ShareLock: Mixing for Cryptocurrencies from Multiparty ECDSA

Omer Shlomovits[1] and István András Seres[2]

[1]KZen Research
[2]Eötvös Loránd University

May 26, 2019

Abstract

Many cryptocurrencies, such as Bitcoin and Ethereum, do not provide any financial privacy to their users. These systems cannot be used as a medium of exchange as long as they are transparent. Therefore the lack of privacy is the largest hurdle for cryptocurrency mass adoption next to scalability issues. Although many privacy-enhancing schemes had been already proposed in the literature, most of them did not get traction due to either their complexity or their adoption would rely on severe changes to the base protocol. To close this gap, in this work we propose ShareLock, a practical privacy-enhancing tool for cryptocurrencies which is deployable on today's cryptocurrency networks.

Cryptocurrency, Cryptographic protocol, Privacy, Anonymity, Threshold Cryptography, Bitcoin, Ethereum, Distributed Key Generation

1 Introduction

Bitcoin [23] and other cryptocurrencies are pseudonymous. Users control their funds with private keys, where their pseudonyms are their public keys. Sending coins from a public key is possible by signing transactions with the corresponding private key. If a linkage between a public key and a user's unique identifier (IP address, name, e-mail, social media account, website etc.) is revealed, financial privacy is permanently lost. Since transactions with referenced public keys are stored in an immutable, public ledger, called blockchain, even a single transaction could reveal to a recipient, say a merchant, her customers wealth and financial behaviour. Moreover, several techniques [19, 21, 1, 27] and for-profit companies (e.g. Chainalysis[1] and Coinalytics[2]) are known to provide de-anonymization services of cryptocurrency users. The likely leakage of financial secrets in a transparent, permissionless setting (e.g. amount of salaries in a company, payments to suppliers) is distressing. For this reason the lack of financial privacy is prohibitive in many potential application scenarios.

[1]https://www.chainalysis.com
[2]https://coinalytics.co

To resolve privacy issues of crpytocurrencies a plethora of privacy-enhancing techniques were proposed in the community. Centralized privacy solutions [31, 5, 15] defeat its purpose by depending on the goodwill of a trusted third party for privacy and for the availability of funds. On the other hand decentralized solutions [28, 3, 30, 19, 18, 6, 4] are usually computationally heavier then their centralized counterparts, however they satisfy strong notions of security (see Section 3).

Coin mixing is a common technique to achieve *k-anonymity* also known as *plausible deniability*. In this specific context k-anonymity entails that an on-chain identity (public key or a derived address) is indistinguishable from at least $k-1$ other on-chain identities. The concept of a coin mixer protocol is that k people mix their equal amount of coins and later they redistribute those coins among each other in a way that not even the participants themselves know which original coins belong to their final recipients. If a mixing protocol uses a trusted third party to redistribute coins among participants, then a corrupted third party may just send back tainted coins, or even worse it might not send back any coins at all. Sadly, as of writing almost exclusively centralized mixers are implemented and used in practice on a large scale [3], with the exception of the CoinJoin protocol (see in Section 1.1). Since these constructions do not provide mixer availability (i.e. mixer party can go offline and steal funds from participants), in the following we will solely focus on decentralized privacy-enhancing protocols, where the mixer party is replaced by a smart contract or by other crpytographic means.

1.1 Related work

In this section we shortly summarize the state of the art of privacy-enhancing techniques for cryptocurrencies. Hereby we do not intend to cover privacy coins, like Monero [25] and Zcash [29], since they already provide meaningful privacy guarantees by default.

CoinJoin [17] was the first decentralized Bitcoin mixer proposed by Gregory Maxwell. CoinJoin users create collaboratively a single Bitcoin transaction with n inputs and n outputs in a way that the mapping between inputs and outputs is concealed. However, the way it is implementeded and used in practice[4] trusts a central server for censorship resistance. A formal definitional framework of CoinJoin can be found in [20].

CoinShuffle, a distributed version of CoinJoin was proposed by Ruffing et al. [28]. Sadly it did not get traction among Bitcoin users due to its relatively complex and computationally heavy off-chain protocol.

Möbius [19] was the first coin mixer protocol directly designed for Turing-complete blockchains, like Ethereum. In the course of the Möbius protocol, users deposit funds to a mixer contract and later they can withdraw their funds by providing a linkable ring signature. The downside of Möbius is that the cost of verifying on-chain a ring signature grows linearly in the size of the anonymity set, thus disallowing large anonymity sets (cca. 50-100).

A similar proposal, called Miximus [3] applies zkSNARKs to prove ownership of coins in the mixed set. The difficulty for Miximus to become accepted is the

[3] https://bestmixer.io
[4] https://wasabiwallet.io

unavoidable necessity in carrying out a community-wide trusted setup for the proving and verifying key generation for the zkSNARK.

On the other hand MixEth [30] does not rely on a trusted setup, while both deposit and withdraw transactions' complexity is modest. Nonetheless since MixEth uses Neff verifiable shuffles [24] to mix participants' public keys, it requires several shuffling rounds to achieve k-anonymity. Such a time-intensive protocol might significantly reduce user experience.

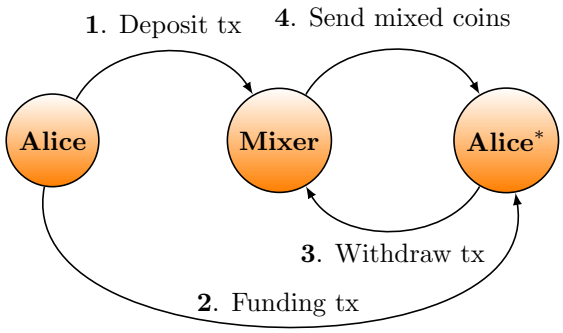

Figure 1: Privacy leakage in previous mixing protocols [19, 3, 30] for account-based cryptocurrencies. Note that Alice can only withdraw her coins from the mixer if her fresh address is funded in order to be able to issue the withdraw transaction. This can only be done via an additional funding transaction which links addresses of Alice and essentially nullifies the anonymity guarantees of the mixer.

The most crucial disadvantage, see Figure 1, of these proposed protocols is that they are not compatible with today's account-based cryptocurrencies, like Ethereum. Currently in Ethereum only the transaction sender can pay for the incurring transaction fees. However, when mixer users would like to withdraw their funds from the coin mixer contract, they should do that from a fresh, unused public key to achieve privacy. Yet fresh addresses do not hold ether, hence one cannot send transactions from such an address, because they cannot pay the gas fee from an address like that (see Section 2.5). Therefore users need to fund their fresh public keys, which essentially links also fresh addresses back to them, thus losing k-anonymity. We note that there is a workaround to solve this problem, however it is quite cumbersome and not well-established: users might advertise their transactions off-chain and pay third parties to send their transactions. Finding such parties is inconvenient and there is not a settled framework for that yet.

