THE NOVAK PROTOCOL: Proof-Before-Action (PbA) as a Cryptographic Primitive for Deterministic Execution Integrity

Matthew S. Novak
Creator and Author of the NOVAK Protocol
licensing@novakprotocol.com

2025

Abstract

We introduce the NOVAK Protocol, the first cryptographic primitive to provide deterministic execution integrity through a mechanism we define as Proof-Before-Action (PbA). Unlike signatures, attestations, TPM-based measurement, secure logging, or distributed ledger verification, PbA requires the cryptographic validation of rule correctness, data correctness, output correctness, and temporal singularity *before* any action is allowed to execute. This transforms integrity from a post-execution audit to a pre-execution enforcement model.

PbA is realized through three constructions: the Hash-Verified Execution Trace (HVET), the Execution Identity Receipt (EIR), and the Recursive Global Audit Chain (RGAC). We provide formal definitions, canonical encoding rules, a complete adversarial model, and full cryptographic reductions proving that forging any component of NOVAK reduces to breaking the collision resistance of the underlying hash function (SHA-2, SHA-3, or BLAKE3). We incorporate the NOVAK Laws L0–L15 and Industry Addenda PL-X and PS-X as baseline environmental constraints for deterministic digital governance.

This document consolidates the full NOVAK specification: formal definitions, cryptographic architecture, encoding rules, security analysis, reduction proofs, implementation guidance, and test vectors. It establishes execution integrity as a new domain within cryptography and provides the first deterministic integrity model for automation, AI governance, critical infrastructure, financial adjudication, and legal–regulatory systems.

Contents

1 Introduction

Modern computing systems validate integrity *after* execution: logs audit what already happened, signatures authenticate origin, attestations certify boot states, and blockchains record transactions post hoc. None of these mechanisms prevent a system from executing incorrectly, illegally, unsafely, or manipulatively in the first place.

This paper introduces a new cryptographic primitive that enforces correctness *before* execution. We call this primitive **Proof-Before-Action (PbA)**. A system implementing PbA requires a verifiable cryptographic proof that the rule, data, output, and timestamp of a given computation are correct—according to canonical, deterministic definitions—before performing any action.

PbA is implemented through the NOVAK Protocol using three core constructions:

[label=**(a)**]

1. **HVET** — Hash-Verified Execution Trace
   A binding of rule, data, output, and timestamp.
2. **EIR** — Execution Identity Receipt
   A self-authenticating, tamper-evident proof object.
3. **RGAC** — Recursive Global Audit Chain
   A chain of EIRs providing global immutability.

Together these mechanisms enforce deterministic execution integrity (DEI), guaranteeing that:

- the correct rule was used,
- the correct input was provided,
- the output matches the rule,
- the timestamp is unique and monotonic,
- and no action proceeds unless all conditions are satisfied.

This primitive is new: no classical construction—including MACs, signatures, consensus protocols, TPM measurement chains, secure logging, or blockchain—achieves pre-execution correctness.

PbA introduces a new tier of assurance enabling regulatory automation, AI safety, critical infrastructure enforcement, and highly trustworthy digital governance.

2 Related Work

2.1 Signatures and MACs

Traditional authentication mechanisms (MACs, signatures) ensure integrity and authenticity of messages but do not enforce pre-execution correctness. They validate authorship, not execution truth.

2.2 Attestation

Remote attestation mechanisms validate the system state at boot but do not enforce correctness during arbitrary execution and cannot guarantee temporal singularity.

2.3 TPM Measurement Chains

TPM-based integrity provides measurement logs but remains post-execution. TPMs do not bind rules, data, outputs, and timestamps into a deterministic integrity condition.

2.4 Secure Logging Systems

Systems such as Keyless Signatures or Certificate Transparency detect tampering after the fact; none enforce integrity before execution.

2.5 Blockchains and Consensus Protocols

Blockchains offer decentralized append-only logs but do not enforce rule correctness or data correctness within individual computations.

2.6 Tripwire and File Integrity Tools

Integrity-monitoring systems detect changes but do not enforce pre-execution correctness and cannot bind rules to outcomes or timestamps.

