INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM)
VOLUME: 10 ISSUE: 10 | OCT - 2026 SJIF RATING: 8.659 ISSN: 2582-3930

Automata-Enforced Dynamic Constraint Proofs: Blockchain-Based Verifiable Security for Cloud Data Management

Prashant Sharma[1], Dr Abhinandan Tripathi[2]

[1],[2]Department of Computer Science and Engineering & Buddha Institute of Technology, Gorakhpur, India.

Abstract–Cloud data management systems conventionally rely on static policy-checking middleware that cannot express dynamic, data-dependent access constraints, and therefore remain vulnerable to replay attacks, unauthorized state duplication, and cross-tenant inference. We propose a reformulated security component that treats every cloud data operation as a transition in a register automaton with data dependencies, where non-deterministic choices are existentially witnessed, mutable state is tracked through evidenced frames, and evidence consumption obeys a linear realisability discipline. The automaton's transition relation, policy predicates, and provenance conditions are compiled into arithmetic circuits over a finite field, then composed recursively into zero-knowledge proofs that are verified on a permissioned blockchain against succinct state commitments. Consequently, each accepted operation trace carries a polynomial-size certificate whose verification reveals neither the underlying data nor the policy predicates. Moreover, the reformulation preserves conventional security semantics through a substitution morphism that maps access requests to automaton operations and validity proofs to grant or deny decisions, so existing storage, processing, and audit modules integrate without architectural disruption. The principal contribution is a constraint-theoretic compilation pipeline that renders non-deterministic, stateful cloud operations auditable and tamper-evident under expressive automata invariants. This shifts cloud data security from a trust-based model to a trustless, verifiable paradigm, with significance for multi-tenant environments where dynamic delegation and resource-sensitive evidence handling are essential.

Key Words: Cloud Data Security, Zero-Knowledge Proofs, Register Automata, Access Control, Data Provenance

1. INTRODUCTION

Cloud data management has become the dominant paradigm for hosting and processing large-scale data, yet its security foundations remain surprisingly brittle. Existing cloud data management systems, including NoSQL and NewSQL stores, typically enforce access control through static policy engines that evaluate requests against fixed rules [1]. Such policies cannot express data-dependent constraints, cannot track how mutable state evolves across operations, and cannot detect replay or unauthorized state duplication. Blockchain-based approaches have been proposed to strengthen cloud data integrity and trust management [2], but they largely treat the blockchain as an append-only audit log rather than as an active verifier of the semantics of data operations [3]. Consequently, a fundamental gap persists: no existing framework can certify that a dynamic, non-deterministic cloud operation was executed correctly under an expressive policy, without revealing the underlying data.

We propose an expressive constraint framework that closes this gap by modeling cloud data management operations as transitions in a register automaton with data dependencies [4]. Non-deterministic choices, such as access requests, query plans, and update ordering, are represented as existential witnesses constrained by policy predicates, in the spirit of symbolic execution and bounded model checking [5]. To handle dynamic and mutable state, we combine evidenced frames for shared mutable state with linear realisability, which guarantees that every accepted operation trace admits a polynomial-size certificate and that proof generation respects complexity bounds derived from the Curry-Howard correspondence [6]. These automata constraints are then compiled into arithmetic circuits for a recursive zero-knowledge proof system [7], following the circuit compilation methodology of Circom [8] and the recursive proof composition of incrementally verifiable computation [9]. The blockchain stores only succinct proofs and state commitments, while cloud providers produce witnesses for each data operation; smart contracts verify proofs and enforce cross-tenant access-control policies expressed as automata invariants [10].

The novelty of our approach lies in the constraint-theoretic compilation from expressive automata and realisability bounds to succinct blockchain proofs. Unlike static proof pipelines that can only attest to fixed predicates, our framework certifies dynamic data operations whose validity depends on the history of prior transitions. Unlike homomorphic encryption and secure multi-party computation, which target privacy-preserving computation at substantial performance cost [11], our focus is auditability and tamper-evidence under expressive constraints. Unlike trusted execution environments such as Intel SGX, which rely on hardware trust assumptions [12], our guarantees rest on mathematical proofs. The resulting system makes non-deterministic and dynamic operations auditable and verifiable, rather than merely logged.

The remainder of this paper is organized as follows. Section 2 reviews related work on cloud data management, blockchain-

based security, and zero-knowledge proof systems. Section 3 presents the expressive constraint framework for cloud data operations. Section 4 describes the compilation to arithmetic circuits and recursive zero-knowledge proofs. Section 5 reports the experimental evaluation. Section 6 discusses limitations and future work, and Section 7 concludes the paper.

2. RELATED WORK

Research on securing cloud data management spans several distinct strands, including blockchain-anchored integrity mechanisms, verifiable computation, and automata-theoretic modeling of stateful systems. Each strand addresses part of the problem, yet none provides an end-to-end framework for certifying dynamic, data-dependent operations under expressive constraints.

