

Zero-Knowledge Predicate Proofs Between AI Agents: A Measured, Cross-Protocol Gateway and the Source-Integrity Gap

Ashok Subbabbhatta Gopalakrishna

Abstract—Multi-agent AI platforms move quickly from staging to production, but the way agents establish trust remains rudimentary: an agent either transmits raw data to a peer or accepts that peer’s natural-language self-report that a value complies with policy. The first over-shares; the second is unverifiable and is exactly the channel prompt injection attacks. Prevailing responses emphasise identity, visibility, and post-hoc detection, and recent proposals for cryptographically enforced agent policy have been evaluated in simulation rather than execution. We take provable data minimisation between agents from proposal to running system. In our Zero-Knowledge Proof Gateway, agents exchange proofs of governance-defined predicates over private data rather than the data itself, so exposure is prevented by design rather than detected afterwards; because no interoperability protocol can carry such a proof, we propose a slot and implement it on both MCP and Agent2Agent from one endpoint. A 32-bit threshold predicate proves in 6.2ms and verifies in 1.0ms with a 608-byte Bulletproofs proof on one commodity vCPU; eleven adversarial experiments and nineteen protocol checks pass; and the system is deployed to Kubernetes with empirically verified network isolation. Our case study proves a retail client order is within its limit without revealing the amount, instantiating the GDPR data-minimisation principle as an enforced technical measure of the kind EU law now names explicitly. We then address the limitation no comparable work resolves: a predicate proof binds a statement to a committed value, never to the system of record. We give a construction fusing an enclave attestation with the proof in both directions, so verifying one artifact certifies jointly that the predicate holds and that the value was read by a specific measured binary, and test it against a mock authority.

Index Terms—Zero-knowledge proofs, multi-agent systems, data privacy, privacy preservation, cryptography, agentic security.

I. INTRODUCTION

MULTI-AGENT AI systems, in which planner agents delegate to specialist agents that invoke tools, and then transfer tasks to other agents across security boundaries, are being developed at a pace faster than the security architecture beneath them. The industry surveys from early 2026 quantify the gap: in one study of 919 security executives and practitioners, 88% of organizations reported a confirmed or suspected AI-agent security incident within the prior year, while 82% of them simultaneously expressed confidence that existing policies guard against unauthorized agent actions, and only about one in five organizations had runtime visibility at

all into what their agents access [1]. An independent survey of 300 enterprise leaders found 97% expecting an agent-driven security or fraud incident within twelve months [2]. Agent fleets have roughly doubled in recent months while confidence in their security has risen faster than the controls that would justify it [1].

The risk underneath these numbers has been termed in the industry as the *lethal trifecta* [3], [4]: an agent that (i) has access to private data, (ii) is exposed to untrusted input, and (iii) can communicate externally or forward data onward can be exploited to exfiltrate data by successful prompt injections. Every inter-agent link in a pipeline compounds the trifecta, because today an agent establishes trust boundaries with peers in one of two ways: it ships the raw data (“here is the order information, check if its value is within the policy”), or it accepts a natural-language claim from the peer (or from itself) (“I verified; the value is within policy”). The first maximizes exposure; the second is unverifiable, non-deterministic, and provides no meaningful non-repudiation.

The security community’s response operates from a different perspective. The OWASP Top 10 for Agentic Applications [5], MITRE ATLAS [7], and Zero Trust reference architectures for agents [8], [9] focus on *who* may act (identity, principle of least privilege), *what happened* (visibility, audit logging), and *detecting* bad actors or agents at runtime. These controls focus primarily on which principal may cross a boundary; they do not control *what data* crosses the boundary when a permitted interaction occurs. A successfully authenticated, fully logged agent that sends a salary or a diagnosis to a peer that only needed a yes/no policy answer has still over-shared, and detection-based tooling will notice the share after the fact.

This paper proposes a complementary control: *provable data minimization between agents*. Data minimisation is not merely a design preference in the European regime; it is a legal requirement under GDPR Article 5(1)(c), and Article 25 asks that it be realised through technical measures rather than assurances [24]. Instead of data or text, agents exchange zero-knowledge proofs (ZKPs) of governance-defined predicates (range membership, set membership, threshold checks, and Boolean compositions) over their private data. The consumer agent learns exactly one bit (the predicate holds) plus a binding commitment and an audit handle; the sensitive value never crosses the boundary at all.

In this work, we make the following contributions:

- **An implemented and measured system, not a simulation.** Prior proposals for cryptographic agent-policy enforcement report proving costs drawn from calibrated distributions rather than execution [31]. Every figure here is measured from a single logged run of open-source code, reproducible in one command.
- **A threat model for inter-agent data exposure vulnerability** (Section II) that separates the failure modes of natural-language self-report (prompt-injected claims, non-deterministic verification, and lack of non-repudiation) and maps the pipeline stages to exposure and mitigation.
- **The Proof Gateway architecture** (Section IV): a governance-owned predicate registry, a verifying gateway, prover sidecars, a hash-chained audit log, and a concrete protocol extension (`zk-attach/v0`) that adds a proof slot to tool calls and handoffs, which no current agent protocol provides [20]–[22]; we realise it on both MCP and Agent2Agent over a shared verification path, showing the carriage mechanism is protocol-agnostic rather than a property of any one protocol.
- **A regulated case study under EU law** (Section V): an execution agent proves a retail client’s order notional satisfies a pre-trade cap without revealing it, instantiating the data-minimisation principle of GDPR Article 5(1)(c) and the data-protection-by-design obligation of Article 25 [24] as an enforced technical measure rather than a policy commitment, in the manner EU law now explicitly contemplates for predicate disclosure over personal data [26].
- **A reference prototype and adversarial evaluation** (Section VI): two interchangeable engines (an auditable Sigma-protocol baseline and Rust Bulletproofs [17], [18]), microbenchmarks across bit-widths and aggregation levels, a latency-budget analysis, eleven adversarial experiments covering tampering, replay, rogue predicates, and forgery, and a Kubernetes deployment whose prover isolation was verified by observation rather than assumed from manifests. The implementation, the logged reproduction run, and all figures are released at <https://github.com/45h0kg/zk-proof-gateway>.
- **Attestation-bound predicate proofs** (Section VII): a construction that mutually binds an enclave attestation and a predicate proof, so neither can be substituted for the other, together with a governance-signed `prover_measurement` predicate type that places prover identity under the same signing authority as the business rule it accompanies. We implement the construction and its six-step verification chain, wired into the live governed path on both protocol surfaces, and test it against a mock attestation authority. We regard source integrity, rather than proving cost, as the binding limitation of cryptographically enforced agent policy.

Section VIII states the limitations and open problems, including the explicit exclusion of verifiable LLM inference (zkML), Section IX places the work against the nearest prior art, and Section X is the conclusion.

