MixEth: efficient, trustless coin mixing service for Ethereum

István András Seres[1], Dániel A. Nagy[1], Chris Buckland[2], and Péter Burcsi[1]

[1]Department of Computer Algebra, Eötvös Loránd University
[2]King's College London

February 19, 2019

Abstract

Cryptocurrencies enable users to transact pseudonymously with each other without relying on trusted parties or intermediaries. These transactions are recorded in an immutable, publicly verifiable ledger. Due to the ledger's transparent nature, privacy is notably reduced. If the link between users' public key and their physical identity is exposed, their pseudonymity is lost. One way to increase users' privacy is to deploy coin mixing services. In this paper, we present MixEth, which is a trustless coin mixing service. MixEth is more efficient than any current proposed trustless coin tumbler. It requires at most only 3 on-chain transactions per user and 1 off-chain message. It achieves strong notions of anonymity and is able to resist denial-of-service attacks. Furthermore the underlying protocol can also be used to efficiently shuffle ballots and cyphertexts in a trustless and decentralized manner.

Keywords: Cryptography, Verifiable shuffle, Ethereum, Privacy, Coin mixer, State Channel

1 Introduction

One of the methods to increase cryptocurrency users' privacy is coin mixing or tumbling. This technique provides *k-anonymity* or *plausible deniability.* The idea is that k users deposit 1 coin each and then in the course of a coin shuffling protocol either a centralized trusted third party or a smart contract mixes the coins and redistributes them to designated fresh public keys. This powerful technique gives users superior privacy and anonymity since their newly received coins can not be linked to them.

Several coin mixing protocols were proposed in the literature both centralized ([7], [27], [14]) and decentralized ([15], [26], [2], [17], [5]). A major drawback of centralized coin mixing is that the availability of the tumbler is entirely dependent on the trusted party and in most cases theft prevention can not be guaranteed ([7], [27]). On the other hand decentralized tumblers achieve availability, theft prevention and satisfy strong notions of anonymity although they are considerably heavier computationally.

The two major techniques to provide decentralized mixing services for Ethereum are Möbius, a ring-signature-based solution [17] and Miximus, a zkSNARK-based proposal [2]. Both of them require usage of a large amount of gas to withdraw funds, which could be prohibitive for many use cases. Möbius requires 335,714n gas (n is the ring size) while Miximus consumes 1,903,305 gas to verify a zkSNARK proof [3].

Our contributions. In this paper, we present a trustless and efficient mixing protocol for Turing-complete blockchains. To show the practicality of the protocol, we introduce MixEth, a privacy-enhancing protocol and a practical tool for Ethereum. MixEth overcomes the above mentioned efficiency issues of an Ethereum-based coin-mixer while retaining the strong notions of anonymity, mixer availability and theft prevention already achieved by previous proposals ([17],[2]). MixEth requires as few off-chain messages and on-chain transactions as Möbius and Miximus, meanwhile it burns significantly less gas.

We also implement MixEth in a state channel to leverage the scalability and instant finality of off-chain scaling solutions.

2 Background

2.1 Notations

Let [] denote the empty tuple. For a tuple $t = (x_1, \dots, x_n)$ we denote $t[x_i]$ as the value stored at x_i. The cardinality of a finite set X is denoted as $|X|$. In the following let $\lambda \in \mathbb{N}$ be the security parameter and its unary representation is 1^λ. If x is uniformly randomly sampled from a set A we write $x \overset{\$}{\leftarrow} A$. The symmetric group of degree n is written as S_n. In a cyclic group $\mathbb{G}$, the standardized generator is denoted as G and we use the additive notation. Secret keys and public keys are denoted as sk and pk respectively (or often times s and sG), while the user the corresponding key belongs to is indicated in subscript. Let PK_i denote the set of public keys belonging to receivers at a particular shuffling round i.

2.2 Cryptographic keys in Ethereum

Ethereum uses Elliptic Curve Cryptography (ECC) to secure users' funds. More specifically, it uses the secp256k1 curve, the same one as used in Bitcoin. If a user wants to create an Ethereum address, first they need to generate a secret key $s \overset{\$}{\leftarrow} \mathbb{Z}_n$, where n is the order of secp256k1 over a finite prime field $\mathbb{F}_p$. The corresponding public key will be sG. Note, that any multiples of G is also a generator of curve points since n, the order of the group is also a prime. Accounts in Ethereum are identified by their addresses which can be obtained by taking the right most 20 bytes of the Keccak hashed public key [28].

