

PASS: A Provenanced Access Subaccount System for Blockchain Wallets

Jay Yu^{1,2}, Shunfan Zhou³, Hang Yin³, and Brian Seong^{4,5}

¹ Stanford University jyu01@stanford.edu

² Pantera Capital jay@panteracapital.com

³ Phala Network {shelvenzhou,hangyin}@phala.network

⁴ Polygon Labs bseong@polygon.technology

⁵ Chainless Company brian@chainless.company

Abstract. Blockchain wallets currently follow a single-entity ownership model: possession of a private key grants unilateral control. This assumption is brittle for emerging settings such as organizational custody, shared governance, and AI agent wallets, where multiple actors must coordinate without exposing secrets or leaking internal activity.

We present PASS, a **Provenanced Access Subaccount System** that replaces role or identity-based control with provenance-based control: assets can only be used by subaccounts that can trace custody back to a valid deposit. A simple Inbox–Outbox mechanism ensures all external actions have verifiable lineage, while internal transfers remain private and indistinguishable from ordinary EOAs.

We formalize PASS in Lean 4 and prove core invariants, including privacy of internal transfers, asset accessibility, and provenance integrity. We implement a prototype with enclave backends on AWS Nitro Enclaves and dstack Intel TDX, integrate with WalletConnect, and benchmark throughput across wallet operations. These results show that provenance-based wallets are both implementable and efficient. PASS bridges today’s gap between strict self-custody and flexible shared access, advancing the design space for practical, privacy-preserving custody.

1 Introduction

Cryptocurrency wallets conventionally assume one key, one owner. In this single-entity ownership (SEAO) model, whoever holds the private key gains permanent and total control of the address, and casts publicly readable, gas-consuming blockchain transactions. However, this model struggles to adapt to emerging blockchain scenarios:

1. **AI and Smart Agent Transactions.** AI agents increasingly manage crypto assets [26,38] and require bounded authority and access guardrails for users’ asset security.
2. **Enterprise Payroll Privacy.** Organizations need to distribute salaries and departmental budgets to employee accounts without revealing sensitive details on-chain [27,15,4].

3. **Consumer-grade Central Limit Order Books.** On-chain order books face high gas costs and latency that prevent real-time trading at scale [31,23], and many consumers prefer not to manage their own keys [9,40].

All three use cases demand support for **internal transfers** that are fast, low-cost (ideally free), and private, with most value movement occurring internally and onchain transactions serving only for settlement. Purely on-chain or custodial solutions cannot meet these requirements: on-chain transfers are slow, expensive, and public, while custodial models introduce trust assumptions and compromise privacy through full custodian visibility. The core challenge involves **private shared state**—asset balances, transfer histories, and user subaccounts that must remain confidential while being collectively maintained. This requires TEEs for confidential computation with auditability, as well as designing appropriate access control models for this new computational paradigm.

In response to these needs, organizations and wallet providers have attempted to retrofit traditional access control paradigms—such as Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC)—to govern internal transfers and shared wallet usage [42,14]. This typically involves layering custom policy engines or custodial wrappers atop the standard single-owner wallet, specifying who may initiate or approve transfers under various conditions [16,36]. However, traditional access-control frameworks such as RBAC and ABAC assume an *endogenous* setting: users create and manage their own files, objects, or services, and policies are attached to these identities or roles. Blockchains are fundamentally *exogenous*: assets are created outside the wallet by smart contracts and transferred permissionlessly in. In this setting, identity or role-based rules are awkward to express and brittle to enforce. What matters instead is provenance: who deposited an asset, and how it flowed within the wallet.

In this paper, we propose PASS, a Provenanced Access Subaccount System that enforces control directly by provenance. The core mechanism is simple: an **Inbox** records external deposits, an **Outbox** signs and broadcasts withdrawals. By leveraging Trust Execution Environment (TEE), all internal movements remain private and fee-less. Every unit of value inside the wallet has a verifiable lineage to an on-chain deposit, while externally a PASS wallet is indistinguishable from a standard EOA. Our contributions include:

- A provenance-based access model aligned with blockchain’s exogenous asset creation, enforced through an Inbox–Outbox design.
- A formal model in Lean 4 proving privacy, accessibility, and provenance integrity.
- A prototype with multi-vendor enclave backends (AWS Nitro, Intel TDX via dstack [41]), a web app frontend integrated with WalletConnect, together with throughput benchmarks.
- Application scenarios, including organizational custody, privacy-preserving transfers, and safe delegation to AI agents.

PASS thus bridges the gap between strict self-custody and flexible shared access, combining strong privacy guarantees with practical deployability.

2 Background and Related Work

Model	Control basis	On-chain overhead	Internal Privacy	Compatibility
EOA	Single private key	None	None	Native EOA
Multisig	t -of- n co-signers	High; quorum per action	None; logic public	Smart contract based
ERC-4337 Smart Accounts	On-chain policy logic	Medium to high; custom logic and gas	None; logic public	Requires ERC-4337 chain support
Custodians	Custom RBAC or ABAC languages	None; Off-chain enforcement	Custodian sees all activity	API mediated
Liquefaction	TEE key encumbrance; policy trees	Medium; Oasis contracts and gas	Strong; TEE hides subpolicies	Requires Oasis deployment
PASS (ours)	Provenance via Inbox-Outbox	None; EOA compatible	Strong; internal transfers invisible	Native EOA plus off the shelf TEEs

Fig. 1. Comparison of wallet architectures. PASS features provenance-based control, off-chain privacy, and native EOA compatibility.

Growing retail and institutional adoption of digital assets [17] has driven diverse wallet designs [21]. We summarize mainstream Ethereum Virtual Machine (EVM) wallet designs in Figure 1.

EOAs and Multisigs. Externally Owned Accounts (EOAs) give unilateral, permanent control to whoever holds the private key [10]. Multisignature wallets (eg., Safe) distribute trust via t -of- n approvals [28], common for treasuries and bridges [34,39,8]. Drawbacks include quorum overhead, symmetric authority, and public logic with attackable frontends [8]. In contrast, PASS offers fine-grained, scoped permissioning without new contracts.

Smart Contract Wallets and Account Abstraction. Wallets like Argent and Loopring add programmability (social recovery, spend limits) via on-chain logic [2,20,25]. ERC-4337 enables custom validation through `UserOperation` contracts [13], and EIP-7702 lets EOAs invoke this logic [30]. These increase flexibility but expose policies publicly and incur gas costs, limiting use cases like payroll. PASS instead enforces policy privately in a TEE, minimizing on-chain footprint.

Custodial Policies. Custody providers (Fireblocks, Turnkey, Privy) use RBAC/ABAC engines with custom policy languages [16,36]. PASS replaces these with provenance: authority follows asset origin and history, akin to a UTXO model. This reduces policy-surface complexity while remaining composable with constraints like whitelists and blacklists.

TEEs and Liquefaction. Trusted Execution Environments (TEEs) provide hardware-isolated contexts with remote attestation [22,29]. Liquefaction [7] keeps wallet keys in a TEE and enforces sub-policies via key encumbrance, ensuring privacy. PASS advances this model by (i) proving invariants in Lean 4, (ii) simplifying enforcement to a provenance rule with Inbox/Outbox subaccounts,

and (iii) working with a standard EOA without relying on Oasis smart contracts [24]. We further validate by implementing PASS on AWS Nitro Enclaves and Intel TDX (via dstack), and showcase applications including AI wallets, payroll, supply chains, and DAO voting.

Other Models. Exchanges pool funds in omnibus addresses with off-chain ledgers [37]; mixers like Tornado Cash hide deposit–withdrawal links [27]. PASS similarly hides internal transfers while retaining a provenance log for auditability; outsiders see only deposits and withdrawals.

3 System Model and Design

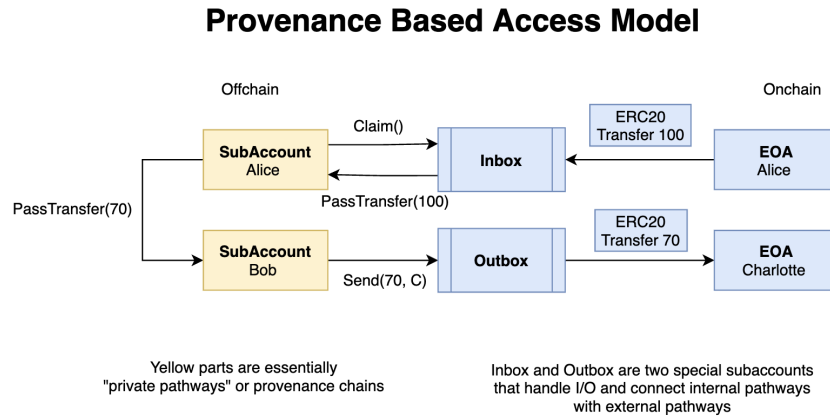


Fig. 2. Overview of PASS. External deposits enter the **Inbox** and are claimed by subaccounts. Internal transfers are private and off-chain. Egress occurs via the **Outbox**, which signs and broadcasts on-chain transactions. All on-chain activity has clear provenance; internal movements remain private.

3.1 High level system model

PASS is a multi-user wallet on a single EOA (sk, pk). The wallet maintains a finite set of subaccounts $\mathcal{U} = \{u_1, \dots, u_n\}$ and a set of assets \mathcal{A} (e.g., ERC-20). A subaccount $u \in \mathcal{U}$ has authority over an amount x of asset $\alpha \in \mathcal{A}$ iff it can obtain a signature from the global sk via the enclave; sk resides only inside a TEE and is never directly accessible. We give the intuitive model here and formalize it in Section 3.2.

