D. Morales, I. Agudo and J. Lopez, "Zero-Knowledge Bitcoin Mixer with Reversible Unlinkability", *Blockchain: Research and Applications*, 2025.
http://doi.org/10.1016/j.bcra.2025.100323
NICS Lab. Publications: https://www.nics.uma.es/publications

Zero-Knowledge Bitcoin Mixer with Reversible Unlinkability

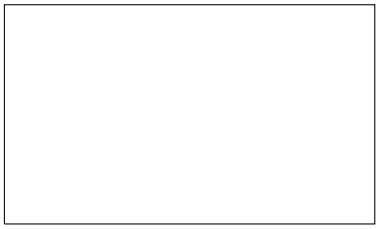
Daniel Morales[b,*], Isaac Agudo[b], Javier Lopez[b]

[a]*Network, Information and Computer Security (NICS) Lab, University of Malaga, Spain*

Graphical Abstract

Zero-Knowledge Bitcoin Mixer with Reversible Unlinkability

Daniel Morales, Isaac Agudo, Javier Lopez



*Corresponding author
Email addresses: damesca@uma.es (Daniel Morales), isaac@uma.es (Isaac Agudo), javierlopez@uma.es (Javier Lopez)

Highlights

Zero-Knowledge Bitcoin Mixer with Reversible Unlinkability

Daniel Morales, Isaac Agudo, Javier Lopez

- We propose a decentralized mixer-based solution to achieve private payments with selective disclosure of data
- A set of conditional disclosers provide payment data to those allowed only, based on unlinkability through a trapdoor function
- The mixer can verify compliance thanks to NI-ZKPs without learning anything about the payment

Zero-Knowledge Bitcoin Mixer with Reversible Unlinkability

Daniel Morales[b,*], Isaac Agudo[b], Javier Lopez[b]

[b] *Network, Information and Computer Security (NICS) Lab, University of Malaga, Spain*

Abstract

Cryptocurrencies, particularly Bitcoin, continue to be the most prevalent use case within the blockchain ecosystem. One of the inherent limitations of blockchain is that it can create a false sense of privacy. All transaction history and the amount of cryptocurrency held are publicly available, and this information can be easily associated with specific individuals. Many works have proposed fully-private solutions, which are ideal but not realistic in many scenarios.

This paper proposes a technical solution that enables private Bitcoin payments by default, but with the option to conditionally disclose payment data. To do so, this solution relies on unlinkability by a decentralized mixer, which can be reversed by a conditional discloser using a trapdoor unlinkability function. The conditional discloser, which also provides accountability of requests, obeys the payer's policies regarding who can access payment data.

To ensure compliance, we propose a mixer that does not learn anything about the payment link, but is guaranteed by Zero-Knowledge Proofs that the payment can be relinked by a specific conditional discloser.

Furthermore, we provide a proof-of-concept implementation of the proofs, using Circom and SnarkJS. We also present a benchmark that demonstrates the feasibility of this solution. It incurs only one additional parameter per on-chain transaction, while the remainder of the verification data is managed off-chain.

Keywords: Bitcoin, Mixer, Unlinkability, Zero-Knowledge Proof

1. Introduction

Blockchain technologies have brought a new computing paradigm in trustless environments based on decentralization, immutability and public verification. From the variety of applications that have been proposed, such as e-voting [1]

*Corresponding author

Email addresses: damesca@uma.es (Daniel Morales), isaac@uma.es (Isaac Agudo), javierlopez@uma.es (Javier Lopez)

or healthcare [2], the original one remains as the most widespread, i.e., digital currencies or cryptocurrencies [3].

All data in blockchain transactions are public by default [4]. To address this issue and guarantee users' right to privacy, many research works have focused on providing technical tools to ensure privacy while maintaining verifiability and correctness in the blockchain ecosystem. More specifically, w.r.t. private payments, solutions have emerged that conceal the link between payers and senders. Some of them, such as Monero [5] or ZCash [6], offer new redesigned blockchains. On the other hand, blockchain mixers provide this service in widely adopted blockchains through an additional layer, e.g., in UTXO-based blockchains like Bitcoin [7, 8, 9], or account-based blockchains like Ethereum [10, 11, 12].

Ensuring full privacy is the best solution for human rights, because it can prevent abuses by companies or governments, or even insecure situations product of the public exposure of the patrimony, for example. However, it is not realistic in many scenarios where some amount of regulation is needed to ensure security. In fact, it is not easy to find the right level of privacy for users without reducing security against criminals, and there is no consensus on it. A clear example is that private payments facilitate money laundering problems [13, Section 1]. Blockchain pseudonyms are already attractive for money laundering [14], which is achieved through the use of many ephemeral addresses and complex cryptocurrency movements. However, transparent transactions allow graph analysis techniques [15], which is not the case when mixers are involved. According to [16], mixers and privacy coins are significant open doors for fraud, unlike crypto-to-fiat interfaces, which are highly secure. Moreover, the survey in [17] highlights the need for future research to focus on the legal traceability and accountability of transactions.

Some efforts are being made to enable compliant payments in cryptocurrency environments, typically through the use of deny lists and Know Your Customer (KYC). For the former, the Treasury's Office of Foreign Assets Control (OFAC) maintains a list of sanctioned addresses for various assets belonging to entities on economic/trade embargo lists[1]. For KYC, regulations in many countries require exchanges to ensure the identification of their customers before accepting transactions. These approaches can be classified as *proactive compliance* and they are not so difficult to implement. Another category is *reactive compliance*, i.e., those procedures that require advanced analysis of transactions such as Customer Due Diligence (CDD), which is identified in [18] as a key safeguard against money laundering. However, this is more difficult to guarantee when it comes to privacy coins, and has even led to some of them being banned from exchange services [19]. Tornado Cash, an Ethereum mixer, provided a user-driven compliance tool to generate proof of funds reports[2], but it was sanctioned by OFAC for being used for money laundering [20, 21].

A clearer consensus is that full transparency by default is not realistic or

[1] https://github.com/0xB10C/ofac-sanctioned-digital-currency-addresses
[2] https://tornadoeth.cash/compliance/

acceptable, not only because it interferes with the human right to privacy or GDPR principles such as data minimization [22], but also because it makes it almost impossible to adopt some use cases in the blockchain ecosystem. A clear example is blind auctions, where providing a public record of bids clearly biases the outcome. In the payment use case, public transactions include the pair of addresses involved and the amount sent. Although blockchain is said to protect user identities through pseudonyms, they can actually be linked to real identities, e.g., by correlating wallet and IP addresses on the node provider side, as long as users do not use privacy mechanisms such as TOR[3]. It is even easier when a user pays for goods or services with cryptocurrency, since the seller will know the total amount of money and the user's entire transaction history. The same happens to the user w.r.t. the seller's wallet. These situations lead to threats to personal and economic safety [4]. To protect against the identity correlation with the wallet address, a typical recommendation is not to reuse the wallet address. However, this lacks usability, and ephemeral wallet addresses can be correlated if they target a common source of funds.

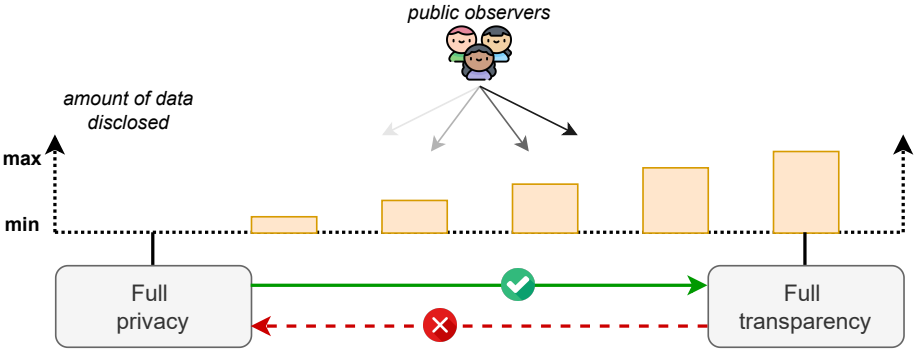


Figure 1: Data exposure levels from a full privacy to a full transparency setting.

It is clear that to address these issues, privacy must be ensured by default, and additional techniques must be used to achieve some level of transparency when necessary. This is a one-way problem, because a system that sets full privacy by default can allow different levels of data disclosure, reaching full transparency in the most extreme case. However, once full transparency is set by default, it is impossible to move towards privacy (see Figure 1).

This paper focuses on how to achieve an agnostic technical solution that provides privacy by default with tailored transparency as desired by the user. Such a solution allows selective disclosure of transaction data, e.g., to accountants, exchanges, investigators, or any desired third party. A solution in this direction already exists in ZCash, where viewing keys allow users to share their blinded transaction data with third parties. However, we note that this does

[3]https://www.torproject.org/

not happen in the Bitcoin network, which is the most capitalized cryptocurrency ($\approx \times 3,744$ the capital of ZCash at the time of writing). Therefore, this work targets a compliant Bitcoin mixer with tailored transparency. In addition, to reduce the impact of money laundering, Bitcoin's UTXO model makes it easier to isolate fraudulent transactions without compromising an entire wallet, compared to Ethereum-based solutions. While a transaction sending fraud ETH contaminates the entire target account because it holds a unique balance field, a UTXO in Bitcoin can remain unspent forever if it is detected as fraudulent.