A recently proposed technique called Zether [6] circumvents these problems. Zether does not rely on active participation of other users, unlike coin mixers. Furthermore it provides not only anonymity but also confidentiality, i.e. also transaction amounts are hidden. Zether is deployable on today's Ethereum, however it is a prohibitively costly privacy solution, because one Zether transaction barely fits into an Ethereum block: one Zether transaction burns 7,188,000 gas.

Our contribution: in this work we propose a threshold ECDSA-based privacy-enhancing solution for cryptocurrencies. It does not rely on trusted third parties, nor trusted setups and only uses minimal blockchain resources. To show the practicality of our approach we present ShareLock, the first lightweight, already deployable, Ethereum-friendly coin mixer, which allows anyone to mix ether or any ERC-20 tokens. We hope that placing the source code in the hands of the community, will serve as the first step on the long, ragged road towards financial privacy on cryptocurrency networks.

Organization. The rest of the paper is organized as follows. In Section 2 we provide some background on (multiparty) ECDSA, Bitcoin and Ethereum. We define the desired security notions in Section 3, while we describe our protocol in Section 4 enclosed with formal proofs of security. We present our implementation and measure performance in Section 5.

2 Background

2.1 Notations

Let $\mathbb{G}$ be an Elliptic Curve group of order q with generator G where the CDH assumption is assumed to be hard. We will use addition as the group operation, upper-case characters for group elements, and lower-case characters for scalars in $\mathbb{Z}_q$. This is consistent with elliptic curve notation.

2.2 Blockchain Model

By our ideal functionality definition stated next in Section 3.1, a mixer must interact with a blockchain. For our security proof to make sense we thus must use a blockchain model running a blockchain protocol. In a blockchain protocol, the goal of all parties is to maintain a global ordered set of records that are referred to as blocks. New blocks can only be added using a special mining procedure that simulates a puzzle-solving race between participants and can be run by any party (called miner) executing the blockchain protocol. Presently, two broad categories of puzzles are used: Proof-of-Work (PoW) and Proof-of-Stake (PoS).

An abstract blockchain protocol consists of three polynomial-time algorithms (UpdateState, GetRecords, Broadcast) [14] with UpdateState used to maintain a local state. GetRecords outputs from a state the longest ordered sequence of valid blocks contained in the state, where each block in the chain itself contains an unordered sequence of records. Broadcast is used to propagate a message to all nodes running the protocol. As in [26], the blockchain protocol is also parameterized by a *validity predicate* that captures semantics of any particular blockchain application.

Badertscher et al. [2] introduced an elegant model for blockchain as a *global* ledger functionality $\mathcal{G}_{ledger}$ where they even show how an abstracted Bitcoin protocol can be implemented. Another type of model is using a *local* ledger functionality [14].

In our interaction with the blockchain we require only standard read/write interaction. We aim for the protocol to work with many different types of

blockchain implementations and therefore any simple and abstracted model will work.

2.3 The ECDSA Signing Algorithm

We focus first on applying ShareLock to Ethereum blockchain, where the signing algorithm used is ECDSA. Applying ShareLock to other blockchains and signing protocols is possible, see Section 4.3. The protocol works as follows: The private key is a random value $x \leftarrow \mathbb{Z}_q$ and the public key is $Q = x \cdot G$. ECDSA signing on a message $m \in \{0,1\}^*$ is defined as follows:

1. Compute $H_q(m)$: the $|q|$ leftmost bits of SHA256(m), where $|q|$ is the bitlength of q.
2. Choose a random $k \in \mathbb{Z}_q^*$
3. Compute $R \leftarrow k \cdot G$. Let $R = (R_x, R_y)$.
4. Compute $r = R_x mod\ q$ and $s \leftarrow k^{-1} \cdot (H_q(m) + r \cdot x) mod\ q$
5. Output (r, s)

The signing verification algorithm $\mathsf{verify}(Q, m, (r, s))$ takes as input the public key, the message and the signature and returns true if the signature is valid. ECDSA signature has a known malleability: For every valid signature $\sigma = (r, s)$ the signature $\sigma' = (r, -s)$ is also a valid signature. In our protocol we will require unique signatures. We can achieve it by either mandating that $min\{s, q - s\}$ is always output and verify rejects if $s > \frac{q}{2}$ or we can output an extra bit v to determine which is the correct signature.

Security: ECDSA holds the property of being existentially unforgeable against chosen message attacks (EU-CMA) as introduced in [13]. Simply put, EU-CMA means that no probabilistic polynomial time adversary can produce a signature on a message he did not see before, even given a signing oracle that can produce signatures on adaptively chosen messages, except with negligible probability in the security parameter.

2.4 Multi-Party ECDSA

Multi-Party ECDSA is a threshold signature scheme. A (t, n)- threshold signature scheme enables distributing the signing among a group of n parties, $P_1, \ldots, P_n$ such that any group of at least $t + 1$ of these parties can jointly generate a signature, whereas groups of size t or fewer cannot. In our protocol we will use threshold signature with $t = n - 1$ such that $n - 1$ malicious parties will not learn the signing key or be able to sign. In our protocol we require one multi-party ECDSA signature which we define here using ideal functionality $\mathcal{F}_{\text{ECDSA}}$. The functionality is defined with two functions: key generation and signing, both require the cooperation of n parties. The verification of a multi-party ECDSA remains identical to that of ECDSA defined in Section 2.3.

The key generation is modeling a distributed key generation protocol (DKG) where the n parties jointly generate a pair of public/private keys. The keypair

produced by the DKG is generated with the same distribution as the single party ECDSA.

The signing is modeling a distributed signing protocol. If all n parties that run the previous DKG are honest the output of the protocol will be a valid signature on a public message m.

To reason about security we use the ideal functionality below, taken from [16]. The key generation is called once, and then any arbitrary number of signing operations can be carried out with the generated key. The functionality is defined in Figure 2.1.