Internally, PASS tracks balances with a map $\mathcal{L} : \mathcal{U} \times \mathcal{A} \rightarrow \mathbb{Z}_{\geq 0}$ (accessed via $\mathcal{L}[u][\alpha]$), an **Inbox** multiset \mathcal{I} of unclaimed deposits (α, x, m) , an **Outbox** FIFO queue \mathcal{O} of pending on-chain actions $(\alpha, x, extDst, \nu)$ together with a global nonce $\nu \in \mathbb{N}$, and an append-only provenance log \mathcal{H} . The core principles of PASS, as shown in Figure 2 are as follows:

1. *Access follows provenance.* A unit of value is usable by u only if there exists a custody chain in \mathcal{H} from an external deposit to u . Informally, $\text{Allow}(u, \alpha, x)$ means balance x is reachable for u via provenance in \mathcal{H} . This implies *consistency*: for every α , the total internal balance equals the on-chain balance at pk , i.e., $\sum_{u \in \mathcal{U}} \mathcal{L}[u][\alpha] = B_{\text{on}}(\alpha)$. Incoming deposits $(\alpha, x, m) \in \mathcal{I}$ are “wrapped” and must be explicitly claimed before use by the sender address.
2. *Off-chain privacy.* Internal transfers $u \rightarrow v$ update only \mathcal{L} and \mathcal{H} ; they do not touch \mathcal{O} or the on-chain balance B_{on} , so internal flow is invisible and fee-less.
3. *First-come, first-served.* Outbox transactions are sequenced by a FIFO queue, which maintains and increments a global nonce ν and directly sends enclave-signed transactions to a RPC node to prevent nonce conflicts and replay attacks.⁶

Organizational Ownership Model. PASS operates under an organizational ownership model, where an entity such as a company, individual, or DAO manages shared control over blockchain assets. We detail the threat model in Section 4.

Inbox-Outbox Gateway External senders transfer assets to pk , creating Inbox entries $(\alpha, x, m) \in \mathcal{I}$. A user performs $\text{claim}(u, \alpha, x, m)$ to move value into $\mathcal{L}[u][\alpha]$, conceptually “wrapping” the deposit. To withdraw or interact with a contract, a subaccount first performs an internal transfer to the **Outbox**, which enqueues $(\alpha, x, \text{extDst}, \nu)$; the enclave then signs the concrete transaction with sk and immediately broadcasts it, incrementing ν . The FIFO discipline and global nonce serialize actions and prevent races and double-spends. Because subaccounts never receive pre-signed transactions, pre-signing misuse is mitigated.

Composable policy scheme. Deposits are claims from \mathcal{I} ; internal transfers require $\text{Allow}(u, \alpha, x)$ and update $(\mathcal{L}, \mathcal{H})$; withdrawals are internal transfers to \mathcal{O} followed by submission. Additional constraints (e.g., RBAC or ABAC checks, whitelists and blacklists) can be incorporated as conjuncts to Allow without changing the provenance-centric rule or its consistency and privacy effects.

3.2 Formal Model

We formalize PASS as a transition system in Lean 4⁷, an interactive theorem prover with a rich dependent type system [19], suitable for policy verification (e.g., AWS Cedar [12]). In this section, we introduce the formal system we modeled in lean, and in Section 4 we will discuss security invariants modeled. An extended discussion of our formal verification system can be found in Appendix A.

State. A wallet state is $S = (\text{pk}, \text{sk}, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ where $(\text{pk}, \text{sk}, \nu)$ are the EOA public key, private key (generated and held in the TEE), and global nonce; \mathcal{I} and \mathcal{O} are Inbox and Outbox; \mathcal{L} is the internal balance map $\mathcal{L}[u][\alpha] \in \mathbb{Z}_{\geq 0}$; and \mathcal{H} is the provenance history. For privacy arguments we also use the external view $W_{\text{pub}} = (\text{pk}, O, A)$, where O contains the Outbox queue and nonce and A records total on-chain balances by asset.

⁶ We assume that the TEE is able to run a Helios light client, as is the case in dstack Intel TDX, discussed and implemented in Section 5.2.

⁷ Lean 4 model: <https://github.com/pass-wallet/pass-lean4-proofs>

PASS Wallet Management Functionality $\mathcal{F}_{\text{PASS}}^\Sigma$ for wallet \mathcal{W}

Global State:
 pk : public address (EOA)
 sk : private key (in TEE)
 ν : global nonce for on-chain transactions
 \mathcal{I} (**Inbox**): set of unclaimed deposits (α : asset, x : amount, m : depositId)
 \mathcal{O} (**Outbox**): FIFO queue ($\alpha, x, extDst, \nu$)
 \mathcal{L} (**Asset Ledger**): internal map $\mathcal{L}[u][\alpha] \in \mathbb{Z}_{\geq 0}$, where u is each user's ID in the system
 \mathcal{H} (**Provenance History**): a list of transaction records appended whenever **ClaimInbox**, **InternalTransfer**, or **Withdraw** is invoked

CreatePassWallet (*creator*):
 $\mathcal{I} \leftarrow \emptyset, \mathcal{O} \leftarrow \emptyset, \mathcal{L} \leftarrow \{\}$; $\nu \leftarrow 0$
 $(pk, sk) \leftarrow \text{TEE.KeyGen}()$
 Return new \mathcal{W} with $S = (pk, sk, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ and reference to *creator*

InboxDeposit (α, x, m) from external P :
 $\mathcal{I} \leftarrow \mathcal{I} \cup \{(\alpha, x, m)\}$
 Return **success**

ClaimInbox ($\alpha, x, depositId$) from u :
 If no (α, x, m) with $depositId$ in \mathcal{I} , return \perp
 Remove (α, x, m) from \mathcal{I}
 $\mathcal{L}[u][\alpha] \leftarrow \mathcal{L}[u][\alpha] + x$
 $\mathcal{H} \leftarrow \mathcal{H} \mid (op = \text{claim}, \alpha, x, depositId, u)$
 Return **success**

CheckAllow ($u_{\text{send}}, \alpha, x, \mathcal{H}$):
 If $\mathcal{H}.\text{GetProvenance}(\alpha, u_{\text{send}}) \geq x$,
 return **true**; else return **false**

InternalTransfer ($\alpha, x, u_{\text{send}}, u_{\text{recv}}$):
 If $\mathcal{L}[u_{\text{send}}][\alpha] < x$ or $\neg \text{checkAllow}(u_{\text{send}}, \alpha, x, \mathcal{H})$, return \perp
 $\mathcal{L}[u_{\text{send}}][\alpha] \leftarrow \mathcal{L}[u_{\text{send}}][\alpha] - x$
 $\mathcal{L}[u_{\text{recv}}][\alpha] \leftarrow \mathcal{L}[u_{\text{recv}}][\alpha] + x$
 $\mathcal{H} \leftarrow \mathcal{H} \mid (op = \text{transfer}, \alpha, x, u_{\text{send}}, u_{\text{recv}})$
 Return **success**

Withdraw ($\alpha, x, u_{\text{send}}, extDst$):
 If $\mathcal{L}[u_{\text{send}}][\alpha] < x$ or $\neg \text{checkAllow}(u_{\text{send}}, \alpha, x, \mathcal{H})$, return \perp
 $\mathcal{L}[u_{\text{send}}][\alpha] \leftarrow \mathcal{L}[u_{\text{send}}][\alpha] - x$
 $\mathcal{O} \leftarrow \mathcal{O} \cup \{(\alpha, x, extDst, \nu)\}$
 $\mathcal{H} \leftarrow \mathcal{H} \mid (op = \text{withdraw}, \alpha, x, u_{\text{send}}, extDst)$
 Return **success**

SignGSM (dom, msg, u):
 If u lacks sign-right for (dom, msg) , return \perp
 $\sigma \leftarrow (sk, msg)$
 Return σ

(Outbox Processing) (periodic or on demand):
 If $\mathcal{O} \neq \emptyset$, dequeue $(\alpha, x, extDst, \nu)$
 Build tx τ : transfer x of α to $extDst$, nonce = ν
 $\sigma \leftarrow (sk, \tau)$; broadcast (τ, σ)
 $\nu \leftarrow \nu + 1$
 Return **broadcasted** + txHash

Fig. 3. Formal Model of PASS wallet functionality, including **CreatePassWallet** for initializing a fresh instance. On creation, pk is set to a chosen EOA address ea , the TEE generates a hidden private key sk , and global data structures are initialized. External deposits enter \mathcal{I} (**Inbox**), while subaccount transfers update \mathcal{L} (**Ledger**) off-chain. Withdrawals queue items in \mathcal{O} (**Outbox**), and GSM operations are signed using sk .

