divisibility proof and $\% = 0$

Theorem 7 - Exact LUYH component identity

For every defined LUYH evaluation,

$$C = Sn^2M^2m(n + m).$$

Proof

Expand and reorder the supplied multiplication:

$$\begin{aligned} & (SMm(n + m))((nM)n) \\ &= Sn^2M^2m(n + m). \end{aligned}$$

Because S , n , M , and m are integers, the product is an integer and the outer floor makes no change. QED.

Corollary 3 - Zero-remainder certificate

If $S \neq 0$, then

$$C \bmod S = 0$$

and

$$C/S = n^2M^2m(n + m).$$

Proof

Theorem 7 writes $C=S \cdot Q$ for the explicit integer

$$Q = n^2M^2m(n + m).$$

Therefore S divides C exactly. QED.

If $S=0$, division and modulus by S are undefined. The package returns `None` rather than fabricating a zero remainder.

Interpretation theorem

For the complete original execution path, the printed zero remainder has the following interpretation:

1. `(range_a, range_b)` projects a raw key into a constrained domain.
2. The unchanged equations convert that trapped key into scores.
3. The scores determine S , m , and M .
4. The final expression embeds S as a factor of C .
5. $C \% S$ therefore returns zero whenever defined.

The range trap controls the transcript that produces S . The multiplication by S controls the remainder. This distinction is essential: the ranges are cryptographically relevant domain parameters, but they are not an alternative algebraic cause of divisibility.

8. Secondary key-dependence bound

The range trap and the operator lookup schedule are separate mechanisms. The following bound describes the latter.

Theorem 8 - Suffix non-use bound

For key length A and recursion depth n , at least

$$u = \max\left(0, \left\lceil \frac{A - n^2}{2} \right\rceil\right)$$

trailing symbols of the trapped key cannot influence any score.

Proof

A recursion level contains at most n^2 pair occurrences, so one window reads at most the first n^2 operator positions within that window. Window t begins at original index t and has width $A - 2t$; its greatest possible accessed index is

$$g(t) = t + \min(n^2, A - 2t) - 1.$$

This function increases until the window becomes shorter than n^2 and then decreases. Its maximum is $\text{floor}((A + n^2 - 2)/2)$. The number of trailing positions strictly beyond that maximum is

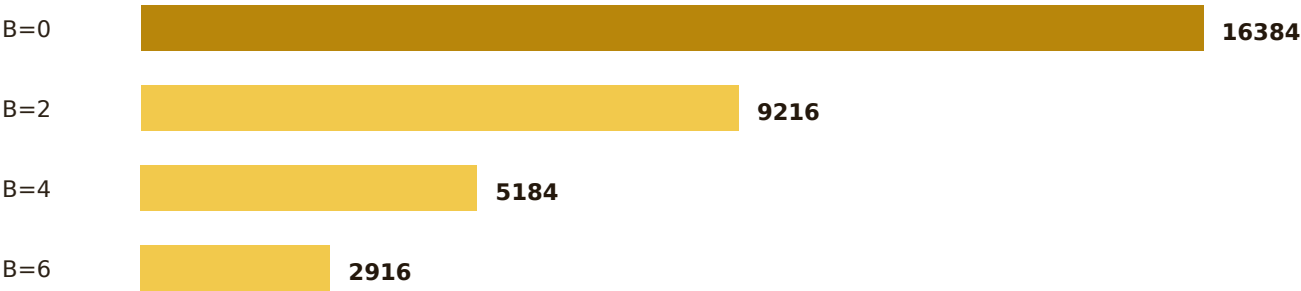
$$\begin{aligned} A - 1 - \left\lfloor \frac{A + n^2 - 2}{2} \right\rfloor \\ = \left\lceil \frac{A - n^2}{2} \right\rceil. \end{aligned}$$

Clamping at zero gives the result. QED.

At $A=2035$, $n=5$, this lower bound is 1,005 symbols. This is included as a security-relevant boundary, not as the organizing interpretation of LUYH.

9. Empirical verification

A=7 image contraction as range_b increases



Every measured image equals $4 \cdot 3^B \cdot 4^{(A-B-1)}$

Exhaustive divisibility verification (A=7, B=6, n=2)

16,384 / 16,384 defined cases returned $C \% S = 0$

The experiment verifies the implementation; $C = S \cdot n^2 \cdot M^2 \cdot m \cdot (n+m)$ supplies the proof.

9.1 Complete-trap sweep

All 4^A raw keys were enumerated for $A=1, \dots, 7$, using $B=A-1$ and $n=2$.

