Privacy Preserving Compliance

Joss Duff

November 13, 2025

1 Abstract

[[[Joss: A better first sentence is either the "privacy and compliance are classically opposed" or the negative outcomes of privacy without compliance and compliance systems without privacy]]] We present a framework for a composable privacy-preserving compliance ecosystem. In this framework, regulatory bodies publish definitions of compliance over any on-chain data: non-association checks, address-history assertions, protocol-interaction requirements, etc. Applications select appropriate compliance definitions and require users to prove compliance on-chain before onboarding to the application. This fulfills dual goals: regulators pushing for compliance and applications/users seeking precise compliance definitions, all without compromising privacy. [[[HFK: those are constituencies, not goals. Goal is more like "facilitating compliance while maximizing privacy; encouraging publication of clearly stated regulations"]]] <==

We present the fundamental building blocks of *compliance definitions* and *constraints*, and discuss how to leverage them to create a novel *domain specific language* (DSL) that intelligently minimizes the computation necessary to prove compliance in this framework. [[[HFK: took out the future work thing here - it suggests the best stuff comes only later. The thesis does in fact "discuss."]]] <==

2 Background and Definitions

Background information on blockchains, privacy systems on blockchains, zero knowledge proofs, and our working definition of "compliance". People familiar with blockchain and ZK can skip this section.

2.1 Blockchain Background

Blockchains are fundamentally data-storage systems with unique properties. Most importantly, data stored on a blockchain (*"on-chain"*) is *immutable*: its history cannot be changed. This property is achieved by allowing anyone[1] around the world to participate in consensus and validation of the blockchain's state by running a node.

The first blockchain, Bitcoin, was created in 2008 and proposed a trustless ledger of transaction histories [39]. Ethereum was created in 2013 and improved on Bitcoin by allowing trustless execution of Turing-complete logic on-chain, referred to as *smart contracts* [12]. This allowed for more complex applications like marketplaces and lending platforms to exist in a decentralized, permissionless, and immutable setting.

Every transaction on a blockchain has a nonstatic monetary cost paid to the block-producing node as an incentive to include the transaction in the next write. Some blockchains with smart contract support, like Ethereum, include an additional transaction cost driven by the complexity of the smart contract logic being executed.

Users on blockchains are represented by a public-key address. The user signs messages and transactions with the associated private key. Users can have multiple public-key addresses. It is impossible to reveal who is associated with a public key until the user reveals their identity. [[[HFK: either directly, or indirectly by <==

[1] There is a small class of "permissioned" blockchains that only allow whitelisted individuals to participate in consensus and validation.

accessing the public key via an off-chain device or service.]]]

A centralized exchange (*"CEX"*) is a company that allows users to trade traditional currency for cryptocurrency. They often require the user to connect their bank account and supply regulatory documentation such as identification cards and proof of residency. This is a common method for acquiring cryptocurrency. CEXs are *custodial* which means they control the address (they know the private key) that owns the cryptocurrency on-chain and allow the user to trade or transfer it. They allow users to send their cryptocurrency from the CEX account to the users' own on-chain address. CEX accounts are not addresses. Rather, the CEX keeps its own addresses and the user's holdings are only an accounting record in the CEX's books.

[[[HFK: I know we have CEXs here for proof of source of funds. But for the intro, I'd add a DEX paragraph.]]] <==

2.2 Privacy in the Blockchain Setting

All data on blockchains are public by default unless the chain supports special privacy features; for example, Aztec Network supports private state. There are also some enterprise permissioned blockchains that restrict visibility levels. However, the vast majority of chains are public by default, including Ethereum and Bitcoin.

For public blockchains, every user's entire transaction history is exposed to the world. This is obviously undesirable for consumers. Most people don't want their purchase or subscription history visible to the entire world. Many applications and protocols exist to provide users varying degrees of privacy.

Blockchain privacy protocols fall into two broad categories:

- *App-layer privacy*: Applications on top of existing blockchains.
- *Base-layer privacy*: Entirely new blockchain architectures with privacy as a core identity.

2.2.1 App-layer privacy

CoinJoin was the first permissionless app-layer transaction mixer [35]. CoinJoin was on Bitcoin and combined multiple input UTXOs from different parties into multiple outputs, obscuring which inputs correspond to which outputs. Tornado Cash was the first transaction mixer to use ZK proofs [43], and saw massive popularity [23]. The black-hat group Lazarus user Torando Cash to launder stolen funds and the Tornado Cash developers were put on trial [57]. Railgun is a transaction mixer that focuses on compliance [49]. Privacy Pools is a recent transaction mixer that also focuses on compliance with Association Sets [14]. Ultra-Anon is an experimental privacy token that has the property of *plausible deniability*: it isn't revealed if users utilized privacy [25]. Stealth addresses are ephemeral public keys that can receive any tokens while hiding the receiver address [13]. zkBob is a wallet that has private transfer for certain stablecoins [63]. Webb Protocol requires specific interface contracts to be deployed on multiple chains and results in a cross-chain anonymity set [53]. Worm is an application implementation of "EIP-7503: Zero-Knowledge Wormholes" [31], which is another transactional privacy solution with the property of plausible deniability.

2.2.2 Base-layer privacy

Zcash is a privacy ledger based on Bitcoin's Unspent Transaction Output (UTXO) model. Zcash is implemented from the 2014 Zerocash paper [8] and is one of the earliest private blockchains. Monero is another of the earliest private blockchains and similarly leverages the UTXO system. Monero is based on the 2013 CryptoNote paper [58]. Aztec is a general purpose blockchain that allows for private state with smart contract support [61]. Aleo [2], based on the 2018 paper "ZEXE: Enabling Decentralized Private Computation" [10], is also a blockchain with private state and smart contracts. The Miden blockchain supports the private state similarly [46]. Penumbra is a private network and decentralized exchange for the Cosmos ecosystem [42]. Arc [34] and Tempo [54], founded by Circle and Stripe, respectively, are stablecoin-focused blockchains with some degrees of transaction privacy.

TODO: mimblewimble, secret network, Iron fish?, neptune cash, anoma, payy, [[[HFK: not sure if this <==

is too early to say, but Aztec is not really base-layer since it is L2. We might want to let that go in an intro section, but my suggestion that is that we put in a footnote at least regarding L2]]]

2.2.3 Custodial mixers

Custodial mixers are transaction privacy solutions that require users to send funds to a trusted intermediary who then redistributes the funds. This accomplishes the privacy goal of breaking the traceability between the input and output transaction, but custodial mixers have the major weakness that operators could steal funds or keep records. This is fundamentally weaker privacy because two parties have knowledge of a user's transaction history, rather than just the user themselves [[[the plural gender correctness nails us badly here since it suggests two users. Here is a way to work about that without the he/she approach: "because that intermediary has knowledge of a user's transaction history, rather than that history being private and held only by the user"]]] . Centralized exchanges can be used as custodial mixers. <==

[[[Joss: TODO some custodial mixer examples. 2 is enough]]] <==

[[[HFK: While this is not a legal document, we might note that in some jurisdiction, the intermediary could be subject to government subpoena]]] <==

2.2.4 Anonymity Set

An *anonymity set* is the group of entities among which an observer cannot reliably identify which specific entity possesses a particular attribute or performed a particular action [45, 48]. Anonymity scales with set size: the larger the anonymity set, the less identifiable each entity is [24].

For transaction-mixer applications on blockchains, any address that has deposited can withdraw, but it is never revealed which depositing addresses have an associated withdrawal. Each withdrawal could have come from any of the depositing addresses. Therefore, the anonymity set of any given withdrawal from the mixer is the set of all addresses that have deposited in the mixer. This is oversimplified, and in practice anonymity sets can be reduced with chain analysis [30, 37]. Anonymity here is referring to the obfuscation of the link between a depositing and withdrawing address.

2.3 Zero Knowledge Proof

Zero Knowledge (ZK) proofs are a cryptographic technique that allows a party to prove specific claims about a computation without revealing any additional information about the computation. The origins of this concept predate blockchain [7, 27], but it was described elegantly and simply by Vitalik Buterin in [14] as follows:

> The "claim" proven by a ZK proof is expressed as a type of program that is often called a *circuit*. Mathematically, it suffices to think of it as a function $f(x, w) \rightarrow \{\text{True}, \text{False}\}$, where x is the public input, w is the private input, and $f(\cdot)$ is the function being computed. A ZK proof proves that, for a given x known both by the prover and the verifier, the prover knows a w such that $f(x, w)$ returns True.

SNARKs are a specific type of ZK proof that has the unique property of *succinctness*: The proof size doesn't grow with the size of the computation, and the proof can be verified very quickly relative to the computation [44].[2]

A drawback with some SNARK schemes, such as Groth16 [28], is that they rely on a *trusted setup*: a time-intensive ceremony involving multiple participants. A trusted setup needs to be performed once per circuit [33] for Groth16 and some other SNARKs.

[2]The SNARK acronym comes from Succinct, Non-interactive ARguments of Knowledge. The non-interactive term serves to distinguish SNARKs from the interactive proofs between a prover and a verifier that are often used to illustrate the concept of zero knowledge for a general audience.

