Signet: Scalable Network-Driven Proof of Notification for Blockchain Systems

Elham Ehsani Moghadam* Marc Wyss* Jonghoon Kwon* Marc Frei*
Yih-Chun Hu† Adrian Perrig*‡ Alberto Sonnino‡§
*ETH Zurich †University of Illinois at Urbana-Champaign ‡Mysten Labs §University College London

***Abstract*—In blockchain systems, event notification is a valuable feature that eliminates the need for clients to actively monitor each event recorded by the blockchain. However, providing proof of notification—a verifiable guarantee that a notification was sent—is challenging to scale in high-throughput environments. This paper presents Signet, a novel system that enables the network itself to provide scalable and verifiable proof of notification. Leveraging path-aware networks, Signet utilizes Autonomous Systems (ASes) along the agreed-upon traffic path as impartial witnesses. These ASes, relied upon by both the notifier and recipient, provide lightweight proofs of packet observation to be submitted to the blockchain. This approach minimizes computational overhead and storage costs while maintaining security and scalability. Our evaluation demonstrates that the routers can handle over a million notifications per second per core, with under a microsecond of per-packet overhead, and that the smart contract operations on the blockchain incur negligible costs.**

***Index Terms*—Blockchain, Proof of Notification, Autonomous Systems, Smart Contracts, Path-Aware Networking**

I. Introduction

In distributed systems where trust is limited [1], proving that a message was sent and received remains a fundamental challenge [2]–[4]. This problem is particularly acute in blockchain ecosystems, where participants frequently interact without mutual trust [5]. Many blockchain applications depend on the timely detection of on-chain events, such as payment completions, contract state transitions, or transaction confirmations. These notifications are critical for automating business logic, such as releasing digital goods after payment or reacting to oracle updates [6]. In latency-sensitive contexts, such as decentralized exchanges, liquidation engines, or automated settlement systems, delayed or missed notifications can directly cause significant financial loss [7].

When a notification is lost or delayed, assigning blame is difficult: the notifier may insist the message was sent (perhaps multiple times), while the recipient may deny its arrival, even if it was received. This ambiguity motivates the need for verifiable proof of notification: how can a notifier prove, in a decentralized setting, that a message was sent? Proof-of-notification systems aim to resolve this by collecting cryptographic evidence and submitting it to smart contracts that enforce service-level agreements through financial incentives [8]. Typically, notifiers and recipients lock collateral, and disputes are adjudicated through on-chain proofs [9].

Existing solutions have shown success in environments with high latency tolerance and modest throughput demands in the kilobyte to megabyte range, as seen in traditional blockchain systems. They, however, struggle under the stringent performance demands of modern blockchain applications, which operate in a latency-sensitive setting [10], [11] and require gigabit-scale throughput [12].

These solutions typically fall into two broad categories: The first approach relies on independent third parties [13], such as committees or hand-picked nodes trusted by both notifier and recipient, to mediate delivery and witness message transmission. While viable in some contexts, this model is difficult to generalize in open and decentralized settings, where discovering mutually trusted entities is non-trivial. Moreover, these designs may introduce bottlenecks as the witnesses need to receive and process every message, adding network latency and potentially throttling throughput [13].

The second approach, often employed in systems such as payment channels [9], [14], [15], relies on acknowledgments (ACKs) from recipients, often subdividing messages into smaller chunks tied to micro-payments. This mechanism suffers when recipients act maliciously by withholding ACKs, and the required round-trips introduce latency on the forwarding path. These designs also impose computational overhead on both the notifier and recipient, typically relying on public-key cryptography, making them unsuitable for high-throughput, latency-sensitive applications.

A practical proof-of-notification protocol must therefore satisfy several stringent requirements: add negligible latency to the critical path of message delivery, which includes processing delays and forwarding latency, support high throughput by handling millions of notifications per second, scale with the number of participants and messages, and provide both accountability and compatibility with the trust assumptions of current decentralized systems.

Satisfying these requirements entails three main challenges: (i) Identifying trusted third parties is difficult; it is often unclear whom to trust. (ii) Mediating message delivery and requiring multiple acknowledgment round-trips introduces observable latency overhead. (iii) Requiring witnesses to pro-

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) under Grant No. RS-2024-00440780, "Development of Automated SBOM and VEX Verification Technologies for Securing Software Supply Chains," as well as by the Werner Siemens Stiftung (WSS) Centre for Cyber Trust at ETH Zurich, armasuisse Science and Technology, and the Zurich Information Security and Privacy Center (ZISC).

Corresponding author: Elham Ehsani Moghadam (eehsani@inf.ethz.ch)

cess every message or offloading expensive cryptographic operations to the edge of the network creates throughput bottlenecks. These limitations become particularly evident for high workloads, such as those produced by decentralized data stores [12], and latency-sensitive applications, such as decentralized finance [11], where even a few hundred extra milliseconds can have a significant impact.

In this paper, we present Signet,[1] a new protocol for verifiable proof of notification. Signet overcomes the above challenges by introducing a novel architecture for scalable and verifiable proof of notification, built on path-aware networking (PAN) [16]. Signet is built on three key design principles. To overcome challenge (i), it leverages Autonomous Systems (ASes) along the notification packets' forwarding path as natural witnesses. ASes already mediate packet delivery and are implicitly trusted by both the notifier and recipient for transport, and their geographic and topological distribution contributes to load balancing and robustness. (ii) Signet leverages these ASes' border routers to generate a lightweight, compact cryptographic proof of notification packet observation. To allow processing millions of notifications per second and minimizing critical path latency, Signet employs a TESLA-based broadcast authentication scheme [17], relying solely on symmetric cryptography. (iii) Signet enables the notifier to obtain a proof of notification that is verifiable on-chain without requiring synchronous acknowledgments or additional handshakes. Proofs are transmitted asynchronously to the notifier, who later submits them to a smart contract for arbitration.

Together, these design principles make Signet compatible with existing decentralized trust models by leveraging existing trust relationships between participants, and operate at wire speed by embedding the proof generation process into the network fabric and decoupling it from application logic.

We implement a prototype of Signet and evaluate it in a controlled environment. Our results show that Signet supports gigabit-scale throughput while adding only negligible latency on the order of microseconds. It imposes minimal computational overhead on participating ASes.

In summary, this paper makes the following contributions.

- We identify transit ASes as natural witnesses for attesting packet-level notifications.
- We propose Signet, a new protocol for verifiable proof of notification, combining this observation with the lightweight TESLA protocol. To enable this combination, we design a secure scheme for using TESLA-based authentication, tailored for efficient proof generation by ASes and asynchronous on-chain verification. We design Signet with a focus on deployability in existing infrastructures and integration with modern blockchain systems.
- We present a structured security analysis of Signet, demonstrating its resistance to arbitrarily Byzantine adversarial behaviors of end entities.
- We implement a Signet prototype and evaluate it in a realistic environment. Signet introduces negligible latency and supports gigabit-scale throughput by handling millions of notifications per second, with minimal computational overhead.

[1]Our system is named Signet (a small seal, often on a ring) to evoke the idea of a witness affixing a cryptographic evidence of having observed a notification, similar to a signet's impression.