1.1. Contributions

This paper presents a technical solution for a decentralized Bitcoin mixer with reversible unlinkability, where transactions are private to the public and the mixer view by default, but a trusted conditional discloser chosen by the payer can link the payer's address to the recipient's address on request, as long as the requester is allowed by the payer. This allows the entire spectrum from full transparency to full privacy to be covered in a single solution. In addition, the conditional discloser provides accountability for the identity of the requesters, thus enabling active transparency, i.e., in both directions. Since the mixer is not allowed to see the link between senders and receivers, it must be provided with proofs of compliance that ensure that conditional disclosers can apply reversible unlinkability.

More specifically, this solution is intended for Bitcoin mixers based on mixnets, where a set of semi-honest nodes mix the transactions without obtaining any knowledge of the link. Reversible unlinkability is achieved through a trapdoor function based on public key cryptography, and compliance verification through a Zero-Knowledge Proof (ZKP). Unlike Ethereum, Bitcoin has no smart contracts and its scripting language is very limited. Therefore, we let the data that allows the conditional linking of transactions to be added on-chain, while the ZKP verification is performed off-chain by the mixer.

We provide specific designs and a proof of concept for the proof of compliance protocol based on decryption and secret sharing based mixnets. In addition, a benchmark evaluation is provided to analyze the overhead for the payer.

1.2. Paper organization

The rest of the paper is organized as follows: Section 2 summarizes and compares the main contributions for blockchain private payments in the literature, mainly those based on mixers. Next, Section 3 introduces the technical background, and Section 4 provides our main contribution, i.e., the protocol design, its security requirements, the implementation of the proofs, and a proof of concept. Section 5 presents an evaluation of the proof of concept and a threat analysis. Finally, Section 6 gathers some conclusions and future work.

1.3. Technical overview

In our design, we assume a decentralized mixer service where a set of servers $\boldsymbol{M} = \{M_1, ..., M_N\}$ blind the link between the payer's and the receiver's address. To be compatible with Bitcoin, which does not have the programmability

power of smart contracts in other blockchains, we assume a mixnet layer to deploy the mixer service, either by using a decryption mixnet as described in [9] (which relies on a *decrypt-and-permute* technique) or a secret sharing mixnet as described in [30, 31] (which relies on a *re-randomize-and-permute* technique). In Section 5 we show that secret sharing mixnets are more suitable for our compliance solution in terms of efficiency when generating a valid proof. In addition, threshold signatures are used to protect the clients' funds to be stolen from a single malicious server in the mixer, i.e., a sufficiently large subset of $\boldsymbol{M}$ must cooperate to unlock the UTXO sent by the payers.

Our design relies on two main contributions to solve two problems: achieving (1) reversible unlinkability and (2) compliance.

First, for reversible unlinkability, we rely on trapdoor linkable transactions, which means that the input transaction (sent by the sender to the mixer) and the output transaction (sent by the mixer to the receiver) are computationally linked. We do so by using a trapdoor function and computing two parameters l, l', where $l' = f_\tau(l)$, and appending l to the input transaction and l' to the output transaction. This allows anyone with τ to verify the link, while the relation (l, l') remains undisclosed without knowledge of τ. In our design, we introduce a role named *conditional discloser* (that can be either centralized or decentralized) that has knowledge of τ and can relate transactions processed by the mixer. To instantiate the trapdoor function, we rely on public key cryptography, which is also desirable for the next contribution.

Second, for compliance, we rely on non-interactive ZKP. This allows the payers to let the mixer service knowing that there is at least one conditional discloser that can relink the transaction. For that, the payer proofs that l and l' are related with respect to the key pair of the conditional discloser. However, since the mixer is decentralized, it must not learn that l and l' are related because l' will be added to the output transaction that also contains the address of the recipient. Because of this, the ZKP must ensure that the ciphertext sent to the mixer (prior to the mixing phase) contains a proper pair of data l' and $recipient_{addr}$, without learning their actual values, but also being ensured that l' is generated by encrypting l with the public key of the conditional discloser.

2. Related work

This section discusses the primary research in the field of blockchain mixers, which is summarized in Table 1. A symbol ✓ means that the mixer achieves the specified property, and a symbol ✗ means that it does not. To clarify, the property *anonymity with respect to X* means that X remains unaware on the link $(payer_{addr}, recipient_{addr})$. Therefore, our solution hides the payer address from the recipient, and the link between the payer and the recipient from any public observer and from the mixer itself. With respect to the *availability* property, a symbol ✓ in payer or in mixer means that the payment service protects against the payer or the mixer going off-line and therefore avoiding to complete the payment. Finally, with respect to compliance, *proactive compliance* means that an address can be verified to be fraud before submitting the payment, while

Table 1: Related work comparison. * Unless the whole mixnet cheats. ** As long as 2/3 of the nodes are honest. † If *pk* of a Zether account can be linked to the ETH address, then anonymity is lost. ‡ Only briefly mentioned. **n.a.** Not applies.

	Anonymity (with respect to)				Theft prevention	Account.	Availability		Compliance		#Txs	Bitcoin compatible
	Payer	Recipient	Outsider	Mixer			Payer	Mixer	Proac.	Reac.		
Mixcoin	✗	✓	✓	✗	✗	✓	✓	✗	✗	✗	2	✓
Blindcoin	✗	✓	✓	✓	✗	✓	✓	✗	✗	✗	2	✓
TumbleBit	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	4	✓
CoinJoin	✗	✓	✓	n.a.	✓	✗	✗	n.a.	✗	✗	1	✓
CoinShuffle	✗	✓	✓	n.a.	✓	✗	✗	n.a.	✗	✗	1	✓
CoinParty	✗	✓	✓	✓*	✓**	✗	✓	✓**	✗	✗	2	✓
Möbius	✓	✗	✓	✓	✓	✗	✓	✓	✗	✗	2	✗
Tornado C.	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓	2	✗
Zether	✓†	✓†	✓	✓	✓	✗	✓	✓	✗	✗	1	✗
Haze	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓‡	2	✗
Ours	✗	✓	✓	✓*	✓**	✗	✓	✓**	✓	✓	2	✓

reactive compliance means that the address can be verified even if it became fraud after the payment. A remark is that in CoinJoin and CoinShuffle, the users are those involved in the mixing process. CoinJoin does not achieve anonymity against the users, but CoinShuffle does, unless the whole mixnet cheats.

Some of the early works proposed centralized mixers such as Mixcoin [7] or Blindcoin [23]. In these solutions, the mixer is a TTP that receives the set of k input transactions (with the same amount of funds) and applies a private, randomized permutation to the set of output transactions, accomplishing k-anonymity. However, while the mixer in [7] learns the link between the payer and recipient, in [23] the users are protected against such a situation. Both [7, 23] achieve accountability by providing the payer with a signed warrant by the mixer that works as a commitment contract, stating that the mixer has indeed received the funds. If the mixer steals the funds, the payer can expose the mixer's actions publicly. TumbleBit [24] is also centralized and proposes a tumbler where payers can deposit their funds in escrow using multi-sig addresses with the tumbler. Solving an off-chain puzzle via RSA encryption allows the recipient to obtain 1 BTC previously deposited by the payer in the tumbler.

Despite being accountable, centralized mixers still pose a threat to cryptocurrency theft, so the focus of most novel research has been on decentralized mixers. The earlier works, CoinJoin [8] and CoinShuffle [25], offer solutions where there is no intermediary and all the payers partake in a distributed protocol to mix the transactions, using a permutation mixnet which ensures that no one can link payers to recipients. Nonetheless, these solutions result in an overhead to the payers. To tackle this overhead problem, CoinParty [9] proposes a middle-ground solution, wherein users send and receive transactions via an outsourced and decentralized mixer that uses threshold signatures to avoid a single point of failure. This solution ensures that funds cannot be stolen unless more than 1/3 of the mixer nodes are corrupted, otherwise no valid subset of nodes will agree on generating a signature to spend an UTXO to an invalid address. The mixing method utilizes a permutation mixnet similar to [25], but the mixing is done by the nodes instead of the payers.

