Secure and Anonymous Decentralized Bitcoin Mixing

Jan Henrik Ziegeldorf*, Roman Matzutt, Martin Henze, Fred Grossmann, Klaus Wehrle

Communication and Distributed Systems (COMSYS),
RWTH Aachen University, Ahornstrasse 55, 52074 Aachen, Germany

Abstract

The decentralized digital currency Bitcoin presents an anonymous alternative to the centralized banking system and indeed enjoys widespread and increasing adoption. Recent works, however, show how users can be reidentified and their payments linked based on Bitcoin's most central element, the blockchain, a public ledger of all transactions. Thus, many regard Bitcoin's central promise of financial privacy as broken.

In this paper, we propose *CoinParty*, an efficient decentralized mixing service that allows users to reestablish their financial privacy in Bitcoin and related cryptocurrencies. *CoinParty*, through a novel combination of decryption mixnets with threshold signatures, takes a unique place in the design space of mixing services, combining the advantages of previously proposed centralized and decentralized mixing services in one system. Our prototype implementation of *CoinParty* scales to large numbers of users and achieves anonymity sets by orders of magnitude higher than related work as we quantify by analyzing transactions in the actual Bitcoin blockchain. *CoinParty* can easily be deployed by any individual group of users, i.e., independent of any third parties, or provided as a commercial or voluntary service, e.g., as a community service by privacy-aware organizations.

Keywords: Bitcoin, digital cash, Privacy-preserving protocols, Pseudonymity, anonymity and untraceability

1. Introduction

The digital currency Bitcoin was proposed in 2008 by Nakamoto [46] as an anonymous alternative to the centralized banking system. Since then, Bitcoin enjoys widespread adoption, e.g., in July 2015 the market cap of circulating Bitcoins exceeded $4 Billion [8]. To keep track of the balances and to establish trust in the currency, all Bitcoin transactions are stored in a distributed public ledger, the *blockchain*, instead of using centralized entities, e.g., banks. To receive, store, and spend Bitcoins, users maintain cryptographic identities called *addresses*, which correspond to Elliptic Curve Digital Signature Algorithm (ECDSA) public keys. Addresses and thus transactions are anonymous as long as addresses cannot be linked to their owners, which is usually ensured by using fresh unlinkable addresses as often as possible. It is especially this promise of financial privacy that has drawn great interest towards Bitcoin.

Recent works, however, raise serious doubts about Bitcoin's built-in privacy protection. They show how transactions and addresses of a user can often be linked together by analyzing the transaction graph in the publicly available blockchain, e.g., [2, 43, 49, 50]. Even worse, [6, 31] show that addresses can be linked to IP addresses, thereby completely de-anonymizing their owners. This raises the question how a user, Alice, can reestablish her financial privacy. Of course, Alice could generate fresh unlinkable addresses, but transferring funds to them would again link the address back to her. Hence, what Alice basically needs is a way to send funds from a de-anonymized address to her new address in an unlinkable manner. As of today, many commercial mixing services [11] have emerged that help Alice solve this problem: Alice sends her funds to the mixing service, which, after deducting a small fee, pays her back to a fresh address with the funds of some random other user. Thereby, ownership of addresses becomes untraceable to any passive observer of the blockchain, countering the de-anonymization approaches described in [2, 43, 49, 50].

[*] Corresponding author.
Email address: `ziegeldorf@comsys.rwth-aachen.de` (Jan Henrik Ziegeldorf)
URL: `www.comsys.rwth-aachen.de` (Jan Henrik Ziegeldorf)

Authors' version of a manuscript that was accepted for publication in *Future Generation Computer Systems*. Changes may have been made to this work since it was submitted for publication. Please cite the published version: http://dx.doi.org/10.1016/j.future.2016.05.018

Notably, a passive observer cannot even distinguish the mixing from other contemptuous non-mixing transactions in the blockchain. This property is desirable since it affords large anonymity sets and enables users to plausibly deny their participation in the mixing operation.

This first generation of mixes, however, suffers from two severe drawbacks which are well-known to the Bitcoin community: First, users need to blindly trust the operators not to steal their funds. Indeed, many allegations of theft circulate the Bitcoin community [10]. Second, the mixing service knows and may keep transcripts about how funds were mixed. A mixing service may then be breached, incentivized or forced to reveal these transcripts, e.g., by governmental agencies [28, 34, 55]. These prohibitive shortcomings have led to new decentralized approaches that promise stronger security and anonymity guarantees [40, 52, 60]. Most of these approaches require that all mixing transactions are issued in one single *atomic* transaction with multiple inputs and outputs. This prevents malicious peers from aborting the protocol after they have received their funds thereby leaving other peers unpaid. However, the characteristic form of such *group transactions* renders them easily identifiable in the blockchain, which introduces two severe limitations when used in mixing services: First, a user's anonymity set is limited to the number of users participating in the same mixing operation. Second, bundled mixing transactions are clearly identifiable in the blockchain, which deprives users of any means to plausibly deny that they participated in the mixing.

Motivated by the apparent lack of privacy demonstrated by the recent de-anonymization attacks and the shortcomings of existing mixing services in addressing this issue, we re-consider the problem of how a user can securely and anonymously mix her funds with those of other users to reestablish and preserve her financial privacy. In the following, we present a brief problem statement and then outline our contributions.

1.1. Problem Statement

We state the problem of secure and anonymous mixing as follows: n users, who each have at least v Bitcoins available at their input addresses $I_1, ..., I_n$ want to mix the amount v to a set of output addresses $O_1, ..., O_n$ such that (1) each user U_i receives back v Bitcoins on her output address O_i, and (2) output addresses are unlinkable, i.e., only user U_i knows that O_i belongs to her. We abstractly refer to a system that solves this problem as a *mixing service*. An ideal mixing service should fulfill the following requirements:

Correctness: Bitcoins must not be lost, stolen, or double-spent by any inside or outside party even in the presence of a malicious adversary. The mixing service should eventually succeed to mix funds of honest users, i.e., be resilient to Denial of Service (DoS) attacks from any inside or outside party. The resulting mixing should be random and no inside or outside party should be able to bias, reduce, or predict the randomness.

Anonymity: Mixing funds from input to output addresses must be anonymous, i.e., a malicious adversary must not be able to link any user U_i to her output address O_i. The anonymity set should be maximized.

Deniability: Users should be able to plausibly deny having participated in a mixing operation, i.e., a mixing operation should not produce any irrefutable evidence that binds a user to a mixing operation.

Scalability: The protocol should scale to large numbers of users without imposing prohibitive overheads upon the mixing service, the users, or the Bitcoin network.

Cost-efficiency: The protocol must be cost-efficient in terms of the involved mixing and transaction fees. Collection of mixing fees should be supported but optional.

Applicability & Usability: The mixing service should be fully backwards-compatible with the current Bitcoin network to allow for an incremental and practical deployment. The service should be easily usable and accessible, i.e., user interaction should be minimized and require only standard software, e.g., a Bitcoin client.

Clearly, the first generation of centralized mixes [9, 11, 13, 19] does not provide correctness in the presence of a malicious mix that steals funds [10]. Furthermore, anonymity is not fully achieved because the mixing service knows the permutation applied to the output addresses and may disclose this knowledge to third parties. While the systems proposed in [15, 40] protect against theft, users still need to trust the mix to protect their anonymity. Recent approaches, mostly through some form of decentralization, provide anonymity even if the mixing service itself is corrupted. However, they are either inefficient [7, 41, 44, 60], uneconomical [7, 41, 56], achieve only suboptimal anonymity and deniability [40, 51, 52, 56, 60], are incompatible with the current Bitcoin network without substantial modifications [5, 44], or have not evolved beyond discussions on forums and blogs [27, 41, 51, 60]. We defer a detailed analysis of related work to §6 and conclude from this preliminary analysis that secure and anonymous mixing of Bitcoins is still an open and important problem.

1.2. Our Contribution

In this paper, we present *CoinParty*, a usable, efficient and fully decentralized Bitcoin mixing service with strong guarantees of correctness, anonymity, and deniability. To achieve this goal, we introduce a set of *mixing peers* that, in a distributed yet secure fashion, carry out the mixing in multiple one-to-one standard Bitcoin transactions. Thereby, we replace the disadvantageous group transaction pattern used in related work and overcome its inherent restrictions on anonymity and deniability. Using multiple sequential transactions instead of one atomic group transaction, the key challenge is to ensure that all mixing transactions succeed even when a large fraction of users or mixing peers fail or behave maliciously. We achieve this by employing a threshold ECDSA signature scheme. This scheme allows to distributedly generate *escrow addresses*, from which funds can only be redeemed via a *threshold transaction*, i.e., only when a majority of the controlling peers agrees to do so. *CoinParty* thereby realizes the advantages of *both* the early centralized approaches and those of the later decentralized approaches in one system, achieving notable improvements compared to related work:

No trusted third party: *CoinParty* does not depend on any central trusted party, but instead runs as a distributed protocol executed by a set of mixing peers and thus eliminates any single point-of-failure w.r.t. DoS attacks or forced access by third parties. *CoinParty* remains provably secure and operable even if up to $\tau < m/3$ of the protocol peers are corrupted by a malicious adversary. According to well established results from the domain of Secure Multiparty Computation (SMC) [36], this threshold is optimal.

Improved anonymity: To any observer of the blockchain and even to the participants in a mixing, *CoinParty*'s threshold transactions are indistinguishable from other non-threshold standard Bitcoin transactions. Thus, *CoinParty*'s anonymity set comprises *all contemptuous* transactions that have the same value as the chosen mixing amount. We conduct a quantitative analysis on the blockchain to show that this indeed increases anonymity by up to two orders of magnitude over related work. Based on our analysis, we also recommend suitable mixing parameters that achieve these improvements.

Plausible deniability: *CoinParty* is first among related approaches for mixing Bitcoins to provide its users with plausible deniability against any passive observer and even against other participants of the mixing.

Applicability: We implement and evaluate a prototype of *CoinParty* to show that is is fully compatible with the existing Bitcoin network and incurs only small overheads even when scaling to large numbers of users. Users can easily access our *CoinParty* mixing service via a standard web-browser and no additional software is required.

A previous version of this work appeared in [62]. Although we uphold the same name and the basic design principles, we extend and improve on our previous works in many ways: First, [62] requires a trusted dealer to distribute initial randomness, while the system proposed in this work is completely decentralized. Second, [62] does not describe how users bootstrap to the mixing network. In this work, we provide a usable and secure bootstrapping mechanism that allows users to participate in a mixing using only their web browsers. Third, *CoinParty*'s resilience against DoS attacks is improved compared to [62] by redesigning different protocol phases. Fourth, we present in detail an extension that allows to recover from malicious attacks as well as to identify and punish the attacker. Finally, we re-evaluate all parts of the system, provide an updated anonymity analysis, consider further related work that appeared after [62], and generally provide more detail on all aspects of *CoinParty* and the related work.

The remainder of the paper is structured as follows: We present a short primer on the Bitcoin digital currency in §2. The basic cryptographic primitives and more advanced building blocks for our approach are presented in §3. The protocol and system design of *CoinParty* is laid out in §4. We provide a comprehensive analysis, evaluation, and discussion of correctness, performance, anonymity, and further system requirements in §5. Finally, *CoinParty* is compared to related work in §6 and §7 concludes this paper.

2. A Bitcoin Primer

Bitcoin is best understood as a decentralized peer-to-peer network that keeps track of all money transfers between its users. All transfers are grouped into blocks and recorded in a public ledger, the blockchain, which thus contains the complete history of accepted Bitcoin transactions. The blockchain is constantly validated by the Bitcoin participants. Adding a new block to the blockchain requires a prior proof-of-work such that diverging from the blockchain at an earlier point in time would require redoing all proof-of-works for the successor blocks. This protects against tampering and rules out double-spending of Bitcoins as long as the majority of computation power is contributed by honest non-colluding participants.

Addresses. Addresses are cryptographic identities used to receive, store and send Bitcoins. An address is essentially the hash of an ECDSA public key and the user in possession of the private key is the owner of the address. Addresses serve as pseudonyms and using fresh ones as often as possible is the basis for anonymity in Bitcoin.

Transactions. A transaction is a command to transfer Bitcoins from one address, the *input address*, to another address, the *output address*. The owner of the input address signs the transaction with the corresponding private key to prove that she is authorized to spend the stored funds. The signed transaction is broadcasted to the Bitcoin network, verified by the other peers and eventually included into the blockchain. A new transaction is usually accepted by other Bitcoin users after six subsequent blocks have been processed and a diversion of the blockchain is practically ruled out. Besides the simplest form of one-to-one transactions, Bitcoin also allows group transactions, i.e., transactions from multiple input addresses to multiple output addresses, all of which may be owned by different users. A group transaction is only considered valid by the Bitcoin network, if it has been signed with all private keys corresponding to the specified input addresses. This means that a group transaction can only ever be executed in its entirety or not at all, but never in parts. Thus, group transactions are atomic operations just as standard one-to-one transactions. Many existing mixing schemes (cf. §6) leverage this property to protect against theft from malicious peers.

Transaction fees. A transaction always redeems all funds available at the inputs. If less than these funds are transferred to the output addresses, the difference can be collected as a transaction fee. Transaction fees incentivize *miners*, i.e., peers who invest computing power to find new blocks, to include the transaction in the blockchain. Although transaction fees are voluntary, usually a fee of 0.0001 BTC per 1 kB of transaction size [12] is paid.

3. Cryptographic Building Blocks

The core building block of *CoinParty* is a threshold ECDSA cryptosystem, which we use to realize *escrow addresses* (§4.2) and *threshold transactions* (§4.4). Threshold cryptography is a special form of the general secure multiparty computation (SMC) problem. Thus, we briefly lay out the essential definitions, the security model and results of SMC in §3.1, which allow us to rigorously analyze the security of our proposed system. We then describe how to realize generic SMCs based on secret sharing in §3.2. These techniques are directly used in our proposed system, e.g., for verifying the integrity in our oblivious shuffling protocol (§4.3). Principally, generic SMC constructions could also be used to implement the required threshold ECDSA primitive, which serves as a building block for generating escrow addresses and threshold transactions. However, custom protocols are often far more efficient than generic constructions and thus we present existing efficient constructions for threshold ECDSA in §3.3.

3.1. SMC Definitions and Results

Problem Definition: The underlying challenge of SMC is the question how a group of m peers with private inputs $x_1, ..., x_m$, e.g., shares of a private key, can compute some known functionality $\mathcal{F}(x_1, .., x_m)$, e.g., a digital signature, in absence of a trusted third party and without learning each other's private inputs. Rigorous security definitions for SMC protocol are quite involved and we refer to [36] for an excellent and detailed overview. In short, most definitions involve the following properties that are also critical to our work: i) *Privacy:* Parties learn nothing about the other parties' inputs except for what can be derived from the output of the computation. ii) *Correctness:* If the computation succeeds, then the output is correctly computed and received by the honest parties. iii) *Robustness:* The computation cannot be disrupted or halted by corrupted parties, e.g., following the previous example, an attacker cannot stop the honest peers from generating a valid digital signature. Robustness is especially important in the context of our work: If an attacker could carry out a Denial-of-Service attack against *CoinParty*'s threshold transaction primitive, Bitcoins would be irreversibly lost.

Adversary models. Security of SMC protocols is usually analyzed in either the *semi-honest* or the *malicious* model. Semi-honest adversaries, also referred to as *passive* adversaries, correctly follow the protocol but may analyze the protocol transcript to gain additional information about the participants' private inputs. A semi-honest adversary is thus a threat to privacy but not to correctness or robustness. Contrarily, a *malicious* adversary is not bound by the protocol specifications and may *actively* deviate from the protocol in arbitrary ways, e.g., to prevent the computation of the result or to forge it. Because of the involved monetary values, we argue that it is mandatory to provide security against malicious adversaries in any mixing service.