II. PROBLEM SETTING AND THREAT MODEL

A. Trust Domains and Actors

Our primary environment is a *single-organization multi-agent system*: agents operated by one enterprise but spanning different trust boundaries (trading desk, risk, HR, procurement), where internal least-privilege policies apply between agents similar to how they apply between departments. A secondary environment, *cross-organization* agent-to-agent interaction, gears up the motivation (the peer is another company) but adds identity-federation and legal machinery that is out of scope here; the proof layer is identical in both.

Actors: a *prover agent* holding private data fetched from a source of truth; a *relying (verifier) agent* that must establish a policy fact; a *gateway* operated by the platform; a *governance team* defining policy predicates; and an *adversary* who may fully compromise the prover agent’s LLM context (prompt injection), tamper with messages in transit, replay old messages, attempt to register weakened predicates, and edit logs after the fact. We assume the gateway and the governance signing keys are not compromised (the gateway is small, deterministic, and auditable, which is the reason the LLM is not part of the verification path), and we also assume standard cryptographic hardness assumptions (discrete logarithm; random-oracle Fiat–Shamir [13]).

B. Exposure Across the Pipeline

Table I maps the stages of a contemporary agent pipeline to their current exposure and the proposed mitigation.

C. Why Proofs Rather Than LLM Self-Report

The alternative to sending the data is today usually an LLM’s own assertion. There are three failure modes that make such assertions unsuitable as a trust primitive, and each is directly addressed by verifiable proofs:

1) *Prompt injection compromises the claim*: An agent whose context contains untrusted input can be induced to assert compliance falsely; the assertion channel and the attack channel are the same channel [3]–[5]. A ZKP moves the trust verification out of the LLM entirely: soundness holds against a *fully malicious* prover, so a compromised agent can falsify the prose but cannot produce a verifiable proof of a false predicate (Section VI, tests T1/T7).

2) *Non-determinism and hallucination on verification tasks*: LLM outputs are probabilistic, vary across samples, and are not reliable on precise numeric checks; a verification check implemented as “ask the model” has no soundness parameter. Proof verification is a deterministic algorithm with soundness error negligible in the security parameter [10].

3) *No non-repudiation or auditability*: A natural-language claim binds no agent: it names no precise statement, no policy version, and can be disavowed. A proof is bound, through the Fiat–Shamir transcript, to a specific predicate identifier *and version*, a request nonce, the reference of the action being authorized, and the two agent identities, and the audit log stores the commitment and proof hash; any interested party can later re-verify the exact claim that was accepted (tests T3/T4/T6).

TABLE I
INTER-AGENT DATA EXPOSURE ACROSS PIPELINE STAGES: CURRENT PRACTICE VS. PROPOSED PROOF-GATEWAY MITIGATION.

Pipeline stage	Current exposure	Proposed mitigation
Planner → sub-agent delegation Tool call	Full task context, including sensitive fields, copied into the sub-agent's prompt Arguments serialized in the clear into the tool schema; logged by middleware and providers	Delegate the task with predicate proofs replacing sensitive fields; sub-agent receives policy facts, not values zk-attach/v0 attachment carries commitment + proof; sensitive argument omitted or replaced by a reference
Agent → agent hand-off	Raw records or natural-language summaries cross the boundary	Gateway-verified predicate result + audit handle crosses; data does not
Memory / RAG store	Values persisted in shared vector stores or scratchpads readable by later, differently-privileged sessions	Persist commitments and proof outcomes; re-verification is possible without re-exposure
Audit layer	Logs replicate the sensitive values a second time (log-store becomes a shadow database)	Hash-chained entries bind commitment, proof hash, predicate id@version, agents, and result, with no values

III. PRELIMINARIES

A. Zero-Knowledge Proofs

An interactive proof system for a language L between prover P and verifier V satisfies [10]: *completeness* (for $x \in L$, honest P convinces V except with negligible probability); *soundness* (for $x \notin L$, no cheating prover convinces V except with negligible probability); and *zero-knowledge* (for $x \in L$, every verifier's view can be simulated without the witness, so the interaction reveals nothing beyond the truth of the statement). Goldreich, Micali, and Wigderson showed every language in NP admits a zero-knowledge proof [11], which grounds our use of Boolean *compositions* of atomic predicates.

B. Sigma Protocols and Fiat-Shamir

A Sigma protocol is a three-move (commit–challenge–respond) proof of knowledge; Schnorr's identification protocol [12] is the canonical instance, proving knowledge of x with $y = g^x$ via $(a=g^w, e, z=w+ex)$ and check $g^z = ay^e$. The Fiat-Shamir transform [13] replaces the verifier's random challenge with a hash of the transcript, yielding non-interactive proofs in the random-oracle model; the hash input must bind the *entire* statement and context to avoid malleability [14]. OR-composition of Sigma protocols [15] proves that at least one of two statements holds without revealing which; this is the building block of the bit proofs proposed. A small-number Schnorr transcript is given in Appendix A.

C. Pedersen Commitments

A Pedersen commitment [16] to v with blinding r over a group of prime order q with independent generators G, H is $C = vG + rH$. It is *perfectly hiding* (for any v , C is uniform over the group), *computationally binding* under the discrete-log assumption, and *additively homomorphic*: $C_1 + C_2$ commits to $v_1 + v_2$. Homomorphism is what allows a verifier to derive a commitment to $\text{cap} - v$ from a public cap and C_v without learning v , which is the basis of the threshold predicate here.

D. Range Proofs and Bulletproofs

A range proof shows a committed v lies in $[0, 2^n)$. Bulletproofs [17] achieve proof size $2\lceil \log_2 n \rceil + 9$ group/field elements, logarithmic in n , with no trusted setup, under

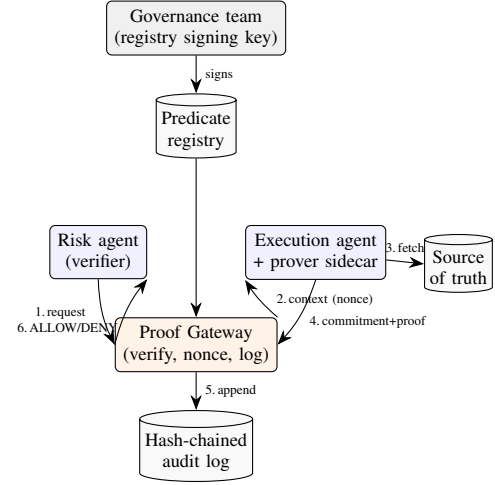


Fig. 1. Proof Gateway architecture. Sensitive values remain inside the prover's trust cell (right); only commitments and proofs cross agent boundaries.

the discrete-log assumption, made non-interactive via Fiat-Shamir. The original work reports a 688-byte single 64-bit proof (secp256k1 encoding) verifying in 3.9ms, with batch verification amortizing to 470 μ s, and aggregation of m proofs adding only $O(\log m)$ elements [17]; the dalek ristretto255 implementation realizes 32(9 + 2 $\log_2 n$) bytes, i.e., 672 bytes at 64 bits [18]. Bulletproofs+ tightens the 64-bit proof to 576 bytes with comparable computation [19]. We adopt Bulletproofs as the practical mechanism because financial predicates over 32–64-bit integer amounts are well matched to its performance profile.