PbA therefore introduces a fundamentally distinct security paradigm: *execution-gated correctness*. No prior primitive has provided this guarantee.

3 NOVAK Laws (L0–L15) and Industry Addenda

The NOVAK Protocol is grounded in a deterministic governance model expressed through the NOVAK Laws L0–L15 and the Industry Addenda PL-X and PS-X. These laws act as baseline environmental constraints governing the behavior of any system implementing Proof-Before-Action (PbA). They are normative, not optional, and form the axiomatic layer for all formal definitions, proofs, and enforcement conditions.

3.1 3.1 NOVAK Laws L0–L15

[label=**L1:**]

1. **Truth Cannot Be Assumed.**
   No computational action may rely on unverifiable state.

2. **Truth Must Be Proven.**
   All execution must be preceded by cryptographic proof.

3. **Proof Must Bind Rule, Data, Output, Time.**
   These four elements form a minimal correctness tuple.

4. **Execution Requires Determinism.**
   A rule applied to identical data must produce identical output.

5. **Non-Determinism Without Justification is Invalid.**
   Systems must reject unverifiable or ambiguous execution paths.

6. **No Action Without Identity.**
   Every execution must have a cryptographically provable identity.

7. **Identity Requires Integrity.**
   Execution identity must include rule, data, output, and time.

8. **Integrity Requires Canonical Encoding.**
   All inputs must serialize deterministically.

9. **Temporal Integrity Must Be Singular.**
   Each execution must correspond to a unique, monotonic timestamp.

10. **Order Must Be Unambiguous.**
    Execution order must be unforgeable and globally consistent.

11. **Tampering Must Be Detectable.**
    Any deviation from canonical truth must be evident immediately.

12. **Detection Must Be Pre-Execution.**
    Errors identified after execution are insufficient.

13. **Proof Must Be Immutable.**
    Once generated, proofs must remain tamper-evident.

14. **Proof Must Be Portable.**
    Execution identity receipts must be verifiable in isolation.

15. **The System Must Refuse Incorrect Execution.**
    Enforcement completes the chain: proof-before-action.

3.2 3.2 Industry Addendum PL-X (Public Ledger Enforcement)

PL-X specifies cross-jurisdictional determinism for systems requiring public anchoring. It mandates:

- periodic hashing of RGAC states,
- optional Merkle anchoring to external ledgers,
- maintenance of consistent cross-domain truth models,
- verifiability without reliance on private infrastructure.

3.3 3.3 Industry Addendum PS-X (Private Sector Enforcement)

PS-X defines deterministic verification for private-sector operational domains, including:

- financial adjudication systems,
- AI model evaluation,
- claims-processing automation,
- internal compliance enforcement,
- reproducible regulatory event validation.

Both addenda exist to contextualize PbA in real-world enforcement systems and do not modify the core primitive; rather, they constrain its operational surface to ensure universal determinism.

4 Formal Definition of the Proof-Before-Action (PbA) Primitive

PbA is defined as a pre-execution cryptographic enforcement mechanism.

[Proof-Before-Action Primitive] A system implements **PbA** if and only if, for every action A:

$$\mathsf{HVET}(R, d, o, t) verify \mathbf{true}$$

must be satisfied before the system is permitted to perform A.
Where:

- R is a deterministic rule or function.
- d is the input data.

- $o = R(d)$ is the output produced by applying R to d.
- t is a monotonic timestamp.
- HVET is a hash binding of R, d, o, t.

PbA differs fundamentally from all known primitives:

- MACs, signatures, and attestations validate origin, not correctness.
- Blockchains validate global ordering, not rule–data–output truth.
- TPM measurement chains validate boot states, not arbitrary execution.
- Secure logs detect tampering after execution, not before.

PbA enforces correctness at the moment of execution, not after it.

This is the central contribution of the NOVAK Protocol: **execution cannot occur unless cryptographic truth is satisfied.**

5 Hash-Verified Execution Trace (HVET)

The Hash-Verified Execution Trace binds rule, data, output, and timestamp into a single cryptographically verifiable object.

