CoinParty: Secure Multi-Party Mixing of Bitcoins

Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, Klaus Wehrle
Communication and Distributed Systems (COMSYS), RWTH Aachen University, Germany
lastname@comsys.rwth-aachen.de

ABSTRACT

Bitcoin is a digital currency that uses anonymous cryptographic identities to achieve financial privacy. However, Bitcoin's promise of anonymity is broken as recent work shows how Bitcoin's blockchain exposes users to reidentification and linking attacks. In consequence, different *mixing services* have emerged which promise to randomly mix a user's Bitcoins with other users' coins to provide anonymity based on the unlinkability of the mixing. However, proposed approaches suffer either from weak security guarantees and single points of failure, or small anonymity sets and missing deniability. In this paper, we propose *CoinParty* a novel, decentralized mixing service for Bitcoin based on a combination of decryption mixnets with threshold signatures. *CoinParty* is secure against malicious adversaries and the evaluation of our prototype shows that it scales easily to a large number of participants in real-world network settings. By the application of threshold signatures to Bitcoin mixing, *CoinParty* achieves anonymity by orders of magnitude higher than related work as we quantify by analyzing transactions in the actual Bitcoin blockchain and is first among related approaches to provide plausible deniability.

Categories and Subject Descriptors

K.4.4 [**Electronic Commerce**]: Cybercash, digital cash

Keywords

Bitcoin; Anonymity; Secure Multi-Party Computation

1. INTRODUCTION

Bitcoin was proposed in 2008 by Nakamoto [24] as a decentralized digital currency. The market cap of circulating Bitcoins amounts to nearly $5 Billion [4] as of December 2014 which shows the wide adoption of Bitcoin. Instead of using central entities, i.e., banks, to establish trust in the currency, Bitcoin stores all transactions in a distributed public ledger, the *blockchain*, to prevent double spending and keep track of the balances. Bitcoins are stored at and transferred between addresses, cryptographic identities corresponding 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. It is especially this promise of financial privacy that has drawn great interest towards the Bitcoin currency.

CODASPY'15, March 2–4, 2015, San Antonio, Texas, USA.

ACM 978-1-4503-3191-3/15/03 ...$15.00.
http://dx.doi.org/10.1145/2699026.2699100.

However, this promise is broken, as recent works [2, 22, 25, 26] show how a user's transactions and addresses can often be linked back together by analyzing the transaction graph in the publicly available blockchain. Though, any user, Alice, can generate fresh unlinkable addresses, transferring funds to them would again link the address back to her. Basically, what Alice needs is a way to send funds to her new address in an unlinkable manner. As of today, `bitcoin.it` lists no less than thirteen commercial mixing services [6] that promise to provide exactly this: If Alice sends her funds to the mixing service, the mix will, for a small fee, pay her back with the funds of some random other user after a certain waiting period. To any outsider observing the blockchain the resulting mixing transactions are indistinguishable from other contemptuous transactions in the blockchain. This property is desirable since it provides for large anonymity sets and allows 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. Second, the mixing service knows how funds were mixed and may be forced or otherwisely incentivized to reveal this knowledge. This has led to new decentralized approaches that promise better security and stronger anonymity [20, 28, 31]. However, most of these approaches require that all mixing transactions are issued in one single *atomic* transaction with multiple inputs and outputs. This serves to prevent malicious peers from aborting the protocol after they have received their funds thereby leaving another peer unpaid. However, the characteristic form of such *group transactions* renders them easily identifiable in the blockchain. This introduces two severe limitations when they are used in mixing services: First, the resulting anonymity set of the mixing is limited to the number of users participating in a particular mixing operation. Second, since such bundled mixing transactions are clearly identifiable in the blockchain, users have no means of plausibly denying that they participated in the mixing.

Our contribution. In this paper, we propose *CoinParty*, an efficient decentralized Bitcoin mixing service with stronger anonymity guarantees, plausible deniability, and lower costs. *CoinParty* takes a two stage approach. Our core idea is to introduce a set of *mixing peers* that in a decentralized yet secure fashion carry out the mixing in multiple one-to-one Bitcoin transactions thereby replacing the disad-

vantageous group transactions used in the related work. The key challenge is to ensure that all mixing transactions succeed even when mixing peers fail or behave maliciously. We show how to achieve this by employing a threshold variant of the ECDSA scheme realized using general Secure Multi-Party Computation (SMC) protocols. This scheme allows to distributedly create Bitcoin addresses from which funds can only be redeemed in a *threshold transaction*, i.e., only when a majority of the controlling peers agrees to do so. *CoinParty* can thus realize the advantages of *both* the early centralized approaches and of the later decentralized approaches, achieving notable improvements compared to the related work:

Improved anonymity: To outsiders and the fellow mixing participants observing the blockchain, *CoinParty*'s threshold transactions are indistinguishable from other non-threshold transactions. Thus, *CoinParty*'s mixing transactions are anonymous among *all contemptuous* transactions with the same value. We conduct a quantitative analysis on the blockchain to show that this increases anonymity by orders of magnitude.

Plausible deniability: In [9] the authors briefly mention *deniability* as one desirable property for mixings. *CoinParty* is first among related approaches for mixing Bitcoins to provide its users with plausible deniability.

No fees: *CoinParty* issues multiple small transactions that do not require transaction fees. Since *CoinParty* is run in absence of any trusted third party, i.e., a service provider, no mixing fees are charged either. Related approaches require at least transaction or mixing fees.

Applicability: *CoinParty* is fully compatible with the existing Bitcoin network. We evaluate a proof-of-concept implementation in a real-world network setting to show that *CoinParty* incurs only small overheads even when scaling to large numbers of participants.

The remainder of the paper is structured as follows: We present background information on the Bitcoin digital currency and SMC in Section 2. Section 3 formalizes the problem and requirements of Bitcoin mixing. Section 4 describes the protocol and system design of *CoinParty*. Section 5 provides a comprehensive analysis, evaluation, and discussion of the mixing correctness, performance, anonymity, and further system requirements. Finally, Section 6 compares *CoinParty* to related work and Section 7 concludes this paper.

2. BACKGROUND

We briefly cover the relevant background on Bitcoin and SMC as the underlying foundation of our approach.

2.1 Bitcoin

Bitcoin is best understood as a decentralized P2P network that keeps track of all money transfers between its users. Transfers are recorded in a public ledger, the blockchain, which is constantly validated by the Bitcoin participants through a proof-of-work. Double spending of Bitcoins is thereby ruled out as long as the majority of computation power is contributed by honest, non-colluding participants.

Addresses. Bitcoin users can have a virtually unlimited amount of cryptographic identities, called addresses. Addresses are used to store and receive Bitcoins. An address is basically the hash of an ECDSA public key and a user in possession of the corresponding private key is said to *own* the address. Addresses serve as pseudonyms and frequently using fresh ones is the basis for anonymity in Bitcoin.

Transactions. A transfer of Bitcoins between addresses is called a transaction. To issue a transaction, a user specifies one or more *input addresses* $I_1, ...I_n$ from where the transaction amount is collected. The transaction amount can be distributed arbitrarily to one or more *output addresses*, $O_1, ..., O_m$. We formally write $\{I_1, ..., I_n\} \overset{\nu_1,...,\nu_m}{\rightarrow} \{O_1, ..., O_m\}$, for a transaction of ν_i Bitcoins to O_i, respectively, or simply $I \overset{\nu}{\rightarrow} O$ when only one particular input and output address is relevant in the transaction. The user signs the transaction with the private key corresponding to the input address to proof that she indeed owns the respective address. The complete transaction is then broadcasted into the Bitcoin network where it is grouped and validated together with other transactions in *blocks*. The transaction is usually considered valid by the other Bitcoin participants after six further blocks have been processed. Blocks are linearly chained in the blockchain which represents the complete *accepted* history of Bitcoin transactions and has been the primary target for deanonymization attacks.

Transaction fees. The difference between the sum of a transaction's input and output values can be collected as a transaction fee. Transaction fees motivate *Bitcoin miners*, i.e., users who invest computing power to find new blocks, to include the transaction in the blockchain. Fees are not mandatory and many transactions are indeed processed with no fee at all, e.g., those not exceeding a size of 1 KB [7].

