Blockchain: Research and Applications 5 (2024) 100176

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Research Article

Blockchain privacy and regulatory compliance: Towards a practical equilibrium

Vitalik Buterin [a], Jacob Illum [b], Matthias Nadler [c], Fabian Schär [c,*], Ameen Soleimani [d]

[a] Ethereum Foundation, Switzerland
[b] Chainalysis, United States of America
[c] University of Basel, Switzerland
[d] Privacy Pools, United States of America

ARTICLE INFO

Keywords:
Blockchain
Privacy
Regulation
Smart contracts
Zero-knowledge proofs

ABSTRACT

We study Privacy Pools, a novel smart contract-based privacy-enhancing protocol. The protocol introduces a mechanism for users to reveal certain properties of their transaction without having to reveal the transaction itself. The core concept involves allowing users to publish a zero-knowledge proof, demonstrating that their funds (do not) originate from known (un-)lawful sources, without publicly revealing their entire transaction history. This is achieved by proving membership in custom association sets, which are designed to demonstrate compliance with regulatory frameworks or social consensus. We illustrate how this mechanism can create a separating equilibrium between compliant and non-compliant withdrawals. Our work describes the technical underpinnings, incentives, and broader implications of this mechanism, highlighting how Privacy Pools-like protocols can create more private yet compliant blockchain transactions.

1. Introduction

Public blockchains are transparent by design. The basic idea is that anyone should have the option of validating transactions without having to rely on centralized third parties. This reduces dependencies and may provide a neutral foundation for various applications, including but not limited to finance and self-sovereign identities.

However, the existence of a public dataset that contains each transaction of every blockchain address is problematic from a privacy point of view. Whenever someone transfers an asset to another address and/or interacts with a smart contract, the transaction will be forever visible on the blockchain.

Consider the following example: Alice goes to a restaurant and uses her blockchain wallet to pay for dinner. The recipient now knows Alice's address and can analyze all past and future activities from that address. Similarly, Alice now knows the wallet address of the restaurant and could use this information to obtain the wallet addresses of other guests or look into the revenue of the restaurant. Third parties who know the restaurant's wallet address and have the information that Alice is dining there (e.g., from social media) can easily derive Alice's address and study her past and future transactions as well.

Although the restaurant example might be considered a hypothetical scenario, the fundamental concept applies to every transaction conducted on a public blockchain. Each action performed on a public blockchain is publicly recorded and accessible to everyone, enabling third parties to analyze users' financial transactions and behavioral patterns.

These issues have led to the emergence of privacy-enhancing protocols. They allow users to deposit funds into the protocol using one address and withdraw them from the protocol at a later point in time using another address. All deposits and withdrawals are still visible on the blockchain, but the link between a specific deposit and its withdrawal counterpart is no longer public.

One of the best-known privacy-enhancing protocols is Tornado Cash. It succeeded in solving the above-mentioned issues and allowing users to retain some privacy. However, besides legitimate users trying to protect their data, Tornado Cash has also been used by various bad actors. Deposit data suggest that hacker groups have transferred funds from illicit sources through the protocol [1]. Evidence that the privacy-enhancing protocol has also been used by a North Korean hacker group eventually led to the placement of the protocol's smart contract addresses on the list of Specially Designated Nationals and Blocked Per-

* Corresponding author.
E-mail address: f.schaer@unibas.ch (F. Schär).

https://doi.org/10.1016/j.bcra.2023.100176
Received 14 September 2023; Received in revised form 7 December 2023; Accepted 7 December 2023
Available online 22 December 2023

sons (commonly known as the SDN list) maintained by the Office of Foreign Assets Control (OFAC) in the United States.

Essentially, the critical issue with Tornado Cash was that legitimate users had limited options to dissociate themselves from the criminal activity that the protocol had attracted. Tornado Cash provides a compliance tool that allows a user to create a proof of which deposit a given withdrawal came from. While this mechanism does allow people to prove their innocence, it comes at the cost of having to trust a centralized intermediary and creates information asymmetries [2]. Consequently, the mechanism saw little use.

This paper discusses an extension to this approach that enables users to publicly prove an informative, but still broad, claim about which deposits their withdrawal could have come from. It allows for *membership proofs* ("I prove that my withdrawal comes from one of these deposits") or *exclusion proofs* ("I prove that my withdrawal does *not* come from one of these deposits"). The general concept has been proposed by Privacy Pools [3]. The present paper discusses this proposal and explains how to use its building blocks to reach a separating equilibrium between honest and dishonest protocol users.

Note that Privacy Pools provide additional options by extending the users' action set. They can still provide more detailed proofs to specific counterparties, if needed. However, there will be cases in which a membership or exclusion proof is sufficient. Moreover, the option to publish these proofs publicly does have many advantages over bilateral disclosure.

Our paper is structured as follows. After this short introduction, we provide a technical background on zero-knowledge proofs and Privacy Pools. In Section 3, we discuss how association sets are used and constructed. In Section 4, we elaborate on further technical details and special cases. In Section 5, we discuss our findings and turn to practical considerations. Finally, in Section 6, we present the conclusion. We encourage the reader to also explore other privacy-enhancing proposals that strive to achieve a similar balance through different implementations, e.g., Ref. [4].

2. Technical background

In this section, we provide a short technical overview and discuss the technical building blocks and general principles of Privacy Pools-like protocols.

2.1. Blockchain privacy before ZK-SNARKs

Historically, blockchain proponents have argued that blockchains can preserve privacy despite all transactions being transparent because they offer *pseudonymity*: You do not need to reveal any information about your offline identity to use a blockchain. Instead, users are identified by numerical "addresses".

Satoshi Nakamoto's Bitcoin white paper makes this exact claim, arguing that "privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else but without information linking the transaction to anyone" [5]. Unfortunately, this level of privacy has proven to be far insufficient in the face of modern clustering and analysis tools [6,7]. Non-financial applications make maintaining privacy even more difficult, as they often require users to publish other kinds of information about themselves on-chain. For example, registering a name on decentralized domain name services such as ENS [8] involves making a transaction on the Ethereum blockchain and creating a public link between your transactions and your ENS name.