FUNCTIONALITY 2.1 (The ECDSA Functionality $\mathcal{F}_{\text{ECDSA}}$)

Functionality $\mathcal{F}_{\text{ECDSA}}$ works with parties $P_1, \ldots, P_n$, as follows:

- Upon receiving $\mathsf{KeyGen}(\mathbb{G}, G, q)$ from all parties $P_1, \ldots, P_n$:
  1. Generate a key pair (ak, ask) by choosing a random $ask \leftarrow \mathbb{Z}_q^*$ and computing $ak = ask \cdot G$. Then, store $(\mathbb{G}, G, ak, ask)$.
  2. Send ak to all $P_1, \ldots, P_n$.
  3. Ignore future calls to KeyGen.
- Upon receiving $\mathsf{Sign}(sid, m)$ from all $P_1, \ldots, P_n$, if KeyGen was already called and sid has not been previously used, compute ECDSA signature on m in the following way:
  1. Choose a random $k \in \mathbb{Z}_q^*$ and set $R \leftarrow k \cdot G$ and $r = R_x mod\ q$.
  2. Compute: $s = k^{-1}(H_q(m) + r \cdot ask)$.

  Send (r, s) to all $P_1, \ldots, P_n$.

2.5 Bitcoin and Ethereum: UTXO and account-based transaction models

Cryptocurrencies either have an unspent transaction output (UTXO)-based or an account-based transaction model, where Bitcoin and Ethereum are the two leading examples of these classes respectively. Therefore in the following we use these two cryptocurrencies as our role models.

In a UTXO model coins are represented as references to previously unspent transaction outputs. All of the unspent transactions are kept in each fully-synchronized node, hence the name of this model: UTXO. A Bitcoin transaction contains, possibly several, inputs, references to UTXOs, and possibly several outputs, freshly created UTXOs. In each transaction all the inputs should reference valid UTXOs and should be signed by their legitimate owner and additionally the sum of the inputs should be greater or equal than the sum of the outputs. A user's balance is the sum of all the UTXOs which belong to her. The flow of ownership throughout UTXOs can be modeled as a path in a directed acyclic graph (DAG), where nodes are the UTXOs and edges are the transactions redeeming and creating new UTXOs.

In some cases a UTXO transaction model might be tedious. When a user purchases a good or service whose price is greater than the value of her individual UTXOs, she needs to merge multiple UTXOs as the inputs in the buying transaction in order to perform the purchase. Since in the Bitcoin community address/public key reuse is discouraged, users have some low level privacy.

However due to the public nature of the transaction ledger, this is still quite minimal due to several deanonymization techniques and transaction graph analysis tools [27, 10].

On the other hand, from a privacy point of view the situation is even worse for account-based cryptocurrencies, such as Ethereum. In an account-based model the system keeps track of all the account's balances in a global state. The balance of an account is checked to make sure it is larger than or equal to the spending transaction amount. Therefore in these systems users are incentivised not to spread their wealth between several accounts, rather hoard it in a single, monolithic account. This behaviour makes deanonymization even easier than that of for UTXO-based currencies.

In Ethereum there are two types of accounts: externally owned accounts (EOA) and contract accounts. EOAs are controlled by an asymmetric cryptographic keypair, while contracts are controlled by their code. The native currency of Ethereum is called ether and denominated in wei: $1ether = 10^{18}wei$. The Ethereum blockchain keeps track of the state of every account. The global state can be changed by initiating a transaction from an EOA. Each transaction contains the following data: sender address, destination address, a signature σ, the transferred amount in wei, a nonce to deter replay attacks, an optional data field which serves as the input arguments to a contract, a *GasLimit* and a *GasPrice* value. The nonce is incremented after every sent transaction. The signature σ signs the transaction data and the nonce, while is verified against the sender's public key.

A transaction either transfers wei from one account to another or triggers the execution of a contract's code. Contracts can send messages to other contracts, mimicking function calls. Every transaction and code execution is replicated on all nodes in the network. Every executed operation has a specified cost expressed in terms of *gas* units. For instance, storing 256 bits of data on the blockchain costs 20,000 units of gas while changing it costs 5,000 and reading it costs 200. The sender pays for all contract operations that the transaction calls. Transaction senders can define *GasLimit*, the overall gas consumption she is allowing the transaction to consume and *GasPrice*, namely how many wei she is willing to pay for one unit of gas. Therefore the overall cost of a transaction measured in wei is the amount of *gasPrice* multiplied by the amount of gas burnt by the transaction. If a transaction does not use all the gas provided to it, then excess fees are refunded to the account that issued the transaction. If the gas limit falls short of the gas needed to process the transaction, the miner will collect the fee but not change the blockchain's state. Transaction fees or gas costs are collected by the miners. Naturally, miners are incentivised to pick transactions with higher *GasPrice* first.

The size of Ethereum blocks is measured by the sum of gas consumed by all transactions in that block. This value is capped and is called the *block gas limit*. Currently the block gas limit is around 8 million gas[5]. For that reason every Ethereum user is incentivised in achieving their desired goals by using as little gas as possible.

Ethereum contracts can be read and executed by a Turing-complete execution environment, called the Ethereum virtual machine (EVM). The most popular language to develop contract code is Solidity. Contracts' code writ-

[5]https://etherscan.io/chart/gaslimit

ten in Solidity are compiled to EVM bytecode later executed by the EVM and stored on the Ethereum blockchain. The EVM has access to a global persistent storage system and each contract account has separate storage available to it.

For more details the reader is referred to the Bitcoin [23] and the Ethereum [32] whitepapers.

3 Security Definition

This section defines and motivates the ideal mixing functionality $\mathcal{F}_{\text{MIX}}$. After defining $\mathcal{F}_{\text{MIX}}$ we will demonstrate how it intuitively achieves the security goals of a mixer. In section 4 we provide a secure implementation for $\mathcal{F}_{\text{MIX}}$ that can work on Ethereum [32] and afterwards an implementations that can work on Bitcoin [23].

3.1 The $\mathcal{F}_{\text{MIX}}$ Ideal Functionality

We define $\mathcal{F}_{\text{MIX}}$ using three functions: setup, clean and abort. Setup is called once. It is assumed that a blockchain payment system is present in the background, modeled as described in Section 2.2 allowing transfer of coins between addresses (deterministic representations of public keys). We treat the blockchain calls in our model in an intuitive way without being explicit about the syntax, considering the blockchain a trusted setup. The functionality is defined in Functionality 3.1.

The basic idea behind $\mathcal{F}_{\text{MIX}}$ is simple: parties $P_1, \ldots, P_n$ play both senders and recipients. Each party deposits a predefined amount amt using blockchain transactions to an address associated with $\mathcal{F}_{\text{MIX}}$. The depositing is not done in the functionality, but we assume it is done outside of it. Once the deposit amount passed a fixed threshold, $\mathcal{F}_{\text{MIX}}$ will send amount amt to n freshly generated random addresses. $\mathcal{F}_{\text{MIX}}$ will sample a fully random permutation, apply it on the output addresses and will output to each party P_i a private key for the permuted new address to allow the party to spend from this address on the blockchain.

