

Liquefaction: Privately Liquefying Blockchain Assets

James Austgen <i>Cornell Tech</i> james@cs.cornell.edu	Andrés Fábrega <i>Cornell University</i> andresfg@cs.cornell.edu	Mahimna Kelkar <i>Cornell Tech</i> mahimna@cs.cornell.edu	Dani Vilardell <i>Cornell Tech</i> dv296@cornell.edu
Sarah Allen <i>IC3, Flashbots</i> sarahallen@cornell.edu	Kushal Babel <i>Cornell Tech, Monad Labs</i> babel@cs.cornell.edu	Jay Yu <i>Stanford University</i> jyu01@stanford.edu	Ari Juels <i>Cornell Tech</i> juels@cornell.edu

Abstract—Inherent in the world of cryptocurrency systems and their security models is the notion that private keys—and thus assets—are controlled by individuals or individual entities.

We present *Liquefaction*, a wallet platform that demonstrates the dangerous fragility of this foundational assumption by systemically breaking it. Liquefaction uses trusted execution environments (TEEs) to *encumber* private keys, i.e., attach rich, multi-user policies to their use. In this way, it enables the cryptocurrency credentials and assets of a single end-user address to be freely rented, shared, or pooled. It accomplishes these things *privately*, with no direct on-chain traces.

Liquefaction demonstrates the sweeping consequences of TEE-based key encumbrance for the cryptocurrency landscape. Liquefaction can undermine the security and economic models of many applications and resources, such as locked tokens, DAO voting, airdrops, loyalty points, soulbound tokens, and quadratic voting. It can do so with no on-chain and minimal off-chain visibility. Conversely, we also discuss beneficial applications of Liquefaction, such as privacy-preserving, cost-efficient DAOs and a countermeasure to dusting attacks. Importantly, we describe an existing TEE-based tool that applications can use as a countermeasure to Liquefaction.

Our work prompts a wholesale rethinking of existing models and enforcement of key and asset ownership in the cryptocurrency ecosystem.

1. Introduction

In blockchain systems, private keys represent the ownership of assets. Each private key has full control over funds held in its associated public address. Thus, to protect against theft, participants should keep their private keys secret from others. From this comes the widely held assumption that private keys are held by individual entities. We call this the *Single-Entity Address-Ownership* (SEAO) assumption.

The SEAO assumption. In the design and culture of blockchain interfaces, tools, and media, the SEAO assumption is pervasive. For example, address blacklists and whitelists used to categorize blockchain actors [43], [58] rely on the SEAO assumption. If a single address could be

used by different entities safely—i.e., without their stealing funds from one another—then the SEAO assumption would be in jeopardy. In this case, whitelists wouldn’t work. An address belonging to a whitelisted individual could be used by, e.g., rented out to, an individual not on the whitelist.

Many cryptocurrency protocols and systems in fact rely critically on the SEAO assumption to achieve their security and economic models. This is true when they associate an asset, reward, or privilege non-transferably with an address, as illustrated by the following examples.

Example 1 (Locked tokens). *In airdrops or token-generation events, tokens are often distributed in a locked or unvested state [8]. Ownership is assigned to specific wallet addresses and programmed to be non-transferable until the completion of a predefined lockup or vesting period.*

Example 2 (Loyalty rewards). *Some decentralized cryptocurrency exchanges (DEXes) offer loyalty rewards or trading discounts to frequent traders [25] or token holders [1]. They associate rewards with particular high-activity addresses in order to incentivize use of the DEX.*

Example 3 (SBTs). *Soulbound tokens (SBTs) are special non-fungible tokens that serve as digital identity documents for individual human users [32]. An SBT is intended to be “bound to a user’s soul,” that is, assigned permanently to a particular address and held forever by a single individual.*

Other blockchain systems depend critically on the SEAO assumption for goals that include voting integrity—i.e., preventing vote-buying—enforcing models of token economics, ensuring transaction traceability, etc.

1.1. Liquefaction

The goal of this work is to demonstrate the inherent fragility of the SEAO assumption and its sweeping repercussions, both destructive and constructive. To this end, we introduce *Liquefaction*, a privacy-preserving cryptocurrency wallet platform that *breaks the SEAO assumption*.

Broadly speaking, Liquefaction enables the transfer or *liquefaction* of assets or privileges that are meant to be

nontransferable / illiquid. Recalling Example 1 above, a user with locked tokens can use Liquefaction to transfer ownership of the tokens prior to the end of their lockup or vesting period. In Example 2 above, a user receiving loyalty discounts from a cryptocurrency exchange could use Liquefaction to safely rent out her address to other users, freely selling the DEX’s discounts and subverting its loyalty system. Similarly, Liquefaction would enable users to transfer access to their SBTs in Example 3, completely breaking the defining “soulbound” property of SBTs.

To demonstrate the extensive implications of breaking the SEAO assumption, Liquefaction enables blockchain addresses to be managed according to a rich set of access-control policies. These policies permit flexible renting, sharing, pooling, and partitioning of blockchain assets and privileges among a group of entities. They also include support for complex forms of delegation. Liquefaction operates within a blockchain-compatible trust model: Policies are enforced with high assurance by the Liquefaction platform, not by policy creators or other users.

Liquefaction also enforces a strong notion of *privacy*. Liquefaction produces no direct on-chain indication of which addresses it controls or what transactions it has generated. Even the creator of an address, having delegated control of resources associated with it, gains no private information about the activity of delegates.

In short, Liquefaction can *privately liquefy* a wide range of assets, even if they’re meant to be illiquid. Liquefaction demonstrates the fragility of the SEAO assumption and how this fragility can have broad repercussions for the blockchain ecosystem. Fortunately, by opening up new models of flexible, confidential asset ownership, Liquefaction enables not just attacks, but also beneficial new blockchain applications and markets. Critically, as we also explain, there is a practical existing countermeasure that enables blockchain applications to avoid exposure to Liquefaction if desired.

Key encumbrance. Liquefaction breaks the SEAO assumption using a notion called *key encumbrance* [23], [41].

An encumbered secret key sk is one that is neither known nor managed directly by a user or administrator. Instead, sk is generated by an application that enforces an access-control policy over the key’s full lifecycle. Liquefaction applies such a policy to the secret keys associated with addresses it controls. (For example, if the owner of an address A with secret key sk rents A out, Liquefaction can bar the owner from accessing sk during the rental period.)

Proposed key-encumbrance applications [41], [53] typically rely on the use of *trusted execution environments* (TEEs) to enforce access-control policies on keys. This is our approach with Liquefaction.

1.2. Liquefaction Design and Applications

We have implemented a demonstration version of Liquefaction. Its backend runs in Oasis Sapphire, a TEE-based blockchain, but Liquefaction can be used as a wallet for any blockchain, e.g., Ethereum. Liquefaction is robust to liveness failures or service discontinuation in Oasis Sapphire.

We report on the details of our design and implementation of Liquefaction and explore its practical ramifications.

Liquefaction access-control policies. A particularly important element of Liquefaction’s design is its system of *access-control policies*, which dictate how individual players can use and delegate access to an encumbered key sk . To ensure broad usability, Liquefaction wallets work with arbitrary smart contracts, whose logic and persistent effects are computationally infeasible to reason over. It is therefore challenging to ensure that Liquefaction policies are both expressive—i.e., support a broad range of applications—and sound—i.e., cannot be subverted through attacks. These properties need, moreover, to be maintained as a policy changes over time to handle complex delegation histories. To address these challenges, we introduce a formal delegation model for encumbered keys that is designed to be conceptually simple, yet expressive and prevent policy conflicts.

Adversarial ecosystem implications. In breaking the SEAO assumption, Liquefaction erodes the security models associated with a host of applications. As another key contribution, our work is the first to enumerate these applications broadly and in many cases is the first to identify them. They include quadratic voting, soulbound tokens (SBTs), airdrops, and blockchain loyalty points. Liquefaction can also obfuscate transaction histories, enable stealthy wash trading, erode the fidelity of on-chain cryptocurrency analytics, and corrupt reputation-dependent systems. Liquefaction can also facilitate certain types of fraud, such as faking theft of funds by a known bad actor.

For concrete study and demonstration, we focus on blockchain voting. We fully realize and report the details of a Dark DAO for voter bribery as theorized in [23], as well as a new Dark DAO “lite” variant realized with Liquefaction but allowing user participation with an ordinary (non-Liquefaction) wallet.

We emphasize that whether or not particular applications of Liquefaction constitute attacks is a subjective matter—as evidenced by, e.g., public bribery markets for blockchain voting [28].

Beneficial new applications. Liquefying assets means creating new financial instruments, capabilities, and markets. Liquefaction also enables clearly beneficial applications.

One example is a mitigation against dusting attacks, where funds are sent to an address to taint its funds. Liquefaction permits provable sequestering or blackholing of such funds. Another example is privacy-preserving DAOs. Such DAOs can circumvent detrimental leakage of intelligence—as famously exemplified by the failure of ConstitutionDAO to win a copy of the U.S. Constitution at auction due to public fundraising [76].

1.3. Why Explore Liquefaction?

The SEAO assumption is increasingly subject to erosion. NFT fractionalization [3], liquid-staking [33], and re-staking [27] have created popular new forms of liquidity for NFTs and cryptocurrency. The liquefying of governance

rights in vote-buying marketplaces [51] has shown how market forces encourage adversarial uses of liquidity.

Some forms of liquidity can be achieved transparently using smart contracts. But as our many examples in this work show, smart contracts are of limited utility in liquefying assets, as privacy is often a critical requirement. For instance, soulbound tokens (see Example 3) can in principle be shared through smart contracts—but that is exactly why they are typically sent only to end-user (non-contract) addresses.

At the same time, however, attestation-capable TEEs are pervading new computing platforms [63], [67] and blockchain systems [6], [64], making private key encumbrance increasingly easy to realize. An inevitable outcome of this trend is that users will privately liquefy assets. Liquefaction sheds light on the full scope of the inevitable threats and opportunities.

Happily, the recently proposed notion of *Complete Knowledge* (CK) offers a countermeasure to key encumbrance [41] where it is problematic for a blockchain application. CK enables such applications to avoid exposure to Liquefaction and similar tools. A key lesson of our work here is thus that as tools like Liquefaction increasingly realize new applications that liquefy assets, *many blockchain applications will need to use CK to protect their security and economic models against SEAO-assumption breaks*.

Contributions

In showing the fragility of the SEAO assumption and the implications of breaking it, we make several contributions:

- *Liquefaction*: We introduce Liquefaction, a general-purpose key-encumbered wallet with rich delegation / key-encumbrance policies that shows the fragility of the SEAO assumption and the implications of breaking it.
- *Key-encumbrance policies*: We present the first general model for key-encumbrance policies, offering a principled approach to formulating such policies in a flexible but sound manner (Section 3). Liquefaction demonstrates the practicality of our model.
- *Implementation*: We report on a fully functional implementation of Liquefaction that uses Oasis Sapphire as a back end (Section 4) and ensures liveness even in the case of a service failure (Section 5.1).
- *Ecosystem impact*: We explore the potential effects of Liquefaction across the blockchain ecosystem (Section 6). We enumerate and explore applications whose security model it can erode and report specifically on Dark DAO implementations that leverage Liquefaction. We also discuss use of CK as a countermeasure and, conversely, positive applications of Liquefaction.

We present background for our work in Section 2, discuss related work in Section 7, and conclude in Section 8.

2. Background

Liquefaction uses TEE-based key encumbrance to break the SEAO assumption in cryptocurrency and decentralized

finance (DeFi) systems. Here we offer brief background on these concepts.

2.1. Cryptocurrency and DeFi

A distinctive feature of the blockchain ecosystem as a whole is that assets are *fully controlled by means of private cryptographic (signing) keys*—in contrast to traditional web services, which offer less rigid forms of access control, e.g., through password resets. Consequently, blockchain assets are especially suitable for the type of complete programmatic control upon which Liquefaction is predicated.

2.2. Trusted Execution Environments (TEEs)

TEEs [56], [57] are secure areas within a computer processor that provide confidentiality and integrity for the code and data executed within them. TEEs create a protected execution environment, often implemented through extensions to the processor’s instruction set architecture, which ensures that code running inside the TEE cannot be read or tampered with by unauthorized entities outside the TEE. Typical examples of TEEs include Intel SGX [22], ARM Trustzone [4], etc.