Feasibility results. We emphasize that threshold cryptography is essentially a SMC problem. Thus, the following well-established results [36] also hold for all threshold cryptography primitives used in our work and provide bounds on the achievable security guarantees: Given m peers of which at most τ are corrupted by a malicious adversary. For $\tau < m/3$, SMC protocols with full privacy, correctness, and robustness can be achieved, e.g. [4, 25]. For $\tau \geq m/3$, protocols with privacy and correctness but without robustness can be achieved, e.g., [21]. Since, robustness is a crucial property for *CoinParty* as argued above, we can thus tolerate at most $\tau < m/3$ malicious peers in our setting.

3.2. Generic SMC via Secret Sharing

Among the different protocols for generic SMCs, many are based on linear secret sharing, e.g., Shamir's scheme [53]. Secret sharing schemes allow to distribute a secret value over different peers which can still perform operations on their shares but learn nothing from them about the secret. Secret sharing is also used in the specialized constructions for threshold ECDSA (§3.3) and during our shuffling protocol (§4.3).

On the highest level, SMC based on secret sharing proceeds as follows: The desired functionality $\mathcal{F}(x_1, .., x_m)$ is first expressed as an arithmetic circuit of addition and multiplication gates, which is subsequently evaluated by the peers. Each peer first secret-*shares* its private input to the other peers. Together, the peers then carry out the required *additions* and *multiplications* on the secret shares, i.e., without learning the secret inputs. At the end, the output is obtained in secret-shared form and can be *recombined*. In this fashion, theoretically any functionality can be implemented [4, 25, 36], while practically processing and communication overheads limit what is feasible [58, 61]. To instantiate the presented high-level construction, we still need to provide protocols for the four required primitive operations, i.e., Share, Recombine, Add, and Multiply. In the following, we outline how we instantiate the four primitives with security in the required malicious model.

$[a] \leftarrow$ Share(a, t, m) shares a secret a across m peers such that nothing can be learnt about a from any subset of t or less shares. Specifically, using Shamir's secret sharing scheme [53], a peer draws a random polynomial $f_a \in \mathbb{Z}_p[X]$ of degree t with $f_a(0) = a$, computes $[a] = (f_a(1), ..., f_a(m))$ and sends $[a]_i$ to peer $i = 1, ..., m$. In the malicious adversary model, we additionally need to make sure that a malicious peer cannot equivocate, i.e., send inconsistent shares to the other peers. This is usually achieved through commitments, e.g., in Pedersen's Verifiable Secret Sharing (VSS) [47], which we use in this work. Using Pedersen's VSS, Share is secure for any $\tau \leq t \leq m$ malicious peers.

$a \leftarrow$ Recombine$([a])$ is dual to Share and reconstructs the secret a from any set of at least $t + 1$ shares using Lagrange interpolation, i.e., $a = f_a(0) = \sum_{i=i_1}^{i_{t+1}} [a]_i \cdot \prod_{j=0\ j\neq i}^{m} \lambda_{i,j}$ with $\lambda_{i,j} := \frac{j}{i-j}$ the Lagrange basis polynomial at point $x = 0$. In the malicious model, we need to filter out inconsistent shares sent by malicious users to forge the value of recombined secrets. Interpreting the shares $[a]_i$ as a Reed-Solomon code [48] for a, as proposed by [42], and then using the widely known Welch-Berlekamp algorithm [59] yields a Recombine operation that can correct up to $m/3 - 1$ erroneous shares. Therefore, Recombine is secure against $\tau < m/3$ malicious peers.

$[a + b] \leftarrow$ Add$([a], [b])$ computes the sum of two secret-shared values a and b revealing neither the values nor the result. In Shamir's and any other additive secret sharing scheme, Add can be computed locally by each peer simply by adding the private shares, i.e., $[a+b]_i = [a]_i + [b]_i$. For simplicity, we use the notation $[a+b] = [a] + [b]$ instead of $[a + b] \leftarrow$ Add$([a], [a])$ throughout this work. Scalar multiplication can be computed locally as $[s \cdot a] = s \cdot [a]$, which is equivalent to executing $s - 1$ additions. Since Pedersen's VSS is linear, Add has the same security as Share and can tolerate $\tau \leq m$ malicious peers [47].

$[a \cdot b] \leftarrow$ Multiply$([a], [b])$, i.e., multiplication of two secret-shared values a and b, can be implemented in an interactive protocol [4], which is secure against $\tau < m/2$ passively and $\tau < m/4$ actively corrupted peers. A protocol that is secure against $\tau < m/3$ actively corrupted peers has only recently been described and fully proved in [3]. Throughout this work, we use the shorthand $[a \cdot b] = [a] \odot [b]$ for $[a \cdot b] \leftarrow$ Multiply$([a], [b])$.

```
Protocol EcDkg

Input: Generator $G$ of Elliptic Curve $\mathcal{P}_E$, point $H \in_R \mathcal{P}_E$
Output: Share of private key $[d]_i$, public key $D = dG$.

  1.1 Choose and share secret $s_i$ and nonce $r_i$ using random polynomials $S_i(x) = \sum_{k=0}^{t} s_{i,k}x^k$ and $R_i(x) = \sum_{k=0}^{t} r_{i,k}x^k$, with $s_{i,0} := s_i$ and $r_{i,0} := r_i$.
  1.2 Broadcast $C_{i,k} \leftarrow s_{i,k}G + r_{i,k}H, k = 1...t$.
  1.3 Check $[s_j]_iG + [r_j]_iH = \sum_{k=0}^{t} i^k C_{j,k}$. If check fails, broadcast a complaint against $P_j$. Disqualify any $P_j$ that was complained against at least $t + 1$ times.
  1.4 Compute share $[d]_i \leftarrow \sum_{j \in Q}[s_j]_i$ of the private key, with $Q$ the set of qualified peers.
  1.5 Broadcast $S_{i,k} \leftarrow s_{i,k}G, 0 \leq k \leq t$ .
  1.6 Check $[s_j]_iG = \sum_{k=0}^{t} i^k S_{j,k}, 1 \leq j \neq i \leq n$. If check $j$ fails, complain by broadcasting $[s_j]_i, [r_j]_i$. If valid complaints against $P_j$ are received, reconstruct $s_j$ and compute $S_{j,0}$.
  1.7 Compute public key $D = dG$ by $D \leftarrow \sum_{j \in Q} S_{j,0}$.
```

Protocol 1: The Distributed Key Generation scheme by Gennaro et al. [23] adapted to Elliptic curves as executed by a peer P_i.

```
Protocol ThresholdEcdsa

Input: Generator $G$ of Elliptic Curve $\mathcal{P}_E$, point $H \in_R \mathcal{P}_E$,
       message $m$, private key share $[d]_i$
Output: Signature $(r, s)$

  2.1 Draw random nonce $[k]_i, kG \leftarrow$ EcDkg$(G, H)$. Set $(x, y) \leftarrow kG$. Repeat until $x \neq 0$. Set $r \leftarrow x$.
  2.2 Compute $[k^{-1}]_i \leftarrow$ Invert$([k]_i)$.
  2.3 Hash message $e \leftarrow \texttt{Hash}(m)_{0:|n|}$.
  2.4 Compute $[s]_i \leftarrow [k^{-1}]_i \odot (e + r([d]_i)$.
  2.5 $s \leftarrow$ Recombine$([s]_i)$. Repeat 1) - 5) until $s \neq 0$.
  2.6 Return signature $(r, s)$.

Subprotocol Invert$([k]_i, G, H)$

  I.1 Draw random blind: $[r]_i, \_ \leftarrow$ EcDkg$(G, H)$
  I.2 Blind share: $[u]_i \leftarrow [k]_i \odot [r]_i$
  I.3 Recombine and invert: $u^{-1} \leftarrow$ Recombine$([u]_i)^{-1}$
  I.4 Return $[k^{-1}] \leftarrow u^{-1}[r]_i$
```

Protocol 2: The Threshold ECDSA signature scheme proposed by Ibrahim et al. [30] as exectued by a peer P_i.

3.3. *Threshold ECDSA*

The basic idea to prevent theft in *CoinParty* is to let users transfer funds to escrow addresses that are owned and controlled not by one but multiple mixing peers. After shuffling the funds, the mixing peers then need to collaborate to transfer funds back to the users which prevents malicious peers from cheating. To receive and spend funds from escrow addresses, we need two cryptographic primitives, i) a distributed key generation (DKG) scheme for ECDSA keys and ii) a threshold ECDSA signature scheme.

Principally, we could use the construction for generic SMC presented in the previous section to implement these primitives. Indeed, [32, 33] propose a generic framework for Secure Two-Party Computation and implement 1024 bit RSA signature generation in it. Although they use a representation as boolean circuit instead of an arithmetic circuit and RSA instead of ECDSA, the reported runtime of more than 15 h for the generation of one signature clearly shows the need for more efficient custom protocols.

We briefly present the DKG and signature schemes that are used as building blocks for *CoinParty*. For both schemes, the domain parameters $(q, (a, b), G, n)$ of an elliptic curve $\mathcal{P}_E$ are given: $n \in \mathbb{N}$ is the order of the generator $G \in \mathcal{P}_E$ that generates the elliptic curve $\mathcal{P}_E := \{(x, y) \in \mathbb{Z}_q^2 \mid y^2 = x^3 + ax + b\}$ with $a, b \in \mathbb{Z}_q$. In the context of Bitcoin, the parameters of the standardised elliptic curve `Secp256k1` are used.

Distributed Key Generation. We adapt the DKG scheme due to Gennaro et al. [23] to use an elliptic curve as the underlying cyclic group, referred to as EcDkg in the following. We briefly describe the high-level ideas and refer to Protocol 1 and [23] for the details. The following steps are executed by each of the m peers: Peer i (P_i) chooses a random secret s_i and verifiably shares this secret to the other peers $P_{j \neq i}$ (Steps 1.1 and 1.2). Any misbehaving peer is disqualified (Step 1.3). The private key d then corresponds to the sum of the secrets s_i of the remaining qualified peers Q, i.e., $d := \sum_{i \in Q} s_i$. Note that d is never explicitly computed by anyone. Instead, P_i computes its individual secret share $[d]_i$ of the private key d by summing up the shares received from the other peers $P_{j \neq i}$ (Step 1.4). To compute the public key $D = dG$ without revealing d, P_i broadcast its individual part of the public key $S_i := s_iG$ and computes $D = \sum_{j \in Q} S_j = \sum_{j \in Q} s_jG = (\sum_{i \in Q} s_i)G = dG$ using its own share and the shares received from $P_{j \neq i}$ (Steps 1.5 and 1.7). A second round of commitments and checks protects against malicious adversaries (Step 1.6).

As proved in [23], EcDkg is secure against $\tau < m/2$ semi-honest and malicious peers. It is important to note that this result depends on the availability of a reliable broadcast channel. In the Internet, one can usually only assume

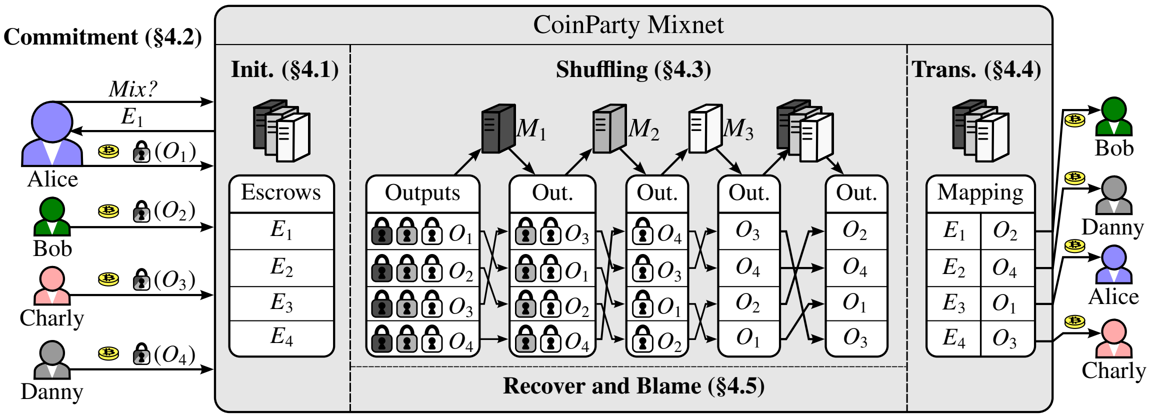


Figure 1: Overview of *CoinParty* with four participants and three mixing peers. Mixing peers initialize a mixing operation by creating a sufficient amount of escrow addresses (§4.1). The users commit the required funds to these escrow addresses and provide in an encrypted manner their fresh unlinkable output addresses (§4.2). In the shuffling phase (§4.3), mixing peers form a decryption mixnet to obliviously shuffle the provided output addresses. Finally, the funds are transferred from the escrow to the unlinkable output addresses (§4.4). If an error occurs, e.g., due to malicious peers, the mixnet recovers and determines the misbehaving peers (§4.5).

private point-to-point channels, e.g. realized using TLS in this work. Different protocols exist to implement reliable broadcast over such private channels, but these only achieve a security threshold of $\tau < m/3$. In our setting, the security of Gennaro's scheme for DKG [23] is thus reduced to $\tau < m/3$.

Threshold Signatures. To achieve threshold ECDSA signatures, we adopt the scheme by Ibrahim et al. [30] as detailed in Protocol 2. First, peers run the previously described EcDkg scheme (Step 2.1) and P_i interprets the output as a share of a random nonce k and the corresponding point kG on the elliptic curve $\mathcal{P}_E$. The x-coordinate of the point kG defines the first part r of the signature. Then, P_i inverts $[k]_i$ using the Invert protocol given in [30] (Step 2.2). As in standard ECDSA, P_i builds and truncates the hash of the message to be signed (Step 2.3). P_i computes and recombines the second part s of the signature (Steps 2.4 and 2.5).

Ibrahim et al. [30] show that their protocol is secure against $\tau < m/2$ semi-honest but only $\tau < m/4$ malicious peers. This is due to their employed Multiply primitive which is only secure against an $\tau < m/4$ adversary. Using the Multiply primitive described in §3.2 which provides security against $\tau < m/3$ malicious peers, the presented ThresholdEcdsa becomes secure against $\tau < m/3$ malicious peers.

4. Protocol and System Design

In this section, we present, *CoinParty*, a novel mixing service that fulfills our requirements stated in §1.1. The core idea of *CoinParty* is to realize the *insecure* centralized mixing model *securely* in a decentralized fashion via threshold cryptography: In the centralized model, users deliver their funds in escrow to the mixing service and are paid back at a later point with the funds of some random other user. The main difference introduced by *CoinParty* is a set of mixing peers that replace the centralized mixing service with a distributed mixing protocol, thereby obviating the need for a trusted third party and protecting against theft from the centralized mixing service itself. This approach allows us, for the first time, to combine the advantages of previous centralized and decentralized approaches in one system.

System Overview. Figure 1 gives an overview of one exemplary protocol run of a *CoinParty* mixnet with $m = 3$ mixing peers and $n = 4$ participating users. We briefly step through each of the five protocol phases and then present each phase in detail. The *initialization phase* (Fig. 1, §4.1) involves only the mixing peers and can take place at any time before an actual mixing operation. It is used to precompute expensive parts of the later phases, most importantly a set of escrow addresses $E_1, ..., E_n$. Escrow addresses are Bitcoin addresses that are under the shared control of the mixing peers. During the subsequent *commitment phase* (Fig. 1 §4.2), the mixnet accepts participants for the upcoming mixing operation. Notably, this is the only phase in which users need to interact with the mixnet. A user *Alice*, who wants to join a mixing operation, simply sends a request to any mixing peer and receives an individual escrow address, e.g., E_1 in Fig. 1, together with a set of parameters and conditions for the mixing. If *Alice* agrees to the conditions, she commits the required funds to the escrow address by simply issuing a standard Bitcoin transaction

Protocol Initialization

Input: Mixing parameters $\mathcal{P} := (n_{min}, n_{max}, v, t_{in}, \omega)$, generator $G \in \mathcal{P}_E$, $H \in_R \mathcal{P}_E$

Output: Escrow addresses $([d_1]_i, E_1), ..., ([d_{n_{max}}]_i, E_{n_{max}})$, partial signatures $(r_1, [s'_1]_i, [k_1^{-1}]_i), ..., (r_{n_{max}}, [s'_{n_{max}}]_i, [k_{n_{max}}^{-1}]_i)$.

