XMSS Encoding Formal Verification and Formal Verification Agent

banri / Nyx Foundation

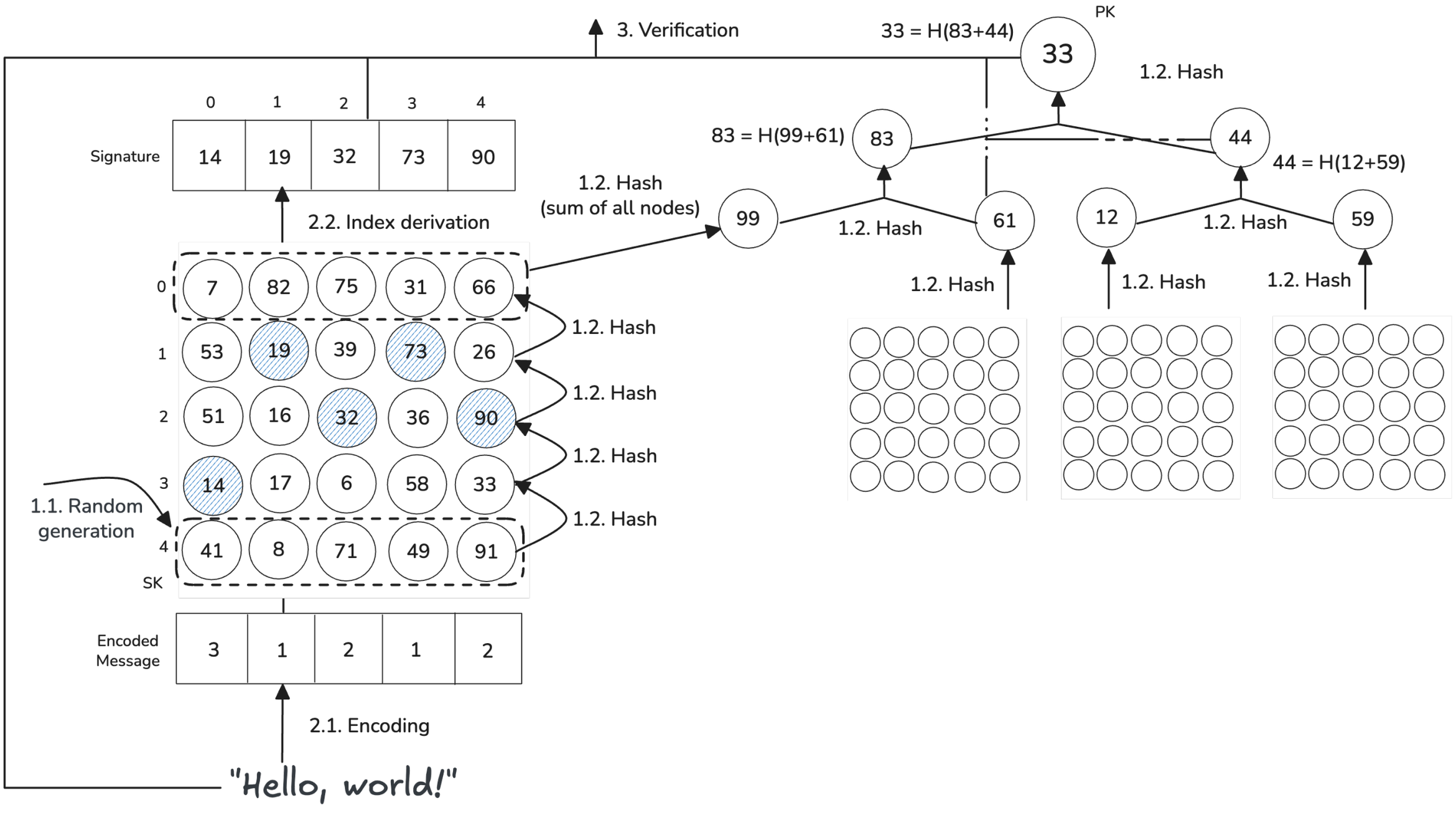
My Profile and Background

- I'm a researcher at Nyx Foundation based in Tokyo
- I majored in mathematics and I've had been an AI engineer for about 5 years
- At that time I thought that we should formally verify the cryptographic schemes that will be used in lean Ethereum
- I went ahead and did the formal verification of a part of XMSS that is a Top Single Layer (TSL) encoding scheme
- We had the opportunity to talk with Justin Drake and were invited
- I succeeded in first half of formal verification of TSL
- Actually I'm developing the formal verification agent and utilized it for that proof and we'll make it public after publishing a paper