2.3 Decision Diffie-Hellman Problem and Chaum-Pedersen Protocol

The Decision Diffie-Hellman assumption (DDH) is a standard cryptographic hardness assumption which underlies the security of many cryptographic protocols. Roughly speaking DDH states that no efficient algorithm can distinguish between the two distributions (aG, bG, abG) and (aG, bG, cG), where $a, b, c \overset{\$}{\leftarrow} \mathbb{Z}_{|\mathbb{G}|}$. It is believed that the DDH assumption holds for elliptic curves with prime order over a prime field with large embedding factor [6], specifically DDH holds for the secp256k1 curve used in Ethereum.

The language $\mathcal{L}_{DDH}$ is defined to be the set of all tuples (G, aG, bG, abG) where $G \in \mathbb{G}$ is of order prime q. The Chaum-Pedersen protocol enables a prover $\mathcal{P}$ to prove to a verifier $\mathcal{V}$ that $(G, A, B, C) \in \mathcal{L}_{DDH}$ in zero-knowledge for groups of prime order [10]. The protocol is organized as follows:

1. $\mathcal{V}$: $s \overset{\$}{\leftarrow} \mathbb{Z}_q$, then sends $commit(s)$
2. $\mathcal{P}$: $r \overset{\$}{\leftarrow} \mathbb{Z}_q$, then sends $y_1 = rG$, $y_2 = rB$.
3. $\mathcal{V}$ opens commitment by sending s
4. $\mathcal{P}$ sends $z = r + as \pmod q$
5. $\mathcal{V}$ checks $zG = y_1 + sA \pmod q \wedge zB = y_2 + sC \pmod q$

Note that in the following a non-interactive version of this protocol will only be considered. This can be achieved by applying the Fiat-Shamir heuristic.

2.4 ECDSA with arbitrary generator element

Elliptic Curve Digital Signature Algorithm (ECDSA) is a key component of MixEth. ECDSA is widely deployed in practice, where in most cases signatures are generated and verified with respect to a fixed generator element of the underlying group [13]. Since all generators are equal from a security point of view, a single generator element is usually fixed in order to promote standardization and assist usability.

However, in MixEth, we require the use of arbitrary generator elements. Such an extension is needed for withdrawing funds from the mixer as the shuffled public keys remain public keys with respect to non-standardized generator elements. Therefore the usual Sig and Vf algorithms for signing and verifying messages require an additional parameter $G^{'}$, which is not necessarily the standardized generator element. Key generation algorithm works as usual $(pk, sk) \overset{\$}{\leftarrow} \mathsf{KGen}(1^\lambda)$, on the other hand $\sigma \overset{\$}{\leftarrow} \mathsf{Sig}(G', sk, m)$ and $0/1 \leftarrow \mathsf{Vf}(G', pk, \sigma, m)$ accept new generators.

3 Security goals

We are aiming to achieve and prove the same notions of security as the ones defined in [17], namely anonymity, availability and theft prevention. These notions of anonymity, availability and theft prevention were introduced in [17].

We are going to assume that at most $n-2$ recipients are malicious (n is the number of recipients). Otherwise, no meaningful notion of security can be achieved. Furthermore we presume that recipients are on-line during the entire course of mixing in order to be able to monitor and potentially challenge any incorrect shuffle. Finally we assume that honest participants will always exercise their rights to shuffle and they do not disclose any private information used in their shuffles.

- **Anonymity**: *Sender* anonymity is achieved if an adversary can not determine to whom honest senders are sending funds, assuming that honest senders' deposits are indistinguishable. *Recipient* anonymity is achieved if honest recipients withdrawal transactions are indistinguishable.
- **Availability**: means that honest recipients can always withdraw their assets from the mixer, even if all participants are compromised. Adversary $\mathcal{A}$ wins the availability security game if they manage to get the tumbler into a state where honest recipient can not withdraw their funds.
- **Theft prevention**: We would like to ensure that neither coins can be withdrawn twice, nor withdrawn by anyone other but the intended recipient.

4 MixEth

4.1 Initializing the tumbler and depositing period

A MixEth contract is deployed on the Ethereum blockchain at $id_{contract}$ address and is initialized with the amt parameter which denotes the denomination of ether to be mixed. Every sender must deposit exactly amt ether to a specific public key. Deposits with incorrect ether value or invalid public key are rejected. Public keys in subsequent deposit transactions are written into the $initPubKeys[]$ array.