Transitions. Operations $op \in T$ update S with type-checked rules: *InboxDeposit* adds an unclaimed (α, x, m) to \mathcal{I} ; *ClaimInbox* moves (α, x, m) from \mathcal{I} to $\mathcal{L}[u][\alpha]$ and appends to \mathcal{H} ; *InternalTransfer* moves value between subaccounts; *Withdraw* debits $\mathcal{L}[u][\alpha]$, enqueues $(\alpha, x, extDst, \nu)$ in \mathcal{O} , and logs; *SignGSM* authorizes domain-scoped signable messages. A representative rule is:

$$(pk, sk, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H}) \xrightarrow{\text{transfer}(\alpha, x, u_s, u_r)} (pk, sk, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}', \mathcal{H}')$$

with

$$\begin{aligned} & \text{if } \mathcal{L}[u_s][\alpha] < x \text{ or } \neg \text{checkAllow}(u_s, \alpha, x, \mathcal{H}) \text{ then } \perp; \\ & \mathcal{L}'[u_s][\alpha] = \mathcal{L}[u_s][\alpha] - x, \quad \mathcal{L}'[u_r][\alpha] = \mathcal{L}[u_r][\alpha] + x; \\ & \mathcal{H}' = \mathcal{H} \mid (op = \mathbf{transfer}, \alpha, x, u_s, u_r). \end{aligned}$$

Here `checkAllow` queries provenance in \mathcal{H} and can be extended with auxiliary constraints such as user whitelists or blacklists, without changing the core rule. Figure 3 provides a formal specification of the methods in a PASS wallet.

Provenance Attestation. The TEE can generate cryptographic attestations over the integrity of \mathcal{H} by signing its digest: $\sigma = \text{Sign}_{\text{sk}}(\text{Hash}(\mathcal{H}))$. This enables external verification of custody chains for compliance auditing and regulatory requirements without revealing internal transfer details. Attestations can also cover other system components like the global nonce ν for freshness proofs or balance states for solvency verification, following EIP-712 structured data standards for interoperability.

3.3 Asset Model and General Signable Messages

PASS provides an authorization scheme over a heterogeneous set of assets \mathcal{A} , including fungible tokens, such as ether and ERC-20 tokens, ERC-721 non-fungible tokens, and “general signable messages” across a variety of DApps for sign-in and identity verification. Each asset $\alpha \in \mathcal{A}$ has a well-defined custody path in the provenance log \mathcal{H} , and a subaccount $u \in \mathcal{U}$ has usable balance x of α iff $\text{Allow}(u, \alpha, x)$ holds.

Token assets. For token assets such as ether, ERC-20, and ERC-721, deposits are captured as Inbox entries $(\alpha, x, m) \in \mathcal{I}$. Claims allocate balance into $\mathcal{L}[u][\alpha]$, internal transfers update \mathcal{L} and \mathcal{H} privately, and withdrawals enqueue $(\alpha, x, \text{extDst}, \nu)$ into \mathcal{O} for enclave signing and on-chain broadcast, ensuring consistency where:

$$\forall \alpha \in \mathcal{A}_{\text{tokens}} : \sum_{u \in \mathcal{U}} \mathcal{L}[u][\alpha] = B_{\text{on}}(\alpha, \text{pk}).$$

In particular, ERC-721 tokens are modeled with unique identifiers, where each token ID is treated as a distinct asset α_{tokenId} with unitary balance ($x \in \{0, 1\}$), ensuring uniqueness and preventing double-ownership under provenance.

General Signable Messages (GSMs). In addition to on-chain assets, wallets often sign arbitrary messages, such as for EIP-4361 [32] logins and off-chain DAO votes [33]. PASS models these as a special class \mathcal{A}_{GSM} of virtual assets. Each GSM asset is scoped to a domain dom , so holding $\alpha = (\text{dom})$ represents authority to sign messages from that domain. A signature operation: $\text{SignGSM}(u, \text{dom}, m) \mapsto \sigma$ is permitted iff $\text{Allow}(u, \text{dom}, 1)$ holds in \mathcal{H} . At key generation, the wallet creator is endowed with *default signing authority* over all domains, represented as an initial allocation of every $\alpha = (\text{dom}) \in \mathcal{A}_{\text{GSM}}$ to the root subaccount. These domain assets can be internally transferred to delegate signing authority. Thus provenance of GSMs begins at the owner and flows through transfers, providing a well defined provenance path in \mathcal{H} as with token assets.

By treating both tokens and GSMS as types of elements of \mathcal{A} , PASS provides a unified provenance rule for subaccount authorization. A subaccount cannot spend an ERC-20 token, transfer an NFT, or sign a domain message unless it possesses the corresponding asset under \mathcal{L} and \mathcal{H} . This unified authorization scheme allows for fine-grained delegation, (e.g., giving u rights over `example.com` while withholding ERC-20 funds) and consistent enforcement across on-chain and off-chain actions.

4 Properties of PASS

4.1 Threat Model

PASS operates under an organizational ownership model where a single entity (company, DAO, or individual) manages the EOA (pk, sk) and TEE infrastructure. This model is typical for use cases described in Section 1. Protected assets include the private key sk , balance ledger \mathcal{L} , provenance log \mathcal{H} , and access control policy $\text{Allow}(u, \alpha, x)$. In our model, we make the following trust assumptions:

- **TEE integrity.** Hardware isolation is sound; sk never leaves the enclave. Remote attestation guarantees that the enclave runs the genuine PASS binary.
- **Cryptography.** Standard primitives (signatures, hash functions, PRFs) are secure against polynomial-time adversaries.
- **Governance majority.** The deploying entity holds majority control in the governance mechanism (via DAO vote or admin privileges), ensuring it can invoke the blockchain-governed KMS for MPC-based key recovery and trigger fallback contracts when needed.

Adversary Model. An adversary \mathcal{A} may control the host OS and network, observe or modify all communication between enclaves, and submit arbitrary transaction requests. In particular, \mathcal{A} may attempt to:

- Exploit software bugs in the PASS implementation or TEE runtime,
- Forge or replay transactions to break consistency of \mathcal{H} ,
- Subvert provenance by violating $\text{Allow}(u, \alpha, x)$

Threats Out of Scope. Physical TEE compromise, supply-chain attacks, and malicious operators with unrestricted access (excluded due to organizational security controls) are regarded out of scope because the organizational model assumes controlled facilities and vetted hardware.

Security Goals. Given the assumptions above, PASS guarantees:

- **Integrity:** $\sum_{u \in \mathcal{U}} \mathcal{L}[u][\alpha] \leq \text{extDep}(\alpha)$ for all α , i.e., no assets can be created or double-spent internally.
- **Correctness:** If $\text{Allow}(u, \alpha, x)$ holds, then u can eventually realize x via Withdraw.
- **Privacy on-chain:** `InternalTransfer` operations do not affect `externalTrace(W)`, and to external observers, a PASS wallet is indistinguishable from an EOA.
- **Liveness:** For any authorized $\text{op} \in \text{Ops}$, if one enclave and quorum remain honest, then $\exists \tau$ s.t. $\text{Exec}(c, \text{op}) = \top$ within τ

4.2 Privacy and Safety

After establishing the threat model of PASS, we will address the central privacy and safety guarantees in our formal verification model. We sketch three central lemmas here, and create mechanized proofs in Lean, discussed in Appendix A.

Proposition 1 (Internal Transfer Privacy). *If two public states W_1, W_2 share the same pk , identical Outbox (T, ν) , and equal per-asset totals in A , then*

$$\text{externalTrace}(W_1) = \text{externalTrace}(W_2).$$

Proof Sketch. InternalTransfer updates only $(\mathcal{L}, \mathcal{H})$, never \mathcal{O} or total on-chain balances. OutboxProcessing in Figure 3 emits on-chain actions solely from \mathcal{O} (lines 2–4), hence the trace depends only on \mathcal{O} .

Corollary: EOA indistinguishability. If W_{pass} and W_{eoa} share the same pk , Outbox, and per-asset totals, then

$$\text{externalTrace}(W_{\text{pass}}) = \text{externalTrace}(W_{\text{eoa}}).$$

Thus, from an external observer’s perspective, a PASS account is indistinguishable from a standard EOA: internal transfers leave no observable footprint.

Proposition 2 (Asset Accessibility). *For any asset α , letting $L = \{u \mid \mathcal{L}[u][\alpha] > 0\}$,*

$$\sum_{u \in L} \mathcal{L}[u][\alpha] = \text{totalBalance}(\alpha),$$

and each $u \in L$ can spend its full balance $\mathcal{L}[u][\alpha]$ if and only if $\text{checkAllow}(u, \alpha, \mathcal{L}[u][\alpha], \mathcal{H})$ holds.

Proof Sketch. ClaimInbox credits \mathcal{L} ; InternalTransfer preserves the sum over u ; Withdraw debits \mathcal{L} and enqueues to \mathcal{O} , which OutboxProcessing realizes on-chain. checkAllow enforces reachability from provenance in \mathcal{H} .

Proposition 3 (Provenance Integrity). *Let $\text{extDep}(\alpha)$ be net external deposits (deposits minus realized withdrawals). For all reachable states,*

$$\sum_{u \in \mathcal{U}} \mathcal{L}[u][\alpha] = \text{extDep}(\alpha).$$

Proof Sketch. Induct on transitions. InboxDeposit increases extDep only; ClaimInbox moves already-deposited value into \mathcal{L} ; InternalTransfer conserves the sum over u ; Withdraw decreases the sum, and after OutboxProcessing broadcasts, extDep is reduced accordingly.

General signable messages (GSM). Model GSM rights as a domain-scoped virtual asset class \mathcal{A}_{GSM} . Let $\alpha = (\text{dom}) \in \mathcal{A}_{\text{GSM}}$ and permit

$$\text{SignGSM}(u, \text{dom}, m) \text{ iff } \text{Allow}(u, \alpha, 1).$$

Then (P2)–(P3) lift to GSM: there is always at least one domain signer, and GSM “balances” obey provenance conservation under transfers in \mathcal{H} .

