ENI6MA ALGO1 Proof Variants: Formal Model and Proofs

A. ENI6MA ALGO1 proof variants: formal model and proofs

1) Overview: **Scope.** This text formalizes and explains (with equations and technical narrative) the principal ENI6MA proof mechanism, focusing on the ALGO1 family and its challenge/response variants.

Framing. ENI6MA ALGO1 is **not** a SNARK/STARK/R1CS "circuit proof" system. It is best modeled as a multi-round identification protocol based on position-wise bucket membership: the verifier accepts when the prover selects, in each round, a zone whose witness contains the corresponding secret character for some configured secret.

ALGO1 proves knowledge of a secret string by forcing a prover to make a sequence of correct round-by-round choices among a small finite set of options. The resulting security arguments are combinational (counting-based) and become practically meaningful when paired with a time-bounded challenge envelope and one-time-use consumption policy.

Two witness-generation families are treated explicitly: a foliation-based construction that partitions rotated alphabets into 6 disjoint slices, and a sampling-based construction that draws fixed-length substrings with wraparound and then assigns fixed-width blocks to zones. A compact response format and a nonce/matrix challenge construction are treated as auxiliary layers. Where procedural clarity is useful, the constructions are specified as short pseudocode blocks rather than as implementation-specific references.

The development proceeds by first fixing notation and the acceptance predicate, then specifying the witness constructions as deterministic functions of entropy and time anchors. Completeness follows directly from the predicate: an entity that knows an admissible S can always select zones whose witnesses contain s_i in each round. Soundness is treated in the natural sense for this mechanism, namely as a blind-guess forging bound: per round, the number of winning zones is t_i, yielding an overall bound of the form $\prod_i (t_i/6)$; the cleanest uniqueness regime reduces to 6^{-L}. A separate distinction is maintained between transcript validity and knowledge: reconstructability and well-formedness checks can be useful integrity filters, but they do not establish knowledge unless coupled to the membership predicate against $\mathcal{S}$. The online threat model is then stated explicitly: TTL expiry, one-time consumption, and audit grounding constrain the attempt budget, converting per-attempt bounds into system-level risk budgets under rate limits and policy controls. This separation also clarifies how validation outcomes should be interpreted and reported.

---

2) 1) High-level idea (plain English): Think of a secret string $S = s_1 s_2 \dots s_L$ embedded in the verifier. In each round i, the system generates a public table with **6 zones**, each zone showing a **witness string** of characters. The prover "responds" by choosing a zone (or an equivalent bearing direction) each round.

The verifier accepts if, for some configured secret S, the prover's chosen witness in round i contains the secret character s_i:

- If you **know** S, you can pick a correct zone each round.
- If you **don't know** S, you are reduced to guessing among zones; success probability falls exponentially with L (with caveats depending on variant).

This is the core proof principle: consistent, per-position membership evidence.

a) Implications and positioning: The key implication of this framing is that ENI6MA's "proof" is best understood as a multi-round identification check rather than a general-purpose circuit proof: the verifier is not validating an arbitrary statement, it is validating repeated, position-wise membership evidence tied to a configured secret. This can be superior to many ad-hoc authentication puzzles because it yields a clean, quantifiable success probability for blind guessing: if each round has exactly one correct zone, an attacker without the secret must succeed through a sequence in a space of size 6^L. It is also operationally attractive compared to heavyweight proof systems (SNARK/STARK-style) when the goal is deterministic reconstruction and auditability: ALGO1's verifier is simple (string membership) and the public "hyperplane" can be regenerated from compact parameters, which reduces the amount of opaque cryptographic material that must be transported, parsed, and stored. Compared to a classic "static password" check, the multi-round structure provides a natural place to attach per-challenge lifecycle rules (nonce burn, TTL, audit logs) so that a single captured response is not indefinitely reusable. The trade-off is equally important: because acceptance is defined by membership of the true secret characters, transcripts inevitably correlate with the secret, so this protocol should not be marketed or deployed as zero-knowledge authentication. In short, ALGO1's strength is an explicit, composable combinational barrier plus deterministic verifiability; it is "superior" primarily in deployments that prize explainability, reproducibility, and tight integration with

time-bounded challenge envelopes over transcript privacy and traditional hardness-assumption reductions.

---

3) 2) Glossary of parameters and symbols: This section defines all parameters used in equations and pseudocode.

You will reason about the protocol almost entirely through a small set of symbols. Keeping them explicit prevents category errors (e.g., confusing "structural validity of a response" with "knowledge of a secret"), and it lets you quantify brute-force cost directly in terms of L and n. It also forces you to separate *reconstruction parameters* (used to deterministically regenerate $W_{i,z}$ from a compact challenge envelope) from *security parameters* (which drive the size of the response space and the number of winning choices per round).

a) Implications and positioning: The glossary is more than notation: it is the protocol's public contract. The immediate implication is that ENI6MA's security properties become testable rather than rhetorical—once $n = 6$, L, $\mathcal{S}$, $W_{i,z}$, τ, E, and the bearing mapping are explicit, you can compute concrete quantities like 6^L and you can audit whether a given verifier actually enforces the acceptance predicate or only checks reconstructability. This explicitness is often superior to "black-box" authentication designs where critical parameters are implicit ("some randomness," "some nonce," "some secret") and implementations silently diverge due to encoding or time-unit mistakes. In ENI6MA, correctly distinguishing τ (time envelope), E (pool-selected entropy), and π (embedded prime constant used in nonce derivation) matters because different layers depend on different variables, and confusing them changes what is and is not bound into a response. The glossary also clarifies threat modeling: $\mathcal{S}$ is embedded/configured (so secrecy may be an operational assumption), while the hyperplanes and bearings are part of the transcript (so privacy is limited). Finally, being explicit about $W_{i,z}$ as a *string* and acceptance as a *membership test* prevents category errors with ZK systems: ALGO1 does not prove a generic relation over commitments; it proves repeated membership across rounds. That clarity can be "superior" in real deployments because it reduces the chance of security-by-misunderstanding, which is a common cause of failures even in systems that use strong primitives. It also speeds cross-team review by standardizing terminology and invariants.

b) 2.1 Sets, strings, and indices:

- $\mathcal{S}$: the configured set of acceptable secrets (one or more strings) provided by configuration.
- $S \in \mathcal{S}$: a specific secret string.
- $S = s_1 s_2 \dots s_L$: secret characters by position, with $L = |S|$.
- i: round index, typically $i \in \{1, \dots, L\}$ for ALGO1 proof checking.
- n: number of zones/hyperplanes; in the constructions below, $n = 6$.
- z: a zone index, $z \in \{0, 1, 2, 3, 4, 5\}$.

c) 2.2 Alphabets ("rings"):

- M: number of alphabets (rings). A common instantiation uses $M = 3$ rings: uppercase, lowercase, numeric.
- $\mathcal{A}_j$: the j-th alphabet, an ordered list of characters.
- $|\mathcal{A}_j|$: size of alphabet j.