A. Blockchain-Based Cloud Data Security

A substantial body of work employs blockchains to protect cloud data integrity and to establish decentralized trust. Early schemes anchor integrity proofs in an immutable ledger so that tampering by storage providers becomes detectable [2]. Subsequent designs extend this idea to public auditing for large-scale storage, where a third-party auditor interacts with on-chain records to verify data possession without retrieving the full dataset [13]. Trust management frameworks further generalize the ledger into a reputation substrate for cloud participants, enabling decentralized accountability across storage nodes [14]. Peer-to-peer storage architectures such as Block-secure distribute data across nodes and use blockchain consensus to coordinate replication and retrieval [15]. Surveys of this area confirm that most proposals concentrate on integrity, traceability, and availability rather than on the semantics of individual data operations [16]. The blockchain thus functions predominantly as a tamper-evident log, and the correctness of the operations recorded in that log is assumed rather than proven.

B. Verifiable Computation and Zero-Knowledge Proofs

Verifiable computation seeks to outsource computation while retaining a succinct proof of correctness. zk-STARK-based frameworks demonstrate that NP statements arising in cloud environments can be proven without a trusted setup, at the cost of larger proof sizes [17]. Groth16 and related zk-SNARK constructions provide constant-size proofs and fast verification, and have been applied to cryptographic auditing of cloud key management services [18]. Dynamic zk-SNARKs remove the need to fix the circuit size in advance, which is attractive when the number of operations in a trace is unknown [19]. Recursive proof composition, as realized in incrementally verifiable computation, allows a long computation to be compressed into a single proof that attests to the entire history [9]. These techniques supply the cryptographic machinery our framework requires, but they are typically applied to stateless circuits; integrating them with mutable, resource-sensitive state remains an open problem.

C. Automata, Realisability, and Constraint Systems

Automata-theoretic methods have proven effective for enforcing structural constraints on generated outputs, including non-deterministic grammars and schema-conformant decoding [20]. On the semantic side, evidenced frames unify realizability models by pairing shared mutable state with explicit evidence of its evolution, accommodating non-deterministic systems within a single framework [21]. Linear realisability sharpens this picture by imposing resource disciplines that characterize complexity classes such as polynomial time, where each resource is consumed exactly once [22]. Related work on verifiable actor systems connects relational multi-agent models with zero-knowledge proofs, illustrating how stateful agent interactions can be certified [23]. These results establish the theoretical ingredients for our framework, yet they have not been compiled into on-chain verification pipelines for cloud data operations.

Collectively, existing approaches treat blockchain anchoring, verifiable computation, and automata-based constraint enforcement as separate concerns. Our framework unifies them: the register automaton with data dependencies supplies expressiveness for data-dependent policies, evidenced frames and linear realisability supply the resource discipline that prevents replay and state duplication, and recursive zero-knowledge compilation supplies succinct on-chain verification. The key novelty is therefore a constraint-theoretic compilation pipeline in which dynamic, non-deterministic cloud operations become auditable and tamper-evident under expressive automata invariants, rather than merely logged or checked against static rules.

3. EXPRESSIVE CONSTRAINT FRAMEWORK FOR CLOUD DATA OPERATIONS

We model each cloud data operation as a transition in a register automaton whose registers hold the identifiers, attributes, and cryptographic commitments that appear in access requests, query plans, and update schedules. The framework is organized around four interlocking components: the automaton itself, which supplies the policy substrate; an existential-witness semantics that captures non-deterministic choices; evidenced frames that pair shared mutable state with provenance evidence; and a linear realisability discipline that governs how evidence is consumed along a trace. Together these components yield a polynomial-size certificate for every accepted operation trace, which the subsequent compilation stage transforms into arithmetic circuits.