2.2 Secure Multi-Party Computation

SMC considers the basic problem of how a group of peers can compute some known functionality $\mathcal{F}(x_1, .., x_n)$ in absence of a trusted third party without anyone learning the private inputs $x_1, ..., x_n$. Among the different proposed solutions, constructions based on linear secret sharing are widely used and we quickly cover the basics here. At the beginning, each input party creates $[x] := (f_x(1), ..., f_x(i), ..., f_x(m))$, a sharing of her private input x where $f_x \in \mathbb{Z}_p[X]$ is a random t-degree polynomial with $f_x(0) = x$. $[x]$ is called a t-out-of-n secret sharing of x, where $[x]_i$ is called the i-th share of x. Any subset of t or less shares does not reveal anything about the secret, while any $t + 1$ or more shares are sufficient to reconstruct x using Lagrange interpolation. t is thus often called the reconstruction threshold:

$$x = f_x(0) = \sum_{i=i_1}^{i_{t+1}} [x]_i \cdot \underbrace{\prod_{j=0 \ j \neq i} \frac{j}{i-j}}_{=:\lambda_i} \tag{1}$$

To realize the desired functionality $\mathcal{F}$ securely, it is first expressed as an arithmetic circuit which is subsequently evaluated by a set of *privacy peers*. The input peers share their secrets to the m privacy peers, i.e., privacy peer P_i receives $[x]_i$, and the privacy peers then compute the required additions and multiplications on these shares, i.e., without learning the secret inputs. In *linear* secret sharing schemes, addition and thus also scalar multiplication can be computed locally, i.e., $[x] + [y] = [x + y]$ and thus $[x] \cdot s = [x \cdot s]$. Multiplication of two secret-shared values $[x]$ and $[y]$ can be implemented in a simple protocol which requires one round of communication between the privacy peers. Based on ad-

dition and multiplication, theoretically any functionality can be implemented, while practically processing and communication overheads limit what is feasible.

Adversary models. Security of SMC protocols is usually analyzed in either the *semi-honest* adversary or the *malicious* adversary model [18]. Semi-honest adversaries, also referred to as passive adversaries, are assumed to follow the protocol correctly, but may analyze the protocol transcript to gain additional information about the participants' private inputs, e.g., to break anonymity. In the *semi-honest* model, we typically set the reconstruction threshold to $t = \lfloor (m-1)/2 \rfloor$ which achieves security against an adversary that corrupts any minority of the m privacy peers. Contrarily, a *malicious* adversary is not bound by the protocol specifications and may actively try to cheat. Because of the involved monetary values, we argue that it is mandatory to provide security against malicious adversaries in the context of our work. Notably, all of the SMC primitives we use in this work are based on Damgård et al.'s protocol for general Secure Multi-Party Computation [14]. Damgård's construction builds on linear secret sharing and is secure against a malicious adversary that corrupts less than $m/3$ of the m privacy peers, i.e., the reconstruction threshold is $t = \lfloor (m-1)/3 \rfloor$. An efficient implementation of [14] exists in the VIFF framework [30] on which we base our protocol implementation.

3. PROBLEM STATEMENT

Motivated by recent work on de-anonymization of Bitcoin transactions [2, 22, 25, 26], we consider the problem of how a user can mix Bitcoins with other users to preserve her financial privacy. Formally, n input peers which each have a certain amount of ν Bitcoins available at input addresses $I_1, ..., I_n$ want to mix that amount to a set of output addresses $O_1, ..., O_n$ such that (1) each input peer receives back ν Bitcoins on her output address, and (2) input and output addresses are unlinkable, i.e., only input peer i knows that I_i and O_i belong together. Essentially, this means that each input peer i issues the transaction $I_i \xrightarrow{\nu} O_{\pi(i)}$ where π is a random and *secret* permutation over $\{1, ..., n\}$. Such a mixing service needs to fulfil the following requirements:

Mixing correctness: Bitcoins must not be lost, stolen, or double-spent by any inside or outside party even in the presence of a malicious adversary. Honest input peers should receive their funds in a timely manner.

Anonymity: The mixing must be anonymous, i.e., a malicious adversary must not be able to create a link between input address I_i and output address O_i for any input peer i.

Deniability: Input peers should be able to plausibly deny having participated in a mixing operation.

Performance: The protocol should scale to large numbers of input peers without imposing prohibitive overheads upon the mixes or the Bitcoin network.

Compatibility: The mixing protocol must be fully compatible with the current Bitcoin network and produce legitimate Bitcoin transactions.

Cost-efficiency: The protocol must be cost-efficient in terms of involved transaction and mixing fees.

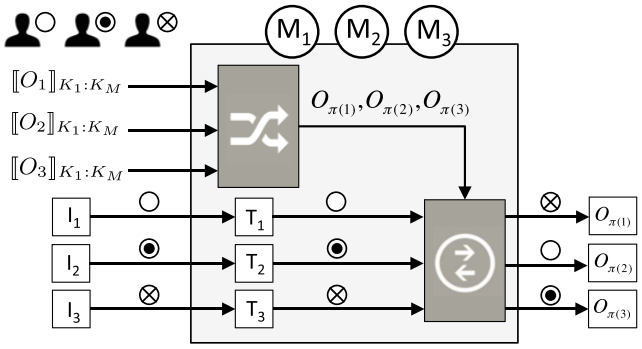


Figure 1: Overview of *CoinParty* with three participants. The shuffling and transaction (grey) are executed collaboratively by the mixing peers M_1, M_2 and M_3.

Clearly, the first generation of centralized mixes [5,6,8,11] does not provide correctness in the presence of a malicious mix that steals funds. Furthermore, anonymity is provided not even against weaker passive adversaries because the mixing service knows the permutation applied to the output addresses. The improvements proposed in [9, 20] protect against theft from malicious adversaries, but users still need to entrust the mix with their anonymity. Recent works on decentralized mixes provide anonymity also in the presence of a malicious adversary corrupting the mixing service itself. However, they are either inefficient [21, 23, 31], uneconomical [21], achieve only suboptimal anonymity and deniability [20, 28], are incompatible with the current Bitcoin network without substantial modifications [3,23], or have not evolved beyond discussions on forums and blogs [16,21,27,31]. Thus, secure and anonymous mixing of Bitcoins is an open problem. In the following, we propose *CoinParty* which combines the advantages of centralized and decentralized approaches to Bitcoin mixing and fulfils the stated requirements.

4. PROTOCOL DESIGN

CoinParty takes a decentralized approach at Bitcoin mixing by introducing a set of mixing peers which emulate a Trusted Third Party through SMC to realize a secure and anonymous mixing of Bitcoins between the participating users. Note that in the context of this work, we refer to the privacy peers executing the SMC protocol as *mixing peers*. Figure 1 gives an overview of one exemplary protocol run of the *CoinParty* system with three participants. *CoinParty* runs in three phases, (1) commitment (Section 4.1), (2) shuffling (Section 4.2), and (3) transaction (Section 4.3). A fourth error and reversion protocol phase (Section 4.4) is invoked when an error or malicious behaviour is detected in the three previous phases. We describe each of these phases in detail now, while deferring discussions of security and anonymity to Sections 5.1 and 5.2.

4.1 Commitment

The goal of the commitment phase is to make the input peers commit the required funds for mixing to a temporary escrow address T_i. Of course, the escrowed funds must be protected against theft by malicious mixing peers. To achieve this, T_i is generated from a fresh *threshold* ECDSA key pair so that T_i is under joint control by the mixing peers and a majority of mixing peers must collaborate to create a valid signature in order to transfer funds. Inspired by the

scheme of Ibrahim et al. for threshold signatures [17], we show how to generate escrow addresses T_i and the corresponding key pair in a distributed fashion so that no central trusted entity is required:

(C1) Using Pseudo-Random Secret Sharing (PRSS) [13], each mixing peer M_j obtains a share $[d_i]_j$ of an unknown random value d_i that represents the private key.

(C2) Each M_j computes locally her share of the public key Q_i: $[Q_i]_j = [d_i]_j \cdot G$, with G the generator of the elliptic curve (i.e., `secp256k1` in the case of Bitcoin).

(C3) Each mixing peer M_j broadcasts her share $[Q_i]_j$ to the other mixing peers.

(C4) Each M_j receives the shares of the other peers and reconstructs the public key $Q_i = \sum_{j=1}^{m} \lambda_j [Q_i]_j = \sum_{j=1}^{m} \lambda_j [d_i]_j G = d_i G$, with λ_j the Lagrange basis polynomial at point $x = 0$.

(C5) Each M_j creates the address T_i from Q_i according to the technical specifications of the Bitcoin protocol.