IV. METHOD: THE PROOF GATEWAY

A. Components

Fig. 1 shows the architecture. Five components cooperate:

1) *Predicate registry (governance-owned)*: A catalog of predicate definitions, each comprising an identifier, version, type (range_leq, set membership, Boolean composition), parameters (cap, bit-width, unit), and owner, and each signed by a governance key. This ownership decision is in itself a security control: if agents could define their predicates, a compromised agent would simply register a loose check (e.g., a cap of 2^{31}) and comply with it. In the prototype the gateway verifies the governance signature both on publication and again

on every read, so registry tampering is identified at use time (test T5).

2) *Proof gateway*: A light, deterministic, LLM-free service that issues fresh request contexts (nonce, request id, predicate id@version, requester and prover identities, timestamp), verifies submitted proofs against registry-defined parameters, and appends audit entries. It is the only component trusted for verification, and it contains no language model.

3) *Prover sidecar*: A proof engine located alongside the data holder, exposing `prove(predicate, context)`. Engines are plugged behind one interface; the prototype has a Sigma-protocol baseline and a Bulletproofs engine (Section VI).

4) *Audit log*: Append-only entries $h_i = \text{SHA256}(h_{i-1} \parallel \text{entry}_i)$ binding the commitment, proof hash, predicate id@version, agent identities, engine, and result. Verification is replayable forever; the sensitive value appears in no entry (test T6).

5) *Requesting and providing agents*: LLM agents remain in the loop for orchestration but are removed from the trust path.

B. Protocol Flow

The end-to-end flow is: request $\rightarrow$ context issuance $\rightarrow$ source fetch $\rightarrow$ commitment $\rightarrow$ proof generation $\rightarrow$ verification $\rightarrow$ audit append $\rightarrow$ decision (Fig. 3). Two properties are enforced by construction. *Context binding*: the Fiat-Shamir transcript absorbs the full request context, including the reference of the specific action being authorized (e.g., the order identifier), before any commitment, so a proof verifies only for the exact request and action that solicited it; replay under a different nonce, action, agent pair, or predicate version fails (tests T3/T4). *Freshness*: nonces are gateway-issued, so a prover cannot pre-compute proofs for future requests unless the deployment explicitly allows pre-staging (Section VI-C).

C. Source Integrity: What a ZKP Does and Does Not Bind

A range proof binds the statement to the *committed* value; it cannot by itself show the committed value equals the system-of-record value. Two deployment topologies close this gap. **(A) Source-side prover (preferred)**: the prover sidecar is alongside the source of truth; the co-location itself is the primary defense, as the orchestrator agent never holds the value, and the source furthermore signs the commitment to bind it to the system of record for transport. **(B) Agent-side prover with data attestation**: the source returns the value together with a signature over the commitment opening; not as secure (the agent can see the value locally) but deployable without source-system changes, and it still removes the value from all *inter-agent* traffic, which is the scope of this paper. In either topology the guarantee also assumes the governed channel is the only path to the protected action: middleware must deny by default any governed action lacking a valid attachment, because no proof system constrains an endpoint that remains reachable without one.

D. The zk-attach/v0 Protocol Extension

No mainstream agent-interoperability protocol carries a proof slot: the Model Context Protocol standardizes tool discovery and invocation over JSON-RPC and delegates authorization to OAuth-style flows [20]; Agent2Agent agent cards declare identity and authentication schemes for message parts [21]; function-calling schemas type tool arguments in the clear [22]. All authenticate *principals*; none can attest *data properties*. We propose a backward-compatible attachment carried in MCP `tools/call` parameters or as an A2A message part, and implement both surfaces over a single endpoint (Section VI):

```
"zk_attachment": {
  "schema": "zk-attach/v0",
  "predicate_id": "pretrade_notional_cap",
  "predicate_version": 1,
  "engine": "bulletproofs-ristretto255",
  "context": {
    "request_id": "...", "nonce": "...",
    "requester": "...", "prover": "...",
    "action_ref": "ord-9912", "ts": 0,
    "proof_b64": "..." }
}
```

Fig. 2. Proposed proof-attachment slot for tool calls and handoffs. The sensitive argument is omitted or replaced by a reference; absence of a required attachment for a registered predicate is deny-by-default. The attestation payload of Section VII is an optional member of this same envelope, not a new schema version; see Section VII-D.

E. Composition Across Multi-Hop Chains

Delegation chains raise the question of where verification happens. **(a) Chained per-hop proofs**: each hop requests and verifies its own proof; latency adds per hop, but the audit chain records every boundary at full granularity. **(b) Aggregated proof at a trust boundary**: m range statements are proved in a single Bulletproofs aggregate whose size grows only $O(\log m)$ [17]; one verification clears the boundary, at the cost of coarser audit granularity and prover-side work that grows linearly in m . Section VI quantifies the tradeoff: at $m=16$, one 928-byte aggregate replaces 10,752 bytes of individual proofs and cuts total verification time roughly in half, while proving cost (~ 173 ms on the test vCPU) argues for asynchronous aggregation rather than in-loop use.

F. Scalability and Decentralized Verification

The gateway is logically central because policy definition and audit belong in one governed place, but verification itself is stateless and embarrassingly parallel: any replica holding the governance public key can verify any proof, so the verification role can be scaled horizontally or pushed into verifier-side sidecars that are as deterministic and model-free as the gateway itself. Decentralizing verification is sound only if audit completeness is preserved, for example through signed verification receipts returned to a shared append-only log; otherwise a compromised relying party could verify, act, and never log. The audit chain head is the one serialization point, addressable with per-domain chains or a transparency-log design. Policy definition remains centralized by design, since that centralization is the governance control of Section IV-A, not a scalability accident.

V. INSTANTIATION: CLIENT LIMIT COMPLIANCE WITHOUT DISCLOSURE

A. Scenario and Regulatory Grounding

A trading-execution agent prepares an order for a retail client. Before submission, a risk agent must establish that the order notional does not exceed a per-client cap. Such client-level limits are ordinary risk and client-protection practice; the point of interest here is not that the limit exists but what enforcing it costs in disclosed personal data.

The regulatory interest for this paper, however, is not the control itself but the data that crosses the boundary while it is enforced. A retail client’s order notional is personal data: it relates to an identified natural person. GDPR Article 5(1)(c) requires that personal data be adequate, relevant, and limited to what is necessary for the purpose, and Article 25 requires controllers to implement appropriate technical measures giving effect to data minimisation by design and by default [24]. A risk agent that must establish only whether a threshold is respected does not need the amount; transmitting it is processing beyond what the purpose requires. The conventional implementation nonetheless sends the value and relies on organisational controls, which is a policy commitment rather than a technical measure. A predicate proof discharges the same control while making the minimisation property structural: the amount is not withheld by agreement, it is absent from the exchange.