Typical defaults: - $|\mathcal{A}_0| = 26$ (A–Z) - $|\mathcal{A}_1| = 26$ (a–z) - $|\mathcal{A}_2| = 10$ (0–9)

d) 2.3 Entropy and timestamp:

- τ: a session timestamp (microseconds since epoch). Used in multiple places: entropy pool indexing, nonce payload, and regeneration.
- E: session entropy, modeled as a 512-bit integer selected from a fixed entropy pool.
- π: a 512-bit prime-like constant used in nonce/matrix derivation.

e) 2.4 Witnesses, bearings, transcripts:

- $W_{i,z}$: the witness string shown/derived for round i and zone z.
- Prover response can be represented as either:
  - zone indices $(z_1, \dots, z_L)$, or
  - bearings $(b_1, \dots, b_L)$ with $b_i \in \{U, D, L, R, F, B\}$ mapped to zones.

Canonical bearing→zone mapping: - $L \mapsto 0$, $U \mapsto 1$, $R \mapsto 2$, $B \mapsto 3$, $D \mapsto 4$, $F \mapsto 5$.

f) 2.5 Acceptance predicate: For a transcript $(z_1, \dots, z_L)$, the core acceptance predicate is:

$$\text{Accept} \iff \exists S \in \mathcal{S} \text{ s.t. } \forall i : \ s_i \in W_{i,z_i}.$$

The knowledge-checking verifier evaluates the predicate above by reconstructing (or observing) the witness strings and testing per-position membership. A structural verifier may instead only check that a response is well-formed and reconstructable under the public challenge envelope; that weaker check does not, by itself, establish knowledge of S.

---

4) 3) Variant map (witness construction + verification predicate): Here, a "variant" denotes a specific witness construction together with a specific verification predicate.

a) Implications and positioning: The variant map has direct security consequences because "verification" is not a single predicate across all deployments. In particular, it is common to distinguish (i) a knowledge-checking validation mode that reconstructs witnesses and enforces membership against $\mathcal{S}$, and (ii) a structural validation mode that checks only reconstructability and well-formedness under the challenge envelope. Treating these as interchangeable silently downgrades the assurance level: reconstructability alone does not imply knowledge of S. Making the variants explicit also clarifies which parameters are intended to bind a transcript (e.g., τ and envelope policy) and where improvements would strengthen the bound (e.g., binding additional challenge structure into witness generation, or eliminating duplication that increases t_i). This separation is operationally useful: lightweight responders can emit compact paths, while strict verifiers can still enforce a knowledge predicate under expiry and one-time-use policies.

b) 3.1 ALGO1-Plain (Variant A):

- Witness construction: rotate full alphabets then partition each alphabet into $n = 6$ contiguous slices via quotient/remainder.
- Verification: requires L collected witnesses and checks per-position membership against some configured secret.

c) 3.2 ALGO1-Nonce (Variant B):

- Witness construction: per round, derive offsets then sample fixed lengths (30 uppercase, 30 lowercase, 12 numeric) with wraparound; split into 6 fixed blocks (5/5/2).
- Typically driven by a challenge nonce that carries τ and a matrix row count.
- Verification: same membership predicate, but witnesses can be reconstructed from τ and bearings.

d) 3.3 Minimal payload verification flows (knowledge-checking vs structural): Two verifier roles should be treated as distinct objects. A knowledge-checking verifier reconstructs witnesses and enforces the secret-membership predicate (proof-of-knowledge style). A structural verifier reconstructs witnesses (and any nonce structure) and checks only well-formedness and reconstructability; by construction, that check can succeed even when the prover does not know S.

e) 3.4 Nonce/matrix subcircuit (Variant N): Separately, ENI6MA defines a nonce/matrix "circuit" that: - derives per-row values and hashes from entropy pool entries, π, and τ, - and supports validation by brute-force recovery of entropy indices from row hashes.

---

5) 4) Shared core primitive: entropy-driven rotation offsets: Both ALGO1 variants compute a deterministic offset per round and alphabet.

a) Discussion: The offsets are best viewed as a *reconstruction primitive*. They let you regenerate the same witness tables from compact inputs (notably τ and access to the same entropy source) without transmitting full tables. From a proof perspective, the important point is what offsets do *not* buy you: because they reduce modulo small alphabet sizes, they do not create factorial-scale permutation hardness. The exponential barrier comes from repeating a small-choice test across L rounds (the 6^L response space), not from any single-round offset complexity.

b) Implications and positioning: Deterministic offsets are the mechanism that turns ENI6MA's challenges into reproducible objects: a verifier that shares the same embedded pool and receives (or regenerates) the same τ can recompute the same offsets and therefore the same hyperplanes without transporting bulky witness material. This is operationally superior to schemes that require shipping a full challenge table or large commitments because it minimizes parsing complexity and reduces the amount of sensitive, easily logged data moving through systems. Determinism also improves auditability: given a fixed τ, the offset computation is pure and repeatable, so discrepancies can be attributed to implementation bugs rather than randomness. The security trade-off is that offsets are computed modulo small alphabet sizes (26/26/10 under defaults), so the set of possible per-round rotations is small and should not be confused with a factorial permutation space; the protocol's brute-force resistance comes primarily from multi-round composition (6^L transcript space) and enforcement controls (TTL/burn), not from single-round rotation entropy. Compared to "shuffle with RNG" designs, cyclic-rotation-style determinism is less fragile across platforms and languages, which can be a practical advantage when independent verifiers must agree exactly. This also improves reconciliation: distributed verifiers can agree on challenges bit-for-bit, avoiding nondeterministic drift that would otherwise create false rejects and costly operational debugging. In short, offsets are a reliability and reconstruction primitive; their superiority is in reproducibility and minimal state, not in providing standalone cryptographic hardness. This is particularly valuable in regulated, multi-verifier deployments and cross-language clients.

c) 4.1 Offset equation: Let E be the session entropy (a `U512` integer). For round i and ring j:

- define a modifier:

$$m(i, j) := 3i + j$$

- define effective entropy:

$$E_{i,j} := E + m(i, j) \pmod{2^{512}}$$

- offset is:

$$o_{i,j} := E_{i,j} \bmod |\mathcal{A}_j|$$

Interpretation. $o_{i,j}$ chooses where in alphabet $\mathcal{A}_j$ we start rotating/sampling for round i.

d) 4.2 Pseudocode (depth 3):

```
PROCEDURE ComputeAlphabetOffset(E, i, j, alphabet_size):
  modifier <- 3*i + j
  E_eff <- (E + modifier) mod 2^512
  offset <- E_eff mod alphabet_size
  RETURN offset
```

