From Logs to Proofs

Cryptographic Compliance for Agentic Payments

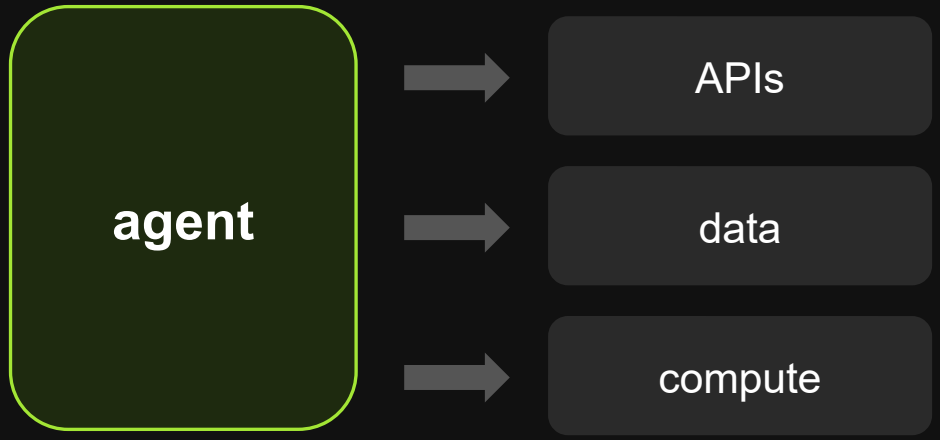
Aggre Hiroyuki (aggre) — Lemma Oracle / FRAME00, INC.

AGNTCon + MCPCon Japan 2026

a new customer

the AI agent

agents pay



under delegated authority

autonomously

no human in the loop

machine speed

continuous decisions · repeated payments

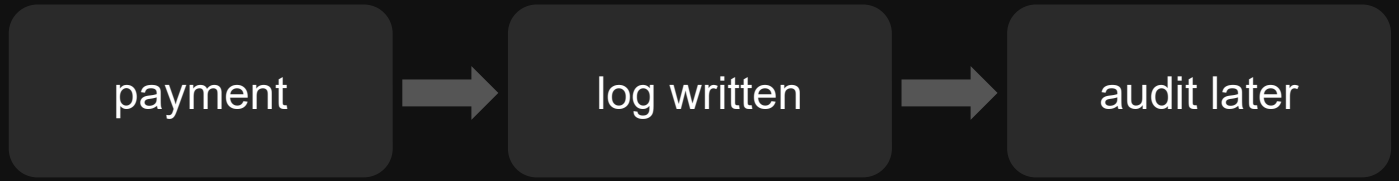
one question

who authorizes each one?

wrong, or manipulated?

a confident explanation is not permission

today: logs



A writable log can be altered or deleted. A later audit cannot undo execution.

protect the logs · enforce the action separately

the shift

Require a proof of policy compliance in the execution path.

before, not after

Target boundary · no accepted evidence, no payment authorization

where does it go?

```
const proofFetch = wrapFetchWithProof(
  fetch, artifact, gate, lemmaClient, proofOptions
);
return wrapFetchWithPayment(proofFetch, client);
```

Trust402 reference demo · buyer-side fetch composition

the stop is executable

```
mockRoleProve.mockRejectedValue(new Error("Circuit error"));
await expect(wrappedFetch("https://example.com/api"))
  .rejects.toThrow("Role proof generation failed");
expect(mockBaseFetch).not.toHaveBeenCalled();
```

Repository test excerpt · mocked prover · control-flow guarantee

$0.01 ✓

reference demo · financial data · $10 gate

$500 → stop

reference demo · role proof fails

why a proof?

reduce trust in the party producing the evidence

verify the condition itself

A signed decision authenticates the issuer’s assertion.
A policy proof lets the verifier check the encoded condition.
Keep the authority source trusted; reduce trust in the evaluator.