4.3 Liveness

We define liveness as the property that any authorized operation remains eventually executable, despite enclave crashes, vendor compromise, or partial governance failure. Our model assumes deployment of PASS access logic on dstack, implemented in Section 5 and detailed in Appendix B.

Let \mathcal{E} denote active enclaves, \mathcal{C} containerized wallet backends, and \mathcal{P} governance policies. Each container $c \in \mathcal{C}$ stores sealed state

$$\sigma_c = \langle S, k_c, \text{meta} \rangle \quad \text{with} \quad S = (\text{pk}, \text{sk}, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H}),$$

where $k_c = \text{Derive}(\mathcal{K}, c)$ is derived from an MPC-managed Key Management Service (KMS) and meta records attestation. An operation $\text{op} \in \text{Ops}$ (deposit claim, transfer, withdrawal) executes as: $\text{Exec}(c, \text{op}) = \top$ iff $\text{Attested}(c) \wedge \Gamma(S)$, where $\text{Attested}(c)$ checks σ_c against registry \mathcal{R} and Γ is the quorum predicate over \mathcal{P} . To ensure vendor-agnostic portability, state migration $\mu : \mathcal{E} \rightarrow \mathcal{E}$ satisfies

$$\forall e_i, e_j \in \mathcal{E}, \quad e_j.\text{decrypt}(\sigma_{c, e_i}, k_c) = \sigma_{c, e_i},$$

Theorem 1 (Liveness Theorem). *If (i) the MPC KMS tolerates f_{KMS} faults in a t -of- n threshold, (ii) governance quorums are f_{gov} -resilient, and (iii) at least one enclave is uncompromised, then every $\text{op} \in \text{Ops}$ executes within bounded time τ :*

$$\forall \text{op} \in \text{Ops}, \exists \tau \geq 0 : \text{Exec}(c, \text{op}) = \top.$$

Proof Sketch. Progress follows from three guarantees: (1) the MPC KMS ensures k_c is always recoverable under threshold availability; (2) Γ encodes quorum rules that eventually relax after timeout Δt , ensuring some subset S satisfies $\Gamma(S) = \top$; (3) migration μ allows any enclave to re-host σ_c , preventing vendor lock-in or enclave failure. Algorithm 1 formalizes recovery and execution under these assumptions.

Algorithm 1 RecoverAndExecute(c, op)

- 1: Verify enclave e attestation Q_e against \mathcal{R} .
 - 2: Derive $k_c \leftarrow \text{Derive}(\mathcal{K}, c)$; decrypt σ_c .
 - 3: **if** $\text{CompromiseDetected}(e')$ **then**
 - 4: Rotate k_c and re-encrypt σ_c .
 - 5: **end if**
 - 6: Ensure $\Gamma(S) = \top$ for $S \subseteq \mathcal{P}(c)$.
 - 7: Execute op ; append to \mathcal{H} .
-

Key Properties. (1) *Portability:* any attested enclave can recover σ_c , independent of vendor, given that k_c is managed by KMS; (2) *Monotonic Recovery:* recovery steps form $\rho : \mathcal{S} \rightarrow \mathcal{S}$ with $\rho(s) \succeq s$; (3) *Domain Continuity:* certificates remain bound to $\mathcal{R}(c)$ under migration $\mu(c)$, preventing hijack or replay.

Implementation details of container orchestration, attestation, and KMS protocols are given in Appendix B.

5 Practical Implementations

5.1 Prototype Implementations

To demonstrate feasibility, we have implemented a prototype of PASS that captures the core architectural components.⁸ The prototype consists of:

- *Provenanced Access Model*, the core PASS architecture outlined in Section 3.1 with the Inbox-Outbox structure, asset ledgers, and provenanced rule as a Rust executable.
- *Web Application Frontend*, an AWS-hosted Typescript interface that allows users to deposit assets, initiate internal transfers, generate on-chain transactions, and sign arbitrary messages on Ethereum Sepolia testnet via WalletConnect v2.⁹
- *Multi-vendor TEE deployment*, we deploy the PASS enclave executable in both AWS Nitro Enclaves and Intel TDX with the dstack framework.

We evaluate these two enclave deployments as complementary paths with distinct tradeoffs. **AWS Nitro Enclaves** simplify deployment through AWS integration and attestation, but concentrate trust in a single vendor and store the provenance ledger externally, potentially weakening liveness and asset accessibility guarantees. **Intel TDX with dstack** provides a portable, open-source runtime across diverse hardware with decentralized operators and MPC-based key management, preserving full security properties by executing both Inbox-Outbox logic and provenance ledger inside the enclave, though at the cost of added governance complexity for operator authorization and policy enforcement.

Blockchain Connectivity. To ensure reliable interaction, PASS embeds Helios [1], a consensus-verified light client within the TEE, enabled via dstack deployment. Helios verifies blockchain data against consensus proofs rather than trusting external RPC endpoints, removing reliance on centralized providers while preserving real-time performance. This guarantees that PASS’s Outbox operates only on verified state, even if external network providers are compromised.

5.2 Performance Benchmarks

Methodology. To evaluate PASS, we benchmarked its performance using both AWS Nitro Enclaves and Intel TDX as secure backend environments. We selected a suite of representative wallet operations—wallet creation, balance queries, claims, transfers, withdrawals, provenance queries, and a comprehensive end-to-end workflow. To further stress-test PASS, we conducted batch experiments involving 10,000 consecutive deposit and claim operations, as well as transfer-then-withdraw scenarios utilizing multi-threaded processing. Each operation was run 100 times over 10 trials per environment, measuring throughput in operations per second (ops/sec). Both environments ran on identical virtual machines

⁸ Prototype can be found here: <https://github.com/jayyu23/pass-wallet-app>

⁹ Web application: <https://pass-wallet.jayyu.xyz/>

(4 cores, 2.5 GHz, 16 GB RAM) to ensure a fair comparison. Table 4 summarizes our results, with full details available in Appendix C.

Fig. 4. Internal PASS operation throughput (ops/sec) across AWS Nitro Enclaves and Intel TDX. All operations are in-memory state changes within the TEE with no blockchain interaction. Each value represents the average of 100 operations repeated 10 times per environment.

Operation	AWS Nitro	Intel TDX
Wallet Creation	6,531	14,970
Account Balance Queries	530,943	935,428
Inbox-to-Subaccount Claims	14,856	33,064
Internal Subaccount Transfers	32,953	71,292
Withdrawal Request Creation	18,291	39,702
Transaction History Queries	3,757	7,685
End-to-End Workflow ¹⁰	35,663	78,433
Batch Deposit & Claim	481	724
Multi-threaded Operations (5 threads)	32,189	55,124

Analysis. The observed performance differences primarily stem from their distinct hardware architectures. AWS Nitro Enclaves employ a disaggregated system design, where the host server and the TEE hardware are physically separated. Communication occurs via vsock, and the enclave itself runs on a dedicated, lower-performance coprocessor. In contrast, Intel TDX leverages a special instruction set to provide enclave functionality directly on the main processor, avoiding the overhead of data transfer. As a result, Intel TDX is able to deliver significantly higher throughput for in-enclave operations compared to AWS Nitro Enclaves.

These preliminary benchmarks show that both Nitro and TDX backends can sustain high throughput for critical wallet operations, confirming that PASS is practical for real-world deployment.

Comparison with Related Work. To contextualize our performance results, we compare PASS with other TEE-based blockchain systems and traditional custodial solutions. Most existing wallet solutions operate under entirely different trust and security models that preclude meaningful performance comparisons. Hardware wallets prioritize air-gapped security over throughput, while custodial services sacrifice transparency for performance optimizations that PASS cannot adopt without compromising its core security guarantees. Additionally, no prior work combines TEE-based execution with comprehensive provenance tracking and rule-based access controls, making PASS’s architecture fundamentally distinct from existing approaches.

Liquefaction. The Liquefaction system [7] represents the closest architectural parallel to PASS, yet demonstrates significantly higher latencies. Liquefaction exhibits up to 1,036.9 seconds for finalized confirmation due to its dependence on external blockchain oracles for operation authorization. This design choice fundamentally limits scalability and user experience. In contrast, PASS’s innovative Inbox-Outbox architecture enables all internal operations to execute within the

TEE without blockchain dependencies, achieving sub-millisecond latencies while maintaining equivalent security guarantees through cryptographic auditability.

Traditional Custodians. While custodial solutions do not publish detailed performance benchmarks, they achieve high throughput by centralizing trust and sacrificing transparency—precisely the problems PASS was designed to solve. Our TEE-based approach introduces only 5-7% performance overhead compared to native execution [18], yet delivers fundamentally superior security properties: cryptographic auditability, decentralized operation, elimination of single points of failure, and user-controlled access policies. PASS thus achieves the performance characteristics of centralized systems while providing the security guarantees traditionally associated only with self-custody solutions.

6 Applications

PASS is a powerful primitive that enables applications across three major blockchain verticals: access control, custody & compliance, and scalability. Each vertical captures a distinct research area, with provenance-based subaccounts providing a unifying abstraction.

Access Control. PASS’s multi-user subaccount model enables flexible permission scopes and safe delegation. As AI agents begin managing crypto assets, there is a clear need to constrain their authority. With PASS, an agent can be granted a subaccount holding only limited funds or specific signing rights, allowing it to operate autonomously without risking the entire wallet; if compromised, damage is capped to that subaccount. Similarly, DAO voting can be delegated by granting signature rights restricted to governance DApps, without granting spending authority. Organizations can also provision *identity subaccounts* that map to individuals or teams under a shared ENS or corporate identity, each with scoped balances and signing authority tracked through provenance.