---

6) 5) Variant A: ALGO1-Plain ("interactive_proof"):

a) Discussion: Variant A gives you the cleanest mathematical object: for each ring alphabet, a cyclic rotation followed by a true partition into $n = 6$ disjoint slices. That structure is exactly what makes the simplest soundness argument possible: for a fixed round and ring, each character belongs to exactly one zone slice. When secret characters are drawn from disjoint rings (e.g., digits live only in the numeric ring), you typically get a unique winning zone per round for that character, and blind guessing becomes a 6^{-L} process. The cost is conceptual rather than technical: since acceptance is defined by membership of real secret characters, transcripts are correlated with the secret and you should not expect zero-knowledge-style privacy.

b) Implications and positioning: Variant A has the cleanest combinational structure: for each ring, it performs a true partition (foliation) of a rotated alphabet into $n = 6$ disjoint slices. The implication is that, for many secrets (especially those whose characters live in disjoint alphabets like uppercase

vs lowercase vs digits), there is exactly one "correct zone" per round for the relevant secret character. That yields the sharpest blind-guess bound $(1/6)^L$ and makes the protocol's brute-force story easy to audit: uniqueness is structural, not probabilistic. This can be superior to many interactive "challenge puzzles" where multiple answers might accidentally satisfy the check or where the challenge generation is underspecified; here, the witness generation is deterministic and the acceptance predicate is explicit membership. Variant A is also superior as a reference implementation because it is conceptually minimal: it demonstrates the core ALGO1 idea without introducing a ledger, nonce payload formats, or responder-side minimization, which makes it a good baseline for reasoning about correctness. The trade-offs are equally important and should be treated as design constraints: because the verifier's check is literally "does the witness contain this secret character," the transcript (zone choices and the displayed hyperplanes) is correlated with the secret and is not zero-knowledge; repeated transcripts can leak partial information about the secret's per-position bucket. Therefore, Variant A is strongest when used as a local demonstrator or as a building block behind stronger envelope controls (time-bounded challenges, burn-after-use) rather than as a privacy-preserving authentication protocol in adversarial observation settings.

c) 5.1 Witness construction (formal specification): For each round i and ring j:

1) Rotate alphabet $\mathcal{A}_j$ by $o_{i,j}$:

$$\mathcal{A}'_{i,j}[k] := \mathcal{A}_j[(k + o_{i,j}) \bmod |\mathcal{A}_j|]$$

2) Partition $\mathcal{A}'_{i,j}$ into $n = 6$ slices using quotient/remainder:

Let:

$$q_j := \left\lfloor \frac{|\mathcal{A}_j|}{n} \right\rfloor, \qquad r_j := |\mathcal{A}_j| \bmod n$$

Slice lengths:

$$\ell_j(z) := q_j + \mathbf{1}[z < r_j]$$

Start index:

$$\text{start}_j(0) := 0, \qquad \text{start}_j(z+1) := \text{start}_j(z) + \ell_j(z)$$

Slice:

$$\text{Slice}_{i,j}(z) := \mathcal{A}'_{i,j}[\text{start}_j(z) : \text{start}_j(z) + \ell_j(z)]$$

3) Witness for zone z is concatenation across rings:

$$W_{i,z} := \mathsf{concat}\big(\text{Slice}_{i,j}(z)\big)_{j=0}^{2}$$

Note on variable lengths. Because $|\mathcal{A}_j|$ may not be divisible by 6, some zones contain one extra character for that ring. With defaults: - For uppercase/lowercase: $26 = 6 \cdot 4 + 2$ so zones $0, 1$ have 5 letters; zones 2..5 have 4 letters. - For digits: $10 = 6 \cdot 1 + 4$ so zones 0..3 have 2 digits; zones $4, 5$ have 1 digit.

So $|W_{i,z}|$ depends on z.

d) 5.2 Pseudocode: witness generation (depth 3):

```
PROCEDURE HyperplaneV1(E, alphabets[0..2], n=6, round i):
  FOR z IN 0..5:
    witness <- ""
    FOR j IN 0..2:
      witness <- witness + ZoneSliceV1(E, alphabets[j], n, i, j, z)
    table[z] <- witness
  RETURN table

PROCEDURE ZoneSliceV1(E, alphabet A, n, round i, ring j, zone z):
  offset <- ComputeAlphabetOffset(E, i, j, |A|)
  A_rot <- RotateLeft(A, offset)

  q <- floor(|A| / n)
  r <- |A| mod n

  start <- 0
  FOR t IN 0..(n-1):
    slice_len <- q + (t < r ? 1 : 0)
    IF t == z:
      RETURN A_rot[start : start + slice_len]
    start <- start + slice_len
  RETURN ""  // unreachable if z in range
```

e) 5.3 Statement proved (what the verifier checks): Given a transcript $(z_1, \ldots, z_L)$, verifier accepts iff:

$$\exists S = s_1 \ldots s_L \in \mathcal{S} \text{ s.t. } \forall i : \ s_i \in W_{i,z_i}.$$

f) 5.4 Completeness proof (Variant A): **Theorem (Completeness, Variant A).** If the prover knows a secret $S \in \mathcal{S}$ and, for each round i, selects a zone z_i such that $s_i \in W_{i,z_i}$, then verification accepts.

Proof. The verifier explicitly checks that each witness string contains the corresponding secret character for some configured secret of matching length. If the prover's selected witness contains s_i for each i, the verifier's checks succeed and it accepts. □

g) 5.5 Soundness as a knowledge-error / guessing bound (Variant A): Variant A implicitly aims for a "unique correct zone" per round (for a fixed secret character), which yields a $(1/6)^L$ style bound.

Lemma (Per-ring partition uniqueness). For any fixed round i and ring j, the slices $\text{Slice}_{i,j}(0), \ldots, \text{Slice}_{i,j}(5)$ form a partition of $\mathcal{A}_j$ (after rotation). Each character of $\mathcal{A}_j$ appears in exactly one zone slice.

Proof. Rotation is a permutation; quotient/remainder slicing forms disjoint contiguous blocks whose lengths sum to $|\mathcal{A}_j|$. □

If each secret character s_i belongs to exactly one of the alphabets (e.g., a digit is only in the numeric ring), then exactly one zone witness contains s_i for that round. A secret-ignorant prover who guesses zones has:

$$\Pr[\text{pass round } i] \leq \frac{1}{6}$$

and hence:

$$\Pr[\mathsf{Accept}] \leq \left(\frac{1}{6}\right)^L.$$

Caveat. This is a combinational bound, not a reduction to a hardness assumption like discrete log.

h) 5.6 Why Variant A is not zero-knowledge (plain English): The transcript reveals which of 6 slices contains each secret character position. Over time, this leaks structure about the secret. There is no simulator that can produce an indistinguishable transcript without implicitly matching the membership predicate, so standard ZK claims do not apply.