A type of SNARK proofs, PLONK, mitigates this drawback by allowing trusted setups to be re-used across circuits. With PLONK proofs, it is not necessary to initiate a new trusted setup for every new circuit [26].

2.4 Compliance

Compliance is referred to as the goals and properties of a system to attempt to adhere to a law or policy. Compliance isn't immunity from prosecution, it is a best effort to prove legality of actions. For example, US companies have systems in place to prevent money laundering in order to be compliant with the US Bank Secrecy Act (BSA) - also referred to as Anti-Money Laundering (AML) [56]. A typical effort to comply with AML in the blockchain space is blocking addresses associated with sanctioned individuals from interacting with a protocol.

A goal of this framework is to be flexible enough to accommodate compliance definitions over all aspects of a transaction: the payer, the payee, the content of the transaction, and even the actors' transaction histories. Flexibility results in the ability to express any definition of compliance.

3 Problem

Blockchains are fundamentally pseudonymous and public: user identities are obfuscated behind addresses, but all activity between addresses is public by default. In practice, entities are able to associate identities to addresses when on-ramping traditional funds to cryptocurrency and therefore track individuals' flows of funds on-chain. In order for cryptocurrency to be practical for consumers, it must at the very least have the same level of privacy as traditional finance. In traditional finance typically two entities know information about a user's funds; the user themself and the exchange that is custodian of the funds. In blockchain settings, all entities can track user funds and activity. Neither consumers nor corporations will want their entire trading, purchasing, or subscription histories publicly available.

Privacy is a growing field in the blockchain space [43, 31, 9, 25], but is plagued by malicious actors using privacy to launder funds. In an effort to reduce money laundering, regulatory bodies are pursuing legal action against developers of privacy protocols [32, 57, 20]. This introduces the need to explicitly meet regulatory compliance inside private on-chain applications.

Current approaches to blockchain compliance face an inherent contradiction: privacy requires that only the individual knows certain information, while compliance traditionally requires revealing information to prove compliance criteria. This tension has led to two unsatisfactory outcomes:

1. Privacy systems without compliance attract illicit actors and face legal challenges.
2. Compliance systems without privacy expose all transaction histories publicly, making them unacceptable for consumer and commercial adoption.

The goal is to maximize privacy while maintaining the ability to prove compliance. Current research attempting to bridge privacy and compliance have proposed solutions with strict domain limitations such as a new blockchain architecture, specific on-chain protocols, or privacy violations. We investigate a framework that allows for compliance in any blockchain smart contract environment and in any privacy protocol without compromising privacy provided by the protocol.

3.1 Industry Interest

The two privacy protocols currently operational that have some measure of compliance are Privacy Pools and Railgun. Both have received a great deal of interest from users; Railgun was launched in 2022 and reached $131 Million (USD) deposited [22] as of November 2025. Privacy Pools launched in March of 2025 and reached $3.3 Million (USD) deposited [21] as of November 2025. [[[HFK: update these figures in the final thesis. I put dates in there now to protect against "currently" surviving into the final version.]]] The <==

users currently in the blockchain space desire both compliance and privacy. See section 5 for a comparison of both to our framework.

Large traditional financial corporations are still yet to integrate significantly with blockchains. In a 2025 survey, 300 major financial firms were asked what factors keep them from becoming more involved in blockchain technology. In this multiple choice question, 52% cited lack of regulatory certainty and 50% cited lack of privacy as factors [41]. We see it is not just users, but also institutions who desire compliance and privacy. In fact, this problem is a leading blocker for these institutions. Solving this would result in a significant amount of capital being allocated on-chain.

In 2024, $2.2 billion (USD) was stolen in crypto hacks and exploits [55]. Tracing stolen funds from all hacks is too large a task, so auditors prioritize only the largest exploits. Our framework provides strong **proactive** compliance guarantees. Non-compliant actors are stopped from interaction with compliant applications. Compare this to retroactive compliance measures where non-compliant actors are able to interact with private protocols who keep a history of all private transactions. Not only does this create more work for auditors trying to trace non-compliant individuals, it also introduces the risk of leaking compliant user's private information.

[[[HFK: Here we can go beyond financial stuff. Supply chain is a good example here. It can help a B2B supply chain run on a public L1 underpinning. Also within a permissioned system, it can reduce the degree of central control by the permissioning agent. The concept could fit in the Oracle framework – or at least Mark Rakhmilevich saw some merit to it.]]] <==

4 Privacy Preserving Compliance

It is necessary to introduce a stricter definition of *privacy-preserving compliance.* A compliance system is privacy preserving if it has *verifiable compliance*, *no deanonymization*, and inherits the *maximum theoretical anonymity* of its adjacent privacy system. Our framework maintains these properties and therefore meets the criteria of a privacy-preserving compliance system. The definition of privacy-preserving compliance aims to provide a path for compliance without compromising the benefits of having a privacy system on-chain.

No deanonymization: *deanonymization* refers to revealing private information about an address, like balances or transaction history. No entity should have the ability to deanonymize user data in an on-chain privacy system[3].

Any protocol that is able to view keys of private user data has deanonymization. The ability for a third party to reveal a user's private details is fundamentally weaker privacy, and unnecessary. We can guarantee that only users can reveal their private transactions, and do so without compromising compliance. In fact, we provide stronger compliance guarantees too. Viewing keys are used for retroactive compliance checks, and our framework has proactive compliance: non-compliant actors are stopped before making their transaction rather than after.

The existence of the ability to deanonymize users is a risk vector to users and compromises trustlessness, as highlighted in [11]. Individuals with access to private information can and will abuse it. Examples include the case of Twitter employees selling user personal data to Saudi Arabia [40, 18], or US telecom providers illegally selling user data [19]. Additionally, any centralized collection of user data can be hacked. In 2024, data that US telecommunication companies were legally required to collect was hacked [51]. In 2025, personal data held by the Ukrainian government was hacked by Russia [47]. In 2022, 23 terabytes of personal information on 1 billion Chinese residents was leaked from a police database and sold [60].

An on-chain privacy system with deanonymization does not benefit from the properties of being on-chain in the first place. A privacy system where a 3rd party also knows secret user data is as "private" as a

[3]Caveat: A user can always choose to reveal their own private transaction history. There is no way to prevent this and it is acceptable because the user is initiating their own deanonymization. "No deanonymization" refers to third-party deanonymization.

traditional exchange or bank. In order for a on-chain privacy system to provide unique value to users, it must not allow for deanonymization.

Verifiable compliance: Compliance criteria should be publicly viewable and a user's compliance status should be verifiable. For example, a user's compliance status should not be determined behind a third-party API as this runs the risk of the third-party not properly enforcing the compliance definition, introducing risk to all parties involved. This trust assumption could potentially be mitigated with techniques such as *trusted execution environments* (TEEs) that verify that a third-party correctly executed some computation [5]. The approach in this framework is to publish compliance definitions publicly and have users generate their own ZK proofs that can be cryptographically verified.

Maximum theoretical anonymity of compliant privacy: There exists a theoretical maximum anonymity set for any privacy system that aims to be compliant. Assume a privacy system has anonymity set P and a compliance definition has N addresses that meet the criteria. If this compliance definition is enforced for entry and existence in the privacy system's anonymity set, then any address that participates has a maximum anonymity set of $min(|P|, N)$. This is because the user is most identifiable in the smallest set it is a member of. Our framework provides this maximum theoretical anonymity.

5 Related Work and Attempted Solutions

Here, we review other work that attempts to solve the compliant privacy problem. We omit full descriptions of each solution and focus on how our solution is distinguished from prior work by a feature comparison.

[[[Still need to compare this thesis to each item listed below]]] <==

Blockchain specific: Solutions that are blockchain specific propose a new blockchain architecture or would require protocol level changes to existing architectures. Trivially, a solution that requires blockchain architecture changes is incompatible with existing blockchains and cannot benefit from the network effect of robust ecosystems on currently deployed blockchains. New blockchain architectures are always worth exploring, but we maintain that the best solution to the compliant privacy problem isn't limited by blockchain architecture and can be applied to any chain, bringing compliance to any application regardless of which chain they chose. Compliance should be an option available to all current and future applications across all chains.