Mixing peers precompute a set of escrow addresses in advance and announce a different escrow address T_i to each input peer i on demand. If an input peer receives two different escrow addresses $T_i \neq T_i'$, e.g., from a malicious mixing peer that tries to divert funds to her own address, the input peer aborts the protocol immediately and notifies the other mixes of the equivocation. Otherwise, the input peer transfers the required funds ν from her input address I_i to the escrow address T_i, i.e., issues transaction $I_i \xrightarrow{\nu} T_i$. The mixing peers wait the recommended six blocks until the commitment is considered accepted by the Bitcoin network.

4.2 Address Shuffling

The goal of the shuffling phase is to permute the set of output addresses given by the input peers under a secret permutation π, such that nobody, not even the mixing peers, can link input addresses $I_1, ..., I_n$ to the corresponding output addresses $O_1, ..., O_n$. *Verifiable shuffling* is a well-known problem, e.g., in anonymous communications, and can be solved using decryption mixnets as proposed in [10, 12] and applied to Bitcoin mixing in [28]. While our use of decryption mixnets for address shuffling is inspired by [10, 12, 28], we use a different approach to verify the integrity of the shuffling which allows for deniability and improves performance. We first outline our shuffling protocol and then discuss our modifications.

(S1) Each input peer i encrypts and broadcasts her output address O_i using the public keys $K_1, ..., K_m$ of the mixing peers in a layered encryption $[\![O_i]\!]_{K_1:K_m} := E_{K_1}(E_{K_2}(...E_{K_m}(O_i)))$. Also, each input peer i secret-shares the hash $H(O_i)$, i.e., sends $[H(O_i)]_j$ to $M_{j=1..m}$.

(S2) The mixing peers now enter m rounds of decryption and shuffling. M_j removes the outermost decryption E_{K_j}, then applies a private permutation π_j and sends $S^j = [\![O_{\pi_j \circ ... \circ \pi_1(1)}]\!]_{K_{j+1}:K_m}, ..., [\![O_{\pi_j \circ ... \circ \pi_1(n)}]\!]_{K_{j+1}:K_m}$ to the next mixing peer M_{j+1}.

(S3) M_m removes the last layer of encryption E_m. M_m then sorts the output addresses lexicographically (permutation π_m) and broadcasts the resulting S^m.

(S4) Each M_j computes a share of a checksum $[C]_j := [\sum_{i=1}^{n} H(O_i)]_j = \sum_{i=1}^{n} [H(O_i)]$, broadcasts $[C]_j$ to the other mixes, waits for the other peers' shares and then reconstructs C.

(S5) Each M_j validates that (1) S^m is lexicographically ordered and (2) the correctness of the checksum $C = \sum_{i=1}^{n} H(O_{\pi_m \circ ... \circ \pi_1(i)})$ by hashing the addresses in S^m. Otherwise, M_j enters the error and reversion phase.

(S6) On success, each M_j seeds a pseudo-random number generator (PRNG) with the checksum C to obtain a common final permutation π_{m+1} that is applied to S^m to get $S^{m+1} = O_{\pi(1)}, ..., O_{\pi(n)}$ with $\pi := \pi_{m+1} \circ \pi_m \circ ... \circ \pi_1$.

Though our shuffling protocol is inspired by the related work [10, 12, 28], we introduce two core modifications. First, we use secret sharing to validate the correctness of the shuffling in Steps (S4) and (S5). This obsoletes involving input peers in the verification of the shuffling which strengthens anonymity guarantees and allows for deniability as analyzed in Sections 5.2 and 5.3. Second, in [28] the last mixing peer controls the outcome of the shuffling which is undesirable as we explain in Appendix A. In [10, 12] this is prevented at the costs of a second encryption layer. Our Steps (S3) and (S5) prevent M_m from controlling the shuffling while Step (S6) ensures that the final shuffling is indeed random. We thus fix this vulnerability of the shuffling protocol in [28] while maintaining its superior performance compared to [10, 12].

4.3 Transaction

In the transaction phase, the mixing peers create transactions $T_i \xrightarrow{\nu} O_{\pi(i)}$ which requires them to compute one ECDSA signature per transaction. Since the private key d_i corresponding to an escrow address T_i is *shared* across the mixing peers, the standard ECDSA algorithm cannot be used. Instead, the mixing peers need to collaboratively sign the transaction to spend any of the funds located at T_i. We employ a threshold variant of the ECDSA algorithm according to Ibrahim et al. [17], which we use to create and sign a Bitcoin transaction as follows:

(T1) The mixing peers use PRSS [13] to obtain shares $[k]_i$ of an unknown random value k. They use the reciprocal protocol from [17] to obtain $[k^{-1}]_i$.

(T2) Each mixing peer M_j creates $e = \texttt{SHA-1}(T_i \xrightarrow{\nu} O_{\pi(i)})$ according to the Bitcoin protocol specifications.

(T3) Using the generator of the curve G, each M_j computes $[kG]_j = [k]_j G$ and broadcasts her share $[kG]_j$. M_j receives the other peers' shares and reconstructs kG.

(T4) With $(x, y) := kG$ and $R := x \bmod n$, the mixing peers compute $[S] = [k^{-1}] * (e + [d] \cdot R)$ and reconstruct S.

(T5) The mixing peers output the ECDSA signature (R, S) and with it build and broadcast $T_i \xrightarrow{\nu} O_{\pi(i)}$.

We introduce the following modifications to improve the performance and robustness compared to [17]: The mixing peers precompute the first part of the signature R, e.g., along with the precomputation of the escrow addresses T_i. This only requires mixing peers to precompute $[k]$, its reciprocal

$[k^{-1}]$ (T1) and then kG (T3). We further split the computation of the second signature part S (T4) into two parts, i.e., $[S] = [k^{-1}] * e + [k^{-1}] * [d] \cdot R$. Then, also the critical multiplication on shares $[k^{-1}] * [d] \cdot R$ can be precomputed. To create the actual signature, the mixing peers now only need to compute locally the hash e (T2), the scaler multiplication $[k^{-1}] \cdot e$ (T4) and then reconstruct $[S]$. Thus, signature generation becomes much faster and more robust against halting or failing peers.

4.4 Error and Reversion

When an error is detected during the commitment or shuffling phase, the mixing peers transfer all funds from the escrow addresses T_i back to the input addresses I_i. The necessary steps are the same as in the transaction phase. As we analyze in Section 5.1, these transaction are guaranteed to succeed even in the presence of malicious adversaries such that no funds can be stolen, diverted, or lost.

Detecting a malicious mix that announces a different $T_i' \neq T_i$ to input peer i in the commitment phase is straightforward. It is slightly more difficult to hold mixing peers accountable for malicious behaviour during the shuffling phase. However, as this has been previously explained in [12, 28], we restrict ourselves to briefly sketch the basic idea. Other than for mixnets used for anonymous communications [12], the shuffled messages, i.e., the output addresses, need not be kept secret in case of an error. If the mixing fails, input peers simply dispose of the potential contaminated, yet unused output addresses O_i. They can then reveal the randomness used to produce the layered encryptions $[\![O_i]\!]_{K_1:K_m}$ which allows honest mixing peers to trace the malicious mix by successively reconstructing the intermediate shufflings S^j. Because mixing peers need to sign all their messages, they can then be held accountable for their wrong doing. Note that as a consequence input peers need to generate fresh output addresses for the next mixing, which is accpeted good practice even for correct protocol runs.

5. DISCUSSION OF SYSTEM PROPERTIES

In this section, we show that *CoinParty* fulfils the requirements presented in Section 3. We explain why *CoinParty* achieves a random and correct mixing even in the presence of malicious adversaries in Section 5.1. In Section 5.2 we quantitatively evaluate the achieved anonymity. This is also the basis for our discussion of deniability in Section 5.3. We present a comprehensive performance evaluation of *CoinParty* in Section 5.4. Finally, Sections 5.5 and 5.6 briefly discuss compatibility and cost-efficiency of *CoinParty*.

5.1 Mixing correctness

To prove the correctness of our protocol in the presence of malicious adversaries, we now show that each protocol phase, i.e., commitment, shuffling, and transaction, is either completed correctly or an error is detected and funds are restored to the input addresses of the participants.

5.1.1 Commitment phase

Malicious Input Peers. In the commitment phase, input peers transfer the required mixing amount to the escrow addresses T_i. The correctness of the commitment phase only depends on the correctness of the transactions $I_i \overset{\nu}{\rightarrow} O_i$, which is ensured by the Bitcoin network itself and independent of our system. Further, we note that input peers cannot stall this protocol phase, since refusal to commit funds just means that the mixing starts without the particular input peer. An input peer could DoS our system by repeatedly requesting escrow addresses, however this can be thwarted using standard puzzle mechanisms. Thus, the commitment phase is secure against any number of malicious input peers.

