# Whom do you trust?
# PRISM: Lightweight Key Transparency for All

Sebastian Pusch[1], Ryan Quinn Ford[1], Joachim von zur Gathen[2], and Alexander Markowetz[1]

[1] Phillips-Universität Marburg, Germany
{ryanquinnford,sebastianpusch}@acm.org,
markowet@informatik.uni-marburg.de
[2] Universität Bonn, Germany
gathen@bit.uni-bonn.de

**Abstract.** End-to-end encrypted (E2EE) messaging platforms serving hundreds of millions of users face a fundamental vulnerability: users must trust service providers to distribute authentic public keys. This problem creates opportunities for sophisticated man-in-the-middle attacks and surveillance. While key transparency systems promise to eliminate this trust requirement, existing solutions have failed to achieve practical deployment due to prohibitive cost in computation and bandwidth, and inadequate infrastructure.

Our main innovation is the integration of a zero-knowledge virtual machine to create a "rollup" architecture on a third-party data availability layer via which every user automatically checks the integrity of the whole key directory. Counterintuitively, this approach yields substantial performance improvements over custom-built zk proof circuits and enables verification of targeted policies within the cryptographic proof system.

We introduce PRISM, the first practically deployable key transparency protocol that eliminates hidden backdoors in E2EE services through automatic, trust-minimized verification. Our system advances beyond previous approaches by proving not just structural validity of key directory updates, but their semantic correctness as well.

Previous solutions require some form of manual interaction by the user. This burden prevented wide spread adoption. Our solution however eliminates user intervention entirely.

This paper is intended as an overview rather than an exhaustive specification. Our implemented system[3] already integrates additional components whose full complexity exceeds the scope of this short presentation.

**Keywords:** key transparency · data availability · STARK · SNARK

## 1 Introduction

The proliferation of end-to-end encrypted communication systems has fundamentally transformed digital privacy and security, enabling billions of users to

[3] Source code available at https://github.com/deltadevsde/prism

communicate sensitive information with cryptographic guarantees of confidentiality. How much trust does this guarantee require? Such an assurance critically depends on the integrity of the underlying public key distribution mechanism.

Assume, for instance, that Alice initiates a secure communication with Bob via a messenger. Alice retrieves Bob's public key from the service provider. A corrupted or compromised messenger service could simply send Alice their own key instead, allowing them to possibly read, alter or inject messages.

This fundamental challenge in public key cryptography was articulated early in its development. As Zimmermann (1994) already observed:

> *This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. It is the 'Achilles heel' of public key cryptography, and a lot of software complexity is tied up in solving this one problem.*

Even modern key distribution mechanisms remain vulnerable to sophisticated attacks by adversaries capable of compromising key servers, let alone attacks by malicious service providers themselves.

In this paper, we introduce PRISM, a scalable key transparency protocol that enables real-time auditing by all users and prevents key tampering by construction. PRISM is the first practically deployable key-transparency solution to enable automatic verification of the service provider. This is achieved by providing constant size succinct proofs (SNARKs) to embedded light clients over a decentralized Data Availability (DA)(Al-Bassam et al. (2019)) layer.

Current approaches to key verification place substantial burdens on end users, requiring manual verification procedures such as comparing cryptographic fingerprints or scanning QR codes. Empirical research (Vaziripour et al., 2017) demonstrates that these manual verification methods are fundamentally impractical for widespread adoption. Only a small percentage of users successfully complete these verification procedures and an even smaller fraction understands their purpose or implications for security. This usability gap represents a critical barrier to the effectiveness of current security mechanisms, as security systems that cannot be used practically by their intended audiences fail to provide meaningful protection.

In contrast, PRISM removes this burden entirely: no intervention is required from the user. PRISM batches updates into epoch-based state transitions and produces succinct proofs that can be verified automatically by clients on user devices in the background, ensuring a globally consistent directory and preventing split-view attacks.

To achieve the latter, most of the prior works we present in the next chapter make the mysterious assumption of a public bulletin board (PBB): an incorruptible, append-only medium where updates are posted. In much of the literature the PBB is treated like a law of nature - gravity, speed of light, and the bulletin board. PRISM also relies on this abstraction, but unlike previous work, we state concrete requirements for such a PBB. Later, we show how a real-world data availability layer can provide exactly these properties.

Architecturally, PRISM is positioned between a data availability layer and end-user applications such as messenger services. By relying on multiple layers, PRISM leverages the immutability and global consistency of the DA layer while providing lightweight, verifiable proofs directly to clients.

In each epoch, PRISM publishes a succinct state commitment together with public succinct proofs attesting to the correctness of the state transition on the PBB. Crucially, end-user applications embed light clients that need only the latest commitment and proof to verify the entire history. This allows every user to obtain the same global view without trusting the service provider. Previous systems demand active intervention by a user to even verify her own keys at high costs of bandwidth and computing. PRISM does this automatically for all user and all keys with a lightweight load.

PRISM also makes a clear separation of concerns: the Key Directory (KD) maintains the authenticated, account-based state and proves correct batched updates, while Service Providers handle user authentication and label derivation.

Figure 1 illustrates the responsibilities and data flow in PRISM. Users register or update keys through the Service Provider (SP), which manages identifiers and public keys. PRISM is agnostic to the identifier format (e.g. raw, hashed, or VRFed). It proves valid state transitions, posts proofs to the PBB, and provides the SP with lookup proofs and stored data. Users then fetch the relevant proof data for the global state directly from the bulletin board while receiving lookup proofs and public keys from the SP. This design ensures that the SP cannot create a split-view, since every user can independently verify the global state.

## 2 Related Work

Traditional public key infrastructures (PKIs) operate under a “trust on first use” (TOFU) model, whereby users implicitly trust that the public keys they receive correspond to their intended communication partners. It is under this model that the aforementioned opportunities for man-in-the-middle attacks and surveillance arise. Some argue that TOFU is good enough. Trust will always be required at some level, and delegating key distribution to a reputable entity seems like a safe bet at first glance. We are unfortunately not only required to trust the entity’s own goodwill on first use. This trustworthy provider may control servers that have been compromised, or may later be compelled to interfere in key distribution for targeted users by a nation state. For instance, Microsoft created a backdoor in Skype in 2013 (Greenwald et al. (2013)) allowing the NSA to surveil all officially “end-to-end encrypted” messages.