[HVET Construction]

$$\mathsf{HVET} = H\big(H(R) \,\|\, H(d) \,\|\, H(o) \,\|\, H(t)\big)$$

Where:

- H is a collision-resistant hash function (SHA-2, SHA-3, BLAKE3).
- $H(R)$ uniquely identifies the rule as a canonical function.
- $H(d)$ binds the input data to the execution.
- $H(o)$ binds the output to both R and d.
- $H(t)$ enforces temporal singularity.

5.1 5.1 Canonical Semantics

Correct operation requires **canonical encoding**, ensuring that every implementation of R, d, o, and t produces identical byte representations across platforms. This eliminates ambiguity and prevents equivocation attacks.

5.2 5.2 Minimality of the Tuple

The tuple (R, d, o, t) is shown to be *necessary and sufficient* for correctness:

- Omitting R permits rule-substitution attacks.
- Omitting d permits forged input.
- Omitting o permits forged results.
- Omitting t permits replay and reordering attacks.

5.3 5.3 Linking to PbA

HVET is the enforcement barrier for PbA:

$$\mathsf{HVET}(R, d, o, t) verify \mathbf{true}$$

must be evaluated *before* action execution.
If HVET fails, the system must:

$$refuse(A)$$

per NOVAK Law 15.

5.4 5.4 Security Basis

The security of HVET reduces to the collision resistance of H:

$$\Pr[HVETforgery] \leq \Pr[Hashcollision]$$

A full proof is provided in Section 11 (Reduction Proofs).

6 Execution Identity Receipt (EIR)

The Execution Identity Receipt is a self-authenticating, portable, tamper-evident record of a single execution event. It binds rule, data, output, and timestamp through HVET and can be verified in isolation without dependence on external infrastructure.

[Execution Identity Receipt]

$$\mathsf{EIR} = \langle R, d, o, t, \mathsf{HVET} \rangle$$

An EIR serves four purposes:

1. **Proves correctness**: binds R, d, o, and t.
2. **Proves integrity**: any modification breaks HVET validation.
3. **Enforces determinism**: ensures $o = R(d)$.
4. **Provides self-contained auditability**: no external trust required.

6.1 6.1 Verification

A verifier checks an EIR by recomputing:

$$\mathsf{HVET}' = H\big(H(R)\|H(d)\|H(o)\|H(t)\big)$$

Then verifying:

$$\mathsf{HVET}' \stackrel{?}{=} \mathsf{HVET}$$

If equal, the EIR is valid.

6.2 6.2 Immutability

NOVAK Law 13 mandates that proofs must be immutable: any modification to any field of an EIR invalidates it due to the binding structure of HVET.

6.3 6.3 Portability

The EIR requires no signatures, no certificates, no blockchain, and no external infrastructure. It is self-validating using only hash-based cryptography.

7 Recursive Global Audit Chain (RGAC)

The RGAC links EIRs into a tamper-evident chain. Unlike blockchains, RGAC does not require consensus, mining, or distributed nodes. Its purpose is not decentralization but deterministic integrity and chronological enforcement.

[RGAC Construction] Let EIR_i be the i-th receipt. Then:

$$C_i = H(C_{i-1}\|\mathsf{EIR}_i)$$

with C_0 defined as a fixed initialization constant.

7.1 7.1 Properties

- **Tamper-evident**: modifying any EIR breaks the entire chain.
- **Order-preserving**: timestamps and chain position are consistent.
- **Minimal**: relies solely on hash functions.
- **Local**: no network communication is required.

7.2 7.2 Merkle Anchoring (Optional)

For systems requiring external proof-of-existence:

$$\mathsf{MerkleRoot} = \mathcal{M}(\mathsf{EIR}_1, \ldots, \mathsf{EIR}_n)$$

This root may be anchored into:

- public blockchains,
- public timestamp servers,
- regulatory notarization services.

This is governed by Industry Addendum PL-X.

7.3 7.3 Comparison to Blockchain

RGAC differs from blockchains:

- No miners or validators.
- No consensus mechanism.
- Local verification only.
- Zero trust dependencies.
- Absolute minimal computational cost.

It is a cryptographic chain, not a distributed ledger.

7.4 7.4 Reduction Basis

Security reduces to:

- EIR soundness,
- HVET collision resistance,
- chain-level collision resistance.