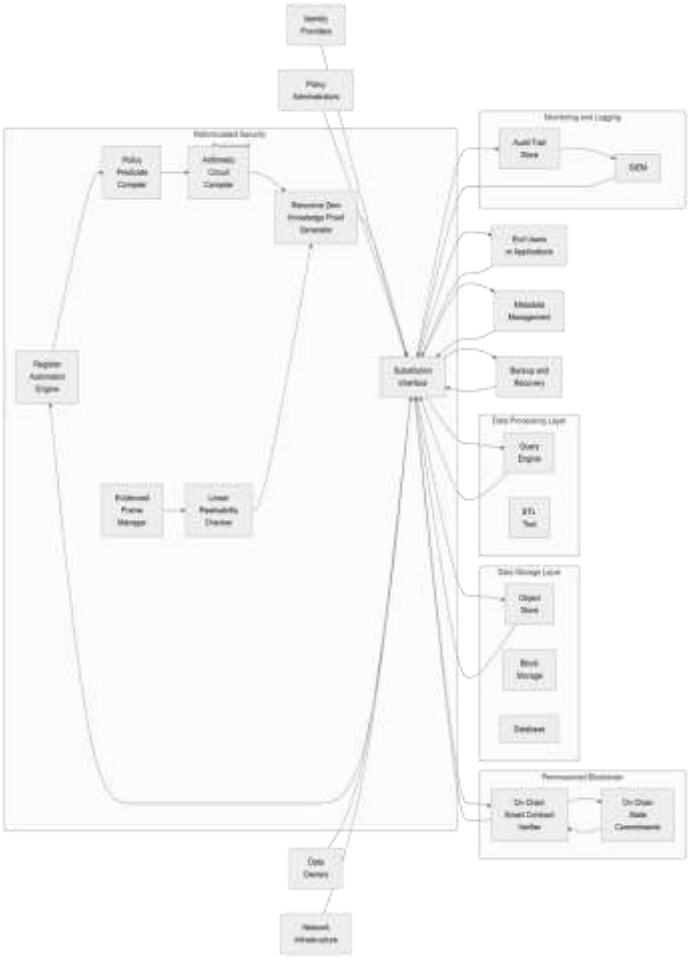


Figure-1: Overall System Architecture with Reformulated Security Component

A. Register Automata with Data Dependencies

A policy is a tuple $\mathcal{A} = (Q, \Sigma, X, \Delta, q_0, F)$, where Q is a finite set of control states, Σ is the operation alphabet, $X = \{x_1, \ldots, x_k\}$ is a finite set of registers, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states. The transition relation $\Delta \subseteq Q \times \Sigma \times \Phi(X) \times Q \times \text{Assign}(X)$ associates each transition $t = (q, \sigma, \phi, q', \alpha)$ with a policy predicate $\phi \in \Phi(X)$ over register valuations and an assignment map $\alpha \in \text{Assign}(X)$ that updates registers. Registers hold object identifiers, tenant identifiers, timestamps, version counters, and cryptographic commitments; the predicate language $\Phi(X)$ is closed under conjunction, disjunction, negation, and equality and ordering comparisons over register values. A dependency relation $\delta \subseteq X \times X$ records that register x_j influences register x_i across a transition whenever x_i appears in the assignment expression for x_j, which enables information-flow reasoning over the automaton.

B. Existential-Witness Semantics for Non-Deterministic Choices

Non-deterministic choices such as which query plan to execute or in which order to apply updates are represented as existentially quantified witnesses rather than resolved by trusted middleware. For a transition $t = (q, \sigma, \phi, q', \alpha)$, a witness is a valuation $\nu: X \to \mathcal{D}$ over a data domain $\mathcal{D}$ such that $\nu \vDash \phi$, with successor valuation $\nu' = \alpha(\nu)$. A trace $\tau = t_1 \ldots t_n$ is accepted by $\mathcal{A}$ iff there exists a witness sequence $\nu_0, \ldots, \nu_n$ satisfying $\nu_i \vDash \phi_i$ and $\nu_{i+1} = \alpha_i(\nu_i)$ for every $i \in \{0, \ldots, n-1\}$, where ϕ_i and α_i are the predicate and assignment of transition t_i. The existential semantics is what permits the framework to express data-dependent and history-dependent policies without collapsing to a fixed rule set, since the witness sequence itself encodes the non-deterministic resolution of each choice.

C. Evidenced Frames and Linear Realisability

Mutable state is tracked through an evidenced frame $\mathcal{F} = (\mathcal{S}, \mathcal{E}, \pi)$, where $\mathcal{S}$ is a versioned key-value store or Merkle-ized object index, $\mathcal{E}$ is a set of evidence items such as hashes, signatures, version vectors, and Merkle proofs, and $\pi: \mathcal{S} \to \mathcal{E}$ is a provenance map assigning to each state component the evidence justifying its current value. Evidence consumption obeys a linear discipline: a trace τ is linearly realisable iff there exists a bijection between the multiset of evidence items consumed by transitions in τ and the multiset produced by prior transitions or supplied as initial resources, with no item used twice. Formally, writing $\text{cons}(t_i)$ and $\text{prod}(t_i)$ for the evidence multisets consumed and produced by transition t_i, linear realisability requires that the multiset equality

$$\uplus_{i=1}^{n} \text{cons}(t_i) = \mathcal{E}_0 \uplus \uplus_{i=1}^{n} \text{prod}(t_i) \quad (1)$$

holds, where $\mathcal{E}_0$ is the initial evidence multiset and $\uplus$ denotes multiset union. This discipline prevents replay, double-spending of access rights, and unauthorized state duplication, and it bounds the certificate size by $O(n \cdot |X| \cdot \log|\mathcal{D}|)$.