3.1 Set up session keys:

- (a) Generate tuples $([sk_j]_i, K_j) \leftarrow \textsc{EcDkg}(G, H)$, $j = 1...m$.
- (b) Send $[sk_j]_i$ to $M_{j=1..m}$ and receive $[sk_i]_j$ from $M_{j=1...m}$.
- (c) $sk_i \leftarrow \textsc{Recombine}([sk_i])$.

3.2 Generate escrow addresses and precompute partial signatures:

- (a) Generate tuples $([d_j]_i, d_jG) \leftarrow \textsc{EcDkg}(G, H)$, $j = 1...n_{max}$. Transform public keys d_jG to Bitcoin addresses E_j.
- (b) Precompute $\textsc{TresholdEcdsa}$ for each escrow address $E_{j=1...n_{max}}$:
  - $[k_j]_i, k_jG \leftarrow \textsc{EcDkg}(G, H)$. $(x, y) \leftarrow kG$. Repeat until $x \neq 0$. Set $r \leftarrow x$.
  - Compute $[k_j^{-1}]_i \leftarrow \textsc{Invert}([k_j]_i)$.
  - Compute $[s'_j]_i \leftarrow r[d_j]_i \odot [k_j^{-1}]_i$.

Protocol 3: Details of the initialization protocol as executed by M_i. Mixing peers set up session keys which are secret-shared among each other. They then generate escrow addresses and precompute partial signatures.

to E_1. Alice also generates a fresh output address, e.g., O_1 in Fig. 1, and uses the public key of each mixing peer to create a layered encryption of O_1. After a certain time, the window for participation is closed and the *shuffling phase* (Fig. 1, §4.3) starts. Mixing peers one after the other decrypt one layer and secretly shuffle the outputs. At the end of the *shuffling phase*, mixing peers obtain a random shuffle of the output addresses $O_1, ..., O_n$ provided by the users during the *commitment phase*. Notably, the permutation of the addresses is unknown even to the mixing peers themselves, thus achieving unlinkability between the users and their output addresses. In the final *transaction phase* (Fig. 1, §4.4), the funds committed to the escrow addresses are transferred to the shuffled output addresses in n standard Bitcoin transactions. Transactions are signed with a threshold signature, which ensures that all transactions succeed and that malicious peers cannot steal funds or halt the transaction phase prematurely.

As we discuss in detail in §5, the *initialization*, *commitment*, and *transactions phases* succeed even in the presence of up to $\tau < m/3$ malicious mixing peers and any number of malicious users. However, just as in the other decentralized mixing approaches [40, 51, 52], a single malicious user may cause the *shuffling phase* to fail. In this case, a fifth phase, *recover and blame* (Fig. 1, §4.5), is entered which identifies and punishes the misbehaving users and transfers funds back to the participating users without shuffling. Notably, our design directly allows mixing peers to retain funds of malicious users thereby providing at least reactive protection against DoS attacks from malicious users. This is not possible in the previous decentralized approaches that are based on group transactions [40, 51, 52].

4.1. Initialization

The initialization phase is executed before a new mixing operation starts. It has two tasks: i) set up and share session key pairs and ii) generate escrow addresses and precompute partial signatures. This phase does not require any interaction with the future users participating in the mixing and can thus be executed at any time before the actual mixing operation in order to speed up the subsequent protocol phases.

We assume that mixing peers $M_1, ..., M_m$ are already connected to each other and have established private channels, e.g., using TLS. We do not explicitly consider how a mixnet would be bootstrapped in this work. In [40, 52], users bootstrap the mixnet among each other in an ad-hoc fashion, which is also easily possible in *CoinParty*. Another approach is to maintain public directories of mixing peers and mixnets.

The mixing peers have also agreed on a set of parameters $\mathcal{P} = (n_{min}, n_{max}, v, t_{in}, \omega)$ and an elliptic curve $\mathcal{P}_E$ prior to initilization. The values n_{min} and n_{max} are a lower and upper bound on the number of participants. The mixing value v denotes the amount of Bitcoins each user has to commit to the mixing by the deadline t_{in}. The last parameter, ω, denotes the time window allocated for the mixing, i.e., users receive their funds back at latest by $t_{in} + \omega$. As we show in §5.2.2, anonymity increases with the size of the mixing window ω.

Protocol Commitment

Input: Mixing parameters $\mathcal{P}$, escrow addresses $E_1, ..., E_{n_{max}}$.
Output: Escrow addresses $E_1, ..., E_n$ that have been committed to.

Bootstrap:

4.1 User U_j chooses any mixing peer M_i as entry peer and sends a request for participation.

4.2 M_i assigns E_j to U_j and announces the choice to $M_{k \neq i}$.

4.3 M_i sends to U_j: session ID S_{U_j}, parameters $\mathcal{P}$, escrow address E_j, identities $(M_1, ..., M_m)$, public keys $(K_1, ..., K_m)$.

4.4 U_j contacts $M_1, ..., M_m$, verifies $(K_1, ..., K_m)$, then presents S_{U_j} to check consistency of $\mathcal{P}$ and E_j.

Commit:

4.4 U_j generates a fresh unlinkable Bitcoin address O_j.

4.5 U_j encrypts O_j in layers, i.e., $[\![O_j]\!]_{K_1:K_m} := E_{K_1}(E_{K_2}(...E_{K_m}(O_j)))$ with the public keys $K_1, ..., K_m$ of the mixing peers.

4.6 U_j computes $[C_l^j] \leftarrow \textsc{Share}(\textsc{Hash}([\![O_j]\!]_{K_l:K_m}))$ for each layer $l = 1...m$ and $[C_{m+1}^j] \leftarrow \textsc{Share}(\textsc{Hash}(O_j))$.

4.7 U_j broadcasts $(S_{U_j}, [\![O_j]\!]_{K_1:K_m})$ to all $M_{i=1...m}$ and sends $(S_{U_j}, [C_1^j]_i, ..., [C_{m+1}^j]_i)$ to $M_{i=1...m}$.

4.8 U_j transfers ν Bitcoins to E_j in a transaction T_j.

4.9 Mixing peers verify that T_j has been accepted by the Bitcoin network.

Protocol 4: Details of the commitment protocol: A user U_j who wants to join the mixing operation first bootstraps to the mixing network, using any mixing peer M_i as entry point. After verifying parameters $\mathcal{P}$ and the assigned escrow address E_j, the user announces her encrypted output address O_j and secret shares checksums C^j. Finally, the user transfers the required amount *nu* of Bitcoins to the assigned escrow address E_j and thereby fully commits to the mixing operation.

Protocol 3 shows the detailed initialization protocol executed by each mixing peer M_i. First, the mixing peers generate one ECDSA key pair per peer that is used as a session key (Step 3.1). Note that only M_i learns the private key sk_i in clear, but all other mixing peers M_j hold a share $[sk_i]_j$ and know the public key K_i. When an error occurs in later protocol phases, the mixing peers use their shares of sk_i to check whether M_i misbehaved.

The second part of Protocol 3 is dedicated to generating escrow addresses and precomputing partial signatures on them (Step 3.2). First, the mixing peers generate n_{max} key pairs again using EcDkg and transform the public keys into Bitcoin addresses E_j according to the specifications of the Bitcoin protocol (Step 3.2a). These addresses are used as escrow addresses in the next phase, i.e., participants of the mixing need to transfer their funds to these addresses before the mixing starts. Since the private key is shared among the mixing peers, they can only collaboratively redeem funds from an escrow address via threshold signatures. For each escrow address E_j one threshold signature is precomputed partially (Step 3.2b). The signature can only be completed after the shuffling phase, when it is determined to which output address the escrowed funds should be transferred.

4.2. Commitment

After the mixing operation has been initialized (§4.1), the mixing network enters the commitment phase and starts to accept participants. The goal of this phase is to have at least n_{min} and at most n_{max} users commit the required ν Bitcoins to one of the previously computed escrow addresses within the time bound t_{in}. The commitment phase is the only phase that requires a user to interact with the mixing peers. A user only needs a standard Bitcoin client and a web browser to participate, which makes *CoinParty* accessible to a broad user base (cf. §5.6).

Protocol 4 shows the detailed steps of the commitment phase. To bootstrap to the mixing network, the user U_j chooses any mixing peer M_i as entry point (Step 4.1). Upon request, M_i assigns U_j an escrow address E_j and announces its decision to all other mixing peers (Step 4.2). Then, U_j is provided a session ID S_{U_j}, the parameters of the mixing $\mathcal{P}$, and an unused escrow address E_j chosen from the pool of addresses precomputed during initialization (Step 4.3). Since U_j could potentially have chosen a malicious mixing peer as entry point, it needs to verify the received information: U_j simply contacts and authenticates the other mixing peers $M_{k=1...m}$ by their public keys K_k and presents S_{U_j} upon which each mixing peer responds with $\mathcal{P}$ and E_j. If U_j receives ambiguous information (i.e.,

```
Protocol Shuffle
Input: Layered encryptions $\llbracket O_j \rrbracket_{K_1:K_m}$, $j = 1...n$, shared checksums $[C_l^j]$, $j = 1...n, l = 1...m+1$, Prng
Output: A shuffle $S = O_{\pi(1)}, ..., O_{\pi(n)}$ under random and unknown permutation $\pi$

  5.1 Repeat for each $M_{i=1...m}$
      (a) $M_i$ receives $S^i = \llbracket O_{\pi_{i-1} \circ ... \circ \pi_1(1)} \rrbracket_{K_i:K_m}, ..., \llbracket O_{\pi_{i-1} \circ ... \circ \pi_1(n)} \rrbracket_{K_i:K_m}$ and removes the outermost encryption $E_{K_i}$
      (b) $M_i$ applies a secret permutation $\pi_i$ and broadcasts $S^{i+1} = \llbracket O_{\pi_i \circ ... \circ \pi_1(1)} \rrbracket_{K_{i+1}:K_m}, ..., \llbracket O_{\pi_i \circ ... \circ \pi_1(n)} \rrbracket_{K_{i+1}:K_m}$
      (c) All mixing peers $M_{k \neq i}$ collaboratively check that $M_i$ decrypted correctly:
          • Compute checksum $C_{i+1} \leftarrow$ Recombine($[C_{i+1}]$) with $[C_{i+1}] \leftarrow [\sum_{j=1}^{n} C_{i+1}^j] = [\sum_{j=1}^{n}$ Hash($\llbracket O_j \rrbracket_{K_{i+1}:K_m}$)$]$.
          • Compute checksum $C'_{j+1} \leftarrow \sum_{j=1}^{n}$ Hash($\llbracket O_{\pi_i \circ ... \circ \pi_1(j)} \rrbracket_{K_{i+1}:K_m}$).
          • If $C_{i+1} \neq C'_{i+1}$, invoke BlameAndRecover($S^i, S^{i+1}, [C_{i+1}^1], ..., [C_{i+1}^n]$).
  5.2 All mixing peers
      (a) Sort output addresses lexicographically, i.e., $S' \leftarrow \pi_{lex}(S^{m+1})$.
      (b) Seed Prng with the checksum $C \leftarrow \sum_{i=1}^{m+1} C_i$ and draw random permutation $\pi_{rand}$ from PRNG.
      (c) Apply $\pi_{rand}$ to $S'$ and output $S \leftarrow O_{\pi(1)}, ..., O_{\pi(n)}$ with $\pi := \pi_{rand} \circ \pi_{lex} \circ \pi_m \circ ... \circ \pi_1$.
```

Protocol 5: Details of the shuffling protocol: Users provide layered encryptions of their output addresses and secret-shared checksums. Mixing peers form a mixnet to shuffle these addresses obliviously and check correctness after each step. A final fixed ordering combined with a common but random permutation ensures randomness of the shuffle even in the presence of malicious mixing peers.

deviating information from $\tau \geq m/3$ peers), U_j aborts the mixing and sends a report of the equivocation to the mixing network. Mixing peers can then collaboratively single out the misbehaving peers.

When the user U_j has validated $\mathcal{P}$ and E_j, U_j proceeds to commit her funds. First, U_j generates a fresh and unlinkable output address O_j and asymmetrically encrypts it in m layers (Steps 4.4 and 4.5). The encryption layers prevent mixing peers from learning which user submitted which output address and will be lifted one after the other in the subsequent shuffling phase. To prevent a malicious mixing peers from substituting U_j's output addresses with his own, e.g., to divert funds, U_j additionally provides verification information that allows the honest mixing peers to verify the integrity of the shuffled information (Steps 4.6 and 4.7). In particular, U_j hashes each encryption layer of her output address O_j, i.e., for each layer $l = 1...m$, U_j computes $C_l^j :=$ Hash($\llbracket O_j \rrbracket_{K_l:K_m}$). Since the checksum C_l^j would enable mixing peers to link U_j to the encrypted output address $\llbracket O_j \rrbracket_{K_l:K_m}$, U_j secret shares C_l^j over the mixing peers. Finally, U_j transfers the required v funds to E_j (Step 4.8). After this point, U_j is fully committed to the mixing operation and no further interaction with U_j is required. The mixing peers wait until U_j's commitment T_j has been accepted and confirmed by the Bitcoin network (Step 4.9).

4.3. Shuffle

The goal of the shuffling phase is to *obliviously* and *verifiably* permute the output addresses of the participating users. A shuffle is oblivious if no one, not even the mixing peers know the applied permutation, i.e., the link between a user and her output addresses. Verifiability means that the outputs are exactly the shuffled inputs, i.e., no inputs are deleted, substituted, added, or otherwisely modified. Decryption mixnets, as proposed by Chaum [17], are one solution that has been successfully applied, e.g., to anonymous communications [16, 20] and even to Bitcoin mixing [52]. Our shuffling protocol is also based on decryption mixnets, but takes a fundamentally different approach to ensure verifiability of the shuffling, which improves performance and enables deniability. We first briefly present our shuffling protocol and then compare our protocol to [16, 20, 52].

Protocol 5 shows the detailed shuffling protocol executed by the mixing peers. The input has already been provided by the users during the previous commitment phase (cf. §4.2). Now, each mixing peer M_i, one after the other, decrypts one layer of the encrypted output addresses (Step 5.1a), shuffles them (Step 5.1b), and broadcasts the results. The other mixing peers verify that M_i decrypted correctly using the corresponding checksums (Step 5.1c). Finally, after m rounds of decryption and shuffling, the output addresses are obtained in the clear. Note that after Step 5.1, any link between a user and her output address has been destroyed as long as at least two honest users have secretly and randomly

```
Protocol Transaction
Input: Shuffled addresses $O_{\pi(1)}, ..., O_{\pi(n)}$, escrow addresses $E_1, ..., E_n$, partial signatures $(r_1, [s'_1], [k_1^{-1}]), ..., (r_n, [s'_n], [k_n^{-1}])$
Output: Transactions $E_1 \overset{v}{\rightarrow} O_{\pi(1)}, ..., E_n \overset{v}{\rightarrow} O_{\pi(n)}$

  6.1 Mixing peers agree on a schedule $t_{in} \leq t_1 \leq t_2 \leq ... \leq t_n \leq t_{out}$.
  6.2 At time $t_j$, each $M_i$ computes
      (a) $e \leftarrow \textsc{Hash}(E_j \overset{v}{\rightarrow} O_{\pi(j)})$
      (b) $[s_j]_i \leftarrow e[k_j^{-1}]_i + [s'_j]_i$
      (c) $s_j \leftarrow \textsc{Recombine}([s_j]_i)$
  6.3 $M_i$ announces the transaction $T_j : E_j \overset{v}{\rightarrow} O_{\pi(j)}$ with signature $(r_j, s_j)$ to the Bitcoin network.
```

Protocol 6: Details of the transaction protocol: Mixing peers agree on a random schedule to announce transactions to thwart timing correlation attacks. Transactions are constructed on the fly and signed using our modified threshold ECDSA scheme based on [30].

shuffled the encrypted addresses in Step 5.1b. The mixing peers proceed to sort the addresses lexicographically (Step 5.2a). They then draw a common random permutation from a Prng (Step 5.2b) and obtain the final shuffling (Step 5.2c). The last step is important to guarantee the randomness of the shuffling as detailed in the following.