TEEs have been widely integrated into blockchain systems [77], from serving as sources of randomness to supporting privacy-preserving applications.

Recent research has explored challenges associated with integrating TEEs into blockchain systems [10], [85]. Certain attacks, such as rollback attacks or liveness failures, can compromise the security of TEE-based blockchain systems [50], but certain TEE-based blockchains, such as Oasis Sapphire, include native rollback protections [38].

2.3. Key Encumbrance

The pervasive conceptual model of key ownership today assumes that a key is controlled / owned by a single entity. For example, the private key for a cryptocurrency address is typically custodied in a user’s wallet (hardware or software) and under unilateral control by the user. This model is logical in a world where sharing private keys with others exposes a user’s assets or privileges to theft.

TEEs, however, can disrupt this conventional model through a notion called *key encumbrance*.¹ Key encumbrance eliminates direct user control of a private key sk in favor of strictly programmatic control of sk . It can achieve selective delegation and fine-grained access control for private keys—and consequently blockchain assets and privileges—across a set of users.

As an example, key encumbrance enables a briber to effectively rent a private key sk for the purpose of voting. This is only possible if sk is encumbered in such a way that its owner cannot vote at the same time and potentially override the briber’s vote. In other words, key encumbrance

¹ Secure multi-party computation (MPC) can in principle do the same, although less practically.

in this case restricts the owner of sk *herself* from signing certain messages at certain times. Liquefaction implements rich policies that include restrictions of this kind.

A proposed countermeasure to key encumbrance is *complete knowledge* (CK) [41]—proof that a single entity / principal has direct *unencumbered* knowledge of the key and consequently lacks restrictions on its use.

3. Liquefaction Access-Control Model

In this section, we introduce a model that formalizes access policies for Liquefaction-encumbered keys. In our blockchain setting, since all actions require submitting appropriate *signed* transactions, we will treat the ability to sign messages as a proxy for access to resources or capabilities.

3.1. Basic Model

We model a set \mathcal{P} of players. Let \mathcal{M} denote the space of all possible messages (e.g., a suitable subset of $\{0, 1\}^*$). Intuitively, our goal is to use access-control policies to model the privilege of a particular player P in obtaining a signature (from the Liquefaction wallet) on a message m . For the purpose of this section, we assume the signatures are defined using a signature scheme $\Sigma = (\text{Gen}, \text{Sign}, \text{Ver})$.

Access-control policy. An access-control policy Γ details for each $(P, m) \in \mathcal{P} \times \mathcal{M}$, whether P is allowed to obtain a signature on m . We allow policies to be stateful—they can depend on a general system state st . The state st is split into three components: (1) an internal state intst , which is kept by the Liquefaction wallet, (2) an oracle state ost retrieved from oracle \mathcal{O} , which, e.g., models the current trusted blockchain state, and (3) an external state extst , which is provided by the player as part of requesting a signature. We write $\text{st} = (\text{intst}, \text{ost}, \text{extst})$.

Formally, an access-control policy is now a function $\Gamma: \mathcal{P} \times \mathcal{M} \rightarrow \{0, 1\}$, with the semantics that $\Gamma(P, m, \text{st}) = 1$ denotes that P is allowed to obtain a signature on m , given state st . We say that Γ is *stateless* if for all $P, m, \text{st}, \text{st}'$, it holds that $\Gamma(P, m, \text{st}) = \Gamma(P, m, \text{st}')$. When sufficiently clear, we will simply write $\Gamma(P, m)$ in these cases. We use $\Gamma = 0$ and $\Gamma = 1$ to denote the policies that approve no messages and all messages, respectively.

Policy updates. We will allow for a Liquefaction wallet policy to be updated to represent changes in access. As an example, we can allow a player to transfer control of her asset to another player through the wallet itself, i.e., without transferring the asset to a different address.

To this end, we define *policy update functions*—an update function β takes as input the calling player P , the current policy Γ_{curr} , the new policy Γ_{new} , and the state st , and outputs a bit denoting whether the policy update (from Γ_{curr} to Γ_{new}) is allowed. Our formalism allows for β to be an arbitrary predicate, but we discuss some practical choices later in Section 3.2 and Section 4.

Liquefaction wallet. Intuitively, a Liquefaction wallet is a key-pair (sk, pk) associated with a particular policy Γ , and

Liquefaction Wallet Management functionality $\mathcal{F}_{LW}^{\Sigma, \mathcal{O}}$
$\text{wallets} = \emptyset$
<ul style="list-style-type: none"> ◦ Upon input (“LWGen”, Γ, β) from P: Sample new key-pair $(\text{sk}, \text{pk}) \leftarrow \Sigma.\text{Gen}()$ Sample a unique id id_L. Add $L := (\text{id}_L, \text{sk}, \text{pk}, \Gamma, \beta, \emptyset)$ to wallets. Return $(\text{id}_L, \text{pk}, \Gamma, \beta)$
<ul style="list-style-type: none"> ◦ Upon input (“LWSign”, $\text{id}_L, m, \text{extst}$) from P: If id_L is not found in wallets, return \perp; Retrieve $L = (\text{id}_L, \text{sk}, \text{pk}, \Gamma, \beta, \text{intst})$. Retrieve ost from \mathcal{O}; let $\text{st} = (\text{intst}, \text{ost}, \text{extst})$. If $\Gamma(P, m, \text{st}) = 0$, return \perp. Add (P, m, ost) to $\text{wallets}.L.\text{intst}$. Return $\sigma = \Sigma.\text{Sign}(\text{sk}, m)$
<ul style="list-style-type: none"> ◦ Upon input (“LWUpdate”, $\text{id}_L, \Gamma', \text{extst}$) from P: If id_L is not found in wallets, return \perp; Retrieve $L = (\text{id}_L, \text{sk}, \text{pk}, \Gamma, \beta, \text{intst})$. Retrieve ost from \mathcal{O}; let $\text{st} = (\text{intst}, \text{ost}, \text{extst})$. If $\beta(P, \Gamma, \Gamma', \text{st}) = 0$, return \perp. Update $\text{wallets}.L.\Gamma$ to Γ'; return success.
<ul style="list-style-type: none"> ◦ Upon input (“LWVerify”, $\text{id}_L, (M, \sigma, f_{\text{vid}}), \text{extst}, P$) from Q: If id_L is not found in wallets, return \perp. Verify that σ is a valid sig. on (Q, M, f_{vid}) using P's key. Retrieve $L = (\text{id}_L, \text{sk}, \text{pk}, \Gamma, \beta, \text{intst})$, and ost from \mathcal{O}. Let $\text{st} = (\text{intst}, \text{ost}, \text{extst})$. If $f_{\text{vid}}(\text{st}) = 0$, return 0. If $\forall m \in M, \Gamma(P, m, \text{st}) = 1$, return 1; else return 0.

Figure 1. Liquefaction Wallet Management Functionality

an update function β . In Figure 1, we specify a general wallet manager functionality that can create, interact with, and update wallets. When an operation is performed, it is implicitly assumed that the functionality authenticates the interaction with the player. The manager functionality includes the following operations:

- LWGen creates a new Liquefaction wallet with a fresh encumbered key-pair (sk, pk) , where sk is only held by the manager functionality, and thus encumbered. Access-control management is specified through Γ and β as discussed previously.
- LWSign allows a player with the correct access to obtain a signature on a message m from the wallet's encumbered key.
- LWUpdate allows for updates to the access policy Γ , as specified by β .
- LWVerify allows a player chosen by P to verify that P has some specified access (given by message set M) to the wallet in particular states—given by a validity function $f_{\text{vid}}(\text{st}) \rightarrow \{0, 1\}$.

Remark 1 (Liquefying Liquefaction). Note that a player P 's access within a Liquefaction wallet could itself be “liquefied” by P —i.e., controlled through a separate Liquefaction instance and partitioned among a different set of players without the knowledge of the wallet itself. As an example, given access for a message set M , P could partition this

further on her own by giving Q access to M' and R access to $M \setminus M'$. Intuitively, this will be done by P encumbering the key she uses to authenticate to the Liquefaction wallet.

Such hierarchical liquefaction can be prevented if the Liquefaction wallet uses proofs of complete knowledge for authentication (see Section 5.2 for details).

3.2. Practical Instantiations of Γ and β

While our formalism allows for significant generality in Liquefaction wallets, important practical considerations arise in practice. We describe some of these in this section, and detail how they influence our choices for permissible policies Γ and for the policy-update function β .

First, a tension exists between the expressiveness of policies in Liquefaction and the feasibility for Liquefaction to identify or rule out potential policy conflicts. For example, the policy-update function β might permit an extremely expressive range of policies—e.g., policies realizable as arbitrary code. In this case, though, then determining whether a policy update introduces conflicts in asset control—i.e., conflicting ownership claims over assets—would be uncomputable in general. Consequently, β must bound the expressiveness of permissible policies.

A second important practical consideration is *conceptual simplicity*. The wallet policy and how it can change must be easy for ordinary users to understand. Only then can users reason about and benefit from applications built on Liquefaction. For example, a user lending an asset in a Liquefaction wallet has the simple assurance that she will regain control of the asset unconditionally after a predetermined interval of time elapses (as opposed to depending on complex conditions).

As a reasonable middle-ground, our implementation relies on a policy principle called *asset-time segmentation*.

Key principle for policies Γ : Asset-time segmentation. As a blockchain wallet system, Liquefaction manages control over blockchain assets (e.g., cryptocurrency or tokens). Liquefaction follows a key guiding principle for any policy: *a given asset is controlled exclusively by a single player P at any given time*. This basic principle is what we refer to as asset-time segmentation.

While asset-time segmentation is conceptually simple, one subtlety arises in the case of fungible assets, e.g., ether in Ethereum. To avoid unduly inflexible policies that must allocate an entire asset type to a single player, Liquefaction permits individual holdings to be shared among multiple players as distinct assets. These individual holdings may be thought of as virtual sub-accounts.

For instance, a policy for a wallet holding 15 ETH may specify that Alice controls the asset 5 ETH over the next week, meaning that she may spend up to 5 ETH during that time. Bob may simultaneously control 10 ETH during a concurrent interval of time. We regard their two allocations as distinct assets, realizable as a partitioning of the ETH balance in the wallet.

Formally, suppose that we have a set $\mathbf{A} = \{A_1, \dots, A_k\}$ of assets. Let $\text{spent}(P, \text{intst}, A_i) \rightarrow \mathbb{Z}^{\geq 0}$ de-

note the quantity of asset A_i currently spent by P . For a message m , let $\text{spend-req}(m, A_i) \rightarrow \mathbb{Z}^{\geq 0}$ denote that quantity of asset A_i that m is requesting to spend.

Now, we can define an asset-time segmented policy Γ as below:

- Γ is associated with a balance function $\text{bal} : \mathcal{P} \times \mathbf{A} \rightarrow \mathbb{Z}^{\geq 0}$ and a time function $T : \mathcal{P} \times \mathbf{A} \rightarrow \mathbb{Z}^+ \cup \infty$.
- A player P has access only to her allocated asset balance; that is, $\Gamma(P, m, \text{st}) = 1$ iff for all A_i , it holds that $\text{spent}(P, \text{intst}, A_i) + \text{spend-req}(m, A_i) \leq \text{bal}(P, A_i)$.
- Access is provided only until a specified expiration time. That is, whenever $\text{spend-req}(m, A_i) > 0$ and $t > T(P, A_i)$, it holds that $\Gamma(P, m, \text{st}) = 0$. Here, t denotes the current time (provided by \mathcal{O}).

Policy-update function β . Intuitively, a player P should only be able to update policies for assets that she currently controls; she should not be able to arbitrarily modify access currently held by other players.

Formally, a policy update $\Gamma \rightarrow \Gamma'$ will be allowed, or in other words, $\beta(P, \Gamma, \Gamma', \text{st}) = 1$, only if the following hold:

- Γ' is also an asset-time segmented policy.
- P cannot revoke access held by other players. That is, $[\Gamma(Q, m, \text{st}) = 1] \Rightarrow [\Gamma'(Q, m, \text{st}) = 1]$ for $Q \neq P$.
- Any new access given to a different player Q comes from P . That is, if $\Gamma(Q, m, \text{st}) = 0$ and $\Gamma'(Q, m, \text{st}) = 1$, then it must be that $\Gamma(P, m, \text{st}) = 1$.