FUNCTIONALITY 3.1 (The Mixing Functionality $\mathcal{F}_{\text{MIX}}$)

Functionality $\mathcal{F}_{\text{MIX}}$ is parameterized with linear function $f()$ and works with parties $P = \{P_1, \ldots, P_n\}$ as follows:

- Upon receiving $\mathsf{Setup}(amt, t, sid)$ from some party $\mathcal{U} \in P$, check that sid has not been previously used, and t has not passed, if not do:
  1. Save amt, t
  2. Sample random permutation π on $1, \ldots, n$
  3. Generate a keypair and corresponding address B. Send B to all parties.
  4. Generate n keypairs $kp_i|_{i=1}^{n}$ where $kp_i = \{ask_i, ak_i\}$ and corresponding addresses: $out_1, \ldots, out_n$
  5. Send $kp_{\pi(i)}$ to P_i
  6. Send $m = ak_1|\ldots|ak_n$ to adversary $\mathcal{S}$
- Upon receiving $\mathsf{Clean}(sid)$ from some party $\mathcal{U} \in P$, If B.$balance \geq n \cdot amt$ and t has not passed:
  1. Send amount amt from B to each address $out_{\pi(1)}, \ldots, out_{\pi(n)}$
  2. Compute $ask = f(ask_1, \ldots, ask_n)$, $m = ak_1|\ldots|ak_n$ and send σ to adversary $\mathcal{S}$, where σ is ECDSA signature on message m with private key ask
- Upon receiving $\mathsf{abort}(sid)$ from some party $\mathcal{U} \in P$, if $\mathsf{Clean}(sid)$ was never called before, check if a transaction exists transferring at least amt from $\mathcal{U}$ to B. If true: send amount amt to $\mathcal{U}$

For reasons of simplification and to focus on the security goals of mixing (see Section 3.2) we let all reading from or writing to the blockchain always succeed, return results immediately and without additional costs. To make the security more complete and representing real life, once a specific blockchain is specified, it is recommended to define the Validation predicate [26], as well as specific models for transaction costs, accounts or UTXOs. We leave this improved modeling for future work.

We will use in our proof a weaker version of the functionality $\mathcal{F}_{\text{MIX}}$, which we will mark $\mathcal{F}'_{\text{MIX}}$ that allows the adversary to bias the permutation π by choosing the indices of the malicious parties. This also gives a better account of the achieved Anonymity (described in 3.2) in the presence of static malicious parties, making the anonymity set dependent only on the honest parties.

In our design of $\mathcal{F}_{\text{MIX}}$ we made two decisions that while not affecting the security goals (3.2), do have consequences on how the mixer operates. These choices were driven by our secure implementation. First, $\mathcal{F}_{\text{MIX}}$ is generating *random* fresh addresses for the recipients, while one could have expect that each sender s_i will be able to choose the address she wants to mix into r_i. This is motivated by use case such as paying to a merchant with a fresh address. In our model this is still possible: The sender s_i will first mix and receive clean coins, than she will send the clean coins to the merchant. Second, $\mathcal{F}_{\text{MIX}}$ is sending funds to all outputs at the same time unlike all previous mixer proposals, where participants need to withdraw their mixed coins by themselves. Since the deposit/withdraw pattern is leaking privacy, see Section 1.1, we deliberately

decided to let the mixer sending out mixed coins in order to retain k-anonymity.

Finally, we add some leakage to $\mathcal{S}$ in the ideal functionality to make the security proof go through. $\mathcal{F}_{\text{MIX}}$ is sending all public keys to Adversary $\mathcal{S}$ as part of setup and a digital signature σ signed by a joint function f of all private keys. Output public keys are not a secret in a mixer but only their mapping to input public keys: we allow the adversary to learn them. Leaking all secret keys $ask_1, \ldots, ask_n$ will reveal the mapping and therefore we cannot do it, but a signature will not reveal any private key, otherwise the signature scheme is not EU-CMA. Therefore, we allow the adversary to learn a multi-party signature.

3.2 Security Goals

Previous works [19, 30] have used game based definitions to define three security goals of mixing: Anonymity, Availability and Theft prevention. In this section we informally define these notions and also argue that the ideal functionality $\mathcal{F}_{\text{MIX}}$ achieves these goals:

Anonymity We would like to ensure that sender and recipient addresses are anonymous; i.e., that for a given sender, it is not possible to distinguish between their recipient and any other recipient using the mixer. We consider this goal with respect to three types of attackers: (a) an eavesdropper who is acting as neither the sender nor the recipient; (b) a malicious sender (or set of senders); (c) a malicious recipient (or set of recipients);

1. **Anonymity against outsiders**: an outsider sees n incoming transactions and n outgoing transactions . Generating fresh addresses for recipients is done off-chain which "breaks" the links between transactions on the blockchain.

2. **Anonymity against senders**: Senders are sending money into a single address and then a set of n recipients are getting the pool of funds redistributed to fresh new addresses. $\mathcal{F}_{\text{MIX}}$ generates random fresh keys to every party independent of senders addresses and outputs a random permutation of the keys to known parties that interacts with it. In that sense we get n anonymity set, assuming $n > 2$.

3. **Anonymity against recipients**: a recipient will know only his local public key. For malicious recipient the anonymity set will be $n - t$ for t malicious recipients.

Theft prevention Theft prevention means that the scheme does not allow coins to be either withdrawn twice or withdrawn by anyone other than the intended recipients. The ideal functionality is sending exactly the same amount to all parties in an equal manner. Each party receives a single private key to an address with the same amount she deposited.

Availability Availability means that no one can prevent the sender from using the mixer; and once the money was sent to the mixer, no one can prevent the honest recipient from withdrawing it. The only problematic state in $\mathcal{F}_{\text{MIX}}$ is if

some party is gone while others already sent funds on the blockchain. For this case, the abort function can be used: $\mathcal{F}_{\text{MIX}}$ will send amount *amt* back to the sender. Assuming asynchronous network it is not clear if a party transaction was delayed or never sent. Parties that already finished depositing can wait for the other parties to join, send abort to reclaim the input funds or send enough coins to B to trigger a Clean.

3.3 Security Model

We prove security according to the standard simulation paradigm with the real/ideal model [7, 12], in the presence of *malicious adversaries* and *static corruptions*. We assume all parties can access the blockchain, post transactions with immediate finality and zero cost. We conjecture that global functionality $\mathcal{G}_{ledger}$ [2] can be used for modelling interaction with the blockchain without changing the security of the model. We leave the incorporation of $\mathcal{G}_{ledger}$ to future work.
We prove the security of our protocol in hybrid model with ideal functionality $\mathcal{F}_{\text{E-ECDSA}}$, defined next in Section 4.1. The soundness of this model is justified in [7] (for stand-alone security) and in [8] (for security under composition). Specifically, as long as subprotocol that securely compute the functionality is used (under the definition of [7] or [8] respectively). It is guaranteed that the output of the honest and corrupted parties when using real subprotocol is indistinguishable to when calling a trusted party that computes the ideal functionality.