Malicious Mixing Peers. A malicious adversary that controls a fraction of the mixing peers can try to corrupt the ECDSA key generation and with it the generation of escrow addresses T_i. However, to implement the key generation we use the general SMC protocol from [14], which is secure against a malicious adversary that corrupts less than $m/3$ of the mixing peers. In other words, the protocol in Section 4.1 will generate a valid address T_i with the honest mixing peers holding consistent shares of the corresponding private key d_i, even if one third of the mixing peers are compromised and behave arbitrarily, e.g., submit wrong inputs. Thus, the commitment phase is secure against an $m/3$ malicious adversary.

5.1.2 Shuffling phase

We show that the shuffling finishes and that its integrity holds even in the presence of a malicious adversary. Unlinkability and randomness of the shuffling are proved in Section 5.2. We define *integrity* analogous to [10]: Either exactly the given output addresses are contained in the final shuffling, or the honest mixing peers are informed that some input peer's output address has been substituted. Note that though our shuffling phase is inspired by [10,12,28], our method of verifying the integrity of the shuffling is different and thus requires a dedicated proof of correctness.

Malicious Input Peers. Input peers only broadcast a layered encryption of their output address and secret-share its hash value. Other than related work based on decryption mixnets [10, 12, 28], the input peers are not involved in the verification of the shuffling. Malicious input peers have no incentive in announcing a wrong or broken address, since this would only result in the loss of their own funds. However, malicious input peers can share out a wrong hash value in order to mount a DoS attack against the verification step. In this case, the protocol proceeds by transferring back the funds to the input addresses, which is the same as one run of the transaction phase and thus not further analyzed here. While such malicious behaviour of input peers can be traced back to the peer, e.g., by reconstructing the hashes from the sharings $[H(O_i)]$ on error, it cannot be prevented. We note that related work [10, 12, 28] is also vulnerable to such DoS attacks by single malicious input peers. However, unlike related work which involves the input peers in the verification of the shuffling, our approach is robust against halting input peers or random failures.

Malicious Mixing Peers. Any malicious mixing peer M_j can substitute the outputs in the shuffling with her own output addresses by encrypting them with the known public keys of the remaining mixes and announcing $S'^j = [\![O_1']\!]_{K_{j+1}:K_m}, ..., [\![O_m']\!]_{K_{j+1}:K_m}$. However, this will be detected in (S5) unless the attacker finds $O_1', ..., O_m'$ with $C = \sum_{i=1}^{m} H(O_i) = \sum_{i=1}^{m} H(O_i')$. For all but the last mixing peer, this is clearly infeasible since M_j, $j < m$, needs to announce the shuffling S^j before even learning C in (S4).

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 *before* announcing the shuf-

fling S^m. To steal the mixed funds, M_m must thus find suitable output addresses $O'_1, ..., O'_n$ that sum up to the same checksum C. Since the attacker can generate an arbitrary amount of addresses, this corresponds to solving a high density *Random Modular Subset Sum* (RMSS) problem, which is likely to have a solution. However, we show in Appendix B that large problem instances as involved in *CoinParty* are not practically solvable in reasonable time and thus the attack is thwarted by limiting the time for Step (S3). We can also prevent the attack on the protocol level by introducing random nonces into the hashes, i.e., the input peers share $[H(O_i|n_i)]$ in (S1). Since the nonces n_i link in- and output addresses, the nonces need to be encrypted as well. By shuffling and decrypting them only *after* mixing peer M_m has committed to the final shuffling S^m in (S3), M_m cannot predict the checksum C and mount the RMSS attack anymore. Note that this comes at the cost of a complete shuffling.

The chances of the attacker to recover C do not increase, even if he can compromise up to $n-1$ of the input peers and up to t mixing peers, where $t+1$ is the reconstruction threshold of the secret sharing scheme. Aside from substituting addresses, a malicious attacker can announce an incorrect share of C in Step (S4) to let the verification fail. As in the commitment phase, we can tolerate up to $t < m/3$ inconsistent shares of malicious mixing peers and still correctly reconstruct C. We conclude that the shuffling phase is secure against an $m/3$ malicious adversary.

5.1.3 Transaction phase

The transaction phase does not involve input peers and thus we focus on malicious mixing peers. Steps (T1), (T3), (T4), and (T5) of the transaction phase involve only addition, multiplication, and reconstruction of shares. Thus, as with the commitment phase, the security of theses steps against a $m/3$ malicious adversary follows directly from the security guarantees of Damgård's general SMC protocol [14]. Computing a wrong hash $m' \neq m$ in Step (T2) and using it to construct S in Step (T4) results in inconsistent shares of S. As long as no more than $m/3$ shares are inconsistent, the inconsistent shares can be filtered out using techniques from error detection codes. Since, our modifications shift all critical operations to the precomputation phase, the transaction phase succeeds even if up to $2m/3$ mixing peers fail or halt the protocol, i.e., without announcing inconsistent shares, because reconstruction requires only $t+1 = \lfloor (m-1)/3 \rfloor + 1$ consistent shares. Finally, a malicious peer could announce an incorrect signature and thus an invalid transaction in Step (T5), which will then be rejected by the Bitcoin network. However, since the protocol finishes correctly under the mentioned assumptions, the remaining honest mixing peers will announce the correct transaction to the Bitcoin network.

5.2 Anonymity

We first argue why *CoinParty* performs a random and unlinkable shuffling and then analyse in detail which level of anonymity is guaranteed by this shuffling.

5.2.1 Unlinkability and randomness

We note that unlinkability and randomness depend only on the shuffling phase (Section 4.2). If there is an error in this phase, the protocol enters the error and reversion phase (Section 4.4) and funds are restored to the inputs while the output addresses are considered burnt and are discarded. Thus, errors during the shuffling phase have no impact on unlinkability and we only need to consider correct runs of this phase in our anonymity analysis.

The proof for unlinkability is then basically the same as for Brickell and Shmatikov [10] and we only sketch it here. The argument for unlinkability of in- and output addresses is that by using an encryption scheme E which is IND-CCA2 and length regular, the ciphertexts $[\![O_{i=1...m}]\!]_{K_j:K_m}$ are *indistinguishable*. Concretely, the attacker, given the public key K_j of mix M_j, cannot decide which ciphertext $[\![O_{i'=1...m}]\!]_{K_j:K_m}$ corresponds to the encryption of $[\![O_{i=1...m}]\!]_{K_{j+1}:K_m}$. Thus, the attacker cannot link ciphertexts in S^j and S^{j+1}, i.e., he cannot observe the permutation π_j applied by M_j. Furthermore, the decryption mixnet ensures participation of all mixing peers in the shuffling, i.e., in particular that M_j cannot be skipped. Hence, a single honest mixing peer can ensure the unlinkability of the shuffling.