Most recent works moved from the Bitcoin world and have benefited from

smart contract capabilities to achieve decentralized secure mixers. Möbius [10] proposes a tumbler smart contract that records deposits from stealth addresses, which are derived from a master secret key agreed upon in advance by the payer and the recipient. When the contract receives k incoming transactions, it forms a ring with the stealth addresses. This allows anyone that can provide a valid ring signature to withdraw the amount sent, with guarantees of k-anonymity. While most solutions offer anonymity for the payer w.r.t. the recipient, Möbius provides anonymity for the recipient w.r.t. the payer. As for Tornado Cash [11], it also utilizes smart contracts for receiving deposits and withdrawal requests, but relies on NI-ZKPs to maintain on-chain privacy and enable correctness verification. Specifically, it accepts blinded notes as deposits and note nullifiers as withdrawals. On the other hand, Zether [12] proposes an Ethereum smart contract that allows confidential balances and transactions using Additive Homomorphic Encryption and NI-ZKPs for encryption correctness verification. However, Zether can facilitate anonymous transfers by directing the payer to send transactions to an anonymity set, wherein all but one of the encrypted amounts will be zero. Another work is Haze [26], which works in a similar manner to Tornado Cash but also addresses compliance. However, it prioritizes proactive compliance over reactive compliance, contrary to our focus in this paper.

Finally, there are solutions such as ZCash [6] or Monero [5] that propose private payments in independent blockchains. While Monero is focused on a fully-private setting by default, ZCash allows payers to select the amount of disclosure they want for their payments. Regarding the building blocks, Monero is closer to Möbius because it relies on stealth addresses and ring signatures to achieve privacy, while ZCash is closer to Tornado Cash by using blinded notes as deposits and note nullifiers as withdrawals, in addition to NI-ZKPs to enable correctness verification. The main difference, as previously stated, is that both Monero and ZCash embed their building blocks for privacy in the blockchain layer, which means that they require the migration of clients to their specific blockchains in order to benefit from their services.

We note that, despite of the different aspects addressed in blockchain mixers, compliance has only been addressed very recently in Tornado Cash and Haze [26]. However, Haze mostly prioritizes compliance before withdraws and Tornado Cash assumes the payer to be the one deciding to generate the compliance proof and keeping it. In addition, almost all solutions focus on providing a full-private setting, but do not facilitate a selective disclosure of data.

Finally, a key remark is that the most recent solutions rely on smart contract capabilities to facilitate hiding the links of the payments. This is mainly an aspect of programmability, since smart contracts are Turing-complete, and also because of the structural design of these blockchains, which are normally balance-based blockchains. Therefore, it is easier to include all the logic needed to achieve privacy (signature or ZKP verifications, encrypted pools of transactions, homomorphic operations, etc.) in the smart contract as if it was a centralized entity and deploying it in a blockchain that implies a decentralized execution. Achieving these operations in an UTXO-based blockchain like Bit-

coin is indeed harder and requires extra features: either some sort of *off-chain layer 2* solution or a complete redesign of the blockchain. Notice that both ZCash (UTXO-based) and Tornado Cash (balance-based) rely on the same concept to achieve privacy and verify correctness, however, while Tornado Cash benefits from smart contract capabilities which are on-chain computation in Ethereum, ZCash requires a complete new blockchain to support these features. Because of this and since we want to include transaction privacy support in native Bitcoin (which remains as the most important blockchain for cryptocurrency payments), our solution is focused on a layer 2-like design such as CoinParty, which allows Bitcoin users to transact privately but with accountability support and backwards compatibility.

3. Technical background

3.1. Blockchain mixers

Blockchain mixers allow payers to send payments to recipients without exposing the link, i.e., with privacy, and they achieve that by anonymity and unlinkability. *Anonymity* [27] guarantees that a subject is unidentifiable within a set of peers in a system, referred to as the *anonymity set*. On the other hand, *unlinkability* ensures that multiple items within an anonymous system are equally unrelated based on prior knowledge. Blockchain mixers are formally introduced in Definition 1.

Definition 1. *A **blockchain mixer** is a service that enables anonymous payments by unlinking transactions. More precisely, the anonymity sets for the payer and the recipient are the set of input and output transactions respectively, i.e.,* $\{tx_{IN_1}, ..., tx_{IN_n}\}$ *and* $\{tx_{OUT_1}, ..., tx_{OUT_n}\}$*. The mixer ensures unlinkability by making sure that any input transaction* tx_{IN_i} *appears to be equally related to any* tx_{OUT_j}*.*

A centralized mixer owns a wallet with address m. It accepts payments from input addresses until the anonymity set reaches a predefined threshold, i.e., $I = \{in_1, ..., in_n\}$. Additionally, it receives the output set $O = \{out_1, ..., out_n\}$ through confidential off-chain channels, which is mixed by a private permutation π. Each incoming payment is sent from m to each address in $\pi(O)$. However, to avoid over-reliance in the mixer, some works propose decentralized mixers based on mixnets.

3.2. Mixnets

A mixing network (mixnet) [28] is composed of a set of nodes $\{M_1, ..., M_N\}$ that, given a set of ordered, encrypted input messages $\{\langle x_1 \rangle, ..., \langle x_n \rangle\}$, outputs the set of decrypted and permuted messages $\{x_{\pi(1)}, ..., x_{\pi(n)}\}$, where π is a private permutation and $\langle x \rangle$ means some kind of ciphertext containing x. The mixing process is performed such that no single M_i can correlate a message from the output set with one from the input set.

3.2.1. Decryption-based mixnet

A decryption-based mixnet is constructed upon the concept of *onion encryption* [29], where each server in the mixnet, denoted as M_i, owns an asymmetric key pair (sk_{M_i}, pk_{M_i}). Moreover, each sender encrypts its message sequentially using the public keys of all servers (Equation 1) before sending $\langle x \rangle_j$ to M_1.

$$\langle x \rangle_j = Enc_{pk_{M_1}}(Enc_{pk_{M_2}}(...Enc_{pk_{M_N}}(x_j))) \tag{1}$$

When the mixnet receives n messages $\langle x \rangle_j$, each server M_i sequentially decrypts its layer, applies a private permutation π_{M_i} to the message list, and sends the outcome to M_{i+1}. At the end, M_N outputs the permuted plaintexts, i.e., $\pi_{M_1}(\pi_{M_2}(...\pi_{M_N}(\{x_1, ..., x_n\})))$, where no server can link x_i to the original sender.

3.2.2. Secret-shared-based mixnet

A secret-shared-based mixnet is built on the concept of *secret-shared shuffle*, introduced in [30] as a 2-party protocol and extended in [31] for the n-party case. Each sender shares its input x in secret-shared form $[\![x]\!]$, i.e., each server receives a share of x. When the servers receive n messages, the input to the mixer has the form $[\![\boldsymbol{x}]\!]$, with $\boldsymbol{x} = (x_1, x_2, ..., x_n)$. In addition, each server M_i holds a private permutation π_i. In the end, after applying the permutations using an MPC protocol, each server holds a secret-shared vector $[\![\boldsymbol{y}]\!]$ that satisfies Equation 2.

$$\boldsymbol{y} = \pi_1(\pi_2(...\pi_N(\boldsymbol{x}))) = (x_{\pi_{[1:N]}(1)}, x_{\pi_{[1:N]}(2)}, ..., x_{\pi_{[1:N]}(n)}) \tag{2}$$

The core block for secret-shared shuffle is a 2-party *share translation protocol* [30], where P_1 and P_2 take no inputs, but at the end of the protocol P_1 holds random vectors $\boldsymbol{a}, \boldsymbol{b}$, and P_2 holds a random permutation π and $\boldsymbol{\Delta} = \pi(\boldsymbol{a}) - \boldsymbol{b}$.

3.2.3. Protection against DoS in mixnets

Given a mixnet composed of a fixed set of servers, e.g., $\boldsymbol{M} = \{M_1, M_2, M_3\}$, which is the minimum needed, it is easy to perform a DoS: it just suffices to block one server in order to deny the whole service. This is obviously because the adversary knows that all the incoming encrypted messages are going the follow the same route, i.e., M_1, M_2 and M_3, respectively. To protect against DoS, typical solutions rely on a larger pool of servers $\boldsymbol{M}$ where, for each message, the client selects a proper subset $\boldsymbol{M'} \subset \boldsymbol{M}$. In private messaging applications, where learning the subset $\boldsymbol{M'}$ is enough for the adversary to know the source of the message, *timing obfuscation* and *cover traffic* techniques are used to avoid an easy identification of $\boldsymbol{M'}$. In the case of this work, we do not target privacy in the same way: the focus is set on k-anonymity. Notice that it does not matter for the adversary to learn the subset $\boldsymbol{M'}$ used by a particular payer as long as there are at least other $k-1$ payers using the same path. What is powerful to protect against the DoS specifically is not letting the adversary to know in advance which subset $\boldsymbol{M'}$ will be selected in a particular round. Therefore, once the adversary learns $\boldsymbol{M'}$, the mixing round has already be completed and

a different subset will be selected for the next one, avoiding effective DoS attacks [44]. This type of solution falls within the area of *proactive security* where the adversary is assumed to corrupt nodes dynamically [45, 46].