That predicate proofs are the right instrument for this is not only our view. Regulation (EU) 2024/1183, which establishes the European Digital Identity Wallet, directs Member States to integrate privacy-preserving technologies, naming zero-knowledge proof, so that a relying party can validate whether a statement based on a person’s data is true *without revealing any data on which that statement is based* [26]. That is a description in EU law of the mechanism this paper carries between agents; the same construction that answers “is this person over eighteen” without disclosing a birth date answers “is this order within the client’s limit” without disclosing an amount. The predicate class is identical, and only the attribute changes.

Two further EU instruments bear on the audit component. GDPR Article 22 constrains solely automated decision-making affecting individuals, and the AI Act’s record-keeping provisions for high-risk systems require that such systems technically permit automatic logging of events over their lifetime [25]. We note the scope precisely rather than overclaiming: the AI Act obliges logging capability, not cryptographic tamper-evidence, and the high-risk obligations phase in on a schedule that has been amended since adoption. The hash-chained audit here therefore exceeds the stated baseline rather than being compelled by it, and its value is that a decision can be re-verified later without retaining the personal data that justified it. In the concrete instance here the governance team publishes predicate `pretrade_notional_cap@v1: notional_cents ≤ 109` (a \$10M cap) over 32-bit values; the private notional is \$7.35M.

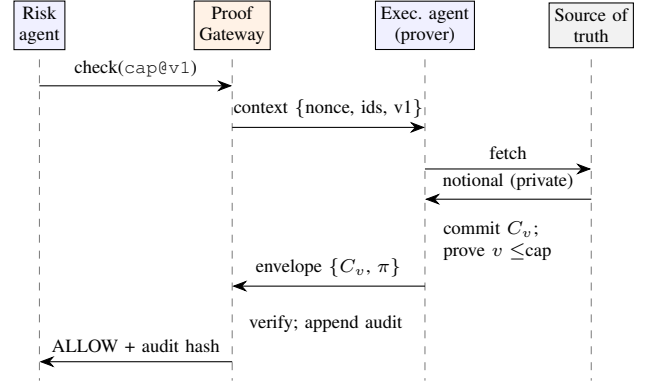


Fig. 3. Case-study sequence. The notional never crosses the dashed boundary between the prover’s trust cell and the rest of the system.

B. Predicate Construction

The statement is

$$\text{PoK}\{(v, r) : C_v = vG + rH \wedge 0 \leq v \leq \text{cap}\}. \quad (1)$$

Both engines reduce $v \leq \text{cap}$ to two range statements via the homomorphism: the verifier computes $C_d = \text{cap} \cdot G - C_v$ itself, and the prover shows $v \in [0, 2^n)$ and $d = \text{cap} - v \in [0, 2^n)$. In the Sigma baseline, each bit b_i of v is committed as $C_i = b_i G + r_i H$ and proved to be 0 or 1 with a Cramer–Damgård–Schoenmakers OR-composition [15] of two Schnorr proofs under a single Fiat–Shamir challenge; the value commitment is *defined* as $C_v = \sum_i 2^i C_i$, and the bit blindings of d are constrained so $\sum_i 2^i C_{d,i} = C_d$ exactly, eliminating any separate linking proof. In the Bulletproofs engine the same two statements are proved directly (or as one 2-aggregate). In both, the transcript absorbs the request context before commitments, binding the proof to this predicate version, nonce, order reference, and agent pair.

C. What Crosses the Boundary

The wire envelope contains the schema tag, predicate id and version, engine identifier, the request context, a Pedersen commitment, and the proof (608 bytes in the Bulletproofs engine). The notional, the order, and any position data appear nowhere in the envelope or in the audit entry; the risk agent receives ALLOW and an audit handle. In the negative scenario, in which an agent’s notional is \$12.5M, the honest prover API refuses to generate a proof at all because the predicate is false, and Section VI shows that a *malicious* prover cannot succeed either.

VI. EXPERIMENTS

A. Experimental Setup

We implemented the architecture end-to-end: a governance-signed registry, gateway, hash-chained audit log, the `zk-attach/v0` envelope, and two engines: (E1) an auditable ~300-line pure-Python Sigma-protocol engine (Pedersen bit decomposition over `secp256k1`, CDS OR-proofs, Fiat–Shamir), and (E2) the Rust dalek Bulletproofs implementation over `ristretto255` [18]. E1 exists for auditability and pedagogy;

TABLE II

RANGE-PROOF ENGINES, SINGLE STATEMENT $v \in [0, 2^n)$ (MEDIAN).

Engine	n	Size	Prove	Verify
Bulletproofs (E2)	8	480 B	1.9 ms	0.42 ms
	16	544 B	3.3 ms	0.62 ms
	32	608 B	6.2 ms	1.02 ms
	64	672 B	11.5 ms	1.78 ms
Sigma bit-OR (E1)	8	7.1 KB	191 ms	194 ms
	16	14.1 KB	362 ms	376 ms
	32	28.3 KB	749 ms	766 ms
	64	56.5 KB	1.45 s	1.50 s

TABLE III

AGGREGATION (PATTERN (B)): m 64-BIT STATEMENTS, ONE BULLETPROOFS PROOF, VS. m SEPARATE PROOFS.

m	Agg. size	m singles	Agg. verify	m verifies
1	672 B	672 B	1.8 ms	1.8 ms
2	736 B	1,344 B	2.9 ms	3.6 ms
4	800 B	2,688 B	4.3 ms	7.1 ms
8	864 B	5,376 B	7.4 ms	14.2 ms
16	928 B	10,752 B	13.7 ms	28.5 ms

E2 is the practical path. All measurements below are from a single logged run on one virtualized vCPU (medians). The environment, as reported by the container, is a single-core Intel Xeon (2.10 GHz reported clock), 4 GB RAM, Linux kernel 6.18 (Ubuntu 24.04), Python 3.12, and rustc 1.75 with `opt-level=3, lto`; because the host is virtualized, the exposed CPU string is not a physical part number and absolute timings will vary by host. These are feasibility evidence, not production benchmarks, and are consistent with published figures [17]–[19]. The full reproduction log ships with the reference implementation.

B. Results

Table II confirms the size asymptotics empirically: E2 grows by 64 bytes per doubling of n (exactly $32(9 + 2\log_2 n)$ bytes, matching [18]), while E1 grows linearly. Table III quantifies composition pattern (b): at $m=16$ the aggregate is $11.6\times$ smaller and verifies about $2.1\times$ faster than sixteen singles, while prover time grows roughly linearly (173 ms at $m=16$), supporting boundary-level aggregation computed asynchronously. Fig. 4 and Fig. 5 visualize both.