4.2 Shuffling period

After the depositing round the shuffling and challenging rounds alternate in turn. Each shuffling round is followed by a challenging round where the correctness of the preceding shuffle can be challenged by anyone. If a challenge is accepted, then shuffler's deposit is lost, their shuffle is discarded and shuffling continues from the set of public keys prior to the discarded shuffle. In the course of a shuffle an honest shuffler multiplies all the public keys with a secret multiplier c and then permutes the transformed public keys. This shuffling is done off-chain, then the shuffler commits to c by sending back to MixEth the new shuffling accumulated constant and the shuffled public keys. This allows anyone to verify the shuffle and, if required, submit a challenge during the succeeding challenge round.

Procedure 1 Off-chain public key shuffling algorithm for the ith shuffling round

```
$PK_i \leftarrow [], c \overset{\$}{\leftarrow} \mathbb{Z}_n, \pi \overset{\$}{\leftarrow} S_{|PK_{i-1}|}$
$C^*_{i-1} \leftarrow read\ from\ MixEth\ contract$
$PK_{i-1} \leftarrow read\ from\ MixEth\ contract\ the\ current\ sequence\ of\ shuffled\ public\ keys$
for $j = 0; j < |PK_{i-1}|; j++$ do
    $PK_i[\pi(j)] = c * PK_{i-1}[j]$
end for
$C^*_i = cC^*_{i-1}$
        Output: $(PK_i, C^*_i)$
```

4.3 Challenging period

Every participant should check the correctness of incoming shuffles, therefore sufficient time should be provided for each challenging round. These are the actions Bob as a receiver needs to perform to check the correctness of the shuffle at ith round if Bob has secret key s_B. In this case Bob should check whether $s_B C_i^* \in PK_i$ or not. If not, Bob should prove to MixEth that the ith round is indeed the first round, where the shuffled public key corresponding to s_B is compromised. The Chaum-Pedersen proof in the challenge transaction ensures that the integrity of the shuffled public key in round $i-1$st is intact, while shuffled public key is compromised in the ith round.

Procedure 2 On-chain verification algorithm of incoming shuffle challenges

```
    Input($PK_i, PK_{i-1}, proof_{ChP}(C^*_{i-1}, s_B C^*_{i-1}, C^*_i, s_B C^*_i)$)

1: $b \leftarrow verifyChaumPedersen(proof_{ChP}(C^*_{i-1}, s_B C^*_{i-1}, C^*_i, s_B C^*_i)$
2: $b^* \leftarrow 0$
3: if $b \wedge s_B C^*_{i-1} \in PK_{i-1} \wedge s_B C^*_i \notin PK_i$ then
4:     $b^* \leftarrow 1$
5: else
6:     $b^* \leftarrow 0$
7: end if    Output: $b^*$
```

4.4 Withdrawing

Let C^*_{final} be the final shuffling accumulated constant. For a recipient B, whose public key $s_B G \in initPubKeys[]$, in the final shuffle there will be $s_B C^*_{final}$. The recipient can prove to MixEth that she knows secret key s_B by signing their public key using a modified ECDSA, which uses C^*_{final} as the generator element instead of the standardized G.

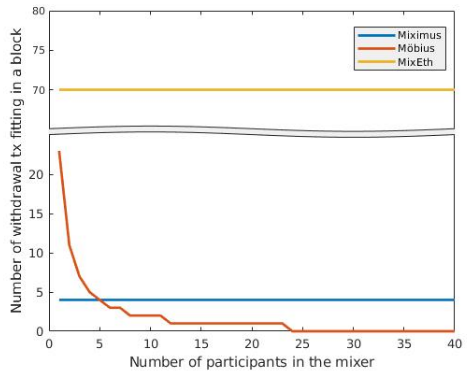

Figure 1: Gas complexity of withdraw txs for various trustless Ethereum coin mixers

5 Informal reasoning about security properties

- **Recipient anonymity** The withdrawing transaction for recipient B sends funds to the public key $s_B C^*$. This public key does not reveal any links to the original $s_B G$ in case if at least one honest participant shuffled and the DDH assumption holds. Adversary can only distinguish between honest recipients public keys with negligible probability.

- **Availability** If an adversary is able to destroy an honest recipient's funds' availability, it implies that adversary $\mathcal{A}$ either breaks the completeness of the Chaum-Pedersen protocol or successfully launched an eclipse attack against the honest recipient, who can not send any transactions to honest Ethereum peers.

- **Theft prevention** If an adversary is able to steal funds from other users than it would imply that they managed to create a valid message/signature, (m, σ) pair for the final shuffled public key of an honest recipient without having access to the secret key of the honest recipient. This contradicts to the assumption that ECDSA is existentially unforgeable.