Key Transparency improves upon TOFU by establishing cryptographically verifiable logs of public key bindings that enable all participants in a cryptographic system to audit and verify the consistency of key distribution. Drawing inspiration from Certificate Transparency (CT) (Laurie et al. (2013)) , which revolutionized the detection of misissued TLS certificates, key transparency extends these principles to provide comprehensive auditability for public-key mappings in secure messaging. The core insight is that by making key bindings publicly ob-

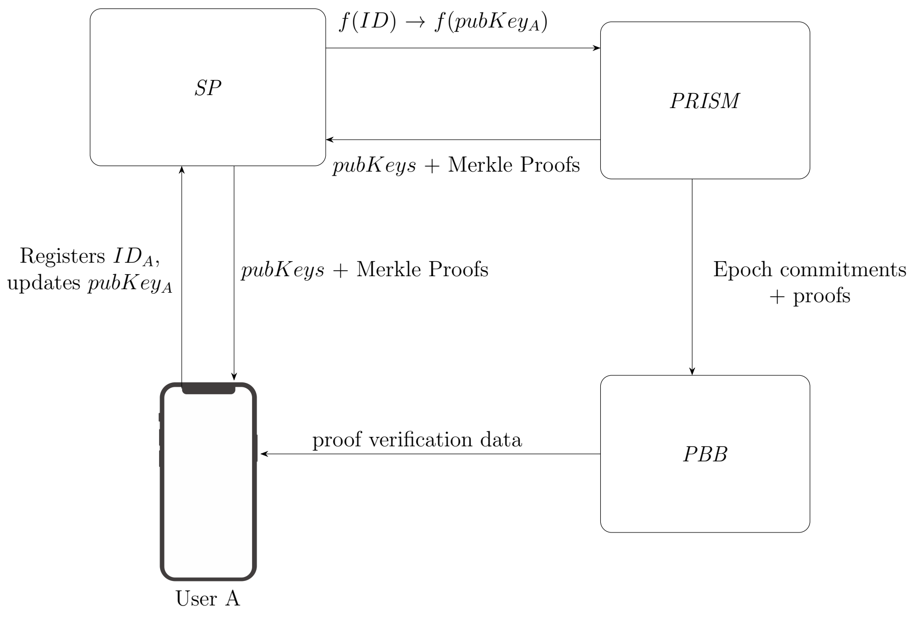
$f(ID) \rightarrow f(pubKey_A)$
$SP$
$PRISM$
$pubKeys$ + Merkle Proofs
Registers $ID_A$, updates $pubKey_A$
$pubKeys$ + Merkle Proofs
Epoch commitments + proofs
proof verification data
$PBB$
User A

**Fig. 1.** Responsibilities and data flow between Service Provider (SP), PRISM, Public Bulletin Board (PBB), and user.

servable and cryptographically verifiable through append-only logs, systems can detect and prevent attacks where adversaries present different keys to different users for the same identity.

The practical importance of key distribution has grown substantially as encrypted messaging platforms (such as Signal, WhatsApp (Lawlor and Levi (2023)), Zoom (Blum et al. (2023)), and iMessage) scale to serve hundreds of millions of users, where manual key verification becomes impractical and traditional PKI approaches fail to provide adequate security guarantees (Lawlor and Levi (2023)).

The existing Key Transparency (KT) systems build on the idea that service providers should not be blindly trusted to serve correct public keys. Instead, providers must commit to a global directory and publish cryptographic proofs so that inconsistencies can be detected. Since CONIKS (Melara et al. (2014)), a large body of work has refined this idea, each with different security goals and performance trade-offs. Despite the progress, a unifying characteristic is that all existing approaches rely on strong assumptions about external verification.

All current KT systems place the primary auditing burden on end users: each client is expected to monitor its own key state and detect if it has been misbound. In Melara et al. (2014) this requirement was explicit, but later systems such as Chase et al. (2018), Len et al. (2023), Malvai et al. (2023), and Len et al. (2024) follow the same model, differing mainly in how efficiently they allow users to fetch and verify their key history. In practice this means that users are still responsible for checking their own keys, while implicitly relying on all other users to do the same. If enough users behave diligently, inconsistencies will eventually be detected, but at internet scale this assumption is fragile: many users are often offline, inconsistent in their checks, or unwilling to perform continuous verification with multiple devices.

Many works (Melara et al. (2014), Chase et al. (2018), Hu et al. (2021), Len et al. (2023), Len et al. (2024)) propose gossip as a mechanism for detecting equivocation: clients exchange observed commitments with peers to ensure they all see the same global state. While elegant in theory, the described gossiping mechanisms require reliable out-of-band channels between otherwise unconnected users, and it is unclear how to deploy such mechanisms globally without partitions. Gossip can certainly serve as an additional safeguard, such as by embedding commitments in-band within encrypted messages, but it should not be relied upon as the sole consistency mechanism, since the assumption that gossip is both universal and timely remains too strong for practical deployment.

Another fundamental assumption underlying most KT designs is the existence of some public bulletin board or equivalent global append-only log. Bonneau (2016) and Tomescu and Devadas (2017) make this explicit by relying on Ethereum or Bitcoin, which are far too slow, heavyweight and expensive for most messaging settings. More recent systems like Chase et al. (2018), Tyagi et al. (2021), Tzialla et al. (2021), Len et al. (2023) and Len et al. (2024) implicitly assume that such a bulletin board exists, but rarely specify what exactly it is, how it is implemented, or how availability and censorship-resistance are ensured. In practice, this leaves a critical gap: without a concrete, lightweight realization of the bulletin board, consistency remains theoretical.