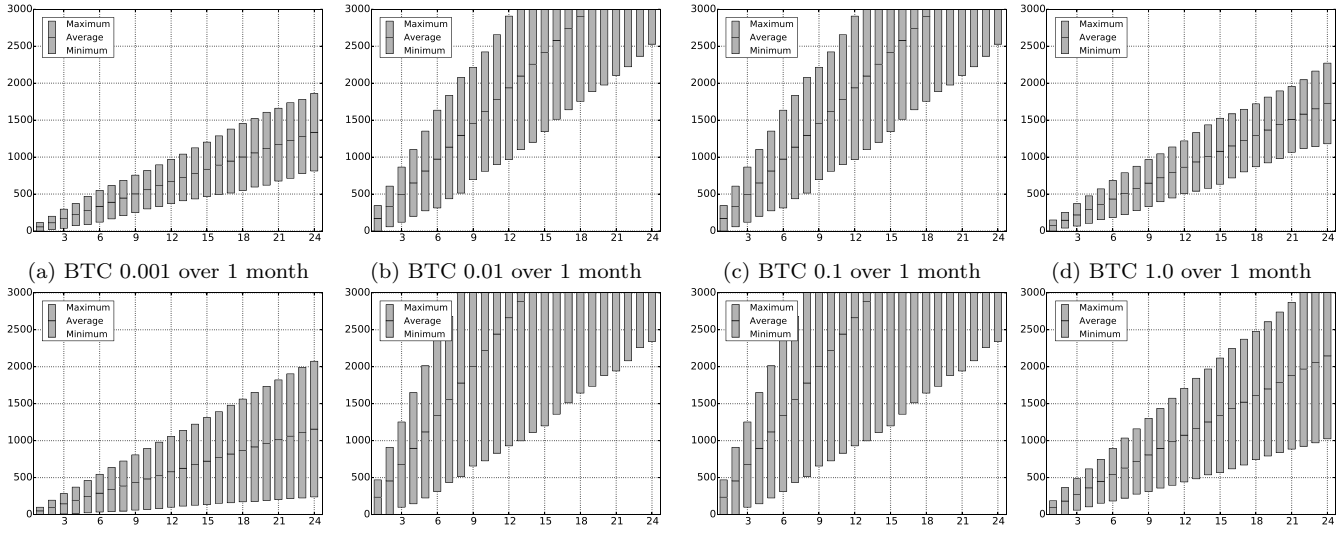
We now show that the shuffling is random. The permutations $\pi_{j<m}$ only ensure that the last mix M_m decrypts the output addresses under an unknown shuffling, but do not contribute towards the randomness of the shuffling since the permutation π_m applied by M_m is a lexicographic ordering and thus fixed. The lexicographical ordering is verifiable by all other peers, thus preventing any attempt of M_m to undermine the randomness of the shuffling (cf. Appendix A). The source of randomness is the final permutation π which is obtained from a PRNG seeded with the checksum C, i.e., the sum over the hashes of all output addresses. Thus, if at least one honest input peer chooses a random output address, C is random within the range of the hash function and hence π is a also random. We conclude that *CoinParty* performs a random and unlinkable shuffling as long as at least one input and one mixing peer honestly follow the protocol.

5.2.2 Anonymity Level

Based on observations of the blockchain an attacker can try to guess the mapping between a mixing participant's input and output address. The set of addresses among which the attacker has to guess is the *anonymity set* and its size the achieved *anonymity level*. A larger anonymity set leads to a smaller probability of a correct guess and hence more anonymity. In the following, we analyze an input peer's anonymity against (i) outside attackers who only observe the blockchain, (ii) other input peers, and (iii) the mixing peers.

Outsiders. Our threshold transactions are indistinguishable from standard Bitcoin transactions. Thus, *CoinParty*'s mixing transactions can only be identified as such by (i) their correlation in time and (ii) the reoccurring output value of ν Bitcoins, i.e., the mixing amount. Generally, *CoinParty* produces mixing transaction chains of length two: First, input peer i commits at least ν funds to the escrow address T_i during $[t_0, t_1]$ in one transaction $I_i \xrightarrow{\nu} T_i$. In the transaction phase $[t_1, t_2]$, ν Bitcoins from T_i are mixed to another participant's output address $O_{\pi(i)}$ in a second transaction $T_i \xrightarrow{\nu} O_{\pi(i)}$. Note that the mixing amount ν is known to the input peers 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.

To establish a *lower bound* for the anonymity level, we first assume that an attacker is able to distinguish the n



(a) BTC 0.001 over 1 month (b) BTC 0.01 over 1 month (c) BTC 0.1 over 1 month (d) BTC 1.0 over 1 month

(e) BTC 0.001 over 1 year (f) BTC 0.01 over 1 year (g) BTC 0.1 over 1 year (h) BTC 1.0 over 1 year

Figure 2: Size of the anonymity set (minimum, average, and maximum) for different mixing windows $w = 1, ..., 24$ hours and mixing values $\nu = 0.001, 0.01, 0.1, 1.0$ BTC in June 2014 (top (a)-(d)) and from June 2013 to July 2014 (bottom (e)-(h)).

mixing transaction chains produced by *CoinParty* from all other transactions in the blockchain during the time frame $[t_0, t_2]$ for commitment and transaction. The attacker however cannot distinguish between the n transactions belonging to the same mixing operations. Thus, the anonymity level against outsiders is at least n, or even $k \cdot n$ if there are k contemptuous mixing operations with n participants each.

We now show that our approach of mixing in multiple one-to-one transactions instead of using one single group transaction significantly increases the anonymity set to sizes similar to those realized by the early centralized mixing services. We note that input peers can vary the transaction amount in the commitment phase by transferring arbitrary amounts $\nu' \geq \nu$ to T_i (they just receive the leftovers $\nu' - \nu$ after the mixing on a fresh change address). The commitment $I_i \rightarrow T_i$ thereby becomes indistinguishable from all other transactions in $[t_0, t_1]$ with an amount $\geq \nu$. However, we cannot hide the tell-tale ν transaction $T_i \stackrel{\nu}{\rightarrow} O_{\pi(i)}$ during the transaction phase $[t_1, t_2]$ with the same trick, since our current design of the shuffling phase limits us to mix the same amount ν between all mixing participants. The size of the anonymity set for the transactions $T_i \stackrel{\nu}{\rightarrow} O_{\pi(i)}$ is thus the number of other transactions with an output value of ν BTC in the time frame $[t_1, t_2]$. We have analyzed the transactions in the blockchain between June 2013 and July 2014 (long term) and in June 2014 (short term) in order to determine how to sensibly choose ν to provide a high anonymity level. We choose a long and short observation period to show that results are consistent for the past and present and are thus also good indication for the future. Although some differences can be observed, $1, 0.1, 0.01$, and 0.0001 BTC are notably popular output values in both time spans. E.g., there are 120 318 transactions for 0.01 BTC in June 2014. These values are promising choices for ν.

Even when using these popular values, releasing all mixing transactions at the same time would render them easily distinguishable from non-mixing transactions by their strong correlation in time. Thus, the length of the transaction phase $[t_1, t_2]$ over which transactions are released to the Bitcoin network, referred to as *mixing window*, has to be chosen reasonably in order to hide *CoinParty*'s mixing transactions among normal Bitcoin transactions. We again analyzed the blockchain to quantify how much different mixing windows increase the anonymity level. Figure 2 plots the minimum, average, and maximum anonymity level for mixing windows of 1 up to 24 hours for the four popular transaction values $1, 0.1, 0.01$, and 0.001 BTC. We moved a sliding window of the respective size, i.e. 1 to 24 hours, over the blockchain and calculated minimum, maximum, and average over all window positions. We observe that increasing the mixing window greatly increases the achieved minimum, average, and maximum anonymity levels. Already a mixing window of 3 hours provides for nearly all cases (except (e)) a minimum additional anonymity level > 0 and an average additional anonymity level of 100 to 500. Results for the long (past) and short (present) observation period are consistent and thus provide a good indication for choosing mixing windows also in future. It is important to note that these anonymity levels, denoted by N, are *in addition* to the anonymity level of the mixing operation itself, i.e., the lower bound established above. Thus, even in case (e) with a mixing window of 3 hours where non-mixing transactions do not necessarily offer additional anonymity, i.e., $N = 0$, the anonymity level is still at least $k \cdot n$, if k parallel mixings each with n participants take place within this mixing window.

Input peers. If all input peers are honest, anonymity of one input peer against the others is practically the same as for protocol outsiders, since input peers do not know which other input peers participate in the shuffling. We achieve this by introducing dedicated mixing peers and modifying the verification of the shuffling to not require involvement of any input peer (Section 4.2), unlike related work [28].

The case is different, when an attacker compromises c of the n input peers. He can then distinguish a mixing transaction from the other non-mixing transactions in the blockchain, if one of the compromised input addresses is mapped to an honest input peer's output address or vice versa. This allows the adversary to tie one transaction with probability

$p = 1 - (1 - c/n)^2$ to the mixing. A plot of p versus c/n is included in Appendix C, e.g., showing that the attacker needs to compromise only $c/n = 50\%$ of the input peers to identify 75% of the mixing's transactions. We now assume the attacker has thereby tied input address I_i to the mixing and now tries to link it to the corresponding output address O_i. It can be showed using basic probability theory that in our approach the attacker then has a success probability of $p' = p/(n-c) + (1-p)/(n-c+N')$ where $N' := N/(1-p)$ to guess the corresponding output address. In related work this probability is $1/(n-c)$. Our scheme thus provides equal anonymity for $N' = 0$ and is better for $N' > 0$. To give an example, for $n = 100$ participants of which $c/n = 10\%$ are compromised, the attacker guesses correctly with $p = 0.011$ for the related work and only $p = 0.003$ for our scheme with $N = 500$ (e.g., for a mixing amount of 0.01 or 0.1 BTC and a mixing window of at least 3 hours according to Figure 2).