Solutions that propose a new blockchain architecture include:

- "A privacy-preserving scheme with multi-level regulation compliance for blockchain" [29]
- Aleo [2], based on "ZEXE: Enabling Decentralized Private Computation" [10]
- "Payy: L2 Ethereum ZK Rollup for Private and Compliant Transactions" [36]
- "Arc: An open Layer-1 blockchain purpose-built for stablecoin finance" [34]
- "Tempo: The blockchain designed for payments website" [54]

Deanonymization: For points raised in 4, no entity other than the user themselves should have the ability to reveal private user information. [[[maybe "no entity should be able to reveal private information about any individual user except that particular user." I still worry that "them" here might imply that both the sender and receiver can reveal information. English sucks in that regard.]]] The the ability to deanonymize users is a risk to users, imposes more work for auditors, and is fundamentally weaker privacy (if indeed there is any privacy at all). [[[HFK: I think this is the first time audit shows up. Might be worth noting much earlier that published compliance proofs facilitates low-cost, efficident audits.]]] <== <==

Solutions that have a mechanism for deanonymization include:

- "A privacy-preserving scheme with multi-level regulation compliance for blockchain" [29]
- "REGKYC: Supporting Privacy and Compliance Enforcement for KYC in Blockchains" [62]
- "zkFi: Privacy-Preserving and Regulation Compliant Transactions using Zero Knowledge Proofs" [16]
- Daze [4]
- "ZEBRA: Anonymous Credentials with Practical On-chain Verification and Applications to KYC in DeFi" [50]
- "SeDe: Balancing Blockchain Privacy and Regulatory Compliance by Selective De-Anonymization" [17]
- "A Regulation Scheme Based on the Ciphertext-Policy Hierarchical Attribute-Based Encryption in Bitcoin System" [59]
- "Payy: L2 Ethereum ZK Rollup for Private and Compliant Transactions" [36]
- "Arc: An open Layer-1 blockchain purpose-built for stablecoin finance" [34]

We commend Railgun [49], Privacy Pools [1, 14], and Haze [4] for meeting the definition of Privacy Preserving Compliance without being tied to any specific blockchain architecture.

Our solution differs from Privacy Pools and Haze by being an open framework adoptable by any application and supporting a wide breadth of compliance measures. Privacy Pools and Haze's compliance is limited to only sanction list checks for a single transaction mixer.

Railgun is similar to our solution as it is also a framework. Users opt in to join the Railgun anonymity set with a compliance proof[4] and can briefly reveal parts of their balance to interact with public on-chain applications. On-chain applications do not directly integrate with Railgun and do not benefit from the compliance or privacy guarantees that Railgun users have. Railgun's compliance checks are limited to sanction list checks and some aspects of specific transactions. This is notably more flexible than other compliant privacy solutions.

Railgun's compliance measure ensures that all Railgun users are compliant. The key difference is that our framework allows any application to have the guarantee that all their users are compliant. Additionally, our framework isn't limited to a specific privacy mechanism or application.

[[[Joss: Will need someone to play devils advocate for this Railgun breakdown. This conversation would inspire a better comparison.]]] <==

[[[Joss: comparison TODO: Panther protocol, Iron Fish]]] <==

6 Proposed Solution

We propose an open framework where regulatory bodies publish compliance definitions, applications require users to prove they meet those requirements, and users generate cryptographic proofs demonstrating compliance, all without revealing private information. Users who don't meet the compliance requirements are unable to interact with compliant applications.

Proving compliance to an application has been mentioned in brief in [] and is live in Railgun and Privacy Pools. Our contribution is a standard framework to allow this compliance technique to be used at-scale across many applications and provides a standard interface for regulators to define compliance.

[4]Railgun refers to their compliance proofs as "Proofs of Innocence"

6.1 Framework Approach

The distinguishing feature of our solution relative to other work in this space is that it defines an open framework for use by players in a competitive ecosystem. It is not a new blockchain. It is not a payment system. It is not a new application. Rather, we are creating a framework for established and new-entrant financial providers to take advantage of the power of the emerging blockchain ecosystem by leveraging the features of our framework.

Our framework is unique in its provision of a standard means for regulators to publish requirements openly in a way that enables proof of compliance. Regulation based on constraints and proofs is valuable not only to financial providers, but also to regulators and users. Regulators can check compliance by verification of proofs, which is much more efficient than having to evaluate full execution histories. Users can show transparently that they are in compliance without having to reveal private data.

Our framework is *chain agnostic*: it can be implemented on any chain with smart contracts and capability to verify ZK proofs. Because on-chain compliance pertains to transactions and transaction histories and not to specific proprietary features of a particular blockchain, our framework enables interoperability across platforms that use our framework. This facilitates a competitive ecosystem in which each member of the ecosystem shares a common foundation of privacy and compliance guarantees.

[[[HFK: maybe a comment here about identity – regulators have well-known public keys and use private keys to sign regulations. OR, reputable institutions may translate regulations into constraints and sign those constraints.]]] <==

6.2 Compliance Definition

Compliance definitions are sets of logical rules over blockchain data (e.g., "address not on sanction list," "funds from KYC'd exchange," "no transaction structuring patterns"). Compliance definitions can also contain public parameters, for example a list of sanctioned addresses.

The logical building blocks of the compliance definition over on-chain data are referred to as constraints, and discussed more in Section 6.6. We envision compliance definitions being able to be compiled into constraints, and constraints being able to compile into circuits. This allows compliance to be proven by proving the constraints that make up a definition.

Each published compliance definition can be used by any number of applications, and applications can require multiple compliance definitions.

Compliance definitions are updatable by their author to account for changing regulations. For that reason, each compliance definition should have an associated timestamp.

6.3 Actors

Actors in our framework consist of regulators, applications, and users.

6.3.1 Actor Interaction

1. Regulators create and publish compliance definitions.
2. Applications select relevant compliance definitions based on their regulatory obligations.
3. Users generate zero-knowledge proofs of compliance definitions, demonstrating they meet requirements without revealing transaction histories, balances, or counterparties.
4. Applications verify proofs on-chain before allowing transactions, preventing non-compliant activity cryptographically rather than detecting it afterward.

6.3.2 Regulators

Regulators create definitions of compliance over on-chain data and publish this definition publicly with a signature. The manner in which it is published is an implementation detail, but the compliance definitions need to be publicly accessible. The compliance definition is a set of logical constraints with annotations. Constraints are discussed more in section 6.6. [[[Joss: we should discuss who deploys the verifier contracts and how this interacts with constraints. It is important for applications that the verifier contract doesn't change]]] [[[HFK: compliance is not just over on-chain data. If the constraint says something about the transaction I am about to submit, then there is a non-chain component since my transaction is not yet on-chain.]]] <== <==

We refer to any entity that publishes a compliance definition as a "regulator". Ideally, the entity is associated with the government regulatory agency, but in practice, particularly initially, the publisher is most likely to be a blockchain analytics company. These companies already assume compliance responsibilities, such as maintaining sanction lists and monitoring for suspicious chain activity.

As this is an open framework, there is no restriction on who can publish compliance definitions. Applications need to be careful to select definitions published by reputable sources. Regulators are also responsible for updating compliance definitions, like adding more constraints or updating parameters, with each update having a new timestamp.

6.3.3 Applications

On-chain applications determine which compliance definition(s) suit their regulatory obligations.

Applications integrate with this framework by making a smart contract change to require users to provide a proof of the required compliance at entry-point functions. If the proof does not verify, then the function reverts. After this change, it is guaranteed that all future users of the application are compliant.

For example, the entrypoint function for some application is `deposit(someVars)`. Integration would consist of changing it to:

```
DEPOSIT($someVars, proof$)
1    require $verifierContract$.VERIFY($proof$)
2
3    // Rest of the application logic
```

[[[Joss: point about verifier contract address not changing]]] <==

6.3.4 Users

Users generate ZK proofs of compliance definitions and supply the proofs as an argument when interacting with a compliance requiring application. The constraint system was constructed to intelligently minimize the amount of proof computation needed for users interacting with many compliance requiring applications.

[[[HFK: Users have to be able to state what sort of compliance they claim to prove. Thus, constraints need to have unique publicly-available IDs. the result is that we have a set of identified constraints, each signed by an identified regulator. This database needs to be maintained in a trustless manner and be accessible. Control of this database is another censorship risk.]]] <==

[[[HFK: we need a discussion here – may a new subsubsection – on user identity. Our framework permits users to be fully anonymous while proving compilance, but the particular application may not be set up for total anonymity. If the application keeps records based on users, there will be some sort of identity in the application, either the user's real identity or a pseudonym. There is, of course, some loss of privacy here. Such a loss may be unacceptable in some settings, but in other cases, it may be part of the required business practices. <==

Consider a stablecoin provider. Transactions could be totally public. A better, though not perfect solution is to have deposits and payments of fungible stablecoins with balances held internally in the stablecoin provider's database/contract. The provider could keep only balances, not transaction histories. This would enable some privacy loss and some forms of censorship, depending on how anonymous the user is. Alternatively, the user could hold the balance and prove as needed that the balance is large enough. While the latter may be better from a privacy standpoint, the former is closer to many current system structures and might be a reasonable first step as stablecoins compete on privacy.

I'm running off at the mouth here. Edit as you see fit!!!

Another point: compliance constraints are separate from the user and the application. Proofs can relate not only to compliance constraints but to application-specific features.