Further restrictions may be added to β as required by the specific application.

Pre-signing considerations. Suppose that a Liquefaction wallet updates its policy from Γ to Γ' such that a player P has access to asset A in Γ , but not in Γ' . Care must be taken then to ensure that P did not obtain a signature σ on a message m while Γ is active, i.e., *before* the update to Γ' , such that σ corresponds to a transaction enabling access to A even *after* the update. This will be checked within β .

As an example, if not correctly handled, pre-signing could allow P to double-spend as follows. P obtains (but does not use) a signature σ from the Liquefaction wallet to transfer an asset A to a different address it controls. P then sells its access to A to another player Q and updates $\Gamma \rightarrow \Gamma'$ to internally switch the owner of A to Q . Finally, P submits σ to the blockchain to transfer A out of the wallet—stealing it out from under Q .

While application-specific semantics dictate how we need to handle pre-signing concerns, we give a general design intuition for β here. Ignoring balances for simplicity, for each player P and asset A_i , β will determine, based on the state st , whether the asset needs to be “sealed”—that is, either it has already been spent, or P has been provided a signature on a message that *could* spend it. Sealed assets may be unsealed later according to application semantics. Now, an update $\Gamma \rightarrow \Gamma'$ will only be allowed if it changes access control solely for “unsealed” assets. Assets with balances can be handled by determining what portion of the balance is unsealed.

Section 4.3 provides more details specific to our implementation.

3.3. Privacy

Liquefaction aims not just to enforce access-control policies correctly, but to do so in a privacy-preserving way. Specifically, a player P should learn only her own portion of a given wallet policy Γ , i.e., only $\Gamma(P, m, \text{st})$, and not $\Gamma(Q, m, \text{st})$ for any other player Q . The only way for P to learn any information about $\Gamma(Q, m, \text{st})$ should be through LWVerify—and only upon authorization by Q .

Formally, we define *policy privacy* as a player P not being able to distinguish whether the current policy is Γ or Γ' when they give identical access to P , i.e., when $\Gamma(P, m, \text{st}) = \Gamma'(P, m, \text{st})$ for all (m, st) .

Note that since the assets held by a Liquefaction wallet are still publicly visible on-chain, policy privacy cannot prevent anyone from observing on-chain transactions. Rather, it conceals players' control of assets within the wallet.

4. Implementation

To illustrate that Liquefaction is practical, we have implemented a wallet prototype which conforms to the core functionality described in Section 3. Our wallet prototype is a smart contract written in the most popular smart-contract programming language, Solidity. Despite its reliance on TEEs, our prototype demonstrates that developing Liquefaction tools and key-encumbrance policies need not require special knowledge of TEEs, thanks to the abstractions provided by TEE-based blockchains. In this section, we describe our implementation and its properties. Our open-source implementation can be found at <https://github.com/key-encumbrance/liquefaction>.

Our wallet prototype operates on Oasis Sapphire, a blockchain implementing the Ethereum Virtual Machine which runs entirely inside a TEE. As a result, Sapphire supports smart contracts whose state is kept private both during execution (in memory) and after execution (in storage).

A function can be called on a Sapphire smart contract in one of two ways: (1) by submitting a blockchain transaction, or (2) by making a free, off-chain query that simulates a blockchain transaction without modifying persistent storage. Both methods require a signed message for authentication and run entirely inside the TEE of an Oasis Sapphire node.

4.1. Wallet Architecture

An instance of Liquefaction, which we call a *wallet*, is realized as a contract on the TEE blockchain that we call the *wallet contract*. At its core, a wallet relies on several components: the backing TEE-based blockchain which provides built-in authentication, timestamping, and the platform on which our wallet contract runs; the wallet contract itself, which stores key material and directly signs messages; specialized, time-bound contracts that enforce policies assigning control of assets; and an *access manager*, authorized to initiate policy transitions.

Access manager. Use of the wallet begins with a player (or smart contract) requesting that the wallet contract generate a

fresh encumbered key sk using the TEE blockchain's built-in entropy generation methods. The player or smart contract who initiates this request is designated the *access manager* (AM) of sk , serving as the authority for initiating policy transitions. For example, Alice might set up an encumbered account to sell her voting power in a DAO. She would then serve as the access manager for sk . She can then decide, for example, whether she wants to accept a particular bribe and assign her voting rights to the counterparty. We emphasize, however, that the encumbered key sk is kept secret at all times within the wallet contract.

The AM can alternatively itself be a smart contract. In this case, sk is exclusively under programmatic control. We refer to the AM in this case as an *autonomous wallet*. (It may be thought of loosely as a private DAO. In fact one of our Dark DAO implementations is realized using an autonomous wallet.)

Access control via sub-policies. As explained in Section 3, use of sk is governed by an access control policy. In our prototype, access control policies are realized via a collection of *sub-policies*. Each sub-policy is implemented as a separate smart contract called a *policy contract*. A sub-policy enforces control of a given asset by a user or users. That is, it authorizes signatures on the asset when called by authorized users, as we explain below. Sub-policies expire after a pre-determined amount of time, which determines the duration of the access granted by the sub-policy. Together, sub-policies encode the complete access control policy associated with sk .

When an encumbered key is first generated, no signing privileges are available by default (thus, $\Gamma = 0$). To grant access to the key, AM authorizes a sub-policy to sign over a set of messages. Further modifications to signing capabilities require the creation of new sub-policies, which must not conflict with existing ones. This implies that policy updates can happen in only two ways: updates to access control arise from the *addition* of new sub-policies that are compatible with existing ones (Figure 2) or from the *expiration* of existing sub-policies (Figure 3). Access to an asset, once granted to a party, cannot be revoked while the corresponding sub-policy is active.

There is only one way for sub-policies to be added to

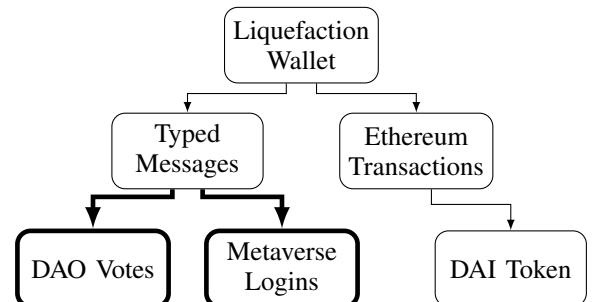


Figure 2. Example showing how new sub-policies are spawned. The Typed Messages sub-policy adds the DAO Votes and Metaverse Login sub-policies to the wallet's delegation tree—shown in bold for emphasis.

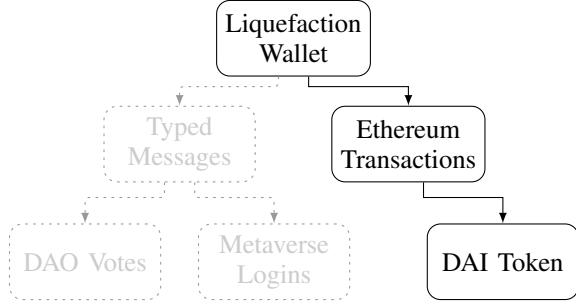


Figure 3. Example showing how sub-policies lose access to signatures no later than their parents. The Typed Messages policy expires; consequently, its sub-policies also lose access.

a wallet: Sub-policies may spawn further, dependent sub-policies, representing more fine-grained delegation of assets. Sub-policies thus form a tree, arranged hierarchically, such that the leaves are non-conflicting.

Policy transitions. A wallet policy Γ encompasses a set of sub-policies; thus $\Gamma = \{\Gamma_1, \Gamma_2, \Gamma_3, \dots, \Gamma_n\}$ for some n , where each sub-policy Γ_j corresponds to a node in the sub-policy delegation tree for the wallet, with Γ_1 at the root. As sub-policies change through creation and expiration, the result is that a wallet transitions among a sequence of different policies $\Gamma^{(1)}, \Gamma^{(2)}, \Gamma^{(3)}, \dots$, each with its own distinct set of sub-policies, i.e., $\Gamma^{(i)} = \{\Gamma_1^{(i)}, \Gamma_2^{(i)}, \dots, \Gamma_{n_i}^{(i)}\}$.

The policy-update function β determines the validity of a policy transition $\Gamma^{(i)} \rightarrow \Gamma^{(i+1)}$ when new sub-policies are created. Each sub-policy has its own local update function to decide which sub-policies may be created through delegation as children in the tree. As β only permits the instantiation of valid new sub-policies (with compliant policy contracts), Liquefaction thus enforces β within sub-contracts themselves, i.e., a sub-policy enforces validity under β of the new sub-policies it spawns.

The key property enforced globally by β in policy transitions is *asset-time segmentation*, as discussed in Section 3.2. This principle is enforced by assigning a given sub-policy *exclusive* access to a given asset until that access lapses at the policy's expiration time. Thus, a Liquefaction wallet (including AM) cannot remove or reduce access already conferred: Once Alice agrees to hand off her voting rights, she cannot also, for instance, assign them to herself before the agreement expires. Asset-time segmentation also prevents conflicts between sub-policies—e.g., Alice selling a single voting capability to two different bribers. Figure 4 illustrates how this works.

Transaction signing. A blockchain transaction takes the form of a signature with private key sk on a transaction message m . A sub-policy Γ_j authorizes specific users to sign transaction messages for particular assets.

To authorize a transaction, a user P sends a message m that she wants to have signed to the policy contract for the pertinent sub-policy Γ_j . The message-signing request is then checked recursively: first, the policy contract for Γ_j verifies that $\Gamma_j(P, m, st) \stackrel{?}{=} 1$; then the parent sub-policy in

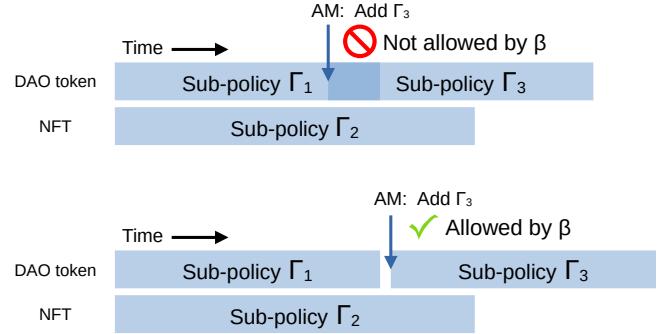


Figure 4. Asset-time segmentation requires assets be issued exclusively to a single sub-policy at a time.

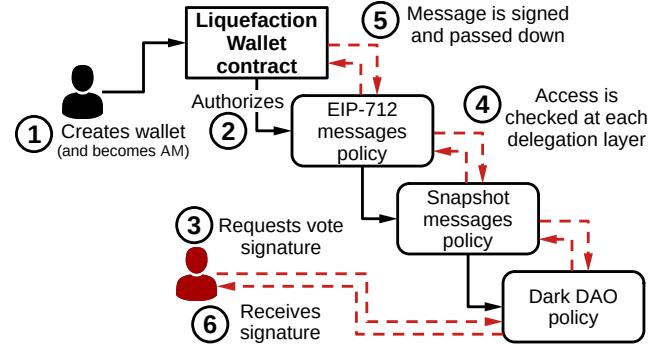


Figure 5. Liquefaction Wallet contract call flow for a Dark DAO policy selling access to vote messages (further described in Section 6.1). Arrows denote smart contract calls: solid arrows require a transaction, while dotted arrows are performed off-chain. The policies shown are a single path in the delegation tree.

the delegation tree checks compliance, etc. Upon successful verification by the root sub-policy Γ_1 , the wallet contract generates a signature on message m and passes it down the delegation tree back to the policy contract for Γ_j .

Figure 5 illustrates this process for an example sequence of sub-policies forming a path in a wallet's delegation tree. The specific sub-policies there are immaterial to the signing process illustrated in the example, but look ahead to the Dark DAO application discussed in Section 6.1.

4.2. Wallet Properties

In this subsection, we discuss a few important properties of Liquefaction wallets.

Policy integrity. Due to the architecture of the Ethereum Virtual Machine, policies cannot arbitrarily modify the access given to them. That's because smart contracts cannot directly access or modify the storage of other contracts. The TEE blockchain authenticates all calls to contracts—even those in off-chain simulations—ensuring correct authentication of entities calling a Liquefaction sub-policy.