More recent systems introduce independent auditors or witness committees. Chase et al. (2018) formalizes the verifiable key directory and assumes that at least one honest auditor checks every update. Malvai et al. (2023) strengthens this by collecting signatures from about two thirds of the auditors, resembling Byzantine quorum systems. Furthermore, Len et al. (2023) and Len et al. (2024) assume the existence of dedicated entities (auditors or users acting as auditors) who monitor all commitments. Although more computationally efficient than gossip or blockchains, these designs still rely on the existence and long-term honesty and integrity of external parties, creating a significant deployment hurdle.

Another conceptual difference to prior systems is that they typically assume the service provider to maintain the key directory. Our design explicitly separates these roles: the messenger provider only consumes directory proofs, while the key directory is maintained independently. This separation changes the trust model and makes certain mechanisms (e.g., provider-controlled VRFs or their periodic rotation) less central. Instead, our focus is on minimizing external assumptions at the directory layer itself. This issue becomes even more acute with more interoperability requirements.

Across all these works, KT systems rely on one of four external mechanisms: (1) users verifying their own keys, (2) gossip among clients, (3) global bulletin boards or blockchains, or (4) third-party auditors or committees. Each mechanism shifts the trust boundary but does not eliminate it. In other words, all existing KT systems assume that someone else will check. This observation motivates our work fundamentally: to reduce these assumptions while retaining verifiability.

Previous SNARK-based approaches to key transparency, such as Verdict (Tzialla et al. (2021)), pioneered the idea of proving Merkle tree updates succinctly. However, these proofs established only structural validity, that the tree was updated correctly, but not semantic correctness, e.g. that updates were properly authorized. PRISM advances this paradigm by embedding authorization logic directly into its proofs: every account update must be signed by an existing valid key. This shift removes the need for client-side validation of update histories, reduces computational overhead, and strengthens security guarantees.

At the same time, we acknowledge an important usability concern noted in related work: many users struggle with key management in practice. Without recovery mechanisms, losing one's keys means irrevocably losing access to the account. We see approaches like social recovery (Linker and Basin (2024)) as promising complements to PRISM's architecture. While not yet implemented here, they represent a natural extension for future work to make key transparency not only verifiable, but also resilient to real-world user behavior.

Like in Verdict, the recursive construction of auditable epoch proofs remains computationally feasible. In optimal scenarios, verification of the latest epoch proof validates all historical updates since system genesis in constant time, providing both scalability and comprehensive auditability.

## 3 Technical Preliminaries

Previous literature has produced several sophisticated key transparency designs. CONIKS pioneered the use of Merkle prefix trees to enable efficient monitoring without revealing the full user directory (Melara et al. (2014)). Chase et al. (2018) then introduced a Privacy-Preserving Verifiable Key Directory that improved upon the scalability and privacy of CONIKs called SEEMless. ELEKTRA (Len et al. (2024)) further advanced these approaches by formalizing multi-device scenarios and incorporating post-compromise security guarantees using Rotatable Zero Knowledge Sets (Chen et al. (2022)). Verdict went down a new avenue, using succinct cryptographic proofs (SNARKs) to prove the correct operation of transparency dictionaries (Tzialla et al. (2021)). Each system addresses important aspects of the key transparency problem, from privacy preservation to formal security definitions.

### 3.1 Blockchains

A blockchain is a distributed ledger system that maintains a continuously growing list of records (blocks) linked through cryptographic hashes. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data. This creates an immutable chain where altering any single record would require changing all subsequent blocks, making tampering computationally infeasible. Blockchain networks operate through a consensus mechanism where multiple independent nodes agree on the validity and ordering of transactions. This distributed consensus eliminates the need for a trusted central authority while ensuring data integrity and availability across the network. Traditional blockchain architectures like Bitcoin (Nakamoto (2009)) and Ethereum (Buterin (2014)) are monolithic, meaning they bundle multiple core functions into a single system:

- Consensus: The process by which network participants agree on the valid state of the ledger
- Execution: The computation and processing of transactions and smart contracts
- Settlement: The finalization of transactions and state changes
- Data Availability: Ensuring that all data needed to validate the blockchain state is accessible to network participants

However, despite these theoretical advances, practical deployment of key transparency systems faces a fundamental infrastructure challenge: Where should transparency logs be hosted and how can their integrity be guaranteed (Lawlor and Lewi (2023))?

Current academic proposals generally assume the existence of reliable, tamper-resistant storage and distribution mechanisms without adequately addressing the underlying infrastructure requirements. This gap between theoretical design and practical deployment has limited real-world adoption of key transparency systems.

The infrastructure requirements for key transparency systems align closely with the properties that blockchain networks provide by design. Key transparency logs require three critical guarantees: immutable storage to prevent retrospective alteration of key bindings, global availability to ensure that all participants can access the same authoritative directory, and verifiable integrity to enable cryptographic proofs of correct operation. Traditional centralized approaches face inherent limitations in providing these guarantees. A single trusted server creates a point of failure and requires users to trust the hosting organization. Federated approaches distribute trust, but introduce complexity in maintaining consistency across multiple providers. Content delivery networks can provide availability, but typically cannot guarantee immutability without additional cryptographic infrastructure.

By contrast, blockchain networks are purpose-built to provide exactly these properties (Mazières and Shasha (2002)). A consensus mechanism ensures that once data is committed to the blockchain, it becomes computationally infeasible to alter it without detection. The peer-to-peer network overlay provides natural global availability without single points of failure. The cryptographic structure of blockchains enables efficient verification of data integrity and provenance.

Key transparency systems require a mechanism to publicly post and distribute critical data including audit logs, cryptographic proofs, and signed tree heads (STHs). This data must be globally accessible and verifiable to maintain the transparency properties that make these systems trustworthy. Historically, traditional blockchains have not been practical platforms for transparency log data because their monolithic architecture forces service providers and users to pay for all bundled functions, regardless of which ones they actually need. For a key transparency system posting data to Bitcoin or Ethereum, the cost includes:

- Consensus overhead for agreeing on transaction ordering
- Execution costs for processing data through the blockchain's virtual machine
- Settlement fees for finalizing state changes
- Data availability charges for distributing the data across the network

However, transparency logs only require data availability - the guarantee that posted data is accessible to all network participants. They do not need the blockchain to interpret and execute the posted bytes in their VM/state machine. The primary requirement is reliable data dissemination with cryptographic integrity guarantees. Data availability in blockchain systems ensures that when a block is proposed, all underlying data is published and retrievable by network participants. This is fundamental to blockchain security because unavailable data can prevent the network from distinguishing between valid and invalid state transitions, compromising both system liveness and safety properties (Al-Bassam et al. (2019)).