Table of Contents

1. XMSS Encoding and TSL

XMSS


3. Verification
PK
33 = H(83+44)
33
1.2. Hash
83 = H(99+61)
83
44
44 = H(12+59)
1.2. Hash
(sum of all nodes)
99
61
12
59
1.2. Hash
1.2. Hash
1.2. Hash
1.2. Hash
Signature
0 1 2 3 4
14 19 32 73 90
2.2. Index derivation
0 7 82 75 31 66
1 53 19 39 73 26
2 51 16 32 36 90
3 14 17 6 58 33
4 41 8 71 49 91
SK
1.2. Hash
1.2. Hash
1.2. Hash
1.1. Random generation
Encoded Message
3 1 2 1 2
2.1. Encoding
"Hello, world!"

At the Top of Hypercube

At the Top of the Hypercube – Better Size-Time Tradeoffs for Hash-Based Signatures

Dmitry Khovratovich [1] Mikhail Kudinov*[2] Benedikt Wagner [1]

June 4, 2025

[1] Ethereum Foundation
{dmitry.khovratovich,benedikt.wagner}@ethereum.org
[2] Eindhoven University of Technology
mishel.kudinov@gmail.com

Abstract

Hash-based signatures have been studied for decades and have recently gained renewed attention due to their post-quantum security. At the core of the most prominent hash-based signature schemes, XMSS and SPHINCS$^+$, lies a one-time signature scheme based on hash chains due to Winternitz. In this scheme, messages are encoded into vectors of positions (i.e., vertices in a hypercube) in the hash chains, and the signature contains the respective chain elements. The encoding process is crucial for the efficiency and security of this construction. In particular, it determines a tradeoff between signature size and computational costs. Researchers have been trying to improve this size-time tradeoff curve for decades, but all improvements have been arguably marginal.