3.3. Non-Interactive Zero Knowledge Proofs

A ZKP protocol is utilized by a prover to convince a verifier that a private witness w and a public statement x belong to a relation R. A Non-Interactive ZKP (NI-ZKP) accomplishes the same task via a single message sent from the prover to the verifier, and the protocol consists of a tuple of algorithms (K, P, V) such that:

> **Setup.** $crs \leftarrow K(1^n)$ outputs a common random string.
> **Prove.** $\rho \leftarrow P(crs, x, w, R)$ takes as input a crs, a public statement $x \in L$, a private witness w, and a relation R, and outputs a proof ρ.
> **Verify.** $V(crs, x, R, \rho)$ outputs 1 if the proof ρ is accepted.

Two main properties of ZKP are defined by Equations 3 and 4. The first property (**completeness**) means that, given a true statement for R, an honest prover with a valid witness should convince the verifier, while the second property (**soundness**) means that, given a false statement for R, it is very hard to compute a valid proof. Another property is **zero-knowledge**, meaning the proof reveals nothing about w except the statement's veracity. For this work, we used a specific type of NI-ZKP called Succinct Non-Interactive Arguments of Knowledge (SNARKs) [32], which also fulfill the **succinctness** property, i.e., that the crs and proof size are $poly(\lambda)$, and the verifier executes in $poly(\lambda + |x|)$ time, with λ being the security parameter.

$$Pr\left[V(crs, x, \rho) = 1 \middle| \begin{array}{c} crs \leftarrow K(1^\lambda) \\ \rho \leftarrow P(crs, x, w) \end{array}\right] = 1 \tag{3}$$

$$Pr\left[\begin{array}{c} V(crs, x, \rho) = 1 \\ \wedge (x, w) \notin R \end{array} \middle| \begin{array}{c} crs \leftarrow K(1^\lambda) \\ \rho \leftarrow P(crs, x, w) \end{array}\right] = \frac{1}{poly(\lambda)} \tag{4}$$

4. A Bitcoin mixer with reversible unlinkability

This section introduces a solution to achieve a Bitcoin payment system with full privacy by default, which can be configured by the users to achieve tailored transparency. As discussed in Section 1, the relation between privacy and transparency is one-way, i.e., different levels of transparency can be achieved from full privacy by default, but no privacy can be achieved from full transparency by default.

To the date, public blockchains cannot regulate these levels of transparency, except by requiring node providers not to reveal certain subsets of data. However, this is typically related to law enforcement and blocking malicious services, and has nothing to do with user privacy.

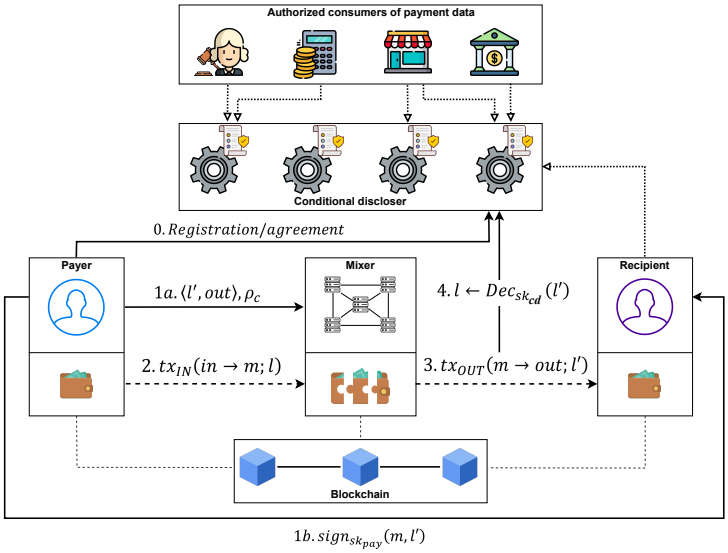


Figure 2: Roles and interactions for a mixer with private and relinkable payments. *Dashed* arrows represent on-chain transactions, while *solid* arrows represent off-chain communication. The wallets interact with the blockchain, and the mixer wallet employs a threshold mechanism.

In the following paragraphs, we introduce a CoinParty-based mixer [9] that is Bitcoin-compatible, together with an additional layer of transparency disclosure control that provides selective disclosure capabilities of payment data and accountability of queries. Such a layer can be considered similar to a Policy Enforcement Point (PEP), which ensures access control (to payment data) only to those authorized by the payer. There is an additional particularity w.r.t. the standard use of CoinParty, which is that the mixer must somehow verify that privacy can be selectively revoked, but without being able to learn the actual payment information. To achieve this, our solution relies on ZKPs.

4.1. Roles and protocol design

The different roles involved in the protocol and the interactions between them are presented in Figure 2. From all the roles, the **payer**, the **recipient**, and the **mixer** are those directly related to the payment. The payer sends some amount of BTC to the recipient through the mixer, which adds the privacy layer. These are the roles that already existed in CoinParty [9]. However, we also add the **conditional disclosers** and the **authorized consumers** of payment data. The former are those in charge of providing selective disclosure of payment data, therefore they are trusted to the payer. The latter, on the other hand, are those willing to learn payment data from a particular payer or recipient. In this work we focus on providing an agnostic technical tool that enables selective disclosure, but it can be applied differently regarding the use case. For example, enabling an accountant to relink the payments is decided

solely by the payer, therefore the accountant is provided access by the allowlist of the conditional discloser trusted by the payer. Another example is to let the mixer having a list of accepted conditional disclosers, therefore refusing to process all payments which are not verifiable to be relinkable by at least one of those disclosers, which may be deployed by regulatory entities.

Regarding the technical aspects, as in CoinParty, we assume a decentralized mixer (a set of parties $\boldsymbol{M} = \{M_1, ..., M_N\}$) with a wallet address m and a mixnet to shuffle outgoing addresses. For simplicity, we will assume for the rest of the paper that mixer and mixnet parties are the same, i.e., $\boldsymbol{M}$, but they can be decoupled. An incoming transaction tx_{IN} from a payer has input address in and output address m. In addition, the address of the recipient out is sent in encrypted form to the mixnet. The mixer wallet private key is secret-shared with all the servers, thus avoiding to spend funds unless the majority agrees. Then, after permuting the output addresses sent by the payers to the mixnet, one outgoing transaction tx_{OUT} is sent by the mixer from address m to each address out. For the selective disclosure of payment information, this solution relies on reversible unlinkability, which is based on the concept of trapdoor-linkable transactions, introduced in Definition 2.

Definition 2 (Trapdoor-linkable transactions). *Two transactions tx_{IN} and tx_{OUT} are linked if tx_{IN} includes a parameter l and tx_{OUT} includes a parameter l' such that $l' = f(l)$, where f is a deterministic function. We say two trapdoor-linkable transactions are related by parameters generated by a trapdoor function f_τ. Without knowledge of the trapdoor τ, the view of (l, l') is indistinguishable from (l, r), where r is a randomly sampled value.*

The linkability parameter l is included in tx_{IN} by the payer (Figure 2, step 2) and l' is included in tx_{OUT} by the mixer (step 3). Since we target trapdoor-linkable transactions, we define $l' = Enc_{pk_{cd}}(l)$, where (pk_{cd}, sk_{cd}) is the asymmetric key pair generated by the conditional discloser and the private key works as the trapdoor. It is important that the pair (l, l') does not expose any additional information than (l, r) without knowledge of sk_{cd}, being r a random value.

To let the mixer learning l' without knowing which payer it belongs to, it is included into the ciphertext that inputs the mixnet, i.e., $\langle l', out \rangle$ (step 1a). In addition, the payer can send an off-chain signed message to the recipient (step 1b) to confirm the address from where the mixer is supposed to send tx_{OUT}.

We note that trust in the conditional disclosers can be easily minimized by accountable decryption based on threshold cryptography [33, 34]. These solutions allow to split private keys into shares, each one held by a different discloser. This implies that more than one discloser must agree in retrieving the link, thus minimizing the probability that a malicious discloser exposes data when a non-authorized consumer queries for it.

In addition to relinkability of transactions, the conditional disclosers provide accountability of the queries received to obtain payment data. In a standard public blockchain, all the history of transactions can be queried through a node

provider, and it may not even request the identity of the requester. However, the conditional disclosers can be equipped with an authentication mechanism such that they log the identity of each requester for a payment and provide such log to the payer.

It remains to introduce how the mixer verifies compliance without learning the payment link, otherwise the payer could just provide dummy (l, l') values such that no transparency can be achieved at all. For that, we introduce the concept of proof of reactive compliance in Definition 3.

Definition 3 (Proof of reactive compliance). *A proof of reactive compliance ρ_c ensures a verifier, in zero-knowledge, that given a first parameter l, there exist a second parameter l' generated with a correct trapdoor τ, s.t. only another designated verifier (who knows τ) can relink both parameters.*

While the conditional discloser is the verifier of the relation (l, l'), the mixer is the verifier of the proof ρ_c, which is computed by the payer and included in the message in step 1a. More specifically, the proof ρ_c is a ZKP that states the following: (1) the private value l' is equal to $Enc_{pk_{cd}}(l)$ and (2) l' is included inside $\langle l', out \rangle$. To verify the ZKP, the mixer uses $\langle l', out \rangle$ and pk_{cd} as public inputs, thus being ensured that the owner of pk_{cd} has the ability to link l and an unknown l', which has not been mixed by the mixnet yet.