An additional technical barrier has been the lack of trust-minimized access to blockchain data. Clients that do not run their own blockchain nodes must trust external services to provide accurate blockchain state information, as these lightweight clients cannot independently verify block validity by participation in the peer-to-peer consensus network.

### 3.2 Celestia: Optimized Data Availability Layer

Celestia (Al-Bassam et al. (2019)) is our underlying blockchain because it is purpose-built to specialize in data availability, precisely the functionality that key transparency systems require. Unlike monolithic blockchains, Celestia focuses exclusively on ordering and publishing data, allowing applications like PRISM to avoid paying for unnecessary execution and settlement overhead.

Celestia's key innovation for transparency systems is its support for trust-minimized light clients. These lightweight clients can independently verify data availability and block validity without relying on trusted remote procedure calls or external data providers. This eliminates the trust assumptions that have historically limited blockchain adoption for transparency applications.

For PRISM, this capability enables embedding WebAssembly light clients directly into end-user applications. These embedded clients can access Celestia's common data layer to verify transparency log operations and prevent split-view attacks on the key directory's root hash. Simultaneously, the proliferation of these light clients strengthens Celestia's overall network security by increasing the number of independent validators monitoring data availability.

This architecture provides key transparency systems with the blockchain properties they need: immutable ordering backed financially by a large pool of assets, global data availability to publish this data to clients, and trust minimized access to the published data to prevent split view attacks. while eliminating the cost and complexity barriers that have prevented adoption of traditional monolithic blockchain platforms. Even well-resourced attackers, such as nation states, face a binary choice: acquire enough stake to censor/reorder or attempt a data-withholding attack; either outcome is detectable and allows honest participants to fork the blockchain. Such attacks are economically possible in theory, but costly, noisy, and risky in practice.

### 3.3 Zero-Knowledge Virtual Machines and State Management

This section introduces a core component underlying PRISM's architecture: Zero-Knowledge Virtual Machines (zkVMs). We provide sufficient background to understand how these technologies enable PRISM's verifiable computation and efficient state management, focusing on the practical considerations that informed our design choices.

Zero-Knowledge Virtual Machines represent a significant evolution in zero-knowledge proof systems, moving from hand-crafted zk proofs toward general-purpose platforms for verifiable computation. Traditional approaches require developers to manually translate computations into constraint systems (Thaler, 2022), zkVMs provide a generalized circuit over an Instruction Set Architecture (ISA), enabling developers to prove arbitrary programs while inheriting formal soundness and completeness guarantees.

The zkVM approach may initially seem counterintuitive–constraining every instruction in an execution trace appears computationally wasteful compared to custom zk proof circuits tailored to a single computation. Indeed, handwritten, special-purpose zk proof circuits would be more computationally efficient for narrowly defined problems. However, zkVMs address a critical practical limitation: the complexity and maintenance burden of writing low-level zk proofs by hand for diverse computational tasks becomes prohibitive for real-world applications.

This trade-off became apparent in previous work such as Verdict (Tzialla et al. (2021)), where the authors were prevented from implementing sparse Merkle tree approaches and adding constraints for application-specific policies due to the complexity of manual circuit construction. By proving the entire execution environment, zkVMs enable developers to prove statements from any conventional programming language that compiles to the target ISA.

Remarkably, zkVM efficiency improvements are outpacing Moore's Law, with tenfold (or more) speedups currently occurring every few months across the ecosystem. This rapid progress stems from the concentrated optimization efforts of multiple teams working on different layers, from novel arithmetization techniques and polynomial commitment schemes to instruction set optimizations and prover architectures (Arun, Setty, & Thaler (2023), Bruestle et al. (2023), Marin et al. (2024)). All of these improvements automatically benefit downstream applications without requiring rewrites of special-purpose zk proofs.

The fundamental architecture of a zkVM consists of several interconnected components:

- An ISA that defines the computational model
- An execution engine that processes instructions while maintaining detailed execution traces
- A proof generation system that transforms execution traces into zero-knowledge proofs

Modern zkVM implementations typically target established ISAs such as RISC-V or custom bytecode formats, enabling direct execution of programs compiled from high-level languages. During program execution, the zkVM maintains a comprehensive record of all computational steps, including instruction fetches, register updates, memory operations, and control flow changes. This execution trace serves as the foundation for proof generation, with each component subject to algebraic constraints that enforce computational correctness. The constraint system includes bound checking for memory accesses, consistency requirements for register states, and proper encoding of instruction semantics.

## 4 PRISM's Architecture

The central technological advancement that enables PRISM's design is the integration of a zero-knowledge virtual machine (zkVM), with a Jellyfish Merkle tree (Gao, Hu, & Wu (2021)) state structure. Counterintuitively, we discovered that combining a zkVM with a Merkle tree flavor optimized for compact proofs, despite being less zero-knowledge friendly, yielded substantial performance improvements despite zkVM overheads and we expect this to decrease further.

PRISM operates as a based rollup on the Celestia blockchain, specifically designed to provide data availability services for rollups. This architecture addresses both the cost and trust-minimization challenges that have historically prevented transparency logs from leveraging blockchain infrastructure.

As a sovereign based rollup, PRISM's transaction ordering is handled directly by Celestia's validator set as part of their normal block production. All submitted PRISM operations are included in Celestia blocks under PRISM's namespace, inheriting the full security and censorship resistance of Celestia's consensus mechanism.