Our protocol improves over [16, 20, 52] in two ways: First, we use secret-shared checksums to efficiently validate the correctness of the shuffling on each stage (Step 5.1c). Using these checksums, mixing peers can *verify the shuffling at each stage* and *without involving the users*. The advantages of this approach compared to [16, 20, 52] are twofold: First, malicious mixing peers are detected immediately and thus less processing and communication resources are wasted on incorrect shufflings. Second, a user does not learn which other users participate in the shuffling, which increases anonymity (cf. §5.2) and enables deniability (cf. §5.3).

As a second improvement, we fix the following design flaw in the shuffling protocol of [52]: Similar to *CoinParty*, [52] relies on decryption mixnets to shuffle output addresses $O_1, ..., O_n$. After the first $m - 1$ shuffling steps in [52], the last mixing peer M_m receives from M_{m-1} the shuffling S^m, with only one layer of encryption left, i.e., E_{K_m}. Now, M_m can first lift E_{K_m} and then, with knowledge of the output addresses, apply a final permutation π_m. By choosing π_m accordingly, M_m can thus select which output address receives funds from which input address. This undermines the randomness of the final shuffling, violates our *correctness* requirement (cf. §1.1), and cannot even be detected by the other mixing peers in the design of [52]. Note that in [16, 20] this is prevented at the costs of a second encryption layer. We specifically introduce Step 5.2 in our protocol Shuffle to prevent this attack: Note that the final permutation π_{rand} is random, if the seed C is random. A single honest user U_j can ensure the randomness of C by choosing her output address O_j and thus the checksums C_i^j at random. We thus fix the vulnerability of the shuffling protocol of [52] while maintaining superior performance compared to [16, 20].

4.4. Transaction

In the transaction phase, the mixing peers transfer the escrowed funds back to the shuffled output addresses of the users. The details are given in Protocol 6. The mixing peers first agree on a random schedule for announcing mixing transactions to the Bitcoin network by using Prng seeded with the hash value of the shared seed used for the shuffling protocol (Step 6.1). Distributing transactions randomly over a sufficiently long mixing window thwarts timing correlation attacks and increases anonymity as we analyze and quantify in §5.2. To be accepted by the Bitcoin network, the transactions $E_j \rightarrow O_{\pi(j)}$ must be correctly signed with the private key d_j corresponding to E_j. Since, d_j is shared across the mixing peers, the standard ECDSA algorithm cannot be used. Instead, the mixing peers need to collaboratively sign the transaction to redeem the funds held in escrow at E_j. We use TresholdEcdsa (Prot. 2, §3.3) to realize such *threshold transactions*. Note that mixing peers only need to complete those steps of TresholdEcdsa in Step 6.2 that have not already been precomputed during the initialization phase. Compared to [30], our extensive precomputations considerably speed up signature generation during *CoinParty*'s online phase. Finally, the signed message is broadcasted to the Bitcoin network (Step 6.3).

Protocol BlameAndRecover

Input: Shuffles S^i and S^{i+1}, shared checksums $[C^j_{i+1}]$, j=1...n.
Output: Set of disqualified users $\bar{\mathcal{U}}$ and disqualified mixing peers $\bar{\mathcal{M}}$.

7.1 Identify malicious mixing peers:

(a) $sk_i \leftarrow \textsc{Recombine}([sk_i])$ and $S'^{i+1} \leftarrow D_{sk_i}(S^i) = [\![O_{\pi_{i-1}\circ\ldots\circ\pi_1(1)}]\!]_{K_{i+1}:K_m}, .., [\![O_{\pi_{i-1}\circ\ldots\circ\pi_1(n)}]\!]_{K_{i+1}:K_m}$.

(b) If $S'^{i+1} \not\equiv S^{i+1}$ set $\bar{\mathcal{M}} \leftarrow \bar{\mathcal{M}} \cup \{M_i\}$ and resume Shuffle with M_{i+1} on S'^{i+1}. Proceed otherwise.

7.2 Identify malicious users:

(a) Compute checksums $C^j_{i+1} \leftarrow \textsc{Recombine}([C^j_{i+1}])$. Let $C_{i+1} \leftarrow \{C^1_{i+1}, ..., C^n_{i+1}\}$ be the set of these values.

(b) Compute checksums $C'^{\pi(j)}_{i+1} \leftarrow \textsc{Hash}([\![O_{\pi_i\circ\ldots\circ\pi_1(j)}]\!]_{K_{i+1}:K_m})$, $j = 1...n$. Let $C'_{i+1} \leftarrow \{C'^{\pi(1)}_{i+1}, ..., C'^{\pi(n)}_{i+1}\}$.

(c) Mixing peers compute $\bar{C}_{i+1} \leftarrow C_{i+1} \backslash (C_{i+1} \cap C'_{i+1})$. Let $\bar{\mathcal{U}}$ the set of users who contributed a checksum to $\bar{C}_{i+1}$.

(d) Invoke ThresholdECDSA issue transaction $T_j \xrightarrow{v} I_j$ for all users $U_j \notin \bar{\mathcal{U}}$ and abort the mixing.

Protocol 7: Details of the blaming and recovery protocol: Mixing peers first determine which mixing peers or users tampered with the shuffle and then transfer all funds back to the honest users.

4.5. Blame and Recover

The fifth phase, recover and blame, is entered whenever the shuffling phase fails and has two tasks, i) to determine which mixing peer and user caused an error and ii) to either fix the error or to transfer the escrowed funds back to all honest users. Protocol 7 shows the protocol in full detail.

By design of Shuffle (Prot. 5), only two possible sources for errors exist: Either mixing peer M_i did not correctly decrypt the i^{th} encryption layer and tampered with the shuffle or a malicious user announced an inconsistent layered encryption or inconsistent verification information, e.g., to mount a DoS attack against the mixing operation.

We first check whether a mixing peer misbehaved since we can recover from this without aborting the shuffle (Step 7.1). The mixing peers recombine M_i's private session key sk_i, which M_i has previously shared during the initialization phase. Using sk_i, they reproduce the decryption step in Shuffle from which they obtain a correct shuffle S'^{i+1} (Step 7.1a). If S^{i+1} and S'^{i+1} are not equal under permutation (denoted $\not\equiv$), M_i has tampered with the shuffle (Step 7.1b). In this case, M_i is simply skipped and M_{i+1} resumes to shuffle on S'^{i+1}. Note that this step does not break unlinkability of the shuffle, since two honest mixes are enough to ensure that the final shuffle is random and secret.

The second case, where M_i decrypted correctly but a user misbehaved, is handled in Step 7.2. Any misbehaving user is identified by recombining the set of individual checksums C_{i+1} provided by the users in the initilization phase (Step 7.2a) and checking them against the set of checksums C'_{i+1} obtained from hashing the encrypted output addresses in the correct shuffle S^{i+1} (Step 7.2b). Note that $C_{i+1} \cap C'_{i+1}$ is the set of consistent checksums and that user U_j provided $C^j_{i+1} \in C_{i+1}$. Thus, we can identify exactly the set of users $\bar{\mathcal{U}}$ who misbehaved and provided inconsistent checksums (Step 7.2c). Since reconstructing the individual checksums breaks unlinkability, we need to gracefully abort the shuffle (Step 7.2d). To this purpose, mixing peers refund all honest users by invoking ThresholdECDSA (Fig. 2, §3.3). Note that ThresholdECDSA is guaranteed to succeed even if up to $\tau < m/3$ mixing peers are actively corrupted, as we show in §5.1. Thus, it is assured that no funds of honest users are stolen or lost.

We propose that mixing peers keep the escrowed funds of the non-honest users in $\bar{\mathcal{U}}$ as a punishment for malicious users who try to sabotage the shuffle by providing inconsistent inputs. For larger mixing amounts v, we argue that this mechanisms then renders such DoS attacks economically infeasible. Interestingly, the same mechanism could be applied to punish mixing peers who tamper with the shuffle additionally to skipping them: Mixing peers could each transfer a certain amount of Bitcoins to a collaboratively controlled address as a financial commitment to the mixing operation. If then malicious behavior of a peer M_i is identified in Step 7.1b, the other mixing peers retain M_i's commitment as a punishment.

5. Discussion of System Properties

In this Section, we discuss the *CoinParty* design with regards to the requirements for an ideal mixing service stated in §1.1. We show that *CoinParty* satisifies correctness (§5.1), anonymity (5.2), and deniability (§5.3) requirements even in the presence of a malicious adversary. A comprehensive performance evaluation of *CoinParty* is presented in §5.4. Finally, §5.5 discusses cost efficiency and §5.6 applicability of *CoinParty*.

5.1. Correctness

Correctness of *CoinParty* mostly depends on the security of the underlying cryptographic primitives as presented in §3.2 and §3.3. In the following, we show that all protocol phases in *CoinParty* are secure according to the definition of security in the malicious adversary model given in §3.1. In other words, we show that honest parties succeed to compute the correct output of all protocol phases even in the presence of $\tau < m/3$ malicious mixing and n malicious users that may arbitrarily interfere with the correct protocol execution.

5.1.1. Initilization Phase

Users do not participate in the initialization phase, thus we only need to discuss security against malicious mixing peers. In the first part of Protocol 3, the mixing peers set up session keys. We need to make sure that i) each mixing peer M_i receives a well-formed key pair (sk_i, K_i) and ii) mixing peers hold consistent shares $[sk_i]_j$ of the private key. It follows directly from the security of the primitives Share, Recombine, and EcDkg that these conditions hold. In the second part of Protocol 3, the mixing peers first generate escrow addresses and then precompute corresponding partial threshold signatures. Security of EcDkg guarantees that mixing peers succeed to generate well-formed escrow addresses and transforming them into Bitcoin addresses is only a local operation. The correctness of the precomputation of the partial signatures is ensured due to the security of the primitives EcDkg, Add, and Multiply. Since all used primitives are secure against at least $\tau < m/3$ adversaries, we conclude that Initialization is secure against $\tau < m/3$ malicious mixing peers.

5.1.2. Commitment Phase

In the commitment phase, users and mixing peers need to interact. Entities among both parties may be compromised and try to sabotage the mixing or tamper with it. We now show that Commitment is secure against any number of malicious users and $\tau < m/2$ malicious mixing peers.

Malicious users. In the commitment phase, users need to provide their encrypted output address, verification information and, most importantly, commit the required mixing amount to the escrow addresses. First, it is easy to see that mixing peers can at any point and without any consequences stop interacting with a malicious user that fails to provide this required information or does not cooperate otherwisely. A user could DoS our system by repeatedly requesting participation thereby depleting the pool of escrow addresses precomputed by the mixing peers. However, this can be thwarted using standard puzzle mechanisms. Second, none of the information sent by a user is used in this phase. Thus, the correctness of this information is irrelevant in this phase. Finally, the correctness of the malicious users financial commitment only depends on the correctness of the transaction to the assigned escrow address, which is ensured by the Bitcoin network itself independently of our system. Since malicious users gain no additional attack vector by grouping together, we conclude that the commitment phase is secure against any number of malicious users.

Malicious mixing peers. An honest user U_j may have chosen a compromised mixing peer M_i as entry peer to the mixnet. The attacker can then try to mount different attacks against U_j. First, the attacker may send U_j the identities $(M'_1, ..., M'_m)$ and public keys $(K'_1, ..., K'_m)$ of a mixnet that is under complete control of the attacker. Establishing trust in a set of identities, i.e., the mixnet $(M'_1, ..., M'_m)$ in our case, is an orthogonal problem for which well-known solutions exist ranging from certificate infrastructures to reputation systems and decentralized trust networks such as PGP. Second, an attacker that has compromised the entry peer M_i, may substitute the chosen escrow address E_j with another address E'_j that the attacker owns. However, U_j validates the mixing parameters and the received escrow address with the other mixing peers. Thus, if the majority of mixing peers is honest, a forged escrow address or set of parameters will be detected. Finally, a malicious M_i can prevent any user from joining the mixing by sending a broken session ID S_{U_j}, which is not recognized by the other mixing peers. User U_j can just choose another mixing peer as entry and will eventually be correctly bootstrapped to the mixing network and the upcoming mixing operation. We conclude that Commitment is secure if the majority if mixing peers is honest, i.e., $\tau < m/2$.

5.1.3. Shuffle Phase

In the following, we first show that the shuffling completes and its integrity and randomness are guaranteed in the presence of $\tau < m/3$ malicious mixing peers. We then analyze the attack vectors for malicious users. Note that unlinkability of the shuffling is proved in §5.2.

Malicious Mixing Peers. Shuffling in the presence of malicious mixing peers is the most complex case in *CoinParty* and we split the security proof into three parts, first showing *integrity* of the shuffling, then its *randomness* and finally showing that malicious peers cannot prevent honest mixing peers from *completing* the shuffling.

Integrity. We define *integrity* analogous to [16]: Either exactly the given output addresses are contained in the final shuffling, or the honest mixing peers are informed that some user's output address has been tampered with. Note that although our shuffling phase is inspired by [16, 20, 52], our method of verifying the integrity of the shuffling is fundamentally different and thus requires a dedicated proof of correctness.

Any malicious mixing peer M_i can substitute the outputs in the shuffling with her own output addresses O'_i by announcing $S'^{i+1} = [\![O'_1]\!]_{K_{i+1}:K_m}, ..., [\![O'_m]\!]_{K_{i+1}:K_m}$. However, honest mixing peers can verify the integrity of the shuffling at each stage $i + 1$ and will detect this unless the attacker M_i finds $O'_1, ..., O'_m$ such that the checksums C_l and $C_{l'}$ are equal for all layers $l, l' \geq i + 1$. For all but the last mixing peer, this is clearly infeasible since M_i, $i < m$, needs to broadcast and thereby commit to the shuffle S^{i+1} before even learning the checksums of the subsequent layers $C_l, l > i + 1$. Note that all checksums $C_{i+2}, ..., C_{m+1}$ are random if at least one user is honest and supplies random individual checksums $C^j_{i+2}, ..., C^j_{m+1}$. Then, the probability p_{guess} of M_i correctly guessing all remaining checksums $C_{i+2}, ..., C_{m+1}$ is negligible, i.e., $p_{guess} \sim 1/2^{256(m-i)}$ for a 256 bit hash function. A scenario where all users are compromised by the attacker clearly does not make sense as the attacker would attack himself only.

The last mixing peer M_m, however, removes the last layer of encryptions and learns the output addresses in clear. M_m can thus derive the checksum C_{m+1} *before* announcing the shuffle S^{m+1}. To steal the mixed funds, M_m must thus only find suitable output addresses $O'_1, ..., O'_n$ such that $\sum^m_{j=1} \textsc{Hash}(O'_j) = \sum^m_{j=1} \textsc{Hash}(O_{\pi(j)})$. Since the attacker can generate an arbitrary amount of addresses, this corresponds to solving a high density *Random Modular Subset Sum* (RMSS) problem. We show in Appendix A that large problem instances of RMSS as involved in *CoinParty* are not solvable in reasonable time and thus the attack is thwarted simply by limiting the time for Step 5.1b in Shuffle. Note that we can also prevent the whole attack on the protocol level by introducing random nonces into the hashes, i.e., users share $[\textsc{Hash}(O_j|n_j)]$. Since the nonces n_j link in- and output addresses, nonces need to be encrypted as well. By shuffling and decrypting the nonces only *after* mixing peer M_m has committed to the shuffle S^{m+1}, M_m cannot predict the checksum C and hence cannot mount the RMSS attack anymore.

