PASS: A Provenanced Access Subaccount System for Blockchain Wallets

Jay Yu
Stanford University
Department of Computer Science
Stanford, CA, USA
jyu01@stanford.edu

ABSTRACT

Blockchain wallets traditionally operate under a single-entity ownership model where possession of a private key grants complete control over all assets. This paradigm becomes limiting as blockchain applications evolve toward more complex access patterns, such as with Trusted Execution Environment (TEE) based private key encumbrance. We present the Provenanced Access Subaccount System (PASS), a novel wallet architecture that enables multiple entities to securely share control of a single blockchain address. Unlike existing approaches that rely on role-based or attribute-based access control, PASS introduces provenance-based access control, where authority over assets is determined by their origin and transfer history, and secret key access is mediated by a custodial Inbox-Outbox mechanism. Our design allows multi-entity control of a single wallet address with a strong notion of privacy, where internal transfers between subaccounts leave no on-chain trace, and a multi-user PASS wallet is indistinguishable from a regular user account. We outline the core design of PASS and formally verify key security properties, such as privacy and policy integrity, with Lean 4. We also implement a prototype in TypeScript with WalletConnect integration for Ethereum Virtual Machine (EVM) blockchains and with a TEE-encumbered secret key, demonstrating PASS's feasibility without any smart contract deployments. This work advances wallet security by combining the flexibility of multi-user access models with strong privacy guarantees and a formal verification approach.

1 INTRODUCTION

Cryptocurrency wallets conventionally rely on the assumption that each blockchain address is controlled by a single persistent entity holding the private key. This Single Entity Address-Ownership (SEAO) model [5] means that the private key `sk` functions as a root of trust: whoever holds `sk` has permanent and total access to all assets in the corresponding public address `pk`, analogous to a superuser in a traditional operating system. We can call this model a private-key based SEAO model. Fundamentally, many systems in the blockchain space, including token whitelists and blacklists [17], airdrop designs [8], and voting mechanisms for decentralized autonomous organizations (DAOs) [9], rely on this assumption.

Recently, however, the private-key-based SEAO model has been challenged by several emerging wallet design patterns. Social recovery wallets use trusted contacts or guardians to help restore access if a key is lost, enabling a multi-party trust model for key management. Embedded wallets and Wallet-as-a-Service solutions abstract private keys away from end-users, often linking wallets to user identities, such as email and OAuth, for easier recovery and shared control [25, 34]. Cryptocurrency wallet designs have also seen the growing use of Trusted Execution Environments [36]. This enables a technique called key encumbrance [16], where a user can encapsulate a cryptographic secret and lease out access to private keys. In particular, Liquefaction demonstrates how to use TEE-based key encumbrance to introduce a rich, multi-user wallet policy, demonstrating that multiple users can securely and privately share and trade presumably non-tradeable assets like Soul-Bound Tokens by attaching dynamic enforcement policies to the key's usage [5]. Such developments prompt a rethinking of "one address, one owner" — it is now feasible for multiple independent actors to jointly or selectively control assets under one address without direct key sharing.

Nevertheless, existing approaches have limitations in their choice of policy expression. Social recovery focuses on key *recovery* rather than fine-grained daily usage control. Liquefaction and many other custodial key management solutions typically rely on variations of Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) policies to partition permissions among users [5, 34]. These policy systems originate from traditional Operating System access models over filesystems and web services [11, 27, 41]. However, these systems face challenges in portability to blockchain contexts, as they assume an *endogenous* asset creation model where users generate and manage their own data, as files and objects. In blockchain contexts, however, asset creation is largely *exogenous*: assets, such as crytocurrency tokens, are created by external smart contracts according to set standards [38], and transferred permissionlessly into user wallets. Traditional RBAC and ABAC models struggle to elegantly handle this provenance of assets in a blockchain context, since control is not about who *created* an asset, but who *received* it from whom.

In this paper, we propose **Provenanced Access for Subaccount Security (PASS)**, a new wallet architecture that uses **provenance-based access control** to manage multiple private subaccounts within a single public wallet address. The core idea is to treat the origin and history of each asset as the basis for authorization. Instead of assigning static roles or attributes to users, PASS tracks how assets flow into and out of the wallet and enforces that each subaccount can only access assets that it has been provenance-authorized to hold. We introduce an Inbox-Outbox mechanism as the interface between the on-chain world and the off-chain subaccounts: all external funds enter through a designated Inbox, and all external withdrawals are funneled through an Outbox. This design ensures that every unit of value in any subaccount has a clear lineage to an on-chain deposit, and any transfer of value between subaccounts remains off-chain and private.

Our contributions are as follows:

- We design the PASS provenance-based access model for blockchain wallets which enforces access controls based on asset origin rather than on identities or roles. PASS preserves the privacy of internal transfers and leaves no on-chain trace of subaccount interactions.
- We adopt a formal verification approach, providing a model of PASS in the Lean 4 theorem prover and defining key security properties, including privacy, accessibility, and provenance integrity.
- We implement a functional prototype of PASS in TypeScript, integrating it with the WalletConnect protocol to support real dApp interactions. Our prototype demonstrates that PASS can be deployed without any new smart contracts or protocol changes making it immediately applicable to existing blockchain environments.

The remainder of this paper is organized as follows. Section 2 reviews related work on blockchain wallet design and security models to place PASS's design in a wider context of wallet design. Section 3 describes a high-level system model of PASS and details the Inbox-Outbox design for provenance enforcement. Section 4 outlines our formal verification approach and the properties proven about PASS in Lean 4. Section 5 presents our prototype implementation and explores potential use cases of PASS. Finally, Section 6 concludes and points to future directions.

2 BACKGROUND

Following increased digital asset adoption from both retail and institutional actors [4, 14, 28], there has been a significant increase in the variety of cryptocurrency wallet designs [20]. In this section, we provide an overview of the mainstream wallet designs within cryptocurrency space, with a focus on Ethereum Virtual Machine (EVM) compatible systems.