II. Motivation and Background

We first motivate the need for Signet as a system for generating and recording verifiable, timely proofs of notification. We then motivate its key design principles, including the use of transit ASes as witnesses, PAN for routing, and TESLA for authentication.

A. From Altruistic to Paid Notifications

In blockchain ecosystems, unlike full nodes that store the entire blockchain history and validate every transaction, light clients (e.g., a Crypto wallet) interact with the blockchain while maintaining only minimal local state. Instead of storing and processing the full ledger, light clients rely on data providers, including full nodes, Remote Procedure Call (RPC) servers, and indexers, for state information and cryptographic proofs to verify transactions without maintaining full history. In common polling-based architectures, light clients repeatedly query these providers to retrieve missing state updates or validate transactions.

This model places a burden on full nodes, which must respond to potentially large volumes of read requests. Full nodes traditionally serve light clients without direct compensation, assuming their stake in the network's success is sufficient motivation. However, this rationale is increasingly unsatisfactory for several reasons. First, it is vulnerable to a tragedy of the commons, where individual incentives undermine collective benefit. Second, the continued provision of services to light clients is largely due to inertia—full node operators have not modified the default code, which is usually maintained by validators. Third, while serving numerous light clients was economically viable in low-throughput blockchains, increased throughput has rendered this cost prohibitive for full node operators.

Polling also struggles in contexts where timeliness is critical; delayed or missed notifications can cause significant financial loss, especially when state updates occur unexpectedly and polling queries are too infrequent. For example, a trader using automated strategies to monitor and react to price updates on a decentralized exchange, a borrower protecting collateral from liquidation when it falls near a threshold, or a settlement service finalizing cross-chain transfers can all suffer missed opportunities or immediate financial losses if notifications arrive late. In such cases, a push-based, asynchronous notification mechanism becomes more efficient, where the full nodes actively inform the interested party when the desired state change occurs. However, much like frequent polling, this per-event approach imposes even greater workload on full nodes, requiring appropriate compensation to ensure reliable service. To compensate for the workload of handling notifications, full nodes may require light clients to subscribe on a paid basis.

B. The Need for Proof of Notification

A paid subscription model necessitates mechanisms to ensure that full nodes dispatch all promised notifications. One approach involves full nodes placing a deposit and receiving a fee from light clients. Should a notification fail to be sent within an agreed time, light clients would be entitled to reclaim the deposit. This approach requires clear accountability. When a dispute arises, the notifier must be able to provide *evidence* that the notification was sent (*proof of notification*). Failure to do so entitles the light client to compensation for a service-level agreement (SLA) breach. A *dispute resolution* entity is also needed to verify the evidence.

A proof of notification must meet key requirements: it should be independently verifiable, ensure authenticity, non-repudiation, and resist forgery. Additionally, it must support high throughput and scale efficiently without imposing significant computational overhead. Naïve approaches, such as requiring recipients to send signed acknowledgments, are impractical since clients may be offline, experience connectivity issues, or even maliciously deny having received the notification. A trusted third party (TTP) could issue proofs, but identifying entities that are simultaneously trustworthy, efficient, and scalable is challenging.

To address these limitations, we propose leveraging transit ASes as natural witnesses. ASes are already trusted to forward traffic, and their involvement avoids introducing new trust dependencies. Selecting transit ASes for proof generation reduces the risk of bottlenecks or targeted attacks, as congestion and DDoS attacks typically occur at the network edges rather than in the core [18]. Furthermore, with multiple witness ASes available along a path, an adversary would need to compromise or disrupt several independent entities to suppress the notification or its proof. However, achieving scalability while generating cryptographically secure, verifiable proofs for every notification remains a substantial challenge.

C. Use Case

To illustrate the practical applicability of Signet, consider the scenario of Alice, a blockchain light client user who operates an automated trading bot on a decentralized exchange. Alice relies on timely notifications about asset price changes and transaction finalizations to execute profitable trades and protect her positions from potential liquidation events.

In traditional settings, Alice would face a stark choice between two extremes. (i) High-stakes, ultra-low-latency users (e.g., in high-frequency trading or arbitrage scenarios) that operate a dedicated full node to bypass the network transmission delay of notifications, guarantee the fastest and most reliable access to blockchain events, regardless of cost. (ii) Low-value, less latency-sensitive users (e.g., retail investors monitoring their wallet balances or users awaiting confirmation of non-urgent transactions) might be content to occasionally poll public RPC endpoints or rely on opportunistic free notifications, tolerating delays or occasional missed events.

However, many practical users fall between these extremes. Alice handles transactions where missed or delayed notifications could result in meaningful, though not catastrophic, financial loss. She and similar users are willing to pay a reasonable fee for timely delivery, but running a full node is economically unjustifiable.

Examples of such scenarios include delayed order execution on a decentralized exchange, missing the settlement of an NFT auction, submitting collateral deposits too late to avoid liquidation, or failing to promptly confirm business payments conducted in cryptocurrency. In these cases, timely notifications help users avoid meaningful financial loss, even though the potential consequences are not catastrophic. They also allow users to minimize losses or maximize profit by enabling quicker reactions to on-chain events.

Signet provides an attractive middle-ground solution. Alice selects from multiple competing notifiers that offer notification services backed by on-chain deposits and explicit service-level agreements (SLAs). She subscribes to notifiers who are contractually obliged, through staked collateral, to send notifications promptly for specific events (e.g., DEX swaps, oracle price updates, or impending liquidations). If a notifier fails to send a timely notification, Alice, if proven correct, can claim the notifier's deposit as compensation for the broken SLA. This economic model incentivizes notifiers to maintain a high level of service while providing Alice with recourse in the case of missed or delayed notifications. The deposit-and-dispute approach should be economically sustainable for both parties. For Alice, subscription fees are lower than the cost of operating her own full node and still guarantee better service and legal recourse than best-effort notification. For notifiers, serving many such clients is profitable as long as they maintain SLA compliance and deposits are calibrated, ensuring that, for example, they are not excessively penalized in rare cases of failures or errors. Additionally, Alice can select multiple notifiers in parallel or switch providers at will, fostering a competitive market for notification services.

D. From Forwarding to Witnessing a Packet

A central insight in Signet's design is the repurposing of ASes along the network path from mere forwarders of data packets to impartial witnesses that generate verifiable proofs of notification. Unlike traditional approaches, which rely on either application-layer third parties or recipient acknowledgments, Signet leverages entities that already mediate data delivery in the network. The transition from simple forwarding to accountable witnessing raises several important questions: (i) Why should ASes be trusted to serve as witnesses? (ii) What incentives do they have to participate? and (iii) How is this new role compatible with their existing operations?

(i) Trust. In practice, notifiers and recipients (e.g., blockchain nodes and clients) implicitly trust on-path ASes to forward their packets correctly. By virtue of this existing trust, ASes are natural candidates for attesting to the fact that specific packets have traversed the network at a particular time.

Moreover, in Signet, misbehavior by a witness is detectable by notifiers or recipients, who can disregard unreliable ASes for future notifications. The trust model thus becomes more

distributed, with misbehaving or non-participating ASes simply bypassed in favor of others willing to participate and fulfill their witnessing commitment. Furthermore, leveraging multiple, topologically diverse witness ASes increases robustness.