We remark that if $\mathcal{F}_{\text{E-ECDSA}}$ is UC-secure [8] and global functionality $\mathcal{G}_{ledger}$ is used, than our protocol can also be proven secure in this framework.

Communication model Our protocol in the hybrid model can work with asynchronous network but for existing implementations of the hybrid functionality [16, 11] a synchronous network is assumed with point to point and broadcast channels. In addition we require the channels to be anonymous.

4 Mixer Protocol

4.1 Extended $\mathcal{F}_{\text{ECDSA}}$

In our mixing protocol we will work in the hybrid model with $\mathcal{F}_{\text{E-ECDSA}}$. This is a natural extended version of $\mathcal{F}_{\text{ECDSA}}$:

FUNCTIONALITY 4.1 (Extended ECDSA Functionality $\mathcal{F}_{\text{E-ECDSA}}$)

Functionality $\mathcal{F}_{\text{E-ECDSA}}$ works with parties $P_1, \ldots, P_n$ and some linear function f as follows:

- Upon receiving $\mathsf{KeyGen}(\mathbb{G}, G, q)$ from all parties $P_1, \ldots, P_n$:
  1. For $i \in \{1 \ldots n\}$ generate a key pair (ak_i, ask_i) by choosing a random $ask_i \leftarrow \mathbb{Z}_q^*$ and computing $ak_i = ask_i \cdot G$.
  2. For $i \in \{1 \ldots n\}$ send to P_i: $(ask_i, ak_1, \ldots, ak_n, f)$
  3. Store $(\mathbb{G}, G, ask = f(ask_1, \ldots, ask_n), ak = ask \cdot G$.
  4. Ignore future calls to KeyGen.
- Upon receiving $\mathsf{Sign}(sid, m)$ from all $P_1, \ldots, P_n$, if KeyGen was already called and sid has not been previously used, compute a signature on m in the following way:
  1. Choose a random $k \in \mathbb{Z}_q^*$ and set $R = k \cdot G$ and $r = R_x mod\ q$.
  2. Compute: $s = k^{-1}(H(m) + r \cdot ask)$.

  Send (r, s) to all $P_1, \ldots, P_n$.

The change from $\mathcal{F}_{\text{ECDSA}}$ is that while in the original KeyGen each party got the shared public key ak, in $\mathcal{F}_{\text{E-ECDSA}}$ each party i gets a secret share ask_i and a list of all local public keys $ak_1, \ldots, ak_n$. The modified functionality is using some linear function f for its secret sharing: The secret shares might be additive, multiplicative or any type of linear relation. In practice, all multi-party ECDSA protocols existing today that securely implement $\mathcal{F}_{\text{ECDSA}}$ [16, 11, 9] are using an additive secret sharing scheme $f = \Sigma$, and providing the secret share and local public keys for the parties as specified in $\mathcal{F}_{\text{E-ECDSA}}$. For protocols implementing $\mathcal{F}_{\text{ECDSA}}$ but not $\mathcal{F}_{\text{E-ECDSA}}$, which could theoretically be possible without effecting the signature scheme security, we claim that the changes needed for $\mathcal{F}_{\text{E-ECDSA}}$ can be added for the same security model as an addition for the keygen protocol. Since, as mentioned, there are protocols that already implement $\mathcal{F}_{\text{E-ECDSA}}$ with proven security, we will use one of them for implementing the mixing protocol.

4.2 ShareLock: a Mixer Protocol

In this section we present our mixing protocol and prove that it securely implements $\mathcal{F}_{\text{MIX}}$. The protocol is written for account-based cryptocurrencies that supports a smart contract scripting language which is Turing complete, i.e. Ethereum.

A few remarks on the construction:

- We assume Anonymous-channels (AC) [22] for the communication between parties $P_1, , P_n$. Otherwise the parties participating in the protocol can trivially connect P_i and $\mathcal{P}_j$ and break anonymity. We discuss in Section 5, how under certain limitations we can avoid implementing AC.
- In choosing *off-chain* index j we assumed no two parties have chosen the same index, This can be done in practice, see 5.2 for more details.

- In step 4 we ask all parties to send the signature to the contract. In practice it can be done by anyone who is incentivised to pay for this transaction. In our implementation it is done by an *Activator* party, not necessarily part of the n parties, see Section 5. In the protocol we do not assume the existence of such party, we use $n-1$ redundancy of this message to guarantee its delivery. In practice $\mathcal{C}$ will reject all messages after the first valid signature.
- The smart contract role can be played by any trustless third party.
- Since there is no dependency between step 2 (funding the contract) and step 3 (generating a signature) they in fact can happen in parallel. Step 2 can happen outside of the protocol, as done in the $\mathcal{F}_{\text{MIX}}$, we wrote it down as part of the protocol for completness.

PROTOCOL 4.2 (ShareLock: Securely Computing $\mathcal{F}_{\text{MIX}}$)

Auxiliary input: Each party has the description $\mathbb{G}, G, q$ of a group, and the number of parties n.
Initialize: Setup a smart contract $\mathcal{C}$ with address B, parameterized with amount amt and timeout t.

- Each party P_i works as follows:
  1. P_i chooses random index $j \in \{1, .., n\}$ and plays role as $\mathcal{P}_j$ in $\mathcal{F}_{\text{E-ECDSA}}$ key generation: $\mathcal{P}_j$ sends $(\mathsf{KeyGen}, \mathbb{G}, G, q)$ to $\mathcal{F}_{\text{E-ECDSA}}$ and receives back $(ask_j, ak_1, \ldots, ak_n, f)$. P_i computes $ak = f(ak_1, \ldots, ak_n)$.
  2. P_i sends amount amt to address B with data ak. $\mathcal{C}$ registers $\{in_i, P_i\}$ where in_i is P_i sending address.
  3. P_i plays $\mathcal{P}_j$ in $\mathcal{F}_{\text{E-ECDSA}}$ signing: $\mathcal{P}_j$ sends (Sign, m, sid) to $\mathcal{F}_{\text{E-ECDSA}}$ where $m = ak_1 | \ldots | ak_n$ and receives back $\sigma = (r, s)$.
  4. P_i sends (ak, m, σ) to $\mathcal{C}$
- $\mathcal{C}$ runs $\mathsf{Verify}(ak, m, \sigma)$, if true, timeout t has not passed and B.$balance \geq n \cdot amt$: $\mathcal{C}$ parse m and sends amt to each address ak_i in m
- If t has passed $\mathcal{C}$ sends back to each registered P_i its funds back to its sending address in_i and self-destructs.