Custody & Compliance. Internal privacy, built-in permissioning, and auditable provenance make PASS well-suited to enterprise and regulatory contexts. A flagship application is *private payroll*: on-chain salary payments today risk leaking sensitive information such as compensation levels or departmental budgets. With PASS, salaries are distributed privately as internal subaccount transfers, leaving a verifiable provenance trail for audits, but without exposing per-employee details on-chain. Only periodic batched payouts exit via the Outbox, balancing privacy with regulatory accountability. Beyond payroll, compliant organizational wallets can allocate explicit departmental budgets while enabling regulator-grade proofs derived from the provenance log, and B2B netting allows vendors to settle obligations internally, with only net amounts leaving the wallet to reduce leakage while maintaining auditability.

Scalability. Finally, PASS supports blockchain scalability use cases where per-transaction gas costs or cross-domain coordination overhead are prohibitive. A flagship example is *gasless appchains*: high-throughput applications such as central limit order books (CLOBs) can internalize matching and settlement as subaccount transitions, with the Outbox emitting only periodic, verifiable batch settlements. Similarly, cross-chain bridges can leverage the Inbox–Outbox ab-

Vertical	Application	Context	PASS Enables
Access Control	AI Agent Wallets	Autonomous agents interacting with crypto assets [38] require bounded signing authority.	Scoped subaccounts grant limited balances and domain-specific rights, enabling safe delegation without exposing the root key.
	DAO Voting Delegation	DAOs depend on address-based voting and proxy delegation [11].	Fine-grained voting rights delegation auditable via provenance; delegation can be restricted to specific DApps or domains.
	Identity Subaccounts	Organizations map ENS or organizational identities to individuals or teams.	Each subaccount holds scoped balances and signing rights, with provenance tracking under a shared organizational identity.
Custody & Compliance	Private Payroll	On-chain payrolls reveal sensitive salary information [4].	Salary distribution through internal gasless transfers with auditable logs; external payouts are batched via the Outbox.
	Compliant Org Wallets	Firms must maintain accountability and BSA/AML recordkeeping [15].	Departmental and individual subaccounts with explicit budgets; provenance ensures regulator-grade auditability and enables external proof exploration without disclosing private flows.
	Private B2B Netting	Supply-chain and recurring multi-party payments require confidentiality.	Vendor subaccounts allow internal netting of obligations, with PASS acting as a clearing house. Only net settlements exit via the Outbox, minimizing leakage while retaining auditability.
Scalability	Gasless Appchains	High-throughput applications (e.g., trading engines) face prohibitive gas costs on-chain.	Internalize application logic as subaccount transitions; Outbox emits verifiable batch settlements, ensuring correctness with minimal gas.
	Cross-chain Bridges	Bridges require strong custody and coordination of operators [39].	Inbox-Outbox abstractions enforce provenance across domains; FIFO nonce handling prevents replay or race conditions.
	L2 Composability	Rollups and L2s fragment assets and user experience [35].	Treat each L2 as a scoped subaccount; maintain private balance transfers internally, and settle externally only when crossing domains.

Fig. 5. Applications of PASS across three blockchain verticals: access control, custody & compliance, and scalability. Each vertical highlights flagship use cases enabled by provenance-based subaccounts.

straction to enforce provenance across domains and ensure FIFO nonce discipline to prevent replay, while L2 composability is achieved by treating each rollup or sidechain as a scoped subaccount, allowing private balance transfers internally and settling externally only on domain exits.

7 Discussion and Future Work

As PASS matures beyond a single-wallet instantiation, several directions emerge for strengthening its composability, verifiability, and practical performance. Appendix D provides a high-level construction of extension designs for PASS.

Composability and InterPASS Transactions. Beyond single-wallet privacy and provenance, a key challenge is enabling secure interaction across wallets in supply-chain or inter-organizational settings. The trivial method of direct EOA transfers compromise privacy, so we envision each PASS instance as a PCN (Payment Channel Network) hub, similar to XTransfer [6], with a central Hub aggregating signed payment intents $\mathcal{T} := (a_i, x_i, s_i, r_i)_{i=1}^n$, computing a netted settlement \mathcal{S} , and updating balances via a multi-party protocol akin to Thora [5]. Optimistically this remains off-chain and private, while disputes invoke atomic on-chain enforcement. This design leverages provenance guarantees for confidential, auditable inter-wallet transfers; full specification is left to future work.

Instantiation on zkVM. The current PASS design primarily relies on trusted execution environments (TEEs) to safeguard keys and enforce Inbox-Outbox provenance rules, but this trust model depends on vendor attestation and hardware security, which may be insufficient in adversarial or regulatory contexts that demand independent verification. zkVMs provide a natural next step by enabling provenance-based control to be verified through succinct cryptographic proofs rather than trusted hardware. By batching transitions over \mathcal{L} , \mathcal{H} , \mathcal{I} , and \mathcal{O} and producing a proof π , a minimalist verifier contract can update commitments and enforce outbox execution while preserving privacy. This direction complements the Chainless stack architecture [31], suggesting a path toward modular, interoperable deployments of PASS with stronger external verifiability and broader applicability.

Expansion of Formal Model and Further Benchmarks. We aim to extend our Lean 4 formalization from wallet logic to enclave assumptions, quorum dynamics, MPC-based recovery, and cross-vendor migration. Empirically, while alternatives (multisigs, ERC-4337, custodial systems) lack benchmarks, PASS already demonstrates throughput; we will expand evaluation to latency, concurrency, and vendor diversity. Together these efforts provide application-driven validation and stronger evidence for PASS as both formally sound and practically deployable.

8 Conclusion

We presented PASS, a Provenanced Access Subaccount System, as a new wallet architecture that enforces access by provenance rather than by identity or role. The Inbox-Outbox mechanism ensures all assets have a verifiable lineage, while internal transfers remain private and indistinguishable from a standard EOA account.

Our contributions are threefold: (i) a formal model in Lean 4 proving privacy, accessibility, and provenance integrity; (ii) a working prototype with enclave backends on AWS Nitro Enclaves and Intel TDX via dstack, integrated with WalletConnect; and (iii) initial benchmarks showing the feasibility of high-throughput operation. These results demonstrate that provenance-based control can be realized today with mainstream blockchain tools.

Looking forward, richer evaluations across multi-vendor TEEs, integration with zkVMs, and exploration of organizational and agent-based custody are

promising directions. By combining rigorous formal guarantees with deployable prototypes, PASS advances the design space between strict self-custody and flexible shared access, offering a path toward practical, privacy-preserving wallet security.

References

1. a16z crypto: Helios: Ethereum light client. <https://github.com/a16z/helios> (2025), gitHub repository, accessed: 2025-09-14
2. Argent: Build with Argent (2023), <https://docs.argent.xyz/>
3. Arun, A., Setty, S., Thaler, J.: Jolt: SNARKs for virtual machines via lookups. Cryptology ePrint Archive, Paper 2023/1217 (2023), <https://eprint.iacr.org/2023/1217>
4. Auer, R., Farag, M., Lewrick, U., Orazem, L., Zoss, M.: Banking in the shadow of bitcoin? the institutional adoption of cryptocurrencies. CESifo Working Paper 10355, Munich (2023), <https://hdl.handle.net/10419/271999>
5. Aumayr, L., Abbaszadeh, K., Maffei, M.: Thora: Atomic and privacy-preserving multi-channel updates. Cryptology ePrint Archive, Paper 2022/317 (2022). <https://doi.org/10.1145/3548606.3560556>, <https://eprint.iacr.org/2022/317>
6. Aumayr, L., Avarikioti, Z., Salem, I., Schmid, S., Yeo, M.: X-transfer: Enabling and optimizing cross-PCN transactions. Cryptology ePrint Archive, Paper 2025/272 (2025), <https://eprint.iacr.org/2025/272>
7. Austgen, J., Fábrega, A., Kelkar, M., Vilardell, D., Allen, S., Babel, K., Yu, J., Juels, A.: Liquefaction: Privately liquefying blockchain assets. arXiv e-prints **2412.02634** (2024), arXiv:2412.02634 [cs.CR]
8. Bambysheva, N.: Breaking: Could bybit's \$1.5b hack have been stopped? ledger, cz react. Forbes (2025), <https://www.forbes.com/sites/digital-assets/2025/02/22/breaking-could-bybits-14b-hack-have-been-stopped-ledger-cz-react/>
9. Brunner, D., Karame, G.O., Li, W., Tschorsch, F.: Who stores the private key? an exploratory study about user preferences of key management for blockchain-based applications. In: Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP). pp. 276–287. SciTePress (2021). <https://doi.org/10.5220/0010226102760287>, <https://publications.ait.ac.at/en/publications/who-stores-the-private-key-an-exploratory-study-about-user-prefer>
10. Buterin, V.: Ethereum: A next-generation smart contract and decentralized application platform. Ethereum whitepaper, <https://ethereum.org/en/whitepaper/> (2014), accessed: 2024-10-16
11. Contributors, E.: Decentralized autonomous organizations (DAOs). <https://ethereum.org/en/dao/>, accessed: 2024-11-13
12. Cutler, J.W., Disselkoben, C., Eline, A., He, S., Headley, K., Hicks, M., Hietala, K., Ioannidis, E., Kastner, J., Mamat, A., McAdams, D., McCutchen, M., Rungta, N., Torlak, E., Wells, A.M.: Cedar: A new language for expressive, fast, safe, and analyzable authorization. arXiv preprint arXiv:2403.04651 (2024), <https://arxiv.org/pdf/2403.04651>
13. ERC-4337 Documentation: UserOperation – ERC-4337 Documentation (2023), <https://www.erc4337.io/docs/understanding-ERC-4337/user-operation>
14. Erinle, Y., et al.: Shared-custodial wallet for multi-party crypto-asset management. *Future Internet* **17**(1), 7