Randomness. Due to the sorting in Step 5.2a of Shuffle, the randomness of the final shuffle depends entirely on the randomness of the final permutation π_{rand} drawn from the Prng. The final permutation π_{rand} is random if the seed C, the sum of the checksums C_i at each shuffling stage i, are random. This is the case if at least one user U_j is honest and supplies random individual checksums $C^j_1, ..., C^j_{m+1}$.

Termination. Any malicious mixing peer M_i may refuse to decrypt the corresponding encryption layer i or announce an incorrect shuffle S^{i+1} to stall or halt Shuffle. We recover from this case by skipping the malicious M_i by entering BlameAndRecover after a timeout.

We conclude that no malicious mixing peer is able to violate the integrity of the shuffle, predict or bias its randomness, or prevent honest mixing peers from executing it correctly. The security of this phase against malicious mixes is only limited by the security of Recombine and BlameAndRecover, which are secure against $\tau < m/3$ malicious mixing peers.

Malicious users. Users are not involved in Shuffle. Shuffle only operates on their input, i.e., the layered encryption of the output addresses and secret shares of the corresponding checksums. Thus, we only have to consider the case where this information is inconsistent. It is easy to see that malicious users have no incentive in announcing a wrong or broken output address, since this would only result in the loss of their own funds and have no effect at all on other users or the mixing peers. However, malicious users can share inconsistent checksums in order to cause the verification step of Shuffle to fail. In this case, Shuffle enters BlameAndRecover which identifies the misbehaving users, refunds honest users, and aborts the mixing operation. Although we can identify malicious users, we cannot proactively prevent deliberate DoS attacks.

Note that related work [16, 20, 52] is also vulnerable to such DoS attacks by single malicious users. Shuffle protocols in related work even fail when benign users randomly fail without any malicious intent. This is due to involving the users in the verification process of the shuffle. Our shuffling protocol proposes a verification scheme that does not involve users and is thus completely robust against halting users and random failures. More importantly, our design allows to punish malicious users by retaining their funds, thereby disincentivizing deliberate DoS attacks economically.

We conclude that SHUFFLE is fully secure against any number of passively corrupted or halting users. While malicious users can cause SHUFFLE to abort, BLAMEANDRECOVER realizes a graceful failover and effectively disincentivizes any malicious behavior among the users.

5.1.4. Transaction Phase

Users are not involved in TRANSACTION and the only user input that is operated on are the users' output addresses. Since an incorrect output address affects only the user that announced it, we do not need to consider malicious users in this phase. Security against malicious mixing peers is then easy to show. First, agreeing on a random schedule simply requires an honest majority. Second, completing the partial signatures is secure because ADD and RECOMBINE are secure against $\tau < m/3$ malicious mixing peers. Finally, the random schedule for announcing transactions is adhered to, since honest peers will refuse to sign transaction T_j before time t_j. Thus, no malicious mixing peer can announce it to the Bitcoin network prematurely and thereby reduce the anonymity set of the mixing. We conclude that TRANSACTION is secure against $\tau < m/3$ malicious mixing peers and any number of malicious users.

5.1.5. Blame and Recover Phase

BLAMEANDRECOVER is invoked whenever verification of a shuffling stage S^{i+1} fails. We have already shown in §4.5 how either the malicious mixing peer or at least one malicious user is identified. It remains to show that we can then either skip the malicious mixing peer or that honest users are refunded.

Both cases are easy to show. In the first case, all mixing peers know the last correct shuffle S^i and can correctly recover M_i's private key sk_i due to the security of SHARE and RECOMBINE. Thus, each mixing peer can decrypt layer i and obtain a correct shuffle S'^{i+1} to resume SHUFFLE just as if M_i had behaved honestly. Note that, for $\tau < m/3$, the session keys of at least two honest mixing peers remain secret. Thus, BLAMEANDRECOVER does not undermine the correctness of SHUFFLE. In the second case, where the shuffling is aborted due to inconsistent input from a malicious user, the mixing peers simply invoke TRANSACTION to refund honest users, which we have already proven secure in the previous section. We conclude that BLAMEANDRECOVER is secure against $\tau < m/3$ malicious mixing peers and any number of malicious users.

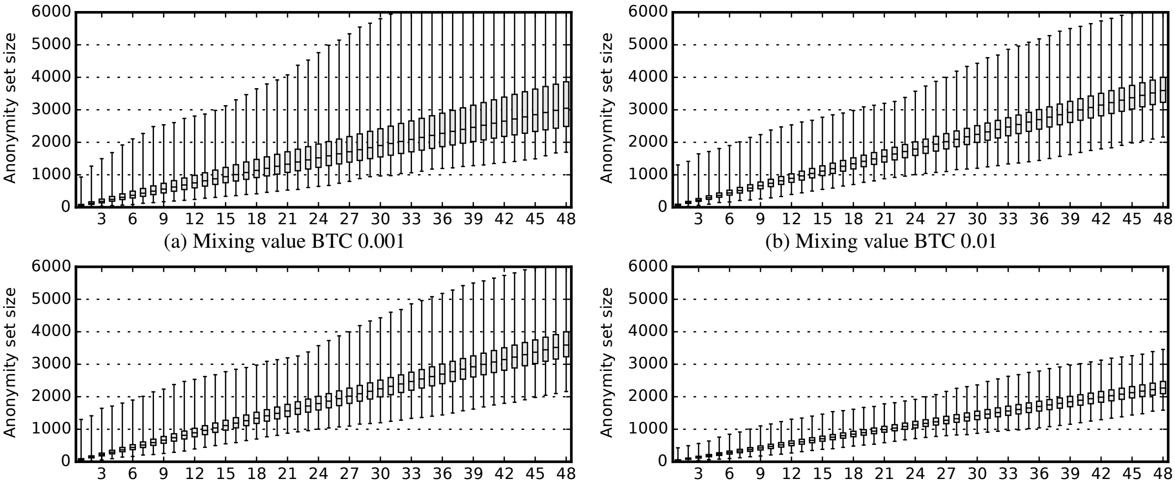
5.2. Anonymity

In the previous section, we showed that honest mixing peers succeed to *correctly* execute the *CoinParty* mixing protocol even in the presence of other malicious mixing peers or users. In this section, we show that *CoinParty* achieves the equally important *anonymity* requirement. We first show that users are anonymous even against any observer and even against the mixing peers themselves (§5.2.1). We then quantify the achieved level of anonymity by analyzing Bitcoin's blockchain and give concrete recommendations on how to maximize it (§5.2.2).

5.2.1. Unlinkability

The unlinkability of a *CoinParty* mixing depends only on the shuffling phase (§4.3). Furthermore, if there is an error in this phase, the protocol enters BLAMEANDRECOVER and funds are restored to the inputs and the tainted output addresses are discarded. Thus, errors during the shuffling phase have no impact on unlinkability and we only need to consider correct runs in our anonymity analysis. The proof for unlinkability is then very similar to [16].

The basic argument for unlinkability between users and their output addresses is that by using an IND-CCA2 and length regular encryption scheme E, the ciphertexts $[\![O_{j=1\ldots m}]\!]_{K_i:K_m}$ at each shuffling stage i are *indistinguishable*. Concretely, for any honest mixing peer M_i that receives S^i, secretly shuffles it and broadcasts S^{i+1}, the attacker cannot decide which ciphertext $[\![O_{j'=1\ldots m}]\!]_{K_{i+1}:K_m} \in S^{i+1}$ corresponds to the decryption of which ciphertext $[\![O_{j=1\ldots m}]\!]_{K_i:K_m} \in S^i$. Thus, the attacker cannot link ciphertexts in S^i to those in S^{i+1}. In other words, the attacker cannot observe the permutation π_i applied by an honest mixing peer M_i. Furthermore, the decryption mixnet ensures participation of all mixing peers in the shuffling, i.e., in particular that M_i can only be skipped when the honest majority of peers decides to do so.



(a) Mixing value BTC 0.001

(b) Mixing value BTC 0.01

(c) Mixing value BTC 0.1

(d) Mixing value BTC 1.0

Figure 2: Size of the anonymity set for mixing values $v = 0.001, 0.01, 0.1, 1.0$ BTC in plots (a)-(d), respectively, and different mixing windows $w = 1, ..., 48$ hours (x-axis) obtained through analysis of all transactions during June 2014 to June 2015. The boxes show the 2^{nd} and 3^{rd} quartiles; the whiskers show the range of the data, i.e., minimum and maximum.

So far, we have shown that the mixnet successfully unlinks output addresses from the users who submitted them. It remains to show that the individual checksums $C^j_{i=1...m}$ are not used in a way that links the output address O_j back to user U_j. Here, the key argument is that mixing peers only hold secret shares $[C^j_i]$ of these checksums, which are never recombined individually during a successful run of Shuffle. Instead only the sum of the checksums $\sum_{j=1}^{n} C^j_i$ is reconstructed at each shuffling stage i, which provides no information about any individual checksum C^j_i. Importantly, no set of $\tau < m/3$ malicious mixing peers can reconstruct individual checksums on their own due to the security of Share and Recombine for any recombination threshold $t \geq m/3$. Note that this argument does not hold anymore when BlameAndRecover is invoked and malicious users are identified by reconstructing their individual checksums. Because this clearly breaks unlinkability, we need to abort the mixing in this case.

We conclude that unlinkability is *guaranteed* against $\tau < m/3$ malicious mixing peers, while only two honest mixing peers M_i and $M_{i'}$ are required to *create* unlinkability: M_i secretly applies a random permutation π_i and $M_{i'}$ applies $\pi_{i'}$. Then π_i ensures unlinkability against $M_{i'}$ and all other mixing peers and vice versa.

5.2.2. Anonymity Level

So far, we have shown that *CoinParty*'s shuffling is correct and random (§5.1.2) as well as unlinkable (§5.2.1). However, any attacker may always try to guess the mapping between a user and her output address and it remains to analyze his chances at succeeding. The set of addresses among which the attacker has to guess is the *anonymity set* and we refer to its size as the achieved *anonymity level*. A bigger anonymity set leads to a smaller probability of a correct guess and hence more anonymity. In the following, we characterize the anonymity sets that *CoinParty* guarantees to honest users against i) any passive observer of the blockchain, ii) other users participating in the mixing, and iii) the mixing peers.

Passive observers. Since the Bitcoin blockchain is publicly available, anyone can pose as an passive observer of a *CoinParty* mixing. Analysing the anonymity against this kind of attacker, the key insight is that our threshold transactions are indistinguishable from standard Bitcoin transactions. Thus, a passive observer can only identify *CoinParty*'s mixing transactions by i) their correlation in time and ii) the reoccurring output value of v Bitcoins, i.e., the mixing amount. Thus, a user's anonymity set against this attacker comprises all output addresses of the transactions in the blockchain that can be tied to a mixing in this way. In the following, we describe this anonymity set and quantify its size in detail.

Generally, *CoinParty* produces mixing transaction chains of length two: First, user U_j commits at least v funds to the escrow address E_j during $[t_{open}, t_{in}]$ in one transaction $I_j \xrightarrow{v} E_j$, where t_{open} is the possibly publicly known time when the mixnet started accepting participants. In the transaction phase $[t_{in}, t_{in} + \omega]$, v Bitcoins are transferred from

E_j to another participant's output address $O_{\pi(j)}$ in a second transaction $E_j \xrightarrow{v} O_{\pi(j)}$. Note that the mixing amount v is known to the users and we thus assume it is also known to the attacker. The question is whether this transaction pattern is unique enough to distinguish mixing from non-mixing transactions in the blockchain.

Let us first assume that this is the case in order to establish a *lower bound* for the anonymity level. The attacker can then exactly identify n mixing transaction chains of length two produced by *CoinParty* among all other transactions in the blockchain during the time frame $[t_{open}, t_{in} + \omega]$. The attacker however cannot distinguish between the n transactions belonging to the same mixing operations. Thus, the anonymity level against passive observers is thus at least n, or even $k \cdot n$ if there are k contemptuous mixing operations with n participants each. The latter $k \cdot n$ bound is due to the property that an observer cannot distinguish with which mixing network a user interacts, termed *mix indistinguishability* in [15].

We now show that a passive observer cannot uniquely identify *CoinParty*'s transaction pattern, i.e., that the established lower bound is much too pessimistic. Note that users can vary the transaction amount in the commitment phase by transferring arbitrary amounts $v' \geq v$ to E_j and just receive the leftovers $v' - v$ after the mixing on a fresh change address. The commitment transaction thereby becomes indistinguishable from all other transactions in $[t_{open}, t_{in}]$ with an amount $\geq v$. However, we cannot apply the same trick to hide the tell-tale transactions of v Bitcoins from the escrow addresses E_j to the shuffled output addresses $O_{\pi(j)}$ during the transaction phase $[t_{in}, t_{in} + \omega]$: If different amounts of Bitcoins were to be mixed, the mixing peers would need to know which output address, i.e. which user, should receive which amount. This would require linking users and their output addresses, which the mixing peers cannot due to the unlinkability of the performed shuffling. The mixing amount v thus hints a passive observer at which transactions could be mixing transactions. The anonymity set thus comprises all transactions with an output value of v BTC during the mixing window. We analyze all transactions in the blockchain between June 2014 and June 2015 in order to determine how to sensibly choose v. We observe that $1, 0.1, 0.01$, and 0.0001 BTC are popular output values, e.g., there are 120 318 transactions for 0.01 BTC in June 2014 alone. These values are recommendable choices for v that promise high anonymity levels.

Even when using popular values as mixing amounts, releasing all mixing transactions at the same time makes them easily distinguishable from non-mixing transactions by their strong correlation in time. Thus, the length of mixing window ω over which transactions are released to the Bitcoin network has to be chosen reasonably in order to hide *CoinParty*'s mixing transactions among normal Bitcoin transactions. We again analyze the blockchain to quantify how much mixing windows ω of different length increase the anonymity level. We move a sliding window of 1 to 48 hours over the blockchain and count for each window position the number of transactions that contain the respective amount v as output. This number is the achieved anonymity level at that particular point in time. Figure 2 plots the 2^{nd} and 3^{rd} quartile (boxes) as well as the minimum and maximum (whiskers) anonymity levels for mixing windows ω of 1 up to 48 hours over all possible points in time during June 2014 to June 2015 for the four recommended mixing values v of $1, 0.1, 0.01$, and 0.001 BTC.

We observe that increasing the size of the mixing window ω greatly increases the achieved anonymity levels. Already a mixing window of 12 hours provides for all considered amounts v an anonymity level that is increased by 500 to 1 000 transactions on average compared to instant transactions. Importantly, these anonymity levels are achieved *in addition* to the base-line established above, i.e. the anonymity level $k \cdot n, k \geq 1$ that any mixing operation with n users always achieves.

Users. Users are different from passive observers since they actively participate in the mixing and may have additional knowledge that could de-anonymize other users. We start with the simple case where all users are honest. Then, anonymity of one user against the other users is practically the same as for any passive observer, since users receive no information about which other users participate in the shuffling. We achieve this by introducing dedicated mixing peers and a new shuffling verification mechanism that does not require involvement of any user (§4.3).

The case is different when an attacker compromises c of the n users, e.g., via a sybil attack. The attacker can then distinguish a mixing transaction from the other non-mixing transactions in the blockchain, if the input addresses of a compromised user is mapped to an honest user's output address and vice versa. In this way, the attacker can tie one transaction to the mixing with a probability $p_{tie} = 1 - (1 - c/n)^2$ and thereby reduce the anonymity level for the honest user in this transaction to the baseline n. A plot of p_{tie} versus c/n is included in Appendix B, e.g., showing that the attacker needs to compromise only $c/n = 50\%$ of the users to identify 75% of the mixing's transactions. Yet, the individual transactions of one mixing operation are still indistinguishable from each other.