2.1 Externally Owned Accounts and Multisignature Wallets

In EVM systems, end users' assets are typically held in Externally Owned Accounts (EOAs), a type of Ethereum account controlled by a cryptographic keypair that gives permanent, unilateral power to whoever holds the private key [7]. Many high-profile cryptocurrency hacks have happened due to key theft and loss [15, 30]. Improvements on this basic model include hardware wallets and mnemonic backups to secure or recover keys [2], but these do not change the single-owner control assumption, nor the permanent access to wallet assets that secret key knowledge yields to an adversary.

Multisignature wallets (multisigs) such as Safe Smart Accounts require multiple keys to authorize transactions [26], distributing trust among parties, and has emerged as an industry standard for high-value transactions, including DAO treasuries [33], cross-chain bridges [40], and centralized exchanges [6]. While multisigs are effective for joint control, they require a quorum of signatures for each action and typically all co-owners are equal in authority. Furthermore, multisigs are typically deployed as public smart contracts, with a centralized dApp frontend that introduces security vulnerabilities, as evidenced by the recent hack of Bybit exchange [6]. In contrast, PASS allows more hierarchical and granular access to authorize specific routine operations to subaccounts, and does not require a smart contract deployment, making it harder to trace using on-chain data alone.

2.2 Smart Contract Wallets and Account Abstraction

Safe Smart Accounts are one example of Smart Contract Wallets that allow for more programmable access to user assets [23]. Argent, Loopring and other smart contract wallets support features like social recovery and daily spending limits by implementing custom logic in a contract controlling the funds [3, 19], which is not possible in a traditional EOA. Ethereum's recent account abstraction efforts with ERC-4337 enabling validation logic in a UserOperation contract [12], and EIP-7702 allowing EOAs to also call smart account logic defined in ERC-4337 [29]. These approaches allow flexible policies but they incur on-chain overhead and complexity. Moreover, because the transfer logic is publicly visible through a smart contract, this is not necessarily scalable to use cases such as institutional payroll.

PASS differs by implementing policy off-chain in a TEE-backed system, so it can enforce complex rules without consuming blockchain resources or revealing policies on-chain. PASS 's Inbox-Outbox design is conceptually similar to account abstraction's separation of user intent and execution, but here the separation is realized off-chain with a trusted hardware component.

2.3 Custodial Key Management and Policies

Institutional wallet services and custodial wallet APIs, like Fireblocks or Turnkey, provide policy engines to control how keys can be used, implementing Role-Based Access Control (RBAC) or Attribute Based Access Control (ABAC) policies with custom policy languages to ensure only certain users or conditions before asset transfers and signatures are allowed [13, 35]. These systems highlight the demand for programmable restrictions on key usage. PASS builds on this idea but shifts to a provenance-centric policy: rather than manually specifying allowed actions per user or asset, PASS automatically grants and restricts access based on how the asset was acquired into the system. This reduces the policy complexity for the wallet operator—the rules are inherent in the design, so there is no need for an additional policy language. Moreover, this provenance-centric design is also composable with additional rules and restrictions, such as whitelists and blacklists.

2.4 Liquefaction

Liquefaction is the most direct precursor to PASS [5]. It introduced the concept of building programmable wallets using *key encumbrance* and TEEs: the private key of a wallet is kept inside a TEE, and the TEE enforces multi-user access policies called *sub-policies*. In Liquefaction, each sub-policy can correspond to a share of assets or specific capabilities, like voting in a DAO, transferring a certain token. Liquefaction emphasizes privacy by ensuring that usage of shared credentials leaves minimal on-chain trace; multiple users can share an address without observers being able to tell. PASS extends the work of Liquefaction in two major directions.

Firstly, PASS uses formal verification to model Liquefaction and similar programmable wallet systems in the Lean 4 theorem prover to prove key security invariants. Secondly, PASS simplifies the access model presented in Liquefaction: instead of arbitrary policy trees and time-bounded sub-keys as in Liquefaction, PASS uses a straightforward provenance rule and two special subaccounts to manage all multi-user flows. In addition, Liquefaction's current implementation relies on smart contracts on Oasis [22], a specialized TEE-blockchain, to enforce policies and faciliate deposit and withdrawals. In contrast, PASS requires no new on-chain contracts; it operates entirely with a single existing EOA and off-chain logic. This makes PASS easier to deploy, at the cost of relying on a cloud-based TEE for correctness rather than enforcement on the Oasis blockchain.

2.5 Other Related Models

Our design is also informed by ideas from centralized exchanges and mixing services. Many centralized exchanges manage a single or a few omnibus blockchain addresses and internally tracks user balances [37]; deposits and withdrawals are batched, and internal transfers are off-chain. PASS can be seen as a more decentralized, versatile, and composable variant of this: users hold sub-accounts within an enclave rather than trusting an exchange operator, the enclave ensures no one can illegitimately affect others' balances, and users can use their PASS wallet assets on any dApp through a wallet connector. Privacy-focused mixers like Tornado Cash sever the link between deposit and withdrawal addresses [24]. PASS similarly hides the linkage between internal senders and recipients, though an external observer still knows the origin and final destination of funds. Unlike mixers, PASS maintains a full provenance chain internally, which is crucial for accountability and preventing misuse by sub-accounts beyond their entitled amounts, and allows for better regulatory compliance in the case of malicious actors.

3 SYSTEM MODEL AND DESIGN

PASS introduces a multi-user wallet system based on a single EOA address with keypair $(\mathsf{sk}, \mathsf{pk})$. The PASS Wallet contains multiple subaccounts, each identified with unique u_i, representing a distinct entity, such as an end user, authorized to use a portion of the wallet's assets. A subaccount u_i has authority over asset α if and only if it can successfully request a signature from the global sk from the PASS wallet. The actual sk is stored in a secure environment, such as a TEE, and inaccessible to any of the user subaccounts. The user's authorization model follows a provenance-based access policy, which we describe intuitively in this section, and provide a formal model for PASS in Section 4.