For this reason, there has been a movement toward improving privacy on public blockchains by introducing more powerful technology. The earliest non-trivial privacy solution that saw a significant degree of adoption was CoinJoin [9]. CoinJoin involved small groups of users coming together and mixing their coins with each other within a single transaction. Looking at the chain, one could only see the total set of inputs and outputs of a given round of the CoinJoin protocol and not which input corresponds to which output. The theory was that a user could participate in many rounds of the CoinJoin protocol with different groups of people, thereby hiding the source of their assets among many possible inputs. Monero took this a step further by using a linkable ring signature scheme [10] to allow users to mix their coins with a few other users' coins without requiring any off-chain interaction. With improvements in technology [11], the number of participants in each mix grew, increasing the *anonymity set* of each transaction—the number of historical transactions that could have been the origin of that transaction. However, such repeated small-group mixing techniques inevitably pose the risks of data leakage [12].

The next logical advancement in the quest for increased cryptographic privacy involved the introduction of general-purpose zero-knowledge proofs. This innovation is evident in Zerocash [13], and it has since been adopted by blockchains such as Zcash [14] and on-chain smart contract systems such as Tornado Cash. Such systems allow the anonymity set of each transaction to potentially equal the entire set of all previous transactions. General-purpose zero-knowledge proofs of the type applied here are more commonly referred to in the industry and academic community as "ZK-SNARKs".

2.2. ZK-SNARKs

ZK-SNARKs are a technology that allows a prover to prove mathematical claims about some combination of public data and private data that the prover holds, in such a way that it satisfies two key properties:

- *Zero-knowledge*: Nothing about the private data is revealed, aside from the fact that the private data satisfy the claim that is being proven.
- *Succinctness*: The proof is short (in bytes) and can be verified very quickly, *even if* the underlying claim being proven involves a heavy and time-consuming computation.

ZK-SNARKs have received much attention among blockchain communities for both of these reasons. What is very time- and resource-intensive is the initial trusted setup of the public parameters used in the ZK system. However, this initial trusted setup needs to be performed only once for the whole system. The succinctness during the proving is key in use cases of ZK-SNARKs for scalability, such as ZK-rollups [15]. For the privacy use cases we describe here, succinctness is not as important, but the zero-knowledge aspect is essential.[1]

The "claim" proven by a ZK-SNARK is expressed as a type of program that is often called a "circuit". Mathematically, it suffices to think of it as a function $f(x, w) \rightarrow \{\texttt{True}, \texttt{False}\}$, where x is the public input, w is the private input, and $f(.)$ is the function being computed. A ZK-SNARK 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`.

2.3. Example: ZK-SNARKs in Zcash and Tornado Cash-like systems

Minor variations exist between different versions of Zcash and different versions of systems that have been inspired by Zcash, such as Tornado Cash. However, the basic logic that they depend on is very similar. This section describes a simple version that roughly corresponds to how these protocols work.

A "coin" consists of a secret s held by its owner. The secret in this context can be any data that is only known to the owner. Two values can be derived from s:

[1] For a more comprehensive introduction to ZK-SNARKS see Refs. [16,17].

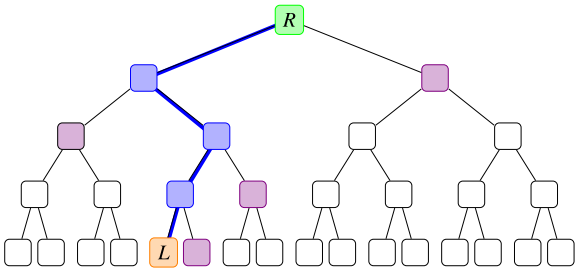

Fig. 1. Structure of a Merkle tree, highlighting the Merkle branch for a given value in the tree. Orange is the leaf L that is being proven; the bottom row of the tree represents the entire dataset. Green is the root hash R. Blue is the path from the leaf to the root. Purple are the sister nodes at each level. Note that the path can be computed by starting with the leaf and hashing it together with the sister node at each level, so there is no need to provide the path itself.

- The public "coin ID" $L = hash(s + 1)$
- The nullifier $U = hash(s + 2)$

The term *hash* refers to a cryptographic hash function, such as SHA256. Given s, the coin ID and the nullifier can be computed. Given a set of nullifiers and public coin IDs, however, the pseudo-random behavior of the hash function ensures that one cannot tell which nullifier is connected to which coin ID unless one knows the secret s that generated both.

The blockchain keeps track of all coin IDs that have already been "created" and all "nullifiers" that have already been "spent". Both sets are ever-growing (unless the protocol wishes to enforce a time limit on when coins must be spent).

The set of coin IDs is stored in a data structure called a Merkle tree: if the tree contains N items, then each adjacent pair of items is hashed (leading to $\lceil \frac{N}{2} \rceil$ hashes), each adjacent pair of those hashes is hashed (leading to $\lceil \frac{N}{4} \rceil$ hashes), and so on until the entire data are committed to a single "root hash".

Given a particular value in the tree and a root hash, one can provide a Merkle branch: the "sister values" that were hashed together at each step along the path from that value to the root. This Merkle branch is useful because it is a small ($\log_2(N)$ hashes) piece of data that can be used to prove that any particular value actually is in the tree. Fig. 1 shows an example of a Merkle tree with a height of 4.

When a user sends a coin to someone else, they provide (i) the nullifier U that they want to spend, (ii) the new coin ID L' of the coin that they want to create (they would ask the recipient to give them this), and (iii) a ZK-SNARK.