The PRISM network consists of three specialized node types that handle different aspects of the transparency log operations:

- Sequencer/Prover: A single designated node that generates cryptographic proofs (specifically, epoch proofs) for operations in each PRISM block and publishes these proofs to a dedicated namespace (Al-Bassam (2019)) on

Celestia. This node ensures the integrity of state transitions through zero-knowledge proofs. It is important to note that in our trust minimized approach, this prover acts mainly as a batcher and proof generator; it does not have exclusive control over the data, since any user operation posted to the Celestia data layer will be seen by full nodes. This provides censorship resistance and ensures no single sequencer can covertly drop or alter updates, but comes at the costs of privacy.

– Full Nodes: These nodes download all raw operations from Celestia (from PRISM's transaction namespace) and apply every state transition, maintaining a local copy of the entire key directory state. Full nodes provide redundant validation regardless of the SNARK proof contents, ensuring robust verification of transparency log operations. In other words, full nodes don't need to trust the prover's zk proof – they verify integrity by replaying transactions themselves, ensuring the state machine rules are followed.
– Light Nodes: Lightweight clients that run on end-user devices and verify epoch proofs without downloading complete PRISM blocks or individual operations. Instead, a light node periodically downloads the latest zkSNARK proof from Celestia and uses it to verify the latest state root. Because the proof is succinct and constant-size, light clients can efficiently check that all prior operations were valid, without processing them individually. This enables embedded light clients (e.g. in a messaging app) to automatically audit the service's key directory in real-time. Thanks to Celestia's data availability sampling, these clients can directly access data from the Celestia network without relying on a centralized remote procedure call, preventing "split-view" attacks where a malicious provider might show inconsistent key directory states to different users

Why Celestia for Data Availability? Celestia's focus on data availability makes it ideal for PRISM's state and proof data. By posting data to Celestia, PRISM avoids the need to pay for expensive on-chain execution on a monolithic blockchain, yet still inherits the strong consensus and availability guarantees of a robust Layer-1. All PRISM operations and proofs are recorded in Celestia's ledger, ensuring that the key directory cannot be forked or hidden without detection. This choice significantly boosts security and scalability for the key transparency system, as Celestia's validators provide ordering and availability while PRISM's own proofs and nodes handle verification. The result is a censorship-resistant, cost-effective architecture: clients get the security of a decentralized ledger and zk proofs, without requiring every device to run a heavy blockchain node.

## 4.1 PRISM State Tree

This section introduces the storage mechanisms used for maintaining and verifying PRISM's state. We contrast the Indexed Merkle tree (IMT) from Verdict with our choice of the Jellyfish Merkle tree (JMT), describing the rationale for its better suitability in the context of zkVMs.

At the heart of PRISM is a key directory implemented as an account-based authenticated data structure and organized in a sparse Merkle tree. In simple terms, each user (or identity) has an Account entry in the directory, stored as a leaf in this global Merkle tree. An account contains the user's current public keys and associated data (e.g. metadata or application-specific fields). PRISM allows each account to hold multiple public keys (for example, to support multiple devices), and keys can be added or revoked by the user.

Every update operation to an account must be authorized by the user's existing key: the protocol requires that any AddKey or RevokeKey operation be signed by one of the account's currently valid (non-revoked) private keys. This ensures semantic correctness of state transitions – only the legitimate owner of an account can alter its keys. If an attacker or malicious service provider tried to inject an unauthorized key change, full nodes would reject the operation for failing the signature check, and no valid zkSNARK proof could be generated for it.

The transition from the hashchain model of prior approaches to an account model represents a crucial architectural shift. This change drastically reduces both storage and proving costs while simultaneously decreasing proof size and minimizing trust assumptions placed on the prover.

The Jellyfish Merkle tree (JMT) is introduced as a multi-versioned radix-16 addressable sparse Merkle tree (Gao et al. (2021)). The JMT introduces two main improvements over typical Addressable Radix Merkle trees. Empty subtrees are replaced with a constant placeholder value, and subtrees with single nodes are replaced with the node itself. These optimizations lead to a more concise proof format for both membership and non-membership proofs, with the number of sibling digests in a proof being logarithmic in the number of existent leaves.

JMT nodes use a versioned key design, enabling easy snapshot isolation of each epoch's state. Every update produces a new root (effectively a new "version" of the tree) while sharing unchanged parts with prior versions. This is ideal for PRISM's use case, as it creates an append-only history of state roots (one per block or epoch) without needing to copy the entire tree each time.

In a key transparency directory, the identifier space is typically very large (all possible phone numbers, email addresses, etc.), but only a small subset are "active". This creates a sparse directory: most potential slots have no entries.

A naïve Merkle tree would either force us to maintain a full fixed-height tree covering the entire identifier space, which means huge storage overhead and long proofs dominated by empty branches, or more complex linking mechanisms between nodes.

The Jellyfish Merkle tree avoids this problem by compressing the structure:

- It only materializes nodes along the paths that are actually used.
- For identifiers that are absent, it can produce short non-membership proofs either by:
    - showing the nearest populated neighbors in sorted order (proving that the queried identifier falls between them and isn't present), or

  - pointing to a placeholder node that represents an entire unused subtree.

Compared to naive Merkle Patricia trees (Wood, 2014), JMT proofs contain fewer sibling hashes on average, because the tree is structured to collapse long sparse paths. In practice, a JMT membership proof is often much shorter than a fixed-depth tree that includes many empty nodes. This translates to smaller proof sizes and faster verification.

Originally, PRISM's design (inspired by the research prototype Verdict from Tzialla et al. (2021)) represented each account's history as an append-only hashchain of operations rather than a mutable object. In that model, the Merkle tree stored, for each user, the hash of the latest entry in a linked list of all their operations (key adds/revokes, etc.). This was meant to avoid expensive on-chain verification of signatures – the idea was that clients could fetch the chain and verify each link and signature themselves, reconstructing the current keys. While this approach saved prover effort in early systems, it had notable downsides: unbounded growth of per-user state (the chain grows with every update) and complexity in client logic to replay all operations. PRISM departs from this hashchain model in favor of a direct account state. Since PRISM now proves the validity of operations and signatures inside the zkSNARK, we no longer need to burden clients with replaying every historical operation. Each account in PRISM's JMT simply reflects the latest authorized state (current set of valid keys, etc.), and the zk proof attests that all policy rules (like signature checks) were followed from the previous state to arrive at this new state. This shift to a mutable account model eliminates the unbounded growth problem and simplifies retrieval – a client can get the current keys in one proof look-up, rather than downloading an entire chain of updates.