Transaction types. A subtle issue in the enforcement of asset-time segmentation is that on-chain smart contracts are arbitrarily expressive. Consequently, there is no way for

a Liquefaction wallet to reason about or rule out policy conflicts for arbitrary on-chain transactions. Instead, Liquefaction constrains wallets to signing any of three common message types used in the ecosystem of EVM blockchains: Ethereum transaction messages [13], ERC-191 standard messages [75], and EIP-712 typed data [9]. The result is constrained transaction semantics for Liquefaction wallets, ensuring that Liquefaction can efficiently identify policy conflicts. (Extensions to other transaction types are possible, but require care to ensure against policy conflicts.)

Shared resources. For conceptual simplicity, we model control of an individual asset as residing with a single player P . As sub-policies are implemented programmatically, however, they can in fact realize more flexible forms of access control. For instance, it is possible to apportion shared access to a single asset in Liquefaction, e.g., allow simultaneous access by players P_1 and P_2 .

Address privacy. Suppose a player Q gains access to a resource in an encumbered wallet—e.g., Q is a briber buying votes in a Dark DAO. Q then can obtain signed transactions from the sub-policy selling the votes, namely transactions casting ballots. These transactions must reveal the address a controlled by the wallet (i.e., corresponding to encumbered key sk). It is possible, however, to conceal a from Q by using a TEE intermediary which publishes signatures / transactions directly to the target blockchain—with careful attention to timing side channels and the size of anonymity set containing a .

4.3. Handling Sub-Balances and Fees

Asset-time segmentation in Liquefaction treats fungible assets in a special, flexible way, as discussed in Section 3.2. An address’s balance in a particular token or currency, e.g., ether, can be apportioned to different players. The balance is divided into sub-balances (analogous to virtual sub-accounts), each of which is treated as a distinct asset. This approach permits concurrent access by multiple applications to a particular asset class in a given address, e.g., to ether in a single encumbered Ethereum address.

We realize sub-balances for ether in encumbered Ethereum accounts through a policy contract in control of all Ethereum transaction messages. Managing per-account ownership of non-ether resources on Ethereum, such as ERC-20 tokens and NFTs, usually involves sending transactions to particular smart contract addresses (e.g., a token’s address); these also spend ether. Therefore, in addition to ether sub-balances, we model transaction destination addresses as separate assets. AM can create sub-policies and grant them exclusive access to send transactions, spending ether, to one or more particular destination addresses.

Goals. Our policy has to ensure that (1) sub-policies cannot deny service to one another, (2) sub-policies cannot spend funds intended for another sub-policy, (3) additional TEE blockchain transaction costs for signing transactions are minimized, and (4) sub-policy access expires as specified by AM during sub-policy creation.

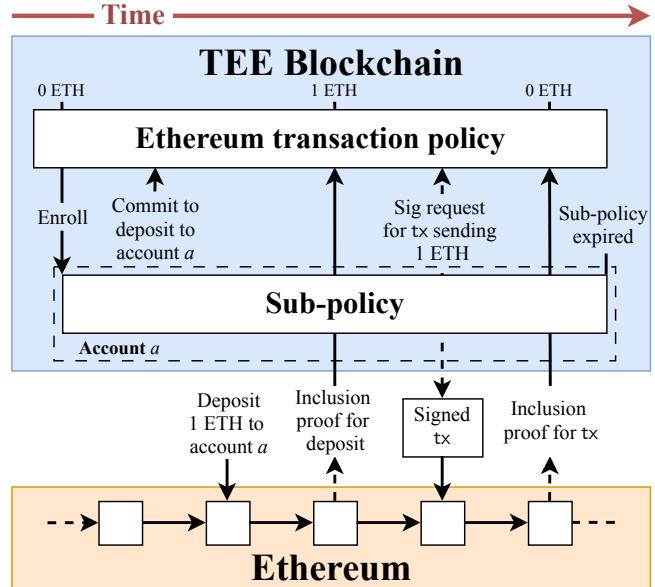


Figure 6. Typical usage of the Ethereum transaction policy: a sub-policy must deposit funds to increase its sub-balance before it can obtain a signed transaction. Solid arrows indicate transactions.

Pre-signing. The policy must prevent sub-policies from pre-signing transactions which would enable them to spend funds or maintain continued access after their access expires.² Although Ethereum transactions do not include expiration timestamps, each account is required to include a sequential transaction nonce in every transaction message it sends [12]. We require all signed transactions to use the current account nonce, which prevents sub-policies from sending more than one transaction after their access expires, fulfilling goal (4) to the maximum extent possible. After a transaction is included on Ethereum, any sub-policy may submit a transaction inclusion proof to increment the account nonce recognized by the policy. This process unlocks signatures for the next transaction and prevents denial of service as required by goal (1). The inclusion proof is a Merkle proof that verifies the transaction is embedded in the transaction trie of a canonical Ethereum block. We assume a trusted oracle of Ethereum block hashes (possible via, e.g., [84]), to function as the root of trust for these proofs.

Accounting. We maintain an internal mapping from a sub-policy to its ether sub-balance. To deposit funds to its sub-balance, a sub-policy must first create an Ethereum transaction which sends funds to the encumbered account from an outside source and then claim responsibility for it to the policy. The policy enforces that only one sub-policy can claim a particular transaction hash.³ After the transaction is included on Ethereum, the sub-policy can

2. Even if an authorized destination address does not support directly extracting funds, sub-policies could collude with block builders to extract ether by using a high transaction priority fee.

3. Sub-policies should claim a deposit transaction hash *before* broadcasting the transaction for inclusion on-chain; otherwise, a different sub-policy could learn the transaction hash and claim it first.

submit an inclusion proof to add the deposited funds to its sub-balance. This process is shown in Figure 6.

To fulfill goal (2) and prevent sub-policies from spending more than their sub-balance, we enforce a spending limit on each signed transaction. Additionally, we deduct transaction costs from the sender’s sub-balance when a transaction inclusion proof is submitted. This deduction occurs atomically with the increment of the recognized account nonce to prevent double-spending.

Aligning with goal (1), we similarly require sub-policies to cover the cost of proving their transactions’ on-chain inclusion to the policy. Sub-policies additionally maintain a sub-balance of the TEE blockchain’s native token (used for transaction fees on the TEE blockchain) within the policy contract. When a valid transaction inclusion proof is provided to the policy, an estimated cost of this action is deducted from the transaction signer’s sub-balance and refunded to the submitter.

When a sub-policy’s access ends, leftover funds remain locked in its sub-balance. If AM grants the sub-policy access again, it can spend funds from its existing sub-balance.

Transaction attribution. Critically, to deduct from the correct sub-balance, the policy must determine which sub-policy signed included transactions. Additionally, we aim to allow sub-policies to sign transactions off-chain (i.e., without requiring changes to the TEE blockchain’s state), at no cost, to fulfill goal (3). Typically, we can attribute a transaction t to sub-policy Γ_i if Γ_i is the sub-policy authorized to sign with t ’s destination address at the time t ’s inclusion is proven. However, a special case arises if Γ_i waits to broadcast t or prove its on-chain inclusion until *after* its access expires and a second sub-policy $\Gamma_j \neq \Gamma_i$ has been given access by AM for the same destination address as t . This situation can only occur if no other transaction was included before t , which would increment the account nonce and invalidate t .

To attribute transactions correctly even in this case, we require newly created sub-policies to commit their transaction requests to blockchain storage until the account nonce increases (i.e., until inclusion is proven for a transaction sent from the wallet account). The policy only releases a transaction signature to a new sub-policy after its transaction request is confirmed on the TEE blockchain. This mechanism allows the policy to attribute a transaction t to sub-policy Γ_j if either Γ_j had committed a request for t or if Γ_j was the last sub-policy with unlimited off-chain signing to t ’s destination address.

Performance. Requiring an inclusion proof to unlock the next transaction introduces a minimum delay between sending transactions which is primarily dependent on the trusted block hash oracle’s judgment of block finality. We evaluated the average time between signing an Ethereum transaction through the policy and proving its on-chain inclusion under three block hash oracles with different Ethereum commitment levels recognized by the community [70], as shown in Figure 7. The strongest assurance against double-spending—block finality—comes at a latency cost of over 17

Metric	Latest	Justified	Finalized
Mean (seconds)	49.0	648.7	1,036.9
Standard Deviation	7.4	125.2	113.8

Figure 7. Average end-to-end transaction inclusion and inclusion proof cycle latency across 160 trials, varied by the trusted block hash oracle’s confirmation requirements: the oracle confirming the *latest* block accepts a block as soon as it is received, *justified* when it receives attestations from two-thirds of Ethereum’s validator set, and *finalized* when it is one epoch behind the most recent finalized block.

Oasis Transaction	Gas Usage	Cost (USD)
Deploy policy	7,777,890	\$0.05382
Add policy to wallet	169,145	\$0.00117
Add sub-policy	87,162	\$0.00060
Deposit commitment	50,769	\$0.00035
Prove deposit inclusion	364,429	\$0.00252
Transaction commitment	140,772	\$0.00097
Prove transaction inclusion	395,679	\$0.00273

Figure 8. Costs of using the transaction encumbrance policy, assuming that 1 ROSE = \$0.06919 (the price as of October 29, 2024) and transactions are priced at 100 Gwei, the Sapphire default.

minutes on average. This latency figure will be substantially reduced by future changes to Ethereum’s consensus [82].

We also measured the TEE-blockchain transaction costs associated with using the policy in Figure 8. All costs are currently orders of magnitude cheaper than the cost of the Ethereum transactions themselves, thereby adding very little additional cost to using the wallet.

4.4. TEE Security Model

We consider an idealized model of Oasis Sapphire’s trusted execution environment [69], treating side-channel issues and platform-level deployment mistakes (see, e.g., [16], [38], [78]) as out of scope for our exploration in this paper. We also assume the integrity, i.e., correct execution, and liveness of the TEE blockchain.

Communications to the TEE blockchain can be observed by a network adversary. The system supports secure channels to application instances, however, and we exclude consideration of side channels resulting from, e.g., analysis correlating Oasis Sapphire traffic with on-chain behavior. (Such side channels can be mitigated through injection of noise, e.g., randomized delays.)

TEEs have a history of side-channel vulnerabilities [31], [59], [79] that violate confidentiality and integrity properties. Within the blockchain community, however, TEEs already are being trusted for practical applications that rely on TEE confidentiality [6], [64]. Also, automated bug-bounty programs for TEE security can be implemented using frameworks like Sting [42] or Keystone [48], encouraging public disclosure and the ethical use of exploits. Thus we believe that despite the risks, key encumbrance is likely to arise in the wild. As we now explain, moreover, some Liquefaction use cases only require ephemeral exposure of sk to a TEE.

Ephemeral key exposure. While there are reasons for concern about the ability of TEEs to protect secret keys in the long term, certain Liquefaction applications do not require persistent TEE access to sk . Instead, they can realize a model of ephemeral key exposure to the TEE.

An example is our dusting-attack mitigation (see Section 6.4). There, sk can be $(2, 2)$ -secret shared. The TEE holds one share and the owner P of sk can hold the other, inputting it to the TEE for temporary reconstruction of sk only to initiate outgoing transactions.

5. Practical Considerations

Our prototype from Section 4 is fully functional, capable of supporting any application of Liquefaction. Yet, real-world deployment of Liquefaction raises additional concerns—both technical and ethical ones—which we discuss in this section.

5.1. Challenge: Liveness

Because Liquefaction encumbers key material within TEEs, no single party has (complete) knowledge of its keys. The safety of users’ financial assets held in a Liquefaction wallet therefore depends on the liveness / availability of the underlying TEE-based blockchain—in our prototype, the Oasis Sapphire blockchain.

Liveness is a key property of blockchains. In practice, though, even some of the most popular blockchains, such as Solana and Avalanche, have experienced downtime—as recently as February 2024 [60], [61].

We have consequently designed Liquefaction to achieve a notion of safety that is *stronger than that of the underlying TEE-based blockchain*. In this section we describe the design of a *fallback system* (Figure 9) whose objective is to ensure that keys are not lost even in the case of a catastrophic failure or sustained outage of the underlying, or *primary*, blockchain. In such a case, the fallback protocol transitions to an external, TEE-based fallback system that protects keys using a fallback key-encumbrance policy. This policy extends the lifecycle of Liquefaction keys, allowing them to be transferred to a new TEE-based blockchain or released to their access managers, removing their encumbrance.