Such sybil attacks seem endemic to Bitcoin mixes where participation is free or very cheap (cf. Section 5.6): The adversary can use the mixing service to generate untraceable Bitcoin addresses which can then be used as input addresses to attack anonymity in another mixing. An obvious solution is to make such attacks expensive by charging mixing fees, resulting in a trade-off with our *cost efficiency* requirement.

Mixing peers. The *mixing peers* inevitably learn which input and output addresses are involved in the mixing operation, as they have to sign the corresponding transactions and release them to the Bitcoin network. However, since mixing peers do not learn which output belongs to which input address as proved in Section 5.2.1, the anonymity level against mixing peers is equal to the number of participants in the mixing n which is as good as the related work [28, 31] and significantly better than [9, 20].

We conclude our analysis with the remark that our results are consistent with the results from the field of anonymous communications: Mixing inputs and outputs must occur over a sufficiently long time span, the longer the time span the higher the achieved level of anonymity. Here, our results provide concrete indication that time windows around 3 hours are reasonable and practical for Bitcoin mixing services. More importantly, the results show that *CoinParty* can provide levels of anonymity that are orders of magnitude higher than those achieved by related work [20, 28, 31], even when a fraction of the input peers is compromised.

5.3 Deniability

A user can simply deny having participated in a mixing if she can plausibly deny owning a Bitcoin address. However, the recent works 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 mixing services. Thus, instead we need to analyze whether a user can plausibly deny that one of her addresses was part of a mixing operation.

Outsiders. Deniability against outsiders is achieved by the indistinguishability of mixing and normal transactions. 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.

Input peers. In our anonymity analysis, we have showed that an adversary that corrupts c input peers can tie a fraction of $p = 1 - (1 - c/n)^2$ of the transactions to the mixing. Thus, an honest input peer is not bound to the mixing with probability $1-p = (1-c/n)^2$ and can deny her participation. Again, we refer to Appendix C for a plot of c/n versus p. Note that due to our modifications to the shuffling protocol (cf. Section 4.2, Appendix A) it is ensured that the shuffling is random. Thus a targeted attack on a specific input peer only succeeds with probability p.

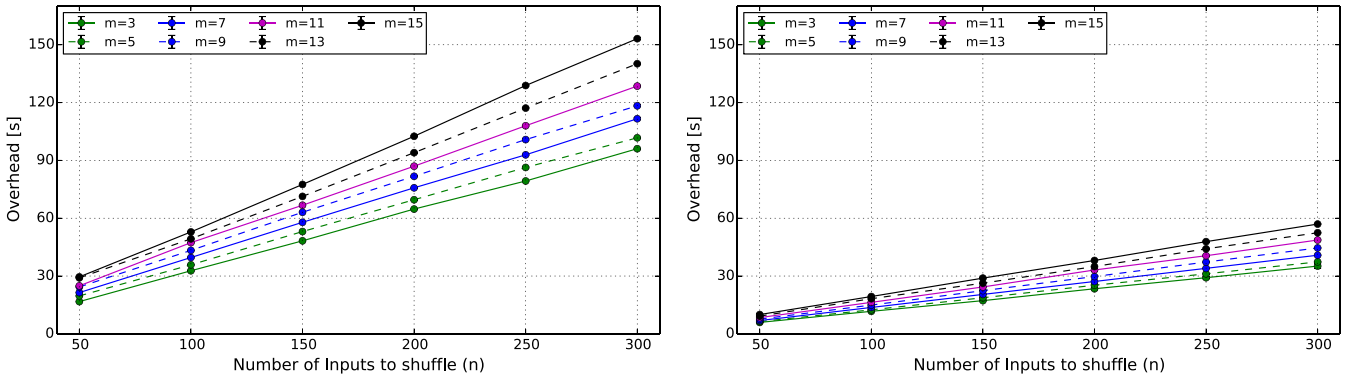
Mixing peers. Deniability against mixing peers is not achieved because they learn which in- and output addresses participated in the mixing during the shuffling phase. Achieving deniability against the mixes would require blindly signing transactions. The cryptographic primitives for blind signatures exist but we consider this future work.

5.4 Performance Evaluation

In this section, we analyze whether *CoinParty* fulfils the *performance* requirement as presented in Section 3. To this end, we present a quantitative evaluation of our prototype in a real world network setting. A qualitative analysis is presented in Appendix D due to space constraints. Input peers in the *CoinParty* protocol only need to compute a small constant amount of ECDSA signatures and encryptions of their output addresses during the commitment phase. We include a brief evaluation of their overhead in Appendix E and concentrate our analysis here on the mixing peers which shoulder most of the overhead. We also consider only successful protocol runs, as the performance of the error and reversion phase (excluding mechanisms for tracing malicious peers) is the same as for a successful transaction phase run.

Quantitative Analysis. We have implemented a prototype of *CoinParty* in Python 2.7 based on the *VIFF* SMC framework [30] which provides primitives secure against malicious adversaries. We use the *Elliptic Curve Integrated Encryption Scheme* (ECIES) over the `secp256k1` curve as encryption scheme E in the shuffling phase. ECDSA is used for signatures on protocol messages in the shuffling phase. VIFF uses the asynchronous communication framework *Twisted* to handle communication between peers while communication in the shuffling phases is implemented using separate TCP sockets between each pair of mixing peers. 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* [15].

We have carried out an extensive evaluation on Microsoft's Azure Cloud, varying the number of mixing and input peers. Each mixing peer runs in a small (A1) instance with one virtual core and 1.75 GB RAM running Ubuntu 12.04 LTS. In all evaluation settings, virtual machines are distributed over different geographical locations in Western and Northern Europe as well as Western and Eastern US. The pairwise round trip times (RTTs) between mixing peers are 50 - 100 ms for Europe $\leftrightarrow$ Europe and US $\leftrightarrow$ US as well as 150 - 200 ms for Europe $\leftrightarrow$ US. Bandwidth measurements range from 5 to 15 MBit/s. Each evaluation setting is aggregated over 10 runs. We repeated all experiments on a single machine (32 cores, 32 GB RAM, Ubuntu 12.04 LTS), running each mixing peer as a separate process on one of the cores and all communication between peers over the local loopback interface. The experiments in the local setting serve



(a) Performance in the cloud network setting.

(b) Performance in the local network setting.

Figure 3: Performance in the cloud (left) and local network setting (right) for 50 to 300 input and 3 to 15 mixing peers.

to distinguish time spent for processing from time spent for communication which is difficult to measure separately due to the asynchronous communication model in VIFF.

Figure 3 shows the time for the complete protocol, i.e., including precomputations, for both the cloud (a) and the local network setting (b). We show the performance for $m = 3, 5, 7, 9, 11, 13, 15$ mixing peers, respectively, running a mixing over $n = 50, 100, 150, 200, 250, 300$ inputs including the standard deviation which was clearly below 1 s for all experiments. The results show that the runtime scales approx. linearly with the number of inputs to mix. Furthermore, the runtime is higher and grows stronger in n if a larger number of mixing peers m is chosen due to the increased number of communication rounds. The comparison with the local setting clearly shows that at least half of the overhead is due to communication. In all settings, *CoinParty* finishes in less than three minutes which clearly shows that *CoinParty* is feasible in real world settings. The online runtime of *CoinParty* is even faster because approx. 70 % of the runtime in both settings are spent on precomputations. In all settings, the communication overhead ranges from a few hundreds of KBs to a few MBs per mixing peer. Hence, we conclude that *CoinParty* fulfils the stated performance goals.

5.5 Compatibility

CoinParty deviates from standard Bitcoin clients in two ways: First, *CoinParty* implements a distributed generation of Bitcoin addresses with shared private keys while in standard Bitcoin clients addresses are generated locally and the client owns the complete private key. Apart from the correct format, generating a valid Bitcoin address thus boils down to generating a correct ECDSA key pair. We have reconstructed the private keys and tested that the generated private and public keys form correct ECDSA key pairs. Second, *CoinParty* replaces the standard ECDSA signature with a threshold ECDSA signature computed collaboratively by the mixing peers. In order to form correct Bitcoin transactions, the generated signatures need to be verifiable using the public key corresponding to the escrow addresses T_i which we have successfully tested. Thus, transactions created by *CoinParty* are fully compatible with the Bitcoin protocol.

5.6 Cost efficiency

A Bitcoin mixing service can principally involve two kinds of fees, (i) transaction fees which are paid for including the issued mixing transactions in the blockchain and (2) mixing fees demanded by the mixing services themselves.