Only when a sufficiently large set of incoming transactions are ensured to be compliant, their off-chain messages $\langle l', out \rangle$ are processed by the mixnet, thus returning unlinkable pairs (l', out) to build the outgoing transactions.

To relink a transaction, the conditional discloser just decrypts l' (step 4) and looks for the incoming transactions of that round to see which one holds l.

4.2. Security requirements

First, the conditional discloser is assumed to be trusted for privacy because it can relink all the payments linked with its public key. In addition, authorized consumers trust the conditional discloser to answer their queries, if they are allowed. As introduced above, trust in the conditional discloser can be minimized through decentralization, by requiring more than one discloser to link a transaction.

Regarding the encryption mechanism, it is mandatory to be IND-CPA with randomness hiding, such as ElGamal, thus protecting against anyone computing l' from l and pk_{cd}, which are public values. Since authorized consumers cannot verify the relation (l, l'), conditional disclosers may be requested to provide a ZKP that ensures l' decrypts to l when using sk_{cd}, therefore avoiding malicious disclosers.

With respect to the mixer, it is assumed to be decentralized, thus avoiding a single point of trust, preventing fund theft, and ensuring payment privacy against the mixer. In our design, the mixer can be semi-honest or covert (the latter is like malicious, but tries not to be detected), depending on the underlying cryptographic protocols (refer to Section 4.3). In addition, the mixer is trusted

ZKP 1: Decryption mixnet proof

Public input: l, x, pk_{cd}, $\{pk_{M_i}\}_{i\in\{1,...,N\}}$
Private input: l', out, $\{r_i\}_{i\in\{1,...,N\}}$, r_l
Computation:

1. $l' = Enc_{pk_{cd}}(l; r_l)$
2. $x_0 = (l', out)$
3. For all $i \in \{1, ..., N\}$: $x_i = Enc_{pk_{M_i}}(x_{i-1}; r_{i-1})$
4. $x == x_N$

to verify that each incoming transaction is compliant before completing the payment.

Finally, the payer is assumed to be malicious because she can try to cheat for convincing the mixer that the incoming and outgoing transactions can be linked when they cannot.

4.3. Implementation of compliance proofs

This section describes the design of the proofs using NI-ZKP, which are generated by the payers and verified by the mixer. In each proof, the private inputs are embedded by the payer and are never exposed to the mixer, while the public inputs are introduced by the mixer at verification time. The computation performed inside the proof is neither exposed. For the rest of the section, we let $x = \langle l', out \rangle$ to be the encrypted message sent by the payer to the mixnet. In addition, pk_{M_i} is the public key of the i^{th} mixnet server, and r represents the randomness used for encryption. We devise two proof circuits, depending on the selected approach for the mixnet.

4.3.1. Mixer with decryption mixnet.

The first server of the mixnet is in charge of verifying the proof ρ_c, which must confirm that both (l, l') are linked and that l' is included in the ciphertext x. ZKP 1 outlines the public and private inputs needed and the computation carried out within the proof. The payer is the only party with knowledge of the private inputs. It is important to note that x_0 is the plaintext version of x, and r_i is the randomness for the i^{th} encryption layer. The proof computes (in zero-knowledge) the encryption of l' and the sequence of encryptions to achieve x. The mixer verifies the correctness of the public keys used for the encryption, i.e., pk_{cd} and $\{pk_{M_i}\}$ are public inputs. Consequently, the mixer is guaranteed that the parameters have not only been produced correctly, but can also be decrypted by the designated party. It is noteworthy that this solution involves

ZKP 2: Recursive proof of verification

Public input: x_i
Private input: x_{i-1}, r_{i-1}, ρ_{i-1}
Computation:

1. $x_i = Enc_{pk_{M_i}}(x_{i-1}; r_{i-1})$
2. $true \leftarrow Verify(\rho_{i-1}, x_{i-1})$

computing $n+1$ public key encryptions within the NI-ZKP, which can be quite costly for a large n.

Avoiding a single verifier. As previously stated, the proof is verified by the first server, whose ciphertext includes the encryption with all the layers. This server holds the potential to collude with the payer and assert to the other servers that ρ_c has been successfully verified, even if it has not. While it may be tempting to have all servers verify ρ_c using the public parameter x, this approach incurs the risk of a linkability attack. Specifically, each server M_i could use the public keys from the previous servers to sequentially re-encrypt the ciphertext obtained from M_{i-1}, thereby obtaining x and bypassing the permutations.

We propose a solution based on recursive NI-ZKP, where each mixnet server M_i proves to M_{i+1} that ρ_c has been properly verified. In other words, $\rho_i(\rho_{i-1}(...\rho_c))$ is verified through recursive NI-ZKP, allowing M_i to check the proof being verified inside ρ_{i-1} (sent by M_{i-1}), but with the verified ciphertext x encoded as a private parameter. However, it is necessary to ensure that the partial ciphertext x_i of M_i is verified in ρ_c, as shown in ZKP 2. It is worth noting that this is not problematic since M_i, which computes the proof for M_{i+1}, possesses knowledge of both x_i and $x_{i+1} = Dec_{sk_{M_i}}(x_i)$.

4.3.2. Mixer with secret-shared mixnet

In this approach, the verification is not performed against a ciphertext, but on a secret-shared value, i.e., opening $[\![x]\!]$ allows the retrieval of (l', out). This method introduces a problem that does not exist in the decryption-based scenario: even if the payer proves the validity of a share to the corresponding mixnet server, the payer can cheat by sending shares of different secrets to the other servers, rendering the proof useless and altering the real value of x. A simple but inefficient solution would imply sending independent proofs ${\rho_c}^{(i)}$ to the servers, where ${\rho_c}^{(i)}$ verifies the share $[\![x]\!]_i$. However, this increases computation and communication overheads $\times N$. Therefore, we propose a solution requiring a single proof ρ_c which is verifiable by all servers in $\boldsymbol{M}$. The verification process consists of two phases: (1) a check for share consistency, which requires a light online interaction by the mixnet parties, and (2) the verification of the proof, which is done locally. We now provide the solution for both Additive and Shamir secret sharing schemes.

ZKP 3: Additive secret-shared mixnet proof

Public inputs: $MAC, (g, h), l, pk_{cd}$
Private inputs: $\{[\![x]\!]_i\}_{i\in\{1,...,N\}}$, $\{[\![r]\!]_i\}_{i\in\{1,...,N\}}$, l', out, r_l
Computation:

1. $x = \sum_{i=1}^{N} [\![x]\!]_i$
2. $r = \sum_{i=1}^{N} [\![r]\!]_i$
3. $MAC == g^x h^r$
4. $l' = Enc_{pk_{cd}}(l; r_l)$
5. $x == (l', out)$

Additive secret sharing. The shares are computed as $x = \sum_{i=1}^{N} x_i$, where $N-1$ shares do not reveal the secret. Initially, the payer sends (ρ_c, MAC) to each M_i, where MAC is a Message Authentication Code for the secret instantiated with a Pedersen commitment scheme as defined in Equation 5 (we note that the commitment randomness is also secret-shared). We benefit from the additive homomorphic property of Pedersen commitments to enable every M_i to recompute the MAC using their secret-shares. To achieve this, each M_i broadcasts $(MAC, g^{x_i} h^{r_i})$ to the other servers, thus each server can recompute the MAC using the partial commitments received from the other servers according to Equation 6.

$$MAC = g^x h^r \tag{5}$$

$$\prod_{i=1}^{N} g^{x_i} h^{r_i} = g^{\sum_{i=1}^{N} x_i} h^{\sum_{i=1}^{N} r_i} = g^x h^r = MAC \tag{6}$$

Next, the proof ρ_c (refer to ZKP 3) is locally verified using the MAC. We should note that the public inputs are common to all M_i and, even if the shares are not verified w.r.t. l', the MAC is, leading to a secure verification.

Shamir secret sharing. The shares are based on a secret polynomial of grade t such that $x = P(0)$. Each secret holder is identified with an integer id $j \in \{1, ..., N\}$, and the share for M_j is computed as $P(j)$. Given a sufficiently large number of shares $t+1$, the polynomial can be reconstructed using interpolation, and the secret can be recovered.