3.1 Core Principles of Provenance-based Access

The fundamental principle in PASS is that **access follows provenance**. A subaccount can only spend or transfer assets that have been explicitly received by or allocated to it, either from an external source or another subaccount. In other words, for any asset that a subaccount possesses, there must exist a chain of custody tracing back to a deposit from outside the PASS wallet. If an asset was never deposited or assigned to a subaccount, that subaccount cannot magically create or use it. This prevents over-spending and enforces a form of *consistency*: the sum of all subaccounts' balances of any asset equals the total asset balance in the on-chain wallet. In addition, the existence of this internal chain of custody enhances regulatory compliance, as it can be possible to publicize the internal transaction log.

Another principle is **off-chain privacy for internal transfers**. Any transfer of assets between subaccounts is an internal operation invisible to the blockchain, as no-onchain transaction occurs. Only when assets enter or leave the PASS wallet do we see any blockchain transactions. This ensures that the pattern of transfers between internal subaccounts is invisible to on-chain observers, enforcing a strong notion of privacy. It also means that internal transfer operations are free and instant, not subject to blockchain fees or delays. This makes PASS particularly desirable in institutional cases, where sensitive data such as payroll amounts should not be visible to an external observer.

Finally, PASS adopts a **first-come, first-served** allocation model. Whichever subaccount first receives an external asset or is sent an asset by another subaccount has the right to use up to that amount. Instead of keeping a internal policy state that introduces consistency complications, any external sender can send funds to the PASS wallet for a specific subaccount, and once received, those funds are under that subaccount's control. This open model aligns with the decentralized ethos, where assets are controlled by whoever their previous owner transfers, rather than pre-configured permissions, and better reflects the exogenous model of asset creation on blockchains as discussed in Section 1.

3.2 Inbox and Outbox Mechanism

To implement these principles, PASS designates two special subaccounts within the wallet enclave: the **Inbox** and the **Outbox**. These act as the gateways for funds moving into and out of the wallet, respectively. Figure 1 shows the overall PASS Wallet transaction flow.

3.2.1 *Inbox for Incoming Assets.*

The Inbox subaccount is the receiving gate for any external deposit. It has the main pk of the wallet of the main PASS Wallet address that others can send others to. In our implementation, external users who want to send funds to a PASS Wallet for a specific user or purpose will first "wrap" these assets by sending these assets to the Inbox. The Inbox thus temporarily holds unclaimed deposits. Then, Alice can log into the PASS Wallet system internally and claim these wrapped assets. Claiming is an internal operation that moves the asset from the Inbox record to Alice's subaccount balance. The asset is now "wrapped" as a PASS-internal asset, similar to the wrapping process for wETH on the Ethereum blockchain [10], and Alice can initiate a private internal transfer to Bob.

3.2.2 *Outbox for Outgoing Transactions.*

The Outbox subaccount is the only entity allowed to initiate actual blockchain transactions from the PASS wallet. When any subaccount wants to send assets out of the PASS system to an external address or perform an on-chain action such as interacting with a smart contract, it must do so via the Outbox. The process is as follows: the subaccount requests an *internal transfer* of some asset to the Outbox. This moves the specified amount of asset from the subaccount's balance to the

Outbox's holding queue. The Outbox then takes that request and uses the wallet's private key within the enclave to create and sign the actual blockchain transaction carrying out the request. For example, if Bob's subaccount wants to withdraw 50 USDC to his external EOA, Bob issues an internal transfer of 50 USDC to the Outbox. The Outbox receives 50 USDC, and then the enclave signs a transaction sending 50 USDC from the PASS wallet address to the recipient's external address on-chain. That signed transaction is then placed in a queue for broadcast to the blockchain by the enclave.

The Outbox employs a FIFO (first-in, first-out) queue for pending transactions, and increments a global nonce. This ensures that multiple subaccounts making withdrawals do not conflict with each other on the wallet's nonce or accidentally double-spend. The Outbox sequences transactions one at a time, incrementing the global account nonce in order. This design also mitigates the risk of "pre-signing" attacks: subaccounts never directly receive a signed transaction to execute later, which could be stolen or misused, but instead the Outbox immediately broadcasts the signed transactions. Thus, at no point is there a valid signed transaction lying around outside the enclave's control.

3.2.3 Internal Transfer Privacy. Internal transfers between two subaccounts within PASS, such as Alice's subaccount transfer of 70 USDC to Bob in Figure 1 do not involve the Outbox at all, as they do not require an on-chain transaction. The PASS wallet will check Alice's account logic to see if she has sufficient balances, and if she is permitted to send this internal transfer, will update the asset mapping ledger and add this internal transfer to the asset's provenance record. This transfer is only recorded in PASS 's internal ledger and provenance log but is never revealed on-chain. To any external observer, nothing semms to have happened; the PASS wallet's total USDC balance remains the same, and no blockchain transaction occurred. Only when Bob eventually withdraws or sends those USDC out of PASS via the Outbox will an on-chain trace appear, and it will simply show PASS wallet sending USDC out, with no indication that Alice was the original source internally.

A key benefit of this Internal Transfer model, combined with the specialized Inbox-Outbox architecture, is that it provides a unified logic for provenance and balance checks. A deposit is simply a claim taken from the inbox, while a withdrawal is an internal transfer to the outbox followed by a submission to the network. This design keeps internal movements invisible on-chain but still enforces correctness and security through the internal ledger and provenance log. At the same time, as discussed in Section 3.4 the authorization logic can be easily composable with additional RBAC or ABAC constraints, such as wallet whitelist or blacklists.