6 Implementation

We implemented MixEth in Solidity with two different approaches [1]. The first one does not apply state channels, all the transactions are made on-chain. This could lead to unwanted gas costs as the number of corrupted shuffles increases. One of our main motivation with MixEth is to provide an efficient and scalable coin mixing protocol which uses as little blockchain resources, storage and gas, as possible. Therefore we also implement and evaluate MixEth applying state channels, namely shuffling and challenging a shuffle occurs off-chain and only deposit and withdrawal transactions happen on-chain. Both of the implementations allow users to mix Ether or other ERC20-compatible tokens.

The bottleneck of coin mixing protocols is the withdrawal transactions' gas costs. A Miximus withdrawal transaction burns 1,903,305 gas, regardless of the number of participating parties. Since the block gas limit is 8,000,266 as of 2018, October 24 only 4 Miximus withdrawal transactions could fit in one Ethereum block. This is even worse for Möbius, since the gas cost for withdrawing coins from a Möbius mixer contract linearly increases with the numbers of participants.

6.1 Fully on-chain implementation

Conceivably mixers would like to minimize off-chain coordination, therefore in our first implementation of the MixEth protocol, we assumed that all transactions will take place on-chain. There is only a single off-chain message from receiver to sender, where receiver delivers their public key to the sender. The rest of the protocol happens entirely on-chain.

On-chain storage is extremely expensive: it requires 20,000 gas to store a 256-bit number, however if a particular storage slot is already taken and one wants to overwrite it with a non-zero element than storing only consumes 5000 gas. To minimize on-chain storage costs, only the last two list of shuffled public keys are stored in the MixEth contract's permanent storage. Note that storing only the latest list of shuffled public keys would not be enough, since honest receivers could not prove to the contract that their shuffled public key is compromised unless also the last-but-one list of shuffled keys is also available for the contract to check the Chaum-Pedersen proof against. Such a storage structure implies that after uploading the new list of shuffled public keys, a challenging period should proceed in order to let receivers check the correctness of the shuffle and whether their shuffled public key is stored in the smart contract. Furthermore we also allow senders to shuffle and deposit new public keys at the same time, meaning that only 3 on-chain transactions (shuffle, withdraw mixed coins and withdraw shuffling deposit) are sufficient to complete the protocol. We left it as a future work to determine the optimal ratio of mixed coins and shuffling deposits in order to incentivise correct shuffling.

A great advantage of the fully on-chain version of MixEth is that it allows dynamic anonymity sets. One could potentially deposit funds to the contract and shuffle public keys and leave funds in the mixing contract for indefinite amount of time. As soon as the anonymity set is large enough a receiver could withdraw their assets. A receiver in a MixEth contract with N senders could withdraw their funds after $N^{'}$ shuffling rounds, where $N^{'}$ is arbitrary. This dynamic nature of the contract could even lead to a single monolithic MixEth contract instead of having multiple MixEth contracts with significantly fragmented anonymity sets. A single MixEth contract is able to support the mixing of ether and ERC-20 compatible tokens as well. However note that the gas complexity of shuffling transactions grow linearly in the number of participants, therefore the fully on-chain implementation is not capable to support extremely large anonymity sets with participants more than cca. 800.

Also note that the on-chain cost of a shuffle transaction could be amortized among participants. Specifically if there is a group of senders or receivers who trust each other they could collectively charge any of them to shuffle once. Since they trust their peer, all the rest of the group does not need to shuffle anymore. Later non-shufflers could reimburse the only shuffler for their services either on-chain or off-chain.

6.2 State channel implementation

We have also adapted MixEth to operate within a state channel. We wrote the implementation within the guidelines of the Counterfactual framework [11]. This allowed us to delegate the pro-

[1] https://github.com/seresistvanandras/MixEth

cesses of setup, liveness disputes and finalisation to the framework so that we could focus on adapting the application logic. Unlike the on-chain implementation the state channel implementation requires that the set of participants be agreed upon upfront. In state channels each update to the state needs to be signed by all other participants, this means that state channel applications are inherently at least $\mathcal{O}(n)$. To co-ordinate these off-chain updates the Counterfactual framework enforces that all applications be turn based, introducing a turn taker for each turn who may propose a new state. The original MixEth implementation was not turn based so we have adapted the application to this constraint, an example of this adaptation is the challenge round. In the on-chain implementation a time period is allowed during which any participant may challenge, we have adapted this by proceeding turn-based through the participants offering each the chance to either challenge or pass. In the case of a breakdown in cooperation in the channel, a liveness fault, it has been shown that all operations succeeding the cooperation breakdown must proceed on chain[16] or be abandoned at some financial cost specified by the application, meaning that if every shuffle were to be succeeded by a challenge round each participant would be forced, by threat of lost deposit, to make an on-chain transaction after each shuffle, incurring $\mathcal{O}(n)$ on chain operations. To mitigate this we removed the challenge after each round and instead introduced a challenge round that takes place after all shuffles have completed, during this round any of the preceding shuffles may be challenged.