First, the payer sends $(\rho_c, \{c_0, c_1, ..., c_t\})$ to every M_j, where c_i are Feldman's polynomial commitments [35], s.t., $c_0 = g^x$, and $g_i = g^{a_i}$, being a_i a coefficient of the secret polynomial $P(y)$. The share consistency check leads every M_i to locally verify its share using Equation 7 and broadcast $\{c_0, c_1, ..., c_t\}$ to the rest of M_j. The slight difference w.r.t. the additive version is that the consistency check is jointly performed using the MAC, i.e., it checks that the shares recon-

ZKP 4: Shamir secret sharing mixnet proof

Public inputs: $\{c_0, c_1, ..., c_t\}$, g, l, pk_{cd}
Private inputs: $\{[\![x]\!]_i\}_{i\in\{1,...,N\}}$, $\{a_1, ..., a_t\}$, l', out, r_l
Computation:

1. $x = Open([\![x]\!]_1, [\![x]\!]_2, ..., [\![x]\!]_N)$
2. Commitment verification:
   (a) $c_0 = g^x$
   (b) For all $i \in \{1, ..., t\}$: $c_i = g^{a_i}$
3. $l' = Enc_{pk_{cd}}(l; r_l)$
4. $x = (l', out)$

struct x. On the other hand, in the Shamir version, each server locally verifies that its share is consistent w.r.t. a secret polynomial, i.e., they do not verify the other servers shares, and for that reason they must agree that the polynomial commitments sent by the payer are the same to each server.

$$g^{[\![x]\!]_i} = \prod_{j=0}^{t} {c_j}^{i^j} = g^{\sum_{j=0}^{t} {a_j}^{i^j}} = g^{P(i)} \tag{7}$$

Next, the proof ρ_c is locally verified using the polynomial commitments. We note that public inputs are common to every M_j and, even if a particular $[\![x]\!]_j$ is not verified w.r.t. l', the commitments are, leading to a secure verification as long as the share consistency check is done correctly.

4.4. Proof of concept

This section presents a proof of concept implementation[4] of the proof of compliance generated by the payer (Section 4.3), which is used in the next section to run a benchmark and test its practical feasibility.

This implementation is assumed to work in tandem with the Bitcoin wallet application and is comprised of two main modules: (1) a cryptographic library, responsible for generating the linkage parameters (l, l'), and (2) a compliance proof generator, which produces the compliance proof ρ_c. Moreover, we assume that the application sends the off-chain messages to the mixer, while the wallet signs and sends the on-chain transactions.

The cryptographic library is implemented in Python and Sagemath (the latter for an easier handling of elliptic curve operations) and it supports the

[4] https://github.com/damesca/zk-mixer

schemes presented in the paper, i.e., a decryption-based mixnet and a secret sharing-based mixnet.

Table 2: Comparison between different zk-SNARKs protocols. Notation: m is the number of wires, n is the number of multiplication gates, a is the number of additon gates, M is the number of non-zero coefficients in the matrices.

Protocol	**Setup**	**Prover work**	**Proof size**
Groth16 [32]	SRS (circuit specific)	$O(n+m)$	$2\mathbb{G}_1 + 1\mathbb{G}_2$
Sonic [36]	SRS (universal)	$O(n\ log\ n)$	$20\mathbb{G}_1 + 16\mathbb{F}_q$
Marlin [37]	SRS (universal)	$O(m\ log\ m)$	$13\mathbb{G}_1 + 8\mathbb{F}_q$
Plonk [38]	SRS (universal)	$O((n+a)\ log\ (n+a))$	$7\mathbb{G}_1 + 6\mathbb{F}$
Fractal [39]	CRS (transparent)	$O(M\ log\ M)$	$O(M)\mathbb{F}$
Bulletproof [40]	CRS (transparent)	$O(n\ log\ n)$	$O(log\ n)\mathbb{G} + 5\mathbb{Z}_p$

Regarding the proof generator, we opted for zk-SNARKs due to their zero-knowledge, succinctness, and non-interactivity. Nevertheless, there are several protocols available, each with distinct characteristics (refer to Table 2). To determine the optimal choice, two main factors are considered: the setup and the proof size, which are closely related. There are three types of setups: (1) *trusted setups*, which generate secret randomness (toxic waste) to create a circuit specific Structured Reference String (SRS), (2) *universal setups*, which also require toxic waste but allow the reuse of the SRS with arbitrary circuits, and (3) *transparent setups*, which create a public Common Reference String (CRS) that does not utilize toxic waste. From Table 2 we note that transparent zk-SNARKs generate the longest proofs, compared to the SRS settings where the proof size remains constant.

In our application, the setup is not an issue since the mixer, i.e., the proof verifier, generates the SRS and the keys, and publicly distributes the proving key to the payers. As long as the secret value from the setup remains undisclosed, no payer can forge a fake proof. Moreover, our application does not require a universal setup since there is only one circuit, i.e., the circuit for ρ_c. After all these considerations, we choose Groth16 as the best option for our application because it provides the fastest proving time and the shortest proof size.

Additionally, Groth16 has been extensively researched and there are quite good developments for its use. This implementation relies on Circom[5] [41] and SnarkJS[6] toolkit. Circom enables writting ZKP programs in a high-level language and compiling them to constraints. On the other hand, SnarkJS facilitates the computation of the proofs. Moreover, Circomlib[7] is a Circom library that includes diverse functionalities such as ECC operations and hash functions. We have implemented new circuits to compute ρ_c, including public key encryption and secret sharing.

[5]https://docs.circom.io/
[6]https://github.com/iden3/snarkjs
[7]https://github.com/iden3/circomlib

There are additional details to consider. This implementation uses altBn128 curve as Groth16 building block. It natively supports operations modulo p, i.e., the curve's field modulo. In this case, field elements are limited to 253 bits. Although this modulo can be expanded, it comes at the cost of introducing bit-wise circuits, hence additional constraints. Since the bottleneck for constraints are SHA2 hash functions, we also consider circuits based on MiMC [42] and Poseidon [43], which are hash functions designed to be efficient in arithmetic circuits.

Decryption-based mixnet proof. Its main block is public key encryption. We have implemented a hybrid encryption scheme, based on EC-Elgamal for key encryption and AES[8] for message encryption. Hybrid encryption is selected as a solution to layered encryption, where ElGamal's output is a pair of points and the input message is a single input.

secret sharing-based mixnet proof. This scenario requires computation for only one encryption, thus computational workload is expected to be light. The additional overhead comes from the additive secret-shares and the commitments. The secret sharing version of the proof has a challenge: x must be shared modulo p. We note that $x = (l', out)$, where l' is a hybrid encryption ciphertext and the address *out* may have, e.g., 20 bytes in Bitcoin. Having field elements limited to 253 bits, concatenation cannot be used to encode these values in x. To solve this issue, we have devised two possible solutions, the first of which has been implemented:

1. To encode $x = (l, out)$ and include l' inside tx_{IN} instead of tx_{OUT}, because l is shorter than l'. However, assuming a P2PKH Bitcoin address of 20 bytes, l is limited to be sampled with 11 bytes maximum.
2. To send l and *out* as different secret shared values, which expands to 253 bits the length of l. However, both secret-shared lists must be ensured to be permuted with the same π.

4.4.1. Integration in Bitcoin transactions

Arbitrary data can be added to Bitcoin transactions inside the PubKey locking script, with a maximum allowed length of 10,000 bytes. More specifically, the DROP opcode allows to remove the previously stacked item so it does not affect the unlocking logic. Therefore, it allows to include l and l' inside the transactions without interfering them, but enabling an off-chain verifier to read both parameters.

We remark that l is a short and randomly sampled value, e.g., 16 bytes for 128 bits of security (but only 11 bytes in the secret sharing-based approach). On the other hand, l' is the ciphertext of a hybrid encryption scheme, therefore it takes 16 bytes for the symmetric ciphertext when using AES and 128 bytes for the key encapsulation when using EC ElGamal (because it takes two curve points, hence 4 scalars of 32 bytes), i.e., a total of 144 additional bytes.

[8] https://github.com/Electron-Labs/aes-circom

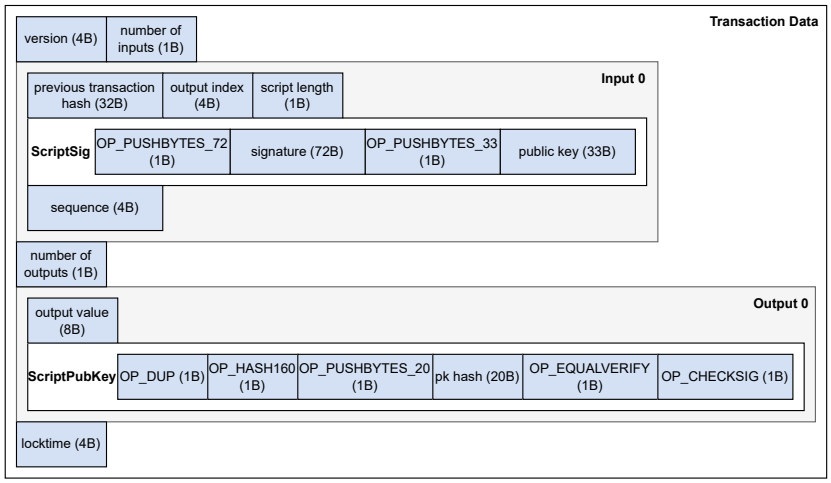


Figure 3: Structure of a basic P2PKH transaction in Bitcoin with one input and one output.