We claim the JMT is better suited for zkVM programs in contrast to custom-tailored zk circuits (Chen et al. (2023)), where fixed-size Merkle path constraints do not apply. Crucially, this relaxation allows us to add circuit logic that verifies the operations on the tree values themselves, enabling the replacement of Verdict's hashchains with an account-based model. This significantly reduces the payload size and verification time for a client requesting keys for a user.

## 4.2 Experiments

To validate PRISM's scalability, we conducted benchmark tests simulating realistic deployment scenarios across varying user base sizes.

**Experimental Setup** Our evaluation models a messaging service with the following parameters:

- User base scale: 10,000 to 1,000,000 pre-existing accounts
- Daily growth rate: 1,000 new account registrations per day (41 per hour)
- Hourly operations:
  - New accounts: 1,000 new users a day, equalling 41 per hour (distributed across 12 five-minute epochs)

    - Key additions: 250 per hour
    - Key revocations: 20 per hour
- Epoch duration: 5 minutes (12 epochs per hour)
- Cryptographic primitives: Curve25519 signature scheme

This configuration reflects realistic usage patterns where most users maintain stable keys while a small fraction regularly add or remove devices.

**Measuring Prover Cost** To quantify proving effort, we use prover gas consumption, a metric introduced in (Succinct Labs, 2025). Unlike raw cycle counts, which treat all instructions equally, prover gas reflects the true cost profile of zkVM operations and correlates with actual proving times on GPUs. It accounts for factors such as precompiled operations and trace structure, producing a more accurate measure of real-world proving cost.

**Results and Analysis** Figure 2 shows prover gas across different user base sizes. As expected, it demonstrates a sublinear scaling. As the user base grows from 10,000 to 1,000,000 accounts, prover gas increases from 82.45 million units to 104.02 million.

This sublinear growth directly reflects the JMT's structural properties. As described in Section 4.1, the number of sibling digests in a JMT proof is logarithmic in the number of existent leaves. The JMT's optimizations produce more concise proofs than standard sparse Merkle trees. The zkVM dedicates the majority of its cycles to verifying these Merkle proofs during state transitions, resulting in the observed scaling behavior.

These results validate PRISM's viability for Internet-scale deployment, demonstrating that proving costs remain manageable even at millions of users without architectural changes.

## 5 Conclusion

We have introduced PRISM, the first industry-ready key transparency system that integrates a zero-knowledge virtual machine with a modular data availability layer. By embedding automated verification into light clients, PRISM removes the long-standing burden of manual key verification and prevents service providers from conducting equivocation attacks. This marks a decisive step from theoretical proposals to real-world deployment, showing that trust-minimized key transparency can be achieved without sacrificing usability.

Our architecture advances prior work in several ways: it proves both the structural and semantic correctness of directory updates; it leverages zkVMs to enable extensible, policy-aware proofs; it separates service providers from the key directory to minimize trust assumptions; and it operates as a sovereign rollup on a specialized data availability layer to ensure efficiency, scalability, and censorship resistance. Together, these innovations establish PRISM as a foundational building block for secure, interoperable communication infrastructures.

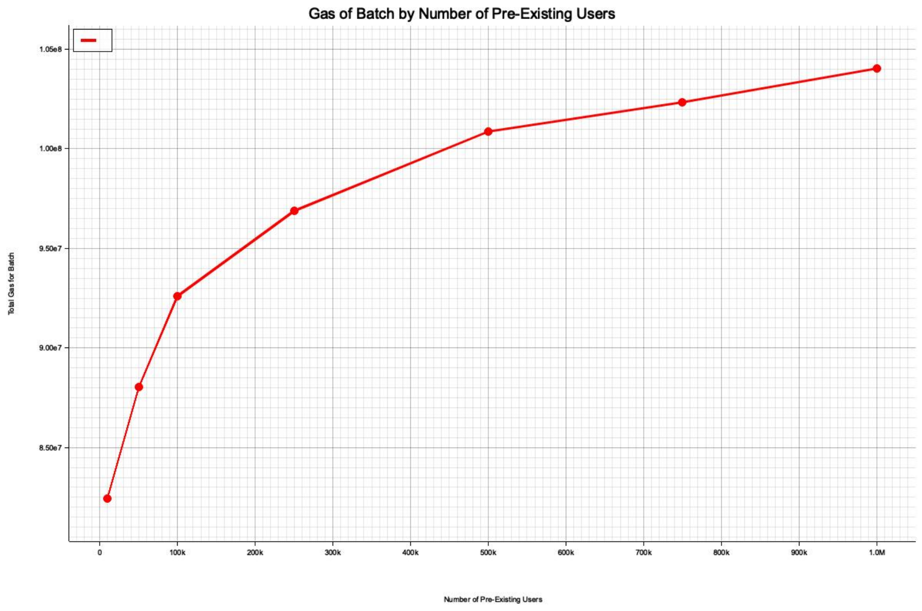

**Fig. 2.** Gas for one batch by number of pre-existing users.

Looking ahead, PRISM paves the way for full interoperability - arguably the single most decisive step in digitalization. The European Commission's Digital Markets Act (DMA) mandates interoperability in messenger services to dismantle entrenched platform silos. Equally, the vision of the Industrial Internet of Things (IIoT) requires millions of heterogeneous machines to exchange data reliably. Likewise, reducing bureaucratic burdens will depend on administrative systems worldwide being able to execute complete digital processes automatically and securely across organizational boundaries. To date, research has largely focused on common protocols while overlooking a critical element: the establishment of common address spaces. These must operate independently of application platforms and be designed to ensure reliability and auditability. PRISM delivers the missing piece of the puzzle.