Transaction fees. Unlike other decentralized Bitcoin mixing protocols, *CoinParty* does not bundle mixing transactions into one joint transaction with many inputs and outputs. Instead, *CoinParty* issues one transaction with one input and a few outputs per mixing input, i.e., n separate transactions. As of today, Bitcoin transactions smaller than 1 KB can be safely sent without fees [7]. Transactions in *CoinParty* do not exceed this size and would not require any transaction fees at all. If desired, transaction fees τ can be paid nevertheless. Input peers then commit at least $\nu + \tau$ funds to T_i but only ν funds are transferred in the mixing transaction $T_i \xrightarrow{\nu} O_{\pi(i)}$.

Mixing fees. Anyone can set up a *CoinParty* mixing peer and collaborate with others to provide mixing services. Even the input peers themselves can function as mixing peers. Thus, *CoinParty* does not require any third party services that could charge mixing fees. On the other side, as we note in Section 5.2, it might be reasonable to charge fees to make sybil attacks expensive money-wise. Mixing fees must be handled with care so that they do not identify mixing transactions which would decrease anonymity and chances of deniability. This effect and possible solutions have been discussed in detail in [9].

6. RELATED WORK

In the following we analyze previously proposed approaches to secure and anonymous Bitcoin mixing and highlight the differences to *CoinParty*, as summarized in Table 1.

1^{st} *Generation Bitcoin mixes* [5,6,8] operate centrally and do not provide any guarantees that Bitcoins are actually mixed or even returned. Furthermore, a centralized mix can link input to output addresses and could be forced to reveal this information, e.g., by subpoena. Most of the 1^{st} *Generation mixes* also demand significant mixing fees. However, mixing transactions cannot be distinguished from non-mixing transactions, if the mixing service provides fresh addresses for each input. Thus, deniability is provided against outsiders but not against the mix itself. Also, sybil attacks can be mounted by malicious participants to decrease anonymity as shown in Section 5.2.

Mixcoin [9] still depends on a centralized mixing service but introduces a mechanism to hold a mix accountable if funds are not returned. However, anonymity against the

Approach	Security	Anonymity Level			Deniability	Mixing Delay	Fees	Comp.
1^{st} *Generation* [5, 8]	None	0	$>> n$	$>> n$	Yes	1 window	Mix fees	Yes
Mixcoin [9]	Accountability	0	$>> n$	$>> n$	Yes	1 TX + 1 window	Mix fees	Yes
CoinJoin [20]	Group TX	0	$n - c$	n	No	1 TX	TX fees	Yes
CoinShuffle [28]	Group TX	n	$n - c$	n	No	1 TX	TX fees	Yes
SMC [27, 31]	Group TX	n	$n - c$	n	No	1 TX	TX fees	Yes
ZeroCash [3, 23]	ZKPs	∞	∞	∞	Yes	-	-	No
CoinParty	2/3 honest	n	$> n-c$	$n+N$	$(1 - c/n)^2$	2 TX + 1 window	None	Yes

Table 1: Comparison of our *CoinParty* system to the related works by the most important design requirements. The denoted anonymity level denotes the anonymity an input achieves against (1) the mixing peers (2) c of n malicious input peers, and (3) outsiders, where N depends on the mixing windows as analyzed in Section 5.2.

mix is not provided and the authors propose as a fix to chain multiple mixes. While this potentially achieves anonymity levels similar to *CoinParty*, chaining mixes incurs significant mixing fees and increases the risk of theft as well as the mixing delay. Transactions issued by Mixcoin are indistinguishable from those created by 1^{st} generation Bitcoin mixes and thus Mixcoin provides the same means for deniability.

CoinJoin [20] was first to achieve security against stealing mixes by using group transactions. Still, CoinJoin depends on a central service to shuffle the output addresses and thus provides no anonymity against insiders. Furthermore, the use of group transactions limits the anonymity level to the number of participants n, prevents plausible deniability and requires transaction fees for larger mixing groups.

CoinShuffle [28], like CoinJoin, uses group transactions to ensure correctness and thus shares the corresponding disadvantages, i.e., limited anonymity levels, no deniability, and potential transaction fees. CoinShuffle improves over CoinJoin by using decryption mixnets for address shuffling which achieves anonymity against insiders. However, in CoinShuffle the last peer is in the unique position to determine the outcome of the shuffling and might exploit this to select preferred input addresses to her own outputs addresses. Our proposed shuffling protocol fixes this issue.

The applicability of SMC to Bitcoin mixing was first recognized by member *hashcoin* of the *bitcointalk* forum [16] who brought up the crude idea to use SMC for shuffling output addresses. Later, Rosenfeld [27] and Yang [31] elaborated on this idea in blog posts. However, the presented schemes do not scale with the number of inputs and have not been thoroughly specified. Furthermore, all of the proposed schemes rely on group transactions with the mentioned disadvantages w.r.t. anonymity, deniability, and costs.

Finally, *ZeroCoin* [23] and later *ZeroCash* [3] have been proposed which basically replace Bitcoin's public transaction history by Zero-Knowledge Proofs (ZKPs) to ensure validity of transactions. A payment in *ZeroCash* is fully unlinkable since it reveals neither the origin or destination nor the amount of the transaction. However, those approaches are extensions to Bitcoin that cannot be deployed without significant modifications to the Bitcoin system.

Compared to related work, *CoinParty* introduces several improvements. First and most important, *CoinParty* leverages threshold ECDSA signatures for Bitcoin mixing, instead of depending on group transactions. This allows the mixing to take place in one-to-one transactions which increases the achievable anonymity level by orders of magnitude as analyzed in Section 5.2 and additionally provides plausible deniability against outsiders (Section 5.3). Second, due to the improved shuffling verification scheme the increased anonymity level and deniability are, with some restrictions, also guaranteed against malicious input peers, other than in *CoinShuffle* where input peers actively participate in the protocol. Finally, using several small transactions as opposed to one large group transaction requires no transaction fees at all. The mixing delay is slightly larger than for *CoinJoin, CoinShuffle*, and the approaches by Yang and Rosenfeld which require only one transaction. Especially, waiting the desired mixing window $[t_1, t_2]$, as analyzed in Section 5.2, increases the mixing delay. However, we believe this can and must be tolerated if strong anonymity is desired. Compared to *MixCoin* which provides a similar anonymity level when chaining multiple mixes, *CoinParty*'s mixing is by orders of magnitude faster and fully secure.

7. CONCLUSION

Different works successfully attacked anonymity of Bitcoin addresses by analyzing transactions in the blockchain [2, 22, 25, 26, 29]. Those works make evident the necessity of mixing services and several such systems have been recently proposed [9, 20, 28]. However, a detailed analysis reveals disadvantages with each of those systems (Sections 3 and 6).

In this paper, we have thus presented *CoinParty*, a novel mixing service for Bitcoins that improves significantly over the related work by combining the advantages of centralized and decentralized mixes in a single system. *CoinParty* achieves this by employing two existing building blocks, i.e., threshold ECDSA [17] and decryption mixnets [12, 28]. We introduce several modifications that improve on robustness, anonymity, and deniability that are relevant also beyond the scope of our work. Importantly, we show by analyzing the actual Bitcoin blockchain how our single transaction pattern provides anonymity by orders of magnitude higher than what is achieved by the group transaction pattern that the majority of related work depends on. An extensive evaluation of our implemented prototype demonstrates that *CoinParty* scales better than related work, e.g., [28], due to the possible separation of input and mixing peers. The threshold ECDSA scheme implemented as part of the prototype is efficient and secure against malicious adversaries and can be used beyond our work, e.g., for securing Bitcoin wallets. 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.

Acknowledgments

This work has been co-funded by the Excellence Initiative of the German federal and state governments and the German Research Foundation (DFG) CRC 1053 "MAKI".

8. REFERENCES

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

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

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

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

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

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

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

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

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

[10] J. Brickell and V. Shmatikov. Efficient Anonymity-Preserving Data Collection. In *SIGKDD*. ACM, 2006.

[11] CoinSplitter, 2014. http://coinsplitter.org/.

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

[13] R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *Theory of Cryptography*. Springer, 2005.

[14] I. Damgård et al. Asynchronous Multiparty Computation: Theory and Implementation. In *PKC*. Springer, 2009.

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

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

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

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

[19] V. Lyubashevsky. The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In *RANDOM*. Springer, 2005.