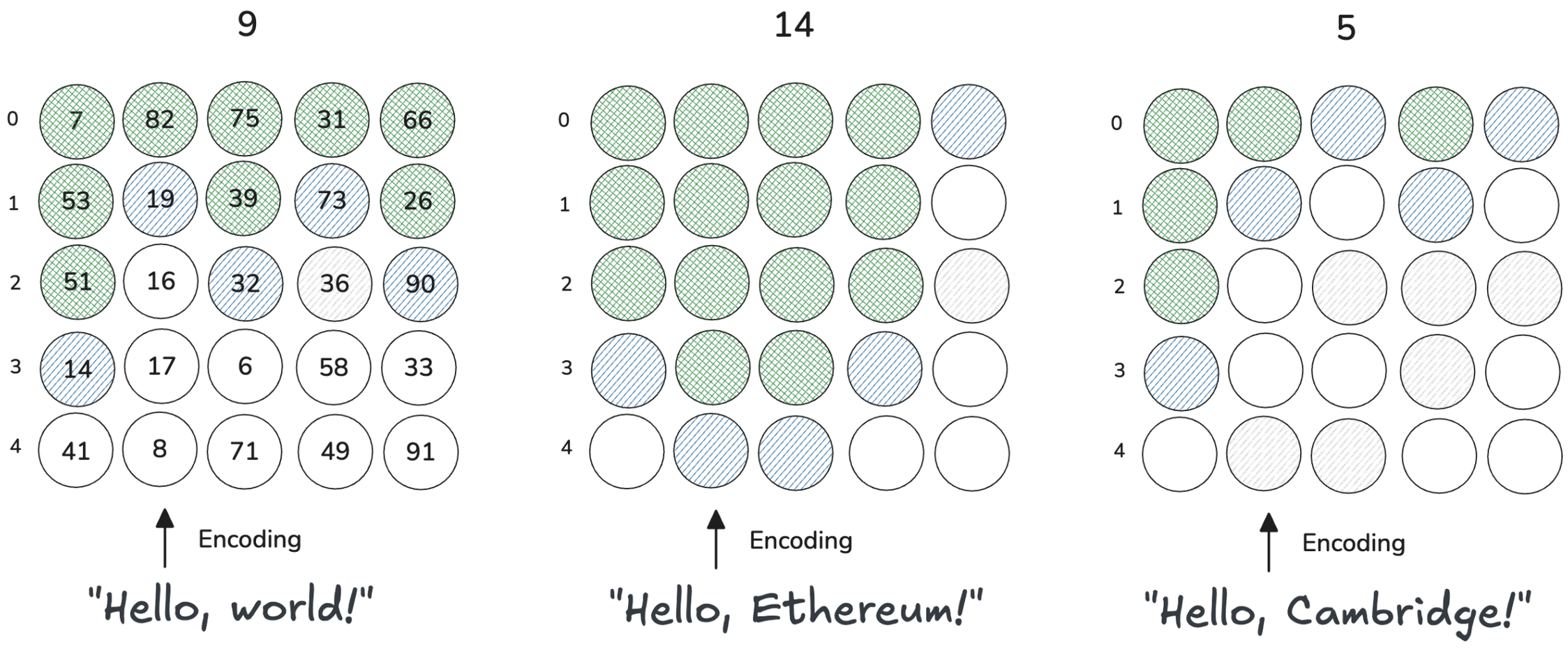
In this work, we revisit the encoding process with the goal of minimizing verification costs and signature sizes. As our first result, we present a novel lower bound for the verification cost given a fixed signature size. Our lower bound is the first to directly apply to general encodings including randomized, non-uniform, and non-injective ones.

Then, we present new encodings and prove their security. Inspired by our lower bound, these encodings follow a counterintuitive approach: we map messages non-uniformly into the top layers of a much bigger hypercube than needed but the encoding itself has (hard to find) collisions. With this, we get a 20% to 40% improvement in the verification cost of the signature while keeping the same security level and the same size. Our constructions can be directly plugged into any signature scheme based on hash chains, which includes SPHINCS$^+$ and XMSS.

Keywords: Hash-based signatures, One-time signatures, Post-quantum, Winternitz, incomparable encodings, lower bounds

XMSS Encoding

Each verification cost corresponds to how many times verifier should calculate the hash functions.



TSL (Top Single Layer)

- One of encoding schemes proposed in the paper

Construction 4 (TSL: Top Single Layer). *Let $\mathcal{M}$ and $\mathcal{R}$ be message and randomness spaces, respectively. Let λ be a parameter. Before we present the encoding, fix the following parameters:*

1. *Select* w, v *such that*[9] $\mathsf{w}^v > 2^{\lambda + \log_4 \lambda}$.
2. *Find the minimum* d_0 *such that* $\ell_{d_0} \geq 2^\lambda$.
3. *Consider a random oracle* $\mathsf{H} \colon \mathcal{M} \times \mathcal{R} \to \{0, 1, \ldots, \mathsf{w}^v - 1\}$

[9] The extra term $\log_4 \lambda$ will ensure that a suitable d_0 exists.

4. *Let* $\Psi \colon \{0, 1, \ldots, \mathsf{w}^v - 1\} \to [\mathsf{w}^v]$ *be a function such that for* $z \xleftarrow{\$} \{0, 1, \ldots, \mathsf{w}^v - 1\}$, *we have*

$$\forall\, x \in [\mathsf{w}]^v \colon \quad \Pr_z [\Psi(z) = x] = \widehat{\mu}(x),$$

where $\widehat{\mu}(x) = \begin{cases} 1/\ell_{d_0}, & \text{if } \mathrm{layer}(x) = d_0, \\ 0, & \text{if } \mathrm{layer}(x) \neq d_0. \end{cases}$

With this, we now present our encoding $f \colon \mathcal{M} \times \mathcal{R} \to [\mathsf{w}]^v$