3.3 General Signable Messages and Non-Asset Actions

In addition to token transfers and asset movements, blockchain wallets are often used to sign arbitrary messages for authentication, such as logging into a dApp via a signature [31] or signing a DAO vote off-chain [32]. PASS extends the provenance model to such signature requests, which we call **General Signable Messages (GSMs)**. We treat a request to sign a message, which is not a transfer of an asset but a cryptographic action, as a kind of virtual asset

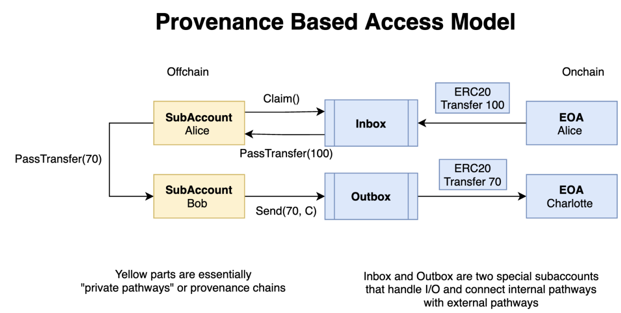

Figure 1: Overview of PASS Wallet architecture. External deposits go into the Inbox and are then claimed by internal subaccounts. Internal transfers move assets between subaccounts privately. Withdrawals are initiated by subaccounts to the Outbox, which then signs and sends out the on-chain transactions. This design ensures all on-chain activity has a clear provenance trail while internal transfers remain off-chain.

operation. In fact, we can model GSM signing rights as a special asset class that is segmented by the requester's identity or domain. Specifically, when a dApp asks a PASS wallet user to sign a message, such as to prove ownership of the address for a login, that request is routed to the appropriate subaccount inside PASS based on the domain of the requester.

By modeling signing authority as a transferrable internal asset, PASS allows subaccounts to hold or transfer "signing rights" for specific domains. Initially, the wallet creator may have signing access over all domains as an administrator. If, for example, the administrator wishes to delegate signing for `example.com` to Alice, they can internally transfer the appropriate GSM asset to Alice's subaccount. When `example.com` requests a signature, the request arrives in the Inbox (or directly through a secure channel like WalletConnect) as an action for Alice. Alice's subaccount can claim the request if it has the right domain-specific asset, and then use the Outbox to generate a signature with PASS 's private key. This signature is returned to the dApp without any on-chain transaction.

Thus, because the same policy engine that governs token transfers — inbox claims, ledger updates, and outbox transmissions — we unify the policy: even GSM signature requests have provenance, as the request came from a domain and was allocated to a subaccount. A subaccount cannot sign arbitrary messages for domains it hasn't been given access to, just as it cannot spend tokens it never received. The initial configuration of a PASS wallet might give the root user subaccount the ability to sign for any domain by default, and those rights can then be internally delegated. This design greatly enhances PASS 's versatility by enabling fine-grained access control, seamless policy enforcement, and a consistent user experience for both tokens and off-chain message signing.

3.4 Design Extensibility

A key advantage of PASS's provenance-based approach is its natural *composability* with other off-chain or on-chain access-control mechanisms. In particular, because PASS enforces that each subaccount only controls assets explicitly received via Inbox or internal transfers, we can combine this rule with additional restrictions (e.g., role-based or time-based constraints) without breaking provenance integrity. For instance, one might require a manager's signature off-chain before the Inbox will release certain deposits, or incorporate a time-lock policy on subaccounts so that assets claimed from the Inbox are only spendable after a vesting period. These auxiliary checks all occur *before* provenance updates in PASS, allowing the core system to remain unchanged. Conversely, PASS can also compose with other TEE-based approaches, such as Liquefaction, by embedding our Inbox–Outbox subaccount logic as a sub-policy within a more complex TEE-driven wallet. The unified wallet would inherit the same provenance integrity guarantees on any allocations delegated to PASS.

4 FORMAL VERIFICATION OF PASS

After presenting a high-level intuition and design of PASS in Section 3, we now consider a formal model of its core components and security invariants by defining PASS as a state machine in Lean4[1]. Lean4 is an interactive theorem prover and functional programming language with a rich dependent type system [18], making it well-suited for modeling and verifying complex security properties, such as in the AWS Cedar policy system [11]. 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.

4.1 A Formal Model of PASS

States: As shown in Figure 2, a PASS wallet can be represented as a tuple $S = (\mathsf{pk}, \mathsf{sk}, v, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$, with $(\mathsf{pk}, \mathsf{sk}, v)$ representing the public key, private key, and nonce typically found in an EOA wallet address, with pk and sk generated securely inside of the TEE. $(\mathcal{I}, \mathcal{O})$ represent the Inbox and Outbox subaccounts outlined in Section 3.2, and $(\mathcal{L}, \mathcal{H})$ are the internal ledger states for asset balances and provenance history respectively.

Transitions: These correspond to operations such as deposits, claims, transfers, withdrawals, and signing. Each operation updates the state according to the rules outlined in Section 3, and our Lean code ensures consistency by type-checking these transitions. For instance, $\mathsf{InternalTransfer}(\alpha, x, u_{\text{send}}, u_{\text{recv}})$ is a state transition

$$(\mathsf{pk}, \mathsf{sk}, v, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H}) \xrightarrow{\text{transfer}} (\mathsf{pk}, \mathsf{sk}, v, \mathcal{I}, \mathcal{O}, \mathcal{L}', \mathcal{H}')$$

defined by:

If $\mathcal{L}[u_{\text{send}}][\alpha] < x$ or $\neg\mathsf{checkAllow}(u_{\text{send}}, \alpha, x, \mathcal{H})$, return $\bot$
$\mathcal{L}'[u_{\text{send}}][\alpha] \leftarrow \mathcal{L}[u_{\text{send}}][\alpha] - x$
$\mathcal{L}'[u_{\text{recv}}][\alpha] \leftarrow \mathcal{L}[u_{\text{recv}}][\alpha] + x$
$\mathcal{H}' \leftarrow \mathcal{H} \,\big|\, (op = \mathtt{transfer}, \alpha, x, u_{\text{send}}, u_{\text{recv}})$
return `success`

[1]Our Lean 4 formal model can be found here: https://github.com/jayyu23/pass-lean4-proofs