15. Financial Crimes Enforcement Network, Department of the Treasury: Threshold for the requirement to collect, retain, and transmit information on funds transfers and transmittals of funds that begin or end outside the united states, and clarification of the requirement to collect, retain, and transmit information on transactions involving convertible virtual currencies and digital assets with legal tender status (Oct 2020), <https://www.federalregister.gov/documents/2020/10/27/2020-23756/threshold-for-the-requirement-to-collect-retain-and-transmit-information-on-funds-transfers-and>, proposed Rule, Docket No. FINCEN-2020-0002; RIN 1506-AB41, 31 CFR Parts 1010, 1020
16. Fireblocks: Governance and policy engine (2025), <https://www.fireblocks.com/platforms/governance-and-policy-engine/>
17. Goldman Sachs Research: Crypto: A new asset class? (2021), <https://www.goldmansachs.com/insights/top-of-mind/crypto-a-new-asset-class>
18. Intel: Performance Considerations of Intel® Trust Domain Extensions on 4th Generation Intel® Xeon® Scalable Processors (2025), <https://www.intel.com/content/www/us/en/developer/articles/technical/trust-domain-extensions-on-4th-gen-xeon-processors.html>
19. Lean Community: Lean Documentation (2025), <https://lean-lang.org/documentation/>
20. Loopring: Loopring Smart Wallet Documentation (2023), <https://docs-wallet.loopring.io/>
21. Mangipudi, E.V., Desai, U., Minaei, M., Mondal, M., Kate, A.: Uncovering impact of mental models towards adoption of multi-device crypto-wallets. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 3153–3167 (2023). <https://doi.org/10.1145/3576915.3623218>, <https://dl.acm.org/doi/10.1145/3576915.3623218>
22. McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative instructions and software model for isolated execution. In: HASP. p. 10 (2013)
23. Moosavi, M., Clark, J.: Lissy: Experimenting with on-chain order books. arXiv preprint arXiv:2101.06291 (2021), <https://arxiv.org/abs/2101.06291>
24. Oasis Labs: Oasis protocol: Privacy-enabled blockchain platform. <https://oasisprotocol.org/> (2024), <https://oasisprotocol.org/>, accessed: 2024-10-06
25. Patairya, D.K.: What are smart contract wallets? Cointelegraph (2024), <https://cointelegraph.com/explained/what-are-smart-contract-wallets>
26. Patlan, A.S., Sheng, P., Hebbar, S.A., Mittal, P., Viswanath, P.: AI agents in cryptoland: Practical attacks and no silver bullet. Cryptology ePrint Archive, Paper 2025/526 (2025), <https://eprint.iacr.org/2025/526>
27. Pertsev, A., Semenov, R., Storm, R.: Tornado cash privacy solution version 1.4 (2019), <https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf>
28. Safe: How do Safe Smart Accounts work? (2025), <https://docs.safe.global/advanced/smart-account-concepts>
29. Schneider, M., Masti, R.J., Shinde, S., Capkun, S., Perez, R.: Sok: Hardware-supported trusted execution environments (2022), <https://arxiv.org/abs/2205.12742>
30. Seehausen, V.: Eip-7702: A win for smart accounts in ethereum’s pectra upgrade? Safe Global Blog (2024), <https://safe.global/blog/eip-7702-smart-accounts-ethereum-pectra-upgrade>
31. Seong, B., Gebheim, P.: Chainless apps: A modular framework for building apps with web2 capability and web3 trust (2025), <https://arxiv.org/abs/2505.22989>

32. Sign-In with Ethereum Documentation: EIP-4361: Sign-In with Ethereum (2023), <https://docs.login.xyz/general-information/siwe-overview/eip-4361>
33. Snapshot Documentation: Voting on Snapshot (2023), <https://docs.snapshot.box/user-guides/voting/vote>
34. Tally: Gnosis Safe Overview (2025), <https://docs.tally.xyz/set-up-and-technical-documentation/using-governor-with-gnosis-safe/gnosis-safe>
35. Thibault, L.T., Sarry, T., Hafid, A.S.: Blockchain scaling using rollups: A comprehensive survey. *IEEE Access* **10**, 93039–93054 (2022). <https://doi.org/10.1109/ACCESS.2022.3200051>
36. Turnkey: Policy quickstart (2025), <https://docs.turnkey.com/concepts/policies/quickstart>
37. Venly: Omnibus vs Segregated Wallets in Crypto (2025), <https://docs.venly.io/docs/omnibus-vs-segregated-wallets-in-crypto>
38. Walters, S., Gao, S., Nerd, S., Da, F., Williams, W., Meng, T.C., Han, H., He, F., Zhang, A., Wu, M., Shen, T., Hu, M., Yan, J.: Eliza: A web3 friendly ai agent operating system. arXiv preprint arXiv:2501.06781 (2025), <https://arxiv.org/pdf/2501.06781v1>
39. Wormhole Foundation: Guardians (2025), <https://wormhole.com/docs/learn/infrastructure/guardians/>
40. Yu, Y., Chang, M., Reijers, W., McAuley, D.: How cryptocurrency users choose and secure their wallets. *Proceedings of the ACM on Human-Computer Interaction* **8**(CSCW2), 1–27 (2024). <https://doi.org/10.1145/3642534>, <https://dl.acm.org/doi/10.1145/3642534>
41. Zhou, S., Wang, K., Yin, H.: Dstack: A Zero Trust Framework for Confidential Containers (2025), <https://arxiv.org/abs/2509.11555>
42. of “What Wallet Features Do Users Want...”, A.: What wallet features do users want for their cryptocurrencies? *Journal name (if peer-reviewed)* (2025), <https://www.tandfonline.com/doi/full/10.1080/00036846.2025.2536752?src=,> survey study, accessed: 2025-09-14

A Formal Verification

We consider the formal model introduced in Section 3.1 of PASS as a state machine in Lean4. Our goal is to prove that, under the PASS design, several critical security properties always hold. We outline these properties and our approach to verifying them.

A.1 Property: Privacy of Internal Transfers

In the PASS design, the `InternalTransfer` operation updates only the off-chain ledger \mathcal{L} and the provenance history \mathcal{H} . It never modifies the Outbox \mathcal{O} , meaning there is no on-chain transaction triggered by an internal transfer. Consequently, external observers see no change in on-chain state. We formalize this by considering the on-chain view that an external party has for a given PASS Wallet. We can model this visible on-chain state as a triple:

$$\mathcal{W}_{pub} = (\text{pk}, \mathcal{O}, A),$$

where:

pk is the externally owned account (EOA) address,
 $O = (T, \nu)$ is the external view of the *outbox*, where T is a list of enqueued transactions and ν is a nonce,
 A is the external view of the Asset ledger \mathcal{L} , containing a list of asset entries, each of the form (α, x) with a unique identifier α and total balance x .

External Trace. The *external trace* of a PassAccount state \mathcal{W}_{pub} is the list of externally visible actions, obtained by mapping each transaction in the outbox T to an on-chain action:

$$\text{externalTrace}(\mathcal{W}_{pub}) = \{ \text{outboxTx}(t) \mid t \in T \}.$$

No internal transfers appear in $\text{externalTrace}(\cdot)$.

States That Differ Only by Internal Transfers. We say that two PassAccount states $\mathcal{W}_1 = (\text{pk}_1, O_1, A_1)$ and $\mathcal{W}_2 = (\text{pk}_2, O_2, A_2)$ *differ only by internal transfers* if:

1. $\text{pk}_1 = \text{pk}_2$,
2. $O_1 = O_2$ (i.e. both the list of outbox transactions and the nonce are the same),
3. For every asset identifier α , the total balance of that asset in A_1 equals the total balance of the same asset in A_2 .

Letting $\text{getAsset}(\mathcal{W}, \alpha)$ return the asset α within state \mathcal{W} or \perp , we require $\forall \alpha$:

$$\text{totalBalance}(\text{getAsset}(\mathcal{W}_1, \alpha)) = \text{totalBalance}(\text{getAsset}(\mathcal{W}_2, \alpha)).$$

Theorem 2 (Privacy of Internal Transfers). *If two PassAccount states \mathcal{W}_1 and \mathcal{W}_2 differ only by internal transfers, then their externally visible traces are identical:*

$$\text{externalTrace}(\mathcal{W}_1) = \text{externalTrace}(\mathcal{W}_2).$$

Proof. By assumption, $O_1 = O_2$. Hence both states have the same outbox transaction list T . Since

$$\text{externalTrace}(\mathcal{W}_i) = \text{map}(\text{outboxTx}, T_i) \quad \text{for } i \in \{1, 2\},$$

and $T_1 = T_2$, it follows that $\text{externalTrace}(\mathcal{W}_1)$ and $\text{externalTrace}(\mathcal{W}_2)$ are identical.