Theorem 4.3 *Protocol 4.2 securely implements $\mathcal{F}_{\text{MIX}}$ in the $\mathcal{F}_{\text{E-ECDSA}}$-hybrid model in the presence of $t \leq n-2$ malicious parties.*

Proof: let $\mathcal{A}$ be an adversary. We construct a simulator $\mathcal{S}$ who invokes $\mathcal{A}$ internally and simulates an execution of the real protocol, while interacting with $\mathcal{F}_{\text{MIX}}$ in the ideal model.
We restrict the adversary to have at least two honest parties for every mix. This can be intuitively motivated since there is no use for mixing with zero or one honest parties: if all parties are malicious there is no anonymity trivially and if only one party is honest, $\mathcal{A}$ can break its anonymity by elimination.

One can see that our protocol above will not work for $n-1$ malicious parties: in step 1 the adversary can always wait until the honest party joined the protocol and then afterwards join itself, thus always getting index n and skewing the

random distribution. In general, since our protocol is not using cryptographically secure coin flip protocol, the distribution will never be random. This is acceptable in the weaker form of $\mathcal{F}_{\text{MIX}}$, namely $\mathcal{F}'_{\text{MIX}}$, where all malicious parties can bias the permutation and choose their index. Adding a secure coin-flip protocol between the parties may be used to generate a public randomness and thus work with the stronger $\mathcal{F}_{\text{MIX}}$. We decided to work instead with $\mathcal{F}'_{\text{MIX}}$ since the security goals (3.2) are unaffected and we save added complexity from ShareLock protocol 4.2.

The intuition behind the proof is simple: because n out of n signature is needed to activate the mixer, the simulator can generate such a signature without knowing the private keys of the honest parties by simply getting the signature from the ideal functionality.

Most of the complexity of the proof is hidden inside the complexity of proving a secure protocol for $\mathcal{F}_{\text{E-ECDSA}}$ (see [16] for such a proof).
Let $I \subseteq [n]$ be the set of corrupted parties and $J = [n]nI$ denote the set of honest parties. We denote corrupted parties by P_I and honest parties by P_J. We construct a simulator $\mathcal{S}$ as follows:

Simulator $\mathcal{S}$:

1. $\mathcal{S}$ sends $\mathsf{Setup}(amt, t, sid)$ to $\mathcal{F}_{\text{MIX}}$ and receives address B, keypairs $\{ask_i, ak_i\}$ for every $i \in I$ and message: $m = ak_1| \dots |ak_n$
2. $\mathcal{S}$ simulates smart contract $\mathcal{C}$ with address B and timeout t. $\mathcal{S}$ publishes $\mathcal{C}$ to P_I.
3. $\mathcal{S}$ simulates $\mathcal{F}_{\text{E-ECDSA}}$ key generation:
   (a) Parses m and defines a public key $ak = f(ak_1, , ak_n)$
   (b) Each party $\mathcal{U}_i \in P_I$ receives $ask_i, ak_1, \dots, ak_n, f$.
4. $\mathcal{S}$ registers any $\mathcal{U}_i \in P_I$ that sends at least amt coins to B.
5. $\mathcal{S}$ monitors the blockchain and waits until one of two options occur:
   (a) B.$balance \geq n \cdot amt$ and $time < t$:
   $\mathcal{S}$ simulates $\mathcal{F}_{\text{E-ECDSA}}$ sign: $\mathcal{S}$ receives (Sign, m, sid) from P_I and checks that all messages are equal to $ak_1| \dots |ak_n$. If for some party $\mathcal{U} \in P_I$ the message is different, $\mathcal{S}$ sends $\mathsf{abort}(sid)$ to $\mathcal{F}_{\text{MIX}}$ on behalf of $\mathcal{U}$ and outputs whatever $\mathcal{A}$ outputs. Otherwise, $\mathcal{S}$ will send $\mathsf{Clean}(sid)$ to $\mathcal{F}_{\text{MIX}}$ and receive back σ which it will forward to P_I.
   (b) $time > t$:
   $\mathcal{S}$ sends $\mathsf{abort}(sid)$ to $\mathcal{F}_{\text{MIX}}$ on behalf of all registered $\mathcal{U}_i \in P_I$ that sent funds to B.

From the standpoint of $\mathcal{A}$ the simulated and real executions are identical: Step 3 in the simulation is parallel to Step 1 in protocol 4.2 simulating $\mathcal{F}_{\text{E-ECDSA}}$ key generation, Step 5(a) in the simulation is parallel to step 3 in protocol 4.2 simulating $\mathcal{F}_{\text{E-ECDSA}}$ signing. Interaction with the blockchain is equivalent in the simulation step 4 and protocol 4.2 step 2. Step 4 in protocol 4.2 is unnecessary in the simulation that can get this information from the signature internally, therefore $\mathcal{S}$ just ignores messages of type (ak, m, σ). $\mathcal{S}$ simulates

timeout t using the blockchain to measure timings; no additional clock is needed. In both real and ideal runs abort is triggered only by expiration of timeout t, resulting in amt coins return to all sending parties that sent funds already to the contract. Therefore the simulation is perfect. ■

Simulating Blockchain $\mathcal{S}$ simulates an entire blockchain to $\mathcal{A}$. Using a global ledger functionality will not work for this protocol because it requires $\mathcal{S}$ to own enough actual coins. We leave how to make the protocol secure with global $\mathcal{G}_{ledger}$ for future work.

The need for $\geq n-1$ signers Our security assumes at least two honest parties, using at least $n-1$ signers ensure that at least one party will be honest. Without honest party, Adversary $\mathcal{A}$ could have signed a wrong message m'. Signing the wrong message is impossible in the simulator and thus an adversary can find difference between ideal and real runs and break security. In more details: with $t \leq n-1$ malicious parties we are guaranteed that at least one honest party P_j will participate in the real protocol. The honest party will send $\mathcal{F}_{\text{E-ECDSA}}$ the correct message at step 3 of the real protocol which makes sure it is impossible for all parties to request a signature for incorrect message m' since $\mathcal{F}_{\text{E-ECDSA}}$ will fail not receiving the same message from all parties. For $\mathcal{S}$ that simulates $\mathcal{F}_{\text{E-ECDSA}}$ it is not a problem because $\mathcal{S}$ receives the correct message and signature from $\mathcal{F}_{\text{MIX}}$.