We now assume the attacker has thereby tied the input address I_i of an honest user to the mixing and now tries to guess the corresponding output address O_i. This leads to the overall success probability for the attacker to guess the right output address O_i of $p_{succ} = p_{tie}/(n-c) + (1-p_{tie})/(n-c+N')$ with $N' := N/(1-p_{tie})$ and N the additional anonymity level achieved with mixing window ω and mixing amount v as quantified in Figure 2. In related work, this success probability is $p'_{succ} = 1/(n-c)$. Our scheme thus provides equal anonymity against malicious users for the baseline case, i.e., $N = 0$ and is significantly better for $N > 0$. To give an example, for $n = 100$ participants of which $c/n = 10\%$ are compromised, the attacker guesses correctly with $p'_{succ} = 0.011$ for related work and only $p_{succ} = 0.0027$ for our scheme with $N \approx 900$ (e.g., for a mixing amount of 0.01 or 0.1 BTC and $\omega \geq 12$ hours according to Figure 2). This is nearly one order of magnitude smaller.

We note that such sybil attacks seem endemic to Bitcoin mixing services (cf. §5.5): The attacker uses the mixing service to generate untraceable Bitcoin addresses which the attacker then uses as input addresses to mount a sybil attack against another mixing. Since this attack is passive, honest users or mixing peers cannot recognize it, e.g., in order to punish the attacker as we propose for malicious users that mount DoS attacks. An obvious solution is to make such attacks expensive by charging users for participation through mixing or other fees [7]. This solution is easily applicable in *CoinParty* but results in a trade-off with our *cost efficiency* requirement. We are not aware of any related work that solves this problem by design.

Mixing peers. Mixing peers present the third case, different from passive observers and users, since they inevitably learn which output addresses are involved in the mixing operation. However, since mixing peers do not learn which output address belongs to which user as proved in §5.2.1, the anonymity level against mixing peers is equal to the number of participants in the mixing n, i.e., the same as for [52, 60] and better than [15, 40]. We are not aware of any related work that achieves a higher anonymity level against the mixing service itself than the number of users n that participate in one mixing operation.

We conclude our anonymity analysis with a remark on the mixing delay ω imposed by *CoinParty*. On a first sight, the significantly larger mixing delay when compared to other distributed mixing services may seem disadvantageous. However, the contrary is really the case: The ability of *CoinParty* to distribute mixing transactions over a longer period of time is the main reason for the orders of magnitude higher anonymity sets compared to other decentralized mixing services [40, 51, 52, 60]. The atomic group transaction pattern used in these works links all mixing transaction together and limits the achieved anonymity set to the number of participating users. Although *CoinParty* would achieve similar small delays if a window $\omega = 0$ is used, we emphasize that a larger mixing delay must be tolerated if strong anonymity is desired. This observation is consistent with the results in other application areas of decryption mixnets, e.g., for the anonymous communication network Tor or anonymous remailers.

5.3. Deniability

We analyze *CoinParty* w.r.t. our third requirement for ideal mixing, deniability. For a user U_j, the simplest form of plausibly denying participation in a mixing is denying the ownership of the input address I_j used during the commitment. However, recent work on identifying ownership of addresses and even real identities of Bitcoins users render this option questionable. Indeed, these results are the main motivation for the development of any mixing service. Thus, instead we need to analyze whether a user can plausibly deny that one of her addresses was part of a mixing operation. As before, we split our analysis according to the three different attacker types, i) passive observer, ii) other users, and iii) mixing peers.

Passive observers. *CoinParty* does not emit any cryptographic evidence proving to outsiders that a user participated in a shuffling unlike, e.g., the approach proposed in [56] which leaves irrefutable evidence in a public append-only log, or [40, 51, 52], which are based on group transactions that are potentially identifiable as mixing transactions. Furthermore, to any passive observer *CoinParty*'s mixing transactions are indistinguishable from other transactions in the blockchain as explained in §5.2.2. Thus, we argue that if there are at any point many more non-mixing than mixing transactions in the Bitcoin network, a user can plausibly deny having participated in a mixing. Our analysis of the blockchain in the previous section shows that there are indeed many non-mixing transactions of the same form as those issued by *CoinParty*, if a mixing window of sufficient size and popular mixing values are used.

Users. We have previously shown in §5.2.2 that an adversary who corrupts c other users can tie a fraction of $p_{tie} = 1-(1-c/n)^2$ of the transactions to the mixing and present corresponding evidence. On the other side, an honest user is not tied to the mixing with probability $1 - p_{tie} = (1-c/n)^2$, in which case the user can deny her participation.

We refer to Appendix B for a plot of c/n versus p_{tie}. Note that Shuffle (cf. §4.3) ensures that the shuffle is random and even the last mixing peer M_m cannot control the randomness unlike the approach in [52]. Thus, an attacker cannot mount a targeted attack, i.e., each honest user is equally likely to evade the attack with probability $1 - p_{tie}$.

Mixing peers. *CoinParty* does not achieve deniability against mixing peers. Mixing peers learn which in- and output addresses participated in the mixing during the shuffling phase and may present U_j's commitment $I_j \rightarrow E_j$ as evidence. Potentially, deniability against the mixes could be achieved, if mixes blindly signed the mixing transactions. Cryptographic primitives for blind signatures exist, but we consider this future work.

5.4. Performance Evaluation

In this section, we show that *CoinParty* fulfills the *scalability* requirement stated in §1.1. To this end, we present a quantitative evaluation of the runtime and communication overhead of *CoinParty* in the initilization, shuffle, and transaction phases. We measure these phases separably since their individual functionality might be interesting to other areas and applications, e.g., protocol Transaction could be used to secure Bitcoin wallets [14, 24]. We do not evaluate in detail the commitment and blame and recover phases. The commitment phase has only negligible overhead for mixing peers, since they each send at most one fixed message to a new user and one unreliable broadcast within the mixnet. The blame-and-recover phase has almost the same overhead as the transaction phase. Compared to Transaction, it requires at most two additional invocations of Recombine which are negligible compared to the overhead of Transaction. Furthermore, the overhead for users consists of only a small constant amount of ECDSA signatures and ECIES encryptions linear in the number m of mixing peers. This overhead is in the order of a few seconds and we thus concentrate on the overhead imposed on mixing peers in our further evaluation.

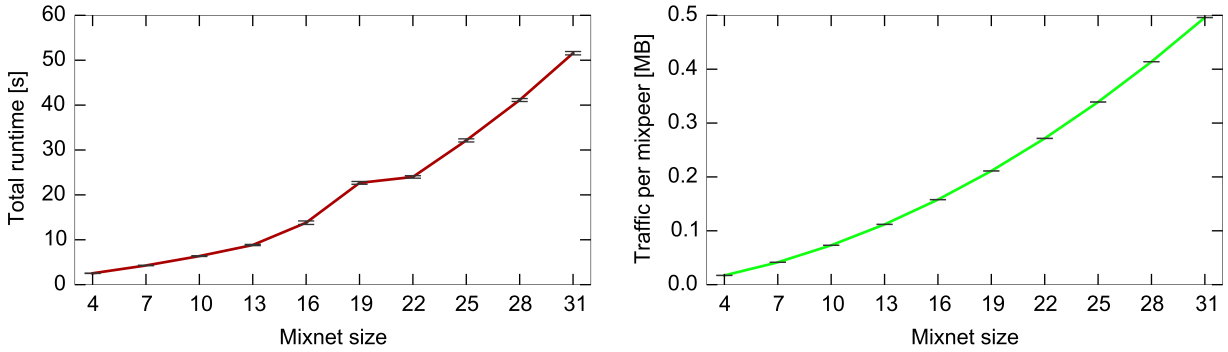
Experimental setup. We have implemented a prototype of *CoinParty* in Python 2.7 partly based on code from the *VIFF* SMC framework [57]. Communication between peers in all phases is handled by the asynchronous communication framework *Twisted* [35]. All point-to-point connections between mixing peers are secured with TLS and ECDSA is used to additionally sign all protocol messages. We use the *Elliptic Curve Integrated Encryption Scheme* (ECIES) over the `secp256k1` curve as encryption scheme E. Functionality related to the Bitcoin protocol, e.g., generating addresses and transactions, has been implemented using *bitcointools* [1]. All functionality related to elliptic curves cryptography is based on *pyelliptic* [26].

We benchmark the selected protocol phases in a local setup in order to quantify the bare processing and communication overheads. A comparison to a cloud setup that accounts for real-world bandwidth constraints and network latency is presented afterwards. In the local setup, each mixing peer is run as a separate process on a single host (16 cores @ 2.6 GHz, 32 GB RAM, Ubuntu 14.04 LTS) with all communication between peers over the local loopback interface. We run all considered phases in mixnets of different sizes $m = 4, 7, 11, ..., 31$ and for different numbers of input users $n = 10, 50, 100, 200$. The choice of the mixnet sizes is due to *CoinParty*'s security threshold $\tau < m/3$, e.g., a mixnet of size $m = 4$ is secure against one malicious peer, $m = 7$ against two, and so forth. For each phase, we present the total runtime and the communication overhead per mixpeer. The runtime is measured from the start of the first mixing peer until the last mixing peer finishes. All experiments are repeated in 20 independent runs. The depicted error bars denote the 99 percent confidence interval.

Initilization phase. Protocol Initialization has two tasks, i) to set up session keys and ii) to generate escrow addresses and precompute signatures. Session keys have to be set up only once for a new mixnet and can then be used for many mixing operations. Only when a mixing peer M_i is falsely accused of malicious behaviour, a new session key for M_i has to be setup (cf. BlameAndRecover, Protocol 7 in §4.5). Since key-setup is a *one-time* effort during the setup of the mixnet, we do not include it in the evaluation of the *reoccurring* initilization for setting up a mixing operation.

The overhead for the second task, generating escrow addresses and precomputing signatures, scales linearly in the number n_{max} of users that are allowed to the mixing. Thus, we present only the overheads for performing the precomputations for a single user and the total overhead is at most n_{max} times that overhead. Figure 3a plots the total runtime for the mixnet and Figure 4b the communication overhead per mixing peer.[1] A mixnet of medium size $m = 16$, e.g., needs 13.35 s to finish precomputations corresponding to one user.

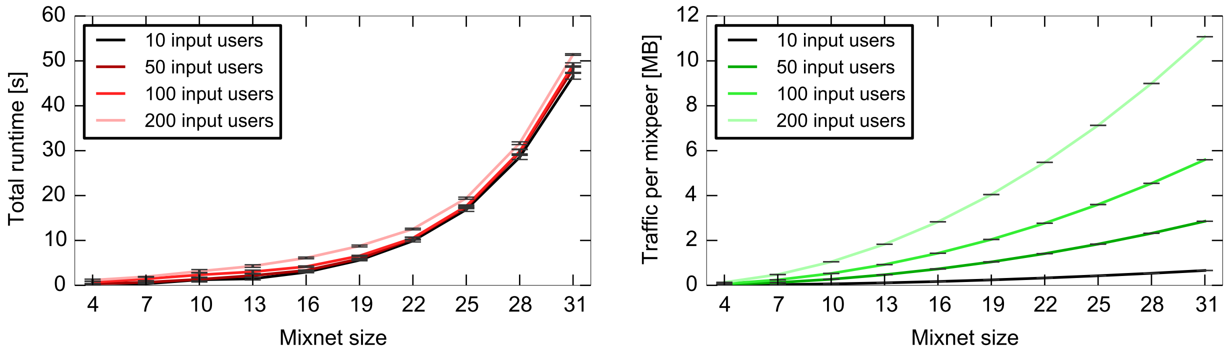
[1] The host we run our evaluation has only 16 cores. For $m = 19$ at least 3 mixing peers run as hyperthreads which are perceivably slower and impact the overall runtime. For $m = 22, 25, ..., 31$ this effect is more and more amortized by the overall increasing runtime of Shuffle. This explains the kink in the curve at $m = 19$.



(a) Runtime, measured from the start until the last mixing peer finishes.

(b) Communication overhead per mixing peer.

Figure 3: Overhead to precompute a single escrow address and partial signature for mixnets of size 4 to 31.



(a) Runtime, measured from the start until the last mixing peer finishes.

(b) Communication overhead per mixing peer.

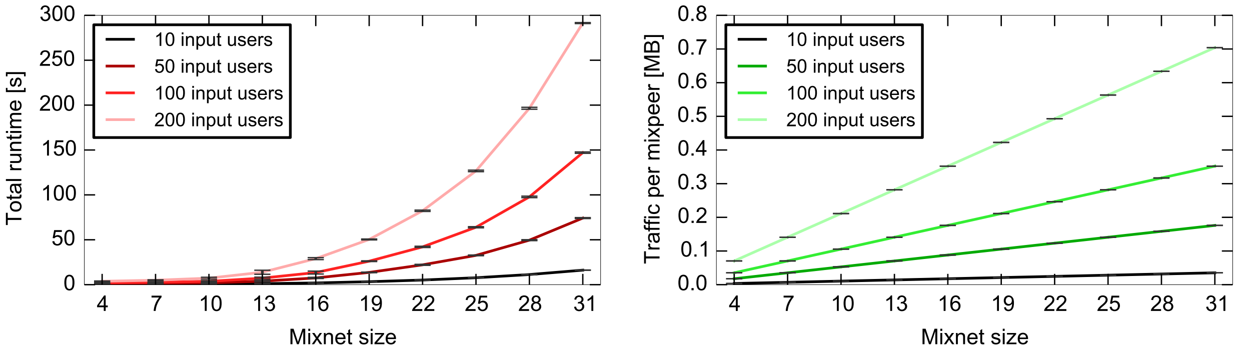
Figure 4: Overhead of the shuffling phase for $n = 10, 50, 100, 200$ users and mixnets of size 4 to 31.

Note that Initialization can be performed at any point in time before the mixing. We can also shift at least two thirds of the overhead, i.e., the overhead for precomputing partial signatures, into the transaction phase, if desired. This makes sense when the mixing window ω chosen for the transaction phase is longer than the precomputation overheads. In our most complex setting, i.e., a mixnet of $m = 31$ with $n = 200$ users, the mixnet needs 2.8 h to finish the initlization phase, of which at least 1.8 h are spent on precomputing the transaction phase. Thus, even for small mixing windows this overhead could be fully shifted to the transaction phase.

A considerable part of the overhead of Initialization is contributed by the EcDkg primitive which is invoked thrice. Note that in a previous version of this work [62], pseudo-random secret-sharing (PRSS) was used instead, which requires a trusted dealer to set up shared randomness in the mixing peers but incurs significantly smaller overheads. In contrast, EcDkg comes at the cost of approximately one order of magnitude higher overheads but is fully decentralized and does not require a trusted dealer.

Shuffling phase. Figure 4a plots the total runtime and Figure 4b the communication overhead per mixing peer for performing Shuffle for $n = 10, 50, 100, 200$ users in mixnets of sizes $m = 4$ to $m = 31$. We observe that both runtime and communication overhead even in the most complex case of $m = 31$ mixing peers and $n = 200$ users are small, i.e., the mixnet finishes in only 51 s and communicates 11 MB per mixing peer.

We also note that the runtime is almost independent of the number of participating users. The reason lies in the Recombine primitive used during verification of each shuffle stage: Recombine incurs one round of communication between all mixing peers, i.e., overhead quadratic in m. However, Recombine is invoked only once per shuffle stage and is thus independent of the number of users n. The remaining overhead of Shuffle is almost completely due to the required n decryptions per stage. However, the overhead incurred by Recombine dominates this overhead.



(a) Runtime, measured from the start until the last mixing peer finishes.

(b) Communication overhead per mixing peer.

Figure 5: Overhead of the transaction phase for $n = 10, 50, 100, 200$ and mixnets of size 4 to 31.

Transaction phase. Figure 5a plots the total runtime and Figure 5b the communication overhead per mixing peer for performing Transaction for $n = 10, 50, 100, 200$ users in mixnets of sizes $m = 4$ to $m = 31$. In contrast to Shuffle, the runtime of Transaction clearly scales linearly with the number n of participating users, e.g., a mixnet of size $m = 31$ takes approximately 149 s to pay off $n = 100$ users and 298 s for $n = 200$ users. This is again due to the Recombine primitive, which, other than for Shuffle, is invoked n times during Transaction. We emphasize against that transactions should be distributed over the whole mixing window, i.e., the mixnet would have to issue only a few single transactions from time to time. A single transaction costs only 1.5 s and only 3.6 kB of communication per mixing peer even in the largest mixnet of size $m = 31$. Even when all all $n = 200$ transactions should be issued at once, the overhead amounts to only 5 min. We conclude that the overheads imposed by the transaction phase are clearly feasible even for large mixnets and many users.