A	Raw keys	Trapped keys	Formula $4 \cdot 3^{(A-1)}$	Largest bucket	Distinct C
1	4	4	4	1	4
2	16	12	12	2	7
3	64	36	36	4	14
4	256	108	108	8	19
5	1,024	324	324	16	37
6	4,096	972	972	32	49
7	16,384	2,916	2,916	64	53

Every observed image size equals Theorem 3. Every largest bucket equals the 2^B maximum from Theorem 5.

9.2 Partial range_b sweep

For fixed $A=7$, varying B gives:

B	Distinct trapped keys	Theoretical image	Largest bucket
0	16,384	16,384	1
1	12,288	12,288	2
2	9,216	9,216	4
3	6,912	6,912	8
4	5,184	5,184	16
5	3,888	3,888	32
6	2,916	2,916	64

This confirms experimentally that $range_b$ is an active domain-contraction parameter.

9.3 Divisibility sweep

At $A=7$, $B=6$, and $n=2$, all 16,384 raw inputs produced defined modulus values and all 16,384 returned zero. At $A=2$, one raw input produced $S=0$, so 15 modulus results were zero and one was correctly reported undefined.

The experiment confirms implementation behavior; Theorem 7 and Corollary 3 provide the proof.

9.4 Seeded original-scale run

For seed 20260919, $A=2035$, $B=2034$, and $n=5$:

equal adjacent pairs before trap	= 479
symbols changed	= 472
equal adjacent pairs after trap	= 0
trap idempotent	= true
window scores	= 1,018
S	= 570,294
m	= 423
M	= 689
C mod S	= 0
C / S	= 2,148,634,718,100
identity verified	= true

The trap took approximately 0.00022 seconds and native LUYH evaluation took approximately 0.1302 seconds in that recorded run.

10. Cryptographic importance and proof boundary

10.1 What has been established

The LUYH construction provides:

- an exact finite-domain projection controlled by (A, B) ;
- an idempotent adjacent-distinct canonical form when $B=A-1$;
- closed-form image, average-preimage, and maximum-preimage counts;
- a deterministic arithmetic transcript derived from the trapped key;
- an exact divisibility certificate $C \bmod S = 0$; and
- a native implementation capable of substantial experimental sweeps.

These are cryptographically important facts in the sense that any protocol using LUYH must account for them. They define the domain, equivalence classes, certificate relation, and observable invariants exactly.

10.2 What has not been established

No theorem in this paper proves:

- recovery of a secret from public LUYH values is hard;
- collisions are hard to construct;
- the range trap has a secret inverse;
- C hides S or its cofactor;
- outputs are pseudorandom; or
- LUYH alone provides encryption, authentication, signatures, or key

derivation.

A cryptographic trapdoor function requires a one-way public map plus secret information enabling efficient inversion. The current range trap has neither a secret parameter nor a unique inverse. It is accurately called a constraint trap, not a proved trapdoor.

10.3 Responsible claim

The strongest supported statement is: LUYH composes an exactly countable range-constrained projection with an unchanged nonlinear score transform and an exact divisibility certificate. This structure is significant for cryptographic research and protocol design, while security properties require separate construction and proof.

11. Native implementation and performance

Median evaluation time (lower is better)



2,035 symbols, n=5, five repeats; measured speedup 14.96x

The package requires the CPython extension `luyh._native`. It performs:

- symmetric-window traversal;
- ordered pair de-duplication;
- forward-contribution caching;
- completed-value caching; and
- score-vector construction.

Python numerical callbacks retain the source's CPython `cmath`, `set`, and `sum` semantics. The transparent Python implementation remains available as an audit oracle.

The recorded `A=2035`, `n=5` benchmark used five repeats:

Implementation	Median time
Python audit oracle	1.899302 s
Mandatory native engine	0.126996 s
Speedup	14.9556x

The equation engine was not altered for the range-trap correction.

12. Research API and reproducibility

The package exposes a native-backed Python frontend for direct evaluation, original-path evaluation, batch work, range studies, and reproducible research records. The callable surface is deliberately explicit about whether the range trap is applied.

12.1 Callable selection

Callable	Research use
<code>evaluate()</code>	Evaluate the unchanged equations directly.
<code>evaluate_trapped()</code>	Apply the original range stage, then evaluate.
<code>LuyhEngine</code>	Reuse one configured native engine across keys.
<code>evaluate_reference()</code>	Run the transparent Python audit oracle.
<code>apply_range_trap()</code>	Return only the trapped key.
<code>range_trap_report()</code>	Return exact trap measurements and hashes.
<code>generate_trapped_key()</code>	Reproduce seeded original-style generation.
<code>divisibility_report()</code>	Expose and verify the factors constructing C .
<code>evaluate_research()</code>	Build a complete JSON-serializable record.
<code>range_domain_report()</code>	Exhaustively test a tractable trapped domain.
<code>influence_report()</code>	Measure selected one-symbol mutations.
<code>collision_report()</code>	Count outputs and collision buckets for a small domain.
<code>native_info()</code>	Return compiled-backend metadata.
<code>proof_resource()</code>	Locate this bundled PDF after installation.