Figure 3 shows the taxonomy of a basic P2PKH transaction in Bitcoin with only one input and one output, where its total length (i.e., the minimum possible) is 191 bytes. To include the linkage parameter, it would suffice to modify the locking script in the output as follows: $\langle linkParam \rangle$ OP_DROP OP_DUP OP_HASH160 $\langle pubKeyHash \rangle$ OP_EQUALVERIFY OP_CHECKSIG. A remark is that this script will be added to both the input transaction (from the payer to the mixer) and the output transaction (from the mixer to the recipient). In the case of the parameter l, it adds a 8.38% overhead of the base size of the transaction. Regarding l', it adds a 75.4% overhead, which is significantly higher. Therefore, it becomes reasonable to assume that l will be included in the input transaction because it is cheaper for the payer. However, if the larger cost of including l' is assumed by the mixer when sending the output transactions, one would expect an increment in the service fees requested to the payers by the mixer. What can be done by the mixer is to send one round of mixed UTXOs inside a single transaction including all the payments to the recipients and therefore amortizing the payment fees.

5. Evaluation and analysis

5.1. Evaluation

This section presents the results acquired through the execution of the proof of concept under diverse settings. The programs were tested using an Ubuntu 22.04.1 LTS virtual machine, operating on an Intel ESXi server with an 8-core CPU (2.2 MHz) and 32 GB of RAM.

The efficiency of the proof varies according to the particular hash function used. Our design employs hashing inside ElGamal encryption (hashed version) and authenticated AES. The number of constraints for the ElGamal block w.r.t.

Table 3: Performance of decryption-based and secret sharing-based ZKPs for transaction compliance. Terminology: *N:* number of nodes, *constr.:* number of constraints in the circuit, *wt:* witness generation time, *pt:* proof generation time, *(σ):* standard deviation. All times are indicated in seconds.

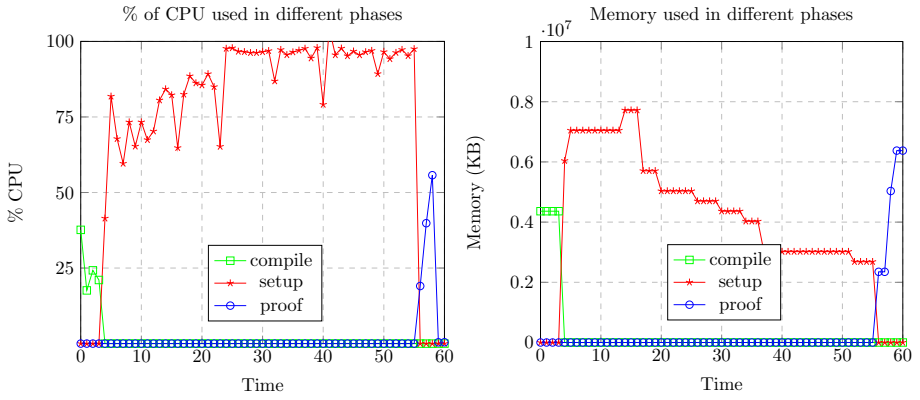
N	**constr.**	**wt (σ)**	**pt (σ)**	**N**	**constr.**	**wt (σ)**	**pt (σ)**
Decryption (sha256)				**Secret Sharing (sha256)**			
3	929,040	8.37 (0.1)	33.10 (0.3)	4	221,950	2.45 (0.1)	9.63 (0.1)
4	1,168,372	10.32 (0.1)	51.90 (0.6)	8	224,008	2.39 (0.1)	9.48 (0.2)
5	1,407,704	12.46 (0.2)	56.27 (0.5)	12	226,514	2.42 (0.1)	9.50 (0.2)
Decryption (mimc7)				**Secret Sharing (mimc7)**			
3	187,476	1.95 (0.0)	9.51 (0.2)	4	36,559	0.77 (0.0)	3.44 (0.1)
4	241,417	2.43 (0.0)	10.31 (0.1)	8	38,617	0.77 (0.0)	3.52 (0.2)
5	295,358	2.85 (0.0)	16.08 (0.2)	12	41,123	0.80 (0.0)	3.58 (0.1)
Decryption (poseidon)				**Secret Sharing (poseidon)**			
3	184,920	6.27 (0.1)	9.20 (0.3)	4	35,920	5.27 (0.1)	3.56 (0.1)
4	238,222	6.79 (0.2)	10.06 (0.2)	8	37,978	5.22 (0.1)	3.55 (0.1)
5	291,524	7.28 (0.1)	15.85 (0.2)	12	40,484	5.26 (0.1)	3.52 (0.1)

the hashing algorithm are the following: 62,906 with sha256, 4,097 with mimc7, and 3,609 with poseidon.

To evaluate the implementation, we conducted different executions of ZKP generation on both the decryption-based and the secret sharing-based mixnet version. In each setting, we chose varying numbers of nodes, with fewer nodes for the first setting compared to the second. There are two reasons for this: firstly, the decryption-based solution allows a higher corruption threshold than the secret sharing-based option, and secondly, the decryption-based setting becomes inefficient before the latter. Table 3 shows the results from several executions, including different hash functions. While the number of constraints does not depend on the number of executions because it is a parameter that depends on the circuit description, for the running times we provide the geometric average over 25 executions per each type of circuit. It is clear that sha256 instances introduce more constraints than mimc7 and poseidon ones. This has a direct impact on the execution time, i.e., generating the witness and the proof.

All the implementations add a constant number of constraints per round in the protocol, rendering them linear. The base cost for the encryption of l with pk_{cd} is fixed for both the decryption-based and the secret sharing-based schemes (although dependent of the hash scheme). Then, even when both scenarios are linear, the decryption-based mixnet grows at a much faster rate than the secret sharing scheme because each round of hybrid encryption imposes significantly more constraints than each new secret holder added in the secret sharing method.

We also gathered data on energy consumption during the proof generation. Figure 4a displays the percentage of CPU used in proof computation, using



(a) Percentage of CPU used

(b) Amount of memory used

Figure 4: Performance metrics to compute a proof of compliance

the decryption-based proof with three layers of encryption and sha256 as the hash function. The figure presents the consumption of each of the three phases: circuit compilation, circuit-specific setup, and witness and proof generation. We note how the setup phase consumes significantly more CPU than other phases, often running at almost full capacity. Nonetheless, it is worth noting that circuit compilation and setup only need to occur once, since our system design does not require circuit modification. In practice, the payer only needs to execute the witness and proof generation functionalities. A similar result is shown in Figure 4b regarding memory consumption, where setup consumes the most.

5.2. Security threats

This section analyzes the security threats of our solution.

If we assume a secure zk-SNARK scheme with a correctly computed setup phase, the soundness property guarantees that each valid proof is truthful but with negligible probability. This impedes a malicious payer to avoid the linkage of the transactions, i.e., tagging tx_{IN} with l and sending $\langle r, out \rangle$ s.t. $r \neq Enc_{pk_{cd}}(l)$, otherwise ρ_c would not pass verification.

It may happen that a malicious mixer colludes with a malicious payer, thus omitting the verification of ρ_c and including a malformed l'. However, as long as the selected conditional discloser is honest, it will be able to notice that l' does not decrypt to any l in the set of incoming transactions in that round. Moreover, it can compute a ZKP ensuring that claim. In addition, a decentralized mixer can be required to post some proof on-chain ensuring it has correctly verified the proofs ρ_c in a mixing round, e.g., by recursive ZKP, therefore solving the "lazy mixer" problem.

If the conditional discloser is malicious, a regulator that queries for l' can be provided with a false l, since it has no way to verify that l' decrypts to l without sk_{cd}. However, a ZKP computed by the conditional discloser can

guarantee correctness. We notice that the conditional discloser cannot forge a false l as long as the mixer correctly verifies ρ_c and adds l' to tx_{OUT}.

Finally, regarding fund theft, we assume a mixer as in CoinParty, where the mixer wallet's private key is secret-shared among all the N mixer parties, therefore requiring $t + 1 \leq N$ shares involved to compute a valid signature. Honest and semi-honest mixer parties will not partake in the signature process unless they affirm the integrity of the mixnet result.

6. Conclusions and future work

In this paper, we have introduced a technical solution for a decentralized Bitcoin mixer with reversible unlinkability, where only a trusted conditional discloser selected by the payer from a set of available disclosers can link the payer's address with the recipient's address of a private payment.

Our solution enables a cryptocurrency space for secure payments with guarantees, where privacy with respect to the public view and the mixer is provided by default. Different levels of transparency can be achieved through a trapdoor unlinkability function using public key encryption and NI-ZKP for compliance proofs. The pool of conditional disclosers allow payers to provide the payment information only to entities they trust.

The design is proven to be secure as long as the payer does not collude with both the mixer and the conditional discloser at the same time, a risk that can be mitigated by decentralization.

The implemented proof of concept ensures the feasibility of this solution, which fits Bitcoin by requiring only one additional parameter in each on-chain transaction, thus avoiding large additional fees for the transactions.