]]]

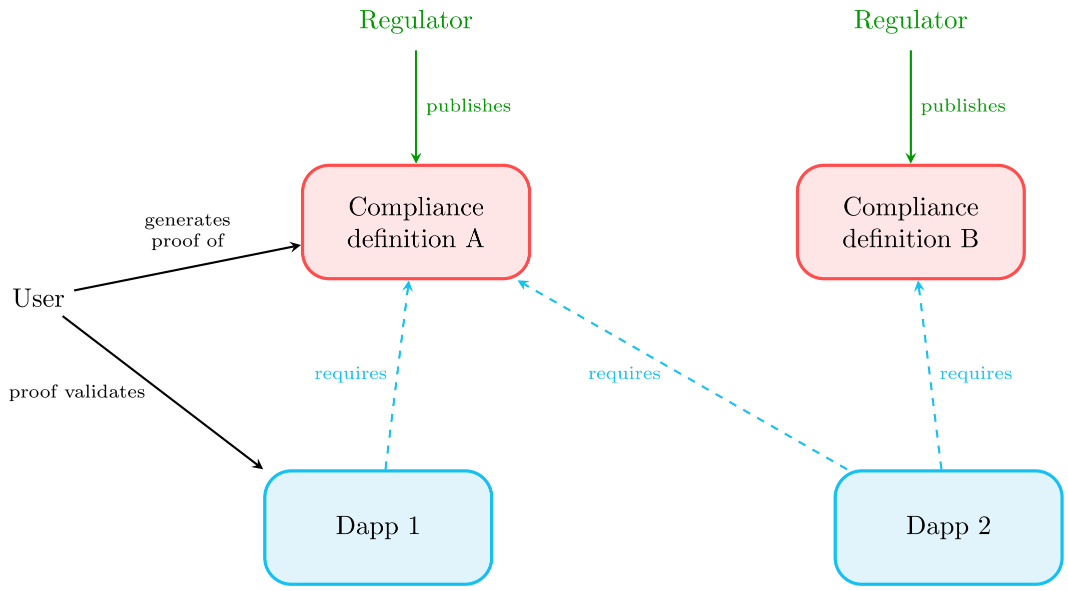

Figure 1: Composable compliance framework. Applications can opt into requiring multiple compliance definitions. Users can generate proofs of any compliance definition. In this example, a user is interacting with Dapp 1 so they must first generate a proof of compliance definition A.

6.4 Composability

The same definition of compliance can be used by any number of protocols, a user can generate proofs for any published definitions of compliance, and proofs for multiple compliance definitions can be required by a protocol. Figure 1 shows dapps opting into any number of compliance definitions. Figure 2 shows a user proving compliance for a dapp that requires multiple compliance definitions.

Compliance definitions can be fine-grained instead of wide-umbrella definitions. For example, instead of having a single definition that encompasses all US rules and regulations, there can be a definition for US consumer compliance and a definition for US corporate compliance.

6.5 Constant-cost compliance:

The cost to verify compliance proof on-chain does not grow with the size of the compliance definition. This is inherited from using zkSNARKs as compliance proofs. The circuit can be very computationally intensive and have pages of criteria, but it will still cost the same amount as a circuit with a single operation.

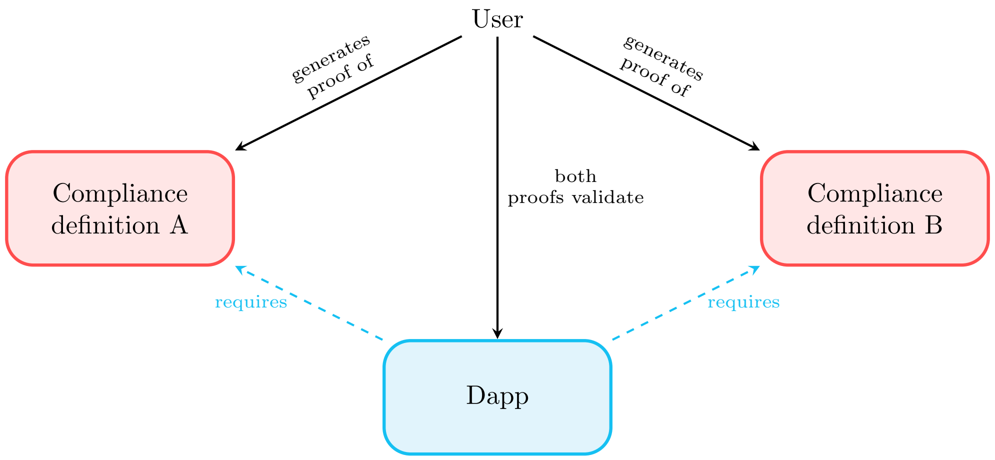

Figure 2: Example compliance scenario. A Dapp requires both A and B compliance. The user generates proofs for both compliance definitions and submits them together. The Dapp verifies both proofs on-chain before allowing the user to interact.

6.6 Constraints

Although regulations vary globally and over time, they often rely on similar logical primitives: checking membership in lists, comparing values against thresholds, or verifying historical patterns. We built our framework to leverage this similarity by decomposing compliance definitions into reusable logic that we refer to as *constraints*. This compositional structure allows for complex requirements to be expressed as combinations of constraints.

A *constraint* is a boolean predicate that evaluates blockchain state using first-order logic. A constraint C is expressed in the general form:

$$C = (Q\, x \in D(s) : P(x, p),\ \text{at chain state } i)$$

where:

- **Quantifier** ($Q \in \{\forall, \exists, \nVdash\}$): the logical quantifier. $\nVdash$ denotes the trivial quantifier for singleton domains (used in atomic constraints)
- **Domain** ($D : \mathcal{S} \to 2^{\mathcal{S}}$): a function that maps a subject to a collection of blockchain entities over which the quantifier ranges. Common domains include:
  - Singleton: $D(s) = \{s\}$ — evaluates a single subject (atomic case)
  - Transaction history: $D(\text{addr}) = \text{history}(\text{addr})$
  - Set of recipients: $D(\text{addr}) = \text{recipients}(\text{addr})$
  - Block ranges: $D(B) = \{[B_i, B_j] \subseteq [B_0, B_{\text{current}}] : \text{condition}\}$
  - Filtered collections: $D(s) = \{x \in D'(s) : \text{filter}(x)\}$
- **Subject** ($s \in \mathcal{S}$): the root blockchain entity from which the domain is derived. Example subjects include:
  - Sender of a transaction: $\text{sender}(\text{txn})$
  - Recipient of a transaction: $\text{recipient}(\text{txn})$
  - Account balance: $\text{balance}(\text{addr})$

  - Result of a contract call: result(contract fn)
  - Transaction payload: payload(txn)

- **Predicate** ($P : \mathcal{S} \times \mathcal{P} \to \{\text{true}, \text{false}\}$): a boolean function applied to each element of the domain. Predicates include:
  - Comparison operations: $P(x, p) = (x \,\phi\, p)$ where $\phi \in \{=, \neq, <, \leq, >, \geq, \in, \notin, \ldots\}$
  - Nested constraints: $P(x, p) = C'(x, p', i')$
  - Aggregate comparisons: $P(X, p) = (\text{agg}(X) \,\phi\, p)$ where $\text{agg} \in \{\sum, \text{count}, \max, \min, \ldots\}$
- **Parameter** ($p \in \mathcal{P}$): the value or set against which elements are evaluated, containing:
  - Public parameters p_{pub} provided by the compliance author (e.g., sanction lists, required values)
  - Private inputs p_{priv} provided by the user (e.g., mixer secret, identifying information)
- **Chain State** ($i \in \mathcal{I}$): the point in time or time range over which evaluation occurs. Examples include:
  - Current block: B_{current}
  - Historical block: B_h for $h <$ current
  - Block range: $[B_i, B_j]$
  - Unix timestamp

The constraint function has the signature:

$$C : \mathcal{Q} \times (\mathcal{S} \to 2^{\mathcal{S}}) \times (\mathcal{S} \times \mathcal{P} \to \{\text{true}, \text{false}\}) \times \mathcal{I} \to \{\text{true}, \text{false}\}$$

where $\mathcal{Q} = \{\forall, \exists, \nVdash\}$, and the constraint evaluates as:

$$C(Q, D, P, i) = \begin{cases} P(s, p) & \text{if } Q = \nVdash \text{ and } D(s) = \{s\} \text{ (atomic)} \\ \forall x \in D(s) : P(x, p) & \text{if } Q = \forall \text{ (universal)} \\ \exists x \in D(s) : P(x, p) & \text{if } Q = \exists \text{ (existential)} \end{cases}$$

Constraints can be composed using boolean operators to form complex compliance requirements:

$$R = C_1 \wedge C_2 \vee C_3 \wedge \ldots$$

6.6.1 Atomic Constraints as Special Cases

Atomic constraints are constraints where the domain is a singleton set $D(s) = \{s\}$ and the quantifier is $\nVdash$. In this case, the constraint reduces to directly evaluating the predicate:

$$C_{\text{atomic}} = (\nVdash s \in \{s\} : s \,\phi\, p, i) \equiv (s \,\phi\, p, i)$$

For clarity and brevity, atomic constraints are typically written in their simplified form:

$$C_{\text{atomic}} = (s \,\phi\, p, i)$$

6.6.2 Example constraint

Sanction list checks are a popular compliance measure in the blockchain space [52, 38]. The constraint "Sender of this transaction is not on sanction list A" can be decomposed as follows:

- **Subject**: sender(txn) — the address that initiated the transaction
- **Operation**: $\notin$ — does not exist in
- **Parameter**: A — the sanction list (a public parameter provided by the regulator)
- **Chain state**: B_{current} — evaluated at the current block

The constraint evaluates to **true** if and only if the subject (sender of the transaction) does not exist in the parameter (sanction list). Formally expressed as:

$$C_{\text{sanction}} = (\text{sender(txn)} \notin A, B_{\text{current}})$$

6.6.3 Example: shared constraints

When multiple compliance definitions require the same constraint, that constraint can be proved once and applied to multiple definitions. For example, an application wishes to be compliant with two regulatory bodies. These regulatory bodies have published compliance definitions R_1 and R_2, respectively. The application requires users to supply a valid proof of R_1 and R_2. R_1 and R_2 are composed of simple constraints:

$$R_1 = C_1 \wedge C_2$$
$$R_2 = C_1$$

A user of the application only needs to compute a proof for R_1 because it's constraints are a super-set of the constraints used in R_2: proving R_1 also proves R_2.