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

Global State:
pk: public address (EOA)
sk: private key (in TEE)
v: global nonce for on-chain transactions
$\mathcal{I}$ (Inbox): set of unclaimed deposits (α : asset, x : amount, m : depositId)
$\mathcal{O}$ (Outbox): FIFO queue $(\alpha, x, extDst, v)$
$\mathcal{L}$ (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 $(creator)$:
$\mathcal{I} \leftarrow \emptyset,\ \mathcal{O} \leftarrow \emptyset,\ \mathcal{L} \leftarrow \{\};\ v \leftarrow 0$
$(pk,\ sk) \leftarrow$ TEE_KeyGen()
Return new $\mathcal{W}$ with $S = (pk, sk, v, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ 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 $\bot$
Remove (α, x, m) from $\mathcal{I}$
$\mathcal{L}[u][\alpha] \leftarrow \mathcal{L}[u][\alpha] + x$
$\mathcal{H} \leftarrow \mathcal{H} \mid (op = \mathtt{claim}, \alpha, x, depositId, u)$
Return success
CheckAllow $(u_{\text{send}}, \alpha, x, \mathcal{H})$:
If $\mathcal{H}$.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 $\neg\mathsf{checkAllow}(u_{\text{send}}, \alpha, x, \mathcal{H})$, return $\bot$
$\mathcal{L}[u_{\text{send}}][\alpha] \leftarrow \mathcal{L}[u_{\text{send}}][\alpha] - x$
$\mathcal{L}[u_{\text{recv}}][\alpha] \leftarrow \mathcal{L}[u_{\text{recv}}][\alpha] + x$
$\mathcal{H} \leftarrow \mathcal{H} \,\big|\, (op = \mathtt{transfer}, \alpha, x, u_{\text{send}}, u_{\text{recv}})$
Return success
Withdraw $(\alpha, x, u_{\text{send}}, extDst)$:
If $\mathcal{L}[u_{\text{send}}][\alpha] < x$ or $\neg\mathsf{checkAllow}(u_{\text{send}}, \alpha, x, \mathcal{H})$, return $\bot$
$\mathcal{L}[u_{\text{send}}][\alpha] \leftarrow \mathcal{L}[u_{\text{send}}][\alpha] - x$
$\mathcal{O} \leftarrow \mathcal{O} \cup \{(\alpha, x, extDst, v)\}$
$\mathcal{H} \leftarrow \mathcal{H} \mid (op = \mathtt{withdraw}, \alpha, x, u_{\text{send}}, extDst)$
Return success
SignGSM (dom, msg, u):
If u lacks sign-right for (dom, msg), return $\bot$
$\sigma \leftarrow (sk, msg)$
Return σ
(Outbox Processing) (periodic or on demand):
If $\mathcal{O} \neq \emptyset$, dequeue $(\alpha, x, extDst, v)$
Build tx τ: transfer x of α to *extDst*, nonce = v
$\sigma \leftarrow (sk, \tau)$; broadcast (τ, σ)
$v \leftarrow v + 1$
Return `broadcasted` + txHash

Figure 2: Formal Model of PASS wallet functionality, including CreatePassWallet for initializing a fresh instance. On creation, *pk* is set to a chosen EOA address *eoa*, 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*.

where checkAllow is a function that queries the provenance of an asset α from the provenance history ledger $\mathcal{H}$. The checkAllow function can optionally be extended with custom rules. We see here in the InternalTransfer function that this state transition preserves wallet consistency while updating only the ledger and history components.

4.2 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 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:

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

where:

pk is the externally owned account (EOA) address,
$O = (T, v)$ is the external view of the *outbox*, where T is a list of enqueued transactions and v 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 externalTrace$(\cdot)$.

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

(1) $\mathsf{pk}_1 = \mathsf{pk}_2$,
(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 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 1 (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 externalTrace$(\mathcal{W}_1)$ and 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 1.1 (Indistinguishability from EOA). *Let $\mathcal{W}_{\text{pass}}$ 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 $\mathcal{W}_{\text{eoa}}$. Then*

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

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

Proof. By assumption, the only potential differences between $\mathcal{W}_{\text{pass}}$ and $\mathcal{W}_{\text{eoa}}$ 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 $\mathcal{W}_{\text{eoa}}$. □

4.3 Property: Asset Accessibility

In addition to keeping internal transactions private, PASS should ensure that for every asset in the system, there exists some subaccount 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 2 (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:

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

(2) The sum of all accessible balances equals the total balance of the asset:

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

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:

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

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

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

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 3 (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:

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

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. □

4.4 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 4 (Provenance Integrity). *For any PASS account $S = (\mathsf{pk}, \mathsf{sk}, v, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ 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,*

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

where U is the set of subaccounts in the system, and $\texttt{externalDeposits}(\alpha)$ *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 $(\mathsf{pk}, \mathsf{sk}, v, \mathcal{I}, \mathcal{O}, \mathcal{L}, \mathcal{H})$ as in Figure 2, 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:

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

Hence,

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

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

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

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

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 $\texttt{externalDeposits}(\alpha)$ (but not yet allocated to any subaccount), the total subaccount balances remain $\leq \texttt{externalDeposits}(\alpha)$.

(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 $\texttt{externalDeposits}(\alpha)$ remains unchanged (assets have simply exited the wallet), the inequality still holds.

Thus, by induction over all possible operations,

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

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. □

5 IMPLEMENTATION AND APPLICATIONS

Having presented the PASS system design in Section 3 and a formal model of its key security invariants in Section 4, we now discuss some of our prototype implementation of PASS, the various applications that this system could be used for, as well as some limitations of PASS in its current form.

5.1 Prototype Implementation

We built a prototype of the PASS system to demonstrate its practicality. The prototype consists of:

- **Secure enclave service** that implements a secure $(\mathsf{pk}, \mathsf{sk})$ keygen, stores the sk securely, and emits signed transactions for PASS.
- **TypeScript frontend** resembling a multisig wallet interface that users interact with to view balances, initiate transfers, and approve outgoing transactions.
- **Integration with WalletConnect**, allowing the PASS wallet to connect to decentralized applications (dApps) as if it were a standard wallet for signing transactions or messages.