The ZK-SNARK contains the following private inputs:

- The user's secret s
- A Merkle branch in the coin ID tree, proving that the coin with the coin ID $L = hash(s + 1)$ was actually created at some point in the past

It also contains the following public inputs:

- U is the nullifier of the coin being spent
- R is the root hash that the Merkle proof is checking against

The ZK-SNARK proves two properties:

- $U = hash(s + 2)$
- The Merkle branch is valid

Outside of the ZK-SNARK, the protocol also checks that

- R is a current or historical root hash of the coin ID tree
- U is not in the set of already-spent nullifiers

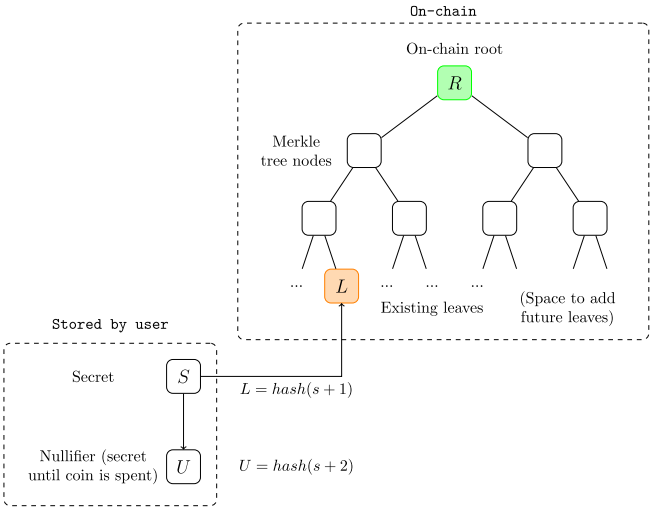

Fig. 2. Some of the data structures involved in a privacy-preserving coin transfer system. The Merkle tree shown is the coin ID tree; the nullifier set is not shown but is also stored on-chain. While a given coin exists but has not yet been spent, the coin ID (L) is on-chain, but the secret (s) and the nullifier (U) are only known by the holder of the coin.

If the transaction is valid, it adds U to the set of spent nullifiers, and L' to the list of coin IDs. See Fig. 2 for an illustration.

Revealing U prevents a single coin from being spent twice. However, *no other information is revealed*. All that the outside world sees is *when* transactions are being sent; they gain no knowledge about the pattern of who is sending or receiving these transactions, or which coin is the "same coin" as the previous coin.

There are two exceptions to the above pattern: *deposits* and *withdrawals*. In a deposit, a coin ID is created without requiring some previous coin to be invalidated. Deposits are not privacy-preserving in the sense that the link between a given L and the external event that allowed the L to be added (in Tornado Cash, a deposit of ETH into the system; in Zcash, new ZEC coins being mined) is public. In other words, deposits are connected to their past transaction history. In a withdrawal, a nullifier is consumed without adding a new coin ID. This can break the withdrawal's link to the corresponding deposit and, by extension, to the past transaction history. However, withdrawals can be linked to any future transactions occurring after the withdrawal event [2].

The first version of Tornado Cash had no concept of internal transfers; it *only* allowed deposits and withdrawals. Later versions, still in the experimental (alpha) stage, also allow internal transfers and coins of arbitrary denominations, including support for the "splitting" and "merging" operations that the handling of arbitrary denominations requires. We will discuss how to extend both basic privacy-preserving coin transfer systems and Privacy Pools to the arbitrary-denomination context in a later section.

2.4. ZK-SNARKs in Privacy Pools

The core idea of Privacy Pools is this: Instead of merely zero-knowledge-proving that their withdrawal is linked to some previously-made deposit, a user proves membership in a more restrictive *association set*. The association set could be the full subset of previously-made deposits, a set consisting only of the user's own deposit, or anything in between. The user specifies the set by providing a Merkle root of the set as a public input.

For simplicity, we do not directly prove that the association set is actually a subset of the previously-made deposits; instead, we just require the user to zero-knowledge-prove two Merkle branches using the same coin ID as the leaf in both cases: (i) a Merkle branch into R, the root of the total set of coin IDs, and (ii) a Merkle branch into the provided association set root R_A. This is illustrated in Fig. 3.

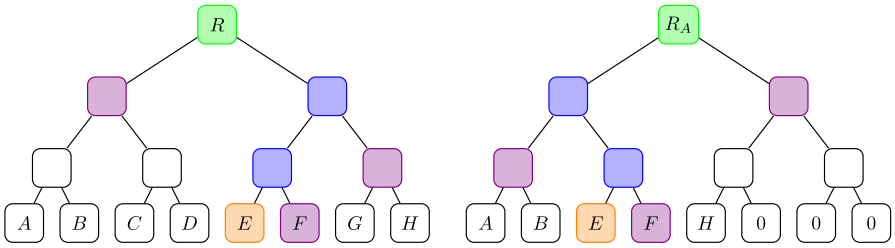

Fig. 3. A user zero-knowledge-proves two Merkle branches: One proving that their coin ID is somewhere in the coin ID tree, and another proving that the same coin ID is somewhere in the tree representing the user's provided association set (represented by its root R_A).

The intention is that the full association set will be made available somewhere, either on-chain or in another location. This is the core concept: Instead of requiring the user to specify exactly which deposit their withdrawal came from, or on the other extreme providing no information at all beyond a proof of non-double-spending, we let the user provide a set of possible origins of their funds, and this set can be as wide or narrow as they wish. We encourage an ecosystem to form that makes it easier for users to specify association sets that align with their preferences. The rest of this paper will merely describe the infrastructure on top of, and the consequences of, this simple core mechanic.

3. Practical considerations and use cases

After this technical introduction, we now turn to the application side and analyze how privacy-enhancing protocols could be used in practice.

3.1. Use cases of association sets

To illustrate the value of this scheme in a law enforcement context, let us consider a simple example. Suppose that we have five users: Alice, Bob, Carl, David, and Eve. The first four are honest, law-abiding users who nevertheless want to preserve their privacy, but Eve is a thief. Suppose also that this is publicly known. The public may not know Eve's real-world identity but they have enough evidence to conclude that the coins sent to the address that we are labeling "Eve" are stolen. This is often the case in practice: Most of the illicit funds that have been identified flowing into Tornado Cash have come from a DeFi protocol exploit, an event that is visible on the public blockchain.