PRISM itself is built on three foundational components: a Zero-Knowledge Virtual Machine, a Jellyfish Merkle Tree, and a data availability layer. Each of these domains is undergoing rapid and independent innovation. In particular, zero-knowledge systems are advancing at a pace far beyond Moore's law and poised to permeate an unprecedented breadth of applications. PRISM not only stands to benefit directly from these improvements - growing ever more practical and economically viable - but may also serve as a catalyst, driving new requirements and dedicated breakthroughs in all three components.

In the near future, our own efforts will focus on recovery mechanisms in the event of a compromised keystore, whether through attack or accident. Key questions include whether parts of the Merkle tree can be reused for querying and recovery, and how recovery processes can be accelerated more generally.

We invite researchers, developers, and practitioners to build upon PRISM—extending its design, exploring its applications, and deploying it widely—so that verifiable key transparency can become a global guarantee for secure and accountable digital communication.

## References

Al-Bassam, M. (2019, June). *LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts.* arXiv. Retrieved 2025-08-05, from http://arxiv.org/abs/1905.09274 (arXiv:1905.09274 [cs]) doi: https://doi.org/10.48550/arXiv.1905.09274

Al-Bassam, M., Sonnino, A., & Buterin, V. (2019, May). *Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities.* arXiv. Retrieved 2025-08-05, from http://arxiv.org/abs/1809.09044 (arXiv:1809.09044 [cs]) doi: https://doi.org/10.48550/arXiv.1809.09044

Albrecht, M. R., Mareková, L., Paterson, K. G., & Stepanovs, I. (2022, May). Four Attacks and a Proof for Telegram. In *2022 IEEE Symposium on Security and Privacy (SP)* (pp. 87–106). San Francisco, CA, USA: IEEE. Retrieved 2025-08-05, from https://ieeexplore.ieee.org/document/9833666/ doi: https://doi.org/10.1109/SP46214.2022.9833666

Arun, A., Setty, S., & Thaler, J. (2023). *Jolt: SNARKs for virtual machines via lookups.* Cryptology ePrint Archive, Paper 2023/1217. Retrieved from https://eprint.iacr.org/2023/1217

Ben-Sasson, E., Chiesa, A., Tromer, E., & Virza, M. (2014, August). Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *23rd USENIX Security Symposium (USENIX Security 14)* (pp. 781–796). San Diego, CA: USENIX Association. Retrieved from https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

Blum, J., Booth, S., Chen, B., Gal, O., Krohn, M., Len, J., ... Stamos, A. (2022). Zoom Cryptography Whitepaper.

Bonneau, J. (2016). EthIKS: Using Ethereum to Audit a CONIKS Key Transparency Log. In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, & K. Rohloff (Eds.), *Financial Cryptography and Data Security* (Vol. 9604, pp. 95–105). Berlin, Heidelberg: Springer Berlin Heidelberg. Retrieved 2025-08-05, from http://link.springer.com/10.1007/978-3-662-53357-4$_7$ (Series Title: Lecture Notes in Computer Science) doi: https://doi.org/10.1007/978-3-662-53357-4˙7

Brorsson, J., Pagnin, E., David, B., & Wagner, P. S. (2024). *Consistency-or-Die: Consistency for Key Transparency.* Retrieved 2025-08-05, from https://eprint.iacr.org/2024/879 (Publication info: Preprint.)

Bruestle, J., Gafni, P., & the RISC Zero Team. (2023). *Risc zero zkvm: Scalable, transparent arguments of risc-v integrity.* Retrieved from https://dev.risczero.com/proof-system-in-detail.pdf (Technical Report / Draft, accessed 1 October 2025)

Buterin, V. (2014). Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.

Chase, M., Deshpande, A., Ghosh, E., & Malvai, H. (2018). *SEEMless: Secure End-to-End Encrypted Messaging with less trust.* Retrieved 2025-08-05, from https://eprint.iacr.org/2018/607 (Publication info: Preprint. MINOR revision.)

Chen, B., Dodis, Y., Ghosh, E., Goldin, E., Kesavan, B., Marcedone, A., & Mou, M. E. (2022). *Rotatable Zero Knowledge Sets: Post Compromise Secure Auditable Dictionaries with application to Key Transparency.* Retrieved 2025-08-05, from https://eprint.iacr.org/2022/1264 (Publication info: A major revision of an IACR publication in ASIACRYPT 2022)

Chen, E., Gu, B., Lawrence, B., Shi, E., & Zhang, Y. X. (2023). *Programmable cryptography: Four easy pieces.* 0xPARC Foundation. Retrieved from https://0xparc.org (Accessed: 2025-10-01)

Crosby, S. A., & Wallach, D. S. (2009). Efficient Data Structures for Tamper-Evident Logging.

Gao, Z., Hu, Y., & Wu, Q. (2021). Jellyfish Merkle Tree. Retrieved from https://developers.diem.com/docs/technical-papers/jellyfish-merkle-tree-paper/

Google. (n.d.). *Key Transparency Overview.* Retrieved 2025-08-05, from https://github.com/google/keytransparency/blob/master/docs/overview.md

Greenwald, G., MacAskill, E., Poitras, L., Ackerman, S., & Rushe, D. (2013, July). Microsoft handed the NSA access to encrypted messages. *The Guardian.* Retrieved 2025-08-15, from https://www.theguardian.com/world/2013/jul/11/microsoft-nsa-collaboration-user-data

Hicks, A. (2023, September). *SoK: Log Based Transparency Enhancing Technologies.* arXiv. Retrieved 2025-08-05, from http://arxiv.org/abs/2305.01378 (arXiv:2305.01378 [cs]) doi: https://doi.org/10.48550/arXiv.2305.01378

Hu, Y., Hooshmand, K., Kalidhindi, H., Yang, S. J., & Popa, R. A. (2021). *Merkle^2: A Low-Latency Transparency Log System.* Retrieved 2025-08-05, from https://eprint.iacr.org/2021/453 (Publication info: Published elsewhere. Minor revision. IEEE S&P 2021) doi: https://doi.org/10.1109/SP40001.2021.00088