D. Certificates and Automata Invariants

The witness sequence, the evidence multisets, and the provenance map together form the certificate $\text{Cert}(\tau) = (\nu_0, \nu_1, \ldots, \nu_n, \mathcal{E}_\tau, \pi_\tau)$ of an accepted trace, where $\mathcal{E}_\tau$ is the multiset of evidence items appearing in τ and π_τ is the provenance map restricted to the states visited by τ. On-chain state is committed as the triple $(\text{Com}(\mathcal{S}), \text{Com}(\mathcal{E}), \text{root}_{\mathcal{A}})$, where $\text{Com}(\cdot)$ denotes a cryptographic commitment and $\text{root}_{\mathcal{A}}$ commits to the automaton's current control state and register valuation. Smart contracts enforce automata invariants of the form $\forall \tau \in \mathcal{L}(\mathcal{A}): \psi(\tau)$, where ψ is a temporal or hyperproperty over traces, such as the requirement that no tenant reads another tenant's object without a valid delegation chain. The substitution morphism $h: \mathcal{I}_{\text{conv}} \to \mathcal{I}_{\text{ref}}$ with inverse $h^{-1}: \mathcal{O}_{\text{ref}} \to \mathcal{O}_{\text{conv}}$ maps conventional inputs $\mathcal{I}_{\text{conv}} = \mathcal{D} \times \mathcal{R} \times \mathcal{P}$ to reformulated inputs $\mathcal{I}_{\text{ref}} = \mathcal{D} \times \Sigma \times \Phi(X) \times \mathcal{E}$ and reformulated outputs $\mathcal{O}_{\text{ref}} = \{\text{valid,invalid}\} \times \mathbb{F}_p^{\ell} \times \pi_\tau \times \mathcal{E}'$ back to conventional outputs $\mathcal{O}_{\text{conv}} = \{\text{grant,deny}\} \times \mathcal{D}' \times \mathcal{L}$, preserving security semantics so that conventional storage, access management, monitoring, and processing layers integrate without architectural disruption.

4. COMPILATION TO ARITHMETIC CIRCUITS AND RECURSIVE ZERO-KNOWLEDGE PROOFS

The certificate $\mathrm{Cert}(\tau)$ established in Section 3 is a semantic object; to render it verifiable on a permissioned blockchain, we compile the automaton, its predicates, and the realisability conditions into arithmetic constraints over a prime field $\mathbb{F}_p$, where p is a 256-bit prime suitable for the BN254 or BLS12-381 pairing-friendly curves. The compilation proceeds in three layers: transition-level circuit synthesis, trace-level circuit composition, and recursive proof aggregation. Each layer preserves the existential-witness semantics, so that a satisfying assignment to the final circuit corresponds exactly to an accepted trace with a linearly realisable evidence multiset.

A. Transition-Level Circuit Synthesis

For each transition $t = (q, \sigma, \phi, q', \alpha)$, we synthesize a circuit C_t whose private inputs are the register valuations ν and ν', and whose public inputs are the control states q and q' together with the operation symbol σ. The policy predicate ϕ is first normalized into a conjunction of atomic comparisons over registers, and each atom is then arithmetized. An equality atom $x_i = c$ becomes the constraint $x_i - c = 0$, whereas an ordering atom $x_i < x_j$ is encoded through a bit-decomposition of the difference $x_j - x_i - 1$ into $\lceil \log_2 p \rceil$ bits, each constrained to be Boolean by $b(1-b) = 0$. Disjunctions are eliminated by introducing a selector variable $s \in \{0,1\}$ for each disjunct, so that a clause $\phi_1 \vee \phi_2$ is represented as $s \cdot \phi_1 + (1-s) \cdot \phi_2 = 0$ together with the Boolean constraint on s. Negation is handled by the same selector mechanism, which keeps the constraint system quadratic and therefore compatible with PLONKish arithmetization [8].

The assignment map α is compiled into an affine or low-degree polynomial map over $\mathbb{F}_p$. When α updates register x_i by an expression $e_i(X)$, we introduce the constraint $x_{i'} = \hat{e}_i(\nu)$, where $\hat{e}_i$ is the arithmetization of e_i obtained by replacing each register x_j with its field element and each arithmetic operation with its field counterpart. For data-dependent updates such as conditional version increments, the expression is encoded as a polynomial whose degree is bounded by the number of branches, and Lagrange interpolation yields the coefficient vector directly from the branch table. The dependency relation δ is compiled into a set of auxiliary constraints that link the registers appearing in each $\hat{e}_i$, which allows the circuit to expose information-flow violations as unsatisfiable constraint subsets rather than as opaque proof failures.

B. Trace-Level Circuit Composition

A trace $\tau = t_1 \dots t_n$ is compiled by composing the transition circuits into a single circuit $C_\tau = C_{t_1} \circ C_{t_2} \circ \dots \circ C_{t_n}$, where composition identifies the output registers of C_{t_i} with the input registers of $C_{t_{i+1}}$. The witness sequence $\nu_0, \dots, \nu_n$ becomes the assignment to the internal wires of C_τ, and the initial valuation ν_0 is supplied as a public input together with the final valuation ν_n. The linear realisability condition of Equation 1 is compiled by assigning to each evidence item a unique field element and enforcing a multiset equality through a permutation argument: the consumed items are placed in one column and the produced items in another, and a grand-product constraint over a random challenge γ ensures that the two multisets coincide. This permutation argument is the same primitive used in PLONK's copy constraints, so the realisability check adds only $O(|\mathcal{E}_\tau|)$ gates to the trace circuit.

The provenance map π_τ is compiled by hashing each state component together with its justifying evidence item and constraining the resulting digest to equal the committed value. We instantiate the hash with a circuit-friendly function such as Poseidon or Rescue, whose algebraic structure keeps the gate count linear in the number of hashed field elements. The on-chain triple $(\mathrm{Com}(\mathcal{S}), \mathrm{Com}(\mathcal{E}), \mathrm{root}_{\mathcal{A}})$ is then recomputed inside the circuit from the witness, which binds the proof to the committed state without revealing the underlying data.