C. End-to-End Case Study and Latency Budget

Running the Section V scenario end-to-end (registry lookup with signature re-check, context issuance, prove, verify, audit append, in-process) yields p50 1,514 ms / p95 1,538 ms with E1; the same flow is bounded by ≈ 6 ms prove + 1 ms verify + I/O with E2. Mapped to pipeline locations: a real-time tool-call loop with a 10–50 ms budget absorbs E2 comfortably, and proof generation can additionally be made asynchronous or pre-staged where the deployment permits nonce pre-issuance; agent handoffs with 100 ms–1 s budgets absorb even two-sided prove+verify with margin; audit-time re-verification is 1–2 ms per entry. E1-class engines are prohibitive everywhere except offline audit and should be regarded as pedagogical.

ZK Proof Gateway — range-proof engines, single vCPU Xeon 2.8 GHz

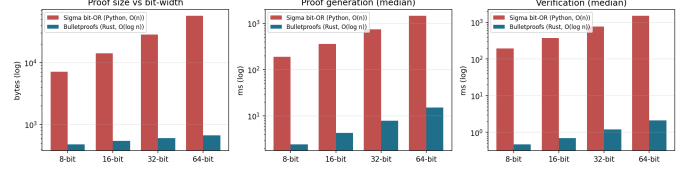


Fig. 4. Proof size, generation, and verification vs. bit-width for the two engines (log scale). The $O(n)$ vs. $O(\log n)$ separation is the practical argument for Bulletproofs at financial bit-widths.

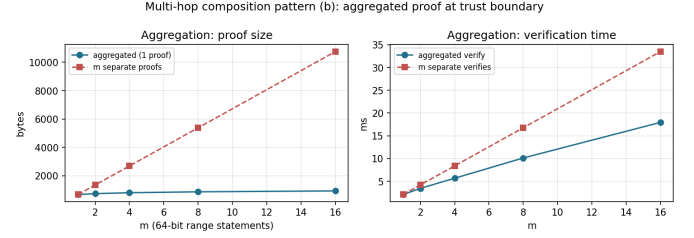


Fig. 5. Aggregated proofs at a trust boundary: size and verification vs. m separate proofs.

D. Adversarial Experiments

Eleven experiments instantiate the Section II adversary; all behave as required. T0: honest baseline verifies. T1: a prover holding \$12.5M ($> \text{cap}$) cannot obtain a proof from the honest API (the statement is false). T2: a single bit-flip in a response scalar, and separately a swap of two bit commitments, are both rejected. T3: replay of a valid proof under a different nonce fails (context binding). T4: replay under a different *predicate version* fails; policy updates invalidate old proofs by construction. T5: registry publication of an unsigned predicate, and of a predicate signed by a non-governance key, are both rejected. T6: an after-the-fact edit of one audit entry breaks the hash chain and is detected. T7: a malicious prover substituting a non-binary “bit” commitment ($C = 2G + rH$, an overflow attempt) with 2,000 randomized forgery attempts at the OR-proof achieved 0 acceptances; the same tamper and cross-context tests against E2 are likewise rejected. Beyond the in-process suite, we exercise the full `zk-attach/v0` attachment over a live JSON-RPC channel between a client execution agent and an execution-venue server, across *two* agent-interopability surfaces served by one endpoint. Five MCP scenarios (a compliant order, a call with no attachment, a tampered proof, a valid proof replayed against a different order, and an over-cap notional) confirm deny-by-default, single-use context binding, and audited outcomes end to end. The same governed action is then exercised over Agent2Agent: an Agent Card advertises it as a skill, `message/send` carries the attachment in a message data part, and an order admitted via A2A lands on the same ledger the MCP surface reports. Both surfaces share one context issuer, one verification chain, and one audit log, so the guarantee is a property of the gateway rather than of either protocol. Nineteen checks pass in total. T7 is an implementation sanity check, not a security proof: soundness rests on the cited constructions [14], [15], [17].

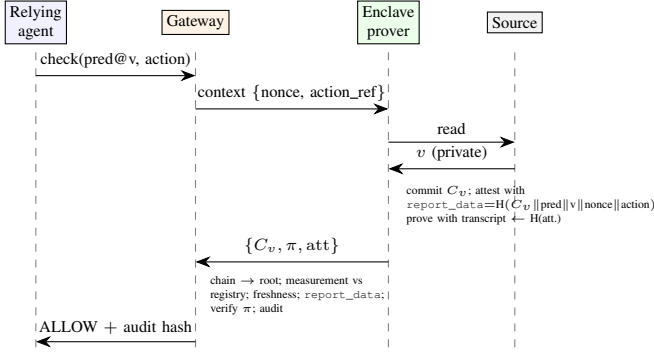


Fig. 6. Attestation-bound flow. One gateway-issued nonce serves both the proof context and the enclave attestation call; `report_data` binds the attestation to the commitment while the transcript binds the proof to the attestation.

VII. ATTESTATION-BOUND PREDICATE PROOFS

Section IV-C states the limitation squarely: a range proof binds a statement to the *committed* value, and topology A narrows the gap only by co-locating the prover with the source of truth. Neither says how a remote verifier confirms *which* binary produced the committed value. This section gives a construction that does, and reports its implementation status honestly.

A. Mutual Binding

The goal is that verifying a single artifact certifies two things jointly: that the predicate holds over a committed value, and that the value was read from the source of truth by a specific, measured binary running in an attested enclave. Prior designs that combine trusted execution with zero-knowledge proofs tend to carry an attestation *alongside* a proof, leaving each independently substitutable. We bind them in both directions, and both are necessary:

Direction A, attestation commits to the proof. The enclave populates the attestation document’s `report_data` field with $\text{SHA-384}(C_v \parallel \text{predicate_id} \parallel \text{version} \parallel \text{nonce} \parallel \text{action_ref})$. A valid attestation therefore cannot be replayed alongside a different commitment, predicate, or request.

Direction B, proof commits to the attestation. Before any commitment is generated, the Merlin transcript absorbs $\text{SHA-256}(\text{attestation_document})$, extending the same context-binding step of Section IV-B with a second, attestation-keyed message. A valid proof therefore cannot be presented under a substituted or unattested attestation.

Fiat-Shamir binds forward, from transcript to proof; `report_data` binds backward, from the enclave’s own output to the commitment it vouches for. The two directions close the same loop from opposite ends, which is why neither alone suffices.

B. Nonce Unification

Fig. 6 shows the resulting flow. The substantive protocol requirement is not the hashing but the nonce. The gateway-issued nonce that already binds proof context must

be the *same* value passed to the enclave’s attestation call, with identical expiry. Were the two allowed to diverge, an adversary could pair a fresh attestation with a stale but still-valid proof, or the reverse. Making one issued nonce serve both roles is what makes the two artifacts inseparable.

C. Governance-Signed Prover Identity

Expected measurements are not configuration. We add a `prover_measurement` predicate type carrying the expected measurement, Schnorr-signed by the same governance key that signs threshold predicates and versioned the same way. The registry then states not only that the cap is a given value, but that the only binary permitted to assert it is the one with a given measurement. Prover identity becomes policy, under the same signing authority as the business rule it accompanies, and a compromised agent can no more substitute a prover than it can loosen a cap.