(ii) Incentives. While forwarding packets is a core business of ASes, actively generating and returning proofs of packet observation is a new value-added service. The incentives for ASes to participate in witnessing are twofold:

Monetary reward: ASes can charge nominal fees for proof services, opening new revenue streams in the evolving Internet infrastructure economy. The design of such payment and pricing mechanisms, however, is largely orthogonal to Signet's core design.

Competitive differentiation: Offering verifiable notification services can make ASes more attractive to customer ASes and service providers that desire higher-tier support and accountability. The market for such services is reinforced by the growing demand for reliable and auditable cross-domain event notifications.

(iii) Minimal Overhead. For widespread adoption, the witnessing function must introduce negligible complexity or operational disruption to ASes. The overhead for ASes is intentionally kept minimal in Signet: TESLA-based proof generation is computationally inexpensive and can be performed at wire-speed without altering normal forwarding behavior. Additionally, since Signet relies on Path-Aware Networking (PAN) [19], the selection of ASes as witnesses is flexible, supporting gradual rollout as more ASes opt in.

E. Background

Path Control with Path-Aware Networking. Path-Aware Networking (PAN) is a network architecture that gives end hosts visibility into and control over the paths their packets take [19]. Unlike today's Internet, where routing is hop-by-hop and largely opaque to endpoints, PAN enables hosts to explicitly select from a set of available end-to-end paths. While our design is not tied to any specific PAN architecture, we select SCION [20] for our data plane implementation. SCION has the advantage of being already deployed and operational in real-world production networks, with a mature implementation [21]–[23]. Moreover, SCION includes extensions that allow network-layer source authentication: namely SPAO [24] and EPIC [25].

TESLA for Efficient Authentication. The TESLA (Timed Efficient Stream Loss-tolerant Authentication) protocol [17] is an authentication scheme designed for environments where computational and bandwidth efficiency are critical. Instead of relying on expensive public-key operations, TESLA uses symmetric message authentication codes (MACs). TESLA divides time into intervals (i), each associated with a secret key K_i, known only to the sender. K_i is used to compute MACs during time interval i, but it is only disclosed after a certain delay. While this delays verification, once the key is disclosed, receivers can confirm both the key's legitimacy and the message's authenticity (see details in Appendix A).

TESLA is not a digital signature scheme and lacks inherent non-repudiation, since any party can forge valid MACs after key disclosure. However, in conjunction with a trusted timestamping service such as a blockchain, TESLA can achieve non-repudiation [26]. Specifically, if a TESLA-authenticated message is irrevocably recorded on-chain *prior* to the disclosure of the corresponding key, its origin and authenticity become verifiable and tamper-resistant. This requires loosely synchronized clocks with a known maximum offset to ensure message commitment precedes key disclosure. TESLA's lightweight authentication make it well-suited for systems requiring a high-throughput authentication scheme.

III. System Model

This section presents our system model, detailing its key components, underlying assumptions, the adversary model, and the design goals.

A. Key Components

Notifier. A full blockchain node that monitors the chain for relevant on-chain events and sends corresponding off-chain notifications to subscribed recipients. The notifier receives payments for its services and locks a deposit on-chain as a commitment to a timely and verifiable notification service.

Recipient. A blockchain client that subscribes to one or more event types and pays notifiers for timely notification. In case of a breach, i.e., when a notification is not sent on time, the recipient can initiate a challenge to claim the notifier's deposit.

Witness. A trusted network entity, specifically, an AS, positioned along the path between the notifier and the recipient. It observes notification packets in transit and generates timestamped lightweight proofs and returns them to the notifier.

Dispute Resolver. A smart contract deployed on the blockchain that resolves disputes regarding notification. It keeps relevant state about subscriptions, deposits, and commitments. Upon a recipient's claim of a missed notification, the contract verifies submitted proofs and corresponding timestamps to determine whether the notifier fulfilled its obligation. Based on the outcome, it either releases the notifier's deposit or transfers it to the recipient.

B. Design Goals

Signet is designed to achieve the following goals: **(G1: High Throughput)** supporting gigabit-scale workloads. **(G2: Low Latency)** introducing negligible latency (on the order of microseconds) to the critical path of message delivery. **(G3: High Scalability)** scaling to the number of participants and messages, such as the witness handling on the order of $\approx 100M$ packets per second. **(G4: Compatibility)** aligning with the trust assumptions of current decentralized systems by relying on entities already trusted in the network, and supporting incremental deployment. **(G5: Accountability)** providing publicly verifiable, unforgeable, and non-repudiable

cryptographic proofs, allowing attribution of misbehavior and resolution of disputes.

C. Assumptions and Threat Model

We assume that the blockchain executes smart contracts correctly, and that its clock provides reliable timestamps for enforcing SLAs, as do the routers' clocks. Notifications and proofs traverse the network within a bounded delay Δ, and a proof can be submitted on-chain within a bounded delay. For each notifier-recipient pair, there exists at least one mutually trusted witness AS. We assume a loose time synchronization, as required by TESLA [26]. TESLA-based MACs are unforgeable and, in conjunction with on-chain timestamped anchoring, provide non-repudiation.

We also assume that a notifier or a recipient can act arbitrarily Byzantine. Mutually trusted witnesses are honest but can be faulty; that is, they can fail to provide a notifier with a valid proof of an observed notification. This assumption is strictly weaker than assuming that all ASes are honest: it only requires that the ASes mutually trusted by both the notifier and the recipient behave honestly. Such a witness may still make mistakes, but will not deliberately favor either side or be susceptible to bribery.

We assume that there is at least one non-faulty witness for a given notifier-recipient pair, which generally holds in diversified, global Internet topologies, but is less robust in less connected settings. No adversary can prevent the delivery of a message within the bounded delay of Δ.

D. Security Goals

Signet ensures the following security properties in the presence of arbitrarily Byzantine notifiers and recipients: (I) No Byzantine recipient can claim the deposit of an honest notifier. (II) An honest recipient can always claim the deposit of a misbehaving Byzantine notifier. A misbehaving notifier avoids sending a notification for a subscribed event within the SLA-defined time bounds. (III) A faulty witness is detectable by notifiers. (IV) A false notification (i.e., a notification for a non-existing event) is detectable by recipients.

IV. Signet Design

A. Overview

Figure 1 shows an overview of Signet. In step ❶, Witness ASes register on-chain with the Signet smart contract, and, at each key-release interval, append their current released key. ❷ A notifier creates an on-chain profile that records its SLA, trusted witnesses, and the deposit. ❸ A recipient subscribes to that profile, specifying its address, events of interest, and its trusted subset of the profile witnesses. ❹ When a subscribed event occurs, the notifier emits a notification packet along a path that traverses at least one of the mutually trusted witness ASes, leveraging the path control of PANs. ❺ A router in a witness AS observes the packet, timestamps it, computes a MAC using its current TESLA key, and returns that proof to the notifier. ❻ The notifier aggregates proofs from each AS within a batching window, computes each batch's Merkle root, and submits the roots on-chain before the corresponding keys are disclosed. ❼ If the recipient detects a missed or delayed notification, it can create a challenge to claim the notifier's deposit. ❽ The notifier has a limited time window to dispute by presenting the proof and the Merkle path. ❾ The smart contract verifies the proof against its stored states. The challenge will be either rejected (if the dispute is valid) or the deposit will be claimed and collected by the recipient.