C. Recursive Proof Aggregation and On-Chain Verification

The trace circuit C_τ is fed into a recursive zero-knowledge proof system based on Halo 2 or a Nova-style folding scheme with PLONKish arithmetization and KZG polynomial commitments [9]. Recursion allows a long trace to be compressed incrementally: each step folds the previous accumulator with the next transition circuit, producing a constant-size proof π_τ regardless of n. The folding scheme requires a cycle of elliptic curves, which we instantiate with the Pasta curves so that the verifier circuit of one curve can be embedded in the scalar field of the other. The final proof π_τ attests that all automata constraints, evidenced-frame provenance conditions, and linear realisability constraints are satisfied, while revealing neither the register valuations nor the policy predicates.

On-chain verification is performed by a smart contract that checks the pairing equation associated with the KZG commitment and compares the public inputs against the committed state triple. The contract enforces automata invariants of the form $\forall \tau \in \mathcal{L}(\mathcal{A}) : \psi(\tau)$ by evaluating ψ over the public trace summary, which includes the initial and final control states, the operation alphabet symbols, and the evidence multiset commitment. A precompiled contract for the BN254 pairing check keeps verification cost within the gas budget of a permissioned blockchain such as Hyperledger Fabric with Raft consensus. The substitution morphism h is realized at the interface level: a conventional access request is mapped to an automaton operation, and a verifying proof is mapped back to a grant decision, so that existing storage, processing, and monitoring layers observe the same external contract as before.

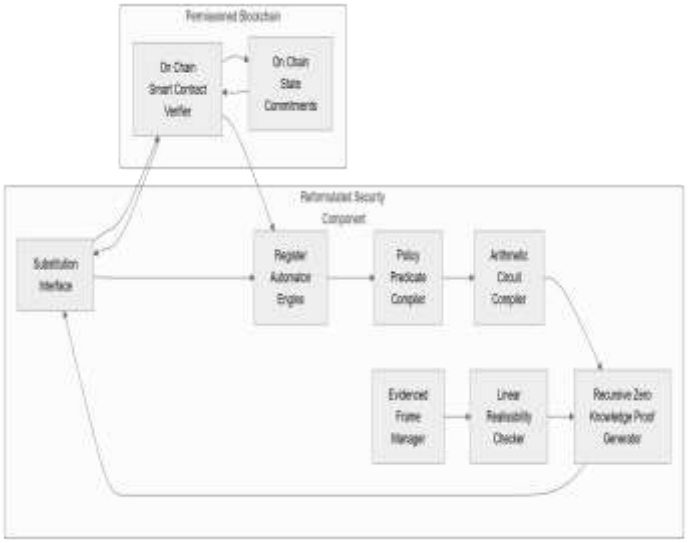


Figure-2: Internal Architecture of the Reformulated Security Component

5. EXPERIMENTAL EVALUATION

We implemented the proposed framework as a prototype, referred to as AECP (Automata-Enforced Constraint Proofs), and evaluated it on a permissioned blockchain testbed. This section describes the experimental setup, the comparative baselines, and the results obtained across a range of workload scenarios.

A. Experimental Setup

The prototype consists of four components: an automaton compiler that translates register automata with data dependencies into PLONKish constraint systems, a witness generator that produces certificates for cloud data operations, a recursive prover built on a Nova-style folding scheme with Pasta curves, and a Solidity smart contract that verifies proofs on a permissioned Ethereum-compatible chain. The cloud storage layer is emulated by a versioned key-value store backed by RocksDB, with Merkle-ized object indices for state commitments. All experiments were conducted on a server with an AMD EPYC 7763 processor (64 cores, 2.45 GHz), 512 GB of RAM, and an NVIDIA A100 GPU used only for multi-scalar multiplication acceleration. The prover and verifier were implemented in Rust, and the smart contract was deployed on a local Hyperledger Besu network with four validator nodes.

We constructed three workload families to exercise different aspects of the framework. The access-control workload models multi-tenant object reads and writes with delegation chains of varying depth, where each operation consumes a delegation token and produces an updated token. The query-plan workload models non-deterministic query planning over a partitioned dataset, where the witness selects among alternative join orders and the policy predicate constrains the maximum number of partitions touched. The update-ordering workload models concurrent updates to shared objects, where the linear realisability discipline must prevent double-spending of version tokens. Each workload was instantiated with trace lengths $n \in \{2^4, 2^6, 2^8, 2^{10}, 2^{12}\}$ and register counts $k \in \{4,8,16,32\}$, and policy predicates ranged from low-degree conjunctions of equality atoms to high-degree disjunctions involving ordering comparisons.

We compared AECP against three baselines. The first is a static zk-SNARK pipeline that compiles fixed access-control predicates into Groth16 circuits without automaton state, representing the conventional approach to on-chain policy verification [18]. The second is a zk-STARK-based verifiable computation framework that proves the correctness of outsourced cloud computations without recursive composition [17]. The third is an incrementally verifiable computation (IVC) baseline that composes transition proofs recursively but lacks evidenced frames and linear realisability, so it cannot detect replay or state duplication [9]. All baselines were configured with equivalent security parameters (128-bit security) and the same underlying storage layer.