1. *Take as input* $(m, r) \in \mathcal{M} \times \mathcal{R}$.
2. *Set* $x := \Psi(\mathsf{H}(m, r))$. *Note that* $x \in [\mathsf{w}]^v$ *and* $\mathrm{layer}(x) = d_0$.
3. *Output* x *as the encoding.*

Formal Verification of TSL

There's two propositions and the first one was done using the formal verification agent.

```
/-- The TSL encoding has a code contained in a single layer `L_{d0}`;
any two distinct outputs are incomparable (coordinatewise). -/
theorem tsl_incomparable
    {w v : Nat} {M R X : Type}
    (P : TslParams w v M R X) :
    incomparableEncoding (w:=w) (v:=v) (M:=M) (R:=R)
      (topSingleLayerEncoding P) := by
  intro x y hx hy hxy
  -- Map code-membership into the target layer `L_{d0}`.
  have hxL : x ∈ layerSet w v P.d0 := by
    exact (topSingleLayerEncoding_code_subset (w:=w) (v:=v) (M:=M) (R:=R) P) hx
  have hyL : y ∈ layerSet w v P.d0 := by
    exact (topSingleLayerEncoding_code_subset (w:=w) (v:=v) (M:=M) (R:=R) P) hy
  -- Apply Lemma 01 instantiated to the common layer `d0`.
  simpa using
    (same_layer_vertices_are_incomparable (w:=w) (v:=v) (d:=P.d0)
      (x:=x) (y:=y) hxL hyL hxy)

/-- VCVio-style target-collision bound for the TSL encoding in the ROM.
We instantiate the cached random-oracle bound (Lemma 02, VCVio variant)
with the TSL postprocessing `psi`. The pre-hash domain `X` is assumed to
be uniformly samplable via `SelectableType X` (used by the ROM). -/
theorem tsl_tcr_bound
  {w v : Nat} {M R X : Type}
  [Fintype R] [DecidableEq (M × R)] [DecidableEq X]
  [OracleComp.SelectableType X] [OracleComp.SelectableType R]
  (P : TslParams w v M R X)
  (μhat : HcV w v → Real)
  (A : VCTCRAdversary M R X) :
  let t := A.qChoose + A.qRespond
  ([= () | TCRGameROM (w:=w) (v:=v) (M:=M) (R:=R) (X:=X) P.psi A]).toReal ≤
    (t : Real) / (Fintype.card R : Real) +
    (t : Real) * ((Finset.univ : Finset (HcV w v)).sum (fun y => (μhat y) * (μhat y))) := by
  classical
  -- Directly reuse the generic VCVio ROM lemma with `ψ := P.psi`.
  simpa using
    (post_processed_ro_tc_bound (w:=w) (v:=v)
      (M:=M) (R:=R) (X:=X) (ψ:=P.psi) (μhat:=μhat) (A:=A))

end TopSingleLayer
```