When each of the five users withdraws, they have the choice of which association set they specify. Their association set must include their own deposit, but they can freely choose which of the other addresses to include. Let us first consider the incentives of Alice, Bob, Carl, and David. On the one hand, they want to maximize their privacy. This pushes them toward making their association sets larger. On the other hand, they want to reduce the chance that their coins will be viewed as suspicious by merchants or exchanges. They have an easy way to do this: They do not include Eve in their association set. Therefore, for all four of them, the choice is clear: Make their association sets {Alice, Bob, Carl, David}.

Eve, of course, also wants to maximize her association set. However, she cannot exclude her own deposit and is therefore forced to make her association set equal to the set of all five deposits. The participants' association set selection is shown in Fig. 4.

Despite the fact that Eve herself provides no information, by a simple process of elimination, we can make a clear inference: Withdrawal #5 could only have come from Eve.

3.2. Association set construction

The previous section illustrates one possible way to use association sets in Privacy Pools-like protocols and how honest actors can disassociate themselves from bad actors. Note that the system does not rely on altruism on Alice, Bob, Carl, and David's part; they have a clear incentive to prove their disassociation.

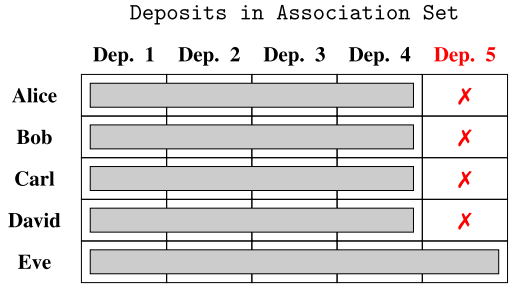

Fig. 4. The gray area in each row represents the respective user's association set. In our simplified example, we assume that Alice, Bob, Carl, and David include all other "good" deposits in their respective association sets and exclude deposit 5, which originates from a known illicit source. Eve, on the other hand, cannot create a proof that disassociates her withdrawal from her own deposit.

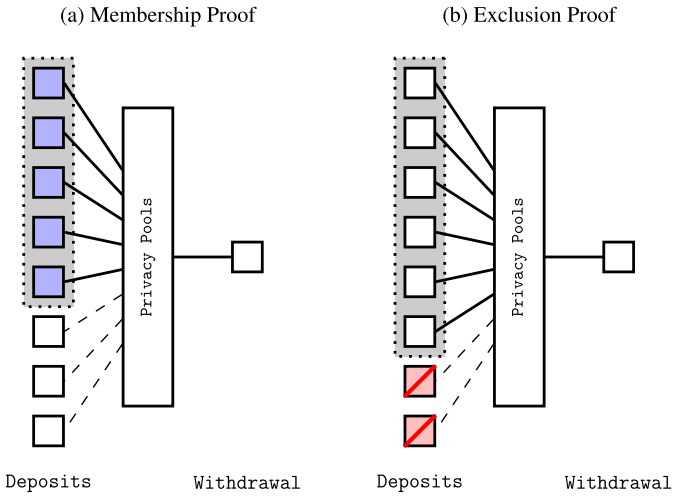

Fig. 5. The membership proof includes a specific collection of deposits in its association set, while the exclusion proof's association set consists of anything but a specific collection of deposits. From a technical perspective, they are identical as both prove against the Merkle root of an association set.

Now, let us have a closer look at the construction of association sets. In general, there are two major strategies for generating association sets. They are described as follows and visualized in Fig. 5.

- **Inclusion (or membership):** Identify a specific set of deposits for which we have specific evidence to believe that they are *low-risk* and construct an association set containing *only those deposits*.
- **Exclusion:** Identify a specific set of deposits for which we have specific evidence to believe that they are *high-risk* and construct an association set containing *everything but those deposits*.

In practice, users will not manually pick and choose deposits to include in their association set. Rather, users will subscribe to intermediaries, which we can call association set providers (ASPs), which

generate association sets that have certain properties. In some cases, ASPs can be constructed entirely on-chain, with no human (or AI) intervention required. In other cases, ASPs would generate association sets on their own and publish the association sets either on-chain or in another location.

We strongly recommend that at least the Merkle root of the association set should be published on-chain; this removes the ability of malicious ASPs to engage in certain types of attacks against users (e.g., giving different users different association sets in an attempt to deanonymize them). The sets as a whole should be available either by API or ideally on a low-cost decentralized storage system such as an IPFS.

The ability to download the entire association set is important because it allows users to generate proofs of membership in the association set locally without revealing any extra information, even to the ASP, about which deposit corresponds to the withdrawal that they are making.

The following are some possible constructions for how ASPs might operate in practice:

- **Add with delay, exclude bad actors**: Any deposit is automatically added to the association set after a fixed period of time (e.g., 7 days); however, if the system detects that a given deposit is connected to known bad behavior (e.g., large-scale thefts or addresses on a government-published sanctions list), the deposit is never added. In practice, this could be implemented through either community-curated sets or existing transaction screening service providers that already perform the work of identifying and tracking deposits connected to bad behavior.
- **$N per month per person**: To join the association set, a deposit's value must be less than a fixed maximum, and the depositor must zero-knowledge-prove that he/she holds some proof-of-personhood token (e.g., either a government-backed national ID system or a lighter mechanism, such as social media account verification). To prevent Sybil attacks (structuring of payments), a nullifier mechanism, with an extra parameter mixed in representing the current month, is used to ensure that each identity can submit a deposit into the association set exactly once per month. This design attempts to implement the spirit of many common anti-money laundering (AML) rules today, where low-value payments below a certain threshold are allowed a much higher level of privacy than high-value payments. Note that this can be implemented entirely as a smart contract, requiring no manual oversight to maintain the ongoing operation.
- **$N per month per trusted community member**: The same as $N per month per person but more restrictive: a user must prove membership in a high-trust community. The high-trust community agrees that its members provide privacy for each other.
- **Real-time AI-based scoring**: AI ASP systems could provide a risk score for each deposit in real-time, and the system would output an association set containing those deposits whose risk score is below a certain threshold. Potentially, the ASP could output multiple sets corresponding to multiple risk score threshold levels.

4. Further technical details

In this section, we analyze how the proposal could support arbitrary denominations and discuss special cases like re-proofing, bilateral direct proofs, and sequential proofs.