Signed decision
issuer says “allow”

Policy proof
condition can be checked

correctness over confidence

soundness: false statements should not pass

trust less of the agent

untrusted · agent and proof producer

protected · verifier and payment signer

Authenticate the authority source; pin the circuit and verification key.

the rule becomes a constraint

```
roleHash === requiredRoleHash;

component leq = LessEqThan(128);
leq.in[0] <== spendLimit;
leq.in[1] <== maxSpend;
leq.out === 1;
```

role-spend-limit-v1 · actual Circom excerpt · intended integer domain

bind authority, not just labels

The proven fields must belong to accepted delegated authority.

A new hash chosen by the requester is not an authorization grant.

verify it yourself

```
import { groth16 } from "snarkjs";
const valid = await groth16.verify(
  verificationKey, publicSignals, proof
);
if (!valid) throw new Error("Invalid proof");
```

Standard verification API · the private witness is not an argument

enforce the verified payment

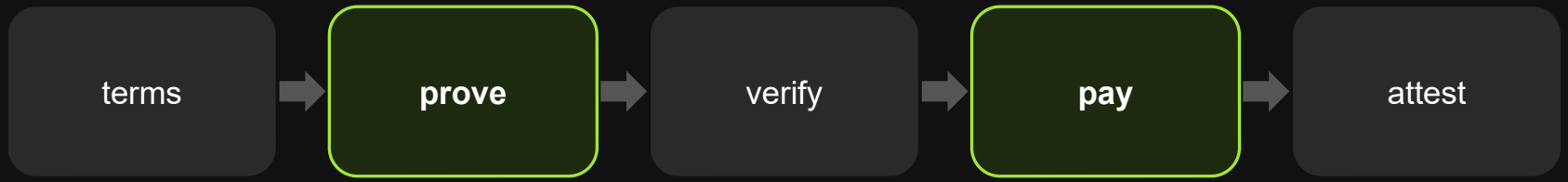
```
const terms = canonicalPayment(challenge);
const statement = statementFor(terms, delegation);
await verifyEvidence(proof, statement, trustedConfig);
await authorizeAndReserve(statement, currentState);
return signer.signAndPay(terms);
```

Target design pseudocode · same terms through verification and signing

failure must not become permission

Alter the log: the payment condition stays unchanged.
Alter the proof: verification must reject.
Delete the proof: the protected action must stop.

one payment, one evidence trail



Target lifecycle · bind evidence to this payment

Short-lived authorization; retain proof, public inputs and the settlement reference.

evidence survives the boundary

partners and auditors can check the same statement

verify more. disclose less.

zero knowledge keeps private inputs out of verification

where should proving run?

Proving belongs in the latency and trust budget.
Measure generation, verification and network time separately.
Local and hosted proving expose different operational risks.

Local prover
control witness access
operate compute

vs

Hosted prover
offload compute
trust witness handling

budget is state

a per-call ceiling is not a remaining balance

version the policy and the proof system

Pin the circuit, verification key and public-input schema.
Test migrations against the same policy outcomes.
Post-quantum migration is engineering work, not a free switch.

govern the boundary

Policy owners approve rules; operators protect keys and verifiers.
Review circuit changes like changes to authorization code.
Monitor failures, preserve receipts, and define recovery paths.

reference today · deployment next

The open source makes the mechanism inspectable.
Production assurance must be established for the whole path.

Reference today
fetch gate + tests
circuit + demo

→

Deployment work
issuer/payment binding
verifier + signer isolation

test errors and adversarial inputs

```
forged evidence    => reject
changed payment    => no signature
missing evidence   => no signature
replayed request   => no second payment
concurrent spend   => no budget overrun
```

Target acceptance criteria · evaluate the protected signing path

who authorizes each one?

a protected boundary verifies evidence and policy

logs + proofs

less trust in producers · less disclosure · reusable evidence

run it. inspect it.

github.com/lemmaoracle/trust402

lemma.frame00.com/trust402

Q&A · what the circuit does not establish

Role equality is not proof of issuer-authorized role membership.
nowSec is public but unused by this role circuit.
Review numeric ranges and the binding of fields to credential roots.

Q&A · what must remain trusted

Correct circuit, secure setup, pinned verifier and accepted authority roots.
Exact payment binding, fresh policy state and an isolated signer.
Soundness protects correctness; privacy and availability need their own controls.