2. Formal Verification Agent

Automation of Formal Verification

Formal Verification Agent: LLM agent for formal verification

- Current specification: the main phase can work automatically

- Goal specification: the whole phases can work full-automatically

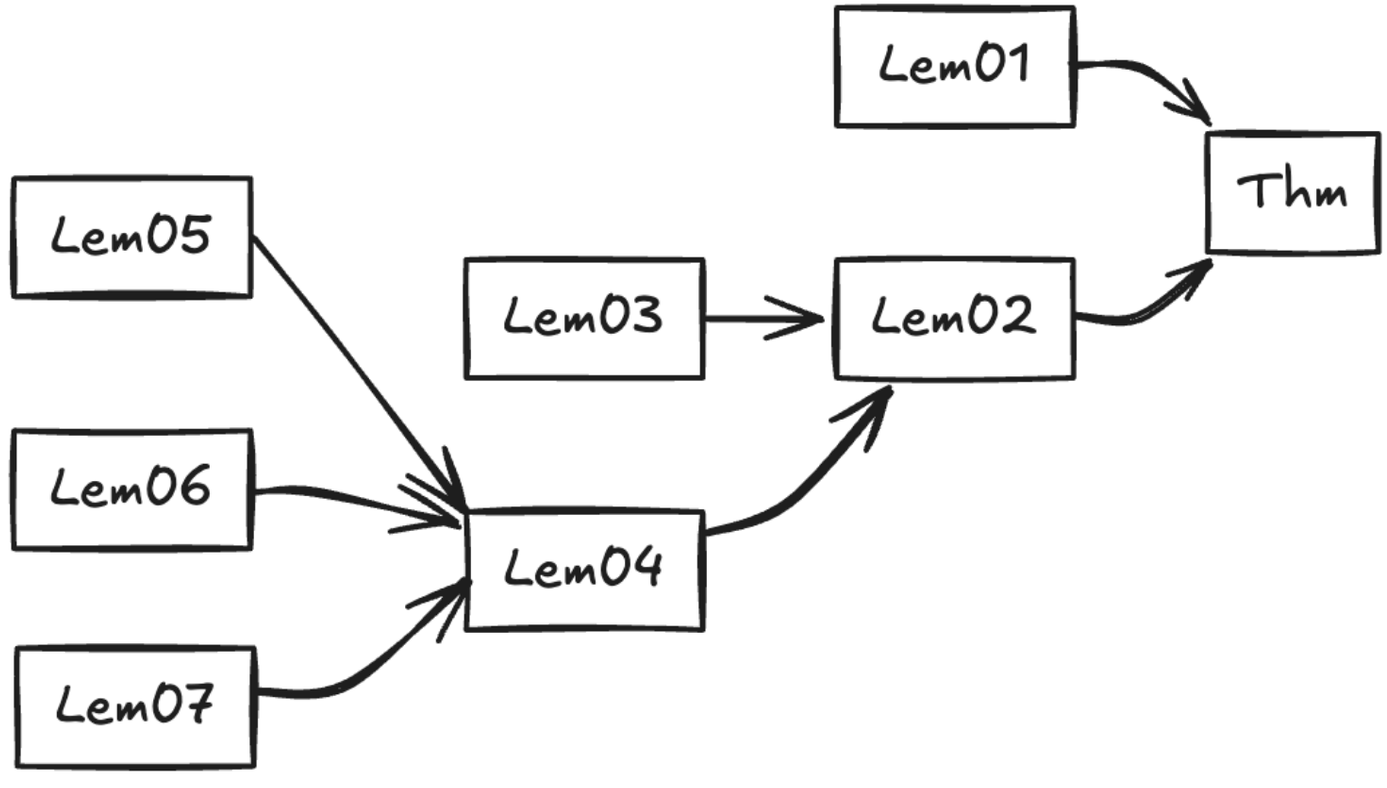
We set completing the formal verification (security proof) of TSL as the first goal.

Tech Stack

- LLM agent: OpenAI Codex CLI
  - model: gpt-5
  - reasoning: high
- Formal proof language: Lean with Mathlib, VCV-io
- Informal proof language: Markdown
- State management tool: TypeScript with Prisma, SQLite (DDD)

Divide-and-Conquer Proving

- The key concept is "Divide and Conquer".
- If the agent come across challenging parts while proving, he regard it as a placeholder and continue the proof.



Focal Theorem / Lemma

- **Focal (Artifact) Theorem**:
  - The currently focused theorem in the round.
  - With its focal lemmas assumed, its proof body is complete; hence “Artifact”.
- **Focal Stub Lemma**:
  - A minimal lemma extracted to progress the focal theorem.
  - The proof body is essentially empty; hence “Stub”.

In other words, they are "relative" theorem / lemma.