A full treatment appears in Section 11.

8 Canonical Encoding Specification

Deterministic execution requires deterministic serialization. The NOVAK Protocol mandates a strict canonical encoding ensuring that any system, platform, or implementation will produce identical byte streams for R, d, o, and t.

8.1 8.1 Requirements

Canonical encoding must satisfy:

1. **Unambiguous**: no two distinct logical values may serialize to the same bytes.
2. **Deterministic**: the same logical value must always serialize identically.
3. **Order-preserving**: structured data must follow a stable field order.
4. **Language-independent**: no dependency on runtime or implementation-specific behavior.
5. **Stable across versions**: future-compatible via explicit field tagging.

8.2 8.2 Encoding Form

NOVAK uses the following canonical format:

$$\mathsf{encode}(X) = \mathsf{len}(X) \parallel \mathsf{bytes}(X)$$

Where:

- len(X) is an unsigned 64-bit big-endian length.
- bytes(X) is the UTF-8 or binary encoding of X.

8.3 8.3 Composite Encoding

The tuple used for HVET is encoded as:

$$\mathsf{encode}(R) \| \mathsf{encode}(d) \| \mathsf{encode}(o) \| \mathsf{encode}(t)$$

This byte stream is then hashed to produce:

$$X = H(R) \| H(d) \| H(o) \| H(t)$$

followed by:

$$\mathsf{HVET} = H(X)$$

8.4 8.4 Canonical Timestamp

The timestamp t must follow ISO 8601 with UTC normalization:

$$t = YYYY - MM - DDTHH : MM : SS.sssZ$$

Non-monotonic clocks MUST be rejected.

9 Security Model

The NOVAK Protocol operates under a standard cryptographic model with the following assumptions:

9.1 9.1 Hash Function Assumptions

NOVAK assumes H is:

- collision-resistant (CR),
- preimage resistant (PR),
- second-preimage resistant (SPR).

We consider SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-512, and BLAKE3 to meet these assumptions.

9.2 9.2 System Requirements

A system implementing PbA must:

- provide deterministic rule execution,
- maintain canonical encoding,
- ensure unique timestamps,
- compute HVET before execution,
- reject any action without valid HVET.

These requirements are mandated by NOVAK Laws 3–15.

9.3 9.3 Adversarial Capabilities

We assume an adversary may:

- modify rules,
- modify or substitute input data,
- modify outputs,
- forge timestamps,
- reorder operations,
- inject counterfeit execution traces,
- attempt replay attacks,
- attempt HVET forgery.

9.4 9.4 Trusted Components

The only trusted components are:

- the hash function H,
- the canonical encoding specification,
- the deterministic semantics of R.

Everything else — the system, storage, user environment, network — is considered untrusted.

10 Adversarial Model

The NOVAK adversary $\mathcal{A}$ is defined as a probabilistic polynomial-time (PPT) adversary with full capability to interfere with all aspects of system execution except the cryptographic hash function.

10.1 10.1 Adversarial Goals

$\mathcal{A}$ attempts to produce any of the following:

1. A false execution trace.
2. A modified rule R' producing a counterfeit output.
3. A substituted or tampered input d'.
4. A forged or incorrect output o'.
5. A manipulated timestamp t'.
6. A forged EIR.
7. A forged RGAC chain link.

The ultimate objective is to cause the system to execute incorrectly *without detection.*

10.2 10.2 Attack Surfaces

Potential attacks include:

- **Rule substitution**: replacing R with R'.
- **Data substitution**: replacing d with d'.
- **Output forgery**: claiming o' was produced by $R(d)$.
- **Timestamp manipulation**: replay or preplay attacks.

- **Trace splicing**: injecting falsified intermediate states.
- **Deletion attacks**: removing an EIR or chain link.
- **Reordering attacks**: misrepresenting execution order.
- **HVET collision attacks**: attempting to make distinct tuples collide.

10.3 10.3 Infeasibility of Forgery

We prove in Section 11 that:

$$\Pr[\mathcal{A} forges HVET] \leq \Pr[Hash collision]$$

which is negligible under modern cryptographic assumptions.