Security goals. Liquefaction’s fallback system can be understood as implementing a *fallback trigger* that causes control of Liquefaction keys to be transitioned from encumbrance policies on the primary blockchain to *fallback encumbrance policies* on the fallback system in the event of a failure in the primary blockchain. Ensuring correct encumbrance of keys throughout their lifecycle requires liveness and soundness for both the primary and fallback systems.

Additionally, however, it requires the key security property of *sound fallback*: a transition from primary to fallback systems and encumbrance policies should occur if and only if the primary blockchain experiences a failure as specified by the fallback trigger.

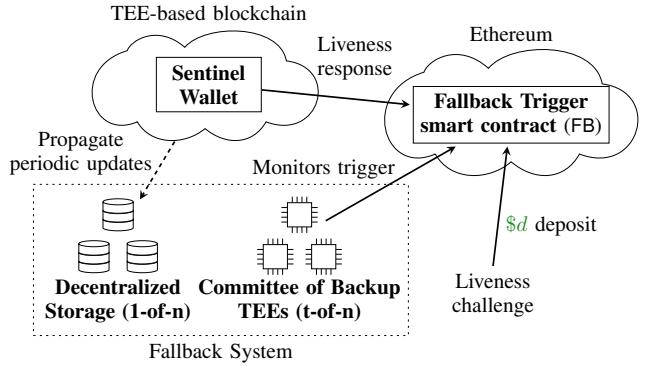


Figure 9. The fallback system is triggered in the rare case that the TEE blockchain goes offline. A smart contract on Ethereum handles challenges to the liveness of the primary system, and a network of backup TEEs take over in the case of a successful challenge.

Securing sound fallback is the main technical challenge in the design of Liquefaction’s fallback system.

Ensuring the soundness and liveness of the fallback system itself also requires careful system design. For clarity of exposition and practicality, however, we focus in what follows on a simple fallback trigger and fallback key-encumbrance policy, as follows:

- *Fallback trigger*: Transition to the fallback system occurs iff a liveness interface in the Liquefaction application known as a *sentinel wallet* (having $\Gamma = 1, \beta = 0$) provably fails to sign messages for a time interval of length at least T (e.g., the industry-standard duration of a week [44]).
- *Fallback key-encumbrance policy*: The fallback system releases a key sk to its access manager, i.e., removes encumbrance.

Of course, these are just examples. Other policies are possible.

Design overview. Briefly, we implement the fallback trigger as a smart contract on a high-reliability blockchain, Ethereum. If at time τ a liveness challenge m is submitted to the smart contract, and a signature from the sentinel wallet on m is not provided to the smart contract before T , the fallback trigger activates. We require a bounty $\$d$ to be included with liveness challenges to cover the cost of response transactions to prevent denial-of-service attacks. See Appendix B for the full details of our fallback system.

5.2. Mitigation: Complete Knowledge

Liquefaction can impact a broad range of applications (Section 6), but applications can prevent mitigate against Liquefaction by requiring that users provide proofs of *complete knowledge* (CK) [41] for their private keys.

A CK proof for a secret key sk shows that some user has obtained *unencumbered access* to sk (similar ideas also appeared in [26]). Such proof, once given, applies to a key for the remainder of its lifecycle. Once a key sk has been disclosed to a user, exclusive possession of the key by an

enclave can never be guaranteed and thus can't be proven to a third party.

CK proofs are thus a mitigation to the adversarial applications enabled by Liquefaction.

We discuss additional details of CK proofs in Appendix C, and other ethical considerations in Appendix D.

5.3. Hiding Liquefaction Wallets

An important practical consideration for Liquefaction wallets is the ability to hide information about wallet contents (and sometimes even the existence of Liquefaction itself) from being publicly provable. For example, if it can be publicly proven that a particular soulbound token is available for rent, this can be used by the token-issuing contract to revoke it, reducing the power of Liquefaction. In a sense, for correct functioning, the Liquefaction wallet must be able to prove to potential renters which soulbound tokens are available for rent, without these proofs being transferable.

The standard way to achieve this is using *designated verifier (DV) proofs*, which are proofs that convince only a designated party \mathcal{V} and no one else. This is accomplished by allowing \mathcal{V} to forge proofs; this prevents \mathcal{V} from convincing another party (including e.g., a public smart contract) by transferring the original proof.

Remarkably, failure of the SEAO assumption also breaks DV proofs ([41], [54]). Intuitively, if \mathcal{V} 's forging key is itself encumbered, then \mathcal{V} regains the power to transfer the proof to others. To get around this, the Liquefaction wallet can itself weaponize CK to restore the DV property. This will be done by first requesting a CK proof from the verifier \mathcal{V} ; we call these DV-CK proofs.

As mentioned before, applications can require CK proofs as a generic solution against key encumbrance. But in the absence of the application requiring CK, Liquefaction can itself be strengthened by weaponizing CK.

6. Blockchain Applications

Liquefaction undermines the economic and security models of applications that rely on the SEAO assumption. Conversely, it enables new, beneficial applications. In this section, we enumerate the impacts of Liquefaction across five blockchain application categories: governance, reputation, privacy, ticketing, and provenance. Each application can be realized on Liquefaction through sub-policy contracts. Our findings are summarized in Figure 10. In addition, we provide implementations for the Dark DAO, soulbound token sharing, and dusting attack mitigation applications, which we include in our open-source repository.⁴ We describe several applications in this section, and refer readers to Appendix A for the details of the rest of the applications shown in Figure 10. The applications we present are just a few examples of the impact of Liquefaction and are not an exhaustive list of its reach. We believe Liquefaction has a systemic impact on the blockchain ecosystem.

4. <https://github.com/key-encumbrance/liquefaction>

6.1. Governance

Decentralized Autonomous Organizations (DAOs) are smart-contract-based communities governed by means of voting. DAO members cast ballots on proposals to decide which actions the DAO takes. Votes are often weighted by the number of DAO tokens held by an address. Usually, the two capabilities granted by owning a DAO's token—voting power and the token's market value—are intertwined: apart from protocol-supported vote delegation, transferring an individual's voting power requires transferring ownership of their tokens. Liquefaction enables the separation of these two capabilities which, for ordinary wallets, are inseparable.

Dark DAOs. We have written an encumbrance policy that liquefies votes in DAOs, realizing a *Dark DAO*, defined in [23] as a “decentralized cartel that buys on-chain votes opaque.” Like an ordinary DAO, a Dark DAO is designed to be trust-minimized: it ensures fair exchange, i.e., Alice receives money from a briber if and only if the briber gains agreed-upon access to Alice’s credential(s). Additionally, a Dark DAO is opaque: participation is private, and to a briber, it ensures policy privacy.

A briber desiring a given election outcome (e.g., a “yes” vote) can simply offer payment for votes toward this outcome, where payments are scaled to the weight associated with a given bribee’s vote. This fair exchange can be implemented using two contracts: an encumbrance policy which broadly encumbers voting messages, and another corresponding to the specific vote-buying contract.

Our voting message encumbrance policy assumes exclusive control of all message signatures pertaining to a popular off-chain DAO voting platform, Snapshot [87]. The policy allows a *voting manager* account (by default, the wallet’s access manager) to delegate the capability of signing vote messages to exactly one account per DAO proposal. The voting manager may delegate the capability to a vote-buying contract, forgoing the ability to sign voting messages itself. This mechanism guarantees to the Dark DAO that a user cannot override a vote that it signs on the user’s behalf.

The briber offers payment for votes in a vote-buying contract which verifies the user has voting power in the DAO and (privately) delegated signing rights to the Dark DAO. After these checks, the contract pays the user by reserving funds which the user may later claim. The entire operation—private voting delegation, voting power verification, and payment—takes place *atomically*, such that if any operation fails, the entire transaction reverts.

Tokenized Dark DAO Lite. We also designed a second Dark DAO system, which we call a *Dark DAO “Lite”*, in a way that achieves greater usability and coordination, but weaker confidentiality. The key idea behind the Dark DAO Lite is its use of a DAO-token derivative, the *DD token*, to hide the complexity of participation. DD tokens are derived from ordinary target-DAO tokens through a conversion process, and like regular DAO tokens they can be traded on existing token markets, such as Uniswap.

The Dark DAO Lite separates the capabilities of a DAO token into two assets, as follows:

Category	Application	Context	Liquefaction Enables	Broken Assumptions	Constructive Use	Adversarial Use
Governance	Voting	Decisions in DAOs are made via community votes on proposals [21].	Dark DAOs, which are private DAOs that can operate within public DAOs or systems [23]	Voters control their own tokens.	Private delegation	Bribery attacks
	Quadratic Voting	Voting power is distributed quadratically to empower small accounts [47].	Using a Dark DAO to square voting power, even in the presence of identity systems	One person controls each account.	N/A	Whales increase voting power by account splitting
	Multisigs	Multiple signers' approval is required to execute via a multisig [37].	Selling access to a key participating in the multisig	Signers control their own keys.	Enriched private access structures	Coordinated theft
Reputation	Soulbound Tokens	Soulbound tokens are nontransferable NFTs [66].	Renting soulbound token proofs of ownership to third parties	The account owner has exclusive access to an NFT.	Facilitating broader group membership	Identity misrepresentation
	Transaction History	Transaction history is a complete and public per-account record [74].	Purchasing accounts to make deposits in an exchange with strict risk requirements	The past account user is the current account user.	Broadening access to credit and service	Identity misrepresentation
	Airdrop Rights	The right to receive an airdrop can be based on account history [18].	Trading encumbered access to accounts that may be eligible for future airdrops	The past account user is the future account user.	Unlocking liquidity now by selling future rights	Subverting the desired recipients of the airdropper
	Loyalty Points	Loyalty points can be given based on account history [2].	Splitting the cost of a single account which accumulates rewards for a group	Accounts cannot be shared or transferred.	Sharing access to points with newer users	Subverting the desired recipients of the rewards
	Wash Trading	Trading volume for an NFT or token can be artificially increased [5].	Less traceable wash trading	Trading accounts that appear unconnected are unconnected.	N/A	Artificial volume which appears authentic
Privacy	Trading Locked Tokens	Smart contracts which issue assets sometimes dictate how the assets must be sold or distributed [20].	Trading locked tokens and bypassing transfer fees or transfer limits	Account owners cannot share portions of their accounts.	Greater token liquidity and more trustworthy derivatives	Early "dumping" of vested tokens, reduced token fee revenue
	Private Asset Trading	Standard asset transfers require public, on-chain transactions [36].	Trades across multiple user wallets without on-chain transactions	Account owners control all assets in their accounts.	Avoiding paying gas for on-chain transactions	Misrepresentation of activity
	Private DAO Treasuries	DAOs' public treasuries may undermine applications of fundraising DAOs [83].	Raising and storing funds in a decentralized, invisible manner	DAO blockchain assets are publicly measurable.	Crowdfunding auction bids	Attack financing
	Secret Contract Payments	Smart contracts can be made to pay out bounties [40].	Paying bounties by revealing the secret keys of encumbered wallets	Smart contract asset transfers are public.	Compensating whistleblowers	Rogue smart contracts
	Hidden Privacy Pools	Privacy Pools is a privacy protocol with built-in accountability [14].	A Dark DAO which enforces inclusion of its members in association-set proofs	Privacy Pools depositors can create any proof.	Stronger in-group privacy	Weakened accountability
Ticketing	Token-Gated Ticketing	Access to an event (or metaverse character) can be based on ownership of an asset or NFT [68].	Transferring event access to someone who does not own the asset	The account owner has exclusive access to an NFT or asset.	Unlocking liquidity by renting a ticket-bearing asset	Undermining event exclusivity or intended audience
Provenance	Faking Theft	When theft happens, people look on chain for evidence [86].	Funds to be sent to a "thief's" encumbered account, secretly accessible to the "victim"	Asset transfers establish loss of ownership.	N/A	Insurance fraud
	Dusting Attack Mitigation	Tokens from an illicit source may be sent unsolicited to any other account [19].	Proving that the target of a dusting attack has not assumed control of illicit assets	All funds sent to a wallet are accessible to its owner.	Proving lack of ownership of unsolicited deposits	N/A
Cross-chain	Overlay Smart Contracts	Many blockchains do not support smart contracts [7].	Treating encumbered addresses as smart contract interfaces	Contracts need native chain support.	Cross-chain bridges	N/A

Figure 10. Table of example applications Liquefaction enables or impacts. Rows marked in red indicate that we have implemented this application.