Another Example An application requires R_1 and R_2:

$$R_1 = C_1 \wedge C_2$$

$$R_2 = C_2 \wedge C_3$$

Constructing a proof that validates for both regulations only requires generating a proof for three constraints, as C_2 is included in both definitions. C_2 is computed once and applied to both R_1 and R_2.

6.6.4 Compliance definition

A compliance definition is a combination of constraints with boolean operators and a time parameter during which the definition is valid. The time parameter is necessary to account for regulations changing over time and to allow for regulations to be published before they go into effect. Given constraints $C_1, \ldots, C_k$, a compliance definition R is defined as:

$$R = (C_1 \wedge C_2 \wedge \cdots \wedge C_k, [t_{\text{start}}, t_{\text{end}}])$$

where t_{start} and t_{end} denote the block heights or timestamps defining the validity period of the compliance definition. The definition R is considered active if and only if the evaluation occurs within this temporal window:

$$\text{active}(R, t) = \begin{cases} \text{true} & \text{if } t_{\text{start}} \leq t \leq t_{\text{end}} \\ \text{false} & \text{otherwise} \end{cases}$$

Alternatively, open-ended validity can be expressed with $t_{\text{end}} = \infty$ for rules with no expiration. For brevity, the time parameter is omitted for compliance definitions and can be assumed to be valid for the current time.

6.6.5 Off-chain data in constraints

Constraints can require off-chain data that is provided by the user during proof generation time. This user input, p_{priv}, is completely private and not revealed to any other entity. This allows for constructing constraints that rely on off-chain data.

6.6.6 Verifier contract

[[[Joss: This is section that will become clear when I start implementation]]] <==

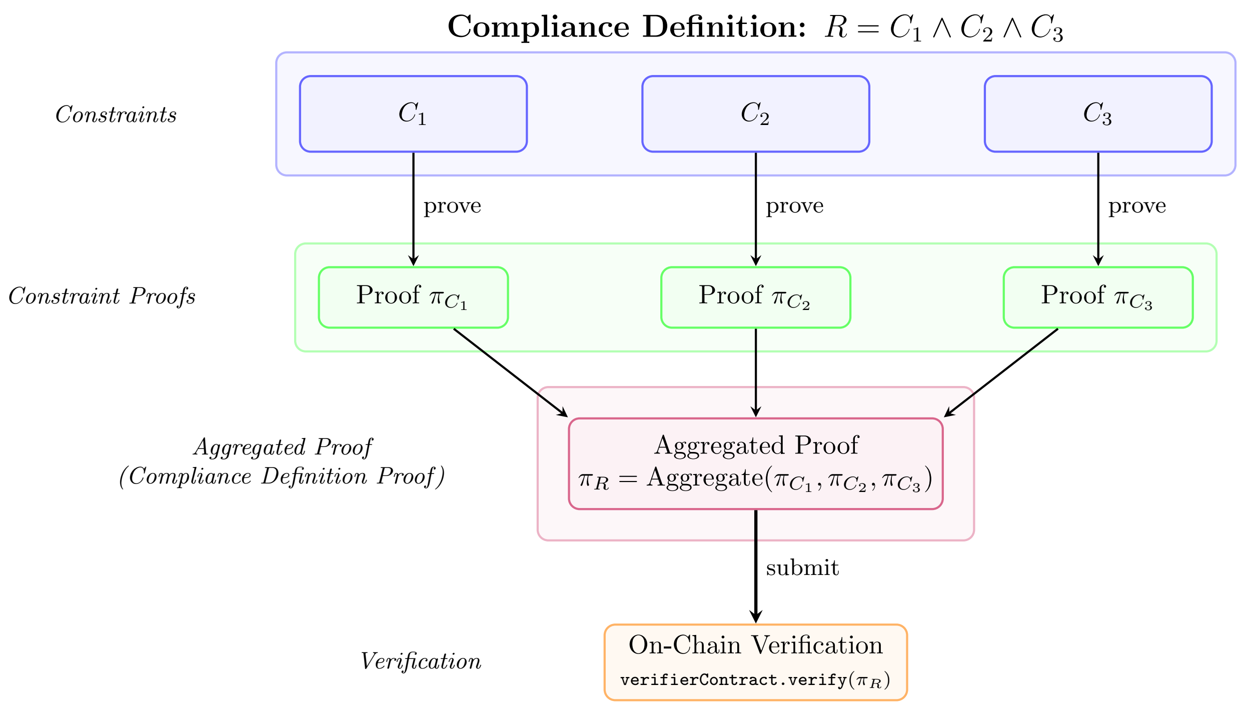

Figure 3: Proof aggregation workflow for compliance definition R. Individual constraints are proved separately, generating proofs π_{C_1}, π_{C_2}, and π_{C_3}. These proofs are then aggregated into a single proof π_R that is submitted on-chain for verification.

6.7 Generating Compliance Proofs

Proofs of compliance definition are aggregate proofs of multiple constraint proofs. When proving a compliance definition, each constraint is proved individually, then all constraint proofs are combined in an aggregation proof which is then submitted on-chain. See Figure 3 for the complete workflow.

Proofs are generated locally by the user and then submitted on-chain as transaction arguments. The proof is public on-chain[5], but it cannot be used to recover the user's private inputs.

The user can locally save all constraints that they have proven, discussed further in section 6.8. This allows users to re-use previous proved constraints. It is often the case that some parts of a compliance definition may already be proved or that the constraints of separate regulatory domains have some subexpressions in common. In these cases, inference over the constraints can lead to significant reduction in the complexity of the constraints to be proved.

[5]Some blockchains allow for private transactions. Proofs submitted as part of a private transaction will not be public.

Example Alice previously proves compliance definition R_1, which requires proving constraints C_1 and C_2. Alice has locally saved her proofs of C_1 and C_2.

$$R_1 = C_1 \wedge C_2$$

Now, Alice wishes to prove R_2.

$$R_2 = C_2 \wedge C_3$$

She can inspect R_2 and sees that it uses C_2, a constraint that she has previously proved and saved. She can skip the computation of C_2 because she already has a valid proof! All she needs to compute is C_3, and R_2, which is the aggregate proof over C_2 and C_3.

6.8 Proof Manager

We envision a client-side program that aids in proof generation. The proof manager takes as input a compliance definition, chain identifier, private user input, and the user address. It outputs a proof of that compliance definition.

The proof manager is responsible for

- Doing inference over required constraints to identify the minimum amount of proving necessary, relying on previously proved constraints.
- Constraint to circuit compilation.
- Fetching on-chain data of the user's transaction history necessary for proof generation.
- Fetching public proof inputs from the published verifier contract.
- Generating constraint proofs.
- Generating aggregate proof over the constrains to prove compliance.
- Saving completed proofs.

6.9 Updating Compliance Definition

Laws and regulations are constantly updated and by governments; therefore, definitions of compliance will change over time. The framework allows updates to compliance definitions.

A compliance definition can be updated in two ways, and both cases involve the regulator changing the verifier contract state.

- Public parameter p_{pub} update (ex: add or remove an address from the sanction list) — modifies the public inputs used in proof verification
- Constraint composition update (ex: add or remove a constraint from the definition) — modifies the circuit
  - When using PLONK, circuit modifications are feasible, as the trusted setup can be reused.

The options for handling updates are to deploy a new contract or update the existing contract.

- Regulator publishes a new verifier contract for the updated compliance and **all** applications must update their smart contracts to reference this new contract address in order to remain compliant.
- Regulator updates their current verifier contract and no change is needed by the application.