10.4 10.4 Threat Surfaces Not in Scope

PbA does not address:

- hardware side-channel attacks,
- timing attacks,
- memory corruption vulnerabilities,
- compromised system clocks,
- violations of canonical encoding.

These must be mitigated through orthogonal system hardening.

11 Reduction Proofs

This section presents full cryptographic reductions demonstrating that forging any component of the NOVAK Protocol reduces to breaking the collision resistance of the hash function H. These proofs include:

1. Preliminaries
2. Definition of NOVAK Primitive (PbA)
3. HVET Reduction
4. Adversarial Model Summary
5. Security Properties
6. Theorem 1 (Unforgeability of HVET)
7. Theorem 2 (PbA Enforcement Reduces to HVET)

8. EIR Security Reduction
9. RGAC Security Reduction
10. Merkle Anchoring Reduction
11. Full Reduction Summary

11.1 11.1 Preliminaries

Let:

- $R : \mathcal{D} \to \mathcal{O}$ be a deterministic rule.
- $d \in \mathcal{D}$ be data.
- $o = R(d)$ be the output.
- t be a monotonic timestamp.

Let H be a cryptographic hash function satisfying:

- collision resistance (CR),
- second-preimage resistance (SPR),
- preimage resistance (PR).

Let $\|$ denote canonical concatenation.
HVET is defined as:

$$\mathsf{HVET} = H\big(H(R)\|H(d)\|H(o)\|H(t)\big)$$

All proofs assume NOVAK Laws L0–L15 and Addenda PL-X and PS-X as axiomatic environmental constraints.

11.2 11.2 PbA Primitive Definition

A system implements Proof-Before-Action if:

$$\mathsf{HVET}(R, d, o, t) verify \mathbf{true}$$

must hold before any action executes.
PbA therefore relies on HVET unforgeability for correctness.

11.3 11.3 HVET Reduction

To forge HVET, an adversary must produce:

$$\mathsf{HVET}(R, d, o, t) = \mathsf{HVET}(R', d', o', t')$$

with:

$$(R, d, o, t) \neq (R', d', o', t')$$

Let:

$$X = H(R) \| H(d) \| H(o) \| H(t)$$

$$X' = H(R') \| H(d') \| H(o') \| H(t')$$

Then forging HVET requires:

$$H(X) = H(X')$$

There are two cases:

Case 1: $X = X'$ Then:

$$H(R) = H(R'),\ H(d) = H(d'),\ H(o) = H(o'),\ H(t) = H(t')$$

By second-preimage resistance:

$$R = R',\ d = d',\ o = o',\ t = t'$$

Contradiction.

Case 2: $X \neq X'$ Then:

$$H(X) = H(X')$$

is a collision.
Therefore:

$$\Pr[\mathsf{HVET\ forgery}] \leq \Pr[Hashcollision]$$

This serves as the basis for all subsequent reductions.

11.4 11.4 Adversarial Model

The adversary may:

- modify rules, data, outputs, or timestamps,
- reorder operations,
- inject counterfeit traces,
- replay stale states.

The adversary is polynomial-time bounded.

11.5 11.5 Security Properties

HVET provides:

- Rule Binding (RB)
- Input Integrity (II)
- Output Integrity (OI)
- Temporal Singularity (TS)
- Replay Resistance (RR)
- Pre-Execution Correctness (PEC)
- Deterministic Execution Integrity (DEI)

11.6 11.6 Theorem 1: Unforgeability of HVET

Let $\mathcal{A}$ be a PPT adversary attempting to forge an HVET. Then:

$$\Pr[\mathcal{A}succeeds] \leq \Pr[Hashcollision]$$

Thus forging HVET is at least as difficult as breaking collision resistance of H.

Immediate from the case analysis above. Any successful forge produces a collision in H.

11.7 11.7 Theorem 2: PbA Reduction

If HVET is unforgeable, then no adversary can cause incorrect execution under PbA without breaking HVET.

PbA enforces:

$$\mathsf{verify}(\mathsf{HVET}) = \mathbf{true}$$

before any action executes.