Cloud Setup. We evaluate each protocol phase again in a cloud setup in order to quantify runtime overheads due to bandwidth constraints and latency in a real-world scenario. In the cloud setup, each mixing peer is deployed in Microsoft's Azure Cloud using a separate DS1 instance (1 core @ 2.2 GHz, 3.5 GB RAM, Ubuntu 14.04 LTS) for each mixing peer. The virtual machines are distributed over different geographical locations in Western Europe and Central US. The pairwise round trip times (RTTs) between mixing peers are 110 ms for communication between Europe and US and 5 – 10 ms within Europe and the US, respectively. Bandwidth was measured at 120 Mbit/s to 500 Mbit/s for intercontinental and intracontinental communication, respectively. We also limit the setting to the largest mixnet size $m = 16$ that could be handled in the local setup without hyperthreading. Since cloud instances also have less processing power than the host used in the local setup, the following results thus present an upper bound for the runtime overheads imposed by communication.

Figure 6 compares the runtimes of the different phases in the local setup (grey bars) and the *added* overheads in the cloud setup (red bars). As in the local setup, precomputing one escrow address is the most expensive operation with a duration of 18 s, amounting to an added overhead of approximately 5 s or 38 % compared to the local setup. For $n = 200$ users, the mixnet initialization phase finishes after 1.03 h, of which at least 40 min are spent on precomputing the transaction phase. Thus, even for our smallest mixing windows we could shift this precomputation from the initialization phase to the transaction phase. In the shuffling phase, the cloud setup almost triples the overheads, e.g., shuffling $n = 200$ users takes 6 s in the local setup compared to 15 s in the cloud setup. This significant increase is dominated by the efforts for performing reliably broadcasts in each shuffle round. The overhead induced by a reliable broadcast is determined by the maximum latency occurring between any two mixing peers. However, with less than 5 min in our most complex setting, i.e., $m = 31, n = 200$, the overhead of the shuffle phase in the cloud setup is still much lower than, e.g., the time the Bitcoin network ultimately needs to validate the final mixing transactions. Finally, the cloud setup adds approximately 38 % runtime overhead to the transaction phase, e.g., increasing runtime for $n = 200$ users from 29 s to 40 s. Since transactions should be distributed over the whole mixing window, i.e., several hours, we consider this overhead to be insignificant. In summary, *CoinParty* performs efficiently even in a real-world cloud setup with high latency and low bandwidth between mixing peers.

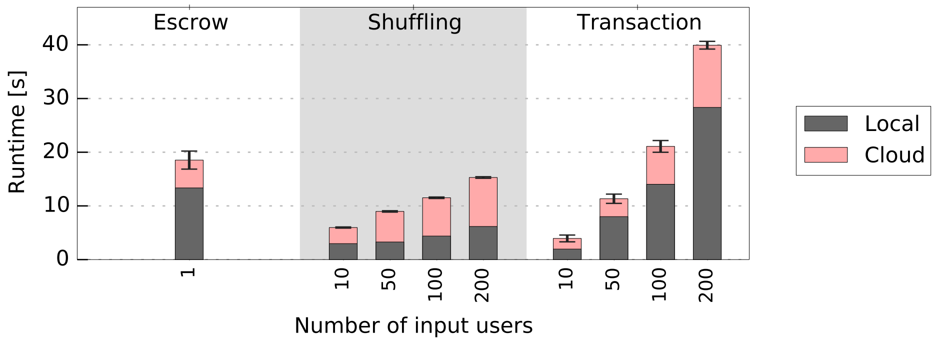


Figure 6: Comparison of the runtime of each protocol phase in the local setup (grey) and cloud setup (light red) for $m = 16$ mixing peers.

Comparison to related work. We briefly relate *CoinParty*'s performance to that of *CoinShuffle* [52], which is among all mixing services the most similar to *CoinParty*. In CoinShuffle, the mixing is executed directly by the users. This corresponds to setting m equal to n in *CoinParty*. A CoinShuffle mixing between $n = 30$ users takes approximately 20 s. The comparable setting in *CoinParty* takes 50 s for Shuffle and 45 s for Transaction. The total of 135 s is approximately 7 times the overhead of CoinShuffle. This increased overhead is what we pay for the improved anonymity, deniability, and resilience against DoS attacks. It is also relativized by the fact that, other than for CoinShuffle, we do not need to scale the number of mixing peers with the number of users. Thus, *CoinParty* could mix $n = 200$ users in a mixnet of size $m = 13$ in approximately 25 s compared to approximately 160 s for CoinShuffle. The separation of mixing peers and users thus allows *CoinParty* to scale much more efficiently to large number of users. In summary, we thus conclude that *CoinParty* fulfils the stated *scalability* requirement.

5.5. *Cost efficiency*

Mixing services can principally require two kinds of fees, i) transaction fees, paid for including the issued transactions in the blockchain, and ii) mixing fees, paid to the mixing services themselves. We briefly discuss both types of fees and show that *CoinParty* requires none.

Transaction fees. Transaction fees are usually paid proportionally to the size of the issued transaction, i.e., usually 0.0001 BTC per 1 kB (cf. §2). *CoinParty* issues n separate small transactions that are well below 1 kB in size. Such small transactions can even be issued without a fee if their outputs are at least 0.01 BTC [12]. When mixing smaller amounts, a transaction fee τ must be paid. Users simply commit at least $v + \tau$ funds to E_j but only v funds are transferred in the mixing transaction $E_j \rightarrow O_{\pi(j)}$. The costs are then τ BTC for the commitment and another τ BTC for the mixing transaction. Note that the related works that are based on a single group transaction require only a transaction fee proportional to one single transaction per user. However, since group transactions quickly exceed the 1 kB threshold, the transaction fee is required even when mixing bigger amounts of BTC. In summary, *CoinParty* does not require transaction fees for larger mixing amounts $v \geq 0.01$ BTC and only the fee for two single transactions per user for small mixing values.

Mixing fees. Anyone, even the users themselves, can set up a *CoinParty* mixing peer and collaborate with others to provide a secure and anonymous mixing service. Thus, *CoinParty* does not depend on any third party services that could dictate a mixing fee. On the other side, as we note in §5.2, it might be reasonable to charge fees to economically disincentivize sybil attacks. Mixing fees, however, must be handled with care because they can potentially identify mixing transactions which would decrease anonymity and chances of deniability. Different solutions have been discussed in [15].

5.6. *Applicability & Usability*

CoinParty deviates from the standard Bitcoin protocol in two ways. First, *CoinParty* implements a distributed generation of Bitcoin addresses used as escrow addresses. Apart from the shared private key, escrow addresses are exactly the same as normal Bitcoin addresses. A user can thus commit funds to escrow addresses using any Bitcoin

Approach	**Correctness**	**Anonymity**			**Deniability**	**Scalability**	**Costs**	**Applic.**
1^{st} *Generation* [9, 13]	None	0	$> n - c$	$>> n$	✓	$> 1000s$	Mix + 2 TX	✓
Mixcoin [15]	Accountability	0	$> n - c$	$>> n$	✓	$> 1000s$	Mix + 2 TX	✓
Blindcoin [56]	Accountability	n	$n - c$	n	I	$> 1000s$	Mix + 4 TX	(✓)
CoinJoin [40]	Group TX	0	$n - c$	n	✗	$\sim 10s - 100s$	1 TX	✓
CoinShuffle [52]	Group TX	n	$n - c$	n	✗	$\sim 100s$	1 TX	✓
SMC [51, 60]	Group TX	n	$n - c$	n	✗	-	1 TX	✗
ZeroCoin [5, 22, 44]	ZKPs	All ZeroCoin users			✓	-	2 TX	✗
CoinParty	2/3 honest	n	$> n - c$	$>> n$	✓	$\sim 100s - 1000s$	2 TX	✓

Table 1: Comparison of our *CoinParty* system to the related works by the most important design requirements. For anonymity, we provide the size of the anonymity set a user enjoys against (1) the mixing service (2) the other n users of which c might be compromised, and (3) outside observers.

client. Second, *CoinParty* introduces threshold transactions to redeem funds from the shared control escrow addresses. Threshold transactions are only generated differently than normal transactions, but are otherwise exactly the same. Thus, *CoinParty* is fully compatible with the Bitcoin protocol. It is also applicable in different settings. Though we separated users and mixing peers throughout this work, this separation is only logical: *CoinParty* can be run by a set of dedicated mixing peers or just as well by the users themselves.

In order to provide high usability, each mixing peer in *CoinParty* runs a web server that allows users to connect to the mixnet using only their browser. We additionally implement the complete commitment phase to run in the user's browser, so that the user only needs her standard Bitcoin client to issue the commitment. No other software is required. Furthermore, we require only one short information exchange with a user, i.e., after providing the required information a user never needs to be contacted again.

6. Analysis of Related Work

We first discuss other approaches to Bitcoin mixing (§6.1) and analyze whether they offer the desired properties for mixing services stated in §1.1. We then briefly discuss related work which is orthogonal to ours in §6.2.

6.1. Approaches to Bitcoin Mixing

We divide our analysis of mixing services into two categories, since approaches in one category tend to share key characteristics: Those that use a centralized architecture [9, 11, 13, 15, 56] in §6.1.1, those that use a distributed architecture [5, 40, 44, 51, 52, 60] in §6.1.2. The results and a comparison to *CoinParty* are summarized in §6.1.3 and Table 1.

6.1.1. Centralized Mixing

1^{st} *Generation Bitcoin mixes* [9, 11, 13] exhibit completely centralized architectures which causes two disadvantages. First, they do not provide any guarantees that the escrowed funds are effectively mixed [45] or even paid back [10]. Mixcoin [15] also has a centralized architecture but improves on 1^{st} Generation mixes by introducing a mechanism to provably blame corrupted mixes for theft of funds. Accountability, the authors argue, incentivizes mixes to behave honestly so as not to damage their reputation. We see different problems with this approach: First, malicious mixes have clear incentives and means to forge a reputation, e.g., the vulnerability of reputation systems to sybil attacks has been widely studied [29]. Second, proving that correct mixings have taken place, i.e., to prevent such sybil attacks, without harming anonymity and deniability of the mixing participants is non-trivial. Third, reputation systems are only reactive and can only punish theft of funds but cannot prevent it outright, e.g., in the case of a mix that after a couple of bad ratings decides to re-open under a new name.

The second major disadvantage of 1^{st} Generation mixes as well as of Mixcoin is that users have no anonymity against the mix itself. Since the mix alone shuffles all funds, it knows all mappings of input to output addresses. The centralized mix then represents a single point of failure that could be breached, incentivized, or forced to reveal this sensitive information to third parties, e.g., to governmental agencies under subpoena [34, 55]. *Blindcoin* [56] directly builds on Mixcoin and in contrast to it and the other approaches [9, 11, 13] achieves anonymity against a compromised mixing service through the use of blind signatures and a public append-only log.

On the other side, centralized mix services have some advantageous properties. First, centralized mixing uses only standard transactions which cannot be distinguished from non-mixing transactions if fresh addresses are used for each mixing operation. This property was termed *mix indistinguishability* by the authors of Mixcoin. Mix indistinguishability affords high degrees of anonymity and deniability against passive observers as well as against other users of the mix. Unfortunately, Blindcoin's improvement in terms of anonymity break this property, because users leave cryptographic evidence in the log which irrefutably binds their output addresses to the mix.

A second desirable property of centralized mixing is its resilience against DoS attacks from malicious users, which can at very low costs, e.g., by simply providing corrupted inputs, cause most distributed mixing protocols [40, 51, 52, 60] to fail. However, a centralized mixing service still represents a single point-of-failure and may thus itself become a target for DoS attacks, e.g., from competitors.

Finally, since each user of [9, 11, 13, 15, 56] imposes only a small overhead of two signatures (three for [56]) on the mix, those approaches can be expected to scale easily to thousands of users even though concrete performance results have not been presented for the respective approaches.

6.1.2. Distributed Mixing

CoinJoin [40] first proposed to use group transactions (§2) for Bitcoin mixings and others [51, 52, 60] have later built on this idea. While the atomic nature of group transactions provides perfect protection against theft, group transactions also have undesirable properties: First, since all transactions belonging to one mixing operation are grouped together, the anonymity achieved against a passive observer is reduced to exactly the participants in the single mixing operation. Second, deniability is broken because group transactions corresponding to a mixing are easily distinguishable from non-mixing transactions. Third, a single malicious user can launch a DoS attack on the whole mixing operation just by denying to sign the group transaction.

Besides these general disadvantages of using group transactions for mixing, CoinJoin does not offer a solution on how addresses can be shuffled without a central mixing service. In this point, CoinShuffle [52] improves over CoinJoin and most of the centralized approaches by introducing a distributed oblivious shuffling protocol based on Chaum's mixnets [18]. As long as at least two of the mixing peers executing the shuffling remain uncompromised, users remain anonymous even in the presence of an active attacker compromising other mixing peers. Others proposed to use SMC for oblivious shuffling [27, 51, 60]. Unfortunately, these approaches were never thoroughly specified and neither are they expected to scale.

A clear advantage of all approaches that are based on the group transaction pattern is that, in contrast to all centralized approaches, they do not require any escrow of the mixed funds. Instead, a mixing operation is executed in one single big transaction, i.e., at the amortized costs of a single small standard transaction per user, which is optimal in terms of transaction fees.

ZeroCoin [44] introduced a completely different breed of distributed mixes. ZeroCoin extends the Bitcoin system with a new type of transaction, i.e., of ZeroCoins. Before spending any ZeroCoins, a user has to mint ZeroCoins from her own Bitcoins and supply them to the system. Transactions of ZeroCoins are unlinkable since they reveal neither the origin respective destination nor the amount of the transaction. This is achieved by proving the correctness of a transaction using Zero-Knowledge Proofs (ZKPs), which do not reveal the particular ZeroCoin that was spent in the transaction. While the initial approach [44] does not scale, it was later significantly improved by *ZeroCash* [5] and *PinocchioCoin* [22]. However, all approaches in this line of research run only as extensions to Bitcoin that would require substantial modifications and at least the majority of Bitcoin nodes to accept those.

6.1.3. Discussion and Comparison

Centralized mixing exhibits favorable properties w.r.t. anonymity, scalability, and deniability under the strong assumption that the mixing service itself can be completely trusted. If this assumption does not hold and the mixing service itself is corrupted, correctness and anonymity guarantees are insufficient. Blindcoin is a notable exception, which achieves anonymity against a compromised mixing service but forfeits deniability in this process. Distributed mixing based on group transactions generally provides stronger correctness guarantees at the costs of lower anonymity sets and no plausible deniability. Also, scalability becomes an issue with these approaches. The Zero-Knowledge based mixes achieve high anonymity but are incompatible with the current Bitcoin system.

The most differentiating aspect and a core contribution of *CoinParty* is the secure replacement of the disadvantageous group transaction pattern of [40, 51, 52, 60] with multiple threshold transactions that are indistinguishable from standard non-mixing Bitcoin transactions. This allows *CoinParty* to combine the advantages of distributed and centralized mixing services in one system: *CoinParty* achieves strong protection against theft and full anonymity against the mixing service as the other distributed approaches. At the same time, *CoinParty* achieves orders-of-magnitude higher anonymity sets and plausible deniability as was previously only provided by the centralized approaches. To the best of our knowledge, *CoinParty* is the first system to combine these advantages in one system.

A second core contribution that sets *CoinParty* apart from the other distributed approaches is the separation of users and mixes. This allows *CoinParty* to scale better than other distributed approaches [40, 52] although still not as good as the centralized approaches. Furthermore, through this separation and our new shuffle verification scheme, we achieve for the first time in a distributed mixing service resilience against DoS attacks from malicious users.