D. Verification and Implementation Status

The gateway’s verification chain validates the attestation to its root of trust, checks the measurement against a registered `prover_measurement` predicate where one exists, checks freshness against the issued nonce, recomputes and compares `report_data`, verifies the proof under the attestation-bound transcript, and records the attestation digest in the audit entry alongside the existing fields.

Schema versioning. The attestation travels as an optional member of the existing `zk-attach/v0` envelope of Section IV-D; it does not define a new schema version. An envelope either carries the attestation payload or does not, and the verifier selects the attested or unattested transcript accordingly, which keeps a v0 verifier that ignores the field interoperable with an attesting prover’s unattested peers. Measurement *policy* is a separate axis: whether a measurement is required is determined by the registry, not by the envelope. We keep these two axes independent deliberately, for reasons the implementation experience below makes concrete.

We implement this construction in the Rust engine as `attest-prove`, `attest-verify`, and `attest-measurement`, covered by 21 unit tests, and wire the verification chain into the live governed path on both the MCP and A2A surfaces, opt-in per deployment. **The attestation authority is currently a mock:** an Ed25519-signed document over module identity, timestamp, measurement, nonce, and user data, with a publicly derivable root seed standing in for a hardware root of trust. The construction, the binding, the verification chain, and the registry integration are real and tested; the hardware is not yet. Running this against AWS Nitro Enclaves and GCP Confidential Space is the remaining step, and Section VIII states what we expect it to cost and what it does not solve.

We are precise about what the mock does and does not establish, because the distinction matters for how this result should be read. What it does not establish is any security property: its root seed is publicly derivable, so it supplies no hardware root of trust, and it says nothing about an adversary who controls the platform. What it does establish is that the

protocol is complete and functionally proven. The bidirectional transcript fusion produces artifacts that verify when and only when both bindings hold; the six-step verification chain executes in order against real signed documents and rejects at each step when its precondition fails; the governance-signed measurement predicate is parsed, matched, and enforced through the same registry path as a threshold predicate; and the whole sequence runs end to end on both protocol surfaces. Substituting a hardware attestation document for the mock changes the issuer of the document and the cost of validating it, not the state machine that consumes it. What is deferred is therefore the hardware timing evaluation and the platform trust root, not the design of the construction.

One implementation experience is worth recording because it is a property of the construction rather than of our code. An early version had the prover always attest while the gateway used the attested verification path only when a measurement predicate was registered. Because the two transcripts genuinely differ, every proof then failed to verify whenever no measurement policy existed. The defect was caught by rerunning the full end-to-end suite, not by unit tests, and the fix separates the two concerns: the transcript choice follows whether the envelope carries an attestation, while the measurement policy remains registry-driven and independent. Mutual binding makes attestation and proof inseparable by design, which means partial deployment is not a spectrum but a discontinuity.

VIII. LIMITATIONS AND OPEN PROBLEMS

1) *Only structured predicates reduce to circuits*: Range, set-membership, threshold, and Boolean compositions cover a meaningful slice of enterprise policy, but fuzzy or judgment-based criteria (“is this communication client-appropriate?”) do not reduce cleanly to arithmetic statements. Wherever an LLM must judge, the architecture binds and audits the judgment’s inputs and outputs, but it cannot make the judgment itself provable.

2) *No protocol slot exists yet*: `zk-attach/v0` is a proposal and not a standard. MCP, A2A, and function-calling schemas would each need a backward-compatible extension [20]–[22]; we regard standardization as the most direct next step, and independent analyses of MCP have already identified the absence of capability attestation as a specification gap [23].

3) *Metadata leakage*: ZKPs mask the witness, not the traffic: which predicate was invoked, by whom, how frequently, and when remains visible to the gateway and network observers, and can itself be of a sensitive nature (a burst of `pretrade_notional_cap` checks reveals trading activity). Padding, batching, and private information retrieval are orthogonal mitigations not evaluated here.

4) *Governance is a socio-technical open problem*: Who defines predicates, how versions roll out as policy changes (our version-binding makes stale proofs fail closed, but migration windows need policy), and what is supposed to happen on proof failure (hard deny versus human escalation) are not resolved; this proposal supplies the mechanism, not the operating model.

5) *Attestation is implemented but not yet hardware-validated*: Section VII gives the construction and tests it against a mock authority. Substituting real hardware introduces three things we have not measured: certificate-chain validation cost at the gateway, the size shift from a 608-byte proof to an envelope dominated by a multi-kilobyte attestation document, and the operational question of measurement churn, since rebuilding the prover changes its measurement and therefore invalidates previously registered values. Until that is done, source integrity still rests on the deployment discipline of topology A or B described in Section IV-C.

6) *Enclave input imposes a deployment requirement*: The construction certifies which binary produced a commitment; it does not by itself certify what that binary was fed. AWS Nitro Enclaves have no filesystem and no network, only a vsock channel to the parent instance, so a prover inside an enclave reaches the source of truth through a proxy on a host this threat model treats as untrusted. The requirement this imposes is standard rather than novel: the enclave must terminate an end-to-end encrypted channel to the system of record *inside* its own memory, tunnelling TLS over vsock so that the parent relays ciphertext it can neither read nor alter. Two conditions make this sound. The trust anchor used to authenticate the source must be part of the measured image, so that a parent cannot redirect the session to an endpoint of its choosing without changing the measurement the registry pins; and any client credential the source requires must be released to the enclave under attestation, so that possession of it is conditioned on the same measurement. Under those conditions the parent retains only the ability to delay or deny traffic, which is an availability concern rather than an integrity one, and the honest residual is that we have specified this requirement rather than measured it.

7) *Verifiable LLM inference is out of scope*: Proving that a model’s own reasoning or inference was computed correctly (zkML) remains far too expensive for agent loops: state-of-the-art proofs for a 13B-parameter LLM inference take approximately 15 minutes to generate [27], versus the 6.2 ms predicate proofs above. We state this explicitly as future work; the claim is that *data-plane* predicates are provable today even though the *model* is not.

IX. RELATED WORK

Table IV summarises how this work differs from the closest systems. The concept of proving policy compliance over private agent data is not ours: Aegis [31] proposed it first. What distinguishes this work is that the cryptography is executed rather than modelled, that the carriage mechanism is demonstrated on two protocols, and that the source-integrity assumption is stated and characterised rather than left implicit.

1) *Zero-knowledge architecture in deployed systems*: Minimal-disclosure cryptography is proven at scale outside agent systems: Zcash’s shielded transactions [39], and zero-knowledge or minimal-knowledge designs in messaging and credential systems. Our contribution is not new cryptography, nor the first application of zero-knowledge proofs to agent policy compliance; it is an end-to-end, measured, and governance-complete realisation of that idea at the inter-agent link.