The enclave service exposes an API for the front-end, such as '`/keygen`' for a new EOA keypair, and '`sign`' for transaction signing. In our current version, we simulate an enclave key management service using a separate server in Python Flask. In a production deployment, we envision using an AWS Nitro Enclave or Intel SGX to run this service so that the private key is protected and operations are attested. This would provide the necessary trust that even the server operator cannot extract the key or violate the PASS policy.

One notable aspect of our implementation is that it avoids any on-chain smart contracts. Unlike Liquefaction [5], which required deploying a custom wallet contract and various policy contracts, PASS operates using a normal EOA. The Inbox and Outbox are conceptual divisions within the enclave; externally, deposits are sent to the same blockchain address that also sends withdrawals. This means our prototype can immediately work with existing blockchain infrastructure: to deposit, someone just sends tokens to the PASS address; to withdraw, the PASS enclave simply creates a transaction from that address. No new contract deployment or gas cost beyond standard transfers is involved. The trade-off is that our logic is off-chain, so the security relies on the correctness of the enclave. We attempt to mitigate this via formal verification of the key wallet actions, as discussed in Section 4.

The current implementation also integrates a WalletConnect handler for signing and transactions, enabling the PASS wallet to sign General Signable Messages (GSMs) and emit standard transactions. This integration allows decentralized applications to interact with PASS as if it were a typical wallet, while all signing requests for both external asset transfers and for GSMs are securely forwarded to the enclave service. Consequently, GSMs are signed under the same formal security guarantees as traditional transactions, ensuring both seamless interoperability with dApps and adherence to the security invariants established in our Lean4 model.

Our prototype demonstrates that PASS can be integrated seamlessly: to dApps, the PASS wallet is just another Ethereum wallet, communicating over a WalletConnect as usual hook. Internally, the user experiences a multi-account interface somewhat analogous to an email inbox managing different accounts. The overhead introduced by PASS is minimal in user experience terms; the main addition is that a user must occasionally perform an extra step of claiming deposits from the Inbox and approving Outbox operations. We have open-sourced the prototype on GitHub for the community to experiment with. The repository includes instructions to run a local enclave server and web interface. [2]

5.2 Potential Use Cases

Organizational Wallets and Institutional Custody. An organization, such as a company, university, or DAO, could use PASS to manage a single blockchain address, denoted with a public ENS, such as `stanford.eth`, while delegating control to multiple individuals internally. For example, a university could maintain `stanford.eth` as the main address, and issue subaccounts `alice@stanford.eth`, `bob@stanford.eth` for different departments or team members. Each person gets a subaccount with certain assets allocated, such as budgets, or certain sign-in rights to certain websites. They can transact internally for payroll and other internal accounting without any fees or public logging, and when needed, withdraw funds to an outside vendor or collaborator through the Outbox. The provenance log ensures that every outgoing fund can be traced to which internal subaccount and thus which department or person it came from, aiding accountability. At the same time, external parties only ever interact with one official address, simplifying public interactions.

Privacy-Preserving Protocols like Tornado Cash. Because internal transfers in PASS are off-chain, the PASS wallet can serve as a mini "mixer". Suppose a user wants to send funds to another user privately. If both use the same PASS wallet or a network of PASS wallets, they could achieve a level of privacy similar to Tornado Cash by using subaccounts [24]. For instance, Alice deposits funds into PASS and then internally transfers to Bob's subaccount. Bob withdraws from Outbox to a new external address. On-chain, one sees Alice deposit to the PASS wallet and the PASS wallet withdraw to Bob's address, but it is not evident that those are linked, especially if multiple deposits and withdrawals happen and amounts are broken up. While not the primary goal of PASS, this capability could be a side benefit. In fact, PASS could implement mixing functionalities entirely within one enclave with multiple users depositing and withdrawing, with the enclave shuffling ownership internally. Moreover, the internal provenance log that PASS preserves allows for better compliance possibilities.

Delegating to AI or Smart Agents. As AI agents start to manage crypto assets, such as for payments and trading [39], there is a growing need to create fine-grained access policies such to specify what assets in a wallet an AI agent should have access to. With PASS, an AI agent could be given access to a subaccount that only holds a limited amount of funds, has access to only certain asset types, or is only allowed to sign messages for a restricted set of whitelisted domains. The AI agent can operate freely within those bounds, perhaps executing trades or interacting with protocols via signatures, but cannot exceed its allotted assets or misuse other assets in the wallet. If the AI is compromised or behaves erratically, the damage is capped to its subaccount. This is much safer than giving an AI the keys to an entire wallet. We can even rotate or revoke the subaccount by transferring remaining assets out and not allowing further deposits to that agent. This aligns with the idea of creating *sub-keys* or limited scopes of authority for different automated agents [34].

Integration with Off-chain Scaling Solutions. The concept of segregating internal transfers from on-chain actions is analogous to how rollups and sidechains work, as they have off-chain transactions batched into occasional on-chain commitments [1]. PASS could potentially integrate with Layer-2 networks by treating a whole sidechain or rollup as a "subaccount" or by using PASS 's provenance model to bridge assets. For example, bridging funds into a rollup could be managed via an Inbox-like mechanism, and withdrawing via an Outbox, with PASS ensuring that assets are not double-spent across L1 and L2. It may also be possible to use PASS formal verification model to model a rollup's internal accounts to provide a unified view of assets across layers, or allow a user to move assets between L2s through an enclave without trusting a centralized exchange. This is an area for future exploration.