Thus, any sequence of purely internal transfers, changing only the private ledger \mathcal{L} in PASS, is invisible to an external observer, preserving the privacy of how assets move within the wallet.

A corollary of this Internal Transfer property is that a PASS account is indistinguishable from a regular EOA account.

Corollary 1 (Indistinguishability from EOA). *Let \mathcal{W}_{pass} be a PASS state that (i) has the same public key pk , (ii) the same outbox, including nonce and transaction list, and (iii) the same total balances of each asset as a reference EOA state \mathcal{W}_{eoa} . Then*

$$\text{externalTrace}(\mathcal{W}_{pass}) = \text{externalTrace}(\mathcal{W}_{eoa}).$$

In particular, an external observer cannot distinguish \mathcal{W}_{pass} from \mathcal{W}_{eoa} .

Proof. By assumption, the only potential differences between $\mathcal{W}_{\text{pass}}$ and \mathcal{W}_{eoa} lie in internal ledger details. From Theorem 1, such internal transfers do not affect the externally visible outbox transactions or total on-chain balances. Thus, their external traces coincide, and $\mathcal{W}_{\text{pass}}$ is indistinguishable from \mathcal{W}_{eoa} .

A.2 Property: Asset Accessibility

In addition to keeping internal transactions private, PASS should ensure that for every asset in the system, there exists some sub-account that can access it and withdraw it from the system. This prevents a situation where funds get "stuck" without any reachable key or policy to move them. We define asset accessibility as follows:

Theorem 3 (Asset Accessibility). *For any PASS account \mathcal{W} and asset identified by α , there exists a list of addresses L such that:*

1. *For every address $u \in L$, the user can access their full balance:*

$$\text{checkBalance}(s, \alpha, u, \text{balance}(u)) = \text{true}$$

2. *The sum of all accessible balances equals the total balance of the asset:*

$$\sum_{u \in L} \mathcal{L}[u][\alpha] = \text{totalBalance}(\alpha)$$

Proof. We construct L as the set of all addresses with non-zero balances in the balance map for α :

$$L = \{u \mid \mathcal{L}[u][\alpha] > 0\}$$

For the first property, given any $u \in L$, the definition of `checkBalance` verifies if the user has sufficient balance for the specified amount. When checking against the user's own balance, this is trivially satisfied since $\mathcal{L}[u][\alpha] \geq \mathcal{L}[u][\alpha]$ for all u .

For the second property, we observe that `totalBalance` is defined as the sum of all balances in the asset's balance map:

$$\text{totalBalance}(\alpha) = \sum_{u \in \mathcal{L}} \mathcal{L}[u][\alpha]$$

By construction, L contains the only nonzero values $u \in \mathcal{L}$, so:

$$\sum_{u \in L} \mathcal{L}[u][\alpha] = \text{totalBalance}(\alpha)$$

Therefore, both properties hold for our constructed list L .

We can prove a similar trait for General Signable Messages, showing that for any inbound GSM, there is some subaccount signer that has access to it.

Theorem 4 (GSM Accessibility). *For any GSM and domain, there exists a user with signing access to that domain.*

Proof. For any domain d , we can directly construct a user u who has signing access to d as follows:

$$u = \text{getSigner}(d)$$

By definition, `getSigner` returns either:

- A specifically assigned signer if one exists in the domain-address map, or
- The default signer otherwise

Therefore, every domain is guaranteed to have at least one user with signing access, ensuring no GSM domain becomes inaccessible.

A.3 Property: Provenance Integrity

Provenance integrity specifies that every asset held by any sub-account in PASS must have an origin traceable to an external deposit into the Inbox. At any state of the system, for each asset entry in a sub-account’s balance, we can find a sequence of internal transfers that leads back to an Inbox deposit from an external address. This property ensures no sub-account can conjure funds from nothing, such as through double-spending or inflation within the system, and that the internal ledger is consistent with on-chain reality. In Lean, we model the state of the PASS wallet including Inbox, Outbox, and all sub-accounts and prove an invariant that whenever a sub-account has x units of asset α , the global state contains a record of at least x units of α having been deposited, minus any that have been withdrawn.

Theorem 5 (Provenance Integrity).

For any PASS account $S = (\text{pk}, \text{sk}, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ and any asset α , the total amount of α held across all subaccounts can never exceed the total amount of external deposits (minus any that have already been withdrawn). Formally,

$$\sum_{u \in U} \mathcal{L}[u][\alpha] \leq \text{externalDeposits}(\alpha),$$

where U is the set of subaccounts in the system, and $\text{externalDeposits}(\alpha)$ is the net total of asset α that has been introduced into the wallet from outside addresses, recorded in the inbox \mathcal{I} .

Proof. We model the system’s state $(\text{pk}, \text{sk}, \nu, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ as in Figure 3, including the Inbox \mathcal{I} , Outbox \mathcal{O} , and the internal ledger \mathcal{L} . We define a *provenance record* $P(\alpha, u)$ that captures the total amount of asset α subaccount u is authorized to hold, based on Inbox claims and internal transfers.

Base Case: Immediately after `CreatePassWallet`, no assets have been deposited, and no subaccount balances exist:

$$\forall u, \mathcal{L}[u][\alpha] = 0, \quad \forall \alpha, \text{externalDeposits}(\alpha) = 0.$$

Hence,

$$\sum_{u \in U} \mathcal{L}[u][\alpha] = 0 \leq 0 = \text{externalDeposits}(\alpha).$$

Inductive Step: Assume the invariant holds after $n - 1$ operations. We prove it remains valid after the n th operation, for each possible type of operation:

- (1) **External Deposit:** Suppose an amount x of asset α is deposited into the wallet:

$$\text{externalDeposits}(\alpha) \leftarrow \text{externalDeposits}(\alpha) + x.$$

No subaccount balances $\mathcal{L}[u][\alpha]$ change. Thus, the left side of our inequality (sum of subaccount balances) is unchanged, while the right side strictly increases by x , so the invariant continues to hold.

- (2) **Inbox Claim:** If subaccount u claims amount x of asset α from the Inbox \mathcal{I} :

$$\mathcal{L}[u][\alpha] \leftarrow \mathcal{L}[u][\alpha] + x, \quad P(\alpha, u) \leftarrow P(\alpha, u) + x.$$

Since x was already accounted for in $\text{externalDeposits}(\alpha)$ (but not yet allocated to any subaccount), the total subaccount balances remain $\leq \text{externalDeposits}(\alpha)$.

- (3) **Internal Transfer:** If subaccount u_1 transfers x of α to subaccount u_2 :

$$\mathcal{L}[u_1][\alpha] \leftarrow \mathcal{L}[u_1][\alpha] - x, \quad \mathcal{L}[u_2][\alpha] \leftarrow \mathcal{L}[u_2][\alpha] + x,$$

and $P(\alpha, u_2)$ increases by x . Because the sum of all subaccount balances does not change, our invariant is preserved.

- (4) **Withdrawal:** If subaccount u withdraws x of α :

$$\mathcal{L}[u][\alpha] \leftarrow \mathcal{L}[u][\alpha] - x,$$

which reduces the total subaccount balances. Since $\text{externalDeposits}(\alpha)$ remains unchanged (assets have simply exited the wallet), the inequality still holds.

Thus, by induction over all possible operations,

$$\sum_{u \in \mathcal{U}} \mathcal{L}[u][\alpha] \leq \text{externalDeposits}(\alpha)$$

for every reachable state. No subaccount can create new assets out of nothing, and all assets in the system are fully accounted for by provenanced deposits.

B Dstack Architecture and Execution Model

B.1 Component Overview

- **Dstack-OS:** Minimal Linux-based VM image with a verifiable boot chain. Measures bootloader, kernel, root filesystem (RootFs) via MRTD and RTMR registers. Integrates `dm-verity` for data integrity and monotonic counters for anti-rollback.
- **Dstack-KMS:** Blockchain-governed MPC network running in TEE nodes for key derivation and rotation. Provides:

$$k_c = \text{HMAC}_{k_{\text{root}}}(c_{\text{id}})$$

Rotation is triggered via governance (t -of- n threshold).

- **Dstack-Gateway/Ingress:** TLS termination inside TEEs, its novel ZeroTrust-TLS protocol binding \mathcal{X}_c to $\mathcal{R}(c)$, enabling verifiable HTTPS ingress.

B.2 Governance Layer

Two contracts form the trust base:

- *DstackKms*: Global registry of KMS nodes and application digests.
- *DstackApp*: Per-application contract specifying quorum rules, allowed image hashes, and authorized enclaves.

B.3 Workflow Phases

1. **Attestation**: Enclave e produces quote Q_e ; verified against \mathcal{R} .
2. **Secret Provisioning**: MPC nodes compute k_c , release to e if Q_e is valid.
3. **Container Launch**: Dstack-OS mounts RootFs, decrypts state σ_c with k_c .
4. **Ingress Binding**: TLS certificate \mathcal{X}_c generated inside e ; bound to $\mathcal{R}(c)$.
5. **Audit Logging**: Migration, key rotation, and governance events appended to \mathcal{H} .

B.4 Fault Tolerance

- Vendor failure: μ enables re-deployment across Intel/AMD TEEs.
- KMS compromise: key rotation via MPC thresholds restores forward/backward secrecy.
- Governance deadlock: Δt fallback in Γ ensures quorum reduction and progress.