4.1. Supporting arbitrary denominations

The aforementioned simplified, privacy-preserving coin systems only support coin transfers in the same denomination. Zcash supports arbitrary denominations using an unspent transaction output (UTXO) model. Each transaction can have multiple inputs (requiring the publication of a nullifier for each input) and multiple outputs (requiring the publication of a coin ID for each output). Each coin ID created must come with an encrypted denomination value. In addition to proving the validity of the nullifiers, each transaction must also come with an additional proof proving that the sum of the denominations of the coins being created does not exceed the sum of the denominations of the coins being spent. Fig. 6 illustrates this additional proof.

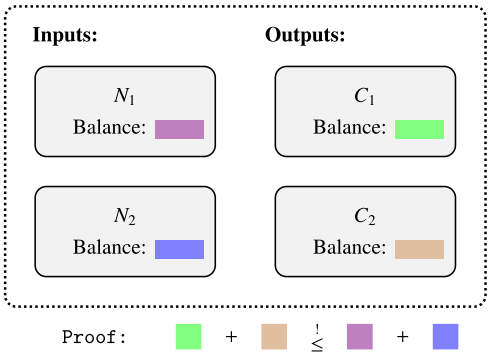

Fig. 6. The ZK-SNARK proves an additional claim that the encrypted denominations represent numbers such that the sum of the numbers on the output side does not exceed the sum of the numbers on the input side. Depending on the construction, it may also require an explicit proof that all of the newly created coin denominations are non-negative.

This design can be extended to support deposits and withdrawals by simply treating the deposit as an (unencrypted) input and the withdrawal as an (unencrypted) output. Alternatively, the design can be restricted to simplify the analysis. For example, one could allow *only* partial withdrawals, allowing transactions to have one encrypted input and two outputs: one unencrypted output representing the withdrawal and an encrypted "*change*" output representing the remaining funds, which can be used in future withdrawals.

A natural question arises about how this design can be extended to support Privacy Pools. Simply plugging it into Privacy Pools "as-is" is not ideal because the transaction graph does not align with what we intuitively expect: if a user makes a deposit of 10 coins and then spends it in four successive withdrawals of 1+2+3+4 coins, what we *want* is to treat all four withdrawals as having the original 10-coin deposit as a source. But what we *get* is shown in Fig. 7, the first withdrawal has the 10-coin deposit as a source, but the second withdrawal has the 9-coin change output created by the first withdrawal as its source, and so forth. This leads to problems in practice because the ASP must verify the intermediate deposits and add them to its association set.

If we want all four withdrawals in this example to be able to claim the original 10-coin deposit as their source, we need to solve two problems at the same time: (i) ensure that each partial withdrawal is not publicly linked to the others and (ii) allow each partial withdrawal to claim the deposit as a member of its association set.

If we only support partial withdrawals (and not more complicated multi-in/multi-out transactions), ensuring that each withdrawal has a single defined corresponding "original deposit", then there are many ways in which we could do this directly. One natural, and very extensible, approach is to propagate some commitments to information through the transactions. For example, we could require a transaction to contain a commitment $hash(coinID + hash(r))$, adding some random value r for blindness, and require the ZK-SNARK to prove that the commitment in a transaction commits to the same value as its parent, if the parent itself is a withdrawal or simply commits to the original deposit's coin ID, if the parent is a deposit. As a result, each transaction in the chain would have to contain a commitment to the original deposit coin ID, and this value would be proven to be included in the transaction's provided association set.

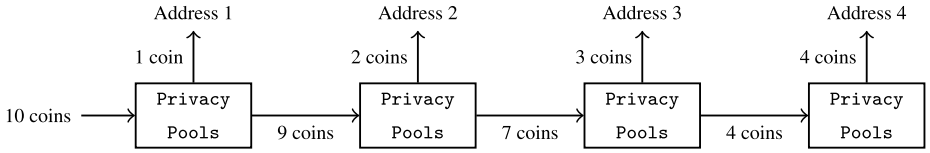

Fig. 7. In the UTXO graph, it appears that each withdrawal's source is the change output of the previous partial withdrawal. But in an economic sense, the "real" source in each case is the original deposit.

To improve privacy against balance-summing attacks (e.g., if I deposit 10 coins, and then withdraw 7.2859 and later 2.7141, those two withdrawals could be correlated based solely on the amounts), we may want to also support *coin merging*: If I have a few coins left, I could merge them along with my next deposit. To adapt to such a scenario, we could require the transaction to commit to a *set* of coin IDs and a transaction with multiple inputs to commit to the *union* of its parents. A withdrawal would contain a proof that *all* of its committed coin IDs are in its association set.

4.2. Special cases

4.2.1. Re-proofing

To withdraw a deposit from a Privacy Pools-like protocol, the user needs the secret deposit information s. The same secret information is then used to construct association set membership proofs. Consider a situation where Alice withdrew her funds, and created and published an association set membership proof. Later, she would like to spend her funds at a merchant that requires a proof against a different set. As long as Alice holds on to her secret information, she will be able to generate a new proof against the merchant's association set. Similarly, Alice could generate a new proof against an updated version of the initial association set. Keeping the secret information around gives Alice more flexibility but may introduce an additional risk of compromising Alice's privacy if the secret is leaked at any point.

Another scenario arises in the context of investigations of a specific event. Suppose that some bad action involving on-chain coins takes place, and an initial investigation reveals a set of possible inputs that those coins could have come from. This could be because the coins in question came from a withdrawal whose association set was a small community or because of a combination of on-chain evidence and other evidence that revealed partial information about who was behind the event. In this case, the other members may want to prove their exclusion from that event to prove their innocence, and the perpetrator's identity would be revealed. Alternatively, if an event is controversial but many people support it even if they are not responsible themselves, they could refuse to provide such a proof.

4.2.2. Bilateral direct proofs

In some scenarios, a user may need to disclose the precise origin of their withdrawal to another party. For example, if Alice wants to deposit her withdrawn funds with a bank, the bank might ask for full information about the funds' origin. In response, Alice can create an association set that contains her deposit only and construct a proof against this set. We expect these proofs to be the exception, and they only contribute towards partial privacy if they are shared bilaterally. Moreover, sharing this proof presumes a strong trust assumption that the recipient will not distribute it further.