B. Design Details

Figure 2 summarizes the Signet smart contract, listing its state variables and functions. Below, we describe Signet's design in detail.

Step ❶: TESLA Key Registration by Witnesses. ASes that want to serve as witnesses register on-chain with the Signet smart contract. Each witness invokes *create_as_tesla_keys()*, specifying a key rotation *interval*, to create an *AsTeslaKeys* object, including a key list initialized with witness AS's first TESLA symmetric key. Witnesses are uniquely identified on-chain by the *ID* of their *AsTeslaKeys* object. At the end of each key release interval, the witness appends its currently released key to the existing key list, timestamped with the current time. The smart contract only allows the owner of the *AsTeslaKeys* object to add keys.

Step ❷: Notifier Profile Creation and Deposit Funding. A notifier creates a profile through *create_notifier_profile()*, specifying:

- ***ID***: Identifier of the profile object.
- ***fee***: Subscription fee charged to recipients.
- ***address***: Profile owner's (notifier's) blockchain address.
- ***SLA***: Service-level agreement, that is, the maximum notification delay of δ milliseconds after the block containing the subscribed event is published.
- ***timeout***: Time window for the notifier to dispute a recipient's challenge by providing proof of on-time notification.
- ***witnesses***: Set of trusted ASes that will generate verifiable proofs when they observe notifications.
- ***deposits***: Funds staked by the notifier to guarantee timely notification. Recipients may claim these funds if the notifier fails to meet the SLA.
- ***IP***: The IP address of the notifier.

After creating the profile, the notifier funds the *deposits* of its *profile* with a certain *amount* of blockchain coins via *add_deposit()* to back the agreement.

Step ❸: Subscription and Path Setup. A recipient subscribes to a notifier's profile using *subscribe_to_notifier()*, selecting a specific set of events (*event_set*), for which it wants to receive notifications, and a subset of the *witnesses* in the profile, as its mutually trusted witnesses (*mutual_witnesses*) with the notifier. The smart contract binds the recipient's on-chain address and selected events with a *deposit*. The recipient's address is derived from the transaction's sender address, as specified in the transaction context (*ctx*). The recipient also specifies its IP address as the notification destination, and

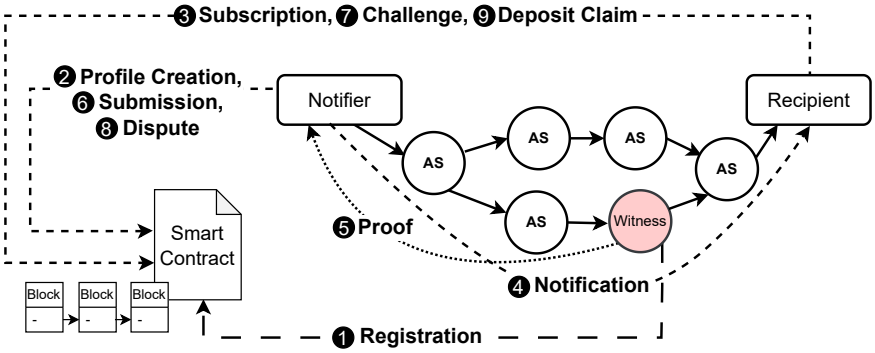

Fig. 1: An overview of Signet, with the arrows showing the sequential main steps.

Smart Contract

State Variables
AsTeslaKeys: ID, interval, owner, keys
Deposit: ID, amount, event_set, recipient
NotifierProfile: ID, fee, address, SLA, timeout, witnesses, deposits, IP
ProofTree: ID, Merkle_root, ts
Challenge: ID, profile, deposit_idx, event, ts

Functions
create_notifier_profile(fee, address, SLA, timeout, witnesses, IP, ctx)
add_deposit(profile, amount, ctx)
subscribe_to_notifier(profile, event_set, mutual_witnesses, coin, ctx)
register_proof(Merkle_root, clock, ctx)
create_challenge(event, profile, deposit_idx, clock, ctx)
dispute_challenge(challenge, clock, profile, as_idx, as_tesla_keys, as_key_idx, proof_tree, mac, Merkle_path, obs_ts)
claim_deposit(challenge, profile, deposit_idx, clock, ctx)

Fig. 2: The Signet smart contract, including the state variables and functions.

each time the address changes, it must append the new IP, analogous to how witnesses add TESLA keys on-chain. Signet then leverages PAN's source-based path selection to choose an end-to-end path that includes at least one of these witnesses.

Step ❹: Notification Transmission. When a subscribed event occurs, the notifier generates a notification packet including headers, a payload containing the notification, and a placeholder for the notification proof (to be filled by the witness). Without loss of generality, we set the payload to the event ID, i.e., the identifier of the corresponding event object. The notifier sends this packet through the selected path.

Step ❺: Proof Generation by Witnesses. Upon observing the notification packet, a router in a designated witness AS records the current time as the observation timestamp. It then concatenates the source address (*n_adr*), destination address (*r_adr*), the payload of the packet (*notif*) and the timestamp (*ts*). It then computes a MAC over the result, using the current interval's TESLA key: *proof_MAC=MAC(key, n_adr||r_adr||notif||ts)*. The witness AS returns the *proof_MAC* and timestamp to the notifier as a verifiable proof of notification, filling the packet's placeholder reserved for the proof. Witness ASes send these proofs via the network-layer source authentication extensions.

Step ❻: Proof Collection and on-chain submission. Upon receiving a proof, the notifier uses the source authentication extension to verify source authenticity and confirm it originated from a valid witness AS, preventing adversarial forgeries. This step is necessary to avoid submitting fabricated proofs from adversaries. Rather than submitting each proof immediately, the notifier aggregates received proofs from a certain AS over a batching window that closes before the current TESLA key is disclosed. At the end of the batching window, the notifier constructs a Merkle tree per witness AS with the collected proofs as leaves. It submits only the root hash of the Merkle tree to the smart contract using *register_proof()* before the key release. The root is recorded on-chain as a *ProofTree* object, including the Merkle root and a timestamp representing the blockchain clock at submission time. Once the corresponding TESLA key to a time interval is later released on-chain by its owner AS, the notifier can use that key to verify the validity of the proof and detect a faulty witness. As long as one of the selected witnesses functions correctly, the notifier can obtain a valid proof.

If the notifier does not receive any proof following a notification transmission, it can retransmit the notification, preferably through alternative paths that traverse one or more trusted witness ASes. This requires δ to be at least multiple RTTs to the witness, allowing for multiple retransmissions if required. This retransmission is independent of the underlying transport's retransmission mechanism, which persists until an acknowledgment is received or the connection is dropped.