4.3 Adapting to Bitcoin

One of the benefits of our protocol is that it has minimal requirements from the blockchain. Concretely we present protocol 4.4 which demonstrates how the protocol can be adapted to Bitcoin blockchain with only minor modifications.

PROTOCOL 4.4 (Bitcoin ShareLock)

Auxiliary input: Each party has the description $\mathbb{G}, G, q$ of a group, and the number of parties n.

- The parties decide on amount amt and timelock t
- Each party P_i works as follows:
  1. P_i chooses random index $j \in \{1, .., n\}$ and plays role as $\mathcal{P}_j$ in $\mathcal{F}_{\text{E-ECDSA}}$ key generation: $\mathcal{P}_j$ sends $(\mathsf{KeyGen}, \mathbb{G}, G, q)$ to $\mathcal{F}_{\text{E-ECDSA}}$ and receives back $(ask_j, ak_1, \ldots, ak_n, f)$. P_i computes $ak = f(ak_1, \ldots, ak_n)$.
  2. P_i constructs and publishes a transaction with inputs value of at least amt, and that can be spent by either:
     (a) pubkey ak
     (b) P_i's pubkey after timelock t expires.
  3. P_i plays $\mathcal{P}_j$ in $\mathcal{F}_{\text{E-ECDSA}}$ signing: $\mathcal{P}_j$ sends (Sign, m, sid) to $\mathcal{F}_{\text{E-ECDSA}}$ where m is a *mix transaction* with all the UTXOs from step (2) as inputs and n outputs: sending amt coins to each address $ak_1, \ldots, ak_n$.
  4. P_i publishes the *mix transaction* to the blockchain.

The protocol do not specify how the parties should decide on amt, t. In practice it can be done by a third party.

Theorem 4.5 *Protocol 4.4 securely implements $\mathcal{F}_{\text{MIX}}$ in the $\mathcal{F}_{\text{M-ECDSA}}$-hybrid model in the presence of $t \leq n-2$ malicious parties.*

Proof sketch: Use same Simulator $\mathcal{S}$ from Section 4.2. ■

The ShareLock protocol for Bitcoin converges nicely to a designated single mix transaction with multiple inputs and multiple outputs. Abort is enabled by allowing the sending party to also spend its input transaction. Timelock is used but we remark that as opposed to a protocol that is based on smart contract here the timelock is a feature and the protocol will work without it. We want to highlight the difference between Sharelock and multisig transaction which will help clarify why the former achieves the mixing security goals and why the latter was never used for that purpose.

A multisig is a Bitcoin Script[6] defining a specific access structure to sign a transaction in a non-interactive way (one message per party). This is a software emulation of cryptographic multi-signature.

ShareLock on the other hand, is first using a DKG protocol to reach a consensus among n receiving parties about what should be the set of outputs of a mixing transaction. Then it is using a second protocol to enforce this agreement between n senders. Multisig is parallel to the second protocol. Adding the off-chain DKG, with separation of the logical indices in the DKG and the sending parties IDs in the second protocol gives us the result we need.

Adapting to other crypto-currencies: We exemplified in this section a protocol for account-based blockchains (4.2) and UTXO-based blockchains (4.4). Together they cover most existing blockchain architectures. In this work we focused on $\mathcal{F}_{\text{ECDSA}}$ and extended version of it $\mathcal{F}_{\text{E-ECDSA}}$. Other blockchains might use a different signing scheme than ECDSA. For other than ECDSA schemes, ShareLock will need a multi-party protocol implementation that can be extended as we did here and proven secure. Our protocol is abstracting the elliptic curve used by the blockchain which makes it easily replaceable for blockchains that rely on different types of elliptic curves.

5 Implementation

In order to show the practicality of our protocol, we implemented it and hereby release our developed tool, called ShareLock and its source code to the cryptocurrency community[7].

5.1 Performance analysis

We developed ShareLock in Rust and Solidity languages. We implemented the threshold ECDSA protocol by Gennaro et al. [11] for the off-chain distributed key generation and threshold ECDSA in Rust, while our Ethereum mixer contract is developed in Solidity language.

[6] https://en.bitcoin.it/wiki/Script
[7] https://github.com/KZen-networks/ShareLock

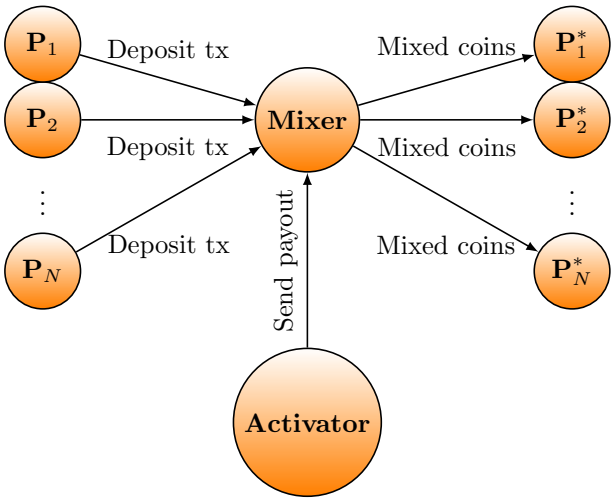

Figure 2: This work's approach to achieve *plausible deniability*, also known as *k-anonymity* for account-based cryptocurrencies. Contracts (EOAs and smart contracts) are denoted as nodes, on-chain transactions are depicted as edges. Participants deposit to a mixer contract. Afterwards off-chain they threshold sign a *payOut* transaction which later can be issued by anyone to "poke" the mixer contract in order to make the mixer contract sending mixed coins out to participants' fresh keys.

The implemented workflow of our mixing protocol works as follows (for visual depiction see Figure 2): users deposit equal amount of funds to the mixer contract. Participants threshold sign a message authorizing the mixer contract to send out mixed coins to their fresh addresses. To make this happen someone needs to "poke" the contract. We call this party an *Activator*. The Activator can be any of the participants or a non-trusted third party. In the near future we expect to see in production-ready adaptations of ShareLock that the role of the activator will be taken by wallet companies who are providing such privacy-enhancing services to their customers and wallet users. Mixer participants will pay some negligible fees for the activator to incentivize her in sending the threshold ECDSA signed transaction. In the last step of the protocol, the "poked" mixer contract autonomously sends out mixed coins to participants' fresh addresses if the threshold signed transaction is verified against the DKG public key.

We expect to see that the computational time complexity of the off-chain part of our protocol remains reasonably lightweight and costs will be dominated by network latency. Gennaro et al. [11] presents promising performance results for their threshold ECDSA protocol. For realistic settings, like 20 parties, it takes less than 0.5 second to sign a message.