3 Phases

There's 3 phases:

1. Retrieval (manual):
   retrieve the propositions from a pen-and-paper proof

2. **Exploration (automatic):**
   do divide-and-conquer proving over multiple rounds

3. Conclusion (manual):
   make a conclusion

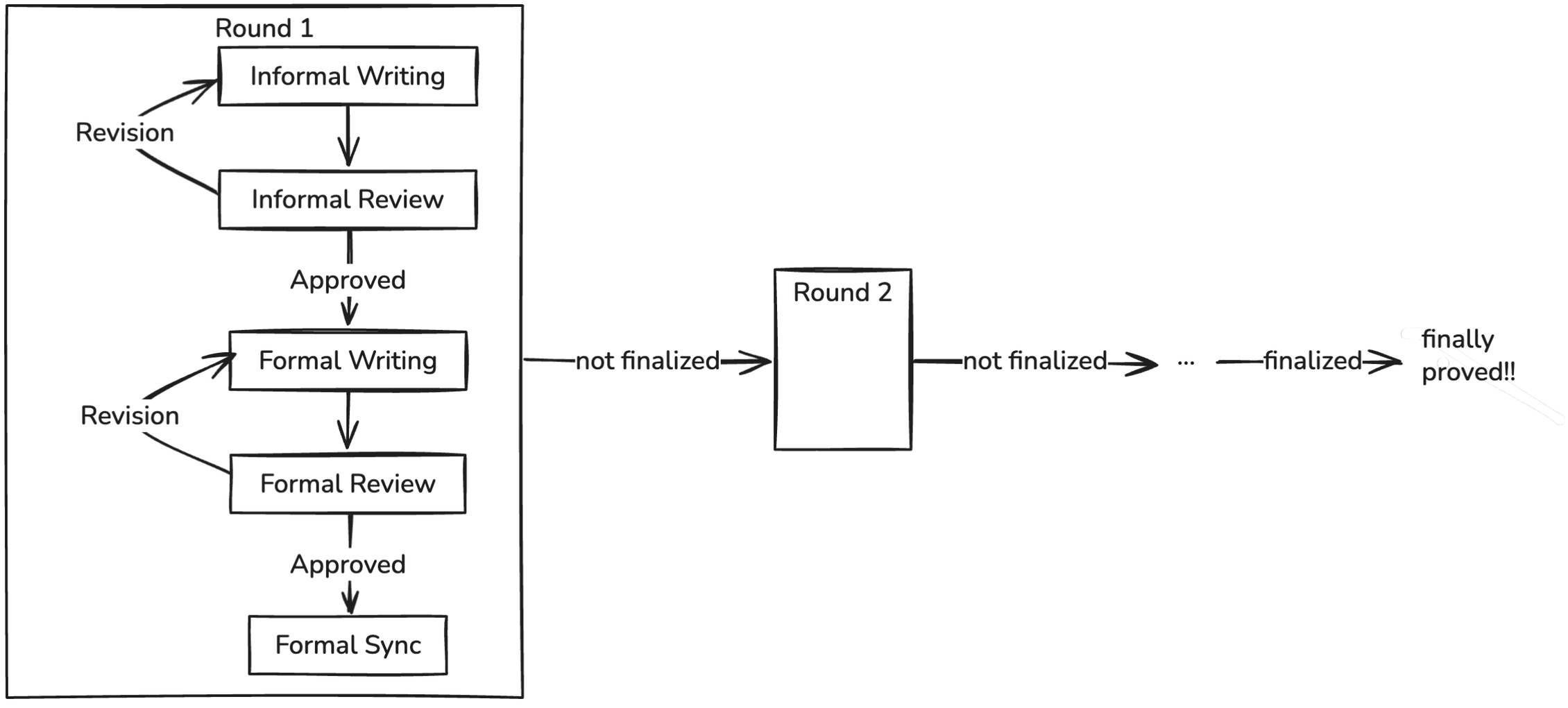
Round and 5 Stages

Exploration stage is loop of rounds.

Each round consists of 5 stages:

1. **Informal Writing**: Write the informal proof (markdown file)
2. **Informal Review**: Review the informal proof and return the result
3. **Formal Writing**: Write the formal proof (Lean file)
4. **Formal Review**: Review the formal proof and return the result
5. **Formal Sync**: Synchronize the formal proof to informal proof

Exploration Flow



Challenges

I'm working on the second half of TSL security proof.

- The second proposition is harder than first one
    - The first one is purely mathematical
      but the second one has also a cryptographic aspect
        - Cryptographic knowledge of LLM is not enough ...?
- There's some unexpected behaviors
    - Rule-based guardrails should be more managed