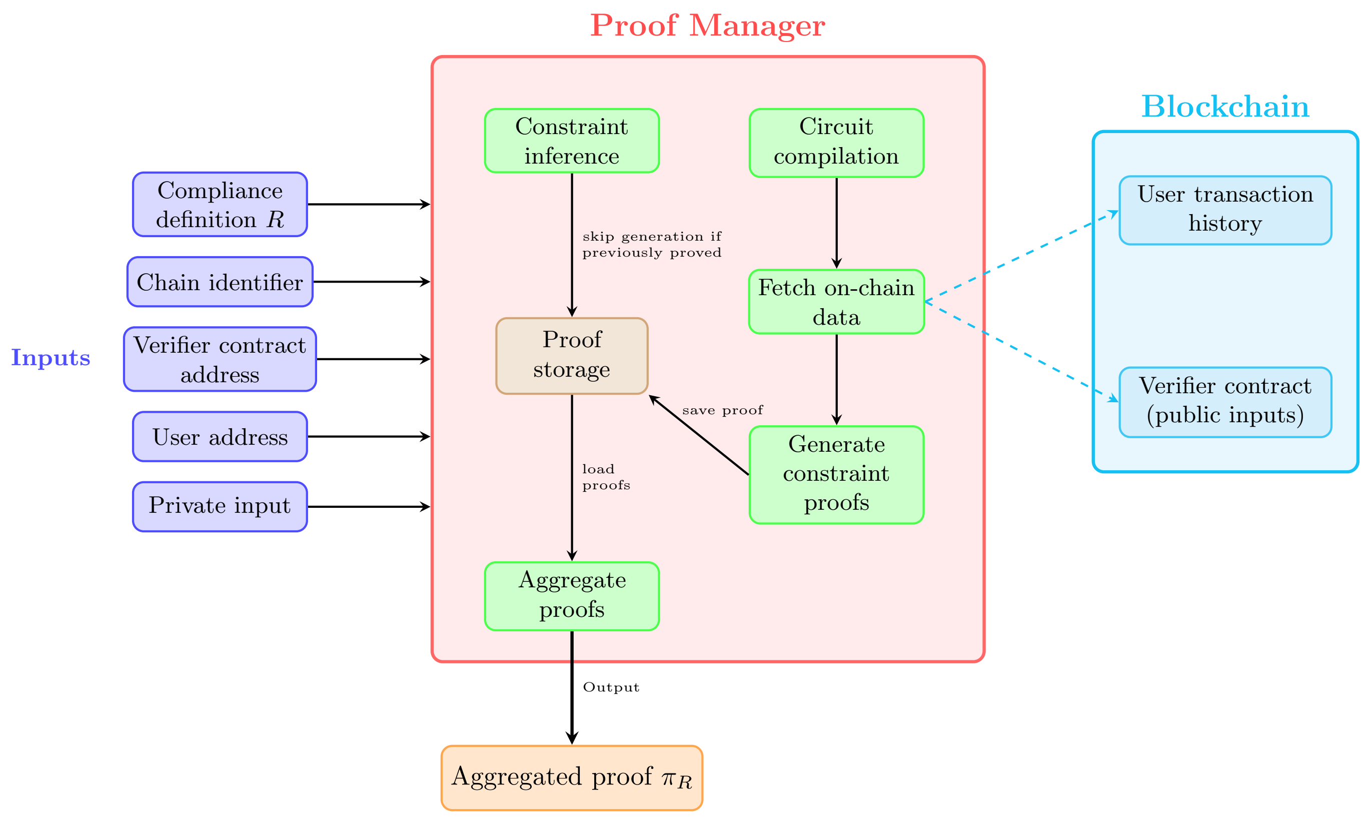

Figure 4: Proof Manager architecture. The system processes inputs through constraint inference (which checks proof storage for previously generated proofs), circuit compilation, fetching on-chain data from both the blockchain transaction history and verifier contract, and proof generation. Generated proofs are saved to proof storage and then loaded by the aggregation step to produce the final compliance proof π_R.

Chainalysis, a leading blockchain analytics company used by government bodies, adds addresses to their sanctions lists every 30 days [15]. Using 30 days as a benchmark as to how often compliance definitions might have to change, it is infeasible to require all applications to make smart contract changes this frequently. Especially because the failure to enforce compliance is potentially a criminal charge. We don't want to introduce the potential for a mistake as small as an application updating a day late to result in legal action. A priority of this framework is to provide applications with a strong guarantee of compliance and minimal overhead to the application.

For these reasons, we choose the path of regulators updating their existing compliance contract instead of publishing a new one on each update. This will keep the compliance verifier contract at the same address and does not require any action from the application. With this technique, applications have the guarantee that all their users meet the compliance definition with no windows of non-compliance.

It should be noted that updating compliance definitions can introduce a brief period of regulatory arbitrage. This is the period after which non-compliant actors are identified but before sanction lists have been updated with their addresses, as discussed in [4]. Currently, blockchain analytics companies are responsible for regularly updating sanctions lists. These blockchain analytics companies compete on speed and accuracy of their sanction list updates. This framework allows compliance definitions' sanction lists to be updated by these analytics companies, which aligns with their current responsibilities and strengthens the incentive for fast updates.

6.9.1 Updating techniques

[[[Joss: TODO either proxy contract or ownable functions. Unsure which is better for this case, and this is a question that will be answered during implementation. Come back to this section when we've decided. Maybe we don't have to enforce one choice over the other, but we should suggest a path because this framework should be simple for regulators to use.]]] <== [[[HFK: this addresses a remark I made about this earlier. I think we have to at least mention the update methodology briefly earlier. I see the current discussion there as worrisome to any reader who does not have confidence that we are actually doing it right.]]] <==

6.9.2 Post Update Re-proving

Users who generated but did not use a compliance proof before the regulator updated the verifier contract may have to re-generate parts of their proof. They will have to regenerate the aggregate proof over the constraints, but might not have to re-prove all constraints in that compliance definition. If a definition consists of constraints C_1 and C_2, and the regulator updates the public parameters of C_1, users will have to reprove C_1 but are able to re-use their proof of C_2.

Re-proving only the constraints that changed in an update is less costly than re-proving the entire compliance definition. Regulators have some incentive to make small updates since ease of proof generation leads to ease of compliance.

[[[HFK: This is a larger topic. Say regulation R_1 is updated to R_2. We might not only publish the full R_2 for new proofs but also $R_{1,2}$ that has the property that $R_1 \wedge R_{1,2} \implies R_2$]]] <==

6.9.3 Cross-chain Update Considerations

If a compliance definition is updated across multiple independent blockchains, there exists a small window of regulatory arbitrage after the update is processed on one chain but before it's processed on another. There is no native solution for atomic cross-chain transactions that accounts for different data layers and virtual machines.

The solution to prevent windows of regulatory arbitrage is to publish the updated compliance before it goes into effect. Recall that the compliance definition has time bounds:

$$R = (C_1 \wedge C_2 \wedge \cdots \wedge C_k, [t_{\text{start}}, t_{\text{end}}])$$

Compliance definitions can be published before t_{start} time. If it is published far enough ahead of t_{start}, it is likely that the update will be propagated across all chains by t_{start} when the update goes into effect.

6.10 Regulator Trust Assumption

Because compliance is updatable, there exists a risk vector of the regulator of a compliance definition turning malicious and publishing an unjust or incorrect update. For example, a regulator adds an address to the sanction list to censor a particular law-abiding citizen.

This framework does not prevent against malicious regulators, but it provides a solution. One core assumption is that any application trusts the author of the selected compliance definitions. If an application is not satisfied with any of the available compliance definitions or untrusting of the regulators, they can easily publish and maintain their own definition. They can create a compliance definition from scratch or fork one of the publicly available definitions. However, "in-house" compliance results in more work for the application's team. A key property of this framework is that publishing compliance definitions is **permissionless**.

Additionally, because compliance definitions are public, any malicious update is also public. Transparent regulation is much safer for users and applications because the regulator must weigh the benefit of the malicious action against the consequences of the public fallout.

7 Prototype

This framework is applicable to any blockchain with a smart contract language and constraints can be written in any syntax as long as they are representable in a ZK circuit. For our initial prototype of this framework, constraints are expressed in the Noir ZK DSL [3] and the application integrating this framework will use the Ethereum Virtual Machine (EVM) expressed through the Solidity smart contract DSL. Noir was chosen because it provides high-level language semantics for writing ZK circuits. The EVM and Solidity were chosen because they have a large and polished ecosystem of smart contract tools.

7.1 Prototype Constraints

These prototype constraints demonstrate the flexibility and expressiveness of our constraint formalism. The constraints below were chosen to highlight some common compliance measures and do not cover all possible constraints. We encourage regulators to be creative when building constraints.

Non-Membership Constraint (`NON-MEM`): Proves that an address is not in a given set. For example, proving an address is not on a list of sanctioned addresses.

$$C_{\text{non-mem}} = (\text{sender}(\text{txn}) \notin p_{\text{sanctioned}}, B_{\text{current}})$$

where $p_{\text{sanctioned}}$ is the public parameter containing the sanction list.

Membership Constraint (`MEM`): Proves that an address is in a given set. For example, proving an address is on an allow-list.

$$C_{\text{mem}} = (\text{sender}(\text{txn}) \in p_{\text{allowlist}}, B_{\text{current}})$$

where $p_{\text{allowlist}}$ is the public parameter containing the allowed addresses.