Step ❼: Challenge Creation. Since the recipient does not receive a missed notification, it can only detect such a failure retroactively. To detect such occurrences, the recipient can occasionally query a trusted source, such as a blockchain indexer, to verify whether it has received all subscribed notifications on time. If a discrepancy is found, indicating a missed or delayed notification, the recipient may initiate a challenge to claim the notifier's deposit. To do so, the recipient submits a *Challenge* to the smart contract using *create_challenge()*. The *challenge* references the on-chain event object (*event*) for which the notification was not received on time. It also indicates the challenged notifier *profile*, and the index of the corresponding deposit (*deposit_idx*). The smart contract ensures that the

referenced on-chain event and notifier profile are valid and that the subscribed event set covers the referenced event. If both conditions are met, it timestamps the challenge with the current clock time and records it on-chain.

Step ❽: Challenge Dispute. Once a challenge is created, the notifier has a limited time (the *timeout* attribute in the notifier profile) to dispute it. To dispute, the notifier submits the corresponding *proof* along with the Merkle proof (*Merkle_path*). The notifier also includes the index of this witness AS among the trusted witnesses (*as_idx*), a reference to the AS's AsTeslaKeys object (*as_tesla_keys*), the corresponding TESLA key interval index (*as_key_idx*), a reference to the submitted Merkle root with its timestamp (*proof_tree*), and the observation timestamp (*obs_ts*).

Step ❾: Verification and Deposit Claim. According to the challenge and the dispute, the smart contract verifies:

- **Trusted Witness:** The witness is valid; that is, the AS alleged to have generated the MAC was declared in the notifier profile.
  The smart contract checks whether the owner of the *as_tesla_keys* object is the same as the AS determined by the *as_idx* among the *witnesses* in the *profile*.
- **Merkle root:** The *proof* is correctly included as a leaf under the committed Merkle root. The *proof* along with the *Merkle_path* must match the Merkle root in the *proof_tree* object.
- **MAC Correctness:** The MAC in the *proof* is correct. The smart contract fetches the TESLA key, at the interval index *as_key_idx* from *as_tesla_keys*, and the notification message payload from the referenced event object. It also fetches the notifier and recipient IP addresses from the profile. Then, verifies that the MAC is generated by the correct key and for the correct message with the correct observation timestamp (*obs_ts*).
- **Key freshness:** The Merkle root was committed on-chain before the key release: $proof\_tree\_ts \leq key\_release\_time$, with *key_release_time* derivable from *as_tesla_keys*.
- **On-time notification:** The proof was submitted on time. The observation timestamp (*obs_ts*), included by the witness in the proof, does not exceed the timestamp of the event object plus the agreed-upon delay bound δ as SLA in the notifier profile: $obs\_ts + \delta \leq proof\_ts$.
- **Challenge not expired:** $challenge\_ts + timeout < dispute\_ts$.

If the notifier's dispute is accepted, the smart contract drops the challenge. Note that recipients are not penalized for raising incorrect challenges, as failures may arise from factors beyond their control (e.g., last-mile issues, congestion). If a challenge is not resolved before its expiration time (*timeout*), the corresponding deposit becomes claimable. When the recipient claims the deposit, the smart contract verifies that the *timeout* has elapsed and subsequently transfers the deposit to the recipient's balance. Both the deposit and challenge objects are then removed from the blockchain. While our design applies the same deposit penalty for both missed and delayed notifications, more complex SLAs are possible; for example, the penalty for a delayed notification could be a fraction of the deposit, proportional to the delay duration.

C. Design Goal Fulfillment

We revisit the design goals listed in §III-B and outline how Signet satisfies each of them:

High Throughput. The symmetric-key cryptography (MAC) used in TESLA offers significantly lower computational overhead than public-key signatures. As a result, proof generation by ASes only requires hashing, timestamping, and MAC computation, which are lightweight and can be executed at line rate on modern routers. On the notifier's side, aggregating proofs into Merkle trees and committing a single root hash once per TESLA interval keeps the on-chain submission overhead constant regardless of notification volume. This ensures that on-chain proof submission does not limit overall throughput. Recipients perform no cryptographic work when receiving notifications, preserving system-wide performance. They only store notifications until cross-checking with an indexer for discrepancies against on-chain events, after which they can be discarded. This adds no overhead to the critical path of proof delivery.

Low Latency. By embedding witnesses directly on the selected AS-level path, Signet eliminates additional round-trips for proof exchange, ensuring minimal delay added to the critical message path. At the witness, although each packet undergoes hashing, timestamping, and MAC computation before forwarding (no additional storage for routers), these operations add only microseconds of processing delay, making the end-to-end notification latency effectively indistinguishable from that of standard packet forwarding.

High Scalability. In Signet, witnesses maintain no per-recipient state: they simply compute TESLA MACs at line rate, allowing them to support an increasing number of recipients. Moreover, because recipients and ASes are geographically and topologically diverse, proof-generation load naturally distributes across different witnesses. Each notifier constructs the Merkle tree of proofs per witness off-chain, ensuring that the on-chain proof submission overhead stays constant regardless of the number of recipients and notifications. The smart contract, however, needs to manage more subscriptions and potential dispute claims as the system grows; thus, Signet's overall scalability also depends on the underlying blockchain's scalability.

Accountability. Signet notification proofs are protected by cryptographically secure MACs derived from one-way functions, ensuring unforgeability and integrity. Time-stamped key disclosures anchored on-chain provide non-repudiation, enabling reliable dispute resolution.

Compatibility. The system builds on existing trust in ASes for reliable packet forwarding, and extends this trust to notification proofs. Meanwhile, notifiers can independently verify proofs upon key disclosure, enabling exclusion of faulty witness

ASes. By supporting multiple witnesses, Signet decentralizes trust further and increases fault tolerance. Furthermore, Signet is incrementally deployable: even with only a few Signet-enabled witness ASes, any notifier-recipient pair with at least one path that passes through one or more of these ASes can use Signet. Moreover, witnesses are incentivized to participate as providing this service can make witness ASes more attractive to their customer ASes, offering a value-added feature.

D. Security Analysis

We show that Signet ensures the security properties described in §III-D.

Property I: No Byzantine recipient can claim the deposit of an honest notifier.

A Byzantine recipient R attempting to obtain a notifier's deposit may try to either submit a false challenge or frame the notifier to commit an incorrect proof.

1) Let R attempt to submit a false challenge claiming that a notification for event e was not transmitted on time by notifier N. Two cases arise:
   a) Event e does not exist on-chain: the smart contract rejects challenges for non-existent events on-chain.
   b) Event e exists on-chain: An honest notifier obtains a valid proof P for sending a notification for event e, given that there is at least one non-faulty witness on the notification path from the notifier to R. Proof P is timestamped by the witness and bound to TESLA keys via cryptographic MACs, ensuring that any challenge can be correctly disputed by the notifier.
2) Let R attempt to impersonate a witness by sending the notifier a fabricated proof to get the notifier to commit this bogus proof on-chain. The use of source authentication in Signet ensures that only untampered proofs from legitimate witnesses are accepted by the notifier. This prevents spoofing, particularly during the vulnerable window before the current TESLA key is disclosed, when TESLA-based authentication is not yet possible.

Property II: An honest recipient can always claim the deposit of a misbehaving Byzantine notifier.