Another more advanced option is that Alice zero-knowledge-proves that one of the following statements is true: (i) "this withdrawal is in this association set", (ii) "I am the bank", or (iii) "according to this specific timestamping service (can be a server or a blockchain), more than 10 seconds have passed since the creation of this proof". Only the bank, which receives the proof in real-time (iii) and knows that they did not create the proof themselves (ii), would be able to trust the proof: If the proof lands in someone else's hands, it would be difficult to convince the recipient that the proof is not forged. This eliminates most of the counter-party risk regarding the leakage of privacy.

4.2.3. Sequential proofs

Let us imagine a long-term future scenario in which Privacy Pools-like systems are not merely used occasionally but rather are used *in the vast majority of transactions*. This is the world that is desired by privacy-first systems like Zcash. It introduces some new complexities that do not appear in the world where Privacy Pools is used occasionally.

To adapt to such a world, the following protocol modification would be required: Along with the deposit and withdrawal transaction types, the protocol would need to support an *internal send* operation, which consumes an existing coin ID and generates a new coin ID owned by someone else. From a protocol analysis perspective, this is equivalent to the sender withdrawing into the recipient's address and then the recipient immediately re-depositing, but it increases efficiency by reducing the number of steps and on-chain proofs from two to one.

Suppose that Alice sends a coin to Bob; that is, she makes an internal send that (perhaps partially) consumes a coin ID owned by Alice and creates a new coin ID with parameters provided by Bob. Bob then wants to immediately spend the coin, sending it to Carl, and he would prefer his spending transaction to be private as well. Here, we have our challenge: *inclusion delays*. In many of the configurations we proposed above, ASPs would not be willing to immediately add Bob's new coin to their association set because they need to watch for the possibility that the source of funds is not Alice, but instead someone who just stole the funds from Alice's wallet. The inclusion delay is there to give Alice time to report the incident or third parties time to detect it.

In another similar use case, "Alice" is a DeFi protocol, and Bob wants to withdraw funds from the DeFi protocol and immediately use those funds to privately pay Carl. This scenario has one fewer human being but is otherwise structurally very similar.

In a rapidly transacting economy, the same funds could move around multiple times per week or even more frequently, and inclusion delays would pose a serious challenge. One possible solution for this problem may simply be as follows: In the case where no coins in a user's wallet are "mature" enough (not yet included in a relevant association set), the user could just send them through a non-privacy-preserving transaction. However, we propose a different alternative that leaks less information.

When Bob pays Carl, Bob also directly gives Carl the Merkle branch and secret that were used to generate the payment. This allows Carl to see what Bob sees: that the payment from Alice was in the history of the coin. If, later on, it turns out that a large number of coins associated with some bad actors were deposited and quickly re-circulated, Carl would be able to prove that his coins came from an ultimate source that was disconnected from the bad actor.

If Carl then sends the coins to David, he would pass along the Merkle branch and secret from Bob, and would also add his own. Now, suppose that David next sends his coins to Emma, but by the time he does this, the deposit that Alice made has been added to the association set. Then, David no longer needs to provide the Merkle branch or secret from Alice; instead, he can simply generate an association set membership proof on Alice's behalf. Once Bob's payment is added to the association set, Bob's Merkle branch and secret similarly become obsolete. The concept revolves around ensuring that each user acquires only the essential and minimal information required to have confidence in the funds they receive. Fig. 8 illustrates this example.

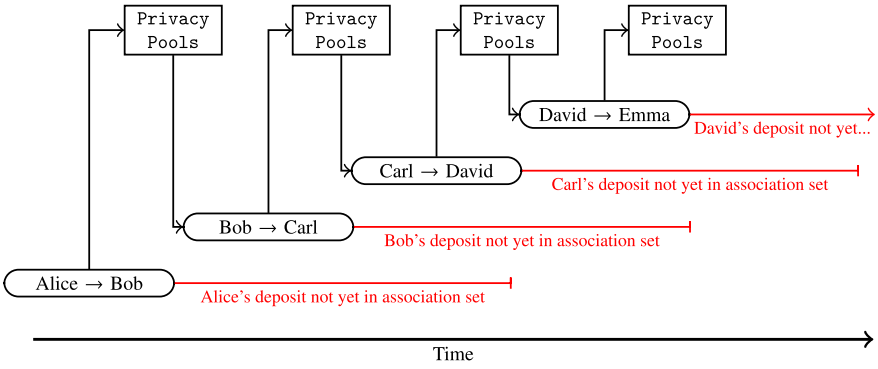

Fig. 8. When David sends his transaction to Emma, he needs to provide the Merkle branch and secret from himself, Carl, and Bob, but not Alice, because Alice's payment to Bob is now in the association set.

In practice, a coin may have multiple "sources". Perhaps Bob is a coffee vendor and received 5 coins from Alice, 4 coins from Ashley, and 7 coins from Anne, and at the end of the day he needed to send 15 coins to Carl to pay for dinner. David, in turn, perhaps received 15 coins from Carl and another 25 coins from Chris, and wanted to deposit 30 coins to Emma, who is an exchange. In these more complicated cases, we follow the same principle: History that is old enough that it has been added to association sets can be ignored, and history that is more recent needs to be passed forward.

5. Discussion

Privacy Pools-like systems allow users to achieve more privacy around their financial transaction data while retaining the ability to prove their disassociation with known illicit activity. We expect that honest users will be incentivized to participate in such a scheme by a combination of two factors: (i) the desire for privacy and (ii) the desire to avoid suspicion.

5.1. Societal consensus and association sets

If there is a perfect consensus on which funds are "good" and which are "bad", the system will lead to a simple separating equilibrium. All users with "good" assets have strong incentives and the ability to prove their membership in a "good"-only association set. Bad actors, on the other hand, will not be able to provide that proof. They could still deposit "bad" funds into the pool, but would not provide them with any benefits. Everyone could easily identify that the funds have been withdrawn from a privacy-enhancing protocol and see that the withdrawal references an association set that includes deposits from questionable sources. More importantly, the "bad" funds would not taint the "good" funds. When funds from legitimate deposits are withdrawn, their owner can simply exclude all known "bad" deposits from their association set.