---

7) 6) Variant B: ALGO1-Nonce ("nonce_interactive_proof"): Variant B introduces a challenge nonce carrying τ and a matrix with a number of rows. The protocol uses τ to set the session entropy selection and then generates per-round witnesses.

a) Discussion: Variant B is the form you analyze when you care about a compact challenge/response interface. A challenge envelope carries a timestamp τ and enough structure to determine the number of rounds, while the prover returns only a short sequence of bearings (zone choices). The verifier regenerates the witnesses deterministically from τ and checks the same membership predicate as before. The sampling rule is deliberately fixed-length (30/30/12 with wraparound), which makes reconstruction simple but introduces an important probabilistic/combinational effect: wraparound can duplicate characters across zones, so some rounds admit multiple winning zones for a given secret character. In soundness calculations, this shows up as a per-round multiplicity t_i and moves the bound from a clean $(1/6)^L$ to $\prod_i (t_i/6)$, with a $(1/3)^L$ worst case when $t_i = 2$ throughout.

b) Implications and positioning: Variant B's primary implication is deployability: it turns ALGO1 into a challenge/response shape where a verifier can generate a challenge envelope, a responder can return a compact bearing sequence, and a verifier can reconstruct witnesses deterministically to validate the membership predicate. This architecture is often superior to designs that transmit full witness content because it reduces what must cross trust boundaries and what might be accidentally logged; it also allows responder implementations to remain lightweight while the verifier retains privileged reconstruction state (embedded pool access). Variant B also composes naturally with policy enforcement: because challenges carry τ (and typically a nonce UUID in ledgered flows), you can enforce "burn-after-use," "expires after TTL," and "tau must match the reserved envelope," turning brute force into an expensive online process. However, Variant B's fixed sampling (30/30/12 with wraparound) has a measurable cryptographic implication: it can duplicate characters across zones, which increases the number of winning choices t_i in some rounds and weakens the blind-guess bound toward $(1/3)^L$ in the worst case. This doesn't negate the protocol's utility; it simply shifts where security comes from: the round count L and the ledger's attempt-limiting controls dominate. In positioning terms, Variant B is superior as a system component when you want deterministic reconstruction, minimal payloads, and enforceable one-time challenges; it is not superior if your benchmark is transcript privacy or strong single-shot hardness without relying on TTL/burn policy.

c) 6.1 Inputs and outputs:

- **Input challenge:** a `MinimalNonce` (possibly wrapped in `CircuitChallengePayload`) containing:
  - timestamp τ
  - matrix rows (count determines number of rounds)
- **Prover response:** minimal payload:
  - τ
  - bearings $(b_1, \ldots, b_L)$
  - `proof_hash` (a SHA-256 over round/bearing/witness strings)

d) 6.2 Witness construction (formal specification): Per round i, for each ring j: - compute offset $o_{i,j}$ as in Section 4. - sample R_j characters sequentially with wraparound: - $R_0 = 30$ uppercase, $R_1 = 30$ lowercase, $R_2 = 12$ digits.

$$\text{Sample}_{i,j}[t] := \mathcal{A}_j[(o_{i,j} + t) \bmod |\mathcal{A}_j|], \quad t = 0..R_j - 1.$$

Then zone slices are fixed-width: - uppercase zone slice: 5 chars, indices $[5z..5z + 4]$ - lowercase zone slice: 5 chars, indices $[5z..5z + 4]$ - numeric zone slice: 2 chars, indices $[2z..2z + 1]$

Witness is concatenation:

$$W_{i,z} := S_{i,0}[5z{:}5] \| S_{i,1}[5z{:}5] \| S_{i,2}[2z{:}2]$$

where $S_{i,j}$ denotes $\text{Sample}_{i,j}$.

e) 6.3 Pseudocode: witness generation (depth 3):

```
PROCEDURE HyperplaneV2(E, alphabets[0..2], round i):
  samples_upper <- SampleRing(E, alphabets[0], i, ring=0, count=30)
  samples_lower <- SampleRing(E, alphabets[1], i, ring=1, count=30)
  samples_digit <- SampleRing(E, alphabets[2], i, ring=2, count=12)

  FOR z IN 0..5:
    upper <- samples_upper[5*z : 5*z+5]
    lower <- samples_lower[5*z : 5*z+5]
    digit <- samples_digit[2*z : 2*z+2]
    table[z] <- upper + lower + digit
  RETURN table

PROCEDURE SampleRing(E, alphabet A, round i, ring j, count R):
  offset <- ComputeAlphabetOffset(E, i, j, |A|)
  FOR t IN 0..(R-1):
    out[t] <- A[(offset + t) mod |A|]
  RETURN out
```

f) 6.4 Statement proved (same predicate): Variant B still uses:

$$\textsf{Accept} \iff \exists S \in \mathcal{S} \text{ s.t. } \forall i : \; s_i \in W_{i,z_i}.$$

The difference is: the verifier can reconstruct W_{i,z_i} from τ and bearings (given access to the embedded entropy pool and the same hyperplane generator).

g) 6.5 Completeness proof (Variant B): Same as Variant A: if a prover selects bearings that map to zones whose witnesses contain the needed s_i, the verifier will accept.

h) 6.6 Soundness bound and the duplication effect (Variant B): Unlike Variant A, Variant B samples **30** uppercase letters from a **26**-letter alphabet (and similarly for lowercase). This forces wraparound and duplicates.

Let uppercase alphabet size be 26. Sampling $R_0 = 30$ implies: - the first 4 sampled letters repeat at the end (positions 26..29).

Because zones are carved from contiguous sample blocks, duplicates can land in different zones, meaning some characters appear in **two** zones in a given round.

Define:

$$t_i := \#\{z \in \{0, \ldots, 5\} : s_i \in W_{i,z}\}$$

Then any adversary who does not know s_i and must guess z_i has:

$$\Pr[\text{pass round } i] \leq \frac{t_i}{6}.$$

Overall:

$$\Pr[\mathsf{Accept}] \leq \prod_{i=1}^{L} \frac{t_i}{6}.$$

Worst-case if $t_i = 2$ for all rounds:

$$\Pr[\mathsf{Accept}] \leq \left(\frac{2}{6}\right)^L = \left(\frac{1}{3}\right)^L.$$

This is still exponential in L, but weaker than the $(1/6)^L$ intuition.

i) 6.7 Nonce matrix values are not currently used to influence witnesses: In the nonce-driven generator, the matrix row values are read but not used to alter witness construction. So the strongest "binding" is to τ (which selects entropy) and the deterministic generator itself, not to the full matrix contents.

Operationally, the appropriate summary is: - The challenge nonce controls **round count** and **timestamp** τ. - Witness table generation is a function of entropy selected by τ and the configured alphabets; the row values do not currently contribute.