Let a notifier N be a misbehaving Byzantine notifier, avoiding a timely notification of a subscribed event by:

1) *Omitting the notification*: Since N can not obtain a valid proof from witnesses without sending a notification and due to the unforgeability and non-repudiation of TESLA MACs in Signet, N cannot dispute the corresponding challenge with a valid proof and loses the claimed deposit.
2) *Delaying the notification*: Each proof P includes an observation timestamp *obs_ts* and a *MAC* tied to the TESLA key of the witness for the corresponding interval. Non-repudiation guarantees that *obs_ts* cannot be modified by N, ensuring that the smart contract can correctly enforce the SLA-defined time bound and charge N with the deposit.

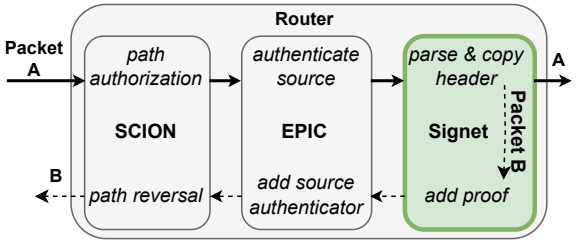

Fig. 3: Architecture of the SCION/EPIC router, highlighting the integrated Signet module.

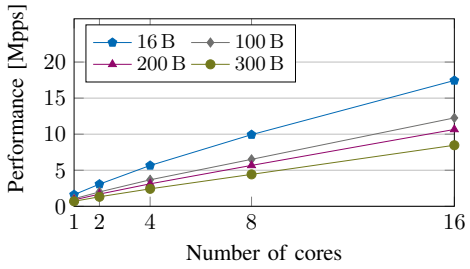

Fig. 4: Forwarding performance of the Signet router vs. number of cores.

Property III: A faulty witness is detectable by notifiers. A faulty witness may fail to provide a notifier with a valid proof for an observed notification. Once the corresponding TESLA key is disclosed, the notifier can verify all proofs of that interval and detect the faulty witness. The notifier can avoid relying on witnesses that have produced missing or invalid proofs in the future.

Property IV: A false notification is detectable by recipients. Sending a notification for a non-existing event is not directly penalized in Signet, but recipients can detect it by querying for the blockchain inclusion proof of the event from an indexer and avoid using notifier N in the future. Note that a double notification is harmless for recipients, since recipients are not charged per notification.

V. Implementation and Evaluation

A. Data Plane Evaluation

We implement and evaluate a proof-of-concept version of the Signet router (the most performance-critical component of our architecture) using Intel's Data Plane Development Kit (DPDK) [27]. Our goal is to demonstrate that high forwarding performance can be achieved despite the additional overhead introduced by Signet's MAC computations, packet cloning, and related operations. We integrate Signet as a module within the SCION/EPIC router (Figure 3). Our evaluation focuses specifically on the Signet module, as the performance of the underlying SCION/EPIC router has been thoroughly studied in prior work [25], [28]. Since extending a packet with additional header fields at the router can be expensive, we avoid this overhead by having the sender reserve space for the notification proof and the timestamp in the original packet. This design enables the router to insert the proof without shifting (i.e., copying) packet data or increasing the packet's length. Preserving the original packet length is important,

as an increase could cause the packet to exceed the MTU and be dropped. The notification packet thus consists of a SCION header, an EPIC header, placeholders for the proof of notification and the timestamp, and the full notification. The proof is generated using HMAC-SHA2.

Measurement Setup. Our testbed includes two machines: a commodity system equipped with an Intel Xeon 2.1 GHz CPU, which runs our implementation of the Signet module , and a Spirent SPT-N4U device, used both for traffic generation and throughput measurement. The two machines are interconnected via a bidirectional 40 Gbps link.

Results. Figure 4 shows the processing performance of the Signet module in mega-packets per second (Mpps). A single core forwards about 1.6 Mpps for 16 B notifications and 0.7 Mpps for 300 B notifications. Throughput scales linearly with additional cores: with 16 cores, forwarding reaches 8.5–17.4 Mpps across the same notification sizes. The high rate stems from low per-packet processing overhead, roughly 0.6 µs for 16 B and 1.4 µs for 300 B. If even higher notification rates are needed, additional CPU cores can be dedicated to the module to sustain proportionally greater throughput.

B. Control Plane Evaluation

We implement the smart contract described in Figure 2 (§III-B) using Sui Move [29]. Signet is designed to operate on high-performance blockchains; we select the Sui blockchain [30] as a representative platform because it offers low-latency transaction processing and native support for Move smart contracts.

Our implementation consists of approximately 500 lines of Move code (including tests). To ensure reproducibility, we publicly release the complete source code[2]. We deploy the smart contract on the Sui testnet to measure gas consumption. At the time of writing (August 2025)[3], each contract function call costs less than 7,000,000 MIST (0.007 SUI), equivalent to under 0.03 USD (August 2025). Most of this cost stems from object creation, a portion of which can be refunded upon object deletion (e.g., when a notifier successfully disputes a challenge or a recipient redeems a notifier's deposit). These results indicate that both initiating and resolving disputes incur negligible costs. Signet smart contract enables high scalability, as the operations are parallelized across notifiers, recipients, and witnesses. The regular operations, such as proof (Merkle root) submission and key updates, are lightweight, and other operations, such as challenge or dispute, are infrequent.

VI. Related Work

Signet enables proof of notification by combining insights from distributed witnessing systems and micropayment-based approaches. As outlined in §III-B, Signet is designed to achieve the following goals: (G1) support gigabit-scale workloads, (G2) introduce negligible latency to the critical path of message delivery, (G3) scale with the number of participants and messages, and (G4) align with the trust assumptions of current decentralized systems, in addition to providing (G5) accountability.

Distributed witnessing. Prior work [13], [31], [32] has explored distance-bounding techniques for geolocation, typically involving a verifier sending a challenge and measuring the response time from a prover. Although mainly applied to geolocating entities [31]–[34], some efforts focus on proving file location [13], [35], [36], which can be adapted to prove that data reached a given location, making them relevant to proof of notification. Notably, Benson et al. [35] and Watson et al. [36] propose systems that enable a client to verify that a file resides at a specific location. These designs scale well and can achieve high throughput by avoiding expensive cryptographic operations on the critical path; however, they require witnesses to persist data on that path to answer future recipient challenges. While that is acceptable in file-sharing (where witnesses store files), it is unsuitable for providing proofs of notification for ephemeral messages. Requiring disk access or forwarding all messages to third parties introduces significant latency, violating our low-latency goal **G2**. Furthermore, their proofs are non-transferable—only the client can verify the data location—so they cannot support decentralized, penalty-enforcing environments, violating **G4**.

GoAT [13], an extension of Watson et al. [36] and a key source of inspiration for Signet, addresses this challenge by using external witnesses as timestamping servers. However, it still requires witnesses to mediate all messages and apply public-key cryptography on the critical path. While acceptable for GoAT's file-transfer use case, this overhead is a bottleneck for Signet, violating **G1** and **G2**.