Keybase. (2019). *Keybase is not softer than TOFU.* Retrieved 2025-08-05, from https://keybase.io/blog/chat-apps-softer-than-tofu

Laurie, B., Langley, A., & Kasper, E. (2013, June). *Certificate Transparency* (Tech. Rep. No. RFC6962). RFC Editor. Retrieved 2025-08-05, from https://www.rfc-editor.org/info/rfc6962 doi: https://doi.org/10.17487/rfc6962

Lawlor, S., & Lewi, K. (2023, April). *Deploying key transparency at WhatsApp.* Retrieved 2025-08-05, from https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/

Len, J., Chase, M., Ghosh, E., Jost, D., Kesavan, B., & Marcedone, A. (2024). *ELEKTRA: Efficient Lightweight multi-dEvice Key TRAnsparency.* Retrieved 2025-08-05, from https://eprint.iacr.org/2024/107 (Publication info: Published elsewhere. Major revision. ACM CCS 2023)

Len, J., Chase, M., Ghosh, E., Laine, K., & Moreno, R. C. (2023). *OPTIKS: An Optimized Key Transparency System.* Retrieved 2025-08-05, from https://eprint.iacr.org/2023/1515 (Publication info: Published elsewhere. Major revision. USENIX Security '24)

Linker, F., & Basin, D. (2024, August). SOAP: A social authentication protocol. In *33rd usenix security symposium (usenix security 24)* (pp. 3223–3240). Philadelphia, PA: USENIX Association. Retrieved from https://www.usenix.org/conference/usenixsecurity24/presentation/linker

Malvai, H., Kokoris-Kogias, L., Sonnino, A., Ghosh, E., Oztürk, E., Lewi, K., & Lawlor, S. (2023). *Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging.* Retrieved 2025-08-05, from https://eprint.iacr.org/2023/081 (Publication info: Published elsewhere. NDSS)

Marin, D., Abdalla, M., Govereau, P., Groth, J., Judson, S., Sosnin, K., ... Zhang, Y. (2024). *Nexus 1.0: Enabling verifiable computation.* Whitepaper, Nexus Labs. Retrieved from https://whitepaper.nexus.xyz/ (Accessed: 2025-10-01)

Mazières, D., & Shasha, D. (2002, July). Building secure file systems out of byzantine storage. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing* (pp. 108–117). New York, NY, USA: Association for Computing Machinery. Retrieved 2025-08-05, from https://doi.org/10.1145/571825.571840 doi: https://doi.org/10.1145/571825.571840

McMillion, B., & Linker, F. (2025, July 6). *Key Transparency Protocol* (Internet-Draft No. draft-ietf-keytrans-protocol-02). Internet Engineering Task Force. Retrieved from https://datatracker.ietf.org/doc/draft-ietf-keytrans-protocol/02/ (Work in Progress)

Melara, M. S., Blankstein, A., Bonneau, J., Felten, E. W., & Freedman, M. J. (2014). *CONIKS: Bringing Key Transparency to End Users.* Retrieved 2025-08-05, from https://eprint.iacr.org/2014/1004 (Publication info: Published elsewhere. USENIX Security '15)

Micali, S., Vadhan, S., & Rabin, M. (1999). Verifiable Random Functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science* (p. 120). USA: IEEE Computer Society.

Nakamoto, S. (2009). Bitcoin: A Peer-to-Peer Electronic Cash System.

Ryan, M. D. (2013). *Enhanced certificate transparency and end-to-end encrypted* *mail.* Retrieved 2025-08-05, from https://eprint.iacr.org/2013/595 (Publication info: Published elsewhere. Major revision. Network and Distributed System Security (NDSS) 2014)

Signal Foundation. (2025). *Technical Information.* Retrieved 2025-08-05, from https://signal.org/docs/

Succinct Labs. (2025). *Prover gas.* Succinct Docs (SP1). Retrieved from https://docs.succinct.xyz/docs/sp1/optimizing-programs/prover-gas (Accessed: 2025-10-01)

Thaler, J. (2022). Proofs, arguments, and zero-knowledge. *Foundations and Trends in Privacy and Security*, *4*(2–4), 117–660. Retrieved from http://dx.doi.org/10.1561/3300000030 doi: https://doi.org/10.1561/3300000030

Tomescu, A., Bhupatiraju, V., Papadopoulos, D., Papamanthou, C., Triandopoulos, N., & Devadas, S. (2018). *Transparency Logs via Append-only Authenticated Dictionaries.* Retrieved 2025-08-05, from https://eprint.iacr.org/2018/721 (Publication info: Published elsewhere. ACM CCS 2019)

Tomescu, A., & Devadas, S. (2017). *Catena: Efficient non-equivocation via bitcoin.* Retrieved from https://api.semanticscholar.org/CorpusID:30108327

Tyagi, N., Fisch, B., Zitek, A., Bonneau, J., & Tessaro, S. (2021). *VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries.* Retrieved 2025-08-05, from https://eprint.iacr.org/2021/627 (Publication info: Published elsewhere. ACM CCS 2022)

Tzialla, I., Kothapalli, A., Parno, B., & Setty, S. (2021). *Transparency Dictionaries with Succinct Proofs of Correct Operation.* Retrieved 2025-08-05, from https://eprint.iacr.org/2021/1263 (Publication info: Preprint. MINOR revision.)

Vaziripour, E., Wu, J., O'Neill, M., Whitehead, J., Heidbrink, S., Seamons, K., & Zappala, D. (2017, July). Is that you, alice? a usability study of the authentication ceremony of secure messaging applications. In *Thirteenth symposium on usable privacy and security (soups 2017)* (pp. 29–47). Santa Clara, CA: USENIX Association. Retrieved from https://www.usenix.org/conference/soups2017/technical-sessions/presentation/vaziripour

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, *151*(2014), 1–32.

Zimmermann, P. (1994, October). PGP User's Guide, Volume I: Essential topics [Computer software manual]. Retrieved from https://web.pa.msu.edu/reference/pgpdoc1.html (PGP Version 2.6.2)