---

8) 7) Variant C: Minimal payload format and proof hashing:

a) Implications and positioning: Minimal payloads shift ENI6MA toward a verifier-centric security model: responders emit a compact path (bearings) and a time anchor (τ), while verifiers reconstruct the heavy objects (witnesses/hyperplanes) locally. The practical effect is reduced transport risk and reduced client-side complexity: fewer derived artifacts traverse logs, proxies, and storage systems, and there is less opportunity for cross-implementation divergence (encoding, ordering, normalization). A `proof_hash` adds a deterministic digest over the reconstructed transcript and can be treated as an audit handle and consistency checksum. The crucial point is predicate selection: a knowledge-checking verifier enforces membership against $\mathcal{S}$, whereas a structural verifier may only check reconstructability and well-formedness. The hash does not replace the knowledge predicate; it only binds a compact response to a deterministically reconstructed transcript.

b) 7.1 Minimal payload: The minimal proof payload format used in ALGO1-Nonce response is:

- τ (timestamp)
- bearings array $(b_1, \ldots, b_L)$
- `proof_hash`

Bearings are letters from $\{U, D, L, R, F, B\}$.

c) 7.2 Proof hash equation: Let W_i denote the witness string for round i selected by bearing b_i (i.e., the zone derived from b_i). Define the transcript hash by hashing the concatenation of round-tagged strings:

$$H := \mathsf{SHA256}\left(\|_{i=1}^{L} \mathsf{bytes}\left(\mathrm{str}(i)\ ||\text{":"}||b_i||\text{":"}||W_i\right)\right).$$

This is used for internal consistency in some interactive flows.

d) 7.3 Pseudocode (depth 3):

```
PROCEDURE ProofHash(bearings[1..L], witnesses[1..L]):
  h <- SHA256_INIT()
  FOR i IN 1..L:
    line <- ToString(i) + ":" + bearings[i] + ":" + witnesses[i]
    h <- SHA256_UPDATE(h, UTF8(line))
  RETURN SHA256_FINAL_HEX(h)
```

e) 7.4 Verifier roles: knowledge-checking vs structural: Two verifier roles must be distinguished:

- A **knowledge-checking verifier** reconstructs witnesses and checks secret membership; this matches the acceptance predicate in Sections 5 and 6.
- A **structural verifier** reconstructs witnesses (and any nonce structure) and checks only well-formedness and reconstructability; it can succeed even when the secret-membership predicate fails.

Equivalently: one verifier checks knowledge of a configured secret, while the other checks only consistency with a reconstructable challenge envelope.

---

9) 8) Variant N: Nonce/matrix subcircuit (generation + validation): This is a separate construction used to create challenge nonces.

a) Implications and positioning: The nonce/matrix subcircuit provides a structured challenge object where many systems use opaque randomness. The implication is that challenges can be audited and (by a pool-holder) validated after the fact: row hashes bind row values, indices, π, and τ, and the validator can recover entropy indices by brute force over the embedded pool. This can be superior to "random nonce only" designs in environments where you need deterministic regeneration, forensic traceability, or a verifiable link from a compact challenge to a privileged entropy corpus. It also provides natural anchors for a ledger: a nonce UUID can be reserved, consumed, rejected (expired, replay), and associated with a payload hash, creating an append-only record of challenge lifecycle events. The trade-offs are important for correct claims: because validation explicitly searches the pool, the construction is not hiding to a pool-holder, and its security is closer to "binding and reconstructability" than to "privacy." Likewise, while SHA-256 collision resistance

supports informal binding arguments, this is not a formal commitment scheme with hiding guarantees. In positioning terms, Variant N is superior as an operational integrity mechanism—tying challenges to a reproducible derivation and enabling strict burn/TTL enforcement—rather than as a stand-alone cryptographic hardness primitive. It strengthens system security when combined with the ALGO1 transcript-space barrier (6^L) and with ledger controls that bound online attempts. Compared to ad-hoc timestamp nonces, this structure supplies deterministic replay/expiry hooks and explicit audit semantics without relying on "hidden randomness" narratives.

b) 8.1 Inputs:

- τ: timestamp
- π: embedded prime-like constant (512-bit)
- an embedded entropy pool containing entries ϵ_k (each a 512-bit-ish value)
- alphabet_count (controls number of columns/moduli)
- a list of entropy indices $I = [i_0, \ldots, i_{L-1}]$ chosen deterministically from τ in some flows:

$$i_r := ((\tau \bmod N) + r) \bmod N$$

where N is the pool size.

c) 8.2 Row values: Let c be the number of columns and let $m_0, \ldots, m_{c-1}$ be moduli derived from `alphabet_count` (for alphabet_count=3, moduli are 30,30,12).

For row r, with epsilon $\epsilon = \epsilon_{i_r}$, column value:

$$v_{r,k} := \mathsf{u64}\big(\mathsf{SHA256}(\mathrm{tag}_k \| \epsilon \| \pi \| \tau)\big) \bmod m_k.$$

d) 8.3 Row hash: For row r with entropy index i_r and values $v_{r,0..c-1}$:

$$h_r := \mathsf{SHA256}(\mathrm{tag} \| i_r \| v_{r,0..c-1} \| r \| \pi \| \tau).$$

e) 8.4 Validation method (brute-force recovery): To validate a nonce, the validator attempts to recover each entropy index by brute-forcing all pool entries until it finds an index that reproduces the given row hash.

This yields a strong **completeness** property (generated nonces validate), but it is not a hiding commitment: the scheme is designed so pool holders can recover indices.

f) 8.5 Pseudocode: nonce generation and validation (depth 3):

```
PROCEDURE GenerateMinimalNonce(entropy_pool, indices[0..L-1], pi, tau, moduli[0..c-1]):
  FOR r IN 0..(L-1):
    epsilon <- entropy_pool[indices[r]]
    values <- DeriveRowValues(epsilon, pi, tau, moduli)
    row_hash <- RowHash(indices[r], values, r, pi, tau)
    rows[r] <- (values, row_hash, row_index=r)
  RETURN Nonce(timestamp=tau, rows=rows)

PROCEDURE DeriveRowValues(epsilon, pi, tau, moduli[0..c-1]):
  FOR k IN 0..(c-1):
    digest <- SHA256("COLUMN_k" || epsilon || pi || tau)
    x <- FirstU64LittleEndian(digest)
    values[k] <- x mod moduli[k]
  RETURN values

PROCEDURE ValidateNonce(nonce, entropy_pool, pi, expected_tau, moduli):
  IF nonce.timestamp != expected_tau:
    RETURN FAIL
  FOR r IN 0..(RowCount(nonce)-1):
    target <- nonce.rows[r].row_hash
    found <- FALSE
    FOR idx IN 0..(PoolSize(entropy_pool)-1):
      epsilon <- entropy_pool[idx]
      values <- DeriveRowValues(epsilon, pi, nonce.timestamp, moduli)
      IF RowHash(idx, values, r, pi, nonce.timestamp) == target:
        found <- TRUE
        BREAK
    IF found == FALSE:
      RETURN FAIL
  RETURN OK
```