Protocol Interaction Constraint (`INTERACT`): Proves that at some point in their transaction history, an address interacted with a specific protocol.

$$C_{\text{interact}} = (\exists \text{txn} \in \text{history}(\text{sender}(\text{txn})) : \text{recipient}(\text{txn}) = p_{\text{protocol}}, [B_0, B_{\text{current}}])$$

where p_{protocol} is the address of the required protocol.

Protocol Avoidance Constraint (`AVOID`): Proves that throughout their transaction history, an address never interacted with a specific protocol.

$$C_{\text{avoid}} = (\forall \text{txn} \in \text{history}(\text{sender}(\text{txn})) : \text{recipient}(\text{txn}) \neq p_{\text{protocol}}, [B_0, B_{\text{current}}])$$

where p_{protocol} is the address of the prohibited protocol.

Age Constraint (`AGE`): Requires that an address be at least a certain age. For example, require an addresses first transaction was at least 6 months ago as an effort to reduce sybil addresses.

$$C_{\text{age}} = (B_{\text{current}} - \text{first-txn}(\text{sender}(\text{txn})).\mathit{block} \geq p_{\text{min-age}}, B_{\text{current}})$$

where $B_{\text{first}}(\text{addr})$ returns the block height of the address's first transaction and $p_{\text{min-age}}$ is the minimum required age in blocks.

Structuring Constraint (`STRUCTURE`): Specific example of US Bank Secrecy Act non-structuring requirement. Proves a user has not sent multiple transactions totaling over $10,000 within a window of time to the same recipient.

$$C_{\text{structure}} = (\forall r \in \text{recipients}, \forall [B_i, B_j] \subseteq [B_0, B_{\text{current}}] \text{ where } (B_j - B_i) \leq p_{\text{window}} :$$

$$\sum_{\text{txn} \to r \text{ in } [B_i, B_j]} \text{amount}(\text{txn}) < p_{\text{threshold}}, [B_0, B_{\text{current}}])$$

where p_{window} is the time window (e.g., 24 hours in blocks) and $p_{\text{threshold}} = 10000$ USD equivalent.

7.2 Integrating applications

In this prototype, applications integrate this framework by requiring proofs of compliance at specified entrypoint functions. Applications choose compliance definitions based on the applications regulatory goals.

We envision a constraint compiler that compiles constraints from a formal syntax into custom circuits, but for prototype purposes, constraints are "hand compiled" into Noir circuits. Constraints are still listed to show the logical definition.

Each compliance definition R is compiled as a ZK circuit using Noir and then published and maintained in a separate verifier Solidity contract, allowing the integrating application to reference compliance logic without implementing it directly.

7.2.1 Example: Compliant Stablecoin

The author of this stablecoin contract is, for example, a US based company and wishes to only allow users who are not on a list of US sanctioned addresses $p_{\text{sanctioned}}$ to interact with this stablecoin.

The compliance definition requires both sender and recipient to satisfy the non-membership constraint:

$$R_{\text{stablecoin}} = C_{\text{non-mem}}(\text{sender}) \wedge C_{\text{non-mem}}(\text{recipient})$$

where both constraints evaluate against the same sanction list $p_{\text{sanctioned}}$.

Contract: $R_{\text{stablecoin}}$Verifier

```
// State: $sanctionedAddresses$ (set of sanctioned addresses)

VerifyCompliance($sender, recipient, proof$)
    // Construct public inputs for verification
    $publicInputs = $ FormatPublicInputs($sender, recipient, sanctionedAddresses$)

    // Verify both parties satisfy NON-MEM constraint
    **assert** Verify($proof, publicInputs$)

    **return** TRUE
```

Contract: CompliantStablecoin

```
// State:
        $balance$ (mapping from address to uint)
        $verifierContract$ (address of $R_{\text{stablecoin}}$Verifier)

Transfer($recipient, amount, proof$)
    $sender = tx.origin$

    // Verify compliance through external verifier contract
    **assert** $verifierContract$.VerifyCompliance($sender, recipient, proof$)

    // Execute transfer
    **assert** $balance[sender] \geq amount$
    $balance[sender] = balance[sender] - amount$
    $balance[recipient] = balance[recipient] + amount$

    **return** TRUE
```

This architecture demonstrates the composability of the framework: the compliance definition $R_{\text{stablecoin}}$ is maintained independently from the stablecoin logic, allowing the sanction list to be updated without modifying the stablecoin contract itself.

8 Benchmarks

- Benchmark graphs with reasoning for each test and implementation code link
- Discussion on which pattern will be most heavily used and what acceptable performance is
- Prove non-membership of the set of 0 addresses
- Prove membership of the set of all addresses
- Prove address genesis property (from coinbase, US citizen, etc)
- Prove they didn't structure historic transactions over 10k
- Prove over large transaction histories
- more...

9 Future Work

9.1 Constraint Domain-Specific Language

To transform written law into a zkSNARK there needs to be a human readable intermediary syntax. The constraint system introduced in 6.6 can be extended into a novel DSL used by regulatory bodies to define compliance. A goal of this DSL is to be easier to read than the formal expression of constraints in raw logic. This would make constraint construction **much** easier for regulators, leading to more framework integration. A larger ecosystem of compliance definitions and a readable compliance syntax would in turn give applications and users more confidence.

This DSL would compile directly into constraints. The addition of this proposed DSL would require no changes to the overall framework and is backward compatible: the framework already is built around the constraint unit. Even the verifier contracts could stay the same. The only modification that would be necessary is a patch to the proof manager to add this compilation step.

Example `NON-MEM` constraint translation The non-membership constraint defined as:

$$C_{\text{non-mem}} = (\text{sender}(\text{txn}) \notin p_{\text{sanctioned}}, B_{\text{current}})$$

Could instead be expressed in a constraint DSL in a more readable fashion:

$$C_{\text{non-mem}} = \texttt{User is not in } p_{\text{sanctioned}}$$

Where "is" expresses the temporal context of the current block, and $p_{\text{sanctioned}}$ is the sanction list.

Example `INTERACT` constraint translation The protocol interaction constraint defined as:

$$C_{\text{interact}} = (\exists \text{txn} \in \text{history}(\text{sender}(\text{txn})) : \text{recipient}(\text{txn}) = p_{\text{protocol}}, [B_0, B_{\text{current}}])$$

Could be translated into:

$$C_{\text{interact}} = \texttt{User has sent transaction to } p_{\text{protocol}}$$

Where "has" means at any point in the user's transaction history and p_{protocol} is the address of the required protocol.

Example published compliance definition Compare the compliance definition $R = C_{\text{non-mem}} \wedge C_{\text{interact}}$ with the current constraint syntax to the same definition using a constraint DSL. Which would you be more likely to integrate with?

Current:

1 $\exists \text{txn} \in \text{history}(\text{sender}(\text{txn})) : \text{recipient}(\text{txn}) = p_{\text{protocol}}, [B_0, B_{\text{current}}]$
2 $\text{sender}(\text{txn}) \notin p_{\text{sanctioned}}, B_{\text{current}}$

Using constraint DSL:

1 `User is not in` $p_{\text{sanctioned}}$
2 `User has sent transaction to` p_{protocol}

9.2 Integration with commitment tree privacy mixers

Commitment tree mixers like Tornado Cash [43] and Privacy Pools [14] unlink the sender and receiver addresses of a transfer, creating privacy for the user. A user enters the mixer by sending funds to the mixer along with a secret. An address can withdraw funds from the mixer if they provide a ZK proof that know a secret corresponding to a deposit. There is no way to reveal association between any two deposit and withdraw transactions in a mixer without the user revealing their secret. The common use case is a privacy-desiring user withdrawing to one of their addresses with zero transaction history.

Inspired by "Derecho: Privacy Pools with Proof-Carrying Disclosures" [6], the compliance properties required to enter the mixer can be transferred to the withdrawing address by proving the owner of the withdrawing address knows the secret of a leaf in the commitment tree, therefore proving that they have access to an account that met the compliance definition to deposit into the mixer. Then an additional proof needs to be made to assure that the withdrawing account hasn't previously violated compliance constraints. Only the constraints required for depositing to the mixer can be carried to the withdrawing address.

Example: Alice signs up to Coinbase and provides required KYC documents. Alice then receives funds at her on-chain address from a known Coinbase address. Alice can trivially prove her address received funds from Coinbase. Alice does not want her entire transaction history to be public so she deposits some funds in a mixer that has integrated this framework and requires a compliance definition with a single constraint: C_{origin}

$$C_{\text{origin}} = (\text{sender}(\text{first-txn}(\text{sender}(\text{txn}))) \in A \text{ at } [B_0, B_{\text{current}}])$$

C_{origin} is read as "the sender of this transaction's first transaction was from a set of addresses A", where A is a set of public Coinbase addresses.

Alice proves this to deposit funds to the mixer. She later withdraws her funds from the mixer to a freshly generated address with no transaction history. Even though this new address never received funds from Coinbase, Alice is able to supply her commitment tree secret into a circuit to prove that she is transferring from an address that successfully proved C_{origin}, and therefore should be able to prove C_{origin} when interacting with protocols with her new address. Work needs to be done testing and implementing this.