- *Voting rights* corresponding to converted target-DAO tokens. A pool of these voting rights may be purchased by a vote-buyer / briber through an auction mechanism. We refer to the pool of voting rights as *self-auctioning*, since the auction process is automated and requires no intervention by DD token holders.
- *DD tokens*, which may be individually owned and correspond to ownership rights in the target DAO plus the right to receive revenue from the auctioning of the pool of fractionalized voting rights.

Through the use of the Dark DAO "Lite," participation

in a Dark DAO is as simple as purchasing DD tokens. Due to the self-auctioning mechanism, the value of held DD tokens will increase relative to the underlying target-DAO token.

See our [code repository](#) for implementation details.

6.2. Reputation

Cryptographic keys are often tied to reputation; this is most apparent when an identity is associated with a particular key. The keys' activities—those messages and transactions signed by them—are then assumed to have

been performed by the associated identity. Markets can be created which sell either ownership to assets held by a key-encumbered wallet or access to the private key itself, since the TEE controls the private key.

Soulbound tokens. Soulbound tokens [32] are presumed to be illiquid under the SEO assumption because they are designed to be non-transferable between accounts. Advocates posit that soulbound tokens will represent “commitments, credentials, and affiliations” of individuals in a decentralized society, becoming the standard interface for measuring reputation. Due to their non-transferability, soulbound tokens have been suggested as a powerful primitive for reputation [32]. However, if the soulbound token is held by an encumbered account, Liquefaction enables proofs of its ownership to be rented to third parties.

To demonstrate that soulbound tokens are already at risk of encumbrance-based sales, we collaborated with a company (Flashbots) to encumber a soulbound token. Flashbots had recently released a collection of soulbound tokens to its employees. They issued a soulbound token from their collection to an address of our choosing—an encumbered account created from a *maximally upgradable* wallet contract,⁵ which generates key material but does not support signing messages ($\Gamma = 0, \beta = 0$). The only function of this wallet is to transfer its key securely to a new wallet contract via a key exchange, thereby making it universally upgradable to any other wallet, encumbered or not. We discussed with Flashbots how this wallet could then be upgraded to sell access to the soulbound token. To prevent this, however, we instead released the key by “upgrading” the wallet via encrypted transfer to a real Flashbots employee. We plan to work with Flashbots to perform a “CK ceremony” among their employees and thereby prove that none of their soulbound tokens are encumbered any longer.

6.3. Privacy

Due to the public nature of transparent blockchains, the balance and transaction activity of all accounts are public. Public records are not always desirable, however; Liquefaction can facilitate private transfers of ownership that do not leave an on-chain record.

Trading locked tokens. Some tokens are distributed and activated with a vesting schedule to ensure the investors and founders are aligned with a platform’s goals. Liquefaction enables holders of locked tokens to liquefy and sell their tokens without initiating an on-chain transfer, while maintaining the appearance of respecting their vesting schedules. Even without token locks, prominent community members with publicly tracked wallets might similarly wish to trade their assets without investor scrutiny. Though some smart contract wallets can already bypass token locks, they do not offer any confidentiality and can be easily blocked by

5. The wallet contract can be found at [0xF4D0...26C4](#) on the Oasis Sapphire Mainnet, and the (previously) encumbered key at [0x2D25...569B](#) on Ethereum.

limiting distribution to EOAs and approved wallet contracts. Liquefaction makes this countermeasure ineffective.

6.4. Provenance

Dusting attack mitigation. Most extant blockchain protocols provide no means for users to refuse asset transfers into their accounts. Thus, accounts may be tainted by the unsolicited receipt of funds originating from an illicit source, such as blockchain addresses on a sanctions list—a *dusting attack* [65]. Victims of dusting transactions from illicit sources may be excluded from business with centralized cryptocurrency exchanges or subject to asset freezes.

Encumbered wallets can provide a defense against dusting. An encumbered wallet which requires asset deposits to be pre-approved before they can be spent can show that the target of the attack does not have, and never had, access to unapproved funds transferred to the account. Our transaction encumbrance policy from Section 4.3 implements this pre-approval mechanism by requiring deposits to be committed before they can be spent. Additionally, upon wallet initialization the access manager can configure this policy to enable exporting proofs of deposit non-ownership.

7. Related Work

Key encumbrance. Early work considered a form of key encumbrance in devices with attestation achieved with physical proximity, e.g., visually, and described several (malicious) applications [55]. A blog post by Daian et al. introduced the concept of key encumbrance in TEEs with remote attestation, mainly for Dark DAOs—confidential marketplaces for bribing blockchain voters [23]. It lacked a TEE-based implementation, however. Li et al. used key encumbrance to realize offline commitment to transfers of cryptocurrency [49]. Related works have explored application-specific encumbrance of web credentials, rather than keys, for internet services such as e-mail and online payments [53] and social-media and government-issued identities [71]. Key encumbrance, as implemented through in-account resource locks, has also been suggested as an architecture for enabling atomic, cross-chain state transitions [34].

Secure credential delegation. Daian et al. put forth the notion of a Dark DAO—a DAO that aims to subvert voting in other DAOs—in [23]. In related work, Matetic et al. propose use of TEEs as a tool for secure credential delegation—which may be unauthorized [53], and Puddu et al. explore malicious uses, including subversion of e-voting [71]. Gunn et al. demonstrate the use of remote attestation to undetectably produce provable records of messages that were designed to be deniable, breaking the user privacy guarantees of these services. [35]

Encumbrance defense. Kelkar et al. proposed and formalized the notion of complete knowledge (CK), the ability to prove that one does not just have access to a secret, but also has fully unencumbered knowledge of it [41]. Similar ideas also appeared independently in [26].

Vote-buying / coercion. There is a considerable literature on the notion of *coercion-resistance* in end-to-end verifiable voting [24], [39], [52]. Broadly speaking, coercion-resistance means that a voter cannot convince a would-be briber or coercer of how she voted. Influential proposed coercion-resistant voting systems include notably Civitas [17] and, more recently, MACI [11]. None of these definitions or system designs contemplate the risk of key encumbrance; Dark DAOs effectively break all of them.

8. Conclusion

Liquefaction demonstrates the significant impact TEEs and key encumbrance policies can have on the blockchain ecosystem. Liquefaction shows that breaking the SEAO assumption is feasible by implementing fine-grained access-control policies for encumbered keys. Importantly, Liquefaction operates confidentially, supporting transactions without on-chain traces and with minimal off-chain visibility.

Our work highlights the limitations of traditional key ownership models for cryptocurrency and the risk of adversaries privately undermining voting integrity, manipulating loyalty systems, and facilitating stealthy wash trading. The obfuscation of transaction history and delegation of control supported by Liquefaction can enable misuse of DAOs, soulbound tokens, and reputation-based systems. Conversely, Liquefaction enables new applications, such as privacy-preserving DAOs, overlay smart contracts, and a mitigation to dusting attacks.

Complete Knowledge (CK) offers a countermeasure to key encumbrance, allowing specific applications to avoid its impacts. As TEEs grow more powerful, developers must consider CK in their designs. In summary, our work calls for a broad reconsideration of asset ownership models and recognition of how technologies like TEEs will affect them.

Acknowledgments

Thanks to Flashbots for productive discussions and collaboration regarding soulbound tokens, and to Andrew Miller for helping carry out the encumbrance of a soulbound token. Thanks also to Amy Zhao for insightful suggestions and comments on this work. This work was funded by NSF CNS-2112751 and generous support from IC3 industry partners and sponsors. Andrés Fábrega was funded in part by a TLDR Research Fellowship.

References

- [1] 1inch Network. The 1inch Foundation will distribute 10 mln 1INCH to the defi community as gas cost refunds. <https://blog.1inch.io/the-1inch-foundation-will-distribute-10-mln-1inch-to-the-defi-community-as-gas-cost-refunds/>, 27 Jul. 2021.
- [2] Dhwanil Agrawal, Natalia Natalia, Gajane Gopalakrishnan, Margaret Natalie Guzman, Michael D McDonald, and Henry M Kim. Loyalty points on the blockchain. *Business and Management Studies*, 2018.
- [3] Sarah Allen, Ari Juels, Mukti Khaire, Tyler Kell, and Siddhant Shrivastava. NFTs for art and collectables: Primer and outlook, 2022.
- [4] Tiago Alves. Trustzone: Integrated hardware and software security. ARM White Paper. <https://api.semanticscholar.org/CorpusID:58463902>, 2004.
- [5] Andrey Sergeevkov. What is NFT wash trading? <https://www.coindesk.com/learn/what-is-nft-wash-trading>, 9 Apr. 2024. Accessed: 2024-10-23.
- [6] Gogul Baliga, Alex Obadia, Arjun Bulusu, and Bernardo Magnani. Block building inside SGX. <https://writings.flashbots.net/block-building-inside-sgx>, 2023. Accessed: 2024-10-06.
- [7] Massimo Bartoletti and Livio Pomponi. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *Financial Cryptography and Data Security*, 2017.
- [8] Binance Academy. Token lockup, 2024. Accessed: 2024-10-31.
- [9] Remco Bloemen, Leonid Logvinov, and Jacob Evans. EIP-712: Typed structured data hashing and signing. Ethereum Improvement Proposals, September 2017. <https://eips.ethereum.org/EIPS/eip-712>.
- [10] Marcus Brandenburger, Christian Cachin, Rüdiger Kapitza, and Alessandro Sorniotti. Blockchain and trusted computing: Problems, pitfalls, and a solution for hyperledger fabric. *CoRR*, abs/1805.08541, 2018.
- [11] V. Buterin. Minimal anti-collusion infrastructure (MACI). Ethereum Research blog post, <https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413>, 2 May 2019.
- [12] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. Ethereum whitepaper, <https://ethereum.org/en/whitepaper/>, 2014. Accessed: 2024-10-16.
- [13] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. EIP-1559: Fee market change for eth 1.0 chain. Ethereum Improvement Proposals, April 2019. <https://eips.ethereum.org/EIPS/eip-1559>.
- [14] Vitalik Buterin, Jacob Illum, Matthias Nadler, Fabian Schär, and Ameen Soleimani. Blockchain privacy and regulatory compliance: Towards a practical equilibrium. *Blockchain: Research and Applications*, 5(1):100176, 2024.
- [15] Bill Chappell. A crowd-funded group lost an auction for a first edition of the U.S. constitution. *NPR*, 19 Nov. 2021.
- [16] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. Sgxpectre: Stealing intel secrets from SGX enclaves via speculative execution. In *IEEE EuroS&P*, 2019.
- [17] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: Toward a secure voting system. In *IEEE S&P*, 2008.
- [18] Coinbase. What is a crypto airdrop? <https://www.coinbase.com/learn/crypto-basics/what-is-a-crypto-airdrop>. Accessed: 2024-11-14.
- [19] Coinbase. What is a crypto dusting attack and how to avoid it. <https://www.coinbase.com/learn/tips-and-tutorials/what-is-crypto-dusting-attack-and-how-to-avoid-it>. Accessed: 2024-10-24.
- [20] CoinMarketCap. Token Lockup. <https://coinmarketcap.com/academy/glossary/token-lockup>. Accessed: 2024-11-13.
- [21] Ethereum.org Contributors. Decentralized autonomous organizations (DAOs). <https://ethereum.org/en/dao/>. Accessed: 2024-11-13.
- [22] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Paper 2016/086, 2016.
- [23] Philip Daian, Tyler Kell, Ian Miers, and Ari Juels. On-chain vote buying and the rise of dark DAOs. Hacking Distributed blog post, <https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/>, 2018.
- [24] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, 2006.
- [25] dYdX. v3 updated fee schedule. <https://dydx.exchange/blog/v3-updated-fee-schedule>, 22 Sept. 2023.