---

10) 9) Formal security claims per variant (claims supported by the construction):

a) Implications and positioning: The statements below are intentionally conservative because security outcomes depend on which predicate is actually enforced, not on terminology. Stating the claims explicitly makes the mechanism audit-friendly: it separates properties supplied by witness generation (completeness and counting-based guessing resistance), properties not supplied (zero-knowledge and hiding), and properties that arise only under deployment controls (expiry, one-time use, replay rejection, and audit logging). This separation matters because knowledge-checking validation and structural validation are sometimes conflated despite enforcing different predicates; reconstructability alone is not a proof of knowledge. Explicit claims also support evolution: if duplication is reduced in the sampling construction or additional challenge structure is bound into witness generation, the parameter t_i and the corresponding bounds improve in a way that is immediately visible. Finally, the statements clarify where practical strength originates: exponential decay in the blind-guess success probability as a function of L, combined with strict online attempt limiting via time-bounded envelopes and consumption policy.

b) 9.1 Variant A (ALGO1-Plain):

- **Completeness:** yes (trivial from membership check).
- **Guessing resistance:** if each secret character maps to exactly one zone per round, then $(1/6)^L$ for random guessing.
- **Not zero-knowledge:** transcript reveals zone-bucket information about secret characters.

c) 9.2 Variant B (ALGO1-Nonce):

- **Completeness:** yes.
- **Guessing resistance:** bounded by $\prod_i (t_i/6)$, with worst-case $(1/3)^L$ due to duplicate placement from wraparound sampling.
- **Not zero-knowledge:** same leakage issue.
- **Challenge binding:** mostly via τ (and thus entropy selection) rather than full matrix-row influence on witnesses.

d) 9.3 Variant N (Nonce/matrix):

- **Completeness:** generated nonces validate under brute-force recovery.
- **Binding (informal):** collision-resistance of SHA-256 suggests it is difficult to forge different row contents with the same row hash, absent collisions.
- **Not hiding:** pool holders can recover indices by design.

11) 10) Worked toy example (intuition without real secrets): Use a toy ring to see why "one zone contains the character" gives a guessing bound.

a) Implications and positioning: This toy example translates ALGO1's security story into the simplest possible model: a round is a bucket choice, and passing is choosing the unique bucket that contains the target. The implication is that you can reason about brute-force cost without needing any cryptographic background: if the best an attacker can do is guess one of n zones each round, then security scales exponentially with the number of rounds. That clarity is superior to many authentication mechanisms whose security intuition is buried in implementation details (encodings, protocol states, library behaviors) and therefore hard to audit. It also highlights exactly where Variant B differs from Variant A: if the generator duplicates characters such that two zones can be "correct" for a given round/character, the bound weakens in a straightforward way by replacing $1/n$ with t_i/n. In deployment planning, this section's implication is practical: you can pick a target acceptable risk per challenge, solve for L using 6^{-L} (or the weaker bound when duplicates exist), and then set TTL, burn, and rate limits so the attacker cannot accumulate enough attempts to reach meaningful success probability. The toy model also underlines a key positioning point: ALGO1's primary defense is not "mystery permutation hardness," it is a quantifiable transcript space combined with strict challenge lifecycle enforcement—often a superior approach in operational environments where you can reliably bound online attempts but cannot assume perfect secrecy of client devices or logs. This is a concrete advantage over intuition-only security.

Let $n = 3$ zones, alphabet $\mathcal{A} = [A, B, C, D, E, F]$ and rotation offset $o = 0$. Partition into 3 slices of length 2: - zone 0: [A,B] - zone 1: [C,D] - zone 2: [E,F]

If the secret char is D, only zone 1 contains it. A guesser has probability $1/3$ of choosing zone 1 that round. Repeat over L rounds to get $(1/3)^L$.

Variant A uses $n = 6$ zones and three rings; Variant B uses fixed sampling that can duplicate characters, sometimes creating two "correct" zones.

---

12) 11) Minimal equation sheet (quick reference): **Acceptance predicate.**

$$\mathsf{Accept} \iff \exists S \in \mathcal{S} \text{ s.t. } \forall i: \ s_i \in W_{i,z_i}.$$

a) Implications and positioning: The equation sheet is the protocol's "portable core": it's what implementers can copy into tests, audits, or a second verifier implementation without dragging in the rest of the narrative. The implication is reproducibility—ALGO1 can be summarized as a small set of deterministic functions (offsets and witness generation) plus one acceptance predicate, which makes discrepancies between implementations easy to localize. This compactness is often superior to complex proof systems where the verifier spans many algebraic objects and where tiny serialization or domain-separation mistakes can invalidate correctness. Here, the explicit presence of t_i in the bound and the explicit definition of offsets also forces honest reasoning: it becomes immediately clear which parameters are "security knobs" (round count L, uniqueness vs duplication) and which are reconstruction knobs (the mapping from (E, τ) to offsets). The sheet also supports safe evolution: if you refactor Variant B to avoid duplicates, you can point to the same t_i formula and show the improvement; if you bind matrix-row values into witness generation, you can extend the witness definition and preserve the acceptance predicate. In short, this section's superiority is that it makes the protocol formally discussable: it allows quick peer review and reduces reliance on prose interpretation, which is one of the most common sources of security regressions in real systems. It also serves as a reviewer checklist: if these equations can't be reproduced exactly, the implementation shouldn't be trusted as a verifier, across languages and platforms.

Offsets.

$$o_{i,j} = (E + (3i + j)) \bmod |\mathcal{A}_j|.$$

Variant A slice length (ring j, zone z).

$$\ell_j(z) = \left\lfloor \frac{|\mathcal{A}_j|}{6} \right\rfloor + \mathbf{1}[z < (|\mathcal{A}_j| \bmod 6)].$$

Variant B guessing bound using multiplicity t_i.

$$\Pr[\mathsf{Accept}] \le \prod_{i=1}^{L} \frac{t_i}{6}, \quad t_i = \#\{z : s_i \in W_{i,z}\}.$$

Worst-case Variant B bound (if $t_i = 2$ always).

$$\Pr[\mathsf{Accept}] \le \left(\frac{1}{3}\right)^L.$$

---

13) 12) Concise technical characterization: A concise, technically correct characterization is:

> ENI6MA ALGO1 is a multi-round identification protocol. In each round, a deterministic hyperplane generator partitions or samples configured alphabets into six zone witnesses. A response is accepted when there exists a configured secret string $S = s_1 \dots s_L$ such that, for every round i, the secret character s_i lies in the witness selected by the prover for that round. Security is primarily a counting-based guessing-resistance property with exponential decay in the number of rounds; it is not a zero-knowledge proof.

a) Implications and positioning: This characterization is intentionally conservative: it avoids attributing properties (simulator-based zero-knowledge, transcript privacy, or reduction-based soundness) that do not follow from the acceptance predicate. It also foregrounds the actual control parameters: the response-path space 6^L, the multiplicity term

t_i that captures whether a given secret character can appear in more than one zone in a round, and the envelope policies (expiry, one-time use, replay rejection, audit logging) that bound online attempts. Finally, the phrasing is stable under evolution: changes such as reducing duplication in sampling or binding additional challenge structure into witness generation translate directly into changes in t_i and the corresponding bounds without requiring reinterpretation of the core predicate.