9.3 Alternative Applications of Constraint Proving

This paper focused on the use case of proving compliance over a transaction history. However, the underlying mechanism of proving arbitrary properties of a transaction history is powerful and can be used in other scenarios. For example, airdrops that only allow addresses with specific properties to claim. It could potentially even move some smart contract execution off-chain by requiring a proof that an invariant holds for a particular address instead of calculating it on-chain.

References

[1] 0xbow. Privacy pools documentation, 2025. `https://docs.privacypools.com/`.

[2] Aleo Network Foundation. Aleo: The Blockchain Built for Payments. `https://aleo.org/`, 2025. Accessed: 2025-10-14.

[3] Aztec Team. Introducing Noir: the universal language of zero-knowledge, October 2022. `https://medium.com/aztec-protocol/introducing-noir-the-universal-language-of-zero-knowledge-ff43f38d86d9`.

[4] S. Baranski, M. Dotan, A. Lotem, and M. Vald. Haze and daze: Compliant privacy mixers. Cryptology ePrint Archive, Paper 2023/1152, 2023.

[5] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. *ACM Trans. Comput. Syst.*, 33(3), Aug. 2015.

[6] J. Beal and B. Fisch. Derecho: Privacy pools with proof-carrying disclosures. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 3197–3211, New York, NY, USA, 2024. Association for Computing Machinery.

[7] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In S. Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88.*, Lecture Notes in Computer Science, vol 403. Springer, 1988.

[8] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[9] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[10] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 947–964, 2020.

[11] J. Burleson, M. Korver, and D. Boneh. Privacy-protecting regulatory solutions using zero-knowledge proofs. Technical report, a16z crypto, 2022.

[12] V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. Web document, 2013. `https://ethereum.org/en/whitepaper/`.

[13] V. Buterin. An incomplete guide to stealth addresses, January 2023.

[14] V. Buterin, J. Illum, M. Nadler, F. Schär, and A. Soleimani. Blockchain privacy and regulatory compliance: Towards a practical equilibrium. *Blockchain: Research and Applications*, 5(1), March 2024.

[15] Chainalysis. Chainalysis sanctions oracle contract. Etherscan, 2022. Ethereum contract address: `0x40C57923924B5c5c5455c48D93317139ADDaC8fb`.

[16] A. Chaudhary. zkfi: Privacy-preserving and regulation compliant transactions using zero knowledge proofs, 2025.

[17] A. Chaudhary and H. Ivey-Law. Sede: Balancing blockchain privacy and regulatory compliance by selective de-anonymization, 2025.

[18] K. Collier. Former twitter employee sentenced to more than three years in prison for spying for saudi arabia. *NBC News*, 12 2022.

[19] R. Daws. FCC fines major telcos for selling users' location data. *Telecoms Tech News*, 4 2024.

[20] N. De. US officials arrest alleged operator of $336m bitcoin mixing service. *CoinDesk*, 2021.

[21] DeFi Llama. Privacy pools protocol analytics, 2025. Accessed: 2025-11-6.

[22] DeFi Llama. Railgun (rail) protocol analytics, 2025. Accessed: 2025-10-27.

[23] DeFi Llama. Tornado cash (torn) protocol analytics, 2025. Accessed: 2025-10-26.

[24] R. Dingledine and N. Mathewson. Anonymity loves company: Usability and the network effect. In *Workshop on the Economics of Information Security*, 2006.

[25] J. Duff and J. Valkema. Ultra Anon: An experimental privacy token with maximum plausible deniability and anonymity set which blurs the lines of public and private state. Eth Denver Hackathon, 2025. `https://devfolio.co/projects/ultra-anon-9f12`.

[26] A. Gabizon, Z. J. Williamson, and O.-M. Ciobotaru. PlonK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953, 2019.

[27] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th ACM Symp. on the Theory of Computing*, 1985.

[28] J. Groth. On the size of pairing-based non-interactive arguments, 2016. `https://doi.org/10.1007/978-3-662-49896-5_11`.

[29] W. Jia, T. Xie, and B. Wang. A privacy-preserving scheme with multi-level regulation compliance for blockchain. *Scientific Reports*, 14(438), 2024. `https://www.nature.com/articles/s41598-023-50209-x`.

[30] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. In *27th USENIX Security Symposium (USENIX Security '18)*, 2018.

https://arxiv.org/abs/1805.03180.

[31] keyvank. Eip-7503: Zero-knowledge wormholes - private proof of burn (ppob). Forum post, 2023. https://ethereum-magicians.org/t/eip-7503-zero-knowledge-wormholes-private-proof-of-burn-ppob/15456/1.

[32] D. Kuhn. Samourai wallet charges raise existential questions for privacy tech. *CoinDesk*, April 2024.

[33] R. Lavin, X. Liu, H. Mohanty, L. Norman, G. Zaarour, and B. Krishnamachari. A survey on the applications of zero-knowledge proofs, 2024. https://arxiv.org/abs/2408.00243.

[34] G. Y. Liao, R. Mayer, A. Soghoian, S. Jain, and E. Tierney. Arc: An open layer-1 blockchain purpose-built for stablecoin finance. Technical report, Circle, August 2025. https://6778953.fs1.hubspotusercontent-na1.net/hubfs/6778953/Arc%20Litepaper%20-%202025.pdf.

[35] G. Maxwell. Coinjoin: Bitcoin privacy for the real world. Bitcoin Forum, August 2013.

[36] C. Moore and S. Ghandi. L2 Ethereum ZK rollup for private and compliant transactions. Whitepaper, 2024.

[37] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018(3):143–163, 2018.

[38] M. Nadler and F. Schär. Tornado cash and blockchain privacy: A primer for economists and policymakers. *Federal Reserve Bank of St. Louis REVIEW*, 2023. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4352337.

[39] S. Nakomoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Bitcoin.org, 2008.

[40] L. H. Newman. Twitter insiders allegedly spied for saudi arabia. *WIRED*, 11 2019.

[41] Paradigm Policy Team. Tradfi tomorrow: Defi and the rise of extensible finance. Policy report, Paradigm, March 2025.

[42] Penumbra Labs. The Penumbra Protocol, 2023.

[43] A. Pertsev, R. Semenov, and R. Storm. Tornado cash privacy solution, 2019. https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf.

[44] M. Petkus. Why and how zk-SNARK works: Definitive explanation. arXiv 1906.07221v1, 2019.

[45] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. https://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, 2010.

[46] Polygon Miden. Miden: The edge blockchain. https://miden.com/, 2024. Accessed: 2025-10-14.

[47] K. Post. Exclusive: Massive data leak potentially exposes ukrainian ids to russian intelligence, hackers. *The Kyiv Independent*, 4 2025.

[48] Privacy Patterns. Anonymity set, 2024. https://privacypatterns.org/patterns/Anonymity-set.html.

[49] Railgun. Railgun developer guide, 2024. https://docs.railgun.org/developer-guide/.

[50] D. Rathee, G.-V. Policharla, T. Xie, R. Cottone, and D. X. Song. Zebra: Anonymous credentials with practical on-chain verification and applications to kyc in defi. *IACR Cryptol. ePrint Arch.*, 2022:1286, 2022.

[51] J. Sakellariadis and M. Miller. China-linked hackers stole wiretap data from telcos, fbi and cisa say. *POLITICO*, 11 2024.

[52] Solomka I. and Liubinskyi B. Zero-knowledge proof framework for privacy-preserving financial compliance. *Mathematical Modeling and Computing*, 12(1):342–354, 2025. https://doi.org/10.23939/mmc2025.01.342.

[53] D. Stone. Webb protocol: A cross-chain private application and governance protocol. Cryptology ePrint Archive, Paper 2023/260, 2023.

[54] Stripe and Paradigm. Tempo: The blockchain designed for payments, 2025. https://tempo.xyz/.

[55] TRM Labs. 2025 crypto crime report: Key trends that shaped the illicit crypto market in 2024. Technical report, TRM Labs, San Francisco, CA, 2025.

[56] United States Congress. Bank secrecy act. 31 U.S.C. § 5318(h), 1970. Requires financial institutions to establish Anti-Money Laundering programs.

[57] U.S. Attorney's Office, Southern District of New York. Tornado cash founders charged with money laundering and sanctions violations. Press Release, August 2023.

[58] N. van Saberhagen. CryptoNote v 2.0, October 2013. Whitepaper.

[59] Y. Wang and J. Gao. A regulation scheme based on the ciphertext-policy hierarchical attribute-based encryption in bitcoin system. *IEEE Access*, 6:16267–16278, 2018.

[60] Wikipedia contributors. Shanghai police database leak. Wikipedia, The Free Encyclopedia, 2025.

[61] Z. J. Williamson. The AZTEC Protocol. Technical report, Aztec, 2018. Whitepaper.

[62] X. Xiong, M. Huth, and W. Knottenbelt. Regkyc: Supporting privacy and compliance enforcement for kyc in blockchains. *IACR Cryptol. ePrint Arch.*, 2025:579, 2025.

[63] zkBob Team. zkbob overview, 2023.