- [26] Stefan Dziembowski, Sebastian Faust, and Tomasz Lizurej. Individual cryptography. In *CRYPTO*, pages 547–579, 2023.
- [27] Team EigenLayer. Eigenlayer: The restaking collective. <https://docs.eigenlayer.xyz/overview/whitepaper>, 2024.
- [28] Victor Eluke. DeFi bribes: The battle for liquidity supremacy. Blocverse Blog, 25 Sept. 2023.
- [29] Ulaş Erdoğan and Doğan Alpaslan. Eip-7212: Precompiled for secp256r1 curve support [draft]. <https://eip.info/eip/7212>, 2023.
- [30] Ethereum Foundation. Light clients. <https://ethereum.org/en/developers/docs/nodes-and-clients/light-clients/>, August 2024. Accessed: 2024-10-10.
- [31] Shufan Fei, Zheng Yan, Wenyi Ding, and Haomeng Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Surv.*, 54(6), July 2021.
- [32] V. Buterin G. Weyl, P. Ohlhaver. Decentralized society: finding web3’s soul. SSRN, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4105763, 11 May 2022.
- [33] Krzysztof Gogol, Yaron Velner, Benjamin Kraner, and Claudio Tessone. Sok: Liquid staking tokens (LSTs). *arXiv preprint arXiv:2404.00644*, 2024.
- [34] Stephane Gosselin and Ankit Chiplunkar. Introducing onebalance. Frontier Research blog post, <https://frontier.tech/onebalance>, 2024. Accessed: 2024-10-23.
- [35] Lachlan Gunn, Ricardo Vieitez Parra, and N Asokan. Circumventing cryptographic deniability with remote attestation. In *Privacy Enhancing Technologies Symposium*, 2019.
- [36] Etherscan Information Center. Understanding an Ethereum Transaction. <https://info.etherscan.com/understanding-an-ethereum-transaction>. Accessed: 2024-11-13.
- [37] Kazuharu Itakura. A public-key cryptosystem suitable for digital multisignature. *NEC research and development*, 71:1–8, 1983.
- [38] Nerla Jean-Louis, Yunqi Li, Yan Ji, Harjasleen Malvai, Thomas Yurek, Sylvain Bellamare, and Andrew Miller. SGXonered: Finding (and partially fixing) privacy flaws in TEE-based smart contract platforms without breaking the tee. Cryptology ePrint Archive, Paper 2023/378, 2023. <https://eprint.iacr.org/2023/378>.
- [39] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES*, pages 61–70, 2005.
- [40] Ari Juels, Ahmed Kosba, and Elaine Shi. The ring of Gyges: Investigating the future of criminal smart contracts. In *ACM CCS*, 2016.
- [41] Mahimna Kelkar, Kushal Babel, Philip Daian, James Austgen, Vitalik Buterin, and Ari Juels. Complete knowledge: Preventing encumbrance of cryptographic secrets. In *ACM CCS*, 2024.
- [42] Mahimna Kelkar, Yunqi Li, Nerla Jean-Louis, Carolina Ortega Pérez, Kushal Babel, Andrew Miller, and Ari Juels. The sting framework: Proving the existence of superclass adversaries. Cryptology ePrint Archive, Paper 2024/1676, 2024.
- [43] DaeYoub Kim and Jihoon Lee. Blacklist vs. whitelist-based ransomware solutions. *IEEE Consumer Electronics Magazine*, 9(3):22–28, 2020.
- [44] Offchain Labs. Arbitrum FAQ. <https://docs.arbitrum.io/learn-more/faq>. Accessed: 2024-10-10.
- [45] Yuga Labs. Apes come home. <https://news.yuga.com/apes-come-home>, Feb 2024.
- [46] Yuga Labs. ApeFest FAQ. Yuga Labs News, <https://apefest.com/faq>, 21 Feb. 2024.
- [47] Steven P Lalley and E Glen Weyl. Quadratic voting: How mechanism design can radicalize democracy. In *AEA Papers and Proceedings*, 2018.
- [48] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys ’20*, New York, NY, USA, 2020. Association for Computing Machinery.
- [49] Ruija Li, Qin Wang, Xinrui Zhang, Qi Wang, David Galindo, and Yang Xiang. An offline delegatable cryptocurrency system. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2021.
- [50] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In *SOSP*, pages 63–79, 2019.
- [51] Thomas Lloyd, Daire O’Broin, and Martin Harrigan. Emergent outcomes of the vetoken model. In *2023 IEEE international conference on omni-layer intelligent systems (COINS)*, pages 1–6. IEEE, 2023.
- [52] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security*, pages 1553–1570, 2020.
- [53] Sinisa Matetic, Moritz Schneider, Andrew Miller, Ari Juels, and Srdjan Capkun. Delegatee: Brokered delegation using trusted execution environments. In *27th USENIX Security Symposium (USENIX Security)*, pages 1387–1403, 2018.
- [54] Paulo Mateus and Serge Vaudenay. On tamper-resistance from a theoretical viewpoint. In *CHES*, pages 411–428, 2009.
- [55] Paulo Mateus and Serge Vaudenay. On tamper-resistance from a theoretical viewpoint. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 411–428, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [56] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® software guard extensions (Intel® SGX) support for dynamic memory management inside an enclave. In *HASP*, pages 1–9, 2016.
- [57] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP*, page 10, 2013.
- [58] Malte Möser, Rainer Böhme, and Dominic Breuker. Towards risk scoring of bitcoin transactions. In *Financial Cryptography and Data Security*, 2014.
- [59] Antonio Muñoz, Ruben Ríos, Rodrigo Román, and Javier López. A survey on the (in)security of trusted execution environments. *Computers & Security*, 129:103180, 2023.
- [60] Avalanche Network. Block finalization stall incident report for Avalanche. <https://status.avax.network/incidents/qmx0s7zk6gkm>, 2024. Accessed: 2024-10-12.
- [61] Solana Network. mb-020624 incident report for Solana. <https://status.solana.com/incidents/n5kcg8dl9pj>, 2024. Accessed: 2024-10-12.
- [62] Margaux Nijkerk. Ethereum mainnet was unable to fully finalize transactions for 25 minutes. *CoinDesk*, 11 May 2023. Accessed: 2024-10-10.
- [63] NVIDIA. Nvidia confidential computing solutions. <https://www.nvidia.com/en-us/data-center/solutions/confidential-computing/>, 2024. Accessed: 2024-10-06.
- [64] Oasis Labs. Oasis protocol: Privacy-enabled blockchain platform. <https://oasisprotocol.org/>, 2024. Accessed: 2024-10-06.
- [65] Office of Foreign Assets Control. Do OFAC reporting obligations apply to “dusting” transactions? <https://ofac.treasury.gov/faqs/1078>, 2022.
- [66] Puja Ohlhaver, E Glen Weyl, and Vitalik Buterin. Decentralized society: Finding web3’s soul. Available at SSRN 4105763, 2022.
- [67] David O’Neill. Unlocking the power of data with arm cca. <https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/unlocking-the-power-of-data-with-arm-cca>, 2024. Accessed: 2024-10-06.

- [68] OpenSea. What is NFT ticketing? <https://opensea.io/learn/nft/what-is-nft-ticketing>. Accessed: 2024-11-13.
- [69] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In *EUROCRYPT*, pages 260–289, 2017.
- [70] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake under scrutiny. In *ACM SAC*, SAC ’23, page 212–221. Association for Computing Machinery, 2023.
- [71] Ivan Puddu, Daniele Lain, Moritz Schneider, Elizaveta Tretiakova, Sinisa Matetic, and Srdjan Capkun. Teevil: Identity lease via trusted execution environments. *arXiv preprint arXiv:1903.00449*, 2019.
- [72] Rachel-Rose O’Leary. The ‘Dark DAO’ Threat: Vote Vulnerability Could Undermine Crypto Elections. *CoinDesk*, July 2018. Accessed: 2024-10-23.
- [73] Sandbox. Sandbox player guide. <https://docs.sandbox.game/en/players/player-guide>, Sep 2024.
- [74] Tanuj Surve. How and where to view crypto transaction histories. Cointelegraph, <https://cointelegraph.com/news/how-and-where-to-view-crypto-transaction-histories>, 6 May 2024.
- [75] Martin Holst Swende and Nick Johnson. Erc-191: Signed data standard. Ethereum Improvement Proposals, January 2016. <https://eips.ethereum.org/EIPS/eip-191>.
- [76] Eli Tan. “Do you believe in second chances?”: Another DAO is raising funds to buy a copy of the US Constitution. *CoinDesk*, 7 Dec. 2022.
- [77] Dalton Cézane Gomes Valadares, Angelo Perkusich, Aldenor Falcão Martins, Mohammed B. M. Kamel, and Chris Seline. Privacy-preserving blockchain technologies. *Sensors*, 23(16), 2023.
- [78] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SG Axe: How SGX fails in practice, 2020. <https://sgaxe.com/files/SGAxe.pdf>.
- [79] Stephan Van Schaik, Alex Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Daniel Genkin, Andrew Miller, Eyal Ronen, Yuval Yarom, and Christina Garman. Sok: Sgx.fail: How stuff gets exposed. In *IEEE S&P (SP)*, pages 4143–4162, 2024.
- [80] Veecon. Veecon 2024. <https://veecon.co/tickets>. Accessed: 2024-10-23.
- [81] Friedhelm Victor and Andrea Marie Weintraud. Detecting and quantifying wash trading on decentralized cryptocurrency exchanges. In *WWW*, page 23–32, New York, NY, USA, 2021. Association for Computing Machinery.
- [82] Vitalik Buterin. Paths toward single-slot finality. https://notes.ethereum.org/@vbuterin/single_slot_finality, 2023.
- [83] Wikipedia contributors. ConstitutionDAO — Wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=ConstitutionDAO&oldid=1225513611>, 2024. Accessed: 2024-11-13.
- [84] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *ACM CCS*, 2022.
- [85] Ying Yan, Changzheng Wei, Xuepeng Guo, Xuming Lu, Xiaofu Zheng, Qi Liu, Chenhui Zhou, Xuyang Song, Boran Zhao, Hui Zhang, and Guofei Jiang. Confidentiality support over financial grade consortium blockchain. In *ACM SIGMOD*, 2020.
- [86] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks. In *IEEE S&P*, 2023.
- [87] Zuza Zuber. Snapshot - the technical architecture. https://snapshot.mirror.xyz/C3bd5Tl3XEbPRt_FIBB97fkhwXk-81w59CMcoofWUk, 2023.

Appendix A. Other Applications

This section describes additional applications for Liquefaction in the blockchain ecosystem.

A.1. Governance

Quadratic voting. Quadratic voting gives greater per-token voting weight to small account holders, thereby reducing the voting power of whales. To win elections under quadratic voting, it is most effective to have a large number of small account holders vote in favor of a proposal. While whales splitting their tokens across Sybil accounts is a known attack on quadratic voting systems, identity systems alone (ensuring one user, one account to negate Sybils) are insufficient in the presence of key encumbrance. A single whale could *square* its voting power by controlling voting the same number of tokens from a set of other identity-verified encumbered wallets owned by other people.

Multisigs. *Multisignature wallets*, which require signatures from multiple individuals to perform an action, are often used to manage treasuries. Liquefaction could be used to steal treasury funds using a secret coordination scheme among the members of the multisig. To do so, individual signatories secretly signal to a publicly known encumbrance policy their willingness to participate in a theft, and if the signing threshold is met among the conspirators, the encumbrance policy aggregates their signatures and executes the theft. A more prosocial use of Liquefaction in multisigs is to enrich the access structure: for example, a signatory might delegate approving low-risk transactions to an assistant.

A.2. Reputation

Transaction history. Transaction histories are often used to assign risk ratings to blockchain addresses. This allows cryptocurrency exchanges to discriminate against and deny service to users associated with high-risk transaction activity. Since key encumbrance allows the ownership of private keys to be rented or sold, the owner of a blockchain address judged as high-risk might purchase ownership of assets held in a low-risk, key-encumbered wallet and use the low-risk wallet to deposit to a cryptocurrency exchange with strict risk requirements. Transaction risk is therefore liquefied.

Airdrop rights. Airdrops are a common means of distributing tokens for an emerging protocol. In an airdrop, users receive tokens to their blockchain addresses according to their use of the protocol, community contributions, and other signs of early support. Using key encumbrance, markets can trade encumbered private keys speculated to be eligible for future airdrops. The benefit of trading private keys directly is that the particular form of the airdrop (e.g., knowledge of the airdrop token address or actions required to claim it) need not be known at the time of sale. This allows for flexible redemption and strong guarantees to the key purchaser. By

selling rights to a potential future airdrop, early system users can get liquidity earlier by selling their role as potential future token holders to speculators who specialize in selling those tokens at the best moment.