B.5 Liveness Proof Sketch

Combining MPC-based key recovery, quorum fallback, and portable containers yields:

$$\forall \text{op} \in \mathcal{O}_c, \quad \Pr[\text{Exec}(c, \text{op}) = \top] = 1$$

under assumptions of f -resilient MPC nodes and at least one uncompromised TEE KMS instance.

C Detailed Benchmark Results

This appendix provides comprehensive benchmark statistics for both AWS Nitro Enclaves and Intel TDX environments. All operations are internal PASS state changes that occur within the TEE with no blockchain network interaction. Each operation was executed 100 times per trial, with 10 independent trials per environment. The tables below show summary statistics including mean, standard deviation, and range.

C.1 AWS Nitro Enclaves Results

C.2 Intel TDX Results

C.3 Performance Analysis

The statistical analysis reveals several key insights about PASS’s internal operation performance:

Table 1. AWS Nitro Enclaves benchmark statistics (ops/sec) across 10 trials

Operation	Mean	Std Dev	Min	Max
Wallet Creation	6,531	531	5,446	7,042
Account Balance Queries	530,943	675	530,216	531,915
Inbox-to-Subaccount Claims	14,856	476	14,326	15,504
Internal Subaccount Transfers	32,953	1,322	31,447	35,088
Withdrawal Request Creation	18,291	1,254	15,873	19,608
Transaction History Queries	3,757	45	3,650	3,802
End-to-End Workflow	35,663	6,234	21,739	40,650
Batch Deposit & Claim	481	13	467	497
Multi-threaded Operations (5 threads)	32,189	884	31,200	34,364

Table 2. Intel TDX benchmark statistics (ops/sec) across 10 trials

Operation	Mean	Std Dev	Min	Max
Wallet Creation	14,970	4,652	8,696	24,390
Account Balance Queries	935,428	338,450	383,142	1,470,588
Inbox-to-Subaccount Claims	33,064	5,197	25,445	41,667
Internal Subaccount Transfers	71,292	17,623	47,170	101,010
Withdrawal Request Creation	39,702	7,434	29,155	51,813
Transaction History Queries	7,685	872	6,452	8,929
End-to-End Workflow	78,433	17,403	59,524	110,497
Batch Deposit & Claim	724	81	636	893
Multi-threaded Operations (5 threads)	55,124	16,744	28,329	81,301

- **Performance Consistency:** AWS Nitro Enclaves demonstrate superior consistency with lower standard deviations across all operations. The coefficient of variation ranges from 0.1% (Account Balance Queries) to 17.5% (End-to-End Workflow).
- **Peak Performance:** Intel TDX achieves higher mean throughput but with significantly greater variance. Standard deviations are 10-500x larger than Nitro, indicating less predictable performance.
- **Production Considerations:** These internal operation benchmarks demonstrate PASS’s ability to handle thousands of wallet operations per second within the TEE, with the actual deployment bottleneck being blockchain transaction submission rather than internal processing.

The statistical summary confirms that both platforms support practical PASS deployment, with Nitro offering predictable performance and Intel TDX providing higher peak throughput at the cost of consistency.

D Design Extensions

D.1 Composability and InterPASS Transactions

While PASS provides privacy and provenance guarantees for a single wallet instance, an important question is how multiple PASS wallets may interact. This

is important in supply-chain use cases where each PASS represents a single company in the supply chain.

Trivially, PASS A can send an on-chain transaction from its Outbox to PASS B’s EOA account, and this will land in the Inbox of PASS B, ready to claim. However, this undermines the privacy guarantees that PASS offers at the EOA level, and is undesirable in cases where inter-organizational asset transfers contain sensitive information, such as order details.

We propose a non-trivial approach inspired by cross-Payment Channel Network (PCN) protocols such as XTransfer [6]. We can treat each PASS instance akin to a PCN hub node, with state \mathcal{L} and transaction history \mathcal{H} . As with XTransfer, we can assume a star topology of PASS nodes, with a central PASS Hub in communication with all PASS nodes. The PASS Hub will aggregate all InterPASS transactions $\mathcal{T} := \{(a_i, x_i, s_i, r_i)\}_{i=1}^n$, where (a_i, x_i, s_i, r_i) represent the asset, amount, sender, and receiver, each signed by the s_i Outbox.

The Hub aggregates these signed payment intents, computes a netted settlement plan \mathcal{S} , and coordinates a secure multi-party update protocol to commit the resulting balances, similar to Thora [5]. In the optimistic case, this process completes entirely off-chain, and each participant’s balance is updated in the PASS ledger without revealing the underlying transaction graph. If a dispute or failure occurs, any participant may trigger an on-chain enforcement transaction that finalizes \mathcal{S} atomically via the direct transfer mechanism, thus guaranteeing safety.

This approach leverages PASS’s provenance and state design to make each instance an atomic node in PCN-style systems like X-Transfer. Here, we outline how PASS Hubs can enable confidential, auditable cross-organization transfers, while full formal specification and protocol implementation is deferred for future work.

D.2 Instantiation on zkVM

The design of PASS presented thus far relies on trusted execution environments (TEEs) such as AWS Nitro or Intel TDX to enforce provenance-based control. TEEs provide strong isolation and efficient execution, allowing internal transfers to remain private while ensuring that every outbox action has a verifiable lineage. However, the trust model is anchored in hardware attestation: users and regulators must accept that the enclave executes the correct code as claimed. While sufficient for many organizational custody scenarios, this assumption limits transparency and long-term auditability. To address this limitation, we explore an alternative instantiation of PASS in a zero-knowledge virtual machine (zkVM), replacing hardware trust with cryptographic proofs of correctness.

Zero-knowledge virtual machines (zkVMs) provide a proving environment in which arbitrary state machines can run with strong correctness guarantees [3]. By producing a succinct proof π of execution, zkVMs offer integrity assurances similar to TEEs but without reliance on vendor-specific hardware. This shift makes zkVMs particularly suitable for contexts such as institutional custody, multi-party supply chains, or inter-organizational accounting where external verifiability and regulator-grade auditability are paramount.

In a zkVM instantiation of PASS, the execution pipeline mirrors the enclave-based design but replaces attestation with cryptographic proofs. The process proceeds in four stages:

1. **Batch formation.** Transaction requests (claims, transfers, withdrawals) are first queued off-chain into a batch.
2. **zkVM execution.** The zkVM program takes as private inputs: (i) the prior batch’s commitments of the ledger \mathcal{L} and provenance log \mathcal{H} , (ii) the queued transitions, including inbox \mathcal{I} deposits and outbox \mathcal{O} withdrawals, and (iii) the proposed new commitments $(\mathcal{L}', \mathcal{H}')$. Inside the zkVM, the transition function is replayed deterministically: starting from the previous roots, applying all operations in sequence, updating balances and provenance entries, and checking consistency constraints.
3. **Proof generation.** The zkVM outputs a succinct proof π certifying that the claimed new roots $(\mathcal{L}', \mathcal{H}')$ are correct with respect to the given transition set.
4. **On-chain verification.** A minimalist verifier contract validates π , updates the recorded commitments, and enforces any pending outbox operations. This ensures that every external action has a verifiable provenance path while internal transfers remain private.

Compared to TEEs, zkVMs strengthen the verifiability model by eliminating hardware trust assumptions, but incur higher computational and financial costs due to proof generation and on-chain verification. Batching amortizes these costs, raising open research questions about optimal batch sizes, data availability strategies, and proof system selection. zkVM backends such as Risc0, SP1, or Plonky3 offer different tradeoffs in proof size, verification cost, and latency. Hybrid deployments are also possible: TEEs can serve as fast-path executors for intra-day operations, with zkVM proofs generated periodically to provide cryptographic audit trails. This hybrid pattern resembles optimistic rollups, where fast execution is combined with verifiable settlement.

Finally, this instantiation aligns closely with the Chainless stack architecture [31], which emphasizes the separation of execution, trust, bridging, and settlement. In this view, zkVM proofs naturally inhabit the execution and trust layers, Inbox/Outbox commitments integrate with bridging, and the verifier contract finalizes settlement. This layered design opens the door to modular, interoperable deployments of PASS across heterogeneous chains and domains. Full formal specification, performance benchmarking, and exploration of hybrid zkVM–TEE implementations remain important avenues for future work.

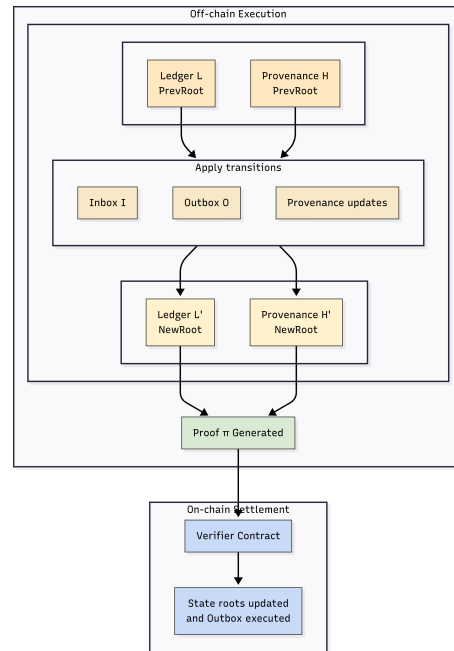


Fig. 6. PASS instantiation inside a zkVM: Inbox \mathcal{I} and Outbox \mathcal{O} transitions are applied to the Ledger \mathcal{L} and Provenance \mathcal{H} , producing new state commitments. The zkVM outputs a proof π , which the on-chain verifier checks before updating roots and enforcing outbox operations.