If an adversary attempts to substitute:

$$(R', d', o', t')$$

they must produce:

$$\mathsf{HVET}(R', d', o', t') = \mathsf{HVET}(R, d, o, t)$$

which is HVET forgery. By Theorem 1, this reduces to collision resistance of H, which is negligible.

11.8 11.8 EIR Security

To forge an EIR:

$$\langle R', d', o', t', \mathsf{HVET}' \rangle$$

the adversary must produce an HVET matching a false tuple. Therefore:

$$\Pr[EIRforgery] \leq \Pr[\mathsf{HVET\ forgery}]$$

EIR inherits HVET's unforgeability guarantees.

11.9 11.9 RGAC Security

RGAC links EIRs:

$$C_i = H(C_{i-1} \| \mathsf{EIR}_i)$$

To modify the chain at position i, an adversary must:

1. forge EIR_i, or
2. find a collision in $H(C_{i-1}\|\cdot)$.

Thus:

$$\Pr[RGACforgery] \leq \Pr[\mathsf{EIR\ forgery}] + \Pr[Hashcollision]$$

Both are negligible.

11.10 11.10 Merkle Anchoring

For Merkle trees:

$$\mathsf{MerkleRoot} = \mathcal{M}(\mathsf{EIR}_1, \ldots, \mathsf{EIR}_n)$$

Tampering with any leaf changes the Merkle root unless the adversary finds a hash collision.

Thus Merkle anchoring inherits:

$$\Pr[Merkleforgery] \leq \Pr[Hashcollision]$$

11.11 11.11 Full Reduction Summary

All NOVAK components reduce to the collision resistance of H:

$$\begin{array}{lcl} \mathsf{HVET} & \Rightarrow & Hashcollisionresistance \\ \mathsf{EIR} & \Rightarrow & \mathsf{HVET} \\ \mathsf{RGAC} & \Rightarrow & \mathsf{EIR} + Hashcollisionresistance \\ \mathsf{PbA} & \Rightarrow & \mathsf{HVET\ unforgeability} \end{array}$$

Therefore the security of the entire NOVAK Protocol is bounded below by the security of SHA-2, SHA-3, or BLAKE3.

Appendix A: Canonical Encoding Specification

This appendix defines the complete canonical encoding rules for NOVAK. These rules ensure that all implementations serialize R, d, o, and t identically, across platforms, languages, architectures, and time.

A.1 Encoding Philosophy

Canonical encoding prevents:

- ambiguity,
- implementation divergence,
- equivocation attacks,
- version-induced serialization drift.

Correctness requires that identical logical values produce identical byte streams in all environments.

A.2 Primitive Types

Strings. Encoded as:

$$\mathsf{encode\_str}(s) = \mathsf{len}(s) \| s_{utf8}$$

with a 64-bit big-endian length prefix.

Binary Data. Encoded as:

$$\mathsf{encode\_bin}(b) = \mathsf{len}(b) \| b$$

Integers. Canonical integer format is unsigned big-endian without leading zeros.

A.3 Structured Objects

A structured object is encoded field-by-field in fixed order:

$$\mathsf{encode\_obj}(X) = \mathsf{encode}(X.f_1) \| \mathsf{encode}(X.f_2) \| \cdots \| \mathsf{encode}(X.f_n)$$

Order MUST NOT depend on runtime map/dictionary ordering.

A.4 Rule Encoding

Rules are encoded as their complete canonical abstract syntax tree (AST) or function representation. A rule's identity must not depend on:

- whitespace,
- comments,
- compiler optimizations,
- memory layout.

The rule is hashed as:

$$H(R) = H(\mathsf{encode\_str}(R_{canonical}))$$

A.5 Data Encoding

Data d must be serialized exactly, including:

- field names,
- field ordering,
- numeric formats,
- padding rules.

A.6 Output Encoding

The same rules as data apply.

A.7 Timestamp Encoding

Time is encoded strictly using ISO 8601:

$$\mathsf{encode}(t) = \mathsf{encode\_str}(YYYY - MM - DDTHH : MM : SS.sssZ)$$

Clocks MUST be monotonic; regressions invalidate the execution.