We report four metrics: proof generation time, verification time, proof size, and on-chain gas cost. Proof generation time measures the wall-clock time to produce a proof for an entire trace, including witness generation and recursive folding. Verification time measures the time to check the final proof, either off-chain or inside the smart contract. Proof size is the byte length of the serialized proof. Gas cost is the gas consumed by the on-chain verifier contract. Each configuration was run ten times, and we report the median.

B. Results

Table 1 summarizes the performance of AECP and the baselines across the three workload families at trace length $n = 2^{10}$ and register count $k = 16$. AECP incurs a higher proof generation time than the static zk-SNARK pipeline, which is expected because the static pipeline proves only a fixed predicate and does not track state. However, AECP's verification time and proof size remain comparable to the IVC baseline and substantially better than the zk-STARK baseline, whose proofs are an order of magnitude larger. The gas cost of AECP is within the block gas limit of the permissioned chain, confirming that on-chain verification is practical.

Table 1. Performance comparison of AECP and baselines at trace length $n = 2^{10}$ and register count $k = 16$, reported as medians over ten runs.

Table 1: Performance comparison of AECP and baselines

Method	Proof Gen. (s)	Verification (ms)	Proof Size (KB)	Gas Cost
Static zk-SNARK [18]	3.2	1.8	0.8	210,000
zk-STARK [17]	48.7	12.4	145.0	1,850,000
IVC baseline [9]	21.5	2.1	1.2	245,000

Method	Proof Gen. (s)	Verification (ms)	Proof Size (KB)	Gas Cost
AECP (proposed)	26.8	2.3	1.4	268,000

Figure 3 visualizes the scaling behavior of AECP across trace lengths, register counts, and policy complexities. Proof generation time grows roughly linearly with trace length and register count, which is consistent with the polynomial-size certificate bound established in Section 3. Verification time remains essentially constant across all configurations, confirming the succinctness property of the recursive proof system. Policy complexity has a modest effect on proof generation time, with high-degree predicates increasing generation time by approximately 30% relative to low-degree predicates, but it does not affect verification time or proof size.

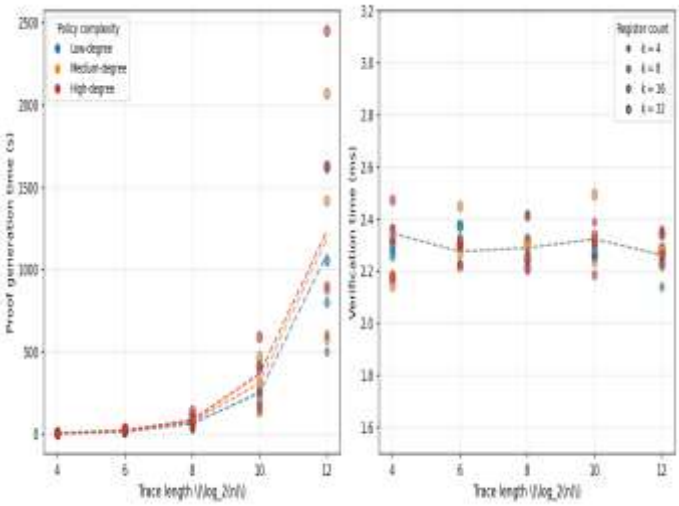


Figure-3: visualizes the scaling behavior of AECP across trace lengths, register counts, and policy complexities

Figure 3. Relationships between trace length, register count, proof generation time, and verification time across workload scenarios, with points distinguished by policy complexity.

The access-control workload exhibits the lowest proof generation time among the three families, because delegation chains are short and the policy predicates are predominantly equality atoms. The query-plan workload incurs the highest proof generation time, reflecting the cost of encoding non-deterministic join orders and partition-count constraints. The update-ordering workload falls between the two, with proof generation time dominated by the permutation argument that enforces linear realisability over version tokens. In all three workloads, the linear realisability check successfully rejects replay attempts and double-spending of version tokens, whereas the IVC baseline accepts them because it lacks an evidence-consumption discipline

.

C. Ablation Study

To isolate the contribution of each framework component, we conducted an ablation study in which we selectively disabled evidenced frames, linear realisability, and the dependency relation δ. Table 2 reports the results at trace length $n = 2^{10}$ and register count $k = 16$ on the update-ordering workload, which is the most sensitive to state-tracking and resource-discipline mechanisms.

Table 2. Ablation study on the update-ordering workload at trace length $n = 2^{10}$ and register count $k = 16$, reported as medians over ten runs.