5.3 Limitations and Security Considerations

Trust in the Enclave. PASS 's current implementation assumes underlying security guarantees for the TEE containing the private key. If the TEE is compromised, an attacker could obtain the private key or manipulate the internal state. The security guarantee of TEEs is a shared assumption with systems like Liquefaction. Enclave technologies (SGX, Keystone, Nitro) have known vulnerabilities, though they continue to improve, and underlying improvements in TEE technologies can reinforce the security of PASS. A hardware compromise is outside the scope of our current threat model, as we assume a secure enclave with remote attestation. However, it means PASS, in practice, might be better suited for settings where such trust assumptions are acceptable, such as in enterprise and consortium settings rather than trust-minimized DeFi contexts. In a fully trustless environment, one would prefer on-chain enforcement (at the cost of revealing policies). It's a trade-off: PASS chooses confidentiality and flexibility over on-chain verifiability.

[2] Our demo application is here: https://github.com/jayyu23/pass-wallet-app

Extending our Formal Verification Model. Our existing Formal Verification model has focused on proving invariants at the application layer of PASS, especially with regards to how blockchain assets flow within the application system. One current limitation of our Formal Verification system is that we do not model the underlying TEE architecture and its security guarantees. Future work could extend our Lean 4 formal verification model to underlying hardware assumption of the underlying Trusted Execution Environment and popular TEE architectures. Similarly, in our current formal verification system we present only a very simplified model of the Ethereum Virtual Machine (EVM) and the global blockchain state. An area of future work could be making our proof systems more composable with more sophisticated formal models of the EVM, such as Nethermind's Clear EVM project, which is also written in Lean 4 [21].

Performance and Scalability. Since the enclave processes every action, it could become a bottleneck if a PASS wallet is used by many subaccounts heavily. In a sense, the enclave is like a mini-private blockchain executor. Modern TEEs can handle significant throughput [42], and our design, with a simple FIFO queue for outgoing transactions, should scale reasonably for dozens of users. But if someone tried to run an exchange with thousands of users on one PASS wallet, they might need to shard across multiple enclaves or addresses. Additionally, monitoring the blockchain for Inbox deposits could become complex if there are many deposits; indexing solutions or event logs might be needed. A related area of future exploration for performance and scalability is in considering how multiple PASS wallets, perhaps each representing a different organization, could transact with each other privately and update a shared state.

Despite these limitations, we believe PASS is a compelling step towards more versatile wallet security. Many of these challenges are surmountable with engineering effort and careful user-centric design.

6 CONCLUSION

We presented PASS, a Provenanced Access Subaccount System, as a novel approach to blockchain wallet design. PASS shifts the paradigm from identity-or role-based access control to provenance-based control, aligning with the unique nature of asset flows in blockchains. By introducing the Inbox-Outbox mechanism, PASS ensures that all internal asset movements are invisible on-chain yet internally verifiable, and that no assets can be spent without a clear lineage.

Our work builds upon ideas from prior multi-user wallet models like Liquefaction while removing the need for special smart contracts and embracing a simpler policy rooted in "you can use what you receive." The formal modeling of PASS in Lean 4 provides a foundation for high-assurance implementations, and our prototype demonstrates that the concept can be realized with mainstream blockchain development tools.

Looking forward, there are several avenues for future work. Completing the formal proofs and perhaps extracting a verified reference implementation from the Lean model would increase confidence in PASS for high-value deployments. Integrating PASS with actual TEE hardware, such as deploying on an AWS Nitro enclave, and measuring performance will be important for practical adoption. We also plan to explore richer policy extensions to PASS: for instance, adding time-based restrictions, where a sub-account's access only valid until a certain time or rate limits, which could be integrated without breaking the provenance principle. Another interesting direction is interoperability: multiple PASS enclaves could potentially pass assets amongst each other, forming a network of enclaves, which could enable scalable, geographically distributed custody solutions.

In conclusion, PASS offers a new way to think about wallet security that we hope will inspire further research and development. As the cryptocurrency ecosystem grows and more diverse users and use cases emerge, solutions like PASS can help bridge the gap between strict self-custody and flexible shared access, all while maintaining strong security guarantees.

REFERENCES

[1] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Sarah Meiklejohn, and George Danezis. 2022. Blockchain Scaling Using Rollups: A Comprehensive Survey. *IEEE Communications Surveys Tutorials* 24, 2 (2022), 1393–1414. https://doi.org/10.1109/COMST.2022.3146163

[2] Myrto Arapinis, Andriana Gkaniatsou, Dimitris Karakostas, and Aggelos Kiayias. 2019. A Formal Treatment of Hardware Wallets. In *Financial Cryptography and Data Security*, Ian Goldberg and Tyler Moore (Eds.). Springer International Publishing, Cham, 426–445.

[3] Argent. 2023. *Build with Argent.* https://docs.argent.xyz/

[4] Raphael Auer, Marc Farag, Ulf Lewrick, Lovrenc Orazem, and Markus Zoss. 2023. *Banking in the Shadow of Bitcoin? The Institutional Adoption of Cryptocurrencies.* CESifo Working Paper 10355. Munich. https://hdl.handle.net/10419/271999

[5] James Austgen, Andrés Fábrega, Mahimna Kelkar, Dani Vilardell, Sarah Allen, Kushal Babel, Jay Yu, and Ari Juels. 2024. Liquefaction: Privately Liquefying Blockchain Assets. *arXiv e-prints* 2412.02634 (2024). arXiv:2412.02634 [cs.CR].

[6] Nina Bambysheva. 2025. 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/

[7] Vitalik Buterin. 2014. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. Ethereum whitepaper, https://ethereum.org/en/whitepaper/. Accessed: 2024-10-16.

[8] Coinbase. [n. d.]. What is a crypto airdrop? https://www.coinbase.com/learn/crypto-basics/what-is-a-crypto-airdrop. Accessed: 2024-11-14.

[9] Ethereum.org Contributors. [n. d.]. Decentralized autonomous organizations (DAOs). https://ethereum.org/en/dao/. Accessed: 2024-11-13.