Micropayments. Micropayments have been extensively studied in blockchain systems, primarily to enable fast, low-cost transactions. In the context of Signet, they can be applied through a design where the notifier delivers notifications incrementally, and the recipient pays a micro-fee for each received piece [8]. The aggregate of these micro-fees replaces the subscription fee as described in §III. This approach inverts the primary design decision of Signet: instead of enabling the notifier to obtain a verifiable proof and allowing the recipient to raise complaints, it ensures that the recipient only pays for received notifications. This inversion introduces new design challenges,where an honest notifier may not be compensated for notifications sent to offline recipients, and a malicious notifier could withold critical messages without risking a deposit. Nevertheless, a key strength of this approach is its compatibility with decentralized environments.

A wide range of payment channel designs exist [9], [37]. Traditional payment channels, such as the Lightning Network [38], Sprites [39], CoinExpress [40], and Rapido [15], allow users to open a channel and conduct multiple off-chain transactions, settling only the final balance on-chain. These

[2]https://github.com/asonnino/signet-contract (commit `40eb556`)

[3]The reference gas price in Sui is determined through a voting mechanism that resets at the beginning of each epoch (approximately every 24 hours), and may vary slightly between epochs.

systems avoid the need for witnesses but require recipients to lock deposits on-chain and perform public-key cryptography on the critical path for every message. While feasible for sizable payments, this overhead creates a bottleneck, conflicting with goals **G1** and **G2**, and poses scalability challenges in the high-throughput settings targeted by Signet, where cryptographic operations on the notifier can limit performance, violating **G3**.

Designing payment channels with trusted hardware [41], [42] could alleviate the need for public-key operations and thus achieve scalability. However, they come with the additional trust assumptions of relying on trusted hardware. It is also unclear whether requiring a payment for each message and sustaining gigabit-scale workloads would not saturate the trusted hardware and thus still fail goals **G1** and **G2**.

Asynchronous payment systems like FastPay [43], Astro [44], and Brick [45] use BFT committees for fast, low-cost, and horizontally scalable payments, eliminating per-message public-key operations and easing throughput concerns. However, payments still require hundreds of milliseconds, so implementing one payment per message would substantially increase latency, violating the low-latency goal **G2**.

VII. Discussions

Witness Distribution and Availability. With only a few Signet-enabled witness ASes, any notifier-recipient pair with at least one path through one or more of these ASes can use Signet. However, a limited number of witnesses may lead to long detours, added latency, and lower availability. As more customers are attracted and business increases for these ASes, other ASes will be motivated to join, in line with Signet's incremental deployability. As adoption grows and the system reaches equilibrium, detours decrease, and coverage broadens. With multiple trusted witnesses for each notifier-recipient pair, robustness against faulty witnesses increases, and both notifiers and recipients can choose from a larger set of witnesses, with competition enhancing reliability.

Privacy Considerations. A limitation of Signet is that recipients' interests in specific blockchain events, as well as their trusted AS witnesses, are committed on-chain when creating subscriptions. While necessary for verifiable and dispute-resolvable notifications, this public record can potentially reveal sensitive user preferences or trust relations and pose privacy concerns, especially for high-value or strategic users.

A possible mitigation is to retain public verifiability while hiding subscription details behind commitments and zero-knowledge proofs. An end host commits to a subscription filter (defining its *event_set*) by posting a commitment C on-chain and privately sharing its opening with the notifier, which verifies the commitment. In a challenge, the recipient provides a zero-knowledge proof of knowledge of an opening of C such that the disputed event is in the corresponding *event_set*, enabling the contract to verify coverage without learning the filter. Implementing such a privacy-preserving subscription method is a promising area for future enhancements of Signet.

Fault Detectability. In the case of *witness* faultiness, Signet provides detectability instead of verifiability: if a witness AS fails to provide a proof or issues a wrong proof, it is not penalized on-chain, but its misbehavior is detectable, and clients can avoid trusting it in the future. This approach provides a lightweight but strong incentive for witnesses to behave honestly, supporting the practicality of the system, and aligning with the realities of Internet-scale deployment.

Compromised ASes. Signet's security model is based on the practical difficulty of attackers compromising network witnesses (ASes), assuming witnesses are chosen judiciously (i.e., large, well-placed ASes). For a malicious party to subvert proof generation, they would need to control or compromise such an AS, an operation that is both highly resource-intensive and detectable.

Byzantine Witnesses. Our threat model assumes faulty, but not Byzantine, witnesses, i.e., an AS may fail to return a proof for a notification it observes, but not act maliciously. Such failures are detectable, as explained in §IV-D.

A truly Byzantine witness, however, could also return bogus proofs (e.g., colluding with recipients to claim notifiers' deposits) or issue proofs for notifications it never observed (e.g., colluding with notifiers to conceal missed notifications). In the case of a bogus proof, the notifier will discover the deception once the corresponding TESLA key is revealed and can confirm that the authenticated proof was invalid. Despite being too late to dispute on-chain, the notifier can choose to stop trusting that AS in future interactions.

Similarly, if a witness consistently issues proofs for notifications it did not observe, the recipient may notice a pattern of frequent failed challenges for the missed notifications while involving the same witness. Over time, such trends can signal misbehavior, allowing recipients to identify and avoid untrustworthy witnesses.

VIII. Conclusion

This paper presents Signet, a novel protocol for verifiable proof of notification that leverages transit ASes as natural witnesses. By integrating the lightweight TESLA protocol with the role of ASes as witnesses, Signet enables efficient proof generation by ASes, supports asynchronous on-chain verification of these proofs, and aligns with the trust assumptions and requirements of modern blockchain systems. Signet achieves non-repudiation for TESLA-based authentication through on-chain proof anchoring and leverages this property together with reliable AS and blockchain timestamping to resist Byzantine adversaries. The evaluation of our prototype demonstrates that Signet achieves high performance in realistic settings: a single core can process more than one million packets per second, while smart contract operations remain cost-efficient.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback and constructive comments, which helped to improve the quality and clarity of this paper.

REFERENCES

[1] S. Bano *et al.*, "Sok: Consensus in the age of blockchains," in *Proc. ACM AFT*, 2019, pp. 183–198.

[2] Z. He, J. Li, and Z. Wu, "Don't trust, verify: The case of slashing from a popular ethereum explorer," in *Proc. ACM Web Conference*, 2023, pp. 1078–1084.

[3] E. Jaghjough Lamrani, "The staking mechanisms in ethereum," 2024.

[4] Z. He and J. Li, "Contract enforcement and decentralized consensus: The case of slashing," *Available at SSRN 4036000*, 2022.

[5] J. F. Doerr *et al.*, "Defi risks and the decentralisation illusion," *BIS Quarterly Review*, vol. 21, 2021.

[6] S. K. Ezzat, Y. N. Saleh, and A. A. Abdel-Hamid, "Blockchain oracles: State-of-the-art and research directions," *IEEE Access*, vol. 10, pp. 67 551–67 572, 2022.

[7] K. Qin, L. Zhou, Y. Afonin, L. Lazzaretti, and A. Gervais, "Cefi vs. defi–comparing centralized to decentralized finance," *arXiv preprint arXiv:2106.08157*, 2021.

[8] M. Al-Bassam, A. Sonnino, M. Król, and I. Psaras, "Airtnt: Fair exchange payment for outsourced secure enclave computations," *arXiv preprint arXiv:1805.06411*, 2018.