---

14) 13) Appendix: Variant index:

- **Variant A:** ALGO1-Plain (quotient/remainder foliation of rotated alphabets).
- **Variant B:** ALGO1-Nonce (fixed 5/5/2 sampling blocks; possible duplicates).
- **Variant C1:** knowledge-checking verifier (reconstruct + secret membership check).
- **Variant C2:** structural verifier (reconstructability/well-formedness check without secret membership).
- **Variant N:** nonce/matrix generator + brute-force validation by row hash matching.

a) Implications and positioning: The index is not merely navigational: it is a security control against variant drift, where different teams use different generators or different predicates under the same label "proof" and accidentally downgrade what is enforced. A concise index provides a stable reference for design review, implementation review, and audit (e.g., requiring a knowledge-checking verifier rather than a structural check). It also supports disciplined composition: a deployment can require Variant B for compact challenge/response, reserve Variant A for baseline analysis, and treat Variant N as a challenge-integrity layer. When the system evolves—new generators, new envelope policies, new proof formats—the index provides a canonical place to record how predicates and constructions map, which materially reduces the risk of accidental substitution.

---

15) 14) Conclusion: brute-force resistance, "permutation accumulation," and ledger-grounded time envelopes: The central result is simple but consequential: ENI6MA ALGO1 does not derive its strength from a single algebraic hardness assumption in the SNARK/STARK sense; its practical strength arises from (i) a high-dimensional response space created by composing many per-round micro-decisions, and (ii) a time-bounded challenge lifecycle (TTL, one-time consumption, audit grounding) that limits online probing. When describing ALGO1 as a "large dimensional permutation accumulator," the construction-aligned interpretation is that each round induces a choice among six zones, and the full response is the accumulation of those choices across L rounds; the response space grows as a Cartesian power, i.e., 6^L. This is the object to count when quantifying brute-force effort, because an attacker is searching for an accepting path through a sequence of round tables, not inverting a single permutation.

a) 14.1 Acceptance is a membership predicate; the brute-force target is an accepting path: The acceptance predicate that matters for proof-of-knowledge validation (knowledge-checking verifier semantics) can be restated as:

$$\mathsf{Accept} \iff \exists S \in \mathcal{S} \text{ s.t. } \forall i : \ s_i \in W_{i,z_i}.$$

For a fixed secret S and a fixed challenge context (i.e., fixed τ and fixed witness generator), define the set of "winning zones" in round i:

$$Z_i := \{z \in \{0, \ldots, 5\} : s_i \in W_{i,z}\}.$$

Let $t_i := |Z_i|$. Then the number of accepting transcripts (zone sequences) for that one secret is:

$$N_{\text{accept}}(S) = \prod_{i=1}^{L} t_i.$$

The total response space is:

$$N_{\text{all}} = 6^L.$$

If an attacker has no information about S beyond "it is one of the configured secrets," and their only strategy is to submit zone sequences and observe accept/reject, then the natural brute-force question becomes: how many sequences must be tried before one lands in the accepting set? In the idealized Variant A structure where $t_i = 1$ for every round (one winning zone per character per round), there is essentially one accepting transcript per secret, so $N_{\text{accept}}(S) = 1$ and a uniform random guess succeeds with probability:

$$p = \Pr[\mathsf{Accept}] = \frac{1}{6^L}.$$

In Variant B, because of wraparound duplicates in the fixed-length sampling, t_i can be 2 in some rounds for some characters, yielding:

$$p \le \frac{\prod_i t_i}{6^L},$$

and in the worst case (if $t_i = 2$ for all rounds) $p \le (1/3)^L$. These are information-theoretic bounds in the narrow sense that they do not rely on computational assumptions: they follow from counting and from the fact that acceptance depends on membership of unknown characters in a small family of zone strings.

b) 14.2 What "information-theoretic perfect secrecy" would mean—and what ALGO1 actually provides: It is tempting to equate a large combinational search space with "information-theoretic perfect secrecy," but these are different properties. Shannon perfect secrecy is a property of encryption schemes: roughly, the observed artifact is statistically independent of the secret message. ALGO1 is not an encryption scheme; it is an identification protocol. Its transcripts are correlated with the secret because the verifier's acceptance predicate is defined directly in terms of the secret's characters appearing in selected witnesses. Therefore, ALGO1 (as implemented) should not be described as providing Shannon-style perfect secrecy of S against an observer who sees transcripts over time.

However, there is a meaningful information-theoretic statement that *does* align with the code and with the "brute force" framing: for a single fixed challenge, under the blind-guess model, the best achievable success probability is bounded by 6^{-L} (or by the t_i-adjusted bound). This is sometimes loosely described as "information-theoretic resistance to brute-force validation," because it is a probability bound that depends only on combinatorics and does not depend on assumptions like discrete-log hardness. If that is what is meant by "perfect secrecy against brute force," the precise and defensible restatement is: **the probability of passing by brute-force guessing can be made negligible by choosing L sufficiently large, and that negligibility is information-theoretic with respect to the guessing model**. It is not secrecy of the transcript; it is a bound on forging success for passing a challenge.

c) 14.3 Counting the "massive permutation space" in construction-aligned terms: The phrase "massive permutation space" should be grounded in what the generator actually does. In Variant A, each ring witness is produced by a cyclic rotation of an alphabet and a foliation partition. A cyclic rotation family on an alphabet of size $|\mathcal{A}_j|$ has exactly $|\mathcal{A}_j|$ possible permutations, not $|\mathcal{A}_j|!$. Under the default alphabets, the maximum number of distinct rotation triples per round is bounded by:

$$|\mathcal{A}_0| \cdot |\mathcal{A}_1| \cdot |\mathcal{A}_2| = 26 \cdot 26 \cdot 10 = 6760.$$

This is not astronomical, and it should not be advertised as factorial-scale permutation hardness. The "large dimensional" effect comes from composition across rounds and from the response path space: regardless of how the table is generated, the responder's transcript space is 6^L, and the verifier only accepts transcripts that match the secret membership predicate. That is the dominant count that drives brute-force cost of *validation*.

If you want to combine both kinds of counting (generator variability plus transcript variability), you can describe the combined state as: "for each round there is a deterministic table determined by offsets, and the attacker must choose one of 6 responses per round." Across L rounds, even if the generator were viewed as producing independent per-round rotations, the number of possible sequences of rotation triples would be at most 6760^L, and the number of possible response sequences is 6^L; the protocol's practical resistance to brute-force validation is governed by the latter when the challenge is fixed, and governed by attempt limiting when the attacker can request fresh challenges.

d) 14.4 Why TTL expiration of the τ envelope changes the brute-force game: In real deployments, brute-force validation is not an offline computation; it is an online attempt process where the attacker learns success by querying a verifier. The system's ledger layer is what turns the theoretical bound into practical security by restricting the number of attempts. In the codebase's ledger implementation, a reserved nonce is created with a TTL (often configured by `NONCE_TTL_SECONDS`) and is enforced as:

$$\text{expired} \iff \text{created\_at} + \text{ttl} < \text{now}.$$

On consume, the ledger enforces: - **Expiry**: expired nonces are rejected. - **Replay**: spent nonces are rejected. - **Tau binding**: if the client supplies a τ value for consumption, the ledger can reject a τ mismatch.