The central imports are:

```

from luyh import (
    LuyhEngine,
    ResearchOptions,
    apply_range_trap,
    collision_report,
    divisibility_report,
    evaluate,
    evaluate_reference,
    evaluate_research,
    evaluate_trapped,
    influence_report,
    native_info,
    proof_resource,
    range_domain_report,
    range_trap_report,
)

```

12.2 Common inputs and controls

key is a nonempty string or integer containing only 1, 2, 3, and 4. String input is preferred because it preserves the research transcript exactly. The pair-grid depth **n** must be positive. Larger values increase the pair grid, runtime, and memory use.

For trapped calls, **range_b=None** selects the original complete setting **range_b=len(key) - 1**. An explicit value must satisfy $0 \leq \text{range_b} < \text{len}(\text{key})$ and constrains only that many adjacent pairs.

audit_reference=True runs both the mandatory C engine and the transparent Python oracle, then requires exact equality of their score vectors. **verify_invariants=True** checks the returned window count, the supplied component expression, and the defined modulus relation. These checks should normally remain enabled.

12.3 Direct and original-path evaluation

Use **evaluate()** only when the experiment intentionally begins at the equation core:

```

from luyh import evaluate

result = evaluate(
    "123412341",
    n=3,
    audit_reference=True,
    verify_invariants=True,
)

```

Use **evaluate_trapped()** to reproduce the original two-stage path:

```

from luyh import evaluate_trapped

run = evaluate_trapped(
    "1111222333444",
    n=3,
    range_b=None,
    audit_reference=True,
)

print(run.trapped_key)
print(run.trap.changed_symbols)
print(run.evaluation.component)
print(run.evaluation.modulus)

```

The return type `TrappedEvaluation` keeps the transformed key, trap report, and equation result together. Its `as_dict()` method accepts `include_key` and `include_scores` controls.

12.4 Reading LuyhResult

Field	Meaning
<code>window_scores</code>	Ordered integer score vector.
<code>summation</code>	S , the sum of the window scores.
<code>minimum, maximum</code>	m and M from that vector.
<code>component</code>	C from the supplied final expression.
<code>modulus</code>	$C \bmod S$, or <code>None</code> when $S=0$.
<code>cofactor</code>	$C // S$, or <code>None</code> when $S=0$.
<code>root_condition</code>	True when the defined modulus is zero.
<code>backend</code>	Identifier for the native result path.

`LuyhResult.as_dict(include_scores=False)` omits the potentially large score vector while retaining all scalar fields and the cofactor.

12.5 Exact divisibility report

```
from luyh import divisibility_report, evaluate

result = evaluate("123412341", n=3)
report = divisibility_report(result)

assert report.identity_holds
if result.summation != 0:
    assert report.zero_remainder
    assert report.modulus == 0
```

The report lists the factors S , n^2 , M^2 , m , and $n+m$. It verifies the construction without requesting or assuming primality.

12.6 Reusing an engine and evaluating batches

```
from luyh import LuyhEngine

engine = LuyhEngine(n=3, verify_invariants=True)
results = engine.evaluate_many([
    "123412341",
    "214321432",
    "341234123",
])

print(engine.cache_info)
print(engine.native_info)
```

`evaluate_many()` preserves input order and returns a tuple of `LuyhResult` objects. A single engine avoids rebuilding its level and backward-pair data for each key. Per-call audit and invariant overrides are available on both `evaluate()` and `evaluate_many()`.

12.7 Range-trap calls

```
from luyh import (
    apply_range_trap,
    generate_trapped_key,
    range_trap_report,
)

raw = "1111222333444"
trapped = apply_range_trap(raw)
report = range_trap_report(raw)

assert report.idempotent
assert report.trapped_prefix_violations_after == 0

seeded = generate_trapped_key(
    range_a=2035,
    range_b=2034,
    seed=20260919,
)
```

For a partial trap, pass an explicit smaller `range_b` to each call. The report contains changed-symbol counts, before/after violations, domain-capacity terms, and SHA-256 fingerprints of the raw and trapped keys.