In cases where there is no global consensus and the conclusion on whether funds are perceived as "good" or "bad" depends on the societal perspective or the jurisdiction, association sets could differ significantly. Let us assume that there are two jurisdictions with distinct rule sets. Subject to jurisdictions, A and B could both use the same privacy-enhancing protocol and choose to issue a proof that satisfies their respective jurisdiction's requirements. Both could easily achieve privacy within their own association set and exclude withdrawals that are not compliant under the respective jurisdiction. If necessary, one could issue a membership proof against the intersection of both association sets and thereby credibly demonstrate that the deposit corresponding to their withdrawal is in line with the requirements of both jurisdictions.

As such, the proposal is very flexible and should be regarded as neutral infrastructure. On the one hand, it is censorship-resistant. It allows anyone to affiliate with the association set of their choosing and remain private within their own community. On the other hand, outsiders can ask for proofs against specific association sets that are in compliance with their regulatory requirements. Therefore, even if there was a community of bad actors within the privacy-enhancing protocol, they could not obfuscate the questionable source of a deposit as long as the information is reflected accurately in the construction of the association set.

5.2. Association set properties

Association sets require certain properties for them to be effective. The sets need to be *accurate* so that users can trust that they can safely spend their funds after withdrawing them. In addition, the properties of each set should be *stable*, meaning they are unlikely to change over time. This limits the need for re-proofing withdrawals against new sets. Finally, to achieve meaningful privacy, it is important to ensure that the association set is sufficiently *large* and includes a wide *variety* of deposits. These characteristics are, however, in conflict with each other. Generally, large and diverse sets may have better privacy properties but are likely to be less accurate and stable, while smaller sets are easier to maintain but provide less privacy.

5.3. Practical considerations and competition

Regulated entities that accept crypto assets must ensure that the laws and regulations they are subject to permit the acceptance of such funds. Today, many of these entities rely on so-called *transaction screening tools*: software or services that analyze the blockchain to identify potentially suspicious activities, connections to illicit addresses, or other non-compliant transactions. Screening tools typically express the risk associated with each transaction through a risk score. This score is based on the destination of the transmitted funds and their transaction history. Privacy-enhancing protocols can be a challenge in that regard. They remove the visible link between deposits and withdrawals. Hence, in the presence of a privacy-enhancing protocol, a risk score would have to consider the proofs and assign a score based on the association set.

The tools and services for transaction screening are mainly provided by specialized companies with expertise in both blockchain analysis and relevant legal fields. Ideally, these companies (and everyone else) have access to all membership proofs and their corresponding association sets to provide accurate risk scores across all transactions. We therefore suggest that all proofs be stored on the blockchain or in another publicly accessible proof repository. The only exception is membership proofs of size one that are shared with a specific counterparty. For obvious reasons, these proofs should not be publicly available.

Having the proofs readily available on-chain introduces additional transaction costs but reduces the coordination effort, levels the playing field, and mitigates the risk that screening tool providers could have a quasi-monopoly due to their knowledge of non-public proofs.

The general setup of Privacy Pools is very flexible. By creating specific association sets, the protocol can be customized to suit a large variety of use cases. Here are two examples of such specialized association sets. (i) A consortium of commercial banks could create an association set that only includes their customers' deposits. This guarantees that any withdrawal creating a proof against this set has undergone the know your customer (KYC) and AML procedures at one of the banks involved but does not reveal which withdrawal belongs to which customer. (ii) In cases where a financial intermediary must document the precise source of funds, they can request the user to provide proof against an association set that only includes the user's deposit. This proof is then exchanged bilaterally with the intermediary, enabling them to track the funds as though the user never utilized Privacy Pools. While this requires the user to trust that the intermediary will not disclose the proof, ideally, it allows the user to comply with regulations, without having to disclose the information to the general public.

5.4. Design choices and alternatives

A setup based on association sets, ZK-proofs, and voluntary disclosure is very flexible. While this is great for ensuring that the proposal can potentially be adapted to various jurisdictions, one should be very careful with respect to specific design choices. In particular, we oppose two potential adjustments. We believe that they are problematic in their trust requirements, and may generate quasi-monopolistic market structures.

In the following, we briefly describe and discuss these alternative approaches.

1. *Centralized access*: Law enforcement agencies, crypto risk scoring providers, or similar actors could get access to see the links between a user's transactions while they remain private from everyone else.
2. *System-wide entry allowlisting*: A privacy system can impose a restriction on what kinds of users can deposit coins into its pool, either requiring them to provide an additional proof or requiring deposits to wait for some time period during which a centralized risk scoring system could reject a deposit.

Both approaches are quite similar, in the sense that they give special privileges to specific entities. This would lead to complex governance questions: Who gets access to this information? Who has the power to manage permissions?

Private firms do not seem to be a good option because any special privileges would likely generate oligopolistic market structures, where a few firms have access to data that would allow them to provide these services, while everyone else would not be able to compete.

Similarly, there would be numerous governance and political questions if the power is given to public institutions, particularly in an international context. Even if a backdoor key is given to an institution that is 100% trustworthy today, does not misuse this power for a political agenda, and has no dependencies on other entities who might pressure it towards misusing its power, it would be naïve to believe that this is a static game. Organizations, their members, nation states, and the political structures within the organization change over time. There might be outside pressure, and the existence of these special privileges may generate additional incentives for bad actors to undermine and gain influence over the organization's governance system.

Moreover, an attack from within or outside the organization or a mistake by a representative of the centralized entity could have far-reaching consequences. We believe that the creation of such a central point of failure should be prevented.

That said, we acknowledge that different transaction sizes and situations may warrant different combinations of proofs. For example, for large transactions, many users will likely end up providing a basic exclusion proof on-chain and additionally provide more detailed information about the source to their counterparty.