TABLE IV

CLOSEST SYSTEMS FOR CRYPTOGRAPHICALLY ENFORCED AGENT POLICY. “SOURCE INTEGRITY” ASKS WHETHER THE CONSTRUCTION BINDS THE PROVED VALUE TO THE SYSTEM OF RECORD. [†]VALIDATED AGAINST A MOCK ATTESTATION AUTHORITY; HARDWARE EVALUATION DEFERRED (SECTION VII-D).

System	ZK proves	Setup	Evaluation	Carriage	Audit	Source integrity
Aegis [31]	policy compliance	transparent	simulated	none	none	not addressed
Dist. Sentinel [32]	graph policy	trusted (Groth16)	measured	none	none	not addressed
zk-MCP audit [36]	message metadata	trusted	measured	MCP, async	audit service	not addressed
NiyamAI [33]	judge executed	trusted	measured	interception	none	not addressed
Proof of Exec. [34]	(no ZK)	—	measured	agent runtime	signed records	trusted recorder
This work	value predicate	transparent	measured, logged	MCP + A2A	hash-chained	attestation-bound[†]

2) *Agentic-security mainstream*: OWASP’s agentic Top 10 [5] and threat taxonomy [6], MITRE ATLAS’s agent-focused techniques [7], and Zero Trust agent architectures [8], [9] target identity, privilege, visibility, and detection. We position the Proof Gateway as complementary: those frameworks decide *whether* an interaction may occur and observe it; ours minimizes *what data* the permitted interaction carries. Notably, insecure inter-agent communication and human-agent trust exploitation appear as named risks in the 2026 OWASP list [5]; predicate proofs address the data-exposure component of both.

3) *ZKPs applied to agent systems*: Closest to this work is the Aegis Protocol [31], which also proves policy compliance over private agent data with a transparent-setup proof system (Halo2); its evaluation, however, is a discrete-event simulation in which proof latencies are sampled from a calibrated log-normal distribution rather than measured, and it does not address predicate governance, replay or action binding, protocol carriage, or audit chaining. Wu and Gong [32] include zero-knowledge predicate proofs between source and requesting sidecars as an optional extension to a taint-propagation architecture, using Groth16 and therefore requiring a trusted setup. Katkar et al. [33] gate tool calls on a zk-SNARK that an isolated judge model executed correctly, hiding model weights rather than payload data. A further neighbour is the zero-knowledge *audit* system of Jing and Qi [36], in which agents exchange full message content over unmodified MCP, generate zk-SNARK proofs asynchronously after the session, and submit them to a third-party audit service that verifies statements about message format, type counts, and token usage; privacy holds against the auditor, not against the communicating counterparty, and the construction relies on a trusted setup. Our contribution differs in layer, purpose, and timing: the sensitive value is never transmitted to the relying agent at all; a value-predicate proof travels *in lieu of* the data, synchronously, at the moment of trust establishment, under a transparent (no trusted setup) proof system. Huang et al. [37] apply DIDs, verifiable credentials, and ZKPs to agent *identity* and attribute-based access control, i.e., the identity layer, which is orthogonal to the data-payload layer here. Proof of Execution [34] verifies that governed agent actions occurred as recorded, but does so with signed execution records produced by a recorder inside the trusted computing base rather than with zero-knowledge proofs, and therefore inherits a trusted-component assumption where we inherit a committed-value assumption. Designs that pair trusted execution with zero-

knowledge proofs for agent systems generally carry the attestation as a layer beside the proof, leaving each independently substitutable; the construction of Section VII instead binds them to one another in both directions, and places the expected prover measurement under governance signature rather than deployment configuration. Stantchev’s integrative review of verifiable law for machine societies [35] surveys object-capability security, verifiable computation, policy-as-code, and agentic payment protocols, and concludes that the gap in this area is architectural rather than technological: the components exist but the discipline composing them does not. This paper is best read as discharging part of that agenda with running code, and as reporting which assumption survives the exercise. General agent threat models [38] inform Section II.

4) *Verifiable machine learning*: zkLLM [27], optimizing systems such as ZKML [28], surveys of the field [29], and production toolchains (EZKL, RISC Zero) [30] pursue proofs of *inference*. We deliberately exclude this: our proofs are over the data agents exchange, not over the models that exchange it.

REPRODUCIBILITY STATEMENT

All results in this paper come from a single logged run of an open-source reference implementation, available at <https://github.com/45h0kg/zk-proof-gateway>, reproducible with one command. The environment, engine versions, and per-stage outputs are recorded in the run log that ships with the code. Proof sizes are deterministic and host-independent; absolute timings depend on hardware, and we report figures from one virtualized single-vCPU host as feasibility evidence rather than production benchmarks. The adversarial suite and the end-to-end agentic protocol checks are deterministic pass/fail tests included in the release. The Kubernetes deployment described in Section VI was installed and exercised on a local cluster; network-policy isolation of the prover was confirmed by observing connections fail with the policy applied and succeed with it withdrawn, rather than assumed from the manifest.

X. CONCLUSION

Mainstream agentic security in 2026 is detection-based: it authenticates the agent, observes its behavior, and attempts to catch misuse. This paper argued for, designed, and prototyped a structurally preventive complement: when the sensitive value is not required for trust establishment, the interaction that establishes trust cannot expose it. The Proof Gateway replaces

raw data and natural-language self-report with governance-defined predicate proofs; a compromised agent can still lie in prose, but it cannot produce a verifying proof of a false statement, cannot replay an old proof against a new request or policy version, and cannot register itself a weaker policy. Our reference prototype shows the costs are already compatible with real-time agent loops for financial bit-widths (608-byte proofs, single-digit-millisecond generation, millisecond verification), and our eleven adversarial experiments behave as the threat model requires. Next steps are: running the attestation construction of Section VII against real Nitro Enclave and Confidential Space hardware and measuring what it costs, together with an implementation of the TLS-over-vsock input path and attestation-gated credential release that Section VIII specifies; a standardization proposal for the `zk-attach/v0` slot in MCP and A2A, benchmarking on a broader catalog of enterprise predicates including set membership and Boolean compositions, and, as longer-horizon future work, revisiting verifiable inference as zkML costs fall.

APPENDIX A

WORKED SCHNORR IDENTIFICATION EXAMPLE

Group: the order-11 subgroup of $\mathbb{Z}_{23}^*$ generated by $g = 2$ ($2^{11} \equiv 1 \pmod{23}$). Secret $x = 7$; public $y = g^x \pmod{p} = 2^7 \pmod{23} = 13$.

- 1) Prover picks nonce $w = 3$, sends $a = g^w \pmod{p} = 8$.
- 2) Verifier sends random challenge $e = 5$.
- 3) Prover sends $z = w + ex \pmod{q} = 3 + 35 \pmod{11} = 5$.
- 4) Verifier checks $g^z \stackrel{?}{=} a \cdot y^e \pmod{p}$: $2^5 = 32 \equiv 9$, and $8 \cdot 13^5 \equiv 8 \cdot 4 \equiv 9 \pmod{23}$. Accept.