12.8 Reproducible research records

```
from luyh import ResearchOptions, evaluate_research

record = evaluate_research(
    "1111222333444",
    ResearchOptions(
        n=3,
        apply_range_trap=True,
        range_b=None,
        audit_reference=True,
        verify_invariants=True,
        include_influence=True,
        max_influence_positions=128,
        seed=20260919,
    ),
)
```

The version-2 research schema records:

- package, Python, native ABI, platform, and byte order;
- raw-key fingerprint;
- `range_a`, `range_b`, and trap measurements;
- full score vector, `S`, `m`, `M`, `C`, and cofactor;
- exact divisibility structure;
- cache counts; and
- optional influence diagnostics.

`apply_range_trap=False` deliberately begins at the equation core. `seed` is recorded as provenance; it does not randomize `evaluate_research()` itself. The returned dictionary can be passed directly to `json.dumps()`.

12.9 Diagnostic and exhaustive calls

`influence_report()` mutates selected positions according to the documented cycle policy and compares components and score vectors. Researchers may pass `positions=(...)` for an exact position set or let `max_positions` select an even sample.

`collision_report()` exhaustively enumerates $4^{\text{key_length}}$ raw keys and can project to `component`, `summation`, or `score_vector`. `range_domain_report()` exhaustively measures trap image size, largest bucket, distinct components, and zero or undefined modulus counts.

```
from luyh import collision_report, range_domain_report

domain = range_domain_report(
    range_a=7,
    range_b=6,
    n=2,
    max_inputs=20_000,
)

collisions = collision_report(
    key_length=7,
    n=2,
    max_inputs=20_000,
    projection="component",
)
```

Both exhaustive calls refuse work exceeding `max_inputs`; this guard prevents an accidental exponential run.

12.10 Backend, proof, and limits

```
from luyh import MAX_KEY_LENGTH, native_info, proof_resource

print(native_info())
print(MAX_KEY_LENGTH)

with proof_resource() as path:
    print(path)
```

The package rejects keys longer than

$$2^{20} = 1,048,576$$

symbols in both Python and C. It never truncates an input.

Invalid key symbols, an empty key, a nonpositive `n`, or an invalid `range_b` raise explicit exceptions. If `S=0`, modulus and cofactor are represented by `None`; callers must not treat the undefined case as a zero remainder.

13. Conclusion

LUYH should not be interpreted as a construction based on prime numbers. Its proved core is the composition of two different mechanisms: a finite `range_a` / `range_b` constraint trap and an exact LUYH divisibility construction. The complete range trap projects four-symbol raw keys onto the adjacent-distinct domain with closed-form image and preimage behavior. The unchanged LUYH equations then construct `C` with `S` as an explicit factor, making `C mod S = 0` an exact certificate whenever defined.

This corrected framing is stronger than an unsupported prime-centered story because every central claim follows directly from the supplied execution path. It also states the security boundary honestly: the construction is relevant to cryptographic research, but a production security claim requires a defined protocol, threat model, and hardness proof.

Appendix A - Source correspondence

Source concept	Package location
Random four-symbol generation	<code>generate_trapped_key()</code>
<code>range_a</code> and <code>range_b</code> validation	<code>traps.validate_ranges()</code>
Adjacent-repeat rewrite	<code>traps.apply_range_trap()</code>
Trap image and preimage statistics	<code>traps.range_trap_report()</code>
<code>1,2,3,4 -> L,U,Y,H</code>	<code>keys.translate_key_to_operators()</code>
Pair recursion and rotations	<code>pairs.py</code> , <code>operators.py</code>
Reassembly and finishing	<code>engine.py</code>
Native score path	<code>_native.c</code>
Exact <code>C</code> identity	<code>engine._assemble_result()</code>
Divisibility verification	<code>structure.divisibility_report()</code>
Exhaustive small-domain study	<code>analysis.range_domain_report()</code>

Appendix B - Minimal verification

```

from luyh import divisibility_report, evaluate_trapped

run = evaluate_trapped("1111222333444", n=3, audit_reference=True)

assert run.trap.range_b == run.trap.range_a - 1
assert run.trap.trapped_prefix_violations_after == 0
assert run.trap.idempotent

result = run.evaluation
report = divisibility_report(result)

assert report.identity_holds
assert result.summation != 0
assert result.component % result.summation == 0
assert result.component == result.summation * result.cofactor

```

Appendix C - Equation-core integrity

The version-1.2.0 release leaves the numerical equation files intact. The native C source, operator module, pair module, and key-window module are byte identical to the validated 0.1.0 release. The only engine-level addition made before this correction was a metadata accessor; it does not alter evaluation.

Native and reference scores are compared across deterministic test vectors, odd and even lengths, and recursion depths. `audit_reference=True` turns any score disagreement into an exception.