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

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

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

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

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

[25] F. Reid and M. Harrigan. An Analysis of Anonymity in the Bitcoin System. In *PASSAT*, 2011.

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

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

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

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

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

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

APPENDIX

A. RANDOMNESS OF SHUFFLING

The randomness of the performed shuffling (Section 4.2) is an important property. If an attacker can control the outcome of the shuffling, he can (i) select which output address receives funds from which input address and (ii) break deniability for specific users in a targeted fashion (Section 5.3). As we show, the closely related CoinShuffle [28] does not achieve a random shuffling. After the first $m-1$ shuffling steps in CoinShuffle, the mixing peer M_m receives from M_{m-1} the shuffling S^{m-1}, with only one layer of encryption left, i.e., E_{K_m}. Note that a malicious 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 completely determine the outcome of the shuffling. This cannot even be detected by the other mixing peers.

CoinParty fixes this issue by requiring M_m to apply a publicly verifiable permutation, i.e., a lexicographic sorting. The randomness of the final shuffling π then depends only on the sum of the hashed output addresses, i.e., $C = \sum_{i=1}^{n} H(O_i)$. To control π, an attacker needs to choose *all* output addresses $O_1, ..., O_m$ which is clearly neither a realistic nor rational attack scenario (as this would also mean that the attacker controls *all* input addresses).

B. OUTPUT ADDRESSES SUBSTITUTION

We show how a malicious adversary can steal funds during the shuffling phase (Section 4.2), 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 $C = \sum_{i=1}^{n} H(O_i) = \sum_{i=1}^{n} H(O'_i)$. Finding such a sequence $O'_1, ..., O'_n$ requires solving a *Random Modular Subset Sum* (RMSS) problem, with modulus $M = 2^{len(H)}$, random elements $\{a_1, ..., a_N\} \in_U [0, M)$ and target sum C. $\delta = N/log(M)$ 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 $a_i = H(O'_i) \in [0, M)$. The best known algorithm for solving such instances has runtime $M^{O(1/log(N))}$ [19]. 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 length 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 is computed from 512-bit hashes, i.e. $M = 2^{256}$. Then, generating 10^{12} hashes amounts to roughly 58 TB of storage and we assume it is not feasible to generate more than $N \leq 10^{12}$ 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) of the shuffling phase 4.2.

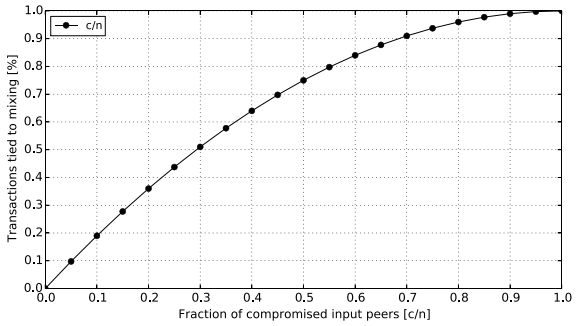


Figure 4: Fraction of identified mixing transactions vs fraction of compromised input peers.

C. SYBIL ATTACKS

An attacker that controls c of the n input peers can tie a fraction $1-(1-c/n)^2$ of the n mixing transactions to the mixing. Figure 4 plots the fraction of identified transactions versus the fraction c/n of compromised input peers. The plot, e.g., shows that an attacker can already tie 75 % of the transactions to the mixing, if he can compromise 50 % of the participants. Because *CoinParty* ensures randomness of the shuffling (Appendix A), the attacker cannot choose which transactions to tie to the mixing.

D. QUALITATIVE OVERHEAD PER MIXING PEER

In addition to our quantitative performance evaluation (cf. Section 5.4), Table 2 qualitatively summarizes the overhead of *CoinParty* for the three protocol phases, i.e., commitment, shuffling, and transaction. Communication rounds contain only the sequential synchronization points between mixing peers, i.e., message exchanges that cannot be batched and processed in parallel. We divide our discussion into precomputation efforts and online overhead. Notably, precomputations allow us to speed up the mixing process by utilizing otherwise idle resources before the actual mixing.

Precomputing the escrow addresses T_i requires n pseudo-random secret sharings (*PRSS*) in order to draw the shares $[d_i]$ of the secret key d_i. Then, n scalar-point multiplications over the elliptic curve (*EC-Mul*) are required to obtain $[Q_i] = [d_i G]$. Finally, we obtain the public key Q_i corresponding to T_i in n reconstructions (*S-Open*) in one round. For the shuffling phase, it is not possible to benefit from precomputations. For the transaction phase, we precompute for each T_i, $[k_i]$, $[k_i^{-1}]$, $[k_i^{-1}] * [d_i] \cdot R$ and kG as explained in Section 4.3. This first requires n PRSS for drawing the $[k_i]$. Computing the inverses $[k_i^{-1}], i = 1...n$, costs n PRSS, n multiplications on shares (*S-Mul*) batched into one round, and n *S-Open* batched into one round. Deriving $[k_i^{-1}] * [d_i] \cdot R, i = 1...n$, requires n *S-Mul* in one round and n *EC-Mul*. Finally, precomputing $k_i G, i = 1..n$, can be done in another n *EC-Mul* and n *S-Open* in one round.

Because of the extensive precomputations, the online phase of *CoinParty* is considerably cheaper. The commitment incurs no significant overhead, because the mixing peers only need to check that all input peers transferred their funds to the respective addresses T_i. In the shuffling phase, the processing overhead per mixing peer consists of one signature verification (*Ver)* for the received shuffling, n decryption operations (*Dec*) as well as one signature operation (*Sig*) to create the next shuffling and finally one share reconstruction for the checksum C. However, the predominant overhead results from the necessary communication, because the shuffling requires m consecutive rounds of decryption and permutation plus one subsequent round for reconstructing C. Finally, the online overhead of the transaction phase is comparably small with just n reconstructions batched into one round to obtain the signature part S from the shares $e[k^{-1}] + [k^{-1}] * [d] \cdot R$.

Phase	Processing per Mixing Peer	Rounds
Precomputation		
Commit	n PRSS + n EC-Mul + n S-Open	1
Shuffle	-	-
Transaction	$2n$ PRSS + $2n$ S-Mul + n EC-Mul + $2n$ S-Open	4
Online		
Commit	-	-
Shuffle	1 S-Open + n Dec + 1 Sig + 1 Ver	$m+1$
Transaction	n S-Open	1

Table 2: Qualitative overhead per mixing peer.

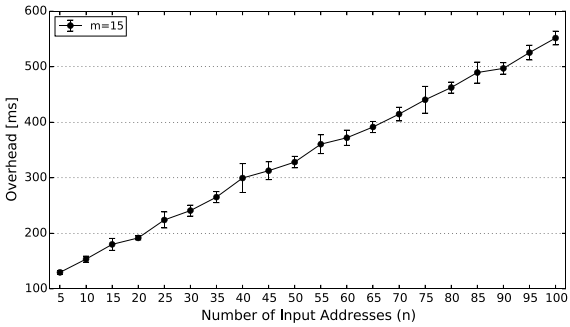


Figure 5: Overhead on one input peer in *CoinParty*.

E. OVERHEAD PER INPUT PEER

Additionally to our performance evaluation of the mixing peers (Section 5.4), we also consider the performance impact on the input peers in this section. The overhead for one input peer i to participate in one mixing operation consists of three steps. First, input peer i must compute the layered encryption of her output address $[\![O_i]\!]_{K_1:K_m}$. This requires m ECIES encryptions, one for each of the m mixing peers. Second, the input peer must compute and share the hash of her output address to all mixing peers, i.e., distribute the sharing $[H(O_i)]$. Third, the input peer must transfer ν Bitcoins to T_i. Peer i can collect the required funds from multiple input addresses, i.e., creating a transaction $\{I_i^1, ..., I_i^n\} \xrightarrow{\nu} T_i$. For each input address I_i^j, input peer i needs to compute one ECDSA signature. Figure 5 shows the resulting processing overhead for a variable number of $n = 5$ to 100 input addresses and $m = 15$ mixing peers. This does not include the communication overhead for distributing $[\![O_i]\!]_{K_1:K_m}$ and $[H(O_i)]$ to the mixing peers, since this can take place any time during the commitment phase $[t_0, t_1]$. The plain processing overhead is considerably small even when collecting funds from $n = 100$ input addresses. Communication will add another $50 - 200$ ms, depending on the latency between the input peer and the mixing peers. With a total overhead well below 1 second, the overhead per input peer is small enough to not influence the applicability of *CoinParty*.