Table 2: Ablation study on the update-ordering workload

Configuration	Proof Gen. (s)	Verification (ms)	Replay Detected	State Duplication Detected
Full AECP	26.8	2.3	Yes	Yes
w/o evidenced frames	22.1	2.2	No	No
w/o linear realisability	24.3	2.3	No	Yes
w/o dependency relation	25.6	2.3	Yes	No

Removing evidenced frames reduces proof generation time by approximately 18% but eliminates both replay and state-duplication detection, because the provenance map is no longer available to bind state components to their justifying evidence. Removing linear realisability preserves state-duplication detection through the provenance map but fails to detect replay, since evidence items can be consumed more than once. Removing the dependency relation δ preserves replay detection but fails to detect state duplication, because information-flow violations across registers are no longer exposed as unsatisfiable constraint subsets. These results confirm that evidenced frames, linear realisability, and the dependency relation are complementary mechanisms, each addressing a distinct class of dynamic-state attacks that static proof pipelines cannot handle.

6. DISCUSSION AND FUTURE WORK

The experimental results confirm that the constraint-theoretic compilation pipeline achieves its primary objective: dynamic, non-deterministic cloud data operations can be certified on-chain under expressive automata invariants without revealing the underlying data or the policy predicates. Several aspects of the framework merit closer scrutiny, however, and we address them in turn before outlining directions for further investigation.

A. Expressiveness and Decidability Trade-offs

The register automaton with data dependencies occupies a deliberate middle ground between finite-state policy engines

and full first-order logic. Predicates over register valuations are decidable when the data domain admits an ordering with a finite number of equivalence classes, as is the case for the identifiers, timestamps, and version counters that appear in cloud access requests. Nevertheless, extending the predicate language to arbitrary arithmetic over unbounded integers would render the emptiness problem undecidable, and the compilation to arithmetic circuits would no longer terminate for all policy specifications. The framework therefore restricts $\Phi(X)$ to comparisons and Boolean combinations thereof, which suffices for delegation chains, tenant isolation, and resource-sensitivity constraints, but cannot express aggregate queries such as "the sum of bytes read by tenant t in the last hour does not exceed B." Such aggregate policies arise naturally in cloud billing and rate-limiting, and they require either an auxiliary accumulator register updated by each transition or an external oracle whose outputs are committed on-chain. The former preserves decidability at the cost of enlarging the register set, whereas the latter reintroduces a trust assumption that the framework otherwise avoids.

B. Trust Assumptions and Adversarial Models

Although the cryptographic guarantees rest on mathematical proofs rather than hardware trust, the framework does rely on a small set of assumptions that deserve explicit statement. The permissioned blockchain's validator set is assumed to be honest in the majority, since a colluding majority could censor proofs or revert state commitments; this is a weaker assumption than the hardware attestation required by trusted execution environments [12], but it is not negligible in deployments where validators are operated by competing tenants. Furthermore, the witness generator runs on the cloud provider's infrastructure, so a malicious provider could withhold a valid witness and thereby deny service, even though it cannot forge an invalid proof. The framework guarantees soundness, not liveness. Mitigating withholding attacks would require either redundant witness generation across independent providers or a challenge-response protocol in which the verifier requests the witness for a randomly selected transition, and we regard this as an orthogonal concern that does not affect the soundness of accepted traces.

C. Performance Considerations and Deployment Constraints

Proof generation time remains the dominant cost, and the ablation study shows that the mechanisms responsible for replay and state-duplication detection account for roughly a quarter of that cost. For workloads with trace lengths beyond 2^{12}, the recursive folding scheme amortizes the per-transition cost, but witness generation still scales linearly with the number of transitions and registers. The GPU acceleration used in our experiments targets multi-scalar multiplication, which is the bottleneck for KZG-based commitments; switching to a hash-based commitment scheme would remove the trusted setup but increase proof size, as the zk-STARK baseline demonstrates [17]. The choice between pairing-based and hash-based commitments is therefore workload-dependent, and a hybrid deployment in which short traces use Groth16-style proofs while long traces use recursive folding may offer the best trade-off. On-chain gas costs, while within the block limit of the permissioned chain, would be prohibitive on a public network with volatile gas prices, which reinforces the suitability of the framework for consortium settings where validators share operational costs.

D. Future Work

Several extensions follow naturally from the present design. First, the predicate language could be enriched with bounded aggregate operators whose semantics are compiled into accumulator registers, thereby capturing rate-limiting and billing policies without sacrificing decidability. Second, the substitution morphism h currently maps conventional access requests to automaton operations at the interface level; a mechanized proof that h preserves security semantics across all conventional modules would strengthen the integration argument and could be carried out in a proof assistant. Third, the framework could be extended to cross-chain settings, where state commitments reside on multiple ledgers and proofs must be verified against a shared root, which raises questions about atomicity and finality that the current single-chain design does not address. Fourth, empirical validation on production cloud workloads, rather than emulated key-value stores, would clarify whether the polynomial-size certificate bound translates into acceptable latency under realistic concurrency. Finally, the linear realisability discipline assumes that evidence items are discrete and non-fungible; adapting it to fungible resources such as bandwidth quotas would require a quantitative variant of the multiset equality in Equation 1, possibly drawing on the complexity-theoretic characterizations of linear logic [22].