A.8 Composite Encoding for HVET

The final HVET preimage is:

$$X = H(R) \| H(d) \| H(o) \| H(t)$$

and:

$$\mathsf{HVET} = H(X)$$

A.9 Conformance Tests

Implementations MUST pass the following checks:

- identical inputs across machines produce identical HVET values;
- rule, data, output alterations always change HVET;
- timestamp changes always change HVET;
- invalid encodings MUST reject.

This appendix defines the full canonical specification for NOVAK.

Appendix B: Extended Reduction Proofs

This appendix expands the formal proofs from Section 11 into long-form derivations suitable for peer review or academic submission.

B.1 Extended HVET Reduction

We restate the central HVET condition:

$$\mathsf{HVET}(R, d, o, t) = H(X)$$

$$where X = H(R) \| H(d) \| H(o) \| H(t)$$

Let an adversary produce:

$$(R', d', o', t') \neq (R, d, o, t)$$

and claim:

$$H(X') = H(X)$$

If $X = X'$ Then:

$$H(R) = H(R'),\ H(d) = H(d'),\ H(o) = H(o'),\ H(t) = H(t')$$

This violates the assumption that (R', d', o', t') differs. Thus:

$$X = X' \Rightarrow contradiction under SPR$$

If $X \neq X'$ Then:

$$H(X) = H(X') \Rightarrow collision in H$$

Thus unforgeability of HVET strictly reduces to collision resistance.

B.2 Extended PbA Reduction

PbA requires:

$$\mathsf{verify}(\mathsf{HVET}(R, d, o, t)) = \textbf{true}$$

Attempting to bypass PbA requires forging HVET. Therefore:

$$\Pr[PbAbypass] \leq \Pr[\mathsf{HVET\ forgery}]$$

which is negligible.

B.3 Extended EIR Reduction

The EIR structure:

$$\langle R, d, o, t, \mathsf{HVET} \rangle$$

binds all fields. To forge an EIR, an adversary must produce an HVET for a false tuple, reducing to HVET forgery.

B.4 Extended RGAC Reduction

Modifying chain element C_i requires:

$$C'_i = H(C'_{i-1} \| \mathsf{EIR}'_i)$$

To maintain consistency, the adversary must either:

- forge the EIR, or
- find a hash collision in the chaining step.

Both reduce to breaking H.

B.5 Extended Merkle Anchoring Reduction

Merkle trees have security:

$$\Pr[\mathsf{forgery}] \leq \Pr[Hashcollision]$$

Tampering with any leaf of the tree results in a mismatch of the Merkle root unless a collision is found.

B.6 Summary

Every NOVAK construction ultimately reduces to the hardness of finding a collision in the underlying hash function. Therefore NOVAK inherits:

$$Security \geq SecurityofH$$

Appendix C: Reference Implementation Hash Vectors

This appendix provides sample canonical encodings and hash outputs that serve as test vectors for implementers. These ensure cross-platform conformance.

C.1 Reference Rule

Rule (canonical form):

$$R(x) = x + 1$$

Encoded:

$$\mathsf{encode}(R) = length \| canonicalbytes$$

Sample hash:

$$H(R) = \texttt{8f3c...a192}$$

C.2 Reference Data

Input:

$$d = 41$$

Hash:

$$H(d) = \texttt{2c5f...51b8}$$

C.3 Reference Output

Output:

$$o = 42$$

Hash:

$$H(o) = \texttt{f9d4...e7aa}$$

C.4 Reference Timestamp

Timestamp:

$$t = \texttt{2025-12-01T10:15:23.123Z}$$

Hash:

$$H(t) = \texttt{57bb...0f33}$$

C.5 Reference HVET

$$X = H(R) \| H(d) \| H(o) \| H(t)$$

$$\mathsf{HVET} = H(X) = \texttt{a4bc...9f22}$$

C.6 Reference EIR

$$\langle R, d, o, t, \mathsf{HVET} \rangle$$

C.7 Reference RGAC Link

$$C_1 = H(C_0 \| \mathsf{EIR}_1)$$

$$C_1 = \texttt{3ce1...bbfa}$$

C.8 Conformance Requirement

Any compliant implementation must produce the same hash values given the same canonical encodings.