5.5. Further research potential

While this study provides an overview of how ZK-SNARK-based privacy-enhancing protocols could be used in a regulated environment, there are several areas that warrant further investigation.

First, it is important to be aware that the privacy obtained through these protocols depends on many different factors. Insufficiently large association sets, inappropriate root choices, and user mistakes may allow a dedicated attacker to link a withdrawal to a specific deposit. Moreover, the choices of other users can adversely affect your own privacy. In an extreme case, everyone else in the pool would publish a membership proof of size one, revealing the direct link between their deposit and withdrawal. Obviously, this would implicitly reveal the link between the only deposit and withdrawal transactions that are left. In a more nuanced example, the constraints from various membership proofs could be used to extract information and potentially link deposits and withdrawals with a high probability. Once the information from these proofs is combined with transactional metadata, the privacy properties of the protocol could be undermined. Last but not least, a malicious ASP could choose to compile the proposed association sets in a way that allows them to maximize the extractable information or inflate the perceived anonymity by adding deposits for which the corresponding withdrawals are known. All of these issues require further research to assess the privacy properties provided. In a similar vein, it would be interesting to further study the properties of the separating equilibrium, model how good and bad actors would behave under certain assumptions, and how *public* proofs of the former would affect the privacy of the latter.

Finally, legal scholars could further investigate specific disclosure requirements. The proposal outlined in this paper is highly adaptable, and insights from legal experts could aid in tailoring the protocol and the ecosystem around it to ensure compliance across various legal jurisdictions.

6. Conclusion

In many cases, privacy and regulatory compliance are perceived as incompatible. This paper suggests that this does not necessarily have to be the case if the privacy-enhancing protocol enables its users to prove certain properties regarding the origin of their funds. For instance, suppose that users can demonstrate that their funds have no ties to deposits from known illicit sources or prove that the funds are part of a specific set of deposits without revealing any further information.

Such a setup can generate a separating equilibrium where honest users are strongly incentivized to prove membership in a given, compliant association set while still enjoying privacy within that set. Conversely, for dishonest users, it is impossible to provide such a proof. This allows honest users to disassociate themselves from third-party deposits that they do not agree with or might otherwise prevent them from using their funds in a regulated environment. We argue that the proposal is quite flexible and can be adapted to potentially satisfy a large variety of regulatory requirements.

The paper should be seen as a humble contribution towards a potential future in which financial privacy and regulation can co-exist. We want to foster a discussion and shift the conversation in a more positive and constructive direction. Cooperation between practitioners, academics from various fields, policymakers, and regulators will be needed to extend and modify this proposal, with the ultimate goal of creating privacy-enhancing infrastructure that can be used in a regulated environment.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: The paper analyzes Privacy Pools-like setups. Ameen Soleimani is a developer of Privacy Pools.

Acknowledgement

Special thanks to Mitchell Goldberg, Katrin Schuler, and Dario Thürkauf for their valuable inputs, Emma Littlejohn for proofreading and Dario Thürkauf for his support with the graphical design.

References

[1] Z. Wang, S. Chaliasos, K. Qin, L. Zhou, L. Gao, P. Berrang, B. Livshits, A. Gervais, On how zero-knowledge proof blockchain mixers improve, and worsen user privacy, arXiv preprint, arXiv:2201.09035, https://doi.org/10.48550/arXiv.2201.09035.

[2] M. Nadler, F. Schär, Tornado cash and blockchain privacy: a primer for economists and policymakers, Fed. Reserve Bank St. Louis Rev. 105 (2) (2023) 122–136, https://doi.org/10.20955/r.105.122-136.

[3] A. Soleimani, Privacy pools, gitHub repository, https://github.com/ameensol/privacy-pools, 2023.

[4] J. Beal, B. Fisch, Derecho: privacy pools with proof-carrying disclosures, Cryptology ePrint Archive, https://eprint.iacr.org/2023/273, 2023.

[5] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf, 2008.

[6] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G.M. Voelker, S. Savage, A fistful of bitcoins: characterizing payments among men with no names, in: Proceedings of the 2013 Conference on Internet Measurement Conference IMC '13, ACM, 2013, pp. 127–140, https://doi.org/10.1145/2504730.2504747.

[7] C. Kang, C. Lee, K. Ko, J. Woo, J.W. Hong, De-anonymization of the bitcoin network using address clustering, in: Blockchain and Trustworthy Systems, Springer, Singapore, 2020, pp. 489–501, https://doi.org/10.1007/978-981-15-9213-3_38.

[8] Ethereum name service, decentralised naming for wallets, websites, & more, https://ens.domains/, 2023.

[9] G. Maxwell, Coinjoin: bitcoin privacy for the real world, https://bitcointalk.org/?topic=279249, 2013.

[10] J.K. Liu, V.K. Wei, D.S. Wong, Linkable spontaneous anonymous group signature for ad hoc groups, in: Information Security and Privacy, Springer, Berlin, Heidelberg, 2004, pp. 325–335, https://doi.org/10.1007/978-3-540-27800-9_28.

[11] B. Goodell, S. Noether, A. Blue, Concise linkable ring signatures and forgery against adversarial keys, https://eprint.iacr.org/2019/654, 2019.

[12] M. Moser, K. Soska, E. Heilman, et al., An empirical analysis of traceability in the monero blockchain, https://arxiv.org/pdf/1704.04299/, 2018.

[13] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: decentralized anonymous payments from bitcoin, in: Proceedings of the 2014 IEEE Symposium on Security and Privacy, IEEE, 2014, pp. 459–474, https://doi.org/10.1109/SP.2014.36.

[14] Zcash, https://z.cash/, 2023.

[15] V. Buterin, An incomplete guide to rollups, https://vitalik.ca/general/2021/01/05/rollup.html, 2021.

[16] M. Petkus, Why and how zk-snark works, CoRR, arXiv:1906.07221, 2019.

[17] A. Berentsen, J. Lenzi, R. Nyffenegger, An introduction to zero-knowledge proofs in blockchains and economics, Fed. Reserve Bank St. Louis Rev. (2023) 280–294, https://doi.org/10.20955/r.105.280-94.