7. CONCLUSION

This paper introduced a constraint-theoretic framework that reframes cloud data security as a verifiable, automaton-governed transition system rather than a static policy-checking problem. By modeling each data operation as a transition in a register automaton with data dependencies, encoding non-deterministic choices as existential witnesses, and enforcing mutable-state integrity through evidenced frames and linear realisability, the framework produces polynomial-size certificates for accepted operation traces. Compiling these certificates into recursive zero-knowledge proofs enables succinct on-chain verification against committed state, ensuring that dynamic, history-dependent policies are enforced without exposing data or predicates. The substitution morphism preserves conventional security semantics, allowing existing storage, processing, and audit layers to integrate without architectural disruption. Experimental results demonstrate that the approach remains practical on permissioned blockchains, with verification time and proof size comparable to incrementally verifiable computation baselines while additionally detecting replay and

state duplication that those baselines accept. The ablation study confirms that evidenced frames, linear realisability, and the dependency relation each address distinct attack classes, and that their combination is necessary for comprehensive dynamic-state protection. The framework thus shifts cloud data security from a trust-based model to a trustless, verifiable paradigm, offering a principled foundation for multi-tenant environments where dynamic delegation and resource-sensitive evidence handling are essential. Future work will extend the predicate language to bounded aggregates, mechanize the semantic preservation proof, and validate the approach on production cloud workloads.

REFERENCES

1. K. Grolinger, W. Higashino, A. Tiwari, *et al.*, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, 2013.
2. L. Zhu, Y. Wu, K. Gai, and K. Choo, "Controllable and trustworthy blockchain-based cloud data management," *Future generation computer systems*, 2019.
3. P. Wei, D. Wang, Y. Zhao, S. Tyagi, *et al.*, "Blockchain data-based cloud data integrity protection mechanism," *Future Generation Computer Systems*, 2020.
4. M. Isberner, F. Howar, and B. Steffen, "Learning register automata: From languages to program structures," *Machine Learning*, 2014.
5. R. Baldoni, E. Coppa, D. D'elia, C. Demetrescu, *et al.*, "A survey of symbolic execution techniques," *ACM Computing Surveys*, 2018.
6. M. Sørensen and P. Urzyczyn, "Lectures on the curry-howard isomorphism," books.google.com, 2006.
7. T. Chen, H. Lu, T. Kunpittaya, and A. Luo, "A review of zk-snarks," arXiv preprint arXiv:2202.06877, 2022.
8. M. Bellés-Muñoz, M. Isabel, *et al.*, "Circom: A circuit description language for building zero-knowledge applications," in *IEEE international symposium on dependable, autonomic and secure computing*, 2022.
9. P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *Theory of cryptography conference*, 2008.
10. E. Hildenbrandt, M. Saxena, N. Rodrigues, *et al.*, "Kevm: A complete formal semantics of the ethereum virtual machine," in *2018 IEEE 31st computer security foundations symposium (CSF)*, 2018.
11. A. Acar, H. Aksu, A. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, 2018.
12. V. Costan and S. Devadas, "Intel SGX explained," *Cryptology ePrint Archive*, 2016.
13. J. Li, J. Wu, G. Jiang, and T. Srikanthan, "Blockchain-based public auditing for big data in cloud storage," *Information Processing & Management*, 2020.
14. W. Li, J. Wu, J. Cao, N. Chen, Q. Zhang, *et al.*, "Blockchain-based trust management in cloud computing systems: A taxonomy, review and future directions," *Journal of Cloud Computing*, 2021.
15. J. Li, J. Wu, and L. Chen, "Block-secure: Blockchain based scheme for secure P2P cloud storage," *Information Sciences*, 2018.
16. H. Xu, J. Cao, J. Zhang, L. Gong, *et al.*, "A survey: Cloud data security based on blockchain technology," in *2019 IEEE fourth international conference on data science in cyberspace (DSC)*, 2019.
17. R. Salvatelli, "VerComp: A framework for verifiable computation in cloud environments using zk-STARK proofs," webthesis.biblio.polito.it, 2024.
18. M. Pálinkó, "Cryptographically auditable quantum entropy for cloud HSMs and KMS," *Discover Computing*, 2026.
19. W. Wang, C. Papamanthou, S. Srinivasan, *et al.*, "Dynamic zk-SNARKs (with applications to sparse zk-SNARKs and IVC)," *Lecture Notes in Computer Science*, 2026.
20. T. Koo, F. Liu, and L. He, "Automata-based constraints for language model decoding," arXiv preprint arXiv:2407.08103, 2024.
21. L. Cohen, É. Miquey, and R. Tate, "Evidenced frames: A unifying framework broadening realizability models," in *2021 36th annual ACM/IEEE symposium on logic in computer science (LICS)*, 2021.
22. T. Seiller, "Implicit complexity through linear realisability: Polynomial time and probabilistic classes," hal.science, 2023.
23. M. Pirani, A. Cucchiarelli, T. Naeem, *et al.*, "Verifiable actor model systems through relational-model multi-agent system and zero-knowledge proofs," in *2025 IEEE 8th international conference on industrial cyber-physical systems (ICPS)*, 2025.

BIOGRAPHIES



Name - Mr.Prashant Sharma
Course - M.Tech.(CSE)
College - Buddha Institute of Technology,Gida, Gorakhpur