6.2. *Orthogonal Work*

Others have in parallel to our work leveraged threshold ECDSA for improving security in Bitcoin, but have applied it to other problems than mixing. Mann and Loebenberger [39] adapt MacKenzie's [38] two-party DSA signature scheme to ECDSA and apply it to provide two-factor authentication for Bitcoin wallets. Bonneau et al. [14] and Goldfeder et al. [24] significantly extend on this idea and propose a novel threshold ECDSA signature scheme that allows to realize distributed Bitcoin wallets with arbitrary access structures. However, this increased flexibility, compared to the schemes by MacKenzie [38] or Ibrahim [30], comes at significantly increased processing and communication overheads. In the context of distributed mixing services, where all mixing peer have the same privileges, such arbitrary access structures are neither required nor do they increase security.

7. Conclusion

The promised anonymity of Bitcoin payments has been successfully attacked by analyzing the transaction graph documented in the public blockchain [2, 43, 49, 50, 54]. Normal Bitcoin users are not only unaware of such passive attacks on their privacy, but also have little means to prevent them. These works make evident the need for a user-centric solution to reestablishing financial privacy in Bitcoin. To this end, a recent line of research investigates secure and anonymous Bitcoin mixing services [15, 40, 52]. However, a detailed analysis reveals disadvantages with each of those systems (§1.1 and §6).

In this paper, we have thus presented *CoinParty*, a novel decentralized mixing service that improves significantly over the related work by combining the advantages of centralized and decentralized mixing services in a single system. *CoinParty* achieves this by introducing dedicated mixing peers that run the distributed mixing protocol. At the core of the mixing protocol is a new verifiable oblivious shuffling protocol based on decryption mixnets [20, 52] and an efficient threshold cryptosystem for ECDSA [30]. We show by analysis of the actual Bitcoin blockchain that *CoinParty*'s threshold transaction pattern is favorable compared to the group transaction pattern of related work in terms of anonymity, deniability and resilience against DoS attacks from malicious attackers. An extensive evaluation of our implemented prototype shows that *CoinParty* scales to large numbers of users due to the possible separation of users and mixing peers. All primitives and the protocols built upon them are secure against attacks from malicious mixing peers and users and are of interest also beyond our work, e.g., for securing Bitcoin wallets [14, 24]. Finally, although we focused on Bitcoin, our work is directly compatible with other crypto-currencies which use the same ECDSA primitive, e.g., Litecoin and Mastercoin.

8. References

[1] Andresen, G., 2014. bitcointools. `https://github.com/gavinandresen/bitcointools`.

[2] Androulaki, E., et al., 2013. Evaluating User Privacy in Bitcoin. In: FC. Springer.

[3] Asharov, G., Lindell, Y., Rabin, T., 2011. Perfectly-Secure Multiplication for Any $t < n/3$. In: Rogaway, P. (Ed.), Advances in Cryptology – CRYPTO 2011. Vol. 6841 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 240–258.

[4] Ben-Or, M., Goldwasser, S., Wigderson, A., 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. STOC '88. ACM, New York, NY, USA, pp. 1–10.
URL `http://doi.acm.org/10.1145/62212.62213`

[5] Ben-Sasson, E., et al., 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In: Security & Privacy. IEEE.

[6] Biryukov, A., Pustogarov, I., May 2015. Bitcoin over tor isn't a good idea. In: Security and Privacy (SP), 2015 IEEE Symposium on. pp. 122–134.

[7] Bissias, G., Ozisik, A. P., Levine, B. N., Liberatore, M., November 2014. Sybil-Resistant Mixing for Bitcoin. In: Proc. ACM Workshop on Privacy in the Electronic Society.
URL `http://forensics.umass.edu/pubs/bissias.wpes.2014.pdf`

[8] Bitcoin Charts, 2014. Bitcoin Charts. `http://bitcoincharts.com/bitcoin/`.

[9] Bitcoin Fog, 2014. Bitcoin Fog. `http://www.bitcoinfog.com/`.

[10] Bitcoin Fog, 2015. Bitcoin Fog Reviews. `http://www.thebitcoinreview.com/site.php?site%20id=759`.

[11] Bitcoin Wiki, 2014. Mixing Services. `http://en.bitcoin.it/wiki/Category:Mixing_Services`.

[12] Bitcoin Wiki, 2014. Transaction fees. `https://en.bitcoin.it/wiki/Transaction_fees`.

[13] BitLaundry, 2014. BitLaundry. `http://bitlaundry.appspot.com/`.

[14] Bonneau, J., Felten, E. W., Kalodner, H., Gennaro, R., Kroll, J. A., Goldfeder, S., Narayanan, A., 2015. Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme.

[15] Bonneau, J., et al., 2014. Mixcoin: Anonymity for Bitcoin with accountable mixes. In: FC.

[16] Brickell, J., Shmatikov, V., 2006. Efficient Anonymity-Preserving Data Collection. In: SIGKDD. ACM.

[17] Chaum, D., 1988. The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of cryptology 1 (1), 65–75.

[18] Chaum, D. L., 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24 (2), 84–90.

[19] CoinSplitter, 2014. CoinSplitter. `http://coinsplitter.org/`.

[20] Corrigan-Gibbs, H., Ford, B., 2010. Dissent: Accountable Anonymous Group Messaging. In: CCS. ACM.

[21] Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N., 2013. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In: Crampton, J., Jajodia, S., Mayes, K. (Eds.), Computer Security – ESORICS 2013. Vol. 8134 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–18.
URL `http://dx.doi.org/10.1007/978-3-642-40203-6_1`

[22] Danezis, G., Fournet, C., Kohlweiss, M., Parno, B., 2013. Pinocchio coin: Building zerocoin from a succinct pairing-based proof system. In: Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies. PETShop '13. ACM, New York, NY, USA, pp. 27–30.
URL `http://doi.acm.org/10.1145/2517872.2517878`

[23] Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T., Jan. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. J. Cryptol. 20 (1), 51–83.

[24] Goldfeder, S., Bonneau, J., Felten, E. W., Kroll, J. A., Narayanan, A., 2014. Securing bitcoin wallets via threshold signatures. Princeton University, http://www. cs. princeton. edu/ stevenag/bitcoin threshold signatures. pdf.

[25] Goldreich, O., Micali, S., Wigderson, A., 1987. How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. STOC '87. ACM, New York, NY, USA, pp. 218–229.
URL `http://doi.acm.org/10.1145/28395.28420`

[26] Guibet, Y., 2014. pyelliptic. `https://pypi.python.org/pypi/pyelliptic`.

[27] hashcoin, 2011. Blind Bitcoin Transfers. Forum Post, `https://bitcointalk.org/index.php?topic=12751.msg315793#msg315793`.

[28] Henze, M., Hiller, J., Hohlfeld, O., Wehrle, K., 2016. Moving Privacy-Sensitive Services from Public Clouds to Decentralized Private Clouds. In: 2016 IEEE International Conference on Cloud Engineering Workshops.

[29] Hoffman, K., Zage, D., Nita-Rotaru, C., Dec. 2009. A survey of attack and defense techniques for reputation systems. ACM Comput. Surv. 42 (1), 1:1–1:31.
URL `http://doi.acm.org/10.1145/1592451.1592452`

[30] Ibrahim, M., et al., 2003. A robust threshold elliptic curve digital signature providing a new verifiable secret sharing scheme. In: Circuits and Systems. IEEE.

[31] Koshy, P., Koshy, D., McDaniel, P., 2014. An Analysis of Anonymity in Bitcoin Using P2P Network Traffic. FC.

[32] Kreuter, B., Shelat, A., Mood, B., Butler, K. R., 2013. Pcf: A portable circuit format for scalable two-party secure computation. In: Usenix Security. Vol. 13. pp. 321–336.

[33] Kreuter, B., Shelat, A., Shen, C.-H., 2012. Billion-gate secure computation with malicious adversaries. In: USENIX Security Symposium. Vol. 12. pp. 285–300.

[34] Ladar Levinson, 2014. Secrets, lies and Snowden's email: why I was forced to shut down Lavabit. The Guardian, `http://www.theguardian.com/commentisfree/2014/may/20/why-did-lavabit-shut-down-snowden-email`.

[35] Lefkowitz, G., 2002. Twisted Software. `https://twistedmatrix.com/trac/`.

[36] Lindell, Y., Pinkas, B., 2009. Secure Multiparty Computation for Privacy-Preserving Data Mining. Journal of Privacy and Confidentiality 1 (1).

[37] Lyubashevsky, V., 2005. The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In: RANDOM. Springer.
URL `http://dx.doi.org/10.1007/11538462_32`

[38] MacKenzie, P., Reiter, M. K., 2001. Two-party generation of dsa signatures. In: Advances in Cryptology—CRYPTO 2001. Springer, pp. 137–154.

[39] Mann, C., Loebenberger, D., 2014. Two-factor authentication for the bitcoin protocol.

[40] Maxwell, G., 2013. CoinJoin. Forum post, `https://bitcointalk.org/index.php?topic=279249`.

[41] Maxwell, G., 2013. CoinSwap. Forum post, `https://bitcointalk.org/index.php?topic=321228`.

[42] McEliece, R. J., Sarwate, D. V., Sep. 1981. On Sharing Secrets and Reed-Solomon Codes. Commun. ACM 24 (9), 583–584.

[43] Meiklejohn, S., et al., 2013. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In: IMC. ACM.

[44] Miers, I., et al., 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In: Security & Privacy. IEEE.

[45] Moser, M., Bohme, R., Breuker, D., 2013. An inquiry into money laundering tools in the bitcoin ecosystem. In: eCrime Researchers Summit

(eCRS), 2013. IEEE, pp. 1–14.

[46] Nakamoto, S., 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. bitcoin.org.

[47] Pedersen, T. P., 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '91. Springer-Verlag, London, UK, UK, pp. 129–140.
URL http://dl.acm.org/citation.cfm?id=646756.705507

[48] Reed, I., Solomon, G., 06/1960 1960. Polynomial Codes Over Certain Finite Fields. Journal of the Society of Industrial and Applied Mathematics 8 (2), 300–304.
URL http://www.jstor.org/pss/2098968

[49] Reid, F., Harrigan, M., 2011. An Analysis of Anonymity in the Bitcoin System. In: PASSAT.

[50] Ron, D., Shamir, A., 2013. Quantitative Analysis of the Full Bitcoin Transaction Graph. In: FC. Springer.

[51] Rosenfeld, M., 2012. Using mixing transactions to improve anonymity. Forum post, https://bitcointalk.org/index.php?topic=54266.

[52] Ruffing, T., Moreno-Sanchez, P., Kate, A., 2014. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In: HotPETS.

[53] Shamir, A., Nov. 1979. How to share a secret. Commun. ACM 22 (11), 612–613.
URL http://doi.acm.org/10.1145/359168.359176

[54] Spagnuolo, M., Maggi, F., Zanero, S., 2014. BitIodine: Extracting Intelligence from the Bitcoin Network. FC.

[55] The Guardian, 2014. US court forces Microsoft to hand over personal data from Irish server. http://www.theguardian.com/technology/2014/apr/29/us-court-microsoft-personal-data-emails-irish-server.

[56] Valenta, L., Rowan, B., 2015. Blindcoin: Blinded, accountable mixes for bitcoin. In: Workshop on Bitcoin Research.

[57] VIFF Development Team, 2010. VIFF, the Virtual Ideal Functionality Framework. http://viff.dk/.

[58] Wang, X., Liu, C., Nayak, K., Huang, Y., Shi, E., 2015. Oblivm: A programming framework for secure computation. In: IEEE Symposium on Security and Privacy (S & P).

[59] Welch, L., Berlekamp, E., Dec. 30 1986. Error Correction for Algebraic Block Codes. US Patent 4,633,470.
URL http://www.google.com/patents/US4633470

[60] Yang, E. Z., 2012. Secure multiparty Bitcoin anonymization. Blog post, http://blog.ezyang.com/2012/07/secure-multiparty-bitcoin-anonymization/.

[61] Zhang, Y., Steele, A., Blanton, M., 2013. Picco: a general-purpose compiler for private distributed computation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, pp. 813–826.

[62] Ziegeldorf, J. H., Grossmann, F., Henze, M., Inden, N., Wehrle, K., 2015. Coinparty: Secure multi-party mixing of bitcoins. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy. ACM, pp. 75–86.

Appendix A. Output Addresses Substitution

We show how a malicious adversary compromising the last mixing peer M_m can steal funds during the shuffling phase (Section 4.3) if he can solve an instance of the *Random Modular Subset Sum* problem. The last mixing peer M_m can substitute the original output addresses $O_1, ..., O_n$ with his own $O'_1, ..., O'_n$ without being detected iff $C_{m+1} = \sum_{i=1}^{n} \textsc{Hash}(O_i) = \sum_{i=1}^{n} \textsc{Hash}(O'_i)$. Finding such a sequence $O'_1, ..., O'_n$ requires solving a *Random Modular Subset Sum* (RMSS) problem, with modulus $M = 2^{len(\textsc{Hash})}$, random elements $\{a_1, ..., a_N\} \in_U [0, M)$ and target sum C_{m+1}. $\delta = N/log(M)$ is called the density of the problem and high density RMSS instances are very likely to have a solution. The attacker can generate such a high density instance of the problem by pre-generating a large number of addresses O'_i and corresponding hashes $a_i = \textsc{Hash}(O'_i) \in [0, M)$. The best known algorithm for solving high density instances has runtime $M^{O(1/log(N))}$ [37]. Clearly, the runtime decreases if N grows, i.e., if we generate a larger pool of hashes to select the n addresses O'_i from. Contrarily, as M, the range of the hash function, grows, the runtime increases.

We make a practical estimate, which sizes of the RMSS problem are solvable in reasonable time. As a concrete example, we assume our checksum C_{m+1} is computed from 512-bit hashes, i.e. $M = 2^{512}$. Then, generating 10^{12} hashes amounts to roughly 58 TB of storage and we assume it is not feasible to generate more than $N \approx 10^{12}$ such hashes. Thus, setting $N = 10^{12}$ gives a lower bound on the runtime. Assuming there is a solution at all, it can then be found in $(2^{512})^{1/log(10^{12})} = 2^{512/12} = 2^{52}$ operations, which is already quite challenging to solve practically. Using 1536-bit hashes, i.e. $M = 2^{1536}$, we get a runtime of 2^{128} operations which is comparable to the complexity of breaking keys for long term security for AES. Note that using 1536-bit or even longer hashes for the checksum computation does not significantly increase the overhead of the protocol. Thus we conclude that the RMSS attack can be thwarted by putting a time-bound on the Step 5.1b of Shuffle (Protocol 5 in §4.3).

Appendix B. Sybil Attacks

An attacker that can compromise c of the n users can tie a fraction $f = 1-(1-c/n)^2$ of the n mixing transactions to the mixing. Figure B.7 plots the fraction f of identified transactions versus the fraction c/n of compromised users. The

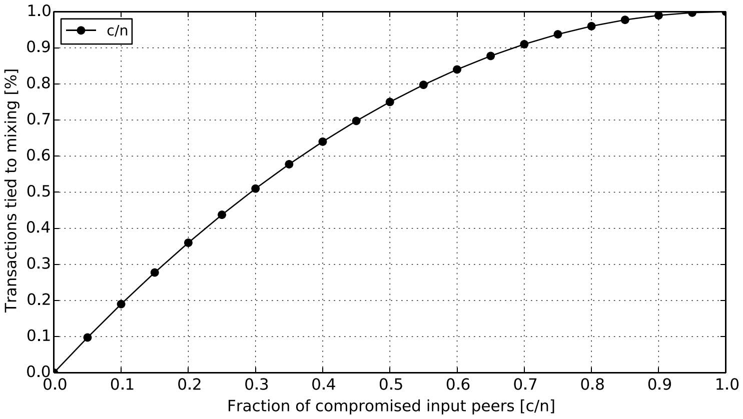


Figure B.7: Fraction of identified mixing transactions vs fraction of compromised users.

plot, e.g., shows that an attacker can already tie 75 % of the honest participants to the mixing, if he can compromise 50 % of the participants.