These rules mean that an attacker cannot take one challenge and try k different bearing sequences against it until one passes. Instead, each attempt consumes (or at least must be associated with) a fresh, valid nonce envelope, and the attacker is constrained by the rate of nonce reservation, the TTL window, and any authorization/rate limiting wrapped around the ledger endpoints. This transforms the attacker's expected work from "enumerate 6^L sequences offline" into "obtain and burn on the order of $1/p$ nonce envelopes online," where p is the per-attempt success probability. In the ideal Variant A setting $p = 6^{-L}$, so the expected number of envelopes to burn is 6^L. Even modest L values quickly make this infeasible under a strict TTL and a strict one-time-use ledger.

e) 14.5 Grounded proof hash ledger records: what is grounded, and why it matters: The ledger service computes a hash of the submitted response payload (e.g., `SHA256(response_payload_json)`) and records it as a response hash and/or in audit events. This provides "grounding" in an operational, tamper-evident sense: every consumed nonce can be associated with an immutable hash of exactly what was submitted, and every rejection can be logged with a reason (expired, replay, tau mismatch). This is not secrecy; it is integrity and accountability. It prevents a class of operational brute-force patterns—such as replaying old payloads against new challenges or denying that a response was ever submitted—because the ledger record becomes the system of record for what was attempted and when.

When combined with minimal payloads, this grounding has an additional implication: the system can avoid ever persisting full witness content in transit or storage while still maintaining a robust audit trail. A responder can transmit bearings and τ (and optionally `proof_hash`), and the verifier can store only the payload hash and the consume event. This reduces sensitive data retention while preserving forensic capability—often a superior design in real systems where logs and databases are frequent sources of leakage.

f) 14.6 Putting it together: why brute-force validation becomes infeasible in practice: The strongest code-supported argument for ENI6MA's brute-force resistance is therefore layered: 1) **Combinational barrier**: the response space is 6^L, and in the cleanest variant there is essentially one winning choice per round, yielding $p = 6^{-L}$. 2) **High-dimensional accumulation**: the attacker must succeed in *every* round, so the success probability multiplies across rounds; this is the "accumulator" effect. 3) **Time envelope**: challenges are tied to a τ value, which is used to select entropy and reconstruct tables, and ledger enforcement can bind consumes to the reserved τ. 4) **TTL + burn**: each challenge envelope expires and is consumable once, limiting attempts and preventing indefinite probing. 5) **Audit grounding**: response payload hashes are recorded, enabling replay detection, rate enforcement, and post-incident analysis.

Under these conditions, the number of "permutations" (more precisely, response sequences) that must be completed to obtain a successful validation is on the order of 6^L in expectation for blind brute force in the ideal uniqueness case, and still exponential in L even under duplication. Because each attempt is time-bounded and consumes a nonce, the attacker must also complete those attempts under TTL constraints and any authorization/rate limits—making the attack infeasible well before reaching the combinational bound. This yields a practical security statement supported by the construction: not Shannon perfect secrecy of transcripts, but a measurable, enforceable, high-dimensional brute-force barrier for passing validation, strengthened by a grounded lifecycle that limits and records attempts.

g) 14.7 Deployment guidance for aligning reality with the intended security story: To ensure the system actually realizes the above benefits, deployments should: (i) make acceptance decisions using a knowledge-checking verifier that enforces secret membership against $\mathcal{S}$, (ii) enforce nonce TTL and one-time consumption in the ledger/service layer for any online verification path, (iii) choose L so that 6^{-L} (or the t_i-adjusted bound) meets the desired per-attempt risk budget, and (iv) store and monitor ledger audit events (including payload hashes) so brute-force patterns are visible and enforceable. When those pieces are aligned, ENI6MA's ALGO1 variants provide a coherent mechanism for resisting brute-force validation attacks through a large-dimensional response space and a grounded, expiring challenge envelope.

h) 14.8 Quantifying brute-force attempt budgets under TTL and burn semantics: It is useful to translate "6^L is huge" into an explicit online-attack budget, because ledger TTL and burn semantics change the attacker's objective from "search the space" to "obtain enough attempts before the envelope closes." Let p be the per-attempt success probability against a single fresh nonce envelope. In the cleanest Variant A model with one winning zone per round, $p = 6^{-L}$; in Variant B, p is upper-bounded by $(\prod_i t_i)/6^L$. If an attacker can make A independent attempts (each consuming a fresh nonce within TTL), then the probability of at least one success is:

$$\Pr[\text{success}] = 1 - (1-p)^A \approx Ap \quad (\text{small } p).$$

This approximation is the key deployment insight: you do not need "infinite hardness," you need $A\,p$ to be negligible under your operational constraints. TTL and burn semantics bound A in multiple ways: each attempt requires (a) reserving a nonce, (b) submitting a response before expiry, and (c) avoiding replay (spent) rejection. If you additionally enforce rate limits (per origin, per verifier ID, per policy ID) and require authorization for reservation/consumption, then A becomes a controlled quantity rather than an attacker-chosen quantity. For example, even if a system allowed an attacker to attempt $A = 10^6$ fresh nonces (already a very large budget for a burn-enforced workflow), with $L = 20$ you have $p = 6^{-20} \approx 3.7 \times 10^{-16}$, so $A\,p \approx 3.7 \times 10^{-10}$—still negligible. This is the practical meaning of the "large-dimensional accumulator" effect: increasing L by a small constant multiplies the attacker's required attempt budget by a factor of 6 (or by $6/t_i$ in the duplicated case), and TTL/burn ensures those attempts must be purchased in real time with real system resources.

i) 14.9 What brute force is (and is not) protecting you from in ENI6MA: In ENI6MA the primary defenses become integrity and compartmentalization: binary integrity checks, least-privilege separation between responders and verifiers, and ledger policy (so that even a correct response can only be used once per reserved envelope and is fully audited). This is why ENI6MA's design couples a combinational proof gate to an operational ledger: the "perfect" part of the brute-force argument is not secrecy of transcripts, but the fact that under the intended threat model (attacker does not know S), success probability is bounded by counting; and under real deployments, the ledger makes the attack observable, rate-limitable, and time-bounded. The net result is a defensible security posture: ALGO1 supplies an information-theoretic forging bound against blind guessing, and the TTL + burn + audit trail transforms that bound into practical protection by constraining and recording the attacker's attempt budget, under realistic, enforceable operational constraints.