Loyalty points. In many online services, rewards, special discounts, or exclusive access are granted to users with a history of patronage. For example, certain decentralized exchanges reward users staking their tokens with trading discounts [1], [25]. Using key encumbrance, a group can coordinate to split the cost of a single key which acquires the ongoing rewards and discounts for the whole group, significantly reducing the amount paid to the service while simultaneously making the shared account appear to be a highly active user. Alternatively, a single user can resell discounted access to the service.

Wash trading. In order to create interest in a particular token, a person or group might engage in illegal *wash trading* to increase the token’s trading volume artificially. Key encumbrance can make wash trading harder to detect, since a wash trader could rent access to any number of key-encumbered accounts, each of which may have vastly different provenance and reputation than the others. Given a catalog of enough wallets, the wash trader would be able to evade detection measured by strongly connected components of transfers (as done by [81]).

Similarly, one might disrupt NFT provenance by having a celebrity hold the NFT in his or her encumbered wallet to boost the value of the NFT without transferring ownership of the asset.

A.3. Privacy

Trading assets held in others’ wallets. Through Liquefaction, one can construct a privacy system that facilitates trading ownership of assets held across multiple accounts. Liquefaction would allow users to create a derivative token that gives access to the full, aggregate balance of an asset held across a series of participating accounts. The derivative would allow users to operate with the full weight of assets held across accounts, for instance to meet the minimum holding requirement to participate in a service, while retaining the privacy of their individual accounts.

Private DAO treasuries. Blockchain transparency can sometimes be problematic for DAOs, as illustrated by ConstitutionDAO’s attempt to bid over \$40 million at auction on an original first-run copy of the U.S. Constitution [15]. The DAO’s smart contract balance publicly revealed exactly how much money ConstitutionDAO raised, which in part explains how it was narrowly outbid by a hedge fund manager [76].

Liquefaction enables DAOs to raise money through decentralized treasuries. Using key encumbrance, donations need not even leave the (encumbered) accounts of the donors until they need to be spent—the Liquefaction wallet simply reassigned the ownership of the donated funds to the DAO, which can spend the funds as needed.

Secret contract payments. Typically, smart contracts explicitly authorize transfers, making them publicly viewable.

Liquefaction can be used to facilitate secret payments. A solicitor can put the bounty in an encumbered wallet whose public address is never revealed. The encumbrance contract can prove that there exists a bounty attached to some certain task. Once a user completes this task, the encumbered wallet will reveal the secret key to these funds to the user.

Privacy Pools: Hidden sub-pools. *Privacy Pools* [14] is a privacy-enhancing protocol that uses a smart contract to offer transaction privacy, much like a mixer such as Tornado Cash. In contrast to a standard mixer, though, Privacy Pools allows users to dissociate their deposits from undesirable addresses and thus from illicit activity. The idea is that the mixing is user-configurable: A user may choose the anonymity set (called an *association set*) that provides privacy for her transaction.

Briefly, on withdrawing funds, a user generates a zero-knowledge proof that the corresponding deposit belongs to a user-selected association set of previous deposits. Users may generate new proofs with reduced association sets as desired, enabling them to exclude deposits from suspect addresses to avoid tainting their own withdrawn funds.

This feature, however, presents a privacy risk: a depositor may (rightly or wrongly) be subjected to exclusion from other users’ association sets and therefore lose the ability to transact privately.

Liquefaction offers a way to secretly enforce a Tornado Cash-like non-exclusion guarantee. Users can join a Dark DAO that requires association-set proofs to include the deposits of all members, i.e., protects all members against exclusion. While members of such a Dark DAO would risk tainting their transactions, they might be willing to join in the hope of still benefiting from collective privacy or simply for ideological reasons.

Since this application only requires limiting single-user key use, secret sharing (mentioned in Section 4.4) can be used to prevent key theft even if the TEE gets compromised.

A.4. Token-Gated Ticketing

Some in-person events (e.g., [46], [80]) rely on *token-gated ticketing*: admission to the event depends on an attendee’s ownership of an asset, such as an NFT, digital identity, or other proof of membership. Though these events are usually billed as exclusive to token holders, Liquefaction can undermine this perceived exclusivity. If the asset enabling entry is held in an encumbered wallet, its owner can delegate admission to the event to someone who does not own the asset. That delegated access could then be redistributed directly or sold in a secondary market.

Metaverse ticketing. Several video games and virtual worlds (e.g., [45], [73]) offer features or access exclusively to owners of some NFTs, whose minimum cost of ownership—often referred to as the floor price—may be high. Through Liquefaction, users can rent or lend temporary access to their virtual players without needing to transfer their assets or share a private key. Renting this access offers an entry point for players who want to participate in

the virtual world, if temporarily, without taking on the risk or cost of purchasing the requisite NFT.

A.5. Provenance

Faking theft. Encumbrance can also be used to rent *poor* reputation. A malicious entity known to steal private keys and the funds they hold could create a key-encumbered wallet which allows for collusion, given a small bribe. This wallet would be the main recipient of the entity’s operations, receiving stolen funds from victims’ accounts. However, anyone wishing to fake the theft of their own funds (from a “fake victim” wallet) could bribe the key-encumbered wallet in advance in exchange for the ownership of funds sent from the “fake victim” wallet. Outside observers would not be able to tell if the fake victim really did have its funds stolen by the entity’s operations, but the complicit “victim” would have guaranteed future access to their “stolen” funds from the recipient wallet.

A.6. Interoperability

Overlay Smart Contracts. Since policies in Liquefaction are implemented through smart contracts, Liquefaction enables addresses to function as smart contracts on blockchains which do not support smart contracts natively. After encumbering a blockchain address in a Liquefaction wallet, the required contract logic can be implemented as an encumbrance policy. This capability can be beneficial for interoperability frameworks such as asset bridges. A Liquefaction policy can implement the lock-mint functionality that many non-smart contract blockchains lack, thereby supporting safe bridge constructions such as zkBridge [84].

Appendix B. Liveness Fallback Details

In this section, we explain how our fallback trigger is implemented and updates are propagated.

Implementing the fallback trigger. Figure 9 shows the design of Liquefaction’s fallback system, which uses a smart contract FB to generate a proof that the fallback trigger has occurred, i.e., an outage of duration at least T has occurred. FB executes not on the primary blockchain (Oasis), but on a separate, high-reliability blockchain, i.e., one whose liveness we assume even if the primary blockchain fails. For this purpose, we use Ethereum, which has experienced only rare, brief outages over its lifetime affecting block finality [62]. FB incentivizes users to generate timely proofs through a challenge mechanism. It is initialized with a (large) bounty $\$b$. This bounty may be a one-time deposit per a Liquefaction instance, as the fallback trigger should be invoked at most once during the lifetime of an instance. On the primary blockchain, support for FB exists in the form of a special *sentinel* wallet that digitally signs pings off-chain at no cost.

In the case of an outage on the primary blockchain, at time τ , any user U_c may send a transaction to FB, along with a deposit $\$d$, that issues a liveness *challenge*. Any user may subsequently ping the sentinel wallet, obtaining a signed, timestamped *response* message r . The first user U_r to relay such a message r to FB before time $\tau+T$ is awarded $\$d$, which U_c loses—the fallback trigger proof has failed. Otherwise, if $\tau+T$ passes without a valid response, U_c receives bounty $\$b$ and the transition to the fallback system is initiated. (See below for details.)

Note that the deposit $\$d$ can be relatively small—small enough that forfeiting it in the case of mistaken judgment about an outage isn’t punitive. Users already have an incentive to ensure that the fallback trigger functions correctly as a way to secure their funds, but the deposit $\$d$ additionally protects against denial-of-service attacks on FB. It makes the cost of submitting many challenge transactions prohibitively expensive and covers the cost of response transactions.

Given the simple fallback key-encumbrance policy we consider here, our fallback system itself has a straightforward and practical design. Rather than requiring a full-blown blockchain, it relies on two sets of services: A fallback committee of n TEE nodes and a set of m decentralized storage repositories.

During normal operation of Liquefaction, *encrypted* copies of system state (both policies and keys) are maintained in each of the decentralized storage repositories. (We give details below.) Copies are encrypted under a public key pk_{FB} that is distributed using t -out-of- n secret-sharing among nodes in the fallback committee upon system setup.

When FB successfully proves the occurrence of a fallback trigger, committee members launch a fallback application in a local TEE. This application runs a light client [30]. Upon input of recent Ethereum blocks, it checks the state of FB to determine whether a valid proof has been created. If so, the committee fetches Liquefaction state from the decentralized storage repositories. The latest copy is (threshold) decrypted by the committee and instantiated in their TEEs. The application then executes the fallback key-encumbrance policy.

In the case of the example fallback key-encumbrance policy we consider—removal of encumbrance—the user who generated a private key sk can export sk from Liquefaction. This simple policy requires no communication among nodes in the fallback committee once the fallback application is launched.

Propagating updates to the fallback system. Liquefaction must propagate updates to the fallback system periodically to ensure storage of fresh system state. The creation of new wallets or decrease of privilege over the encumbered keys must be (encrypted and) propagated to the decentralized storage repositories before the user’s actions are considered finalized. For efficiency, updates that increase privilege over existing keys can be deferred and propagated in batches as the soundness of encumbrance is still preserved in case of fallback. In case of a liveness failure, however, some updates may be missing from the fallback network.

Trust model. Our security assumptions for the fallback committee and decentralized storage systems are lightweight. We require the availability of at least t nodes upon invocation of a fallback. (They need not be persistently online, but should be launched in response to the failure of the primary blockchain—even if manually.) We additionally require only that one decentralized storage repository provide correct and available data storage.

Appendix C. CK Proofs Details

We now expand on our discussion from Section 5.2 and give additional details about proofs of Complete Knowledge.

The most practical protocol for a proof of CK itself involves a TEE application. The application outputs sk locally to the host owner (e.g., displays it on a screen). It can then generate an attestation—on the corresponding public key pk (or corresponding address A)—that such output has taken place. This constitutes a proof of CK. We illustrate how CK might be applied in practice in Example 4.

Example 4 (Airdrop with CK). *Airdrops are executed through smart contracts that control the airdropped assets. To prevent encumbrance of airdropped tokens, such a smart contract can include an:*

- *On-chain CK verification function that checks a proof of CK sent by a given address A and a*
- *Registry of addresses that have passed CK verification.*

The contract then delivers tokens during an airdrop only to addresses present in the registry and therefore known to be unable to encumber tokens.

As detailed in [41], proofs of CK created from mobile device TEEs are practical today on Ethereum, if expensive (about 1.5 million gas). The cost may drop considerably with native support for the elliptic curves used in attestation signatures [29]. Additionally, CK proofs are relatively inexpensive on layer-2 blockchains. We note that while existing mobile-TEE attestations are software based, hardware support is now emerging [67].

Appendix D. Ethical Considerations

Our Liquefaction system has both prosocial and anti-social uses, which we discuss in Section 6. Liquefaction allows accounts to make stronger commitments about their behavior, participate in applications with enhanced privacy, mitigate dusting attacks, and reduce transaction costs. On the other hand, by overturning the SEAO assumption, Liquefaction breaks existing norms that many blockchain application mechanisms rely on.

In traditional security research, it is customary to initiate responsible disclosure for any new software vulnerabilities which are found. However, the disclosure process for open, permissionless systems differs from reporting to traditional software businesses. The lack of central points of contact

and the immutability of smart contracts makes complete, private disclosure impossible – particularly for systemic issues like the SEAO assumption, which affect wide swaths of the blockchain ecosystem rather than specific organizations.

We have adopted a twofold approach to responsible disclosure. First, we privately communicated our findings with major stakeholders, particularly DAOs (Lido, Optimism, and Arbitrum), which we felt were most at risk of attack. We also communicated with vendors such as Oasis and Flashbots, whose platforms could effectively support the hosting of Liquefaction wallets and policies. Second, we have open-sourced the code of our Liquefaction wallet and encumbrance policies. We feel it is important to have a clear demonstration of the practicality of Liquefaction so that the community can understand its long-term implications, especially since practitioners have previously dismissed related works as purely theoretical [72].

We also stress that, as explained in Section 5.2, proofs of complete knowledge (CK) mitigate Liquefaction threats to systems that adopt them. We hope that our work motivates, and leads to, the development of more practical CK tools which constitute a proactive mitigation.