Zero-knowledge: the tuple (a, e, z) is simulatable without x by sampling z, e first and setting $a = g^z y^{-e}$; Fiat–Shamir sets $e = \text{Hash}(\text{transcript})$ to remove interaction [13]. Our bit proofs are OR-compositions of exactly this template [15].

REFERENCES

- [1] Gravitee, “The state of AI agent security 2026 report,” 2026. [Online]. Available: <https://www.gravitee.io/state-of-ai-agent-security>
- [2] Arkose Labs, “2026 agentic AI security report,” Mar. 2026.
- [3] S. Willison, “The lethal trifecta for AI agents: Private data, untrusted content, and external communication,” Jun. 16, 2025. [Online]. Available: <https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/>
- [4] L. Beurer-Kellner *et al.*, “Design patterns for securing LLM agents against prompt injections,” arXiv:2506.08837, 2025.
- [5] OWASP GenAI Security Project, “OWASP top 10 for agentic applications 2026,” Dec. 2025. [Online]. Available: <https://genai.owasp.org>
- [6] OWASP Agentic Security Initiative, “Agentic AI: Threats and mitigations,” Feb. 2025.
- [7] MITRE, “MITRE ATLAS,” v5.1.0, Nov. 2025. [Online]. Available: <https://atlas.mitre.org>
- [8] Anthropic, “Zero trust for AI agents,” 2025. [Online]. Available: <https://claude.com/blog/zero-trust-for-ai-agents>
- [9] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, “Zero trust architecture,” NIST SP 800-207, 2020.
- [10] S. Goldwasser, S. Micali, and C. Rackoff, “The knowledge complexity of interactive proof-systems,” in *Proc. 17th ACM STOC*, 1985, pp. 291–304.
- [11] O. Goldreich, S. Micali, and A. Wigderson, “Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems,” *J. ACM*, vol. 38, no. 3, pp. 691–729, 1991.
- [12] C. P. Schnorr, “Efficient signature generation by smart cards,” *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.
- [13] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Advances in Cryptology—CRYPTO ’86*, LNCS 263, 1987, pp. 186–194.
- [14] D. Bernhard, O. Pereira, and B. Warinschi, “How not to prove yourself: Pitfalls of the Fiat–Shamir heuristic and applications to Helios,” in *ASIACRYPT*, 2012, pp. 626–643.
- [15] R. Cramer, I. Damgård, and B. Schoenmakers, “Proofs of partial knowledge and simplified design of witness hiding protocols,” in *Advances in Cryptology—CRYPTO ’94*, LNCS 839, 1994, pp. 174–187.
- [16] T. P. Pedersen, “Non-interactive and information-theoretic secure verifiable secret sharing,” in *Advances in Cryptology—CRYPTO ’91*, LNCS 576, 1992, pp. 129–140.
- [17] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short proofs for confidential transactions and more,” in *Proc. IEEE Symp. Security and Privacy (S&P)*, 2018, pp. 315–334.
- [18] dalek-cryptography, “Bulletproofs: A pure-Rust implementation over Ristretto,” [Online]. Available: <https://github.com/dalek-cryptography/bulletproofs> [Accessed: Aug. 3, 2026].
- [19] H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo, “Bulletproofs+: Shorter proofs for a privacy-enhanced distributed ledger,” *IEEE Access*, vol. 10, pp. 42067–42082, 2022.
- [20] Anthropic, “Model context protocol specification,” [Online]. Available: <https://modelcontextprotocol.io> [Accessed: Aug. 3, 2026].
- [21] A2A Project (Linux Foundation), “Agent2Agent (A2A) protocol,” 2025. [Online]. Available: <https://github.com/a2aproject/A2A> [Accessed: Aug. 3, 2026].
- [22] OpenAI, “Function calling,” OpenAI API documentation. [Online]. Available: <https://platform.openai.com/docs/guides/function-calling> [Accessed: Aug. 3, 2026].
- [23] N. Maloyan and D. Namiot, “Breaking the protocol: Security analysis of the model context protocol specification and prompt injection vulnerabilities in tool-integrated LLM agents,” arXiv:2601.17549, Jan. 2026.
- [24] European Parliament and Council, “Regulation (EU) 2016/679 (General Data Protection Regulation),” *Official Journal of the European Union*, L119, 2016. Articles 5(1)(c), 22, 25, 32.
- [25] European Parliament and Council, “Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (AI Act),” *Official Journal of the European Union*, 2024. Articles 12, 14, 50.
- [26] European Parliament and Council, “Regulation (EU) 2024/1183 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework,” *Official Journal of the European Union*, 2024. See recitals on selective disclosure and zero-knowledge proof.
- [27] H. Sun, J. Li, and H. Zhang, “zkLLM: Zero knowledge proofs for large language models,” in *Proc. ACM CCS*, 2024, pp. 4405–4419.
- [28] B. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, “ZKML: An optimizing system for ML inference in zero-knowledge proofs,” in *Proc. EuroSys*, 2024, pp. 560–574.
- [29] Z. Peng *et al.*, “A survey of zero-knowledge proof based verifiable machine learning,” arXiv:2502.18535, 2025.
- [30] Zkonduit, “EZKL,” [Online]. Available: <https://github.com/zkonduit/ezkl>; RISC Zero, [Online]. Available: <https://github.com/risc0/risc0> [Accessed: Aug. 3, 2026].
- [31] S. T. R. Adapala and Y. R. Alugubelly, “The Aegis protocol: A foundational security framework for autonomous AI agents,” arXiv:2508.19267, Aug. 2025.
- [32] J. Wu and M. Gong, “Beyond single-agent alignment: Preventing context-fragmented violations in multi-agent systems,” arXiv:2604.22879, Apr. 2026.
- [33] A. Katkar *et al.*, “NiyamAI: An intent-bound AI agent with cryptographically verifiable guardrails using zero-knowledge proofs,” arXiv:2608.07167, Aug. 2026.
- [34] J. Rhodes and G. Kang, “Proof of execution: Runtime verification for governed AI agent actions,” arXiv:2607.05397, 2026.
- [35] V. Stantchev, “Computational jurisprudence: Verifiable law for machine societies,” *Future Internet*, vol. 18, no. 8, art. 437, 2026, doi:10.3390/fi18080437.
- [36] G. Jing and H. Qi, “Zero-knowledge audit for internet of agents: Privacy-preserving communication verification with model context protocol,” arXiv:2512.14737, Dec. 2025.
- [37] K. Huang, V. S. Narajala, J. Yeoh *et al.*, “A novel zero-trust identity framework for agentic AI: Decentralized authentication and fine-grained access control,” arXiv:2505.19301, 2025.
- [38] V. S. Narajala and O. Narayan, “Securing agentic AI: A comprehensive threat model and mitigation framework for generative AI agents,” arXiv:2504.19956, 2025.

- [39] E. Ben-Sasson *et al.*, “Zerocash: Decentralized anonymous payments from Bitcoin,” in *Proc. IEEE Symp. Security and Privacy (S&P)*, 2014, pp. 459–474.