Given these adaptations the application proceeds as follows, all participants including senders, shufflers and receivers, deposit funds in a mutli-signature wallet compatible with the Counterfactual framework, they then follow the installation protocols specified by the framework to install the adapted MixEth logic. Afterwards each participant signs a transaction that transfers an equal amount to each withdrawer from the multi-sig, dependent on correct execution of the channelised MixEth application logic. This application logic proceeds as follows: each sender names a public key of a shuffler as in the deposit stage of the on-chain application, then each shuffler takes it in turn to shuffle. After all shuffles have taken place each withdrawer is given a turn to either declare fraud or no-fraud on any shuffle round. Finally each withdrawer then provides proof of ownership by submitting a valid signature on the modified ECDSA scheme.

If any of these steps does not occur, or does not occur correctly, the protocol aborts and the conditional transfer does not occur. In this case the perpetrator loses a deposit, either through fraud proof or through failure to take their turn when state is published on-chain. A further modification would be to distribute the slashed deposit to each of the other participants, compensating them for their lost time and the gas costs associated with proving the fault of the other party.

Following this protocol the on-chain transactions are now reduced to: one transaction from each participant to deposit funds into the multi-sig, and a set of transactions that send funds from the multi-sig to each of the withdrawers and deposits back to each of the other participants.

Table 1: Proof-of-concept implementation gas cost results. MixEthChannel refers to the implementation which leverages state channels for shuffling and challenging periods

	Deployment	Deposit	Shuffle		Withdraw
			Shuffle upload	Challenge	
Möbius [17]	1,046,027	76,123	0	0	335,714n
Miximus [2]	1,751,378	732,815	0	0	1,903,305
MixEth	5,395,945	99,254	$138{,}653 + 10{,}000n$	227,563	113,265
MixEthChannel	672,276	21,000	0	0	26,749

References

[1] Aztec. The aztec protocol. https://github.com/AztecProtocol/AZTEC, 2018.

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

[3] barryWhiteHat. Miximus gas costs. https://www.reddit.com/r/ethereum/comments/8ss53z/miximus_zksnark_based_anonymous_transactions_is/, 2018.

[4] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 263–280. Springer, 2012.

[5] 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.

[6] Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

[7] 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.

[8] Vitalik Buterin and Nick Johnson. Eip86: Account abstraction. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-86.md, 2017.

[9] Wren Chan and Aspen Olmsted. Ethereum transaction graph analysis. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 498–500. IEEE, 2017.

[10] David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Annual International Cryptology Conference*, pages 89–105. Springer, 1992.

[11] Jeff Coleman, Liam Horne, and Li Xuanji. Counterfactual: Generalized state channels, 2018.

[12] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive, 2017: 635, 2017.

[13] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (ec) dsa signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1651–1662. ACM, 2016.

[14] 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.

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

[16] Patrick McCorry, Chris Buckland, Surya Bakshi, Karl Wüst, and Andrew Miller. You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies.

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

[18] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[19] Pedro Moreno-Sanchez, Muhammad Bilal Zafar, and Aniket Kate. Listening to whispers of ripple: Linking wallets and deanonymizing transactions in the ripple network. *Proceedings on Privacy Enhancing Technologies*, 2016(4):436–453, 2016.

[20] Malte Moser, Rainer Bohme, and Dominic Breuker. An inquiry into money laundering tools in the bitcoin ecosystem. In *eCrime Researchers Summit (eCRS), 2013*, pages 1–14. IEEE, 2013.

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[22] Nchinda Nchinda. Exploring pseudonimity on ethereum. https://media.consensys.net/exploring-pseudonimity-on-ethereum-dda257019eb4, 2016.

[23] Serge Nedashkovsky. Huge ethereum mixer. https://blog.cyber.fund/huge-ethereum-mixer-6cf98680ee6c, 2017.

[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] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[26] 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.

[27] 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.

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