[10] Ivan Cryptoslav. 2022. What Is Wrapped Ethereum (WETH)? *CoinMarketCap* (2022). https://coinmarketcap.com/academy/article/what-is-wrapped-ethereum-weth

[11] Joseph W. Cutler, Craig Disselkoen, Aaron Eline, Shaobo He, Kyle Headley, Michael Hicks, Kesha Hietala, Eleftherios Ioannidis, John Kastner, Anwar Mamat, Darin McAdams, Matt McCutchen, Neha Rungta, Emina Torlak, and Andrew M. Wells. 2024. Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization. *arXiv preprint arXiv:2403.04651* (2024). https://arxiv.org/pdf/2403.04651

[12] ERC-4337 Documentation. 2023. *UserOperation – ERC-4337 Documentation.* https://www.erc4337.io/docs/understanding-ERC-4337/user-operation

[13] Fireblocks. 2025. *Governance and Policy Engine.* https://www.fireblocks.com/platforms/governance-and-policy-engine/

[14] Goldman Sachs Research. 2021. Crypto: A New Asset Class? (2021). https://www.goldmansachs.com/insights/top-of-mind/crypto-a-new-asset-class

[15] Stan Higgins. 2015. Details of $5 Million Bitstamp Hack Revealed. https://www.coindesk.com/markets/2015/07/01/details-of-5-million-bitstamp-hack-revealed/

[16] Mahimna Kelkar, Kushal Babel, Philip Daian, James Austgen, Vitalik Buterin, and Ari Juels. 2024. Complete knowledge: Preventing encumbrance of cryptographic secrets. In *ACM CCS*.

[17] DaeYoub Kim and Jihoon Lee. 2020. Blacklist vs. whitelist-based ransomware solutions. *IEEE Consumer Electronics Magazine* 9, 3 (2020), 22–28.

[18] Lean Community. 2025. *Lean Documentation.* https://lean-lang.org/documentation/

[19] Loopring. 2023. *Loopring Smart Wallet Documentation.* https://docs-wallet.loopring.io/

[20] Easwar Vivek Mangipudi, Umang Desai, Mohammad Minaei, Mainack Mondal, and Aniket Kate. 2023. 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. 3153–3167. https://doi.org/10.1145/3576915.3623218

[21] Nethermind. 2024. Clear–Prove Anything About Your Solidity Smart Contracts. https://www.nethermind.io/blog/clear-prove-anything-about-your-solidity-smart-contracts

[22] Oasis Labs. 2024. Oasis Protocol: Privacy-Enabled Blockchain Platform. https://oasisprotocol.org/. https://oasisprotocol.org/ Accessed: 2024-10-06.

[23] Dilip Kumar Patairya. 2024. What Are Smart Contract Wallets? *Cointelegraph* (2024). https://cointelegraph.com/explained/what-are-smart-contract-wallets

[24] Alexey Pertsev, Roman Semenov, and Roman Storm. 2019. Tornado Cash Privacy Solution Version 1.4. https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf

[25] Privy. 2025. *Privy Documentation*. https://docs.privy.io/

[26] Safe. 2025. *How do Safe Smart Accounts work?* https://docs.safe.global/advanced/smart-account-concepts

[27] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. [n. d.]. Role-Based Access Control Models. *IEEE Computer* 29, 2 ([n. d.]).

[28] Security.org Research Team. 2025. 2025 Cryptocurrency Adoption and Consumer Sentiment Report. (2025). https://www.security.org/digital-security/cryptocurrency-annual-consumer-report/

[29] Valentin Seehausen. 2024. 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

[30] SelfKey Foundation. 2020. A Comprehensive List of Cryptocurrency Exchange Hacks. https://selfkey.org/list-of-cryptocurrency-exchange-hacks/

[31] Sign-In with Ethereum Documentation. 2023. *EIP-4361: Sign-In with Ethereum*. https://docs.login.xyz/general-information/siwe-overview/eip-4361

[32] Snapshot Documentation. 2023. *Voting on Snapshot*. https://docs.snapshot.box/user-guides/voting/vote

[33] Tally. 2025. *Gnosis Safe Overview*. https://docs.tally.xyz/set-up-and-technical-documentation/using-governor-with-gnosis-safe/gnosis-safe

[34] Turnkey. 2025. *Integrating Embedded Wallets*. https://docs.turnkey.com/embedded-wallets/sub-organizations-as-wallets

[35] Turnkey. 2025. *Policy Quickstart*. https://docs.turnkey.com/concepts/policies/quickstart

[36] Turnkey. 2025. *Secure Enclaves*. https://docs.turnkey.com/security/secure-enclaves

[37] Venly. 2025. *Omnibus vs Segregated Wallets in Crypto*. https://docs.venly.io/docs/omnibus-vs-segregated-wallets-in-crypto

[38] Fabian Vogelsteller and Vitalik Buterin. 2015. ERC-20 Token Standard. https://ethereum.org/en/developers/docs/standards/tokens/erc-20/

[39] Shaw Walters, Sam Gao, Shakker Nerd, Feng Da, Warren Williams, Ting-Chien Meng, Hunter Han, Frank He, Allen Zhang, Ming Wu, Timothy Shen, Maxwell Hu, and Jerry Yan. 2025. Eliza: A Web3 Friendly AI Agent Operating System. *arXiv preprint arXiv:2501.06781* (2025). https://arxiv.org/pdf/2501.06781v1

[40] Wormhole Foundation. 2025. *Guardians*. https://wormhole.com/docs/learn/infrastructure/guardians/

[41] Eric Yuan and Jin Tong. 2005. Attributed Based Access Control (ABAC) for Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS)*. IEEE, 561–569. https://doi.org/10.1109/ICWS.2005.25

[42] Jianwei Zhu, Hang Yin, Peng Deng, Aline Almeida, and Shunfan Zhou. 2024. Confidential Computing on NVIDIA Hopper GPUs: A Performance Benchmark Study. *arXiv preprint arXiv:2409.03992* (2024). https://arxiv.org/pdf/2409.03992