[9] K. Kolachala, M. Ababneh, and R. Vishwanathan, "Sok: Payment channel networks," in *BuildSEC*. IEEE, 2024, pp. 28–35.

[10] K. Babel *et al.*, "Mysticeti: Reaching the limits of latency with uncertified dags," *arXiv preprint arXiv:2310.14821*, 2023.

[11] R. Overko, "A study on shared objects in Sui smart contracts," in *IEEE ICBC*, 2024, pp. 1–7.

[12] G. Danezis *et al.*, "Walrus: An efficient decentralized storage network," *arXiv preprint arXiv:2505.05370*, 2025.

[13] D. Maram, M. Kelkar, I. Bentov, and A. Juels, "GoAT: File Geolocation via Anchor Timestamping," in *Proc. FC*. Springer, 2024, pp. 35–72.

[14] N. Papadis and L. Tassiulas, "Blockchain-based payment channel networks: Challenges and recent advances," *IEEE Access*, vol. 8, pp. 227 596–227 609, 2020.

[15] C. Lin, N. Ma, X. Wang, and J. Chen, "Rapido: Scaling blockchain with multi-path payment channels," *Neurocomputing*, vol. 406, pp. 322–332, 2020.

[16] B. Trammell, J.-P. Smith, and A. Perrig, "Adding path awareness to the internet architecture," *IEEE Internet Computing*, vol. 22, no. 2, pp. 96–102, 2018.

[17] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proc. IEEE S&P*, 2000, pp. 56–73.

[18] Z. Cataltepe and P. Moghe, "Characterizing nature and location of congestion on the public internet," in *IEEE ISCC*, 2003, pp. 741–746.

[19] B. Trammell, "Current open questions in path-aware networking," *IRTF, RFC 9217*, 2022.

[20] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, control, and isolation on next-generation networks," in *IEEE S&P*, 2011, pp. 212–227.

[21] C. Krähenbühl *et al.*, "Deployment and scalability of an inter-domain multi-path routing infrastructure," in *Proc. ACM CoNEXT*, 2021, pp. 126–140.

[22] SCION Project, "SCION Internet Architecture," 2025. [Online]. Available: https://github.com/scionproto/scion

[23] F. Wirz *et al.*, "Scaling SCIERA: A journey through the deployment of a next-generation network," in *Proc. ACM SIGCOMM*, 2025, p. 720–741.

[24] B. Rothenberger, D. Roos, M. Legner, and A. Perrig, "Piskes: Pragmatic internet-scale key-establishment system," in *Poc. ACM ASIACCS*, 2020, pp. 73–86.

[25] M. Legner, T. Klenze, M. Wyss, C. Sprenger, and A. Perrig, "{EPIC}: every packet is checked in the data plane of a {Path-Aware} internet," in *USENIX Security Symposium*, 2020, pp. 541–558.

[26] A. Perrig, R. Canetti, J. Tygar, and D. Song, "The TESLA broadcast authentication protocol," *RSA CryptoBytes*, vol. 5, no. Summer, 2002.

[27] DPDK Project, "Data Plane Development Kit," https://dpdk.org, 2024.

[28] C. Krähenbühl, M. Wyss, D. Basin, V. Lenders, A. Perrig, and M. Strohmeier, "FABRID: flexible attestation-based routing for inter-domain networks," in *USENIX Security*, 2023.

[29] S. Foundation, "Sui move," https://docs.sui.io/concepts, 2025.

[30] ——, "Sui: Build beyond," https://sui.io, 2025.

[31] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, "Towards IP geolocation using delay and topology measurements," in *Proc. ACM IMC*, 2006, pp. 71–84.

[32] Y. Wang, D. Burgener, M. Flores, A. Kuzmanovic, and C. Huang, "Towards {Street-Level}{Client-Independent}{IP} geolocation," in *USENIX NSDI*, 2011.

[33] K. Kohls and C. Diaz, "{VerLoc}: verifiable localization in decentralized systems," in *USENIX Security)*, 2022, pp. 2637–2654.

[34] K. E. Jeon, J. She, P. Soonsawad, and P. C. Ng, "BLE beacons for internet of things applications: Survey, challenges, and opportunities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 811–828, 2018.

[35] K. Benson, R. Dowsley, and H. Shacham, "Do you know where your cloud files are?" in *Proc. ACM CCSW*, 2011, pp. 73–82.

[36] G. J. Watson, R. Safavi-Naini, M. Alimomeni, M. E. Locasto, and S. Narayan, "Lost: location based storage," in *Proceedings of ACM Workshop on Cloud computing security workshop*, 2012, pp. 59–70.

[37] A. Gangwal, H. R. Gangavalli, and A. Thirupathi, "A survey of layer-two blockchain protocols," *Journal of Network and Computer Applications*, vol. 209, p. 103539, 2023.

[38] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[39] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Proc. FC*. Springer, 2019, pp. 508–526.

[40] R. Yu, G. Xue, V. T. Kilari, D. Yang, and J. Tang, "Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks," in *IEEE ICCCN*, 2018, pp. 1–9.

[41] J. Lind, O. Naor, I. Eyal, F. Kelbert, E. G. Sirer, and P. Pietzuch, "Teechain: a secure payment network with asynchronous blockchain access," in *Proc. ACM SOSP*, 2019, pp. 63–79.

[42] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proc. ACM SIGSAC CCS*, 2019, pp. 1521–1538.

[43] M. Baudet, G. Danezis, and A. Sonnino, "Fastpay: High-performance byzantine fault tolerant settlement," in *ACM AFT*, 2020, pp. 163–177.

[44] D. Collins *et al.*, "Online payments by merely broadcasting messages," in *IEEE/IFIP DSN*, 2020, pp. 26–38.

[45] G. Avarikioti, E. K. Kogias, R. Wattenhofer, and D. Zindros, "Brick: Asynchronous payment channels," *arXiv preprint 1905.11360*, 2019.

APPENDIX A
TESLA BROADCAST AUTHENTICATION

TESLA divides time into fixed-length intervals (i), each associated with a secret key K_i known only to the sender, constructed with a one-way key chain:

$$K_n \xrightarrow{F} K_{n-1} \xrightarrow{F} \ldots \xrightarrow{F} K_1 \xrightarrow{F} K_0$$

where K_n is randomly generated, and F is a secure pseudo-random function (PRF) that satisfies weak collision resistance. For more security, a derived key $K'_i = F'(K_i)$, where F' is another PRF, is used to compute MACs during time interval i, but K_i is only disclosed after a delay of d intervals.

For a message m sent during interval i, the sender includes the $\text{MAC}_{K'_i}(m)$. The receiver buffers $(m, \text{MAC}_{K_i}(m))$ until K_i is disclosed. At this point, the receiver can calculate the previous key K'_{i-1} as:

$$K'_{i-1} = F'(F(K_i))$$

and verify the authenticity of packets of interval $i-1$. Additionally, it can recover any previous key:

$$K'_v = F'(F^{i-v}(K_i)),$$

where F^{i-v} denotes $i-v$ applications of F.

By relying on symmetric key cryptography and scheduled key release, TESLA achieves lightweight authentication suitable for high-throughput scenarios.