Finally, we leave as future work to reduce the trust in the conditional disclosers through decentralization, using threshold decryption, and also to increase the efficiency of the proofs of compliance through a design improvement of the ZKP circuits. We also intend to embed this solution within Lightning Network as a privacy protection for the settlement payment, since Lightning facilitates fast and cost-effective Bitcoin payments through confidential off-chain channels.

Acknowledgements

This work has been partially supported by the project SecTwin 5.0 funded by the Ministry of Science and Innovation, Spain, and the European Union (Next Generation EU) (TED2021-129830B-I00). The first author has been funded by the Spanish Ministry of Education under the National F.P.U. Program (FPU19/01118).

References

[1] U. Jafar, M. Juzaiddin, A. Aziz, Z. Shukur, J. Wu, H. Wang, Blockchain for electronic voting system—review and open research challenges, *Sensors*, 21:5874, 2021.

[2] H. Tian, J. He, Y. Ding, Medical data management on blockchain with privacy, *Journal of Medical Systems*, 43:1–6, 2019.

[3] O. Ali, M. Ally, Clutterbuck, Y. Dwivedi, The state of play of blockchain technology in the financial services sector: A systematic literature review, *International Journal of Information Management*, 54:102199, 2020.

[4] N. Kshetri, Cryptocurrencies: Transparency Versus Privacy [Cybertrust], *Computer*, 51(11):99–111, 2018. `https://ieeexplore.ieee.org/document/8625935/`.

[5] N. V. Saberhagen, Cryptonote v2.0, `https://www.bytecoin.org/old/whitepaper.pdf`, 2013.

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

[7] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, E. W. Felten, Mixcoin: Anonymity for bitcoin with accountable mixes, *Lecture Notes in Computer Science*, 8437:486–504, 2014.

[8] G. Maxwell, CoinJoin: Bitcoin privacy for the real world, `https://bitcointalk.org/index.php?topic=279249.0`, 2013.

[9] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, K. Wehrle, Coinparty: Secure multi-party mixing of bitcoins, In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015.

[10] S. Meiklejohn, R. Mercer, Möbius: Trustless Tumbling for Transaction Privacy, *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121.

[11] A. Pertsev, R. Semenov, R. Storm, Tornado Cash Privacy Solution Version 1.4, `https://berkeleydefi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf`, 2019.

[12] B. Bünz, S. Agrawal, M. Zamani, D. Boneh, Zether: Towards Privacy in a Smart Contract World, In J. Bonneau, N. Heninger (Eds.), *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pages 423–443, 2020.

[13] K. Kolachala, E. Simsek, M. Ababneh, R. Vishwanathan, SoK: Money Laundering in Cryptocurrencies, In *Proceedings of the 16th International Conference on Availability, Reliability and Security, ARES '21*, Association for Computing Machinery, New York, NY, USA, pages 1–10, 2021. https://doi.org/10.1145/3465481.3465774.

[14] D. M. Sat, G. O. Krylov, K. Evgenyevich, Bezverbnyi, A. B. Kasatkin, I. A. Kornev, Investigation of money laundering methods through cryptocurrency, *Journal of Theoretical and Applied Information Technology*, 2016.

[15] J. Lorenz, M. I. Silva, D. Aparício, J. A. T. Ascensão, P. Bizarro, Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity, In *Proceedings of the First ACM International Conference on AI in Finance, ICAIF '20*, Association for Computing Machinery, New York, NY, USA, 2021.

[16] D. Dupuis, K. Gleason, Money laundering with cryptocurrency: open doors and the regulatory dialectic, *Journal of Financial Crime*, 28:60–74, 2021.

[17] Q. Feng, D. He, S. Zeadally, M. K. Khan, N. Kumar, A survey on privacy protection in blockchain system, *Journal of Network and Computer Applications*, 126:45–58, 2019.

[18] S. Mabunda, Cryptocurrency: The New Face of Cyber Money Laundering, In *2018 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–6, 2018. https://ieeexplore.ieee.org/abstract/document/8465467.

[19] D. M. Meichler, Binance to Delist Privacy Coins in European Countries, https://decrypt.co/142973/binance-delist-monero-zcash-4-europeancountries, May 2023.

[20] U.S. Treasury, U.S. treasury sanctions notorious virtual currency mixer tornado cash, https://home.treasury.gov/news/press-releases/jy0916, 2022.

[21] Chainalysis, Understanding tornado cash, its sanctions implications, and key compliance questions, https://blog.chainalysis.com/reports/tornado-cash-sanctions-challenges/, 2022.

[22] Art. 5 GDPR – Principles relating to processing of personal data, https://gdpr-info.eu/art-5-gdpr/.

[23] L. Valenta, B. Rowan, Blindcoin: Blinded, accountable mixes for bitcoin, *Lecture Notes in Computer Science*, 8976:112–126, 2015.

[24] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, S. Goldberg, TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub, 2016.

[25] T. Ruffing, P. Moreno-Sanchez, A. Kate, CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin, In M. Kutyłowski, J. Vaidya (Eds.), *Computer Security - ESORICS 2014*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pages 345–364, 2014.

[26] M. Dotan, A. Lotem, M. Vald, Haze: A compliant privacy mixer, *Cryptology ePrint Archive*, Paper 2023/1152, 2023.

[27] A. Pfitzmann, M. Köhntopp, Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology, In H. Federrath (Ed.), *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability Berkeley, CA, USA, July 25–26, 2000 Proceedings*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pages 1–9, 2001.

[28] K. Sampigethaya, R. Poovendran, A Survey on Mix Networks and Their Secure Applications, *Proceedings of the IEEE*, 94(12):2142–2181, 2006.

[29] D. L. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms, *Commun. ACM*, 24(2):84–90, 1981.

[30] M. Chase, E. Ghosh, O. Poburinnaya, Secret-shared shuffle, In S. Moriai, H. Wang (Eds.), *Advances in Cryptology – ASIACRYPT 2020*, Springer International Publishing, Cham, pages 342–372, 2020.

[31] S. Eskandarian, D. Boneh, Clarion: Anonymous communication from multi-party shuffling protocols, *Cryptology ePrint Archive*, 2021.

[32] J. Groth, On the size of pairing-based non-interactive arguments, In M. Fischlin, J.-S. Coron (Eds.), *Advances in Cryptology – EUROCRYPT 2016*, Springer Berlin Heidelberg, Berlin, Heidelberg, pages 305–326, 2016.

[33] M. D. Ryan, Making Decryption Accountable, In F. Stajano, J. Anderson, B. Christianson, V. Matyáš (Eds.), *Security Protocols XXV*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pages 93–98, 2017.

[34] D. Nuñez, I. Agudo, J. Lopez, Escrowed decryption protocols for lawful interception of encrypted data, *IET Information Security*, 13(5):498–507, 2019.

[35] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

[36] M. Maller, S. Bowe, M. Kohlweiss, S. Meiklejohn, Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings, In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, Association for Computing Machinery, New York, NY, USA, pages 2111–2128, 2019.

[37] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, N. Ward, Marlin: Preprocessing zksnarks with universal and updatable srs, In A. Canteaut, Y. Ishai (Eds.), *Advances in Cryptology – EUROCRYPT 2020*, Springer International Publishing, Cham, pages 738–768, 2020.

[38] A. Gabizon, Z. J. Williamson, O. Ciobotaru, Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge, *Cryptology ePrint Archive*, Paper 2019/953, 2019.

[39] A. Chiesa, D. Ojha, N. Spooner, Fractal: Post-quantum and transparent recursive proofs from holography, In A. Canteaut, Y. Ishai (Eds.), *Advances in Cryptology – EUROCRYPT 2020*, Springer International Publishing, Cham, pages 769–793, 2020.

[40] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell, Bulletproofs: Short proofs for confidential transactions and more, In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334, 2018.

[41] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, J. Baylina, Circom: A circuit description language for building zero-knowledge applications, *IEEE Transactions on Dependable and Secure Computing*, pages 1–18, 2022.

[42] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, T. Tiessen, Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity, *Cryptology ePrint Archive*, Paper 2016/492, 2016.

[43] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, M. Schofnegger, Poseidon: A new hash function for zero-knowledge proof systems, *Cryptology ePrint Archive*, Paper 2019/458, 2019.

[44] R. Zou, X. Lyu, J. Ma, B. Zhang, D. Wu, BCMIX: A Blockchain-Based Dynamic Self-Reconfigurable Mixnet, *IEEE/ACM Transactions on Networking*, pages 2222–2235, 2023.

[45] A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, Proactive Secret Sharing Or: How to Cope With Perpetual Leakage, *Advances in Cryptology — CRYPTO' 95*, pages 339–352, 1995.

[46] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J.B. Nielsen, T. Rabin, S. Yakoubov, YOSO: You Only Speak Once, *Advances in Cryptology – CRYPTO 2021*, pages 64–93, 2021.