5.2 Off-chain/On-chain Separation

ShareLock Protocol 4.2 can be divided into two processes: *off-chain* between parties $\mathcal{P}_j|_{j=1}^n$ which includes steps 1 and 3 and *on-chain* between parties $P_i|_{i=1}^n$ which includes the rest of the protocol.

The parties $P_i|_{i=1}^n$ play the role of the senders in our scheme and $\mathcal{P}_j|_{j=1}^n$ play the role of the recipients. It is therefore required that it will be impossible to connect between a specific sender to specific recipient. The P_i's are identified by their on-chain IDs which are the input addresses $in_1, \ldots, in_n$ that fund the mixing contract. The recipients on-chain IDs are the output addresses that receive funds from the mixing contract: $ak_1, \ldots, ak_n$. If there is a way to form a link $in_i \to P_i \to \mathcal{P}_j \to ak_j$ it will break the anonymity property. To facilitate a decoupling the protocol only specifies that the logical indices for the DKG are needed to be chosen randomly. While the protocol is not specific on how to decouple P_i from $\mathcal{P}_j$ we claim that in practice it is a reachable goal.

One idea, which we use in our implementation, is using the network layer of the protocol. We simply set the indices of the parties running key generation (which will stay the same for signing) based on the order of the parties joining the protocol. Because of the asynchronous nature of real world networks we expect some random distribution of arrival time, even assuming all parties transmitted the first message at the same time. Adding some random delay to first message will therefore result in random joining time to the key generation protocol. The problem that remains is how to translate this randomness to unique indices $1, .., n$ since each party in the protocol will have a different view of arrival times of messages from other parties. To do that we suggest to run the DKG using a relay server RS. The first stage of the protocol will be a registration step where RS will be the reference point to decide a unique indices based on time of arrival of a registration message from each party. RS will publish this indexing between the participating parties. We note that to avoid a single point of failure the relay server might be implemented using a group of servers running consensus protocol between them.

Limitations of privacy-enhancing solutions In our solution it is possible to achieve full separation between input index and output index of a party. Define $\mathbb{P}_{i,j} = \{P_i, \mathcal{P}_j\}$ as party index and use party P_i for on-chain purposes and party $\mathcal{P}_j$ for off-chain purposes. For example a user can run a full node P_i on one machine and run a threshold computation as $\mathcal{P}_j$ from another machine. Still, a powerful attacker might correlate IP addresses or Tor endpoints. This de-anonymization attack vector is inherent to all cryptocurrencies and privacy-enhancing solutions. No cryptographic tool (zero-knowledge, ring or threshold signatures) could defeat an attacker who is well embedded and infiltrated the network layer with her own nodes to log user activity.

Table 1: Number of on-chain transactions and off-chain messages per a single participant required to run a certain coin mixer protocol. In ShareLock each party deposits into the mixer contract and afterwards one of the participants need to issue a single "poke" transaction (hence the $1/n$ term), which sends out the mixed funds for all the mixer participants. Note that in case of Miximus if one wants to avoid the trusted setup for the zkSNARK, then they need to perform a secure multi-party computation protocol to trust-minimize the proving key generation.

	#Off-chain messages	#Transactions
Centralized		
Mixcoin [5]	2	2
Blindcoin [31]	4	2
TumbleBit [15]	12	4
Decentralized		
Coinjoin [18]	$\mathcal{O}(n^2)$	1
Coinshuffle [28]	$\mathcal{O}(n)$	1
XIM [4]	0	7
Möbius [19]	2	2
Miximus [3]	1+MPC	2
MixEth [30]	1	3
MixEthChannel [30]	$\mathcal{O}(n)$	2
ShareLock	$\mathcal{O}(n)$	1+1/n

Table 2: Proof-of-concept implementation gas cost results. In MixEth, in addition to deposit and withdrawal transactions, one also needs to perform shuffle transactions, therefore the total gas consumption is more than the sum of deposit and withdrawal transactions' gas cost. The number of participants and the number of malicious shuffles are denoted as n and k respectively.

	Deposit	Withdraw	Total
Möbius [19]	76,123	335,714*n	76,123+335,714*n
Miximus [3]	732,815	1,903,305	2,636,120
MixEth [30]	99,254	113,265	578,735+10,000*n+227,563*k
ShareLock	104,306	31,000	135,306

6 Acknowledgements

We would like to thank Claudio Orlandi, Adam Ficsor and Harry Roberts for insightful discussions.

References

[1] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in bitcoin. In *International Confer-*

ence on Financial Cryptography and Data Security, pages 34–51. Springer, 2013.

[2] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Annual International Cryptology Conference*, pages 324–356. Springer, 2017.

[3] barryWhiteHat. Miximus. https://github.com/barryWhiteHat/miximus, 2018.

[4] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. Sybil-resistant mixing for bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 149–158. ACM, 2014.

[5] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

[6] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world.

[7] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 136–145. IEEE, 2001.

[9] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *Threshold ECDSA from ECDSA Assumptions: The Multiparty Case*, page 0. IEEE.

[10] Michael Fleder, Michael S Kester, and Sudeep Pillai. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657*, 2015.

[11] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ecdsa with fast trustless setup. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1179–1194. ACM, 2018.

[12] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[13] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[14] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In *Theory of Cryptography Conference*, pages 529–561. Springer, 2017.

[15] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017.

[16] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1837–1854. ACM, 2018.

[17] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013. *URl: https://bitcointalk. org/index. php*.

[18] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum*, 2013.

[19] Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121, 2018.

[20] Sarah Meiklejohn and Claudio Orlandi. Privacy-enhancing overlays in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 127–141. Springer, 2015.

[21] Malte Möser, Rainer Böhme, and Dominic Breuker. An inquiry into money laundering tools in the bitcoin ecosystem. In *2013 APWG eCrime Researchers Summit*, pages 1–14. Ieee, 2013.

[22] Waka Nagao, Yoshifumi Manabe, and Tatsuaki Okamoto. Relationship of three cryptographic channels in the uc framework. In *International Conference on Provable Security*, pages 268–282. Springer, 2008.

[23] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[24] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

[25] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

[26] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

[27] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[28] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *European Symposium on Research in Computer Security*, pages 345–364. Springer, 2014.

[29] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[30] István András Seres, Dániel A Nagy, Chris Buckland, and Péter Burcsi. Mixeth: efficient, trustless coin mixing service for ethereum.

[31] Luke Valenta and Brendan Rowan. Blindcoin: Blinded, accountable mixes for bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 112–126. Springer, 2015.

[32] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.